Red Hat JBoss Enterprise Application Platform 7.4

設定ガイド

アプリケーションおよびサービスの実行を含む、Red Hat JBoss Enterprise Application Platform の設定およびメンテナンス手順

Last Updated: 2024-04-04

Red Hat JBoss Enterprise Application Platform 7.4 設定ガイド

アプリケーションおよびサービスの実行を含む、Red Hat JBoss Enterprise Application Platform の設定およびメンテナンス手順

法律上の通知

概要

本書は、JBoss EAP の設定や維持に必要な設定タスクと、JBoss EAP でアプリケーションやその他のサービスを稼働するために必要な設定タスクの多くを取り上げることを目的としています。

目次

JBOSS EAP ドキュメントへのフィードバック (英語のみ)

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウントにログインし、課題を送信してください。Red Hat Jira アカウントをお持ちでない場合は、アカウントを作成するように求められます。

手順

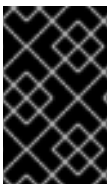
1. このリンクをクリック してチケットを作成します。
2. ドキュメント **URL**、セクション番号、課題の説明 を記入してください。
3. **Summary** に課題の簡単な説明を入力します。
4. **Description** に課題や機能拡張の詳細な説明を入力します。問題があるドキュメントのセクションへの URL を含めてください。
5. **Submit** をクリックすると、課題が作成され、適切なドキュメントチームに転送されます。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 はじめに

本ガイドを使用して JBoss EAP を設定する前に、最新バージョンの JBoss EAP がダウンロードされ、インストールされていることを確認してください。インストールの手順は、JBoss EAP の Installation Guide を参照してください。



重要

ホストマシンによって JBoss EAP をインストールする場所が異なるため、本ガイドではインストール場所を **EAP_HOME** と示しています。管理タスクを行う際には、**EAP_HOME** を実際の JBoss EAP のインストール場所に置き換えてください。

第2章 JBOSS EAP の開始および停止

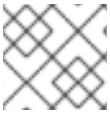
2.1. JBOSS EAP の開始

JBoss EAP は、Red Hat Enterprise Linux、Windows Server、および Oracle Solaris でサポートされ、スタンドアロンサーバーまたはマネージドドメイン操作モードで実行されます。JBoss EAP を起動するコマンドは、基盤のプラットフォームと選択する操作モードによって異なります。

サーバーは最初に一時停止状態で起動され、必要なサービスがすべて起動するまでリクエストを受け入れません。 必要なサービスがすべて起動すると、サーバーは通常の稼働状態となり、リクエストの受け入れを開始します。

JBoss EAP をスタンドアロンサーバーとして起動

```
$ EAP_HOME/bin/standalone.sh
```



注記

Windows Server の場合は、**EAP_HOME\bin\standalone.bat** スクリプトを使用します。

この起動スクリプトは、**EAP_HOME/bin/standalone.conf** ファイル (Windows Server の場合は **standalone.conf.bat**) を使用して、JVM オプションなどのデフォルト設定の一部を設定します。このファイルで設定をカスタマイズできます。

JBoss EAP はデフォルトで **standalone.xml** 設定ファイルを使用しますが、別の設定ファイルを使用して起動することもできます。利用できるスタンドアロン設定ファイルとそれらの使用方法については、スタンドアロンサーバー設定ファイル の項を参照してください。

使用できる起動スクリプトの引数の完全リストとそれら引数の目的については、**--help** 引数を使用するか、サーバーランタイムの引数およびスイッチ のセクションを参照してください。

管理対象ドメインでの JBoss EAP の起動

ドメイン内のサーバーグループのサーバーを起動する前にドメインコントローラーを起動する必要があります。このスクリプトを使用して最初にドメインコントローラーを起動した後、関連するホストコントローラーに対して使用します。

```
$ EAP_HOME/bin/domain.sh
```

注記

Windows Server の場合は **EAP_HOME\bin\domain.bat** スクリプトを使用します。

この起動スクリプトは、**EAP_HOME/bin/domain.conf** ファイル (Windows Server の場合は **standalone.conf.bat**) を使用して、JVM オプションなどのデフォルト設定の一部を設定します。このファイルで設定をカスタマイズできます。

JBoss EAP はデフォルトで **host.xml** ホスト設定ファイルを使用しますが、別の設定ファイルを使用して起動することもできます。利用できるマネージドドメイン設定ファイルとそれらの使用方法については、マネージドドメイン設定ファイル の項を参照してください。

管理対象ドメインを設定するとき、追加の引数を起動スクリプトに渡す必要があります。使用できる起動スクリプトの引数の完全リストとそれら引数の目的については、**--help** 引数を使用するか、サーバーランタイムの引数およびスイッチ のセクションを参照してください。

2.2. JBOSS EAP の停止

JBoss EAP の停止方法は、開始した方法によって異なります。

JBoss EAP の対話的なインスタンスの停止
JBoss EAP を起動したターミナルで **Ctrl+C** を押します。

JBoss EAP のバックグラウンドインスタンスの停止
管理 CLI を使用して、稼働中のインスタンスへ接続し、サーバーをシャットダウンします。

1. 管理 CLI を起動します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect
```

2. **shutdown** コマンドを実行します。

```
shutdown
```

注記

マネージドドメインで実行している場合、**shutdown** コマンドに **--host** 引数を使用してシャットダウンする、ホスト名を指定する必要があります。

2.3. JBOSS EAP の管理専用モードでの実行

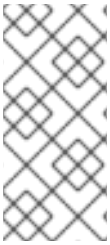
JBoss EAP は管理専用モードで起動することができます。管理専用モードでは、JBoss EAP は管理リクエストを実行および許可できますが、その他のランタイムサービスを起動したりエンドユーザーリクエストを許可したりすることはできません。管理専用モードは スタンドアロンサーバー と マネージドドメイン の両方で使用できます。

管理専用モードでのスタンドアロンサーバーの実行
管理専用モードでのサーバーの起動
JBoss EAP インスタンスを管理専用モードで起動するには、JBoss EAP インスタンスの起動時に **--start-mode=admin-only** ランタイム引数を使用します。

```
$ EAP_HOME/bin/standalone.sh --start-mode=admin-only
```

サーバーが管理専用モードで実行されていることを確認
以下のコマンドを実行して、サーバーの実行モードを確認します。サーバーが管理専用モードで実行されている場合は、結果が **ADMIN_ONLY** になります。

```
:read-attribute(name=running-mode)
{
  "outcome" => "success",
  "result" => "ADMIN_ONLY"
}
```



注記

さらに、以下のコマンドを使用すると、JBoss EAP が起動された最初の実行モードを確認することもできます。

```
/core-service=server-environment:read-attribute(name=initial-running-mode)
```

管理 **CLI** から別のモードでリロード
異なるランタイムスイッチを使用して JBoss EAP インスタンスを停止および起動する他に、管理 CLI を使用して異なるモードでリロードすることもできます。

サーバーを管理専用モードでリロードするには、以下を実行します。

```
reload --start-mode=admin-only
```

サーバーを通常モードでリロードするには、以下を実行します。

```
reload --start-mode=normal
```

サーバーが管理専用モードで起動され、**reload** コマンドに **--start-mode** 引数の指定がない場合は、サーバーは通常モードで起動されます。

管理専用モードでのマネージドドメインの実行
マネージドドメインの場合、ドメインコントローラーが管理専用モードで起動されると、スレーブホストコントローラーからの受信接続を許可しません。

管理専用モードでのホストコントローラーの起動
--admin-only ランタイム引数を渡してホストコントローラーを管理専用モードで起動します。

```
$ EAP_HOME/bin/domain.sh --admin-only
```

ホストコントローラーが管理専用モードで実行されていることを確認
以下のコマンドを実行して、ホストコントローラーの実行モードを確認します。ホストコントローラーが管理専用モードで実行されている場合は、結果が **ADMIN_ONLY** になります。

```
/host=HOST_NAME:read-attribute(name=running-mode)
{
  "outcome" => "success",
  "result" => "ADMIN_ONLY"
}
```

管理 **CLI** から別のモードでリロード
異なるランタイムスイッチを使用してホストコントローラーを停止および起動する他に、管理 CLI を使用して異なるモードでリロードすることもできます。

ホストコントローラーを管理専用モードでリロードするには、以下を実行します。

```
reload --host=HOST_NAME --admin-only=true
```

ホストコントローラーを通常モードでリロードするには、以下を実行します。

```
reload --host=HOST_NAME --admin-only=false
```

ホストコントローラーが管理専用モードで起動され、**reload** コマンドに **--start-mode** 引数の指定がない場合は、ホストコントローラーは通常モードで起動されます。

2.4. JBOSS EAP の正常な一時停止およびシャットダウン

JBoss EAP は正常に一時停止およびシャットダウンできます。これにより、新しいリクエストを許可せずにアクティブなリクエストを正常に完了できます。タイムアウト値は、一時停止またはシャットダウン操作がアクティブなリクエストの完了まで待つ期間を指定します。サーバーが一時停止しても管理リクエストは処理されます。

正常なシャットダウンは、リクエストがサーバーに入るエントリーポイントを中心にサーバー全体のレベルで調整されます。以下のサブシステムが正常なシャットダウンをサポートします。

Undertow

undertow サブシステムはすべてのリクエストが終了するまで待機します。

mod_cluster

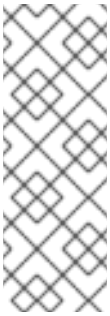
modcluster サブシステムは **PRE_SUSPEND** フェーズでサーバーが一時停止することをロードバランサーに通知します。

ejb3

ejb3 サブシステムはすべてのリモート Jakarta Enterprise Beans リクエストおよび MDB メッセージ配信が終了するまで待機します。MDB への配信は **PRE_SUSPEND** フェーズで停止します。Jakarta Enterprise Beans タイマーは中断され、サーバーが再開したときにタイマーがアクティベートされます。

トランザクション

サーバーは一時停止すると新しいリクエストを受け付けませんが、インフライトトランザクションおよびリクエストは完了するかタイムアウトが期限切れになるまで継続されます。これは、XTS トランザクションに関連する web サービスリクエストも同様です。



注記

デフォルトでは、**ejb** サブシステムでのトランザクションの正常シャットダウンは無効になっています。Jakarta Enterprise Beans 関連のトランザクションが完了してからサーバーを一時停止する場合は、トランザクションの正常シャットダウンを有効にする必要があります。以下に例を示します。

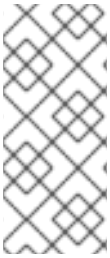
```
/subsystem=ejb3:write-attribute(name=enable-graceful-txn-shutdown,value=true)
```

Jakarta Concurrency

サーバーはすべてのアクティブなジョブが終了するまで待機します。キューに置かれたジョブはすべてスキップされます。現在、Jakarta Concurrency には永続性がないため、キューに置かれスキップされたジョブは失われます。
サーバーが一時停止状態である間、スケジュールされたタスクはスケジュールどおりの時間に実行されますが、**java.lang.IllegalStateException** が発生します。サーバーが再開されると、スケジュールされたタスクは通常どおり実行され、ほとんどの場合でタスクのスケジュールを変更する必要はありません。

Batch

サーバーはタイムアウト期間内の実行中のジョブをすべて停止し、スケジュール済みのジョブをすべて延期します。



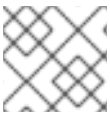
注記

現在、正常シャットダウンは新しいインバウンド Jakarta Messaging メッセージを拒否しません。インフライトアクティビティーによってスケジュールされた Jakarta Batch ジョブおよび Jakarta Concurrency タスクは、現在実行の継続が許可されます。しかし、タイムアウトウインドウを渡す Jakarta Concurrency タスクが提出されると、実行時にエラーが発生します。

リクエストは **request-controller** サブシステムによって追跡されます。このサブシステムがないと、一時停止および再開機能が制限され、リクエストの完了を待たずにサーバーが一時停止またはシャットダウンします。しかし、この機能が必要ない場合は、**request-controller** サブシステムを削除してパフォーマンスを若干向上することができます。

2.4.1. サーバーの一時停止

JBoss EAP 7 には、サーバーの操作を正常に一時停止する **suspend** モードが導入されました。このモードでは、アクティブなリクエストがすべて正常に完了されますが、新しいリクエストは許可されません。サーバーが中断されたら、シャットダウンすることができます。 さらに、元の実行状態に戻ったり、中断状態のままメンテナンスを実行することができます。



注記

管理インターフェイスのサーバーの一時停止よる影響はありません。

管理コンソールまたは管理 CLI を使用するとサーバーを一時停止および再開できます。

サーバーの一時停止状態のチェック
以下の管理 CLI コマンドを使用するとサーバーの中断状態を表示できます。結果の値は、**RUNNING**、**PRE_SUSPEND**、**SUSPENDING**、または **SUSPENDED** のいずれかになります。

- スタンドアロンサーバーの中断状態をチェックします。

```
:read-attribute(name=suspend-state)
```

- マネージドドメインのサーバーの中断状態をチェックします。

```
/host=master/server=server-one:read-attribute(name=suspend-state)
```

中断
アクティブなリクエストが完了するまでサーバーが待機するタイムアウト値を秒単位で指定し、以下の管理 CLI コマンドを使用してサーバーを中断します。デフォルト値は **0** で、即座に中断します。**-1** を値として指定すると、サーバーはアクティブなリクエストがすべて完了するまで無期限に待機します。

以下の各例は、リクエストが完了するまで最大 60 秒待機した後、一時停止します。

- スタンドアロンサーバーを一時停止します。

```
:suspend(suspend-timeout=60)
```

- マネージドドメインのすべてのサーバーを一時停止します。

```
:suspend-servers(suspend-timeout=60)
```

- マネージドドメインの単一のサーバーを一時停止します。

```
/host=master/server-config=server-one:suspend(suspend-timeout=60)
```

- サーバーグループのすべてのサーバーを一時停止します。

```
/server-group=main-server-group:suspend-servers(suspend-timeout=60)
```

- ホストレベルですべてのサーバーを一時停止します。

```
/host=master:suspend-servers(suspend-timeout=60)
```

再開
resume コマンドは、新しいリクエストを受け入れるために、サーバーを通常の実行状態に戻します。ホスト、サーバー、サーバーグループ、またはドメインレベルでコマンドを起動できます。以下に例を示します。

```
:resume
```

サーバーを一時停止状態で起動
サーバーを一時停止状態で起動し、サーバーが再開するまでリクエストを受け入れないようにすることができます。

- スタンドアロンサーバーを一時停止状態で起動するには、JBoss EAP インスタンスの起動時に **--start-mode=suspend** ランタイム引数を使用します。

```
$ EAP_HOME/bin/standalone.sh --start-mode=suspend
```

- マネージドドメインサーバーを一時停止モードで起動するには、管理 CLI コマンドで **start-mode=suspend** 引数を **start** 操作に渡します。

```
/host=HOST_NAME/server-config=SERVER_NAME:start(start-mode=suspend)
```

注記

start-mode 引数をサーバーの **reload** および **restart** 操作に渡すこともできます。

- マネージドドメインサーバーグループのすべてのサーバーを一時停止モードで起動するには、管理 CLI コマンドで **start-mode=suspend** 引数を **start-servers** 操作に渡します。

```
/server-group=SERVER_GROUP_NAME:start-servers(start-mode=suspend)
```

注記

start-mode 引数をサーバーグループの **reload-servers** および **restart-servers** 操作に渡すこともできます。

2.4.2. 管理 CLI を使用したサーバーの正常なシャットダウン

サーバーの停止時に適切なタイムアウト値を指定すると、サーバーは正常にシャットダウンされます。コマンドを実行するとサーバーが一時停止され、すべてのリクエストが完了するまで最大で指定のタイムアウトの期間待機し、その後シャットダウンします。

以下の管理 CLI コマンドを使用してサーバーを正常にシャットダウンします。アクティブなリクエストが完了するまでサーバーが待機するタイムアウト値を秒単位で指定します。デフォルト値は **0** で、即座にサーバーをシャットダウンします。**-1** を値として指定すると、アクティブなリクエストがすべて完了するまで無期限に待機した後、シャットダウンします。

以下の各例は、リクエストが完了するまで最大 60 秒待機した後、シャットダウンします。

- スタンドアロンサーバーを正常にシャットダウンします。

```
shutdown --suspend-timeout=60
```

- マネージドドメインのすべてのサーバーを停止します。

```
:stop-servers(suspend-timeout=60)
```

- マネージドドメインの単一のサーバーを停止します。

```
/host=master/server-config=server-one:stop(suspend-timeout=60)
```

- サーバーグループのすべてのサーバーを正常に停止します。

```
/server-group=main-server-group:stop-servers(suspend-timeout=60)
```

- ホストコントローラーと、マネージドのサーバーをすべてシャットダウンします。

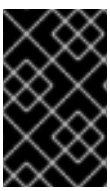
```
/host=master:shutdown(suspend-timeout=60)
```

注記

suspend-timeout 属性は、ホストコントローラー自体ではなく、ホストコントローラーが管理するサーバーにのみ適用されます。

2.4.3. OS シグナルを使用したサーバーの正常なシャットダウン

kill -15 PID などで OS **TERM** シグナルを送信すると、サーバーを正常にシャットダウンできます。デフォルトでは、この値は管理 CLI の **shutdown --suspend-timeout=0** コマンドの値と同じであるため、現在処理中のリクエストを即座に終了できます。タイムアウトは、**org.wildfly.sigterm.suspend.timeout** システムプロパティーで設定でき、サーバーのシャットダウン前に待機する最大秒数を指定します。**-1** を値として指定すると、サーバーは永久に待機します。



重要

マネージドドメインでは、OS シグナルを使用してサーバーをシャットダウンしないでください。サーバーは、管理するホストコントローラーより 管理 CLI を使用してシャットダウン してください。

シグナルの処理を無効にするよう JVM が設定された場合 (**-Xrs** java 引数が JVM オプションに渡された場合など) や、プロセスが送信されたシグナルに応答できない場合 (**KILL** シグナルが送信された場合など)、OS シグナルを使用して正常にシャットダウンすることはできません。

2.5. JBOSS EAP の起動および停止 (RPM インストール)

RPM インストールの場合、JBoss EAP の起動と停止が ZIP またはインストーラーインストールの場合とは異なります。

2.5.1. JBoss EAP の起動 (RPM インストール)

RPM インストールの JBoss EAP を起動するコマンドは、開始する操作モード (スタンドアロンサーバーまたはマネージドドメイン) や実行している Red Hat Enterprise Linux のバージョンによって異なります。

JBoss EAP をスタンドアロンサーバーとして起動 (RPM インストール)

- Red Hat Enterprise Linux 6 の場合

```
$ service eap7-standalone start
```

- Red Hat Enterprise Linux 7 以降の場合:

```
$ systemctl start eap7-standalone.service
```

これにより、**standalone.xml** 設定ファイルをデフォルトで使用して JBoss EAP が起動されます。JBoss EAP を別の スタンドアロンサーバー設定ファイル で起動するには、RPM サービス設定ファイル にプロパティーを設定します。詳細は RPM サービスプロパティーの設定 の項を参照してください。

マネージドドメインでの JBoss EAP の起動 (RPM インストール)

- Red Hat Enterprise Linux 6 の場合

```
$ service eap7-domain start
```

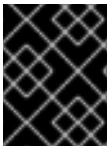
- Red Hat Enterprise Linux 7 以降の場合:

```
$ systemctl start eap7-domain.service
```

これにより、**host.xml** 設定ファイルをデフォルトで使用して JBoss EAP が起動されます。JBoss EAP を別の マネージドドメイン設定ファイル で起動するには、RPM サービス設定ファイル にプロパティーを設定します。詳細は RPM サービスプロパティーの設定 の項を参照してください。

RPM サービスプロパティーの設定
本項では、RPM サービスプロパティーと JBoss EAP インストールのその他の起動オプションを設定する方法について説明します。変更を行う前に設定ファイルをバックアップすることが推奨されます。

RPM インストールで利用可能な起動オプションの完全リストは、RPM サービス設定プロパティー の項を参照してください。



重要

Red Hat Enterprise Linux 7 以降では、RPM サービス設定ファイルは **systemd** を使用してロードされるため、変数式は拡張されません。

- サーバー設定ファイルを指定します。
  スタンドアロンサーバーを起動する場合、デフォルトで **standalone.xml** ファイルが使用されます。マネージドドメインで実行する場合、デフォルトで **host.xml** ファイルが使用されます。

他の設定ファイルを使用して JBoss EAP を起動するには、適切な **RPM** 設定ファイル (例: eap7-standalone.conf) に **WILDFLY_SERVER_CONFIG** プロパティーを設定します。

```
WILDFLY_SERVER_CONFIG=standalone-full.xml
```

- 特定の IP アドレスにバインドします。
  デフォルトでは、JBoss EAP RPM インストールは **0.0.0.0** にバインドします。JBoss EAP を特定の IP アドレスにバインドするには、適切な **RPM** 設定ファイル (例: eap7-standalone.conf) に **WILDFLY_BIND** プロパティーを設定します。

```
WILDFLY_BIND=192.168.0.1
```

注記

管理インターフェイスを特定の IP アドレスにバインドする場合は、次の例のように JBoss EAP 起動設定ファイルに設定を追加します。

- JVM オプションまたは Java プロパティーを設定します。
  JBoss EAP の起動スクリプトに渡す JVM オプションまたは Java プロパティーを指定するには、起動設定ファイルを編集します。スタンドアロンサーバーの場合、このファイルは **EAP_HOME/bin/standalone.conf** になります。マネージドドメインの場合、このファイルは **EAP_HOME/bin/domain.conf** になります。以下の例は、ヒープサイズを設定し、JBoss EAP 管理インターフェイスを指定の IP アドレスにバインドします。

```
JAVA_OPTS="$JAVA_OPTS -Xms2048m -Xmx2048m"
JAVA_OPTS="$JAVA_OPTS -Djboss.bind.address.management=192.168.0.1"
```

注記

場合によっては、標準の **jboss.bind.address** プロパティーを使用せずに **WILDFLY_BIND** プロパティーを使用して JBoss EAP バインドアドレスを設定する必要があります。

注記

同じ名前のプロパティーが RPM サービス設定ファイル (例: **/etc/sysconfig/eap7-standalone**) と JBoss EAP 起動設定ファイル (例: **EAP_HOME/bin/standalone.conf**) にある場合、JBoss EAP 起動設定ファイルのプロパティーの値が優先されます。このようなプロパティーの 1 つが **JAVA_HOME** です。

2.5.2. JBoss EAP の停止 (RPM インストール)

RPM インストールの JBoss EAP を停止するコマンドは、開始された操作モード (スタンドアロンサーバーまたはマネージドドメイン) や実行している Red Hat Enterprise Linux のバージョンによって異なります。

JBoss EAP をスタンドアロンサーバーとして停止 (RPM インストール)

- Red Hat Enterprise Linux 6 の場合

```
$ service eap7-standalone stop
```

- Red Hat Enterprise Linux 7 以降の場合:

```
$ systemctl stop eap7-standalone.service
```

マネージドドメインでの **JBoss EAP** の停止 **(RPM** インストール**)**

- Red Hat Enterprise Linux 6 の場合

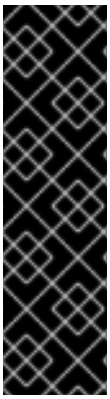
```
$ service eap7-domain stop
```

- Red Hat Enterprise Linux 7 以降の場合:

```
$ systemctl stop eap7-domain.service
```

RPM インストールで利用可能な起動オプションの完全リストは、RPM サービス設定ファイル の項を参照してください。

2.6. POWERSHELL スクリプト (WINDOWS SERVER)



重要

複数の PowerShell スクリプトはテクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

JBoss EAP には、ほとんどの JBoss EAP 管理スクリプトに相当する PowerShell スクリプトが含まれています。これには、Microsoft Windows Server で JBoss EAP を起動する PowerShell スクリプトが含まれます。

JBoss EAP PowerShell スクリプトは、テストされた Windows Server バージョンで実行される PowerShell バージョン 2 以上で動作することを目的としています。

JBoss EAP PowerShell スクリプトは **EAP_HOME\bin** にあり、JBoss EAP のバッチスクリプトとほぼ同様に使用されます。

たとえば、**standalone-full.xml** 設定ファイルを使用してスタンドアロン JBoss EAP サーバーを起動するには、以下の PowerShell コマンドを使用します。

```
.\standalone.ps1 "-c=standalone-full.xml"
```

注記

JBoss EAP PowerShell スクリプトの引数は引用符で囲む必要があります。

第3章 JBOSS EAP の管理

JBoss EAP は簡単な設定を使用し、スタンドアロンサーバーまたはマネージドドメインごとに 1 つの設定ファイルを使用します。スタンドアロンサーバーのデフォルト設定は **EAP_HOME/standalone/configuration/standalone.xml** ファイルに保存され、マネージドドメインのデフォルト設定は **EAP_HOME/domain/configuration/domain.xml** ファイルに保存されます。また、ホストコントローラーのデフォルト設定は **EAP_HOME/domain/configuration/host.xml** ファイルに保存されます。

JBoss EAP はコマンドラインの管理 CLI、Web ベースの管理コンソール、Java API、または HTTP API を使用して設定できます。これらの管理インターフェイスを使用して加えられた変更は自動的に永続化され、XML 設定ファイルは管理 API によって上書きされます。管理 CLI と管理コンソールの使用が推奨され、XML 設定ファイルの手作業による編集は推奨されません。

3.1. サブシステム、エクステンション、およびプロファイル

JBoss EAP では、設定される機能の内容がサブシステムによって異なります。たとえば、アプリケーションおよびサーバーロギングが **logging** サブシステムに設定されます。

エクステンションは、サーバーのコア機能を拡張するモジュールです。エクステンションはデプロイメントが必要なときにロードされ、必要でなくなるとアンロードされます。**Management CLI Guide**で、テクステンションの 追加 方法および 削除 方法を確認してください。

サブシステム は特定のエクステンションの設定オプションを提供します。使用できるサブシステムの詳細は、JBoss EAP サブシステムの概要 を参照してください。

サーバーの要求を満たすために設定される プロファイルは、複数のサブシステムで設定されます。スタンドアロンサーバーには名前のないプロファイルが 1 つあります。マネージドドメインでは、ドメインのサーバーグループによって使用される プロファイル を複数定義できます。

管理コンソールまたは管理 CLI の使用

JBoss EAP インスタンスの設定更新には管理コンソールと管理 CLI の両方を使用でき、サポートされます。どちらを使用するかはユーザーの好みによります。グラフィカルな Web ベースのインターフェイスを好むユーザーは管理コンソールを使用する必要があります。コマンドラインインターフェイスを好むユーザーは、管理 CLI を使用する必要があります。

3.2. 管理ユーザー

デフォルトの JBoss EAP 設定はローカル認証を提供するため、ユーザーは認証の必要なくローカルホスト上で管理 CLI にアクセスできます。

しかし、リモートで管理 CLI にアクセスする場合や管理コンソールを使用する場合 (トラフィックの送信元がローカルホストであってもリモートアクセスとして見なされます) は、管理ユーザーを追加する必要があります。管理ユーザーを追加せずに管理コンソールへアクセスしようとすると、エラーメッセージが出力されます。

グラフィカルインストーラーを使用して JBoss EAP がインストールされた場合は、インストールプロセス中に管理ユーザーが作成されます。

本ガイドでは、**add-user** スクリプトを使用した JBoss EAP の簡単なユーザー管理を取り上げます。 このスクリプトはデフォルトの認証のプロパティーファイルに新しいユーザーを追加するためのユーティリティーです。

LDAP やロールベースアクセス制御 (RBAC) などの高度な認証および承認のオプションについては、JBoss EAP**Security Architecture**の Core Management Authentication を参照してください。

3.2.1. 管理ユーザーの追加

1. **add-user** ユーティリティースクリプトを実行し、プロンプトに従います。

```
$ EAP_HOME/bin/add-user.sh
```

注記

Windows Server の場合は、**EAP_HOME\bin\add-user.bat** スクリプトを使用します。

2. **ENTER** を押して、デフォルトのオプション **a** を選択し、管理ユーザーを追加します。
このユーザーは **ManagementRealm** に追加され、管理コンソールまたは管理 CLI を使用して管理操作を実行する権限が与えられます。代わりに **b** を選択すると、アプリケーションに使用される **ApplicationRealm** にユーザーが追加され、特定のパーミッションは提供されません。

3. ユーザー名とパスワードを入力します。入力後、パスワードを確認するよう指示されます。

注記

ユーザー名には、以下の文字のみを使用できます。 文字の数と順番は自由です。

- 英数字 (a-z、A-Z、0-9)
- ダッシュ (-)、ピリオド (.)、コンマ (,)、アットマーク (@)
- バックスラッシュ (\)
- 等号 (=)

デフォルトでは、脆弱なパスワードは許可されますが、警告が表示されます。

このデフォルト動作の変更に関する詳細は、Add-User ユーティリティーのパスワード制限の設定 を参照してください。

4. ユーザーが属するグループのコンマ区切りリストを入力します。ユーザーがグループに属さないようにする場合は **ENTER** を押して空白のままにします。

5. 情報を確認し、正しければ **yes** を入力します。

6. このユーザーがリモート JBoss EAP サーバーインスタンスを表すかどうかを決定します。基本的な管理ユーザーの場合は **no** を入力します。
ManagementRealm への追加が必要になることがあるユーザータイプの 1 つが、JBoss EAP の別のインスタンスを表すユーザーで、メンバーとしてクラスターに参加することを承認できる必要があります。この場合は、プロンプトで **yes** を選択すると、異なる設定ファイルに追加する必要がある、ユーザーのパスワードを表すハッシュ化された秘密の値が提供されます。

パラメーターを **add-user** スクリプトに渡すと、非対話的にユーザーを作成できます。ログや履歴ファイルにパスワードが表示されるため、この方法は共有システムでは推奨されません。詳細は Add-User ユーティリティーを非対話的に実行 を参照してください。

3.2.2. Add-User ユーティリティーを非対話的に実行

コマンドラインで引数を渡すと **add-user** スクリプトを非対話的に実行することができます。最低でも、ユーザー名とパスワードを提供する必要があります。

警告

ログや履歴ファイルにパスワードが表示されるため、この方法は共有システムでは推奨されません。

複数のグループに属するユーザーの作成
以下のコマンドは、**guest** および **mgmtgroup** グループの管理ユーザー **mgmtuser1** を追加します。

```
$ EAP_HOME/bin/add-user.sh -u 'mgmtuser1' -p 'password1!' -g 'guest,mgmtgroup'
```

代替プロパティーファイルの指定
デフォルトでは、**add-user** スクリプトを使用して作成されたユーザーおよびグループ情報は、サーバー設定ディレクトリーにあるプロパティーファイルに保存されます。

ユーザー情報は以下のプロパティーファイルに保存されます。

- **EAP_HOME/standalone/configuration/mgmt-users.properties**
- **EAP_HOME/domain/configuration/mgmt-users.properties**

グループ情報は以下のプロパティーファイルに保存されます。

- **EAP_HOME/standalone/configuration/mgmt-groups.properties**
- **EAP_HOME/domain/configuration/mgmt-groups.properties**

これらのデフォルトディレクトリーとプロパティーファイル名は上書きできます。以下のコマンドは、ユーザープロパティーファイルの名前と場所を指定して、新しいユーザーを追加します。

```
$ EAP_HOME/bin/add-user.sh -u 'mgmtuser2' -p 'password1!' -sc '/path/to/standaloneconfig/' -dc '/path/to/domainconfig/' -up 'newname.properties'
```

新しいユーザーは **/path/to/standaloneconfig/newname.properties** および **/path/to/domainconfig/newname.properties** にあるユーザープロパティーファイルに追加されます。これらのファイルは存在している必要があり、存在しない場合はエラーが出力されます。

使用できる **add-user** のすべての引数の完全リストとそれら引数の目的については、**--help** 引数を指定するか、Add-User 引数 の項を参照してください。

3.2.3. Add-User ユーティリティーのパスワード制限

add-user ユーティリティースクリプトのパスワード制限は、**EAP_HOME/bin/add-user.properties** ファイルを使用して設定できます。

重要

add-user.properties ファイルは保護されていないプレーンテキストファイルです。コンテンツへの不正アクセスを回避するために保護する必要があります。

意図しないパスワードを設定しないようにするには、キーボードのシステムキーマップが正しいことを確認します。デフォルトのシステムキーマップは **en-qwerty** です。このデフォルト設定を変更して新しいパスワードを作成する場合は、**SimplePasswordStrengthChecker** クラスにある基準をパスワードが満たしていることを確認する必要があります。

JBoss EAP ではデフォルトで、脆弱なパスワードは許可されますが、警告が表示されます。指定の最低要件を満たさないパスワードを拒否するには、**password.restriction** プロパティーを **REJECT** に設定します。

以下の表は、**EAP_HOME/bin/add-user.properties** ファイルで設定できる追加のパスワード要件の設定を示しています。

表3.1 パスワード要件の追加設定

属性	説明
minLength	パスワードの最小文字数。たとえば、**password.restriction.minLength=8** はパスワードを最小文字に制限します。
strength	有効なパスワードに必要なしきい値を設定します。有効なしきい値エントリーには以下が含まれます。 * **VERY_WEAK** * **WEAK** * **MODERATE** * **MEDIUM** * **STRONG** * **VERY_STRONG** * **EXCEPTIONAL** デフォルト値は **MODERATE** です。しきい値が定義され、そのデフォルト値が **SimplePasswordStrengthChecker** クラスに指定されます。 注記: しきい値を指定しない場合は、**MODERATE** がデフォルト値になります。この値は **SimplePasswordStrengthChecker** クラスに指定されます。
minAlpha	パスワードに設定されたアルファベットの最小文字数。たとえば、**password.restriction.minAlpha=1** は、パスワードに最低 1 文字のアルファベット文字を含めるよう制限します。
minDigit	パスワードに設定された数字のの最小数。たとえば、**password.restriction.minDigit=1** は、最低でも 1 つの数字を含むようにパスワードを制限します。

属性	説明
minSymbol	パスワードに設定された記号の最小数。たとえば、**password.restriction.min=1** は、パスワードに記号を最低でも 1 文字含めるよう制限します。
forbiddenValue	簡単に思いつくパスワード (root など) を設定しないように、ユーザーを制限します。たとえば、**password.restriction.forbidden=root,admin,administrator** は、パスワードとして root、admin、または administrator を設定できないように制限します。
mustNotMatchUsername	ユーザーがユーザー名をパスワードとして設定できないように制限します。たとえば、**password.restriction.mustNotMatchUsername=TRUE** と指定すると、ユーザーはユーザー名をパスワードとして設定できなくなります。**false** に設定すると、このルールは無視されます。

その他のリソース

Red Hat カスタマーポータルの 基本的なシステム設定の設定 を参照してください。

3.2.4. 管理ユーザーの更新

add-user ユーティリティースクリプトを使用し、プロンプトに従ってユーザー名を入力すると、既存の管理ユーザーの設定を更新できます。

```
$ EAP_HOME/bin/add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): a

Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : test-user
User 'test-user' already exists and is enabled, would you like to...
 a) Update the existing user password and roles
 b) Disable the existing user
 c) Type a new username
(a):
```

すでに存在するユーザー名を入力すると、複数のオプションが出力されます。

- 既存ユーザーのパスワードを更新する場合は **a** を入力します。
- 既存ユーザーを無効にする場合は **b** を入力します。
- 新しいユーザー名を入力する場合は **c** を入力します。

警告

add-user スクリプトを使用して非対話的にユーザーを更新すると、ユーザーは自動的に更新され、確認のプロンプトは表示されません。

3.3. JBOSS EAP サーバー設定の最適化

JBoss EAP サーバーをインストール し、管理ユーザー を作成 したら、サーバー設定を最適化できます。

Performance Tuning Guide で、本番環境にアプリケーションをデプロイするときに一般的な問題が発生しないようサーバー設定を最適化する方法を必ず確認してください。通常の最適化には、ulimit の設定、ガベッジコレクションの有効化、Java ヒープダンプの作成、スレッドプールサイズの調整 などが含まれます。

また、製品のリリースに既存のパッチを適用するとよいでしょう。EAP の各パッチには、多くのバグ修正が含まれています。詳細は、JBoss EAP**Patching and Upgrading Guide**の Patching JBoss EAP を参照してください。

3.4. 管理インターフェイス

3.4.1. 管理 CLI

管理コマンドラインインターフェイス (CLI) は、JBoss EAP のコマンドライン管理ツールです。

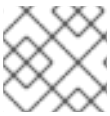
管理 CLI を使用して、サーバーの起動および停止、アプリケーションのデプロイおよびアンデプロイ、システムの設定、他の管理タスクの実行を行います。管理 CLI は、マネージドドメインのドメインコントローラーに接続し、ドメインで管理操作を実行することもできます。

ls、**cd**、**pwd** など、多くの共通するターミナルコマンドを使用できます。管理 CLI はタブ補完をサポートします。

コマンドと操作、構文、およびバッチモードでの実行を含む、管理 CLI の使用に関する詳細は、JBoss EAP**Management CLI Guide** を参照してください。

管理 **CLI** の起動

```
$ EAP_HOME/bin/jboss-cli.sh
```



注記

Windows Server の場合は、**EAP_HOME\bin\jboss-cli.bat** スクリプトを使用します。

稼働中のサーバーへの接続

```
connect
```

上記の代わりに、管理 CLI を起動し、**EAP_HOME/bin/jboss-cli.sh --connect** コマンドを使用すると 1 度に接続できます。

ヘルプの表示
以下のコマンドを実行してヘルプを表示します。

```
help
```

コマンドで **--help** フラグを使用すると、そのコマンドの使用に関する説明が表示されます。たとえば、**deploy** コマンドの使用に関する情報を表示するには、以下のコマンドを実行します。

```
deploy --help
```

管理 **CLI** の終了

```
quit
```

システム設定の表示
以下のコマンドは **read-attribute** 操作を使用して、データソースの例が有効になっているかどうかを表示します。

```
/subsystem=datasources/data-source=ExampleDS:read-attribute(name=enabled)
{
    "outcome" => "success",
    "result" => true
}
```

マネージドドメインで実行している場合、コマンドの前に /**profile=PROFILE_NAME** を付けて更新するプロファイルを指定する必要があります。

```
/profile=default/subsystem=datasources/data-source=ExampleDS:read-attribute(name=enabled)
```

システム設定の更新
以下のコマンドは **write-attribute** 操作を使用して、データソースの例を無効にします。

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=enabled,value=false)
```

サーバーの起動
マネージドドメインで実行している場合、管理 CLI を使用してサーバーを起動および停止することもできます。

```
/host=HOST_NAME/server-config=server-one:start
```

3.4.2. 管理コンソール

管理コンソールは、JBoss EAP の web ベースの管理ツールです。

管理コンソールを使用して、サーバーの開始および停止、アプリケーションのデプロイおよびアンデプロイ、システム設定の調整、サーバー設定の変更の永続化を行います。管理コンソールは管理タスクも実行でき、現在のユーザーが変更を行った後にサーバーインスタンスの再起動またはリロードが必要な場合はライブ通知も行います。

マネージドドメインでは、同じドメインのサーバーインスタンスとサーバーグループをドメインコントローラーの管理コンソールから集中管理できます。

デフォルトの管理ボートを使用してローカルホストで稼働している JBoss EAP インスタンスの場合、Web ブラウザーを使用して http://localhost:9990/console/index.html で管理コンソールにアクセスできます。管理コンソールにアクセスできるパーミッションを持つユーザーで認証する必要があります。

管理コンソールでは、JBoss EAP スタンドアロンサーバーまたはマネージドドメインを操作および管理するために以下のタブが提供されます。

Home (ホーム)

一般的な設定および管理タスクを行う方法を学ぶことができます。ツアーに参加して JBoss EAP 管理コンソールについてよく理解してください。

Deployments (デプロイメント)

デプロイメントを追加、削除、および有効化します。マネージドドメインでは、デプロイメントをサーバーグループに割り当てます。

Configuration (設定)

Web サービス、メッセージング、高可用性などの機能を提供する利用可能なサブシステムを設定します。マネージドドメインでは、異なるサブシステム設定が含まれるプロファイルを管理します。

Runtime (ランタイム)

サーバーの状態、JVM 使用率、サーバーログなどのランタイム情報を表示します。マネージドドメインではホスト、サーバーグループ、およびサーバーを管理します。

Patching (パッチ)

JBoss EAP インスタンスにパッチを適用します。

Access Control (アクセス制御)

ロールベースアクセス制御を使用するときにユーザーとグループにロールを割り当てます。

3.4.2.1. 管理コンソールの属性の更新

リソースを編集するために管理コンソールの適切なセクションに移動した後、適切なパーミッションがあれば属性を編集できます。

1. 編集 リンクをクリックします。
2. 必要な変更を追加します。
   必要なフィールドにはアスタリスク (*) が付いています。ヘルプ リンクをクリックすると、属性の詳細を表示できます。

注記

入力フィールドは、属性のタイプに応じてテキストフィールド、ON/OFF フィールド、またはドロップダウンになります。テキストフィールドによっては、入力時に設定にある値が候補として表示されます。

3. 保存 をクリックして変更を保存します。
4. 必要な場合は、サーバーをリロードして変更を反映します。
   変更を保存するとき、変更の反映にリロードが必要な場合はポップアップが表示されます。スタンドアロンサーバーをリロードするには、ポップアップの 再読み込み リンクをクリックします。マネージドドメインでサーバーをリロードするには、**Topology** リンクをクリックして、適切なサーバーを選択し、再読み込み ドロップダウンオプションをクリックします。

最近実行した設定に関するアクションの履歴を表示するには、管理コンソールの右上にある通知アイコンをクリックします。

3.4.2.2. 管理コンソールの有効化/無効化

/core-service=management/management-interface=http-interface リソースの **console-enabled** ブール値属性を設定すると、管理コンソールを有効または無効にできます。ドメインモードマスターホストの場合は、**/host=master/core-service=management/management-interface=http-interface** を使用します。

たとえば、有効にする場合は以下を設定します。

```
/core-service=management/management-interface=http-interface:write-attribute(name=console-enabled,value=true)
```

無効にする場合は以下を設定します。

```
/core-service=management/management-interface=http-interface:write-attribute(name=console-enabled,value=false)
```

3.4.2.3. 管理コンソールの言語の変更

管理リソースの言語はデフォルトの英語に設定されています。以下の言語の 1 つを選択することもできます。

- ドイツ語 (de)
- 簡体中国語 (zh-Hans)
- ブラジルポルトガル語 (pt-BR)
- フランス語 (fr)
- スペイン語 (es)
- 日本語 (ja)

管理コンソールの言語の変更方法

1. 管理コンソールにログインします。
2. 管理コンソールの右下隅にある **Settings** をクリックします。
3. **Locale** 選択ボックスから必要な言語を選択します。
4. **Save** を選択します。確認ボックスに、アプリケーションのリロードが必要であると表示されます。
5. **Yes** をクリックします。システムによってブラウザーが自動的に更新され、選択したロケールが使用されます。

3.4.2.4. 管理コンソールタイトルのカスタマイズ

JBoss EAP インスタンスのそれぞれを一目で特定できるように、管理コンソールのタイトルをカスタマイズできます。

管理コンソールのタイトルをカスタマイズするには、以下の手順に従います。

1. 管理コンソールにログインします。

2. 管理コンソールの右下隅にある **Settings** をクリックします。
3. **Settings** ウインドウで、**Title** フィールドのタイトルを変更します。
4. **Save** をクリックします。
   確認ボックスに、管理コンソールのリロードが必要であることが表示されます。
5. **Yes** をクリックします。
   システムは Web ブラウザーを自動的に更新し、新しいタイトルがタブヘッダーに表示されます。

3.5. 管理 API

3.5.1. HTTP API

HTTP API のエンドポイントは、HTTP プロトコルに依存して JBoss EAP 管理レイヤーと統合する管理クライアントのエントリーポイントです。

HTTP API は、JBoss EAP 管理コンソールによって使用されますが、他のクライアントの統合機能も提供します。デフォルトでは、**http://HOST_NAME:9990/management** で HTTP API にアクセスできます。この URL は、API に公開される raw 属性および値を表示します。

リソースの読み取り
HTTP **POST** メソッドを使用して他の操作を読み取り、書き込み、および実行できますが、**GET** リクエストを使用すると一部の読み取り操作を実行できます。HTTP **GET** メソッドは以下の URL 形式を使用します。

> http://**HOST_NAME**:9990/management/**PATH_TO_RESOURCE**?
> operation=**OPERATION**&**PARAMETER**=**VALUE**

置き換え可能な値は必ず適切な値に置き換えてください。置き換え可能な **OPERATION** の値は、以下の値に置き換えられます。

Value	説明
attribute	**read-attribute** 操作を実行します。
operation-description	**read-operation-description** 操作を実行します。
operation-names	**read-operation-names** 操作を実行します。
resource	**read-resource** 操作を実行します。
resource-description	**read-resource-description** 操作を実行します。
snapshots	**list-snapshots** 操作を実行します。

以下の URL 例は、HTTP API を使用して読み取り操作を実行する方法を示しています。

例: リソースに対するすべての属性および値の読み取り

```
http://HOST_NAME:9990/management/subsystem/undertow/server/default-server/http-listener/default
```

これは、**default** HTTP リスナーのすべての属性とそれらの値を表示します。

注記

デフォルトの操作は **read-resource** です。

例: リソースに対する属性の値の読み取り

```
http://HOST_NAME:9990/management/subsystem/datasources/data-source/ExampleDS?operation=attribute&name=enabled
```

これは、**ExampleDS** データソースの **enabled** 属性の値を読み取ります。

リソースの更新
HTTP **POST** メソッドを使用して設定値を更新するか、HTTP API を使用して他の操作を実行できます。これらの操作の認証を提供する必要があります。

以下の例は、HTTP API を使用してリソースを更新する方法を示しています。

例: リソースに対する属性の値の更新

```
$ curl --digest http://HOST_NAME:9990/management --header "Content-Type: application/json" -u USERNAME:PASSWORD -d '{"operation":"write-attribute", "address": ["subsystem","datasources","data-source","ExampleDS"], "name":"enabled", "value":"false", "json.pretty":"1"}'
```

これは、**ExampleDS** データソースの **enabled** 属性の値を **false** に更新します。

例: サーバーに対する操作の実行

```
$ curl --digest http://localhost:9990/management --header "Content-Type: application/json" -u USERNAME:PASSWORD -d '{"operation":"reload"}'
```

これは、サーバーをリロードします。

HTTP API を使用して JBoss EAP にアプリケーションをデプロイする方法については、HTTP API を使用したアプリケーションのデプロイ を参照してください

3.5.1.1. custom-constant HTTP ヘッダー

JBoss EAP の HTTP 管理エンドポイントは、クライアントに送信されるすべての応答で事前定義された HTTP ヘッダーのセットを返します。この事前定義された HTTP ヘッダーのセットに加えて、custom-constant HTTP ヘッダーを定義できます。

JBoss EAP は、以下のように custom-constant HTTP ヘッダーをリクエストに適用します。

- JBoss EAP は、リクエストパスに対して設定された接頭辞を照合して、custom-constant HTTP ヘッダーを適用します。
  たとえば、/ や /**management** などのリクエストパスのリクエストに、custom-constant HTTP ヘッダーをマップできます。

- リクエストが複数の接頭辞に一致する場合、JBoss EAP はすべてのマッピングから custom-constant HTTP ヘッダーを適用します。
  たとえば、パス **/management** へのリクエストは、/ と **/management** の両方のマッピングと一致します。JBoss EAP は両方のマッピングからヘッダーを適用します。

- リクエストの処理の最後に、応答がクライアントに返される前に、対応するエンドポイントによって設定されたヘッダーをオーバーライドします。
  たとえば、管理エンドポイントは各応答に **X-Frame-Options** ヘッダーを設定します。**X-Frame-Options** の名前で custom- constant HTTP ヘッダーを定義する場合、custom-constant HTTP ヘッダーがデフォルトのヘッダーを上書きします。

複数の custom-constant HTTP ヘッダーが、単一のマッピングのレスポンスで返されるように定義できます。

custom-constant HTTP ヘッダーを定義するルールは次のとおりです。

- custom-constant HTTP ヘッダーには、RFC-7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content でサポートされている文字のみを使用できます。

- 以下の事前定義された HTTP ヘッダーを上書きすることはできません。

  - **Connection**
  - **Content-Length**
  - **Content-Type**
  - **Date**
  - **Transfer-Encoding**

  事前定義されたヘッダーのいずれかを上書きすると、エラーが発生します。

  - たとえば、**Date** という名前で custom-constant HTTP ヘッダーを設定しようとすると、次のエラーが返されます。

    ```
    {
        "outcome" => "failed",
        "failure-description" => "WFLYCTL0458:Disallowed HTTP Header name 'Date'",
        "rolled-back" => true
    }
    ```

custom-constant HTTP ヘッダーを作成する際の重要な考慮事項:

- JBoss EAP は指定されたパスにアクセスできるかどうかを検証しません。

- サブシステムは、HTTP 管理インターフェイスがサポートするコンテキストを動的に追加できます。

- custom-constant HTTP ヘッダーは、エンドポイントがリクエストへの応答を処理する方法を変更しません。

関連情報

- Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content
- custom-constant HTTP ヘッダーの定義

- custom-constant HTTP ヘッダーを定義する CLI コマンド

3.5.1.2. custom-constant HTTP ヘッダーの定義

custom-constant HTTP ヘッダーが、要求されたパス接頭辞のリクエストに対するすべての応答で返されるように定義します。

重要

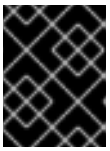
custom-constant HTTP ヘッダーを作成する前に、以下の考慮事項を理解する必要があります。

- JBoss EAP は指定されたパスにアクセスできるかどうかを検証しません。
- サブシステムは、HTTP 管理インターフェイスがサポートするコンテキストを動的に追加できます。
- custom-constant HTTP ヘッダーは、エンドポイントがリクエストへの応答を処理する方法を変更しません。

手順

1. custom-constant HTTP ヘッダーを定義します。

```
/core-service=management/management-interface=http-interface:write-attribute(name=constant-headers,value=[{path="PATH_PREFIX",headers=[{name="HEADER_NAME",value="HEADER_VALUE"}]}])
```



重要

write-attribute 操作を使用すると、**reload-required** プロンプトが表示されます。

2. 変更を反映するためにサーバーをリロードします。

```
reload
```

HTTP 管理インターフェイスへのリクエストが、事前定義された HTTP ヘッダーのセットに加えて、**HEADER_VALUE** の値で HTTP ヘッダー **HEADER_NAME** を返すようになりました。

custom-constant HTTP ヘッダー X-Help の例

```
/core-service=management/management-interface=http-interface:write-attribute(name=constant-headers,value=[{path="/",headers=[{name="X-Help",value="http://mywebsite.com/help"}]}])
```

検証手順

- HTTP 管理インターフェイスにリクエストを送信します。

```
$ curl -s -D - -o /dev/null --digest http://localhost:9990/management/ -u USERNAME:PASSWORD
```

custom-constant HTTP ヘッダー **X-Help** の例の応答例:

```
admin:redhat
HTTP/1.1 200 OK
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
Content-Type: application/json; charset=utf-8
Content-Length: 3312
X-Help: http://mywebsite.com
Date: Tue, 27 Oct 2020 08:13:17 GMT
```

応答には、**X-HELP** custom-constant HTTP ヘッダーが含まれます。

関連情報

- custom-constant HTTP ヘッダー
- custom-constant HTTP ヘッダーを定義する CLI コマンド

3.5.1.3. custom-constant HTTP ヘッダーを定義する CLI コマンド

以下の CLI コマンドは、スタンドアロンおよびマネージドドメインモードで custom-constant HTTP ヘッダーを定義します。

スタンドアロンモード

- 単一の custom-constant HTTP ヘッダーを定義するには、以下のコマンドを使用します。

```
/core-service=management/management-interface=http-interface:write-attribute(name=constant-headers,value=[{path=/PREFIX,headers=[{name=X-HEADER,value=HEADERVALUE}]}])
```

このコマンドにより、XML 設定は以下のようになります。

```
<management-interfaces>
    <http-interface security-realm="ManagementRealm">
        <http-upgrade enabled="true"/>
        <socket-binding http="management-http"/>
        <constant-headers>
            <header-mapping path="/PREFIX">
                <header name="X-HEADER" value="HEADERVALUE"/>
            </header-mapping>
        </constant-headers>
    </http-interface>
</management-interfaces>
```

- 複数の custom-constant HTTP ヘッダーを定義するには、以下のコマンドを使用します。

```
/core-service=management/management-interface=http-interface:write-attribute(name=constant-headers,value=[{path=/PREFIX1,headers=[{name=X-HEADER,value=HEADERVALUE-FOR-X}]},{path=/PREFIX2,headers=[{name=Y-HEADER,value=HEADERVALUE-FOR-Y}]}])
```

ドメインモード

- 単一の custom-constant HTTP ヘッダーを定義するには、以下のコマンドを使用します。

```
/host=master/core-service=management/management-interface=http-interface:write-attribute(name=constant-headers,value=[{path=/PREFIX,headers=[{name=X-HEADER,value=HEADER-VALUE}]}])
```

このコマンドにより、XML 設定は以下のようになります。

```
<management-interfaces>
    <http-interface security-realm="ManagementRealm">
        <http-upgrade enabled="true"/>
        <socket interface="management" port="${jboss.management.http.port:9990}"/>
        <constant-headers>
            <header-mapping path="/PREFIX">
                <header name="X-HEADER" value="HEADER-VALUE"/>
            </header-mapping>
        </constant-headers>
    </http-interface>
</management-interfaces>
```

- 複数の custom-constant HTTP ヘッダーを定義するには、以下のコマンドを使用します。

```
/host=master/core-service=management/management-interface=http-interface:write-attribute(name=constant-headers,value=[ {path=/PREFIX-1,headers=[{name=X-HEADER,value=HEADER-VALUE-FOR-X}]},{path=/PREFIX-2,headers=[{name=Y-HEADER,value=HEADER-VALUE-FOR-Y}]}])
```

関連情報

- custom-constant HTTP ヘッダー
- custom-constant HTTP ヘッダーの定義

3.5.2. ネイティブ API

ネイティブ API のエンドポイントは、ネイティブプロトコルに依存して JBoss EAP 管理レイヤーと統合する管理クライアントのエントリーポイントです。ネイティブ API は JBoss EAP 管理 CLI によって使用されますが、他のクライアントの統合機能も提供します。

以下の Java コードは、ネイティブ API を使用して Java コードから管理操作を実行する方法の例を示しています。

注記

EAP_HOME/bin/client/jboss-cli-client.jar ファイルにある、必要な JBoss EAP ライブラリーをクラスパスに追加する必要があります。

例: ネイティブ API を使用したリソースの読み取り

```
// Create the management client
ModelControllerClient client = ModelControllerClient.Factory.create("localhost", 9990);
```

```
// Create the operation request
ModelNode op = new ModelNode();

// Set the operation
op.get("operation").set("read-resource");

// Set the address
ModelNode address = op.get("address");
address.add("subsystem", "undertow");
address.add("server", "default-server");
address.add("http-listener", "default");

// Execute the operation and manipulate the result
ModelNode returnVal = client.execute(op);
System.out.println("Outcome: " + returnVal.get("outcome").toString());
System.out.println("Result: " + returnVal.get("result").toString());

// Close the client
client.close();
```

3.6. 設定データ

3.6.1. スタンドアロンサーバー設定ファイル

スタンドアロン設定ファイルは **EAP_HOME/standalone/configuration/** ディレクトリーにあります。事前定義されたプロファイルは 5 つあり (**default**、**ha**、**full**、**full-ha**、および **load-balancer**)、それぞれに個別のファイルが存在します。

表3.2 スタンドアロン設定ファイル

設定ファイル	目的
standalone.xml	このスタンドアロン設定ファイルは、スタンドアロンサーバーを起動したときに使用されるデフォルト設定です。このファイルには、サブシステム、ネットワーキング、デプロイメント、ソケットバインディング、およびその他の設定詳細など、サーバーに関するすべての情報が含まれます。メッセージングや高可用性に必要なサブシステムは提供しません。
standalone-ha.xml	このスタンドアロン設定ファイルには、デフォルトのサブシステムすべてが含まれ、高可用性の **modcluster** および **jgroups** サブシステムを追加します。メッセージングに必要なサブシステムは提供しません。
standalone-full.xml	このスタンドアロン設定ファイルには、デフォルトのサブシステムすべてが含まれ、**messaging-activemq** および **iiop-openjdk** サブシステムを追加します。高可用性に必要なサブシステムは提供しません。
standalone-full-ha.xml	このスタンドアロン設定ファイルには、メッセージングおよび高可用性を含むすべてのサブシステムのサポートが含まれます。
standalone-load-balancer.xml	このスタンドアロン設定ファイルには、ビルトインの mod_cluster フロントエンドロードバランサーを使用して他の JBoss EAP インスタンスの負荷を分散するために必要な最低限のサブシステムが含まれます。

デフォルトでは、スタンドアロンサーバーとして JBoss EAP を起動すると **standalone.xml** ファイルが使用されます。他の設定で JBoss EAP を起動するには **--server-config** 引数を使用します。以下に例を示します。

```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-full.xml
```

3.6.2. マネージドドメイン設定ファイル

マネージドドメインの設定ファイルは **EAP_HOME/domain/configuration/** ディレクトリーにあります。

表3.3 マネージドドメイン設定ファイル

設定ファイル	目的
domain.xml	これは、マネージドドメインの主要設定ファイルです。ドメインマスターのみがこのファイルを読み取ります。このファイルには、すべてのプロファイル (**default**、**ha**、**full**、**full-ha**、および **load-balancer**) の設定が含まれています。
host.xml	このファイルには、マネージドドメインの物理ホスト固有の設定情報が含まれています (ネットワークインターフェイス、ソケットバインディング、ホスト名、その他のホスト固有の詳細など)。**host.xml** ファイルには、**host-master.xml** および **host-slave.xml** (詳細は下記参照) の両方の機能がすべて含まれています。
host-master.xml	このファイルには、サーバーをマスタードメインコントローラーとして実行するために必要な設定情報のみが含まれています。
host-slave.xml	このファイルには、サーバーをマネージドドメインのホストコントローラーとして実行するために必要な設定情報のみが含まれています。

デフォルトでは、JBoss EAP をマネージドドメインで起動すると **host.xml** ファイルが使用されます。他の設定で JBoss EAP を起動するには **--host-config** 引数を使用します。以下に例を示します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-master.xml
```

3.6.3. 設定データのバックアップ

JBoss EAP のサーバー設定を後で復元するため、以下の場所にあるものはバックアップしておく必要があります。

- **EAP_HOME/standalone/configuration/**
  - ディレクトリー全体をバックアップして、スタンドアロンサーバーのユーザーデータ、サーバー設定、およびロギング設定を保存します。
- **EAP_HOME/domain/configuration/**
  - ディレクトリー全体をバックアップして、マネージドドメインのユーザーおよびプロファイルデータ、ドメインおよびホスト設定、およびロギング設定を保存します。
- **EAP_HOME/modules/**

- カスタムモジュールをバックアップします。

- **EAP_HOME/welcome-content/**
  - カスタムのウェルカムコンテンツをバックアップします。

- **EAP_HOME/bin/**
  - カスタムスクリプトまたは起動設定ファイルをバックアップします。

3.6.4. 設定ファイルのスナップショット

サーバーの保守や管理をしやすくするため、JBoss EAP は起動時に元の設定ファイルにタイムスタンプを付けたものを作成します。管理操作によってその他の設定変更が行われると、元のファイルが自動的にバックアップされ、インスタンスの作業用コピーが参照およびロールバック用に保持されます。さらに、現在のサーバー設定の現時点のコピーである設定スナップショットを撮ることができます。これらのスナップショットは管理者によって保存およびロードされます。

以下の例では、**standalone.xml** ファイルが使用されますが、同じプロセスが **domain.xml** および **host.xml** にも適用されます。

スナップショットの作成
管理 CLI を使用して、現在の設定のスナップショットを作成します。

```
:take-snapshot
{
    "outcome" => "success",
    "result" => "EAP_HOME/standalone/configuration/standalone_xml_history/snapshot/20151022-
133109702standalone.xml"
}
```

スナップショットのリスト
管理 CLI を使用して、作成したすべてのスナップショットをリストします。

```
:list-snapshots
{
    "outcome" => "success",
    "result" => {
        "directory" => "EAP_HOME/standalone/configuration/standalone_xml_history/snapshot",
        "names" => [
            "20151022-133109702standalone.xml",
            "20151022-132715958standalone.xml"
        ]
    }
}
```

スナップショットの削除
管理 CLI を使用して、スナップショットを削除します。

```
:delete-snapshot(name=20151022-133109702standalone.xml)
```

スナップショットを用いたサーバーの起動
スナップショットまたは自動保存された設定を使用してサーバーを起動できます。

1. **EAP_HOME/standalone/configuration/standalone_xml_history** ディレクトリーへ移動し、ロードするスナップショットまたは保存された設定ファイルを確認します。

2. サーバーを起動し、選択した設定ファイルを示します。設定ディレクトリー **EAP_HOME/standalone/configuration/** からの相対パスを渡します。

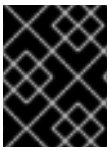
```
$ EAP_HOME/bin/standalone.sh --server-config=standalone_xml_history/snapshot/20151022-133109702standalone.xml
```

注記

マネージドドメインで実行している場合は、代わりに **--host-config** 引数を使用し、設定ファイルを指定します。

3.6.5. 設定変更の確認

JBoss EAP 7 には、稼働中のシステムに加えられた設定変更を追跡する機能があります。この機能を使用すると、管理者は他の許可されたユーザーが追加した設定変更の履歴を確認することができます。



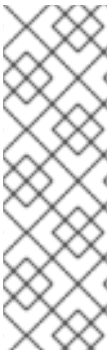
重要

変更はメモリーに保存され、サーバーを再起動すると永続化されません。この機能は 管理監査ログ の代替機能ではありません。

管理 CLI または 管理コンソール から設定変更の追跡と表示を有効にできます。

管理 **CLI** からの設定変更の追跡および表示
設定変更の追跡を有効にするには、以下の管理 CLI コマンドを使用します。**max-history** 属性を使用すると保存するエントリーの数を指定できます。

```
/subsystem=core-management/service=configuration-changes:add(max-history=20)
```



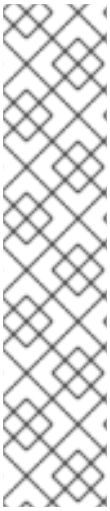
注記

マネージドドメインでは、ホストおよびサーバー関連の設定変更はホストレベルで追跡されます。ホストコントローラーの設定変更を可能にすると、そのホストコントローラーが管理するサーバーすべてで設定の変更が可能になります。以下のコマンドを使用すると、ホストごとに設定の変更を追跡できます。

```
/host=HOST_NAME/subsystem=core-management/service=configuration-changes:add(max-history=20)
```

最近行われた設定変更のリストを表示するには、以下の管理 CLI コマンドを使用します。

```
/subsystem=core-management/service=configuration-changes:list-changes
```



注記

マネージドドメインでは、以下のコマンドを使用してホストの設定変更をリストできます。

```
/host=HOST_NAME/subsystem=core-management/service=configuration-changes:list-changes
```

以下のコマンドを使用すると、特定のサーバーに影響する設定の変更をリストできます。

```
/host=HOST_NAME/server=SERVER_NAME/subsystem=core-management/service=configuration-changes:list-changes
```

このコマンドは、各設定変更とその変更日、変更元、結果、および操作の詳細をリストで表示します。たとえば、以下の **list-changes** コマンドの出力は、変更日が新しい順に設定変更を表示しています。

```
{
    "outcome" => "success",
    "result" => [
        {
            "operation-date" => "2016-02-12T18:37:00.354Z",
            "access-mechanism" => "NATIVE",
            "remote-address" => "127.0.0.1/127.0.0.1",
            "outcome" => "success",
            "operations" => [{
                "address" => [],
                "operation" => "reload",
                "operation-headers" => {
                    "caller-type" => "user",
                    "access-mechanism" => "NATIVE"
                }
            }]
        },
        {
            "operation-date" => "2016-02-12T18:34:16.859Z",
            "access-mechanism" => "NATIVE",
            "remote-address" => "127.0.0.1/127.0.0.1",
            "outcome" => "success",
            "operations" => [{
                "address" => [
                    ("subsystem" => "datasources"),
                    ("data-source" => "ExampleDS")
                ],
                "operation" => "write-attribute",
                "name" => "enabled",
                "value" => false,
                "operation-headers" => {
                    "caller-type" => "user",
                    "access-mechanism" => "NATIVE"
                }
            }]
        },
        {
```

```
            "operation-date" => "2016-02-12T18:24:11.670Z",
            "access-mechanism" => "HTTP",
            "remote-address" => "127.0.0.1/127.0.0.1",
            "outcome" => "success",
            "operations" => [{
                "operation" => "remove",
                "address" => [
                    ("subsystem" => "messaging-activemq"),
                    ("server" => "default"),
                    ("jms-queue" => "ExpiryQueue")
                ],
                "operation-headers" => {"access-mechanism" => "HTTP"}
            }]
        }
    ]
}
```

この例は、設定に影響した以下 3 つの操作の詳細を表しています。

- 管理 CLI から行ったサーバーのリロード。
- 管理 CLI から行った **ExampleDS** データソースの無効化。
- 管理コンソールから行った **ExpiryQueue** キューの削除。

管理コンソールからの設定変更の追跡および表示
管理コンソールからの設定変更の追跡を有効にするには、**Runtime** タブを選択して、変更を追跡するサーバーまたはホストに移動し、ドロップダウンメニューで 設定変更 を選択します。**Enable Configuration Changes** をクリックし、最大の履歴値を指定します。

そのページの表に変更された設定が日付、変更元、結果、および操作詳細とともに表示されます。

3.6.6. プロパティーの置き換え

JBoss EAP では、設定のリテラル値の代わりに式を使用して置換可能なプロパティーを定義できます。式の形式は **${PARAMETER:DEFAULT_VALUE}** になります。指定のパラメーターが設定されると、パラメーターの値が使用されます。設定されない場合は、デフォルト値が使用されます。

式の解決でサポートされるリソースはシステムプロパティー、環境変数、および vault になります。デプロイメントの場合のみ、デプロイメントアーカイブの **META-INF/jboss.properties** ファイルにリストされたプロパティーをソースとすることができます。サブデプロイメントをサポートするデプロイメントタイプでは、プロパティーファイルが EAR などの外部のデプロイメントにある場合は解決がすべてのサブデプロイメントに対してスコープ指定されます。プロパティーファイルがサブデプロイメントにある場合は、解決はそのサブデプロイメントのみに対してスコープ指定されます。

以下の例では、**jboss.bind.address** パラメーターが設定されていなければ、**standalone.xml** 設定ファイルによって **public** インターフェイスの **inet-address** が **127.0.0.1** に設定されます。

```
<interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}"/>
</interface>
```

以下のコマンドを使用して、EAP をスタンドアロンサーバーとして起動するときに **jboss.bind.address** パラメーターを設定できます。

```
$ EAP_HOME/bin/standalone.sh -Djboss.bind.address=IP_ADDRESS
```

ネストされた式
式はネストすることができるため、固定値の代わりにさらに高度な式を使用できます。ネストされた式の書式は、通常の式の場合と同様ですが、ある式が別の式に組み込まれます。 例を以下に示します。

```
${SYSTEM_VALUE_1${SYSTEM_VALUE_2}}
```

ネストされた式は、再帰的に評価されるため、最初に 内部の式が評価され、次に 外部の式が評価されます。式が別の式へ解決する場合は式も再帰的になることがあり、その後解決されます。ネストされた式は式が許可された場所ならどこでも許可されます (ただし、管理 CLI コマンドを除く)。

ネストされた式が使用される例としては、データソース定義で使用されるパスワードがマスクされている場合などがあります。データソースの設定には以下のような行がある場合があります。

```
<password>${VAULT::ds_ExampleDS::password::1}</password>
```

この場合、ネストされた式を使用すると、**ds_ExampleDS** の値をシステムプロパティー (**datasource_name**) に置き換えることができます。上記の行の代わりに以下の行をデータソースの設定に使用できます。

```
<password>${VAULT::${datasource_name}::password::1}</password>
```

JBoss EAP は、最初に式 **${datasource_name}** を評価し、次にこれを外側の大きい式に入力して、結果となる式を評価します。この設定の利点は、データソースの名前が固定された設定から抽象化されることです。

記述子ベースのプロパティー置換
データソース接続パラメーターなどのアプリケーションの設定は、通常は開発デプロイメント、テストデプロイメント、および本番環境によって異なります。Jakarta EE 仕様にはこれらの設定を外部化するメソッドが含まれていないため、このような違いはビルドシステムスクリプトで対応することがあります。JBoss EAP では、記述子ベースのプロパティー置換を使用して設定を外部的に管理できます。

記述子ベースのプロパティー置換は、記述子を基にプロパティーを置き換えるため、アプリケーションやビルドチェーンから環境に関する仮定を除外できます。環境固有の設定は、アノテーションやビルドシステムスクリプトでなく、デプロイメント記述子に指定できます。設定はファイルに指定したり、パラメーターとしてコマンドラインで提供したりできます。

ee サブシステムには、プロパティー置換が適用されたかどうかを制御する複数のフラグがあります。

JBoss 固有の記述子置換は **jboss-descriptor-property-replacement** フラグによって制御され、デフォルトで有効になっています。有効にすると、以下のデプロイメント記述子でプロパティーを置換できます。

- **jboss-ejb3.xml**
- **jboss-app.xml**
- **jboss-web.xml**
- **jboss-permissions.xml**
- ***-jms.xml**
- ***-ds.xml**

以下の管理 CLI コマンドを使用すると、JBoss 固有の記述子でプロパティー置換を有効または無効にできます。

```
/subsystem=ee:write-attribute(name="jboss-descriptor-property-replacement",value=VALUE)
```

Jakarta EE の記述子置換は **spec-descriptor-property-replacement** フラグによって制御され、デフォルトで無効になっています。有効にすると、以下のデプロイメント記述子でプロパティーを置換できます。

- **ejb-jar.xml**
- **permissions.xml**
- **persistence.xml**
- **application.xml**
- **web.xml**

以下の管理 CLI コマンドを使用すると、Jakarta EE の記述子でプロパティー置換を有効または無効にできます。

```
/subsystem=ee:write-attribute(name="spec-descriptor-property-replacement",value=VALUE)
```

3.6.7. Git を使用した設定データの管理

JBoss EAP 7.3 より、Git を使用してサーバー設定データ、プロパティーファイル、およびデプロイメントを管理および永続化できるようになりました。これにより、これらのファイルのバージョン履歴を管理できるだけでなく、1 つ以上の Git リポジトリーを使用して複数のサーバーおよびノード全体でサーバーやアプリケーションの設定を共有することができます。この機能は、デフォルトの設定ディレクトリーレイアウトを使用するスタンドアロンサーバーでのみ動作します。

ローカル Git リポジトリー で設定データの使用を選択でき、remote Git repository からデータを取得することもできます。Git リポジトリーは、スタンドアロンサーバーコンテンツのベースディレクトリーである **jboss.server.base.dir** ディレクトリーで設定されます。**jboss.server.base.dir** ディレクトリーが Git を使用するよう設定されると、JBoss EAP は管理 CLI または管理コンソールを使用して設定へ加えられた各更新を自動的にコミットします。設定ファイルを手作業で編集してサーバー外部で追加された変更は、コミットおよび永続化されません。しかし、Git CLI を使用して手作業の変更を追加およびコミットすることができます。また、Git CLI を使用してコミット履歴の表示、ブランチの管理、およびコンテンツの管理を行うこともできます。

この機能を使用するには、サーバーの起動時に以下の引数を 1 つ以上コマンドラインに渡します。

表3.4 Git 設定管理のサーバー起動引数

引数	説明
--git-repo	サーバー設定データの管理および永続化に使用される Git リポジトリーの場所。これは、ローカルで保存する場合は **local** を指定し、リモートの場合はリモートリポジトリーへの URL を指定します。

引数	説明
--git-branch	使用する Git リポジトリーのブランチまたはタグ名。ブランチまたはタグ名が存在**しない**と作成されないため、この引数には既存のブランチまたはタグ名を指定する必要があります。タグ名を使用する場合、リポジトリーを detached HEAD 状態にし、今後のコミットがブランチにアタッチされないようにします。タグ名は読み取り専用で、通常複数のノード全体で設定をレプリケートする必要があるときに使用されます。
--git-auth	Elytron 設定ファイルへの URL には、リモート Git リポジトリーへの接続時に使用される認証情報が含まれています。この引数は、Git リポジトリーに認証が必要な場合に必要となります。この引数は **local** リポジトリーとは使用されません。

ローカル **Git** リポジトリーの使用
ローカル Git リポジトリーを使用するには、**--git-repo=local** 引数を使用してサーバーを起動します。サーバーの起動時に **--git-branch=GIT_BRANCH_NAME** 引数を追加して、オプションのブランチまたはタグ名をリモートリポジトリーで指定することもできます。ブランチまたはタグ名が存在しないと作成されないため、この引数には既存のブランチまたはタグ名を指定する必要があります。タグ名を使用する場合、リポジトリーを detached HEAD 状態にし、今後のコミットがブランチにアタッチされないようにします。

以下のコマンド例は、**local** リポジトリーの **1.0.x** ブランチを使用して、サーバーを起動します。

```
$ EAP_HOME/bin/standalone.sh --git-repo=local --git-branch=1.0.x
```

local Git リポジトリーを使用する引数を使用してサーバーを起動した場合、JBoss EAP は **jboss.server.base.dir** ディレクトリーがすでに Git に対して設定されているかどうかを確認します。設定されていない場合、JBoss EAP は既存の設定コンテンツを使用して **jboss.server.base.dir** ディレクトリーに Git リポジトリーを作成し、初期化します。JBoss EAP は **--git-branch** 引数によって渡されたブランチ名をチェックアウトします。引数が渡されていない場合は **master** ブランチをチェックアウトします。初期化後、スタンドアロンサーバーコンテンツのベースディレクトリーに **.git/** ディレクトリーと **.gitignore** ファイルがあることを確認できるはずです。

リモート **Git** リポジトリーの使用
Git リポジトリーを使用するには、**--git-repo=REMOTE_REPO** 引数を使用してサーバーを起動します。引数の値は、ローカル Git 設定に手作業で追加した URL またはリモートエイリアスになります。

サーバーの起動時に **--git-branch=GIT_BRANCH_NAME** 引数を追加して、オプションのブランチまたはタグ名をリモートリポジトリーで指定することもできます。ブランチまたはタグ名が存在しないと作成されないため、この引数には既存のブランチまたはタグ名を指定する必要があります。タグ名を使用する場合、リポジトリーを detached HEAD 状態にし、今後のコミットがブランチにアタッチされないようにします。

Git リポジトリーに認証が必要な場合は、サーバーの起動時に **--git-auth=AUTH_FILE_URL** 引数も追加する必要があります。この引数は、Git リポジトリーへの接続に必要な認証情報が含まれる Elytron 設定ファイルへの URL になります。以下は、認証に使用できる Elytron 設定ファイルの例になります。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <authentication-rules>
```

```
      <rule use-configuration="test-login">
      </rule>
    </authentication-rules>
    <authentication-configurations>
      <configuration name="test-login">
        <sasl-mechanism-selector selector="BASIC" />
        <set-user-name name="eap-user" />
        <credentials>
          <clear-password password="my_api_key" />
        </credentials>
        <set-mechanism-realm name="testRealm" />
      </configuration>
    </authentication-configurations>
  </authentication-client>
</configuration>
```

以下は、リモート **eap-configuration** リポジトリーの **1.0.x** ブランチを使用して認証情報が含まれる Elytron 設定ファイルに URL を渡し、フルプロファイルでサーバーを起動するコマンドの例になります。

```
$ EAP_HOME/bin/standalone.sh --git-repo=https://github.com/MY_GIT_ID/eap-configuration.git --git-branch=1.0.x --git-auth=file:///home/USER_NAME/github-wildfly-config.xml --server-config=standalone-full.xml
```

リモート Git リポジトリーを使用する引数を使用してサーバーを起動した場合、JBoss EAP は **jboss.server.base.dir** ディレクトリーがすでに Git に対して設定されているかどうかを確認します。設定されていない場合、JBoss EAP は **jboss.server.base.dir** ディレクトリーにある既存の設定ファイルを削除し、代わりにリモート Git 設定データを置きます。JBoss EAP は **--git-branch** 引数によって渡されたブランチ名をチェックアウトします。引数が渡されていない場合は **master** ブランチをチェックアウトします。この処理の完了後、スタンドアロンサーバーコンテンツのベースディレクトリーに **.git/** ディレクトリーと **.gitignore** ファイルがあることを確認できるはずです。

警告

この処理の完了後、当初使用したものとは異なる **--git-repo** URL または **--git-branch** 名を渡して、サーバーを起動すると、サーバーの起動時にエラーメッセージ **java.lang.RuntimeException: WFLYSRV0268: Failed to pull the repository GIT_REPO_NAME** が表示されます。これは、JBoss EAP が **jboss.server.base.dir** ディレクトリーに現在設定されていないリポジトリーとブランチから設定データをプルしようとし、競合が発生するためです。

リモート **Git SSH** リポジトリーの使用
SSH 認証では、SSH 認証情報を指定して elytron 設定ファイルを指定できます。このファイルで SSH 認証情報を指定した後に、スタンドアロンサーバーインスタンスを起動し、リモートの Git SSH リポジトリーでサーバー設定ファイル履歴を管理できます。

また、**elytron-tool.sh** スクリプトを使用してアクセスできる WildFly Elytron ツールを使用して SSH キーペアを生成し、クレデンシャルストアに保存することもできます。WildFly Elytron ツールは、サーバーに SSH 認証情報を指定していない場合に使用する場合に便利です。

elytron 設定ファイルに認証情報を追加すると、リモート Git SSH リポジトリーに接続できます。

前提条件

- **elytron** 設定ファイルに認証情報を追加している。elytron 設定ファイル で保存されたキーペアの使用 を参照し、**elytron** 設定ファイル での OpenSSH キーの使用 を 参照してください。

手順

- ターミナルで以下のコマンドを実行して、リモートの git SSH リポジトリーに接続します。

```
$ <eap_home_path>/bin/standalone.sh --git-repo=<git_repository_url> --git-auth=<elytron_configuration_file_url>
```

スタンドアロンサーバーが起動し、サーバーの設定ファイル履歴がリモート Git SSH リポジトリーで管理されるようになります。

関連情報

- **elytron-tool.sh** スクリプトで SSH キーペアを生成する方法は、サーバーのセキュリティーの設定方法 ガイドの WildFly Elytron ツールを使用したクレデンシャルストアでのキーペアの管理 を参照してください。
- OpenSSH キーペアの生成および使用に関する詳細は、**elytron** 設定ファイルでの OpenSSH keys 使用 を参照してください。

elytron 設定ファイルでの **OpenSSH** キーの使用
elytron サブシステムは、OpenSSH コマンドラインツールを使用して生成された SSH キーペアをサポートします。このツールは、RSA アルゴリズム、DSA アルゴリズム、および ECDSA アルゴリズムを使用します。

ssh-keygen コマンドを使用して、SSH 鍵ペアを生成できます。

さらに、3 つの要素タイプのいずれかを使用してパスワードを指定することもできます。

- **clear-password**
- **masked-password**
- **credential-store-reference**

前提条件

- SSH キーペアを生成している。以下の例は、サイズが **256** メガバイトの ECDSA 鍵の生成を示しています。パスフレーズは シークレット として設定されている。

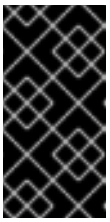
```
[~/.ssh]$ ssh-keygen -t ecdsa -b 256

Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase): secret
Enter same passphrase again: secret
Your identification has been saved in /home/user/.ssh/id_ecdsa.
Your public key has been saved in /home/user/.ssh/id_ecdsa.pub.
```

手順

- 以下の 2 つの方法のいずれかを選択して、**elytron** 設定ファイルでキーペアを指定します。
  - キーペアの認証情報を使用して、設定ファイルでキーペアを指定します。以下に例を示します。

```
<authentication-configurations>
    <configuration name="example">
        <credentials>
            <key-pair>
                <openssh-private-key pem="-----BEGIN OPENSSH PRIVATE KEY-----

b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABDaZzGp
GV

922xmrL+bMHioPAAAAEAAAAAEAAABoAAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIb
mlz

dHAyNTYAAABBBIMTU1m6pmpnSTZ2k/cbKnxXkRpXUmWwqN1SSNLpRswGsUhmLG2
H21br1Z

lEHRiRn6zQmA4YCtCw2hLuz8M8WVoAAADAQk+bMNWFfaI4Ej1AQdlLl6v4RDa2HGjDS
3V4

39h0pOx4Ix7YZKydTN4SPkYRt78CNK0AhhtKsWo2lVNwyfh8/6SeqowhgCG9MJYW8yRR
1R

3DX/eQTx6MV/gSSRLDTpcVWUY0jrBGpMaEvylKoNcabiEo44flklYlG6E/YtFXsmXsoBsj

nFcjvmfE7Lzyin5Fowwpbqj9f0XOARu9wsUzeyJVAwT7+YCU3mWJ3dnO1bOxK4TuLsxD6
j
                                                RB7bJemsfr
                                                -----END OPENSSH PRIVATE KEY-----">
                    <clear-password password="secret"/>
                </openssh-private-key>
            </key-pair>
        </credentials>
    </configuration>
</authentication-configurations>
```

この例は、OpenSSH 形式のキーペアを示しています。シークレット のパスフレーズは **clear-password** タイプとして設定されており、秘密鍵を復号化する必要があります。



重要

elytron サブシステムは、**PKCS8** 形式のキーペアをサポートします。ただし、キーペアを元の形式に復号化する必要がある場合に問題が発生する可能性があるため、**PKCS8** 形式のキーペアをパスフレーズで暗号化しないでください。

  - 設定ファイルの **<ssh-credential>** 要素に秘密鍵が含まれるファイルの場所を指定します。以下に例を示します。

```
<authentication-configurations>
```

```
    <configuration name="example">
      <credentials>
        <ssh-credential ssh-directory="/user/home/example/.ssh" private-key-
file="id_test_ecdsa" known-hosts-file="known_hosts_test"> (1) (2) (3)
          <clear-password password="secret"/>
        </ssh-credential>
      </credentials>
    </configuration>
</authentication-configurations>
```

(1) **ssh-directory** 属性は、キーの場所と既知のホストファイルの場所を指定します。

(2) **private-key-file** 属性は、秘密鍵を含むファイルの名前を指定します。

(3) **known-hosts-file** 属性は、既知の SSH ホストを含むファイルの名前を指定します。

関連情報

- OpenSSH とその機能の詳細は、OpenSSH のドキュメント を参照してください。

Git 使用時のリモート設定データの公開
管理 CLI の **publish-configuration** 操作を使用すると、Git リポジトリーの変更をリモートリポジトリーにプッシュすることができます。JBoss EAP は、サーバー起動時のブート処理中にリモート Git リポジトリーから設定をプルするため、複数のサーバー間で設定データを共有できます。この操作はリモートリポジトリーにのみ使用できます。ローカルリポジトリーでは動作しません。

以下の管理 CLI 操作は、設定データをリモート **eap-configuration** リポジトリーに公開します。

```
:publish-configuration(location="=https://github.com/MY_GIT_ID/eap-configuration.git")
{"outcome" => "success"}
```

Git でのスナップショットの使用
Git のコミット履歴を使用して設定の変更を追跡する他に、スナップショットを作成して特定時の設定を保持することもできます。スナップショットをリスト表示し、削除することができます。

Git 使用時におけるスナップショットの作成
スナップショットは、Git にタグとして保存されます。**take-snapshot** 操作で、スナップショットタグ名とコミットメッセージを引数として指定します。

以下の管理 CLI 操作は、スナップショットを作成し、タグに snapshot-01 という名前を付けます。

```
:take-snapshot(name="snapshot-01", comment="1st snapshot")
{
    "outcome" => "success",
    "result" => "1st snapshot"
}
```

Git 使用時におけるスナップショットのリスト表示
list-snapshots 操作を使用すると、スナップショットタグをすべてリスト表示できます。

以下の管理 CLI 操作は、スナップショットタグをリスト表示します。

```
:list-snapshots
{
```

```
    "outcome" => "success",
    "result" => {
        "directory" => "",
        "names" => [
            "snapshot : 1st snapshot",
            "refs/tags/snapshot-01",
            "snapshot2 : 2nd snapshot",
            "refs/tags/snapshot-02"
        ]
    }
}
```

Git 使用時におけるスナップショットの削除
delete-snapshot 操作にタグ名を渡すと特定のスナップショットを削除できます。

以下の管理 CLI 操作は、タグ名が snapshot-01 のスナップショットを削除します。

```
:delete-snapshot(name="snapshot-01")
{"outcome" => "success"}
```

3.7. ファイルシステムパス

JBoss EAP はファイルシステムパスに論理名を使用します。他の設定は論理名を使用してパスを参照できます。そのため、各インスタンスに完全パスを使用する必要がなく、特定のホスト設定が汎用論理名に解決することができます。

たとえば、デフォルトの **logging** サブシステム設定は **jboss.server.log.dir** をサーバーログディレクトリーの論理名として宣言します。

例: サーバーログディレクトリーの相対パスの例

```
<file relative-to="jboss.server.log.dir" path="server.log"/>
```

JBoss EAP では、複数の標準的なパスが自動的に提供されるため、ユーザーが設定ファイルでこれらのパスを設定する必要はありません。

表3.5 標準的なパス

プロパティー	説明
java.home	Java インストールディレクトリー。
jboss.controller.temp.dir	スタンドアロンサーバーおよびマネージドドメインの共通のエイリアス。ディレクトリーは一時ファイルのストレージとして使用されます。マネージドドメインの **jboss.domain.temp.dir** と スタンドアロンサーバーの **jboss.server.temp.dir** と同等です。
jboss.domain.base.dir	ドメインコンテンツのベースディレクトリー。
jboss.domain.config.dir	ドメイン設定が含まれるディレクトリー。
jboss.domain.data.dir	ドメインが永続データファイルの格納に使用するディレクトリー。

プロパティー	説明
jboss.domain.log.dir	ドメインが永続ログファイルの格納に使用するディレクトリー。
jboss.domain.temp.dir	ドメインが一時ファイルの格納に使用するディレクトリー。
jboss.domain.deployment.dir	ドメインがデプロイ済みコンテンツの格納に使用するディレクトリー。
jboss.domain.servers.dir	ドメインがマネージドドメインインスタンスの出力を格納するために使用するディレクトリー。
jboss.home.dir	JBoss EAP ディストリビューションのルートディレクトリー。
jboss.server.base.dir	スタンドアロンサーバーコンテンツのベースディレクトリー。
jboss.server.config.dir	スタンドアロンサーバー設定が含まれるディレクトリー。
jboss.server.data.dir	スタンドアロンサーバーが永続データファイルの格納に使用するディレクトリー。
jboss.server.log.dir	スタンドアロンサーバーがログファイルの格納に使用するディレクトリー。
jboss.server.temp.dir	スタンドアロンサーバーが一時ファイルの格納に使用するディレクトリー。
jboss.server.deploy.dir	スタンドアロンサーバーがデプロイ済みコンテンツを格納するために使用するディレクトリー。
user.dir	ユーザーのカレントワーキングディレクトリー。
user.home	ユーザーのホームディレクトリー。

標準パスの上書き または カスタムパスの追加 を行うことができます。

3.7.1. ファイルシステムパスの表示

以下の管理 CLI コマンドを使用して、ファイルシステムパスのリストを表示します。

```
ls /path
```

注記

マネージドドメインでは、以下の管理 CLI コマンドを使用して、特定のサーバーのファイルシステムパスをリストできます。

```
ls /host=HOST_NAME/server=SERVER_NAME/path
```

以下の管理 CLI コマンドを使用して、ファイルシステムパスの値を読み取ります。

```
/path=PATH_NAME:read-resource
```

注記

マネージドドメインでは、以下の管理 CLI コマンドを使用して、特定サーバーのファイルシステムパスの値を読み取りできます。

```
/host=HOST_NAME/server=SERVER_NAME/path=PATH_NAME:read-resource
```

3.7.2. 標準パスの上書き

jboss.server.* または **jboss.domain.*** で始まる標準パスのデフォルトの場所を上書きできます。これには 2 つの方法があります。

- サーバーの起動時にコマンドライン引数を渡します。以下に例を示します。

```
$ EAP_HOME/bin/standalone.sh -Djboss.server.log.dir=/var/log
```

- **standalone.conf** または **domain.conf** のいずれかのサーバー設定ファイルで **JAVA_OPTS** 変数を変更し、新しい場所が含まれるようにします。以下に例を示します。

```
JAVA_OPTS="$JAVA_OPTS -Djboss.server.log.dir=/var/log"
```

マネージドドメインの標準パスの上書き
この例の目的は、**/opt/jboss_eap/domain_data** ディレクトリーにドメインファイルを格納し、各トップレベルディレクトリーにカスタム名を付けることです。デフォルトのディレクトリーグルーピングである **by-server** が使用されます。

- ログファイルは **all_logs** サブディレクトリーに格納します。
- データファイルは **all_data** サブディレクトリーに格納します。
- 一時ファイルは **all_temp** サブディレクトリーに格納します。
- サーバーのファイルは **all_servers** サブディレクトリーに格納します。

この設定を行うには、JBoss EAP の起動時に複数のシステムプロパティーを上書きします。

```
$ EAP_HOME/bin/domain.sh -Djboss.domain.temp.dir=/opt/jboss_eap/domain_data/all_temp -
Djboss.domain.log.dir=/opt/jboss_eap/domain_data/all_logs -
Djboss.domain.data.dir=/opt/jboss_eap/domain_data/all_data -
Djboss.domain.servers.dir=/opt/jboss_eap/domain_data/all_servers
```

この結果、パス構造は次のようになります。

```
/opt/jboss_eap/domain_data/
              ├── all_data
              ├── all_logs
              ├── all_servers
              │    ├── server-one
              │    │    ├── data
```

```
│   │   ├── log
│   │   └── tmp
│   └── server-two
│       ├── data
│       ├── log
│       └── tmp
└── all_temp
```

3.7.3. カスタムパスの追加

管理 CLI または管理コンソールを使用してカスタムのファイルシステムパスを追加できます。

- 管理 CLI の場合、以下の管理 CLI コマンドを使用して新しいパスを追加できます。

```
/path=my.custom.path:add(path=/my/custom/path)
```

- 管理コンソールからファイルシステムパスを設定するには、設定タブに移動し、パスを選択して表示をクリックします。ここからは、パスを追加、変更、および削除できます。

このカスタムパスを設定で使用できます。たとえば、以下のログハンドラーは相対パスにカスタムパスを使用します。

```
<subsystem xmlns="urn:jboss:domain:logging:6.0">
  ...
  <periodic-rotating-file-handler name="FILE" autoflush="true">
    <formatter>
      <named-formatter name="PATTERN"/>
    </formatter>
    <file relative-to="my.custom.path" path="server.log"/>
    <suffix value=".yyyy-MM-dd"/>
    <append value="true"/>
  </periodic-rotating-file-handler>
  ...
</subsystem>
```

3.8. ディレクトリーのグループ化

マネージドドメインでは、各サーバーのファイルは **EAP_HOME/domain** ディレクトリーに格納されます。ホストコントローラーの **directory-grouping** 属性を使用すると、サーバーのサブディレクトリーの編成を指定できます。ディレクトリーはサーバーまたはタイプを基にしてグループ化できます。デフォルトではディレクトリーはサーバーを基にしてグループ化されます。

サーバーを基にしたディレクトリーのグループ化
デフォルトでは、ディレクトリーはサーバーを基にしてグループ化されます。管理作業がサーバー中心である場合はこの設定が推奨されます。たとえば、サーバーインスタンスごとにバックアップやログファイルの処理を設定することができます。

ZIP インストールで JBoss EAP がインストールされた場合、デフォルトのディレクトリー構造 (サーバーによるグループ化) は次のようになります。

```
EAP_HOME/domain
        └── servers
            ├── server-one
            │   ├── data
```

```
        ├── tmp
        └── log
    └── server-two
        ├── data
        ├── tmp
        └── log
```

サーバーを基にしてドメインディレクトリーをグループ化するには、以下の管理 CLI コマンドを入力します。

```
/host=HOST_NAME:write-attribute(name=directory-grouping,value=by-server)
```

このコマンドにより、ホストコントローラーの **host.xml** 設定ファイルが更新されます。

```
<servers directory-grouping="by-server">
  <server name="server-one" group="main-server-group"/>
  <server name="server-two" group="main-server-group" auto-start="true">
    <socket-bindings port-offset="150"/>
  </server>
</servers>
```

タイプを基にしたディレクトリーのグループ化

サーバーを基にディレクトリーをグループ化する代わりに、ファイルタイプを基にしてグループ化することもできます。管理作業がファイルタイプ中心である場合は、この設定が推奨されます。たとえば、**data** ファイルのみを簡単にバックアップすることができます。

ZIP インストールで JBoss EAP がインストールされ、ドメインのファイルがタイプを基にグループ化された場合、ディレクトリー構造は次のようになります。

```
EAP_HOME/domain
    ├── data
    │   └── servers
    │       ├── server-one
    │       └── server-two
    ├── log
    │   └── servers
    │       ├── server-one
    │       └── server-two
    └── tmp
        └── servers
            ├── server-one
            └── server-two
```

タイプを基にしてドメインディレクトリーをグループ化するには、以下の管理 CLI コマンドを入力します。

```
/host=HOST_NAME:write-attribute(name=directory-grouping,value=by-type)
```

このコマンドにより、ホストコントローラーの **host.xml** 設定ファイルが更新されます。

```
<servers directory-grouping="by-type">
  <server name="server-one" group="main-server-group"/>
  <server name="server-two" group="main-server-group" auto-start="true">
```

```
    <socket-bindings port-offset="150"/>
  </server>
</servers>
```

3.9. システムプロパティー

Java システムプロパティーを使用すると、JBoss EAP の多くのオプションや、アプリケーションサーバー内で使用する名前と値のペアを設定することができます。

システムプロパティーを使用して JBoss EAP 設定のデフォルトの値を上書きできます。たとえば、以下のパブリックインターフェイスバインドアドレスの XML 設定は、**jboss.bind.address** システムプロパティーによる設定が可能で、このシステムプロパティーが提供されないとデフォルトで **127.0.0.1** が使用されることを表しています。

```
<inet-address value="${jboss.bind.address:127.0.0.1}"/>
```

JBoss EAP でシステムプロパティーを設定する方法は複数あり、以下の方法が含まれます。

- JBoss EAP 起動スクリプトにシステムプロパティーを渡す
- 管理 CLI の使用
- 管理コンソールの使用
- **JAVA_OPTS** 環境変数の使用

JBoss EAP のマネージドドメインを使用する場合、ドメイン全体、特定のサーバーグループ、特定のホストとそのすべてのサーバーインスタンス、または 1 つのサーバーインスタンスにシステムプロパティーを適用できます。他の多くの JBoss EAP ドメイン設定と同様に、より具体的なレベルで設定されたシステムプロパティーはより抽象的なものを上書きします。詳細は ドメイン管理 の章を参照してください。

起動スクリプトにシステムプロパティーを渡す

JBoss EAP 起動スクリプトにシステムプロパティーを渡すには **-D** 引数を使用します。以下に例を示します。

```
$ EAP_HOME/bin/standalone.sh -Djboss.bind.address=192.168.1.2
```

このシステムプロパティーの設定方法は、JBoss EAP が起動する前に JBoss EAP のオプションを設定する必要がある場合に便利です。

管理 CLI を使用したシステムプロパティーの設定

管理 CLI で以下の構文を使用すると、システムプロパティーを設定できます。

```
/system-property=PROPERTY_NAME:add(value=PROPERTY_VALUE)
```

以下に例を示します。

```
/system-property=jboss.bind.address:add(value=192.168.1.2)
```

管理 CLI を使用してシステムプロパティーを設定する場合、一部の JBoss EAP オプション (上記の例の **jboss.bind.address** など) を有効にするにはサーバーの再起動が必要です。

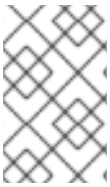
マネージドドメインの場合、上記の例はドメイン全体のシステムプロパティーを設定しますが、ドメイン設定のより具体的なレベルでシステムプロパティーを設定または上書きすることもできます。

管理コンソールを使用したシステムプロパティーの設定

- スタンドアロン JBoss EAP サーバーでは、管理コンソールの **Configuration** タブでシステムプロパティーを設定できます。**System Properties** を選択し、表示 ボタンをクリックします。
- マネージドドメインの場合:
  - ドメインレベルのシステムプロパティーは **Configuration** タブで設定できます。**System Properties** を選択し、表示 ボタンをクリックします。
  - サーバーグループおよびサーバーレベルのシステムプロパティーは **Runtime** タブで設定できます。設定するサーバーグループまたはサーバーを選択し、サーバーグループまたはサーバー名の横にある 表示 ボタンをクリックした後、**System Properties** タブを選択します。
  - ホストレベルのシステムプロパティーは **Runtime** タブで設定できます。設定するホストを選択し、ホスト名の横にあるドロップダウンメニューで **Properties** を選択します。

<ph x="1"/>
システムプロパティーは **JAVA_OPTS** 環境変数を使用して設定することもできます。**JAVA_OPTS** を編集する方法は多くありますが、JBoss EAP では JBoss EAP のプロセスで使用される **JAVA_OPTS** を設定する設定ファイルが提供されます。

スタンドアロンサーバーの場合、このファイルは **EAP_HOME/bin/standalone.conf** になります。マネージドドメインの場合は、**EAP_HOME/bin/domain.conf** になります。Microsoft Windows システムではこれらのファイルに **.bat** 拡張子が付きます。



注記

RPM インストールの場合、RPM サービス設定ファイル で **JAVA_OPTS** を編集してシステムプロパティーを設定することが推奨されます。RPM サービスプロパティーの設定 を参照してください。

適切な設定ファイルで **JAVA_OPTS** にシステムプロパティー定義を追加します。以下は、Red Hat Enterprise Linux システムでバインドアドレスを設定する例になります。

- **standalone.conf** では、**JAVA_OPTS** システムプロパティー定義をファイルの最後に追加します。以下に例を示します。

```
...
# Set the bind address
JAVA_OPTS="$JAVA_OPTS -Djboss.bind.address=192.168.1.2"
```

- **domain.conf** では、プロセスコントローラーの **JAVA_OPTS** 設定の前に **JAVA_OPTS** を設定する必要があります。以下に例を示します。

```
...
# Set the bind address
JAVA_OPTS="$JAVA_OPTS -Djboss.bind.address=192.168.1.2"

# The ProcessController process uses its own set of java options
if [ "x$PROCESS_CONTROLLER_JAVA_OPTS" = "x" ]; then
...
```

-

MODULE_OPTS 環境変数を使用した Java エージェントの追加
MODULE_OPTS=-javaagent:my-agent.jar 環境変数を使用して、起動スクリプトを編集せずに Java エージェントを JBoss Modules に直接追加できます。これにより、ログの設定後にエージェントが初期化されます。以前は、ブートクラスパスでログマネージャーが必要でした。

スタンドアロンサーバーでは、以下のファイルに **MODULE_OPTS** 環境変数を設定できます。

- RHEL では、起動スクリプトは **EAP_HOME/bin/standalone.conf** ファイルを使用します。
- Windows サーバーで、コマンドプロンプトで **EAP_HOME\bin\standalone.bat** ファイルを使用します。
- Windows サーバーで、PowerShell で **EAP_HOME\bin\standalone.ps1** ファイルを使用します。

ドメインのサーバーでは、**module-options** 属性をホスト JVM 設定またはサーバー JVM 設定に追加できます。

3.10. 管理監査ロギング

管理コンソール、管理 CLI、または管理 API を使用するカスタムアプリケーションを使用して実行されたすべての操作をログに記録する、管理インターフェイスの監査ロギングを有効にできます。監査ログエントリーは JSON 形式で保存されます。監査ロギングはデフォルトでは無効になっています。

監査ロギングを設定して ファイル または syslog サーバー へ出力できます。

注記

JBoss EAP には認証されたセッションがないため、ログインおよびログアウトイベントは監査できません。その代わりに、ユーザーから操作を受信すると監査メッセージがログに記録されます。

スタンドアロンサーバーの監査ロギング
監査ロギングはデフォルトでは無効になっていますが、デフォルトの監査ロギング設定はファイルに書き込みします。

```
<audit-log>
    <formatters>
        <json-formatter name="json-formatter"/>
    </formatters>
    <handlers>
        <file-handler name="file" formatter="json-formatter" path="audit-log.log" relative-
to="jboss.server.data.dir"/>
    </handlers>
    <logger log-boot="true" log-read-only="false" enabled="false">
        <handlers>
            <handler name="file"/>
        </handlers>
    </logger>
</audit-log>
```

この設定は、以下の管理 CLI コマンドを使用して読み取ることができます。

```
/core-service=management/access=audit:read-resource(recursive=true)
```

スタンドアロンサーバーの監査ロギングを有効にする場合は、監査ロギングの有効化 を参照してください。

マネージドドメインの監査ロギング
監査ロギングはデフォルトでは無効になっていますが、デフォルトの監査ロギング設定は各ホストおよび各サーバーに対してファイルを書き込みします。

```
<audit-log>
    <formatters>
        <json-formatter name="json-formatter"/>
    </formatters>
    <handlers>
        <file-handler name="host-file" formatter="json-formatter" relative-to="jboss.domain.data.dir"
path="audit-log.log"/>
        <file-handler name="server-file" formatter="json-formatter" relative-to="jboss.server.data.dir"
path="audit-log.log"/>
    </handlers>
    <logger log-boot="true" log-read-only="false" enabled="false">
        <handlers>
            <handler name="host-file"/>
        </handlers>
    </logger>
    <server-logger log-boot="true" log-read-only="false" enabled="false">
        <handlers>
            <handler name="server-file"/>
        </handlers>
    </server-logger>
</audit-log>
```

この設定は、以下の管理 CLI コマンドを使用して読み取ることができます。

```
/host=HOST_NAME/core-service=management/access=audit:read-resource(recursive=true)
```

マネージドドメインの監査ロギングを有効にする場合は、監査ロギングの有効化 を参照してください。

3.10.1. 管理監査ロギングの有効化

監査ロギングはデフォルトでは無効になっていますが、JBoss EAP には監査ロギングのファイルハンドラーが事前設定されています。監査ロギングを有効にする管理 CLI コマンドは、スタンドアロンサーバーとして実行しているかまたはマネージドドメインで実行しているかによって異なります。ファイルハンドラーの属性は 管理監査ロギング属性 を参照してください。

次の手順では、**NATIVE** および **HTTP** 監査ロギングを有効にします。Jakarta Management 監査ロギングを設定する場合は Jakarta Management 監査ロギングの有効化 を参照してください。

syslog 監査ロギングを設定する場合は syslog サーバーへの管理監査ロギングの送信 を参照してください。

スタンドアロンサーバー監査ロギングの有効化
監査ロギングは以下のコマンドを使用して有効にできます。

```
/core-service=management/access=audit/logger=audit-log:write-attribute(name=enabled,value=true)
```

デフォルトでは、このコマンドによって監査ログが **EAP_HOME/standalone/data/audit-log.log** に書き込まれます。

マネージドドメイン監査ロギングの有効化
マネージドドメインのデフォルトの監査ログ設定は、各ホストおよび各サーバーに対して監査ログを書き込むよう事前設定されています。

各ホストの監査ロギングは以下のコマンドを使用して有効にできます。

```
/host=HOST_NAME/core-service=management/access=audit/logger=audit-log:write-attribute(name=enabled,value=true)
```

デフォルトでは、このコマンドによって監査ログが **EAP_HOME/domain/data/audit-log.log** に書き込まれます。

各サーバーの監査ロギングは以下のコマンドを使用して有効にできます。

```
/host=HOST_NAME/core-service=management/access=audit/server-logger=audit-log:write-attribute(name=enabled,value=true)
```

デフォルトでは、このコマンドによって監査ログが **EAP_HOME/domain/servers/SERVER_NAME/data/audit-log.log** に書き込まれます。

3.10.2. Jakarta Management 監査ロギングの有効化

JBoss EAP には、Jakarta Management 監査ロギングのファイルハンドラーが事前設定されていますが、これらのログはデフォルトで無効になっています。監査ロギングを有効にする管理 CLI コマンドは、スタンドアロンサーバーまたは管理対象ドメインとして実行しているかによって異なります。

NATIVE または **HTTP** 監査ロギングを設定する場合は 管理監査ロギングの有効化 を参照してください。

スタンドアロンサーバーの **Jakarta Management** 監査ロギングの有効化
以下のコマンドを使用すると、スタンドアロンサーバーでの Jakarta Management 監査ロギングを有効にできます。

```
/subsystem=jmx/configuration=audit-log:add()
/subsystem=jmx/configuration=audit-log/handler=file:add()
```

これにより、Jakarta Management 監査ロギングが有効になり、定義された **file** ハンドラーを使用してこれらのログを **EAP_HOME/standalone/data/audit-log.log** に書き込むことができるようになります。

マネージドドメインの **Jakarta Management** 監査ロギングの有効化
マネージドドメインの各ホストおよびプロファイルに Jakarta Management 監査ロギングを有効にすることができます。

ホストの **Jakarta Management** 監査ロギングの有効化

1. ホストの **jmx** サブシステムで監査ロギングを有効にします。

```
/host=HOST_NAME/subsystem=jmx/configuration=audit-log:add()
```

2. **jmx** サブシステムの監査ロギングが有効になったら、以下のコマンドでホストにハンドラーを定義することができます。

```
/host=HOST_NAME/subsystem=jmx/configuration=audit-log/handler=host-file:add()
```

デフォルトでは、このコマンドによって Jakarta Management 監査ログが **EAP_HOME/domain/data/audit-log.log** に書き込まれます。

プロファイルの Jakarta Management 監査ロギングの有効化

1. プロファイルの **jmx** サブシステムで監査ロギングを有効にします。

```
/profile=PROFILE_NAME/subsystem=jmx/configuration=audit-log:add()
```

2. **jmx** サブシステムの監査ロギングが有効になったら、以下のコマンドでプロファイルにハンドラーを定義することができます。

```
/profile=PROFILE_NAME/subsystem=jmx/configuration=audit-log/handler=server-file:add()
```

デフォルトでは、このコマンドによって Jakarta Management 監査ログが **EAP_HOME/domain/servers/SERVER_NAME/data/audit-log.log** に書き込まれます。

3.10.3. syslog サーバーへの管理監査ロギングの送信

syslog ハンドラーは、監査ログエントリーが syslog サーバーに送信されるときのパラメーター (syslog サーバーのホスト名および syslog サーバーがリッスンするポート) を指定します。監査ロギングを syslog サーバーへ送信すると、ローカルファイルまたはローカル syslog サーバーへロギングする場合よりも、セキュリティーオプションが多くなります。複数の syslog ハンドラーを同時に定義およびアクティブ化することができます。

デフォルトでは、監査ロギングが有効である場合にファイルへ出力するよう事前設定されています。以下の手順に従って、syslog サーバーへの監査ロギングを設定および有効化します。syslog ハンドラーの属性については 管理監査ロギング属性 を参照してください。

1. syslog ハンドラーを追加します。
   syslog サーバーのホストとポートを指定して syslog ハンドラーを作成します。マネージドドメインでは、**/core-service** コマンドの前に **/host=HOST_NAME** を追加する必要があります。

```
batch
/core-service=management/access=audit/syslog-handler=SYSLOG_HANDLER_NAME:add(formatter=json-formatter)
/core-service=management/access=audit/syslog-handler=SYSLOG_HANDLER_NAME/protocol=udp:add(host=HOST_NAME,port=PORT)
run-batch
```

注記

渡すパラメーターは指定されたプロトコルによって異なります。

TLS を使用して syslog サーバーとセキュアに通信するようハンドラーを設定するには、認証を設定する必要もあります。以下に例を示します。

```
/core-service=management/access=audit/syslog-handler=SYSLOG_HANDLER_NAME/protocol=tls/authentication=truststore:add(keystore-path=PATH_TO_TRUSTSTORE,keystore-password=TRUSTSTORE_PASSWORD)
```

2. syslog ハンドラーへの参照を追加します。
マネージドドメインでは、このコマンドの前に **/host=HOST_NAME** を追加する必要があります。

```
/core-service=management/access=audit/logger=audit-log/handler=SYSLOG_HANDLER_NAME:add
```

3. 監査ロギングを有効にします。
管理監査ロギングの有効化 を参照して監査ロギングを有効にします。

重要

オペレーティングシステムでロギングが有効になっていないと、JBoss EAP で syslog サーバーへの監査ロギングを有効にしても動作しません。

Red Hat Enterprise Linux の **rsyslog** 設定の詳細は、**https://access.redhat.com/documentation/ja/red-hat-enterprise-linux/** にてシステム管理者のガイドの Rsyslog の基本設定 の項を参照してください。

3.10.4. 監査ログエントリーの読み取り

ファイルに出力される監査ログエントリーは、テキストビューアーで参照することを推奨します。syslog サーバーに出力される監査ログエントリーは syslog ビューアーアプリケーションで参照することを推奨します

注記

ログファイルの参照にテキストエディターを使用することは推奨されません。 これは、追加のログエントリーがログファイルに書き込まれなくなることがあるためです。

監査ログエントリーは JSON 形式で保存されます。各ログエントリーは最初にオプションのタイムスタンプ、次に以下の表のフィールドが示されます。

表3.6 管理監査ログフィールド

フィールド名	説明

フィールド名	説明
access	以下のいずれかの値を使用できます。 • **NATIVE** - 操作がネイティブ管理インターフェイスから実行されます。 • **HTTP** - 操作がドメイン HTTP インターフェイスから実行されます。 • **JMX** - 操作が **jmx** サブシステムから実行されます。
booting	起動プロセス中に操作が実行された場合は、値が **true** になります。 サーバーの起動後に操作が実行された場合は **false** になります。
domainUUID	ドメインコントローラーからサーバー、スレーブホストコントローラー、およびスレーブホストコントローラーサーバーへ伝播されるすべての操作をリンクする ID。
ops	実行される操作。JSON へシリアライズ化された操作のリストになります。起動時は、XML の解析で生じたすべての操作がリストされます。起動後は、通常 1 つのエントリーがリストされます。
r/o	操作によって管理モデルが変更されない場合は、値が **true** になります。 変更される場合は **false** になります。
remote-address	この操作を実行するクライアントのアドレス。
success	操作に成功した場合は値が **true** になります。 ロールバックされた場合は **false** になります。
type	管理操作であることを示す **core**、または **jmx** サブシステムからであることを示す **jmx** を値として指定できます。
user	認証されたユーザーのユーザー名。稼働しているサーバーと同じマシンの管理 CLI を使用して操作を行った場合は、特別な **$local** ユーザーが使用されます。
version	JBoss EAP インスタンスのバージョン番号。

3.11. サーバーライフサイクルイベントの通知

JBoss EAP **core-management** サブシステム または Jakarta Management を使用してサーバーライフサイクルイベントの通知を設定できます。サーバーランタイム設定の状態やサーバー稼働の状態が変更されると、通知が発生します。

JBoss EAP のサーバーランタイム設定の状態には、**STARTING**、 **RUNNING**、**RELOAD_REQUIRED**、**RESTART_REQUIRED**、**STOPPING**、および **STOPPED** があります。

JBoss EAP のサーバー稼働の状態には、**STARTING**、**NORMAL**、**ADMIN_ONLY**、**PRE_SUSPEND**、**SUSPENDING**、**SUSPENDED**、**STOPPING**、および **STOPPED** があります。

3.11.1. core-management サブシステムを使用したサーバーライフサイクルイベントの監視

リスナーを JBoss EAP **core-management** サブシステムに登録すると、サーバーのライフサイクルイベントを監視できます。以下の手順は、イベントをログファイルの記録するサンプルリスナーを作成および登録する方法を示しています。

1. リスナーを作成します。
   以下の例のように、**org.wildfly.extension.core.management.client.ProcessStateListener** の実装を作成します。

   例: リスナークラス

```
package org.simple.lifecycle.events.listener;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import org.wildfly.extension.core.management.client.ProcessStateListener;
import org.wildfly.extension.core.management.client.ProcessStateListenerInitParameters;
import org.wildfly.extension.core.management.client.RunningStateChangeEvent;
import
org.wildfly.extension.core.management.client.RuntimeConfigurationStateChangeEvent;

public class SimpleListener implements ProcessStateListener {

    private File file;
    private FileWriter fileWriter;
    private ProcessStateListenerInitParameters parameters;

    public void init(ProcessStateListenerInitParameters parameters) {
        this.parameters = parameters;
        this.file = new File(parameters.getInitProperties().get("file"));
        try {
            fileWriter = new FileWriter(file, true);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void cleanup() {
        try {
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            fileWriter = null;
        }
    }

    public void runtimeConfigurationStateChanged(RuntimeConfigurationStateChangeEvent
evt) {
        try {
            fileWriter.write(String.format("Runtime configuration state change for %s: %s to
%s\n", parameters.getProcessType(), evt.getOldState(), evt.getNewState()));
```

```
            fileWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void runningStateChanged(RunningStateChangeEvent evt) {
        try {
            fileWriter.write(String.format("Running state change for %s: %s to %s\n",
parameters.getProcessType(), evt.getOldState(), evt.getNewState()));
            fileWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

注記

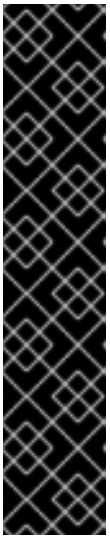
リスナーの実装時には以下に気をつけてください。

- サーバーがリロードすると、サーバーの停止時にリスナーはリッスンを停止し、サーバーの起動時にリスナーはリロードされます。そのため、実装では同じ JVM 内部でリスナーを適切に複数回ロード、初期化、および削除できるようにする必要があります。
- サーバー状態の変更に対応できるようにするため、リスナーへの通知はブロッキング状態になります。実装では、リスナーがブロックまたはデッドロックしないようにする必要があります。
- 各リスナーインスタンスは独自のスレッドで実行され、順番は保証されません。

2. クラスおよびパッケージを JAR にコンパイルします。
コンパイルするには、**org.wildfly.core:wildfly-core-management-client** Maven モジュールに依存する必要があります。

3. JAR を JBoss EAP モジュールとして追加します。
以下の管理 CLI コマンドを使用し、モジュール名と JAR へのパスを提供します。

```
module add --name=org.simple.lifecycle.events.listener --
dependencies=org.wildfly.extension.core-management-client --resources=/path/to/simple-
listener-0.0.1-SNAPSHOT.jar
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

4. リスナーを登録します。
   以下のかんり CLI コマンドを使用してリスナーを **core-management** サブシステムに追加します。クラス、モジュール、およびサーバーライフサイクルイベントを記録するログファイルの場所を指定します。

```
/subsystem=core-management/process-state-listener=my-simple-
listener:add(class=org.simple.lifecycle.events.listener.SimpleListener,
module=org.simple.lifecycle.events.listener,properties={file=/path/to/my-listener-output.txt})
```

上記の **SimpleListener** クラスを基にしてサーバーライフサイクルのイベントが **my-listener-output.txt** ファイルに記録されるようになりました。たとえば、管理 CLI で **:suspend** コマンドを実行すると、以下が **my-listener-output.txt** ファイルに出力されます。

```
Running state change for STANDALONE_SERVER: normal to suspending
Running state change for STANDALONE_SERVER: suspending to suspended
```

これを見ると、稼働状態が **normal** から **suspending** に変わった後、**suspending** から **suspended** に変わったことがわかります。

3.11.2. Jakarta Management 通知を使用したサーバーライフサイクルイベントの監視

Jakarta Management 通知リスナーを登録すると、サーバーのライフサイクルイベントを監視できます。以下の手順は、イベントをログファイルの記録するサンプルリスナーを作成および追加する方法を示しています。

1. リスナーを作成します。
   以下の例のように、**javax.management.NotificationListener** の実装を作成します。

   例: リスナークラス

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
```

```
import javax.management.AttributeChangeNotification;
import javax.management.Notification;
import javax.management.NotificationListener;

import org.jboss.logging.Logger;

public class StateNotificationListener implements NotificationListener {

  public static final String RUNTIME_CONFIGURATION_FILENAME = "runtime-
configuration-notifications.txt";
  public static final String RUNNING_FILENAME = "running-notifications.txt";
  private final Path targetFile;

  public StateNotificationListener() {
    this.targetFile = Paths.get("notifications/data").toAbsolutePath();
    init(targetFile);
  }

  protected Path getRuntimeConfigurationTargetFile() {
    return this.targetFile.resolve(RUNTIME_CONFIGURATION_FILENAME);
  }

  protected Path getRunningConfigurationTargetFile() {
    return this.targetFile.resolve(RUNNING_FILENAME);
  }

  protected final void init(Path targetFile) {
    try {
      Files.createDirectories(targetFile);

      if (!Files.exists(targetFile.resolve(RUNTIME_CONFIGURATION_FILENAME))) {
        Files.createFile(targetFile.resolve(RUNTIME_CONFIGURATION_FILENAME));
      }

      if (!Files.exists(targetFile.resolve(RUNNING_FILENAME))) {
        Files.createFile(targetFile.resolve(RUNNING_FILENAME));
      }
    } catch (IOException ex) {
        Logger.getLogger(StateNotificationListener.class).error("Problem handling JMX
Notification", ex);
    }
  }

  @Override
  public void handleNotification(Notification notification, Object handback) {
    AttributeChangeNotification attributeChangeNotification = (AttributeChangeNotification)
notification;
    if ("RuntimeConfigurationState".equals(attributeChangeNotification.getAttributeName())) {
      writeNotification(attributeChangeNotification, getRuntimeConfigurationTargetFile());
    } else {
      writeNotification(attributeChangeNotification, getRunningConfigurationTargetFile());
    }
  }

  private void writeNotification(AttributeChangeNotification notification, Path path) {
    try (BufferedWriter in = Files.newBufferedWriter(path, StandardCharsets.UTF_8,
```

```
StandardOpenOption.APPEND)) {
      in.write(String.format("%s %s %s %s", notification.getType(),
notification.getSequenceNumber(), notification.getSource().toString(),
notification.getMessage()));
      in.newLine();
      in.flush();
    } catch (IOException ex) {
      Logger.getLogger(StateNotificationListener.class).error("Problem handling JMX
Notification", ex);
    }
  }
}
```

2. 通知リスナーを登録します。
   通知リスナーを **MBeanServer** に追加します。

   例: 通知リスナーの追加

```
MBeanServer server = ManagementFactory.getPlatformMBeanServer();
server.addNotificationListener(ObjectName.getInstance("jboss.root:type=state"), new
StateNotificationListener(), null, null);
```

3. JBoss EAP にパッケージ化およびデプロイします。

上記の **StateNotificationListener** クラスを基にしてサーバーライフサイクルイベントがファイルに記録されるようになりました。たとえば、管理 CLI で **:suspend** コマンドを実行すると、以下が **running-notifications.txt** ファイルに出力されます。

```
jmx.attribute.change 5 jboss.root:type=state The attribute 'RunningState' has changed from 'normal'
to 'suspending'
jmx.attribute.change 6 jboss.root:type=state The attribute 'RunningState' has changed from
'suspending' to 'suspended'
```

これを見ると、稼働状態が **normal** から **suspending** に変わった後、**suspending** から **suspended** に変わったことがわかります。

第4章 ネットワークおよびポート設定

4.1. インターフェイス

JBoss EAP は設定全体で名前付きインターフェイスを参照します。これにより、使用ごとにインターフェイスの完全な詳細を必要とせず、論理名を使用して個々のインターフェイス宣言を参照できます。

また、複数のマシンでネットワークインターフェイスの詳細が異なる場合にマネージドドメインの設定が容易になります。各サーバーインスタンスは、論理名グループに対応できます。

standalone.xml、**domain.xml**、および **host.xml** ファイルにはインターフェイス宣言が含まれます。使用されるデフォルトの設定に応じて、複数の事前設定されたインターフェイス名があります。**management** インターフェイスは、HTTP 管理エンドポイントを含む、管理レイヤーが必要なすべてのコンポーネントおよびサービスに使用できます。**public** インターフェイスは、アプリケーション関連のネットワーク通信すべてに使用できます。**unsecure** インターフェイスは、標準設定の IIOP ソケットに使用されます。**private** インターフェイスは、標準設定の JGroups ソケットに使用されます。

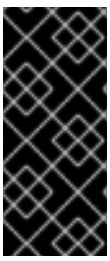
4.1.1. デフォルトインターフェイス設定

```
<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}"/>
  </interface>
  <interface name="private">
    <inet-address value="${jboss.bind.address.private:127.0.0.1}"/>
  </interface>
  <interface name="unsecure">
    <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}"/>
  </interface>
</interfaces>
```

デフォルトでは、JBoss EAP はこれらのインターフェイスを **127.0.0.1** にバインドしますが、適切なプロパティーを設定すると起動時に値を上書きできます。たとえば、以下のコマンドで JBoss EAP をスタンドアロンサーバーとして起動するときに **public** インターフェイスの **inet-address** を設定できます。

```
$ EAP_HOME/bin/standalone.sh -Djboss.bind.address=IP_ADDRESS
```

この代わりに、サーバー起動のコマンドラインで **-b** スイッチを使用することができます。サーバー起動オプションの詳細は、サーバーランタイム引数およびスイッチ を参照してください。



重要

JBoss EAP が使用するデフォルトのネットワークインターフェイスまたはポートを変更する場合は、変更したインターフェイスまたはポートを使用するスクリプトを変更する必要があることに注意してください。これには JBoss EAP サービススクリプトが含まれます。 また、管理コンソールまたは CLI にアクセスするときに適切なインターフェイスとポートを指定するようにしてください。

4.1.2. インターフェイスの設定

ネットワークインターフェイスは、物理インターフェイスの論理名および選択基準を指定して宣言されます。選択基準はワイルドカードアドレスを参照したり、一致が有効となるためにインターフェイスまたはアドレスで必要となる1つ以上の特徴のセットを指定したりできます。使用できるすべてのインターフェイス選択基準は インターフェイス属性 を参照してください。

インターフェイスは管理コンソールまたは管理 CLI を使用して設定できます。以下にインターフェイスの追加および更新の例をいくつか示します。最初に管理 CLI コマンドを示し、その後に対応する設定 XML を示します。

NIC 値があるインターフェイスの追加
NIC 値が **eth0** であるインターフェイスを新たに追加します。

```
/interface=external:add(nic=eth0)
```

```
<interface name="external">
  <nic name="eth0"/>
</interface>
```

複数の条件値があるインターフェイスの追加
稼働時に適切なサブネットのすべてのインターフェイスまたはアドレスと一致し、マルチキャストをサポートする、ポイントツーポイントでないインターフェイスを新たに追加します。

```
/interface=default:add(subnet-match=192.168.0.0/16,up=true,multicast=true,not={point-to-point=true})
```

```
<interface name="default">
  <subnet-match value="192.168.0.0/16"/>
  <up/>
  <multicast/>
  <not>
    <point-to-point/>
  </not>
</interface>
```

インターフェイス属性の更新
public インターフェイスのデフォルトの **inet-address** 値を更新し、**jboss.bind.address** プロパティーによってこの値が起動時に設定されるようにします。

```
/interface=public:write-attribute(name=inet-address,value="${jboss.bind.address:192.168.0.0}")
```

```
<interface name="public">
   <inet-address value="${jboss.bind.address:192.168.0.0}"/>
</interface>
```

マネージドドメインでインターフェイスをサーバーに追加

```
/host=HOST_NAME/server-config=SERVER_NAME/interface=INTERFACE_NAME:add(inet-address=127.0.0.1)
```

```
<servers>
  <server name="SERVER_NAME" group="main-server-group">
    <interfaces>
      <interface name="INTERFACE_NAME">
        <inet-address value="127.0.0.1"/>
```

```
        </interface>
      </interfaces>
    </server>
  </servers>
```

4.2. ソケットバインディング

ソケットバインディングとソケットバインディンググループを使用することにより、ネットワークポートと、JBoss EAP の設定で必要なネットワーキングインターフェイスとの関係を定義できます。ソケットバインディングはソケットの名前付き設定です。ソケットバインディンググループは、ある論理名でグループ化されたソケットバインディング宣言のコレクションです。

これにより、使用ごとにソケット設定の完全な詳細を必要とせずに、設定の他のセクションが論理名でソケットバインディングを参照できるようになります。

これらの名前付き設定の宣言は **standalone.xml** および **domain.xml** 設定ファイルにあります。スタンドアロンサーバーにはソケットバインディンググループが 1 つのみ含まれますが、マネージドドメインには複数のグループを含むことができます。マネージドドメインで各サーバーグループのソケットバインディンググループを作成するか、複数のサーバーグループ間でソケットバインディンググループを共有することができます。

デフォルトで JBoss EAP によって使用されるポートは、使用されるソケットバインディンググループと、個々のデプロイメントの要件に応じて異なります。

JBoss EAP 設定のソケットバインディンググループで定義できるソケットバインディングには 3 つの種類があります。

インバウンドソケットバインディング

socket-binding 要素は、JBoss EAP サーバーのインバウンドソケットバインディングを設定するために使用されます。デフォルトの JBoss EAP 設定には、HTTP や HTTPS トラフィック用などの、事前設定された **socket-binding** 要素が複数提供されます。JBoss EAP**Configuring Messaging**の Broadcast Groups には他の例も記載されています。
この要素の属性については、ソケットバインディング属性 を参照してください。

リモートアウトバウンドソケットバインディング

remote-destination-outbound-socket-binding 要素は、JBoss EAP サーバーのリモートとなる宛先のアウトバウンドソケットバインディングを設定するために使用されます。デフォルトの JBoss EAP 設定には、メールサーバーに使用できるリモート宛先のソケットバインディングの例が含まれています。JBoss EAP**Configuring Messaging**の Using the Integrated Artemis Resource Adapter for Remote Connections には、他の例も記載されています。
この要素の属性については、ソケットバインディング属性 の表を参照してください。

ローカルアウトバウンドソケットバインディング

local-destination-outbound-socket-binding 要素は、JBoss EAP サーバーのローカルとなる宛先のアウトバウンドソケットバインディングを設定するために使用されます。通常、このソケットバインディングはあまり使用されません。
この要素の属性については、ソケットバインディング属性 の表を参照してください。

4.2.1. 管理ポート

JBoss EAP 7 では、管理ポートが集約されました。JBoss EAP 7 は、管理 CLI によって使用されるネイティブ管理と、Web ベース管理コンソールによって使用される HTTP 管理の両方に **9990** ポートを使用

します。JBoss EAP 6 でネイティブ管理ポートとして使用されていた **9999** ポートは使用されなくなりましたが、必要な場合は有効にできます。

管理コンソールに対して HTTPS を有効にすると、デフォルトではポート **9993** が使用されます。

4.2.2. デフォルトのソケットバインディング

JBoss EAP には、事前設定された 5 つのプロファイル (**default**、**ha**、**full**、**full-ha**、**load-balancer**) のソケットバインディンググループが含まれています。

デフォルトのポートや説明などのデフォルトのソケットバインディングに関する詳細情報は、デフォルトのソケットバインディング を参照してください。

重要

JBoss EAP が使用するデフォルトのネットワークインターフェイスまたはポートを変更する場合は、変更したインターフェイスまたはポートを使用するスクリプトを変更する必要があることに注意してください。これには JBoss EAP サービススクリプトが含まれます。 また、管理コンソールまたは CLI にアクセスするときに適切なインターフェイスとポートを指定するようにしてください。

スタンドアロンサーバー
スタンドアロンサーバーとして実行されている場合、設定ファイルごとに 1 つのソケットバインディンググループのみが定義されます。各スタンドアロン設定ファイル (**standalone.xml**、**standalone-ha.xml**、**standalone-full.xml**、**standalone-full-ha.xml**、**standalone-load-balancer.xml**) は、対応するプロファイルによって使用される技術のソケットバインディングを定義します。

たとえば、デフォルトのスタンドアロン設定ファイル (**standalone.xml**) は以下のソケットバインディングを指定します。

```
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
    <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}"/>
    <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9993}"/>
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
    <socket-binding name="txn-recovery-environment" port="4712"/>
    <socket-binding name="txn-status-manager" port="4713"/>
    <outbound-socket-binding name="mail-smtp">
        <remote-destination host="localhost" port="25"/>
    </outbound-socket-binding>
</socket-binding-group>
```

マネージドドメイン
マネージドドメインで実行されている場合、すべてのソケットバインディンググループは **domain.xml** ファイルで定義されます。事前定義されたソケットバインディンググループは 5 つあります。

- **standard-sockets**
- **ha-sockets**
- **full-sockets**

- **full-ha-sockets**
- **load-balancer-sockets**

各ソケットバインディンググループは、対応するプロファイルによって使用される技術のソケットバインディングを指定します。たとえば、**full-ha-sockets** ソケットバインディンググループは、高可用性のために **full-ha** プロファイルによって使用される複数の **jgroups** ソケットバインディングを定義します。

```
<socket-binding-groups>
  <socket-binding-group name="standard-sockets" default-interface="public">
    <!-- Needed for server groups using the 'default' profile  -->
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
    <socket-binding name="txn-recovery-environment" port="4712"/>
    <socket-binding name="txn-status-manager" port="4713"/>
    <outbound-socket-binding name="mail-smtp">
      <remote-destination host="localhost" port="25"/>
    </outbound-socket-binding>
  </socket-binding-group>
  <socket-binding-group name="ha-sockets" default-interface="public">
    <!-- Needed for server groups using the 'ha' profile  -->
    ...
  </socket-binding-group>
  <socket-binding-group name="full-sockets" default-interface="public">
    <!-- Needed for server groups using the 'full' profile  -->
    ...
  </socket-binding-group>
  <socket-binding-group name="full-ha-sockets" default-interface="public">
    <!-- Needed for server groups using the 'full-ha' profile  -->
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
    <socket-binding name="iiop" interface="unsecure" port="3528"/>
    <socket-binding name="iiop-ssl" interface="unsecure" port="3529"/>
    <socket-binding name="jgroups-mping" interface="private" port="0" multicast-
address="${jboss.default.multicast.address:230.0.0.4}" multicast-port="45700"/>
    <socket-binding name="jgroups-tcp" interface="private" port="7600"/>
    <socket-binding name="jgroups-udp" interface="private" port="55200" multicast-
address="${jboss.default.multicast.address:230.0.0.4}" multicast-port="45688"/>
    <socket-binding name="modcluster" port="0" multicast-address="224.0.1.105" multicast-
port="23364"/>
    <socket-binding name="txn-recovery-environment" port="4712"/>
    <socket-binding name="txn-status-manager" port="4713"/>
    <outbound-socket-binding name="mail-smtp">
      <remote-destination host="localhost" port="25"/>
    </outbound-socket-binding>
  </socket-binding-group>
  <socket-binding-group name="load-balancer-sockets" default-interface="public">
    <!-- Needed for server groups using the 'load-balancer' profile  -->
    ...
  </socket-binding-group>
</socket-binding-groups>
```

注記

管理インターフェイスのソケット設定は、ドメインコントローラーの **host.xml** ファイルに定義されます。

4.2.3. ソケットバインディングの設定

ソケットバインディングを設定するとき、**port** および **interface** 属性や、**multicast-address** および **multicast-port** などのマルチキャスト設定を設定できます。使用できるソケットバインディング属性すべての詳細は、ソケットバインディングの属性 を参照してください。

ソケットバインディングは管理コンソールまたは管理 CLI を使用して設定できます。以下の手順では、ソケットバインディンググループの追加、ソケットバインディングの追加、および管理 CLI を使用したソケットバインディングの設定を行います。

1. 新しいソケットバインディンググループを追加します。スタンドアロンサーバーとして実行している場合は追加できないことに注意してください。

```
/socket-binding-group=new-sockets:add(default-interface=public)
```

2. ソケットバインディングを追加します。

```
/socket-binding-group=new-sockets/socket-binding=new-socket-binding:add(port=1234)
```

3. ソケットバインディンググループによって設定されるデフォルト以外のインターフェイスを使用するよう、ソケットバインディングを変更します。

```
/socket-binding-group=new-sockets/socket-binding=new-socket-binding:write-
attribute(name=interface,value=unsecure)
```

以下の例は、上記の手順の完了後に XML 設定がどのようになるかを示しています。

```
<socket-binding-groups>
    ...
    <socket-binding-group name="new-sockets" default-interface="public">
        <socket-binding name="new-socket-binding" interface="unsecure" port="1234"/>
    </socket-binding-group>
</socket-binding-groups>
```

4.2.4. サーバーのソケットバインディングおよび解放ポートの表示

管理コンソールからソケットバインディング名とサーバーの解放ポートを表示できます。サーバーの状態が以下になると、情報が表示されます。

- **running**
- **reload-required**
- **restart-required**

サーバーのソケットバインディングと解放ポートを表示するには、以下を実行します。

1. 管理コンソールにアクセスし、**Runtime** に移動します。

2. サーバーをクリックすると、ソケットバインディング名と解放ポートが右側のペインに表示されます。

4.2.5. ポートオフセット

ポートオフセットとは、該当するサーバーのソケットバインディンググループに指定されたすべてのポート値に追加される数値のオフセットのことです。これにより、同じホストの別のサーバーとの競合を防ぐため、サーバーはソケットバインディンググループに定義されたポート値とオフセットを継承できるようになります。たとえば、ソケットバインディンググループの HTTP ポートが **8080** で、サーバーが **100** をポートオフセットとして使用する場合、HTTP ポートは **8180** になります。

管理 CLI を使用してマネージドドメインのサーバーにポートオフセットとして **250** を設定する例を以下に示します。

```
/host=master/server-config=server-two/:write-attribute(name=socket-binding-port-offset,value=250)
```

ポートオフセットは、マネージドドメインのサーバーと、同じホストで複数のスタンドアロンサーバーを実行する場合に使用できます。

jboss.socket.binding.port-offset プロパティーを使用してスタンドアロンサーバーを起動するときにポートオフセットを渡すことができます。

```
$ EAP_HOME/bin/standalone.sh -Djboss.socket.binding.port-offset=100
```

4.3. IPV6 アドレス

デフォルトでは、JBoss EAP は IPv4 アドレスを使用して実行するように設定されます。以下の手順では、IPv6 アドレスを使用して実行するよう JBoss EAP を設定する方法を示します。

IPv6 アドレスの JVM スタックの設定
IPv6 アドレスを優先するように、起動設定を更新します。

1. 起動設定ファイルを開きます。
   - スタンドアロンサーバーとして実行している場合は、**EAP_HOME/bin/standalone.conf** ファイル (Windows Server の場合は **standalone.conf.bat**) を編集します。
   - マネージドドメインで実行している場合は、**EAP_HOME/bin/domain.conf** ファイル (Windows Server の場合は **domain.conf.bat**) を編集します。
2. **java.net.preferIPv4Stack** プロパティーを **false** に設定します。

   ```
   -Djava.net.preferIPv4Stack=false
   ```

3. **java.net.preferIPv6Addresses** プロパティーを追加し、**true** に設定します。

   ```
   -Djava.net.preferIPv6Addresses=true
   ```

以下の例は、上記の変更を行った後に起動設定ファイルの JVM オプションがどのようになるかを示しています。

```
# Specify options to pass to the Java VM.
#
if [ "x$JAVA_OPTS" = "x" ]; then
```

```
    JAVA_OPTS="-Xms1303m -Xmx1303m -Djava.net.preferIPv4Stack=false"
    JAVA_OPTS="$JAVA_OPTS -
Djboss.modules.system.pkgs=$JBOSS_MODULES_SYSTEM_PKGS -Djava.awt.headless=true"
    JAVA_OPTS="$JAVA_OPTS -Djava.net.preferIPv6Addresses=true"
else
```

IPv6 アドレスのインターフェイス宣言の更新

設定のデフォルトのインターフェイス値は、IPv6 アドレスに変更できます。たとえば、以下の管理 CLI コマンドは **management** インターフェイスを IPv6 ループバックアドレス (**::1**) に設定します。

```
/interface=management:write-attribute(name=inet-
address,value="${jboss.bind.address.management:[::1]}")
```

以下の例は、上記のコマンド実行後に XML 設定がどのようになるかを示しています。

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:[::1]}"/>
    </interface>
    ....
</interfaces>
```

第5章 JBOSS EAP のセキュリティー

JBoss EAP は、独自のインターフェイスおよびサービスのセキュリティーを設定できる機能と、JBoss EAP で実行されるアプリケーションのセキュリティーを提供します。

- 一般的なセキュリティー概念と JBoss EAP 固有のセキュリティー概念については、**Security Architecture** を参照してください。
- JBoss EAP 自体のセキュア化に関する情報は、**How to Configure Server Security** を参照してください。
- JBoss EAP にデプロイされたアプリケーションのセキュリティーに関する詳細は、アイデンティティー管理の設定方法 を参照してください。
- Kerberos を使用した JBoss EAP のシングルサインオンの設定に関する情報は、**How to Set Up SSO with Kerberos** を参照してください。
- SAML v2 を使用して JBoss EAP のシングルサインオンを設定するための情報は **How To Set Up SSO with SAML v2** を参照してください。

第6章 JBOSS EAP クラスローディング

JBoss EAP は、デプロイされたアプリケーションのクラスパスを制御するためにモジュール形式のクラスローディングシステムを使用します。このシステムは、階層クラスローダーの従来のシステムよりも、柔軟性があり、より詳細に制御できます。開発者は、アプリケーションで利用可能なクラスに対して粒度の細かい制御を行い、アプリケーションサーバーで提供されるクラスを無視して独自のクラスを使用するようデプロイメントを設定できます。

モジュール形式のクラスローダーにより、すべての Java クラスはモジュールと呼ばれる論理グループに分けられます。各モジュールは、独自のクラスパスに追加されたモジュールからクラスを取得するために、他のモジュールの依存関係を定義できます。デプロイされた各 JAR および WAR ファイルはモジュールとして扱われるため、開発者はモジュール設定をアプリケーションに追加してアプリケーションのクラスパスの内容を制御できます。

6.1. モジュール

モジュールは、クラスローディングおよび依存関係管理に使用されるクラスの論理グループです。JBoss EAP は、静的モジュールと動的モジュールの 2 つの種類のモジュールを識別します。この 2 つの種類のモジュールの主な違いは、パッケージ化方法です。

静的モジュール
静的モジュールは、アプリケーションサーバーの **EAP_HOME/modules/** ディレクトリーで定義されます。各モジュールは **EAP_HOME/modules/com/mysql/** のようにサブディレクトリーとして存在します。各モジュールには、**module.xml** 設定ファイルとすべての必要な JAR ファイルが含まれるスロットサブディレクトリー (デフォルトでは **main**) が含まれます。アプリケーションサーバーにより提供される API は、Jakarta EE API と他の API を含む静的モジュールとして提供されます。

例: MySQL Java Database Connectivity (JDBC) ドライバーの module.xml ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-8.0.12.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

モジュール名 **com.mysql** は、スロット名 **main** を除くモジュールのディレクトリー構造と一致する必要があります。

カスタム静的モジュールの作成は、同じサードパーティーライブラリーを使用する同じサーバー上に多くのアプリケーションがデプロイされる場合に役立ちます。これらのライブラリーを各アプリケーションとバンドルする代わりに、管理者はこれらのライブラリーが含まれるモジュールを作成およびインストールできます。アプリケーションは、カスタム静的モジュールで明示的な依存関係を宣言できます。

JBoss EAP ディストリビューションで提供されるモジュールは、**EAP_HOME/modules** ディレクトリー内の **system** ディレクトリーにあります。このため、サードパーティーによって提供されるモジュールから分離されます。また、JBoss EAP 上で使用する、Red Hat により提供されるすべての製品によって、**system** ディレクトリー内にモジュールがインストールされます。

モジュールごとに 1 つのディレクトリーを使用して、カスタムモジュールが **EAP_HOME/modules** ディレクトリーにインストールされるようにする必要があります。こうすると、同梱されたバージョンではなく、**system** ディレクトリーに存在するカスタムバージョンのモジュールがロードされるようになります。これにより、ユーザー提供のモジュールがシステムモジュールよりも優先されます。

JBOSS_MODULEPATH 環境変数を使用して JBoss EAP がモジュールを検索する場所を変更する場合は、指定された場所の 1 つで **system** サブディレクトリー構造を探します。**system** 構造は、**JBOSS_MODULEPATH** で指定された場所のどこかに存在する必要があります。

注記

JBoss EAP 7.1 より、**module.xml** ファイルの **resource-root path** 要素で絶対パスを使用できるようになりました。これにより、リソースライブラリーを **EAP_HOME/modules/** ディレクトリーに移動しなくてもリソースライブラリーにアクセスできるようになりました。

例: module.xml ファイルの絶対パス

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="oracle.jdbc">
   <resources>
      <resource-root path="/home/redhat/test.jar"/>
   </resources>
</module>
```

動的モジュール
動的モジュールは、各 JAR または WAR デプロイメント (または、EAR 内のサブデプロイメント) に対してアプリケーションサーバーによって作成およびロードされます。動的モジュールの名前は、デプロイされたアーカイブの名前に由来します。デプロイメントはモジュールとしてロードされるため、依存関係を設定し、他のデプロイメントで依存関係として使用することが可能です。

モジュールは必要な場合にのみロードされます。通常、モジュールは、明示的または暗黙的な依存関係があるアプリケーションがデプロイされる場合にのみロードされます。

事前定義されたモジュール
JBoss EAP 7.2 より、デフォルトのモジュールローダーを使用する場合に事前設定されたモジュールのセットを使用できるようになりました。このモジュールはすべての JBoss Modules API が含まれる **org.jboss.modules** で、JBoss Modules によって提供され、常に利用可能です。Java 9 以上で提供される標準の Java Platform Module System (JPMS) モジュールも標準の名前で利用可能です。JDK 8 を使用する場合、JDK 9 モジュールは JBoss Module によってエミュレートされます。

Java 9 以上で使用できるプラットフォームモジュールのリストは、該当する JDK のドキュメントを参照してください。

Java 8 向けに提供されるプラットフォームモジュールのリストは、Java 8 向けに提供されるプラットフォームモジュール を参照してください。

6.2. モジュールの依存性

モジュール依存関係は、あるモジュールに他の 1 つまたは複数のモジュールのクラスが必要になるという宣言です。JBoss EAP がモジュールをロードするときに、モジュール形式のクラスローダーがモジュールの依存関係を解析し、各依存関係のクラスをクラスパスに追加します。指定の依存関係が見つからない場合、モジュールはロードできません。

注記

モジュールとモジュール形式のクラスローディングシステムに関する完全な詳細については、モジュール を参照してください。

JAR や WAR などのデプロイされたアプリケーションは動的モジュールとしてロードされ、依存関係を利用して JBoss EAP によって提供される API にアクセスします。

依存関係には明示的と暗黙的の 2 つのタイプがあります。

明示的な依存関係

明示的な依存関係は開発者が設定ファイルで宣言します。静的モジュールでは、依存関係を **module.xml** ファイルに宣言できます。動的モジュールでは、デプロイメントの **MANIFEST.MF** または **jboss-deployment-structure.xml** デプロイメント記述子に依存関係を宣言できます。

暗黙的な依存関係

暗黙的な依存関係は、デプロイメントで特定の状態やメタデータが見つかったときに自動的に追加されます。JBoss EAP に同梱される Jakarta EE API は、デプロイメントで暗黙的な依存関係が検出されたときに追加されるモジュールの例になります。
jboss-deployment-structure.xml デプロイメント記述子ファイルを使用して、特定の暗黙的な依存関係を除外するようデプロイメントを設定することも可能です。これは、JBoss EAP が暗黙的な依存関係として追加しようとする特定バージョンのライブラリーをアプリケーションがバンドルする場合に役に立つことがあります。

オプションの依存関係
明示的な依存関係は、オプションとして指定できます。オプションの依存関係をロードできなくても、モジュールのロードは失敗しません。ただし、依存関係は後で使用できるようになっても、モジュールのクラスパスには追加されません。依存関係はモジュールがロードされるときに利用可能である必要があります。

依存関係のエクスポート
モジュールのクラスパスには独自のクラスとその直接の依存関係のクラスのみが含まれます。モジュールは 1 つの依存関係の依存関係クラスにはアクセスできませんが、暗黙的な依存関係のエクスポートを指定できます。ただし、モジュールは、明示的な依存関係をエクスポートするように指定できます。エクスポートした依存関係は、エクスポートするモジュールに依存するモジュールに提供されます。

たとえば、モジュール A はモジュール B に依存し、モジュール B はモジュール C に依存します。モジュール A はモジュール B のクラスにアクセスでき、モジュール B はモジュール C のクラスにアクセスできます。モジュール Aは以下のいずれかの条件を満たさない限り、モジュール C のクラスにアクセスできません。

- モジュール A が、モジュール C に対する明示的な依存関係を宣言する
- モジュール B がモジュール C の依存関係をエクスポートする

グローバルモジュール
グローバルモジュールは、JBoss EAP が各アプリケーションへの依存関係として提供するモジュールです。このモジュールをグローバルモジュールの JBoss EAP のリストへ追加すると、モジュールをグローバルモジュールにすることができます。モジュールへの変更は必要ありません。

詳細は グローバルモジュールの定義 の項を参照してください。

6.3. カスタムモジュールの作成

カスタムの静的モジュールを追加して、JBoss EAP で実行しているデプロイメントがリソースを利用できるようにすることができます。モジュールは 手動 で作成するか、管理 CLI を使用 して作成することができます。

モジュールの作成後、アプリケーションがリソースを使用できるようにするには モジュールを依存関係として追加 する必要があります。

カスタムモジュールの手動作成

カスタムモジュールを手動で作成するには、以下の手順に従います。

1. **EAP_HOME/modules/** ディレクトリーに適切なディレクトリー構造を作成します。

   例: MySQL JDBC ドライバーディレクトリー構造の作成

   ```
   $ cd EAP_HOME/modules/
   $ mkdir -p com/mysql/main
   ```

2. JAR ファイルまたはその他必要なリソースを **main/** サブディレクトリーにコピーします。

   例: MySQL JDBC ドライバー JAR のコピー

   ```
   $ cp /path/to/mysql-connector-java-8.0.12.jar EAP_HOME/modules/com/mysql/main/
   ```

3. **module.xml** ファイルを **main/** サブディレクトリーに作成し、そのファイルの適切なリソースおよび依存関係を指定します。

   例: MySQL JDBC ドライバー module.xml ファイル

   ```
   <?xml version="1.0" ?>
   <module xmlns="urn:jboss:module:1.1" name="com.mysql">
     <resources>
       <resource-root path="mysql-connector-java-8.0.12.jar"/>
     </resources>
     <dependencies>
       <module name="javaee.api"/>
       <module name="sun.jdk"/>
       <module name="ibm.jdk"/>
       <module name="javax.api"/>
       <module name="javax.transaction.api"/>
     </dependencies>
   </module>
   ```

管理 CLI を使用したカスタムモジュールの作成

module add 管理 CLI コマンドを使用してカスタムモジュールを作成できます。

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

1. JBoss EAP サーバーを起動します。
2. 管理 CLI を起動します。

```
$ EAP_HOME/bin/jboss-cli.sh
```

3. **module add** 管理 CLI コマンドを使用して新しいコアモジュールを追加します。

```
module add --name=MODULE_NAME --resources=PATH_TO_RESOURCE --
dependencies=DEPENDENCIES
```

例: MySQL モジュールの作成

```
module add --name=com.mysql --resources=/path/to/mysql-connector-java-8.0.12.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```

独自の module.xml ファイルの提供、外部モジュールディレクトリーの使用、モジュールの代替スロットの指定など、このコマンドのカスタマイズに使用できる引数については、モジュールコマンド引数 を参照してください。また、**module --help** を実行すると、このコマンドを使用したモジュールの追加および削除に関する詳細を表示することもできます。

モジュールを依存関係として追加

アプリケーションがこのモジュールのリソースにアクセスできるようにするには、モジュールを依存関係として追加する必要があります。

- デプロイメント記述子を使用してアプリケーション固有の依存関係を追加するには、JBoss EAP **Development Guide**の Add an Explicit Module Dependency to a Deployment を参照してください。
- モジュールを依存関係としてすべてのアプリケーションに追加する手順については グローバルモジュールの定義 の項を参照してください。

たとえば、以下の手順は複数のプロパティーファイルが含まれる JAR ファイルをモジュールとして追加し、グローバルモジュールを定義して、アプリケーションがこれらのプロパティーをロードできるようにします。

1. JAR ファイルをコアモジュールとして追加します。

```
module add --name=myprops --resources=/path/to/properties.jar
```

2. すべてのデプロイメントが使用できるようにするため、このモジュールをグローバルモジュールとして定義します。

```
/subsystem=ee:list-add(name=global-modules,value={name=myprops})
```

3. アプリケーションは、JAR 内に含まれるプロパティーファイルの 1 つからプロパティーを読み出すことができます。

```
Thread.currentThread().getContextClassLoader().getResource("my.properties");
```

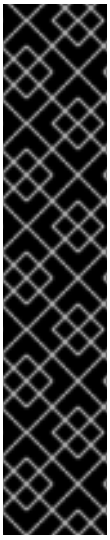
6.4. カスタムモジュールの削除

カスタムモジュールは、手作業 または 管理 CLI を使用 して削除できます。

手作業によるカスタムモジュールの削除
モジュールを手作業で削除する前に、デプロイされたアプリケーションやサーバー設定 (データソースなど) がそのモジュールを必要としていないことを確認してください。

カスタムモジュールを削除するには、**module.xml** ファイルと関連する JAR ファイルまたはその他のリソースが含まれる **EAP_HOME/modules/** 以下にあるモジュールのディレクトリーを削除します。たとえば、**main** スロットのカスタム MySQL JDBC ドライバーモジュールを削除するには、**EAP_HOME/modules/com/mysql/main/** ディレクトリーを削除します。

管理 CLI を使用したカスタムモジュールの削除
module remove 管理 CLI コマンドを使用するとカスタムモジュールを削除できます。



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

1. JBoss EAP サーバーを起動します。

2. 管理 CLI を起動します。

```
$ EAP_HOME/bin/jboss-cli.sh
```

3. **module remove** 管理 CLI コマンドを使用してカスタムモジュールを削除します。

```
module remove --name=MODULE_NAME
```

- 削除するモジュールが **main** 以外のスロットにある場合は、**--slot** 引数を使用します。

例: MySQL モジュールの削除

```
module remove --name=com.mysql
```

module --help を実行すると、このコマンドを使用したモジュールの追加および削除の詳細を表示できます。

6.5. グローバルモジュールの定義

モジュールを依存関係としてすべてのデプロイメントに追加する、グローバルモジュールのリストを定義できます。

注記

グローバルモジュールとして設定するモジュールの名前を知っている必要があります。含まれるモジュールの完全なリストとこれらのモジュールがサポートされているかについては、Red Hat カスタマーポータルの JBoss Enterprise Application Platform (EAP) 7 に含まれるモジュール を参照してください。デプロイメントにおけるモジュールの名前付けの規則は、Dynamic Module Naming の項を参照してください。

以下の管理 CLI コマンドを使用してグローバルモジュールのリストを定義します。

```
/subsystem=ee:write-attribute(name=global-modules,value=[{name=MODULE_NAME_1},
{name=MODULE_NAME_2}]
```

以下の管理 CLI コマンドを使用して、1 つのモジュールを既存のグローバルモジュールのリストに追加します。

```
/subsystem=ee:list-add(name=global-modules,value={name=MODULE_NAME})
```

管理コンソールを使用してグローバルモジュールを追加および削除することもできます。
Configuration タブから **EE** サブシステムに移動し、**Global Modules** セクションを選択します。

外部依存関係からグローバルモジュールにアクセスできるようにする必要がある場合は、明示的に可能にする必要があります。以下グローバルモジュールのサービスを外部で利用できるようにするには、以下のオプションを指定します。

- **services="import"** を、**jboss-deployment-structure.xml** のモジュールに追加します。
- global モジュール定義に **services="true"** を追加します。

  ```
  /subsystem=ee:write-attribute(name=global-modules,value=[{name=module1,services=true}]
  ```

  または、複数のモジュールを追加する場合は以下を使用します。

  ```
  /subsystem=ee:write-attribute(name=global-modules,value=[{name=module1,services=true},
  {name=module2,services=false}]
  ```

  新しいモジュールを既存のリストに追加するには、以下を行います。

  ```
  /subsystem=ee:list-add(name=global-modules,value={name=module1,services=true})
  ```

- 管理コンソールを使用してグローバルモジュールを定義する場合は、**Services** プロパティーの値が **On** であることを確認します。

6.6. グローバルディレクトリーの作成

グローバルディレクトリーは、グローバルモジュールの代わりのアプローチとなります。たとえば、グローバルモジュールに一覧表示されているライブラリーの名前を変更する場合は、グローバルモジュールを削除し、ライブラリーの名前を変更して新しいグローバルモジュールにライブラリーを追加する必要があります。グローバルディレクトリーにリストされたライブラリーの名前を変更する場合は、サーバーの再読み込みを行い、ライブラリー名の変更が全デプロイメントで利用できるようにするだけです。

グローバルディレクトリーを使用すると、以下を実行できます。

- デプロイされたアプリケーション間で複数のライブラリーを共有する
- 通常、アプリケーションライブラリーに追加された共通のフレームワークを共通のロケーションに移動することで、ライブラリーを維持する

グローバルディレクトリーを作成する場合、EE サブシステムはグローバルディレクトリーを設定し、ディレクトリーをスキャンして JBoss Modules のモジュール依存関係を作成します。モジュールの依存関係には、グローバルディレクトリーライブラリーおよび JAR ファイルが含まれます。このモジュール依存関係には、次のリソースローダーも含まれています。

- パスリソースローダーは、ファイルをリソースとしてアプリケーションに提供します。
- リソースローダーは、JAR ファイルに含まれるクラスをアプリケーションに提供します。

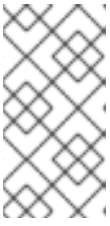
EE サブシステムは、デプロイされた各アプリケーションのシステム依存関係としてモジュールの依存関係を追加します。

前提条件

- オペレーティングシステムに標準ディレクトリーを作成します。この標準ディレクトリーには、アプリケーションにデプロイする必要のあるすべての JAR ファイルとリソースが含まれている必要があります。これにより、ディレクトリーツリーが作成されます。

  アプリケーションにコピーされた共通のライブラリーのリストを示す共通ディレクトリーの例:

  ```
  /my-common-libs/log4j2.xml
  /my-common-libs/libs/log4j-api-2.14.1.jar
  /my-common-libs/libs/log4j-core-2.14.1.jar
  ```

  注記

  サーバーはアプリケーションをデプロイし、グローバルディレクトリーを読み込むため、グローバルディレクトリーを設定してサーバーのライブラリーバージョンを上書きすることはできません。グローバルディレクトリーは、サーバーに同梱されたライブラリーを置き換えることはできません。

手順

1. サーバーの設定に応じて、グローバルディレクトリーを作成します。オプションの **relative-to** 属性を使用して、グローバルディレクトリーを相対パスで設定できます。

   スタンドアロンサーバーにグローバルディレクトリーを作成する例:

   ```
   [standalone@localhost:9990 /] /subsystem=ee/global-directory=my-common-libs:add(path=my-common-libs, relative-to=jboss.home.dir)
   ```

   マネージドドメイン内のサーバーにグローバルディレクトリーを作成する例:

   ```
   [domain@localhost:9990 /] /profile=default/subsystem=ee/global-directory=my-common-libs:add(path=my-common-libs, relative-to=jboss.server.data.dir)
   ```

   マネージドドメインのサーバーでは、**relative-path** 属性を使用して、**domain.xml** で定義される JBoss EAP プロファイルにグローバルディレクトリーを追加できます。**relative-to** 属性には、システムパスまたはカスタムシステムパスのいずれかを指定できます。

   注記

   マネージドドメインでサーバーを実行する場合は、グローバルディレクトリーのコンテンツがすべてのサーバーインスタンスで一貫していることを確認する必要があります。たとえば、各ホストには、グローバルディレクトリーのコンテンツを含むローカルファイルシステムディレクトリーが含まれている必要があります。

2. サーバーインスタンスをリロードして、グローバルディレクトリーをアクティベートします。サーバーが、ルートディレクトリーから開始して、アルファベット順に、各サブディレクトリーレベルを含むディレクトリーツリーの内容をスキャンできるように、サーバーをリロードする必要があります。サーバーは、各ディレクトリーレベルから JBoss Modules モジュールの依存関係に、アルファベット順にファイルを追加します。

   グローバルディレクトリーの内容を変更するか、グローバルディレクトリーで JAR ファイルを変更または追加する場合は、サーバーをリロードして、デプロイされたアプリケーションで変更を利用できるようにする必要があります。たとえば、グローバルディレクトリーの JAR ライブラリーを置き換える場合は、サーバーをリロードして、グローバルディレクトリーを再スキャンし、変更した JAR ライブラリーでデプロイされたアプリケーションを更新します。

関連情報

- JBoss EAP でのグローバルモジュールの定義に関する詳細は、グローバルモジュールの定義 を参照してください。
- ファイルシステムパスに論理名を使用する方法は、ファイルシステムパス を参照してください。
- JBoss EAP プロファイルの詳細は、JBoss EAP プロファイルの管理 を参照してください。
- ドメインコントローラーの使用方法は、ドメインコントローラー を参照してください。

6.7. グローバルディレクトリー設定の現在の値の読み取り

read-resource 操作を使用すると、グローバルディレクトリー設定の現在の値を読み取ることができます。

手順

- サーバーの設定に応じて、**read-resource** 操作を使用してグローバルディレクトリー設定の現在の値を読み取ります。

  スタンドアロンサーバーのグローバルディレクトリー設定の現在の値を読み取る場合の出力例。

```
[standalone@localhost:9990 /] /subsystem=ee/global-directory=my-common-libs:read-resource
{
    "outcome" => "success",
    "result" => {
        "path" => "my-common-libs",
        "relative-to" => "jboss.home.dir"
    }
}
```

  マネージドドメインのサーバーのグローバルディレクトリー設定の現在の値を読み取る場合の出力例。

```
[domain@localhost:9990 /] /subsystem=ee/global-directory=my-common-libs:read-resource
{
    "outcome" => "success",
    "result" => {
        "path" => "my-common-libs",
        "relative-to" => "jboss.server.data.dir"
    }
}
```

6.8. グローバルディレクトリーの削除

選択したサーバーからグローバルディレクトリーを削除できます。サーバー設定ファイルからグローバルディレクトリーリソースのみを削除します。基礎となるディレクトリーまたはそのファイルは削除しないでください。

手順

- スタンドアロンサーバーからグローバルディレクトリーを削除するには、以下のコマンドを使用します。

```
[standalone@localhost:9990 /] /subsystem=ee/global-directory=my-common-libs:remove()
```

- マネージドドメインのサーバー上のグローバルディレクトリーを削除するには、以下のコマンドを使用します。

```
[domain@localhost:9990 /] /profile=default/subsystem=ee/global-directory=my-common-libs:remove()
```

6.9. サブデプロイメント分離の設定

エンタープライズアーカイブ (EAR) の各サブデプロイメントは独自のクラスローダーを持つ動的モジュールです。サブデプロイメントは、**EAR/lib** のクラスへのアクセスを提供する親モジュールの暗黙

的な依存関係を常に持ちます。デフォルトでは、サブデプロイメントはその EAR 内にある他のサブデプロイメントのリソースにアクセスできます。

サブデプロイメントが他のサブデプロイメントに属するクラスにアクセスできないようにするには、JBoss EAP で厳格なサブデプロイメント分離を有効にします。この設定はすべてのデプロイメントに影響します。

すべてのデプロイメントを対象とするサブデプロイメントモジュール分離の有効化
サブデプロイメント分離は **ee** サブシステムから管理コンソールまたは管理 CLI を使用して有効または無効にできます。デフォルトでは、サブデプロイメント分離は false に設定され、サブデプロイメントは EAR 内にある他のサブデプロイメントのリソースにアクセスできます。

以下の管理 CLI を使用して EAR サブデプロイメント分離を有効にします。

```
/subsystem=ee:write-attribute(name=ear-subdeployments-isolated,value=true)
```

EAR のサブデプロイメントは他のサブデプロイメントからリソースにアクセスできなくなります。

6.10. 外部 JBOSS EAP モジュールディレクトリーの定義

JBoss EAP モジュールのデフォルトのディレクトリーは **EAP_HOME/modules** です。**JBOSS_MODULEPATH** 変数を使用すると JBoss EAP モジュールの他のディレクトリーを指定できます。以下の手順に従って、JBoss EAP 起動設定ファイルでこの変数を設定します。

注記

JBOSS_MODULEPATH を JBoss EAP 起動設定ファイルで設定する代わりに、環境変数として設定することもできます。

1. 起動設定ファイルを編集します。
   - スタンドアロンサーバーとして実行している場合は、**EAP_HOME/bin/standalone.conf** ファイル (Windows Server の場合は **standalone.conf.bat**) を編集します。
   - マネージドドメインで実行している場合は、**EAP_HOME/bin/domain.conf** ファイル (Windows Server の場合は **domain.conf.bat**) を編集します。
2. **JBOSS_MODULEPATH** 変数を設定します。 例を以下に示します。

   ```
   JBOSS_MODULEPATH="/path/to/modules/directory/"
   ```

   ディレクトリーのリストを指定するには、ディレクトリーのリストをコロン (**:**) で区切ります。

注記

Windows Server の場合、次の構文を使用して **JBOSS_MODULEPATH** 変数を設定します。

```
set "JBOSS_MODULEPATH /path/to/modules/directory/"
```

ディレクトリーのリストを指定するには、ディレクトリーのリストをセミコロン (**;**) で区切ります。

6.11. 動的モジュールの命名規則

JBoss EAP では、すべてのデプロイメントが、以下の規則に従って名前が付けられたモジュールとしてロードされます。

- WAR および JAR ファイルのデプロイメントは次の形式で名前が付けられます。

  ```
  deployment.DEPLOYMENT_NAME
  ```

  たとえば、**inventory.war** と **store.jar** のモジュール名はそれぞれ **deployment.inventory.war** と **deployment.store.jar** になります。

- エンタープライズアーカイブ (EAR) 内のサブデプロイメントは次の形式で名前が付けられます。

  ```
  deployment.EAR_NAME.SUBDEPLOYMENT_NAME
  ```

  たとえば、エンタープライズアーカイブ **accounts.ear** 内にある **reports.war** のサブデプロイメントのモジュール名は **deployment.accounts.ear.reports.war** になります。

第7章 アプリケーションのデプロイ

JBoss EAP には、管理者向けと開発者向けのアプリケーションデプロイメントおよび設定オプションが多くあります。管理者は、管理コンソール のグラフィカルインターフェイスや 管理 CLI のコマンドラインインターフェイスを使用して本番環境のアプリケーションデプロイメントを管理できます。開発者は、設定可能なファイルシステムの デプロイメントスキャナー、HTTP API、Red Hat Developer Studio などの IDE、および Maven などを含む、多くのテストオプションをアプリケーションのデプロイメントで使用できます。

アプリケーションをデプロイするときにデプロイメント記述子の検証を有効にするには、**org.jboss.metadata.parser.validate** システムプロパティーを **true** に設定します。これには、以下の方法の 1 つを使用します。

- サーバー起動時

```
$ EAP_HOME/bin/standalone.sh -Dorg.jboss.metadata.parser.validate=true
```

- 以下の管理 CLI コマンドでサーバー設定に追加

```
/system-property=org.jboss.metadata.parser.validate:add(value=true)
```

7.1. 管理 CLI を使用したアプリケーションのデプロイ

管理 CLI を使用してアプリケーションをデプロイすると、単一のコマンドラインインターフェイスでデプロイメントスクリプトを作成および実行できます。このスクリプト機能を使用して、特定のアプリケーションデプロイメントおよび管理シナリオを設定できます。スタンドアロンサーバーとして稼働している場合は単一サーバーのデプロイメント状態を管理でき、マネージドドメインで稼働している場合はサーバーのネットワーク全体のデプロイメントを管理できます。

7.1.1. 管理 CLI を使用したアプリケーションのスタンドアロンサーバーへのデプロイ

アプリケーションのデプロイ
管理 CLI で **deployment deploy-file** コマンドを使用し、アプリケーションデプロイメントへのパスを指定します。

```
deployment deploy-file /path/to/test-application.war
```

正常にデプロイされると、管理 CLI には何も出力されませんが、サーバーログにデプロイメントメッセージが記録されます。

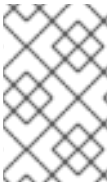
```
WFLYSRV0027: Starting deployment of "test-application.war" (runtime-name: "test-application.war")
WFLYUT0021: Registered web context: /test-application
WFLYSRV0010: Deployed "test-application.war" (runtime-name : "test-application.war")
```

同様に、以下の **deployment** コマンドを使用できます。

- **deployment deploy-cli-archive** を使用してコンテンツを **.cli** アーカイブファイルからデプロイします。CLI デプロイメントアーカイブは、**.cli** 拡張子を持つ **JAR** ファイルです。デプロイする必要があるアプリケーションアーカイブと、コマンドおよび操作が含まれる **deploy.scr** および **undeploy.scr** CLI スクリプトファイルが含まれます。**deploy.scr** スクリプトファイルには、アプリケーションアーカイブをデプロイし、環境を設定するコマンドと操作が含まれます。**undeploy.scr** スクリプトファイルには、アプリケーションアーカイブをアンデプロイし、環境をクリーンアップするコマンドが含まれます。

- **deployment deploy-url** を使用して、URL によって参照されるコンテンツをデプロイします。

deployment enable コマンドを使用して、無効になっている アプリケーションを有効にすることもできます。



注記

--runtime-name オプションで **runtime-name** 属性を指定する場合は、名前に **.war** 拡張子を含める必要があります。そうしないと、Web コンテキストが JBoss EAP によって登録されません。

アプリケーションのアンデプロイ
管理 CLI で **deployment undeploy** コマンドを使用し、デプロイメント名を指定します。これにより、リポジトリーからデプロイメントコンテンツが削除されます。アンデプロイする際にデプロイメントコンテンツを維持する場合は、Disable an Application を参照してください。

```
deployment undeploy test-application.war
```

正常にアンデプロイされると、管理 CLI には何も出力されませんが、サーバーログにアンデプロイメントメッセージが記録されます。

```
WFLYUT0022: Unregistered web context: /test-application
WFLYSRV0028: Stopped deployment test-application.war (runtime-name: test-application.war) in 62ms
WFLYSRV0009: Undeployed "test-application.war" (runtime-name: "test-application.war")
```

同様に、**deployment undeploy-cli-archive** を使用して **.cli** アーカイブファイルからコンテンツをアンデプロイできます。ワイルドカード (*) を使用してすべてのデプロイメントをアンデプロイすることも可能です。

```
deployment undeploy *
```

アプリケーションの無効化
デプロイメントコンテンツをリポジトリーから削除せずに、デプロイされたアプリケーションを無効にします。

```
deployment disable test-application.war
```

deployment disable-all コマンドを使用するとすべてのデプロイメントを無効化することができます。

```
deployment disable-all
```

アプリケーションの有効化
無効になっているデプロイされたアプリケーションを有効にします。

```
deployment enable test-application.war
```

deployment enable-all コマンドを使用するとすべてのデプロイメントを有効化することができます。

```
deployment enable-all
```

デプロイメントのリスト表示

管理 CLI で **deployment info** コマンドを使用して、デプロイメントの情報を表示します。

```
deployment info
```

出力には、ランタイム名、状態、有効であるかどうかなど、各デプロイメントの詳細が表示されます。

```
NAME                RUNTIME-NAME         PERSISTENT ENABLED STATUS
helloworld.war      helloworld.war       true       true    OK
test-application.war test-application.war true       true    OK
```

以下のコマンドは、名前を指定してデプロイメント情報を表示します。

```
deployment info helloworld.war
```

deployment list コマンドを使用して、デプロイメントをすべて表示することもできます。

7.1.2. 管理 CLI を使用したマネージドドメインでのアプリケーションのデプロイ

アプリケーションのデプロイ
管理 CLI で **deployment deploy-file** コマンドを使用し、アプリケーションデプロイメントへのパスを指定します。また、アプリケーションをデプロイするサーバーグループを指定する必要もあります。

- すべてのサーバーグループにアプリケーションをデプロイする場合

  ```
  deployment deploy-file /path/to/test-application.war --all-server-groups
  ```

- 特定のサーバーグループにアプリケーションをデプロイする場合

  ```
  deployment deploy-file /path/to/test-application.war --server-groups=main-server-group,other-server-group
  ```

正常にデプロイされると、管理 CLI には何も出力されませんが、サーバーログに各サーバーのデプロイメントメッセージが記録されます。

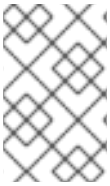
```
[Server:server-one] WFLYSRV0027: Starting deployment of "test-application.war" (runtime-name: "test-application.war")
[Server:server-one] WFLYUT0021: Registered web context: /test-application
[Server:server-one] WFLYSRV0010: Deployed "test-application.war" (runtime-name : "test-application.war")
```

同様に、以下の **deployment** コマンドを使用できます。

- **deployment deploy-cli-archive** コマンドを使用してコンテンツを **.cli** アーカイブファイルからデプロイします。CLI デプロイメントアーカイブは、**.cli** 拡張子を持つ **JAR** ファイルです。デプロイする必要があるアプリケーションアーカイブと、コマンドおよび操作が含まれる **deploy.scr** および **undeploy.scr** CLI スクリプトファイルが含まれます。**deploy.scr** スクリプトファイルには、アプリケーションアーカイブをデプロイし、環境を設定するコマンドと操作が含まれます。**undeploy.scr** スクリプトファイルには、アプリケーションアーカイブをアンデプロイし、環境をクリーンアップするコマンドが含まれます。

- **deployment deploy-url** コマンドを使用して、URL によって参照されるコンテンツをデプロイします。

deployment enable コマンドを使用して、無効になっている アプリケーションを有効にすることもできます。



注記

--runtime-name オプションで **runtime-name** 属性を指定する場合は、名前に **.war** 拡張子を含める必要があります。そうしないと、Web コンテキストが JBoss EAP によって登録されません。

アプリケーションのアンデプロイ
管理 CLI で **deployment undeploy** コマンドを使用し、デプロイメント名を指定します。また、アプリケーションをアンデプロイするサーバーグループを指定する必要もあります。特定のサーバーグループからのアンデプロイについては、アプリケーションの無効化 を参照してください。

すべてのサーバーグループからアプリケーションをアンデプロイします。

```
deployment undeploy test-application.war --all-relevant-server-groups
```

正常にアンデプロイされると、管理 CLI には何も出力されませんが、サーバーログに各サーバーのアンデプロイメントメッセージが記録されます。

```
[Server:server-one] WFLYUT0022: Unregistered web context: /test-application
[Server:server-one] WFLYSRV0028: Stopped deployment test-application.war (runtime-name: test-application.war) in 74ms
[Server:server-one] WFLYSRV0009: Undeployed "test-application.war" (runtime-name: "test-application.war")
```

同様に、**deployment undeploy-cli-archive** コマンドを使用して **.cli** アーカイブファイルからコンテンツをアンデプロイできます。ワイルドカード (*) を使用してすべてのデプロイメントをアンデプロイすることも可能です。

```
deployment undeploy * --all-relevant-server-groups
```

アプリケーションの無効化
特定のサーバーグループからデプロイされたアプリケーションを無効にし、そのデプロイメントを持つ他のサーバーグループのリポジトリーにそのコンテンツを保持します。

```
deployment disable test-application.war --server-groups=other-server-group
```

deployment disable-all コマンドを使用するとすべてのデプロイメントを無効化することができます。

```
deployment disable-all --server-groups=other-server-group
```

アプリケーションの有効化
無効になっているデプロイされたアプリケーションを有効にします。

```
deployment enable test-application.war
```

deployment enable-all コマンドを使用するとすべてのデプロイメントを有効化することができます。

```
deployment enable-all --server-groups=other-server-group
```

デプロイメントのリスト表示
管理 CLI で **deployment info** コマンドを使用して、デプロイメントの情報を表示します。デプロイメント名またはサーバーグループでデプロイメント情報を絞り込むことができます。

以下のコマンドは、名前を指定してデプロイメント情報を表示します。

```
deployment info helloworld.war
```

出力には、デプロイメントと各サーバーグループでの状態が表示されます。

```
NAME              RUNTIME-NAME
helloworld.war    helloworld.war

SERVER-GROUP      STATE
main-server-group  enabled
other-server-group added
```

以下のコマンドは、サーバーグループを指定してデプロイメント情報を表示します。

```
deployment info --server-group=other-server-group
```

出力には、デプロイメントと、指定のサーバーグループに対する状態が表示されます。

```
NAME               RUNTIME-NAME        STATE
helloworld.war     helloworld.war      added
test-application.war test-application.war enabled
```

deployment list コマンドを使用して、ドメインのデプロイメントをすべて表示することもできます。

7.2. 管理コンソールを使用したアプリケーションのデプロイ

管理コンソールを使用してアプリケーションをデプロイすると、使用が簡単なグラフィカルインターフェイスを利用することができます。サーバーまたはサーバーグループにデプロイされたアプリケーションを一目で確認できるほか、必要に応じてアプリケーションを有効または無効にしたり、アプリケーションをコンテンツリポジトリーから削除したりすることができます。

7.2.1. 管理コンソールを使用したアプリケーションのスタンドアロンサーバーへのデプロイ

JBoss EAP 管理コンソールの **Deployments** タブからデプロイメントを表示および管理できます。

アプリケーションのデプロイ
追加 (+) ボタンをクリックします。デプロイメントのアップロード、未管理デプロイメントの追加、または 空のデプロイメントの作成 を選択して、アプリケーションをデプロイできます。デプロイメントはデフォルトで有効になります。

- 新規デプロイメントのアップロード
  サーバーのコンテンツリポジトリーにコピーされ、JBoss EAP によって管理されるアプリケーションをアップロードします。

- 未管理デプロイメントの追加
  デプロイメントの場所を指定します。このデプロイメントはサーバーのコンテンツリポジトリーにはコピーされず、JBoss EAP によって管理されません。

- 空のデプロイメントの作成
  空のデプロイメント形式 (exploded) のデプロイメントを作成します。このデプロイメントの作成後、ファイルをデプロイメントに追加できます。

アプリケーションのアンデプロイ
デプロイメントを選択し、**Undeploy** オプションを選びます。JBoss EAP により、コンテンツリポジトリーからデプロイメントが削除されます。

アプリケーションの無効化
デプロイメントを選択し、無効オプションを選択してアプリケーションを無効にします。これにより、デプロイメントがアンデプロイされますが、コンテンツリポジトリーから削除されません。

アプリケーションの置換
デプロイメントを選択し、置換オプションを選択します。元のバージョンと同じ名前を持つ新しいバージョンのデプロイメントを選択し、**Finish** をクリックします。これにより、元のバージョンのデプロイメントがアンデプロイおよび削除され、新しいバージョンがデプロイされます。

7.2.2. 管理コンソールを使用したマネージドドメインでのアプリケーションのデプロイ

JBoss EAP 管理コンソールの **Deployments** タブではデプロイメントを表示および管理できます。

- Content Repository
  管理されるデプロイメントと管理されないデプロイメントはすべて **Content Repository** セクションで表示されます。ここで、デプロイメントを追加したり、デプロイメントをサーバーグループにデプロイすることができます。

- Server Groups
  1 つまたは複数のサーバーグループにデプロイされたデプロイメントは **Server Groups** セクションにリストされます。ここで、デプロイメントを直接サーバーグループに追加したり、有効にしたりすることができます。

アプリケーションの追加

1. **Content Repository** で 追加ボタンをクリックします。
2. デプロイメントのアップロード または 未管理のデプロイメント作成 を選択し、アプリケーションを追加します。
3. プロンプトに従ってアプリケーションをデプロイします。
   デプロイメントを有効にするには、デプロイメントをサーバーグループにデプロイする必要があります。

アプリケーションのサーバーグループへのデプロイ

1. **Content Repository** でデプロイメントを選択し、**Deploy** を選択します。
2. このデプロイメントをデプロイするサーバーグループを 1 つ以上選択します。
3. 選択したサーバーグループのデプロイメントを有効にするオプションを任意で選択することもできます。

アプリケーションのサーバーグループからのアンデプロイ

1. **Server Groups** で適切なサーバーグループを選択します。
2. 希望のデプロイメントを選択し、**Undeploy** ボタンをクリックします。

また、**Content Repository** でデプロイメントの **Undeploy** ボタンを選択して、複数のサーバーグループから 1 度にデプロイメントをアンデプロイすることもできます。

アプリケーションの削除

1. デプロイメントがサーバーグループにデプロイされている場合は、必ずデプロイメントをアンデプロイします。
2. **Content Repository** でデプロイメントを選択し、削除 を選択します。

これにより、コンテンツリポジトリーからデプロイメントが削除されます。

アプリケーションの無効化

1. **Server Groups** で適切なサーバーグループを選択します。
2. 無効にするデプロイメントを選択し、無効 を選択します。

これにより、デプロイメントがアンデプロイされますが、コンテンツリポジトリーから削除されません。

アプリケーションの置換

1. **Content Repository** からデプロイメントを選択し、置換ボタンをクリックします。
2. 元のバージョンと同じ名前を持つ新しいバージョンのデプロイメントを選択し、置換 をクリックします。

これにより、元のバージョンのデプロイメントがアンデプロイおよび削除され、新しいバージョンがデプロイされます。

7.3. デプロイメントスキャナーを使用したアプリケーションのデプロイ

デプロイメントスキャナーは、デプロイするアプリケーションのデプロイメントディレクトリーを監視します。デフォルトでは、デプロイメントスキャナーは 5 秒ごとに **EAP_HOME/standalone/deployments/** をスキャンし、変更を確認します。デプロイメントの状態を示し、アンデプロイや再デプロイなどのデプロイメントに対するアクションをトリガーするため、マーカーファイルが使用されます。

本番環境では、アプリケーションのデプロイメントに管理コンソールまたは管理 CLI の使用が推奨されますが、デプロイメントスキャナーは開発者の便宜を図るために提供されます。これにより、ペースの早い開発サイクルに適した方法でアプリケーションを構築およびテストできます。デプロイメントスキャナーは、他のデプロイメント方法と併用しないでください。

デプロイメントスキャナーは JBoss EAP をスタンドアロンサーバーとして実行している場合のみ利用できます。

7.3.1. デプロイメントスキャナーを使用したアプリケーションのスタンドアロンサーバーへのデプロイ

デプロイメントスキャナーを設定して XML 、zip 形式、およびデプロイメント形式のコンテンツの自動デプロイメントを有効または無効にすることができます。自動デプロイメントが無効の場合、マーカーファイルを手作業で作成してデプロイメントのアクションをトリガーする必要があります。利用できるマーカーファイルタイプやそれらの目的に関する詳細は、デプロイメントスキャナーのマーカーファイル の項を参照してください。

デフォルトでは、XML および zip 形式のコンテンツの自動デプロイメントは有効になっています。各コンテンツタイプの自動デプロイメントの設定に関する詳細は デプロイメントスキャナーの設定 を参照してください。

警告

デプロイメントスキャナーを使用したデプロイメントは開発者の便宜を図るために提供され、本番環境での使用は推奨されません。デプロイメントスキャナーは他のデプロイメント方法と併用しないでください。

アプリケーションのデプロイ
コンテンツをデプロイメントフォルダーにコピーします。

```
$ cp /path/to/test-application.war EAP_HOME/standalone/deployments/
```

自動デプロイメントが有効の場合、このファイルは自動的に選択され、デプロイされます。さらに、**.deployed** マーカーファイルが作成されます。自動デプロイメントが無効の場合、**.dodeploy** マーカーファイルを手作業で追加し、デプロイメントをトリガーする必要があります。

```
$ touch EAP_HOME/standalone/deployments/test-application.war.dodeploy
```

アプリケーションのアンデプロイ
.deployed マーカーファイルを削除して、アンデプロイメントをトリガーします。

```
$ rm EAP_HOME/standalone/deployments/test-application.war.deployed
```

自動デプロイメントが有効な場合、アンデプロイメントをトリガーする **test-application.war** ファイルを削除することもできます。これは、デプロイメント形式のデプロイメントには適用されないことに注意してください。

アプリケーションの再デプロイ
.dodeploy マーカーファイルを作成し、再デプロイを開始します。

```
$ touch EAP_HOME/standalone/deployments/test-application.war.dodeploy
```

7.3.2. デプロイメントスキャナーの設定

デプロイメントスキャナーは管理コンソールまたは管理 CLI を使用して設定できます。スキャンの間隔、デプロイメントフォルダーの場所、特定のアプリケーションファイルタイプの自動デプロイメントなど、デプロイメントスキャナーの動作を設定できます。また、デプロイメントスキャナーを完全に無効にすることもできます。

利用できるデプロイメントスキャナー属性の詳細は、デプロイメントスキャナーの属性 の項を参照してください。

以下の管理 CLI コマンドを使用してデフォルトのデプロイメントスキャナーを設定します。

デプロイメントスキャナーの無効化

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=scan-enabled,value=false)
```

default デプロイメントスキャナーが無効になります。

スキャン間隔の変更

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=scan-interval,value=10000)
```

スキャンの間隔が **5000** ミリ秒 (5 秒) から **10000** ミリ秒 (10 秒) に変更されます。

デプロイメントフォルダーの変更

```
/subsystem=deployment-scanner/scanner=default:write-
attribute(name=path,value=/path/to/deployments)
```

デプロイメントフォルダーの場所がデフォルトの **EAP_HOME/standalone/deployments** から **/path/to/deployments** に変更されます。

relative-to 属性が指定されていない場合、**path** の値は絶対パスになります。relative-to 属性が指定されている場合は相対パスになります。

デプロイメント形式のコンテンツの自動デプロイメントの有効化

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=auto-deploy-
exploded,value=true)
```

デフォルトで無効になっているデプロイメント形式のコンテンツの自動デプロイメントを有効にします。

zip 形式のコンテンツの自動デプロイメントの無効化

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=auto-deploy-
zipped,value=false)
```

デフォルトで有効になっている zip 形式のコンテンツの自動デプロイメントを無効にします。

XML コンテンツの自動デプロイメントの無効化

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=auto-deploy-xml,value=false)
```

デフォルトで有効になっている XML コンテンツの自動デプロイメントを無効にします。

7.3.3. カスタムデプロイメントスキャナーの定義

新しいデプロイメントスキャナーを追加するには、管理 CLI を使用するか、管理コンソールの **Configuration** タブから **Deployment Scanners** サブシステムに移動します。デプロイメントを確認するためにスキャンする新しいディレクトリーを定義します。デフォルトのデプロイメントスキャナーは **EAP_HOME/standalone/deployments** を監視します。既存のデプロイメントスキャナーの設定に関する詳細は デプロイメントスキャナーの設定 を参照してください。

以下の管理 CLI コマンドは、**EAP_HOME/standalone/new_deployment_dir** を 5 秒ごとにチェックしてデプロイメントを確認する新しいデプロイメントスキャナーを追加します。

```
/subsystem=deployment-scanner/scanner=new-scanner:add(path=new_deployment_dir,relative-
to=jboss.server.base.dir,scan-interval=5000)
```

注記

指定のディレクトリーがすでに存在しないと、このコマンドに失敗し、エラーが発生します。

新しいデプロイメントスキャナーが定義され、デプロイメントを確認するために指定のディレクトリーが監視されます。

7.4. MAVEN を使用したアプリケーションのデプロイ

Apache Maven を使用してアプリケーションをデプロイすると、JBoss EAP へのデプロイメントを簡単に既存の開発ワークフローに取り入れることができます。

アプリケーションをアプリケーションサーバーにデプロイおよびアンデプロイする簡単な操作を提供する WildFly Maven Plugin を使用すると、Maven を使用してアプリケーションを JBoss EAP にデプロイできます。

7.4.1. Maven を使用したアプリケーションのスタンドアロンサーバーへのデプロイ

以下の手順では、Maven を使用して JBoss EAP の **helloworld** クイックスタートをスタンドアロンサーバーにデプロイおよびアンデプロイする方法を示します。

JBoss EAP クイックスタートの詳細は、JBoss EAP **Getting Started Guide**の Using the Quickstart Examples を参照してください。

アプリケーションのデプロイ
Maven **pom.xml** ファイルで WildFly Maven Plugin を初期化します。これは、JBoss EAP クイックスタートの **pom.xml** ファイルで設定されているはずです。

```
<plugin>
  <groupId>org.wildfly.plugins</groupId>
  <artifactId>wildfly-maven-plugin</artifactId>
  <version>${version.wildfly.maven.plugin}</version>
</plugin>
```

helloworld クイックスタートディレクトリーで以下の Maven コマンドを実行します。

```
$ mvn clean install wildfly:deploy
```

デプロイする Maven コマンドの実行後、ターミナルウインドウにはデプロイメントの成功を表す以下の出力が表示されます。

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.981 s
[INFO] Finished at: 2015-12-23T15:06:13-05:00
[INFO] Final Memory: 21M/231M
[INFO] ------------------------------------------------------------------------
```

アクティブなサーバーインスタンスのサーバーログでデプロイメントの成功を確認することもできます。

```
WFLYSRV0027: Starting deployment of "helloworld.war" (runtime-name: "helloworld.war")
WFLYUT0021: Registered web context: /helloworld
WFLYSRV0010: Deployed "helloworld.war" (runtime-name : "helloworld.war")
```

アプリケーションのアンデプロイ
helloworld クイックスタートディレクトリーで以下の Maven コマンドを実行します。

```
$ mvn wildfly:undeploy
```

アンデプロイする Maven コマンドの実行後、ターミナルウインドウにはアンデプロイメントの成功を表す以下の出力が表示されます。

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.237 s
[INFO] Finished at: 2015-12-23T15:09:10-05:00
[INFO] Final Memory: 10M/183M
[INFO] ------------------------------------------------------------------------
```

アクティブなサーバーインスタンスのサーバーログでアンデプロイメントの成功を確認することもできます。

```
WFLYUT0022: Unregistered web context: /helloworld
WFLYSRV0028: Stopped deployment helloworld.war (runtime-name: helloworld.war) in 27ms
WFLYSRV0009: Undeployed "helloworld.war" (runtime-name: "helloworld.war")
```

7.4.2. Maven を使用したマネージドドメインでのアプリケーションのデプロイ

以下の手順では、Maven を使用してマネージドドメインで JBoss EAP の **helloworld** クイックスタートをデプロイおよびアンデプロイする方法を示します。

JBoss EAP クイックスタートの詳細は、JBoss EAP **Getting Started Guide**の Using the Quickstart Examples を参照してください。

アプリケーションのデプロイ
マネージドドメインでアプリケーションをデプロイする場合、アプリケーションをデプロイするサーバーグループを指定する必要があります。これは、Maven の **pom.xml** ファイルで設定されます。

pom.xml の以下の設定は WildFly Maven Plugin を初期化し、 **main-server-group** をアプリケーションがデプロイされるサーバーグループとして指定します。

```
<plugin>
  <groupId>org.wildfly.plugins</groupId>
  <artifactId>wildfly-maven-plugin</artifactId>
  <version>${version.wildfly.maven.plugin}</version>
  <configuration>
    <domain>
      <server-groups>
        <server-group>main-server-group</server-group>
      </server-groups>
```

```
    </domain>
  </configuration>
</plugin>
```

helloworld クイックスタートディレクトリーで以下の Maven コマンドを実行します。

```
$ mvn clean install wildfly:deploy
```

デプロイする Maven コマンドの実行後、ターミナルウインドウにはデプロイメントの成功を表す以下の出力が表示されます。

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.005 s
[INFO] Finished at: 2016-09-02T14:36:17-04:00
[INFO] Final Memory: 21M/226M
[INFO] ------------------------------------------------------------------------
```

アクティブなサーバーインスタンスのサーバーログでデプロイメントの成功を確認することもできます。

```
WFLYSRV0027: Starting deployment of "helloworld.war" (runtime-name: "helloworld.war")
WFLYUT0021: Registered web context: /helloworld
WFLYSRV0010: Deployed "helloworld.war" (runtime-name : "helloworld.war")
```

アプリケーションのアンデプロイ
helloworld クイックスタートディレクトリーで以下の Maven コマンドを実行します。

```
$ mvn wildfly:undeploy
```

アンデプロイする Maven コマンドの実行後、ターミナルウインドウにはアンデプロイメントの成功を表す以下の出力が表示されます。

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.750 s
[INFO] Finished at: 2016-09-02T14:45:10-04:00
[INFO] Final Memory: 10M/184M
[INFO] ------------------------------------------------------------------------
```

アクティブなサーバーインスタンスのサーバーログでアンデプロイメントの成功を確認することもできます。

```
WFLYUT0022: Unregistered web context: /helloworld
WFLYSRV0028: Stopped deployment helloworld.war (runtime-name: helloworld.war) in 106ms
WFLYSRV0009: Undeployed "helloworld.war" (runtime-name: "helloworld.war")
```

7.5. HTTP API を使用したアプリケーションのデプロイ

HTTP API を使用してアプリケーションを JBoss EAP にデプロイするには、**curl** コマンドを使用します。HTTP API の使用に関する詳細は、HTTP API を参照してください。

7.5.1. HTTP API を使用したアプリケーションのスタンドアロンサーバーへのデプロイ

デフォルトでは、HTTP API は **http://HOST:PORT/management** でアクセスできます (例: **http://localhost:9990/management**)。

アプリケーションのデプロイ

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type: application/json" -u USER:PASSWORD -d '{"operation" : "composite", "address" : [], "steps" : [{"operation" : "add", "address" : {"deployment" : "test-application.war"}, "content" : [{"url" : "file:/path/to/test-application.war"}]},{"operation" : "deploy", "address" : {"deployment" : "test-application.war"}}],"json.pretty":1}'
```

アプリケーションのアンデプロイ

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type: application/json" -u USER:PASSWORD -d '{"operation" : "composite", "address" : [], "steps" : [{"operation" : "undeploy", "address" : {"deployment" : "test-application.war"}},{"operation" : "remove", "address" : {"deployment" : "test-application.war"}}],"json.pretty":1}'
```

JSON リクエストをプログラムで生成する方法の詳細は、この Red Hat ナレッジベースの記事 を参照してください。

7.5.2. HTTP API を使用したマネージドドメインでのアプリケーションのデプロイ

デフォルトでは、HTTP API は **http://HOST:PORT/management** でアクセスできます (例: **http://localhost:9990/management**)。

アプリケーションのデプロイ

1. デプロイメントをコンテンツリポジトリーに追加します。

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type: application/json" -u USER:PASSWORD -d '{"operation" : "add", "address" : {"deployment" : "test-application.war"}, "content" : [{"url" : "file:/path/to/test-application.war"}],"json.pretty":1}'
```

2. デプロイメントを指定のサーバーグループに追加します。

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type: application/json" -u USER:PASSWORD -d '{"operation" : "add", "address" : {"server-group" : "main-server-group","deployment":"test-application.war"},"json.pretty":1}'
```

3. サーバーグループにアプリケーションをデプロイします。

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type: application/json" -u USER:PASSWORD -d '{"operation" : "deploy", "address" : {"server-group" : "main-server-group","deployment":"test-application.war"},"json.pretty":1}'
```

アプリケーションのアンデプロイ

1. 割り当てられたサーバーグループすべてからデプロイメントを削除します。

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type:
application/json" -u USER:PASSWORD -d '{"operation" : "remove", "address" : {"server-
group" : "main-server-group","deployment":"test-application.war"},"json.pretty":1}'
```

2. コンテンツリポジトリーからデプロイメントを削除します。

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type:
application/json" -u USER:PASSWORD -d '{"operation" : "remove", "address" :
{"deployment" : "test-application.war"}, "json.pretty":1}'
```

7.6. デプロイメントの動作のカスタマイズ

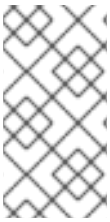
7.6.1. デプロイメントコンテンツのカスタムディレクトリーの定義

JBoss EAP では、デプロイされたコンテンツを格納する場所をカスタマイズし、定義できます。

スタンドアロンサーバーでのカスタムディレクトリーの定義
デフォルトでは、スタンドアロンサーバーのデプロイされたコンテンツは **EAP_HOME/standalone/data/content** ディレクトリーに格納されます。この場所を変更するには、サーバーの起動時に **-Djboss.server.deploy.dir** 引数を渡します。

```
$ EAP_HOME/bin/standalone.sh -Djboss.server.deploy.dir=/path/to/new_deployed_content
```

選択する場所は JBoss EAP インスタンスすべてで一意になる必要があります。



注記

jboss.server.deploy.dir プロパティーは、管理コンソールまたは管理 CLI を使用してデプロイされたコンテンツの格納に使用されるディレクトリーを指定します。デプロイメントスキャナーによって監視されるカスタムデプロイメントディレクトリーを定義する場合は、デプロイメントスキャナーの設定 を参照してください。

マネージドドメインでのカスタムディレクトリーの定義
デフォルトでは、マネージドドメインのデプロイされたコンテンツは **EAP_HOME/standalone/data/content** ディレクトリーに格納されます。この場所を変更するには、ドメインの起動時に **-Djboss.domain.deployment.dir** 引数を渡します。

```
$ EAP_HOME/bin/domain.sh -Djboss.domain.deployment.dir=/path/to/new_deployed_content
```

選択する場所は JBoss EAP インスタンスすべてで一意になる必要があります。

7.6.2. デプロイメント順序の制御

JBoss EAP では、サーバー起動時にアプリケーションがデプロイされる順序を細かく制御できます。複数の EAR ファイルに存在するアプリケーションのデプロイメント順序を徹底することができ、再起動後の順序を永続化することもできます。

jboss-all.xml デプロイメント記述子を使用してトップレベルのデプロイメント間で依存関係を宣言できます。

たとえば、最初にデプロイされた **framework.ear** に依存する **app.ear** がある場合、以下のように **app.ear/META-INF/jboss-all.xml** ファイルを作成できます。

```
<jboss xmlns="urn:jboss:1.0">
  <jboss-deployment-dependencies xmlns="urn:jboss:deployment-dependencies:1.0">
    <dependency name="framework.ear" />
  </jboss-deployment-dependencies>
</jboss>
```

注記

デプロイメントのランタイム名を **jboss-all.xml** ファイルの依存名として使用することができます。

これにより、**app.ear** の前に **framework.ear** がデプロイされます。

重要

app.ear で **jboss-all.xml** ファイルを作成し、**framework.ear** をデプロイしない場合、サーバーは **app.ear** のデプロイを試行して失敗します。

7.6.3. デプロイメントコンテンツのオーバーライド

デプロイメントオーバーレイ を使用すると、デプロイメントアーカイブのコンテンツを物理的に変更せずに既存デプロイメントにコンテンツをオーバーレイすることができます。これにより、アーカイブを再構築せずに実行時にデプロイメント記述子、ライブラリー JAR ファイル、クラス、Jakarta Server Pages ページ、およびその他のファイルをオーバーライドできます。

これは、異なる設定が必要な異なる環境にデプロイメントを適応する必要がある場合に便利です。たとえば、アプリケーションのライフサイクルに従って開発からテスト、ステージ、および実稼働とデプロイメントを移動する場合、目的の環境に応じてデプロイメント記述子の交換、アプリケーションのブランディングを変更するための静的 Web リソースの変更、JAR ライブラリーの別バージョンへの置き換えなどを行う可能性があります。また、方針やセキュリティーの制限によりアーカイブを変更できない、設定変更が必要なインストールにも便利な機能です。

デプロイメントオーバーレイを定義する場合、デプロイメントアーカイブのファイルを置き換える、ファイルシステム上のファイルを定義します。さらに、デプロイメントオーバーレイの影響を受けるデプロイメントを指定する必要もあります。変更を有効にするには、影響を受けるデプロイメントを再デプロイする必要があります。

パラメーター

以下のパラメーターのいずれかを使用して、デプロイメントオーバーレイを設定できます。

- **name**: デプロイメントオーバーレイの名前。
- **content**: ファイルシステム上のファイルをアーカイブの置き換えるファイルにマップするコンマ区切りリスト。各エントリーの形式は **ARCHIVE_PATH=FILESYSTEM_PATH** です。
- **deployments**: このオーバーレイがリンクされるデプロイメントのコンマ区切りリスト。
- **redeploy-affected**: 影響を受けるデプロイメントをすべて再デプロイします。

使用方法の詳細を表示するには **deployment-overlay --help** を実行してください。

手順

1. **deployment-overlay add** 管理 CLI コマンドを使用してデプロイメントオーバーレイを追加します。

```
deployment-overlay add --name=new-deployment-overlay --content=WEB-
INF/web.xml=/path/to/other/web.xml --deployments=test-application.war --redeploy-affected
```

注記

管理対象ドメインでは、**--server-groups** を使用して該当するサーバーグループを指定するか、**--all-server-groups** を使用してすべてのサーバーグループを指定します。

2. デプロイメントオーバーレイの作成後、既存のオーバーレイへのコンテンツの追加、オーバーレイのデプロイメントへのリンク、またはオーバーレイの削除を行うことができます。

3. オプション: オーバーレイ設定を指定し、**<overlay>** 要素を使用して、HTML、イメージ、またはビデオなどの静的 Web リソースが含まれる外部ディレクトリーにリンクできます。**<overlay>** 要素は、JAR オーバーレイの手順と同様に、Web アプリケーションの静的ファイルをオーバーレイする静的ファイルを指定します。この要素は、アプリケーションファイル **jboss-web.xml** にあります。この要素設定では、アプリケーションを再パッケージ化する必要はありません。

   以下の例は、**<overlay>** 要素のシステムプロパティーの置換を示しています。**{example.path.to.overlay}** は、**/PATH/TO/STATIC/WEB/CONTENT** の場所を定義します。

   例: jboss-web.xml ファイルの <overlay> 要素

```
<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-web_10_0.xsd"
           version="10.0">
        <overlay>{example.path.to.overlay}</overlay>
</jboss-web>
```

   jboss-descriptor-property-replacement が **true** に設定されている場合、<overlay> 要素 にシステムプロパティーを指定できます。

   jboss-descriptor-property-replacement を設定するには、以下の管理 CLI コマンドを使用します。

```
/subsystem=ee:write-attribute(name=jboss-descriptor-property-replacement,value=true)
```

   このコマンドは、以下の XML コンテンツを JBoss EAP 設定の **ee** サブシステムに追加します。

```
<subsystem xmlns="urn:jboss:domain:ee:4.0">
    <jboss-descriptor-property-replacement>true</jboss-descriptor-property-replacement>
</subsystem>
```

注記

<overlay> 要素は、EAP プロジェクトにすでに存在するデプロイメントファイルをオーバーライドしません。複数の **<overlay>** 要素に同じファイルが含まれている場合、優先順位はアプリケーションの **jboss-web.xml** ファイル内のオーバーレイ要素の順序に基づいて決定されます。

7.6.4. ロールアウト計画の使用

ロールアウト計画
マネージドドメインでは、ドメインまたはホストレベルのリソースで目的となる操作は複数のサーバーに影響する可能性があります。このような操作には、サーバーに適用される操作の順序を説明するロールアウト計画や、一部のサーバーで実行に失敗した場合に操作を元に戻すかどうかを指示するポリシーなどが含まれます。指定されたロールアウト計画がない場合、デフォルトのロールアウト計画 が使用されます。

以下は 5 つのサーバーグループが関係するロールアウト計画の例になります。操作は順次 (**in-series**) または同時 (**concurrent-groups**) にサーバーグループへ適用することができます。構文の詳細は ロールアウト計画の構文 を参照してください。

```
{"my-rollout-plan" => {"rollout-plan" => {
    "in-series" => [
        {"concurrent-groups" => {
            "group-A" => {
                "max-failure-percentage" => "20",
                "rolling-to-servers" => "true"
            },
            "group-B" => undefined
        }},
        {"server-group" => {"group-C" => {
            "rolling-to-servers" => "false",
            "max-failed-servers" => "1"
        }}},
        {"concurrent-groups" => {
            "group-D" => {
                "max-failure-percentage" => "20",
                "rolling-to-servers" => "true"
            },
            "group-E" => undefined
        }}
    ],
    "rollback-across-groups" => "true"
}}}
```

上記の例を見ると、3 つの段階を経てドメインのサーバーに操作が適用されることが分かります。あるサーバーグループのポリシーによってサーバーグループ全体で操作のロールバックが引き起こされると、他のサーバーグループもすべてロールバックされます。

1. サーバーグループ **group-A** と **group-B** には同時に操作が適用されます。**group-A** のサーバーには操作が順次適用され、**group-B** のすべてのサーバーは操作を同時に処理します。**group-A** で操作の適用に失敗したサーバーが 20 % を超えると、グループ全体でロールバックが実行されます。**group-B** で操作を適用できなかったサーバーがあると、このグループ全体でロールバックが実行されます。

2. **group-A** と **group-B** のすべてのサーバーが完了すると、**group-C** のサーバーに操作が適用されます。これらのサーバーは操作を同時に処理します。**group-C** では操作を適用できなかったサーバーが 2 台以上あると、グループ全体でロールバックが実行されます。

3. **group-C** のすべてのサーバーが完了すると、操作が **group-D** と **group-E** に同時に適用されます。**group-D** のサーバーには操作が順次適用され、**group-E** のすべてのサーバーは操作を同時に処理します。**group-D** で操作の適用に失敗したサーバーが 20 % を超えると、グループ全体でロールバックが実行されます。**group-E** で操作を適用できなかったサーバーがあると、このグループ全体でロールバックが実行されます。

ロールアウト計画の構文
ロールアウト計画は以下のいずれかの方法で指定できます。

- **deploy** コマンド操作ヘッダーでロールアウト計画を定義します。詳細は ロールアウト計画を使用したデプロイ を参照してください。

- **rollout-plan** コマンドを使用してロールアウト計画を保存し、**deploy** コマンド操作ヘッダーでプラン名を参照します。詳細は 保存したロールアウト計画を使用したデプロイ を参照してください。

上記の方法で最初に使用するコマンドは異なりますが、どちらも **rollout** 操作ヘッダーを使用してロールアウト計画を定義します。以下の構文を使用します。

```
rollout (id=PLAN_NAME | SERVER_GROUP_LIST) [rollback-across-groups]
```

- **PLAN_NAME** は、**rollout-plan** コマンドを使用して保存されたロールアウト計画の名前です。

- **SERVER_GROUP_LIST** はサーバーグループのリストです。各サーバーグループで操作を順次実行する場合はコンマ (,) を使用して複数のサーバーグループを区切ります。各サーバーグループで同時に操作を実行する場合はキャレット (^) を区切り文字として使用します。

  - 各サーバーグループでは、以下のポリシーをかっこで囲んで設定します。複数のポリシーはコンマで区切ります。

    - **rolling-to-servers**: ブール値。**true** に設定すると、操作はグループの各サーバーに順次適用されます。**false** に設定した場合、または指定がない場合は、操作はグループのサーバーに同時に適用されます。

    - **max-failed-servers**: 整数値。グループにおける操作の適用に失敗したサーバー合計数の最大率 (パーセント) で、失敗したサーバーの率がこの値を超えるとグループのすべてのサーバーが元に戻されます。指定がない場合のデフォルト値は **0** で、操作の適用に失敗したサーバーが 1 台でもあるとグループ全体でロールバックが引き起こされます。

    - **max-failed-servers**: **0** から **100** までの整数値。グループにおける操作の適用に失敗したサーバー合計数の最大率 (パーセント) で、失敗したサーバーの率がこの値を超えるとグループのすべてのサーバーが元に戻されます。指定がない場合のデフォルト値は **0** で、操作の適用に失敗したサーバーが 1 台でもあるとグループ全体でロールバックが引き起こされます。

注記

max-failed-servers と **max-failure-percentage** の両方がゼロ以外の値に設定された場合、**max-failure-percentage** が優先されます。

- **rollback-across-groups**: ブール値。1 つのサーバーグループのサーバーすべてで操作をロールバックする必要があると、すべてのサーバーグループでロールバックの実行を引き起こすかどうかを指定します。デフォルトは **false** です。

ロールアウト計画を使用したデプロイ
ロールアウト計画の完全詳細を直接 **deploy** コマンドに提供するには、**rollout** 設定を **headers** 引数に渡します。形式の詳細は ロールアウト計画の構文 を参照してください。

以下の管理 CLI コマンドは、順次のデプロイメントに **rolling-to-servers=true** を指定するデプロイメント計画を使用して、アプリケーションを **main-server-group** サーバーグループにデプロイします。

```
deploy /path/to/test-application.war --server-groups=main-server-group --headers={rollout main-server-group(rolling-to-servers=true)}
```

保存したロールアウト計画を使用したデプロイ
ロールアウト計画は複雑になることがあるため、ロールアウト計画の詳細を保存するオプションがあります。これにより、毎回ロールアウト計画の完全詳細を必要とせずに、ロールアウト計画を使用するときにロールアウト計画の名前を参照することができます。

1. **rollout-plan** 管理 CLI コマンドを使用してロールアウト計画を保存します。形式の詳細は ロールアウト計画の構文 を参照してください。

   ```
   rollout-plan add --name=my-rollout-plan --content={rollout main-server-group(rolling-to-servers=false,max-failed-servers=1),other-server-group(rolling-to-servers=true,max-failure-percentage=20) rollback-across-groups=true}
   ```

   これにより、以下のデプロイメント計画が作成されます。

   ```
   "rollout-plan" => {
       "in-series" => [
           {"server-group" => {"main-server-group" => {
               "rolling-to-servers" => false,
               "max-failed-servers" => 1
           }}},
           {"server-group" => {"other-server-group" => {
               "rolling-to-servers" => true,
               "max-failure-percentage" => 20
           }}}
       ],
       "rollback-across-groups" => true
   }
   ```

2. アプリケーションのデプロイ時に保存されたロールアウト計画名を指定します。
   以下の管理 CLI コマンドは、保存されたロールアウト計画 **my-rollout-plan** を使用してすべてのサーバーグループにアプリケーションをデプロイします。

   ```
   deploy /path/to/test-application.war --all-server-groups --headers={rollout id=my-rollout-plan}
   ```

保存されたロールアウト計画の削除
削除するロールアウト計画の名前を指定して **rollout-plan** 管理 CLI コマンドを使用すると、保存されたロールアウト計画を削除できます。

```
rollout-plan remove --name=my-rollout-plan
```

デフォルトのロールアウト計画
複数のサーバーに影響する操作はすべてロールアウト計画を使用して実行されます。操作リクエストにロールアウト計画が指定されていない場合は、デフォルトのロールアウト計画が生成されます。計画には以下の特徴があります。

- ハイレベルなフェーズは 1 つのみです。操作に影響するすべてのサーバーグループには同時に操作が適用されます。
- 各サーバーグループ内では、操作がすべてのサーバーに同時に適用されます。
- サーバーグループのいずれかのサーバーに操作を適用できないと、グループ全体でロールバックが実行されます。
- あるサーバーグループに操作を適用できないと、他のすべてのサーバーグループでロールバックが実行されます。

7.7. デプロイメント形式のデプロイメントの管理

JBoss EAP 7.1 以前では、ファイルシステム上のファイルを操作するのがデプロイメント形式のデプロイメントを管理する唯一の方法でした。JBoss EAP 7.1 より、管理インターフェイスを使用してデプロイメント形式のデプロイメントを管理できるようになりました。これにより、新しいバージョンのアプリケーションをデプロイしなくても、デプロイメント形式のアプリケーションのコンテンツを変更できるようになりました。

注記

JavaScript や CSS ファイルなど、デプロイメントの静的ファイルが更新されると、即座に更新が反映されます。Java クラスなどの他のファイルが変更された場合は、変更の反映にアプリケーションの再デプロイが必要になることがあります。

最初に、空のデプロイメント を使用するか、既存のアーカイブデプロイメントをデプロイメント した後、コンテンツを追加または削除 します。

デプロイメントのコンテンツの表示 を参照してデプロイメントのファイルを閲覧するか、ファイルのコンテンツを読み取ります。

空のデプロイメント形式のデプロイメントを作成

必要時にコンテンツを追加できる空のデプロイメント形式のデプロイメントを作成できます。以下の管理 CLI コマンドを使用して空のデプロイメント形式のデプロイメントを作成します。

```
/deployment=DEPLOYMENT_NAME.war:add(content=[{empty=true}])
```

empty=true オプションは、空のデプロイメントの作成を確認するために必要です。

既存のアーカイブデプロイメントのデプロイメント

既存のアーカイブデプロイメントをデプロイメントしてコンテンツを更新できます。デプロイメントする前に デプロイメントを無効化 する必要があります。以下の管理 CLI コマンドを使用してデプロイメントをデプロイメントします。

```
/deployment=ARCHIVE_DEPLOYMENT_NAME.ear:explode
```

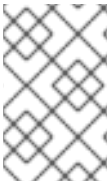
これで、このデプロイメントの コンテンツを追加または削除 できるようになります。

注記

管理コンソールから既存のアーカイブデプロイメントをデプロイメントすることもできます。**Deployments** タブからデプロイメントを選択し、ドロップダウンメニューで **Explode** を選択します。

デプロイメント形式のデプロイメントへのコンテンツの追加
デプロイメントにコンテンツを追加するには、**add-content** 管理 CLI 操作を使用します。コンテンツを追加するデプロイメントの場所へのパスを指定し、アップロードするコンテンツを提供します。アップロードするコンテンツは、ローカルファイルストリーム、URL、JBoss EAP コンテンツリポジトリーにすでに存在するコンテンツのハッシュ、またはコンテンツのバイトアレイとして提供できます。以下の管理 CLI コマンドは、**input-stream-index** オプションを使用してローカルファイルのコンテンツをデプロイメントにアップロードします。

```
/deployment=DEPLOYMENT_NAME.war:add-content(content=[{target-
path=/path/to/FILE_IN_DEPLOYMENT, input-stream-index=/path/to/LOCAL_FILE_TO_UPLOAD}]
```



注記

add-content 操作を使用してコンテンツをデプロイメントに追加するとき、デプロイメントのコンテンツはデフォルトで上書きされます。この挙動を変更するには、**overwrite** オプションを **false** に設定します。

デプロイメント形式のデプロイメントのコンテンツの削除
デプロイメントからコンテンツを削除するには、**remove-content** 管理 CLI 操作を使用し、デプロイメントの削除するコンテンツへのパスを指定します。

```
/deployment=DEPLOYMENT_NAME.war:remove-content(paths=[/path/to/FILE_1, /path/to/FILE_2])
```

7.8. デプロイメントのコンテンツの表示

JBoss EAP の管理インターフェイスを使用すると、管理されたデプロイメントのファイルに関する 情報を閲覧 し、ファイルの 内容を読み取る ことができます。

7.8.1. デプロイメントのファイルの閲覧

管理されたデプロイメントのファイルやディレクトリーを閲覧するには、**browse-content** 操作を使用します。引数を指定しないと、デプロイメント構造全体が返されます。 特定のディレクトリーへのパスを指定するには、**path** 引数を使用します。

注記

また、管理コンソールからデプロイメントの内容を閲覧することもできます。**Deployments** タブに移動してデプロイメントを選択し、ドロップダウンメニューで 表示 を選択します。

```
/deployment=helloworld.war:browse-content(path=META-INF/)
```

上記は、**helloworld.war** デプロイメントの **META-INF/** ディレクトリーにあるファイルとディレクトリーを表示します。

```
{
    "outcome" => "success",
    "result" => [
        {
            "path" => "MANIFEST.MF",
            "directory" => false,
            "file-size" => 827L
        },
        {
            "path" => "maven/org.jboss.eap.quickstarts/helloworld/pom.properties",
            "directory" => false,
            "file-size" => 106L
        },
        {
            "path" => "maven/org.jboss.eap.quickstarts/helloworld/pom.xml",
            "directory" => false,
            "file-size" => 2713L
        },
        {
            "path" => "maven/org.jboss.eap.quickstarts/helloworld/",
            "directory" => true
        },
        {
            "path" => "maven/org.jboss.eap.quickstarts/",
            "directory" => true
        },
        {
            "path" => "maven/",
            "directory" => true
        },
        {
            "path" => "INDEX.LIST",
            "directory" => false,
            "file-size" => 251L
        }
    ]
}
```

以下の引数を **browse-content** 操作に指定することもできます。

archive

アーカイブファイルのみを返すかどうか。

depth

返すファイルの深さを指定します。

7.8.2. デプロイメントコンテンツの読み取り

管理されたデプロイメントでファイルの内容を読み取るには、**read-content** 操作を使用します。引数を指定しないと、デプロイメント全体が返されます。または、**path** 引数を指定して、特定のファイルへのパスを提供します。以下に例を示します。

```
/deployment=helloworld.war:read-content(path=META-INF/MANIFEST.MF)
```

上記は、管理 CLI に表示 または ファイルシステムに保存 できるファイルストリームを返します。

```
{
    "outcome" => "success",
    "result" => {"uuid" => "24ba8e06-21bd-4505-b4d4-bdfb16451b95"},
    "response-headers" => {"attached-streams" => [{
        "uuid" => "24ba8e06-21bd-4505-b4d4-bdfb16451b95",
        "mime-type" => "text/plain"
    }]}
}
```

7.8.2.1. ファイルの内容の表示

attachment display コマンドを使用して **MANIFEST.MF** ファイルの内容を読み取ります。

```
attachment display --operation=/deployment=helloworld.war:read-content(path=META-
INF/MANIFEST.MF)
```

上記は、**helloworld.war** デプロイメントからの **MANIFEST.MF** ファイルの内容を管理 CLI に表示します。

```
ATTACHMENT 8af87836-2abd-423a-8e44-e731cc57bd80:
Manifest-Version: 1.0
Implementation-Title: Quickstart: helloworld
Implementation-Version: 7.4.0.GA
Java-Version: 1.8.0_131
Built-By: username
Scm-Connection: scm:git:git@github.com:jboss/jboss-parent-pom.git/quic
 kstart-parent/helloworld
Specification-Vendor: JBoss by Red Hat
...
```

7.8.2.2. ファイルの内容の保存

attachment save コマンドを使用して、**MANIFEST.MF** ファイルの内容をファイルシステムに保存します。

```
attachment save --operation=/deployment=helloworld.war:read-content(path=META-
INF/MANIFEST.MF) --file=/path/to/MANIFEST.MF
```

上記は、**helloworld.war** デプロイメントからの **MANIFEST.MF** ファイルを **path/to/MANIFEST.MF** のファイルシステムに保存します。**--file** 引数を使用してファイルパスを指定しないと、一意な添付 ID を使用してファイル名が付けられ、管理 CLI の作業ディレクトリー (デフォルトは **EAP_HOME/bin/**) に保存されます。

第8章 ドメイン管理

本項では、マネージドドメインの操作モードに固有する概念や設定について説明します。

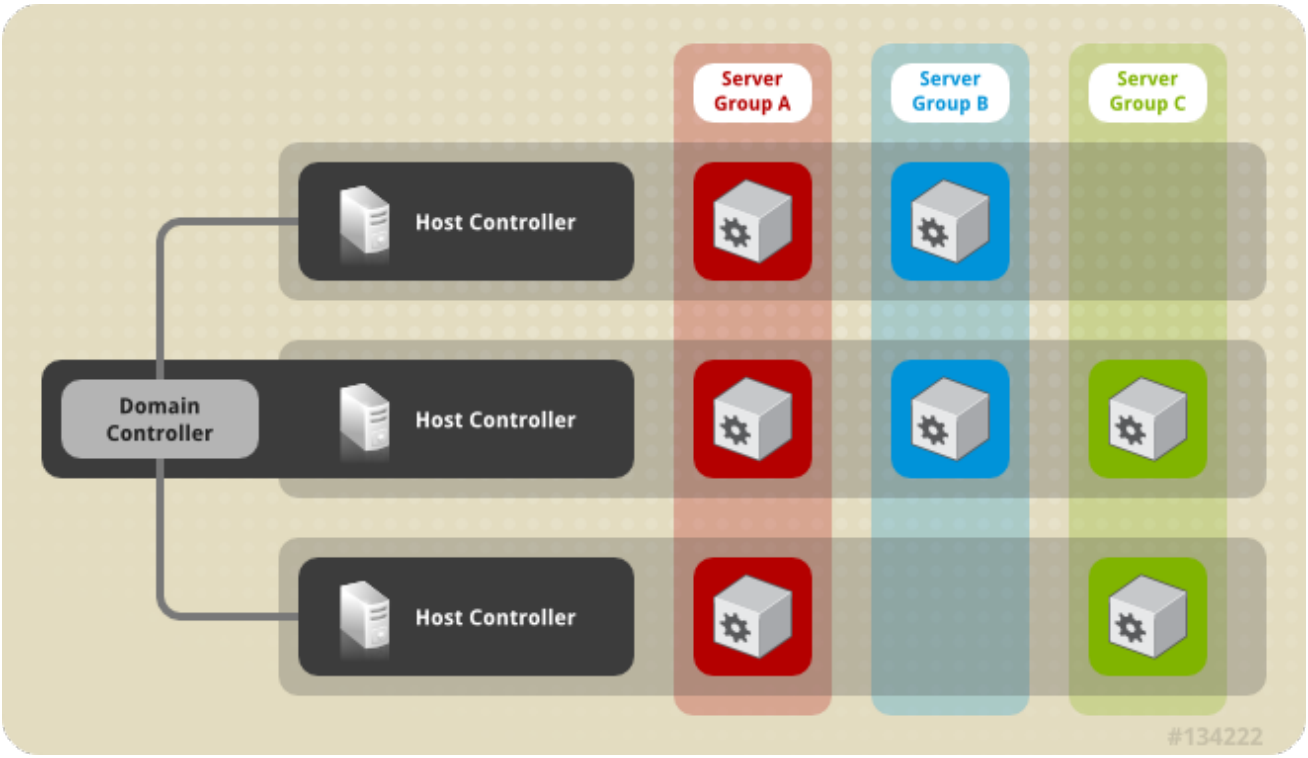
マネージドドメインのセキュア化については、JBoss EAPHow to Configure Server Security のSecuring a Managed Domainの項を参照してください。

8.1. マネージドドメイン

マネージドドメインの操作モードを使用すると、単一の制御ポイントから複数の JBoss EAP インスタンスを管理できます。

一元管理された複数の JBoss EAP サーバーは、ドメインのメンバーと呼ばれます。ドメインの JBoss EAP インスタンスは共通の管理ポリシーを共有します。

各ドメインは1つのドメインコントローラー、1つ以上のホストコントローラー、およびホスト毎に 0 個以上のサーバーグループによって設定されます。



ドメインコントローラー は、ドメインが制御される中心点であり、各サーバーはドメインの管理ポリシーに従って設定されます。ドメインの管理ポリシーに従って各サーバーが設定されるようにします。ドメインコントローラーはホストコントローラーでもあります。

ホストコントローラー はドメインコントローラーと対話して、ホスト上で実行されているアプリケーションサーバーインスタンスのライフサイクルを制御し、ドメインコントローラーがこれらのインスタンスを管理できるようにします。各ホストに複数のサーバーグループが含まれるようにすることが可能です。

サーバーグループ は、JBoss EAP がインストールされている サーバー インスタンスの集まりで、すべてが1つとして管理および設定されます。ドメインコントローラーはサーバーグループにデプロイされたアプリケーションとその設定を管理します。そのため、サーバーグループの各サーバーは同じ設定とデプロイメントを共有します。

ホストコントローラーは特定の物理または仮想ホストに割り当てられます。異なる設定を使用する場合は、同じハードウェア上で複数のホストコントローラーを実行でき、ポートとその他のリソースが競合

しないようにします。1つのドメインコントローラー、1つのホストコントローラー、および複数のサーバーを、同じ物理システムの同じ JBoss EAP インスタンス内で実行することができます。

8.1.1. ドメインコントローラー

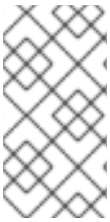
ドメインコントローラーは、ドメインの集中管理点として動作する JBoss EAP サーバーインスタンスです。1つのホストコントローラーインスタンスがドメインコントローラーとして動作するよう設定されます。

ドメインコントローラーの主なロールは次のとおりです。

- ドメインの集中管理ポリシーを維持する。
- すべてのホストコントローラーが現在のコンテンツを認識するようにする。
- 実行中のすべての JBoss EAP サーバーインスタンスがこのポリシーに従って設定されるよう、ホストコントローラーをサポートする。

デフォルトでは、集中管理ポリシーは **EAP_HOME/domain/configuration/domain.xml** ファイルに格納されます。このファイルは、ドメインコントローラーとして実行するよう設定されたホストコントローラーのこのディレクトリーに必要です。

domain.xml ファイルには、ドメインのサーバーに適用できるプロファイル設定が含まれています。プロファイル には、そのプロファイルで使用できるさまざまなサブシステムの詳細設定が含まれています。ドメイン設定には、ソケットグループの定義とサーバーグループの定義も含まれます。



注記

ホストとサーバーが JBoss EAP 6.2 以降を実行している場合、JBoss EAP 7 ドメインコントローラーは JBoss EAP 6 ホストとサーバーを管理できます。詳細は、JBoss EAP 6 インスタンスを管理するよう JBoss EAP 7.x ドメインコントローラーを設定 を参照してください

詳細は マネージドドメインの起動 および ドメインコントローラーの設定 の項を参照してください。

8.1.2. ホストコントローラー

ホストコントローラーの主なロールはサーバーを管理することです。ドメイン管理タスクを委譲し、ホスト上で実行される個別のアプリケーションサーバープロセスを開始および停止します。

ホストコントローラーはドメインコントローラーと対話し、サーバーとドメインコントローラー間の通信を管理できるようにします。ドメインの複数のホストコントローラーは1つのドメインコントローラーのみと対話できます。そのため、単一のドメインモード上で実行されるホストコントローラーおよびサーバーインスタンスはすべて単一のドメインコントローラーを持ち、同じドメインに属する必要があります。

デフォルトでは、各ホストコントローラーは、ホストのファイルシステムの未デプロイメントの JBoss EAP インストールファイルにある **EAP_HOME/domain/configuration/host.xml** ファイルから設定を読み取ります。**host.xml** ファイルには、特定のホストに固有する以下の設定情報が含まれています。

- このインストールから実行されるサーバーインスタンスの名前。
- ローカルの物理インストールに固有する設定。たとえば、**domain.xml** に宣言された名前付きインターフェイスの定義は **host.xml** にある実際のマシン固有の IP アドレスへマップできます。さらに、domain.xml の抽象パス名を **host.xml** のファイルシステムパスへマップできま

す。

- 次の設定のいずれか。
  - ホストコントローラーがドメインコントローラーへ通知して、ホストコントローラー自体を登録し、ドメイン設定へアクセスする方法。
  - リモートドメインコントローラーの検索および通知方法。
  - ホストコントローラーがドメインコントローラーとして動作するかどうか。

詳細は マネージドドメインの起動 および ホストコントローラーの設定 の項を参照してください。

8.1.3. プロセスコントローラー

プロセスコントローラーは小型のライトウェイトなプロセスで、ホストコントローラープロセスを起動し、そのライフサイクルを監視します。ホストコントローラーがクラッシュすると、プロセスコントローラーによって再起動されます。さらに、ホストコントローラーの指示に従ってサーバープロセスを開始しますが、クラッシュしたサーバープロセスは自動的に再起動しません。

プロセスコントローラーのログは **EAP_HOME/domain/log/process-controller.log** ファイルに記録されます。プロセスコントローラーの JVM オプションは、**PROCESS_CONTROLLER_JAVA_OPTS** 変数を使用して **EAP_HOME/bin/domain.conf** ファイルに設定します。

8.1.4. サーバーグループ

サーバーグループとは、1 つのグループとして管理および設定される複数のサーバーインスタンスのことです。マネージドドメインでは、各アプリケーションサーバーインスタンスは唯一のメンバーである場合でも 1 つのサーバーグループに属します。グループのサーバーインスタンスは同じプロファイル設定とデプロイされたコンテンツを共有します。

ドメインコントローラーとホストコントローラーは、ドメインにある各サーバーグループのすべてのインスタンスに標準設定を強制します。

ドメインを複数のサーバーグループで設定できます。異なるサーバーグループを異なるプロファイルやデプロイメントで設定できます。たとえば、ドメインは異なるサービスを提供する異なるサーバー層で設定できます。

異なるサーバーグループが同じプロファイルやデプロイメントを持つこともできます。これにより、最初のサーバーグループでアプリケーションがアップグレードされた後に 2 つ目のサーバーグループでアプリケーションが更新されるアプリケーションのローリングアップグレードが可能になり、サービスの完全停止を防ぎます。

詳細は、サーバーグループの設定 の項を参照してください。

8.1.5. サーバー

サーバーはアプリケーションサーバーインスタンスを表します。マネージドドメインでは、すべてのサーバーインスタンスがサーバーグループのメンバーになります。ホストコントローラーは各サーバーインスタンスを独自の JVM プロセスで起動します。

詳細は サーバーの設定 の項を参照してください。

8.2. ドメイン設定のナビゲート

JBoss EAP は、規模の小さいマネージドドメインと規模の大きいマネージドドメインの両方をサポートするスケーラブルな管理インターフェイスを提供します。

管理コンソール

JBoss EAP 管理コンソールは、ドメインのグラフィカルビューを提供し、ドメインのホスト、サーバー、デプロイメント、およびプロファイルの管理を容易にします。

Configuration (設定)

Configuration タブからドメインで使用される各プロファイルのサブシステムを設定できます。必要な機能に応じて、メインの異なるサーバーグループが異なるプロファイルを使用することがあります。
希望のプロファイルを選択すると、そのプロファイルで利用できるサブシステムがすべて表示されます。プロファイルの設定に関する詳細は JBoss EAP プロファイルの管理 を参照してください。

Runtime (ランタイム)

Runtime タブから、サーバー、サーバーグループ、およびホストの設定を管理できます。ホストまたはサーバーグループごとにドメインを閲覧できます。
Hosts から、ホストプロパティーと JVM を設定でき、そのホストのサーバーを追加および設定することもできます。

Server Groups から、新しいサーバーグループを追加でき、プロパティーや JVM を設定できます。また、そのサーバーグループにサーバーを追加および設定することもできます。 選択したサーバーグループの全サーバーの起動、停止、一時停止、およびリロードなど、操作を実行できます。

Hosts または **Server Groups** から、新しいサーバーを追加でき、サーバープロパティーや JVM を設定できます。選択したサーバーの起動、停止、一時停止、およびリロードなどの操作を実行できます。また、JVM の使用率、サーバーログ、サブシステム固有の情報など、ランタイム情報を表示することもできます。

Topology から、ドメインのホスト、サーバーグループ、およびサーバーの概要や詳細情報を表示できます。再ロードや停止などの操作をそれぞれに実行できます。

Deployments (デプロイメント)

Deployments タブから、デプロイメントをサーバーグループに追加およびデプロイできます。コンテンツリポジトリーのすべてのデプロイメントや、特定のサーバーグループにデプロイされたデプロイメントを表示できます。
管理コンソールを使用したアプリケーションのデプロイに関する詳細は マネージドドメインでのアプリケーションのデプロイ を参照してください。

管理 CLI

JBoss EAP 管理 CLI は、ドメインのホスト、サーバー、デプロイメントおよびプロファイルを管理するコマンドラインインターフェイスを提供します。

適切なプロファイルを選択するとサブシステムの設定にアクセスできます。

```
/profile=PROFILE_NAME/subsystem=SUBSYSTEM_NAME:read-resource(recursive=true)
```

注記

本ガイドの手順や例では、スタンドアロンサーバーとして稼働している場合に適用されるサブシステム設定の管理 CLI コマンドが含まれている可能性があります。例を以下に示します。

```
/subsystem=datasources/data-source=ExampleDS:read-resource
```

マネージドドメインで実行するために管理 CLI コマンドを調整するには、設定するプロファイルを指定する必要があります。例を以下に示します。

```
/profile=default/subsystem=datasources/data-source=ExampleDS:read-resource
```

適切なホストの指定後、ホストを設定し、そのホストのサーバーで操作を実行することができます。

```
/host=HOST_NAME/server=SERVER_NAME:read-resource
```

適切なホストの指定後、そのホストのサーバーを設定できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:write-
attribute(name=ATTRIBUTE_NAME,value=VALUE)
```

適切なサーバーグループの指定後、サーバーグループを設定し、選択したサーバーグループのすべてのサーバーで操作を実行することができます。

```
/server-group=SERVER_GROUP_NAME:read-resource
```

deploy 管理 CLI コマンドを使用し、適切なサーバーグループを指定すると、マネージドドメインでアプリケーションをデプロイできます。手順は、マネージドドメインでのアプリケーションのデプロイ を参照してください。

8.3. マネージドドメインの起動

8.3.1. マネージドドメインの起動

ドメインおよびホストコントローラーは、JBoss EAP に同梱される **domain.sh** または **domain.bat** スクリプトを使用して起動できます。使用できる起動スクリプトの引数の完全リストとそれら引数の目的については、**--help** 引数を使用するか、サーバーランタイムの引数およびスイッチ のセクションを参照してください。

ドメイン内のサーバーグループのスレーブサーバーを起動する前にドメインコントローラーを起動する必要があります。最初にドメインコントローラーを起動した後、ドメイン内の関連する他のホストコントローラーを起動します。

ドメインコントローラーの起動
専用のドメインコントローラー向けに事前設定された **host-master.xml** 設定ファイルを使用してドメインコントローラーを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-master.xml
```

ドメインの設定に応じて、ホストコントローラーの接続を可能にするために設定を追加する必要があります。また、以下のドメイン設定例を参照してください。

- 1 台のマシンでマネージドドメインを設定
- 2 台のマシンでマネージドドメインを設定

ホストコントローラーの起動
スレーブホストコントローラー向けに事前設定された **host-slave.xml** 設定ファイルを使用してホストコントローラーを起動します。

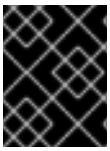
```
$ EAP_HOME/bin/domain.sh  --host-config=host-slave.xml
```

ドメインの設定に応じて、ドメインコントローラーへ接続し、ドメインコントローラーとの競合を回避するために設定を追加する必要があります。また、以下のドメイン設定例を参照してください。

- 1 台のマシンでマネージドドメインを設定
- 2 台のマシンでマネージドドメインを設定

8.3.2. ドメインコントローラーの設定

ドメイン内のホストをドメインコントローラーとして設定する必要があります。



重要

RPM インストールで JBoss EAP をインストールした場合、複数のドメインまたはホストコントローラーを同じマシン上に設定することはサポートされません。

<domain-controller> 宣言に **<local/>** 要素を追加して、ホストをドメインコントローラーとして設定します。**<domain-controller>** には他のコンテンツを含めないでください。

```
<domain-controller>
  <local/>
</domain-controller>
```

ドメインコントローラーとして動作するホストは、ドメインの他のホストがアクセスできる管理インターフェイスを公開する必要があります。HTTP インターフェイスは標準の管理インターフェイスです。

```
<management-interfaces>
  <http-interface security-realm="ManagementRealm" http-upgrade-enabled="true">
    <socket interface="management" port="${jboss.management.http.port:9990}"/>
  </http-interface>
</management-interfaces>
```

最小のドメインコントローラー設定ファイルの例 (**EAP_HOME/domain/configuration/host-master.xml**) には、以下の設定が含まれます。

8.3.3. ホストコントローラーの設定

ホストコントローラー自体をドメインに登録するようにホストコントローラーを設定する必要があります。

重要

RPM インストールで JBoss EAP をインストールした場合、複数のドメインまたはホストコントローラーを同じマシン上に設定することはサポートされません。

設定の **<domain-controller>** 要素を使用して、ドメインコントローラーへの接続を設定します。

```
<domain-controller>
  <remote security-realm="ManagementRealm">
    <discovery-options>
      <static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}" port="${jboss.domain.master.port:9990}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

最小のホストコントローラー設定ファイルのサンプル **EAP_HOME/domain/configuration/host-slave.xml** には、ドメインコントローラーに接続する設定が含まれています。この設定は、ホストコントローラーの起動時に **jboss.domain.master.address** プロパティーを指定するものと想定しています。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -
Djboss.domain.master.address=IP_ADDRESS
```

ドメインコントローラーの検出に関する詳細は、ドメインコントローラーの検出およびフェイルオーバー の項を参照してください。

ドメインの設定によっては、認証を提供して、ホストコントローラーがドメインコントローラーによって認証されるようにする必要がある場合があります。管理ユーザーと秘密の値の生成や、その値を用いたホストコントローラー設定の更新に関する詳細は、2 台のマシンでマネージドドメインを設定 を参照してください。

8.3.4. ホスト名の設定

マネージドドメインで実行されている各ホストには一意な名前を付ける必要があります。管理を容易にし、複数のホストで同じホスト設定ファイルを使用できるようにするために、以下の優先順位を用いてホスト名が決定されます。

1. 設定されている場合、**host.xml** 設定ファイルのホスト要素名属性。
2. **jboss.host.name** システムプロパティーの値。
3. **jboss.qualified.host.name** システムプロパティーの最後のピリオド (.) の後に続く値。 最後のピリオド (.) がない場合は全体の値。
4. POSIX ベースのオペレーティングシステムでは、**HOSTNAME** 環境変数のピリオド (.) の後に続く値。 Microsoft Windows では **COMPUTERNAME** 環境変数のピリオド (.) の後に続く値。最後のピリオドがない場合は、全体の値。

関連する **host.xml** 設定ファイルの上部にある **host** 要素に設定されたホストコントローラーの名前。例は次のとおり。

```
<host xmlns="urn:jboss:domain:8.0" name="host1">
```

以下の手順に従って、管理 CLI を使用してホスト名を更新します。

1. JBoss EAP ホストコントローラーを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml
```

2. 管理 CLI を起動し、ドメインコントローラーに接続します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --
controller=DOMAIN_CONTROLLER_IP_ADDRESS
```

3. 以下のコマンドを実行して新しいホスト名を設定します。

```
/host=EXISTING_HOST_NAME:write-attribute(name=name,value=NEW_HOST_NAME)
```

これにより、**host-slave.xml** ファイルのホスト名属性が以下のように変更されます。

```
<host name="NEW_HOST_NAME" xmlns="urn:jboss:domain:8.0">
```

4. 変更を反映するためにホストコントローラーをリロードします。

```
reload --host=EXISTING_HOST_NAME
```

ホストコントローラーの名前が設定ファイルに設定されていない場合は、起動時にホスト名を渡すこともできます。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml  -Djboss.host.name=HOST_NAME
```

8.4. サーバーの管理

8.4.1. サーバーグループの設定

以下はサーバーグループ定義の例になります。

```
<server-group name="main-server-group" profile="full">
  <jvm name="default">
    <heap size="64m" max-size="512m"/>
  </jvm>
  <socket-binding-group ref="full-sockets"/>
  <deployments>
    <deployment name="test-application.war" runtime-name="test-application.war"/>
    <deployment name="helloworld.war" runtime-name="helloworld.war" enabled="false"/>
  </deployments>
</server-group>
```

サーバーグループの設定は、管理 CLI を使用するか、管理コンソールの **Runtime** タブから行います。

サーバーグループの追加
以下の管理 CLI コマンドを実行すると、サーバーグループを追加できます。

```
/server-group=SERVER_GROUP_NAME:add(profile=PROFILE_NAME,socket-binding-
group=SOCKET_BINDING_GROUP_NAME)
```

サーバーグループの更新
以下の管理 CLI コマンドを実行すると、サーバーグループ属性を更新できます。

```
/server-group=SERVER_GROUP_NAME:write-attribute(name=ATTRIBUTE_NAME,value=VALUE)
```

サーバーグループの削除
以下の管理 CLI コマンドを実行すると、サーバーグループを削除できます。

```
/server-group=SERVER_GROUP_NAME:remove
```

サーバーグループ属性
サーバーグループには次の属性が必要です。

- **name**: サーバーグループの名前。
- **profile**: サーバーグループプロファイル名。
- **socket-binding-group**: グループのサーバーに使用されるデフォルトのソケットバインディンググループ。サーバーごとに上書きできます。

サーバーグループに含まれる任意の属性は次のとおりです。

- **management-subsystem-endpoint**: **remoting** サブシステムからエンドポイントを使用してサーバーグループに属するサーバーを再びホストコントローラーに接続するには **true** に設定します。これは、**remoting** サブシステムが存在しないと機能しません。
- **socket-binding-default-interface**: このサーバーのソケットバインディンググループのデフォルトインターフェイス。
- **socket-binding-port-offset**: ソケットバインディンググループによって提供されたポート値に追加するデフォルトのオフセット。
- **deployments**: グループのサーバーにデプロイするデプロイメントコンテンツ。
- **jvm**: グループの全サーバーに対するデフォルトの JVM 設定。ホストコントローラーはこれらの設定を **host.xml** の他の設定とマージし、サーバーの JVM を開始するために使用される設定を作成します。
- **deployment-overlays**: 定義されたデプロイメントオーバーレイと、このサーバーグループのデプロイメントとの間をリンクします。
- **system-properties**: グループのサーバーに設定するシステムプロパティー。

8.4.2. サーバーの設定

デフォルトの **host.xml** 設定ファイルは 3 つのサーバーを定義します。

```
<servers>
  <server name="server-one" group="main-server-group">
  </server>
  <server name="server-two" group="main-server-group" auto-start="true">
    <socket-bindings port-offset="150"/>
  </server>
  <server name="server-three" group="other-server-group" auto-start="false">
```

```
    <socket-bindings port-offset="250"/>
  </server>
</servers>
```

server-one という名前のサーバーインスタンスは **main-server-group** に関連付けられ、そのサーバーグループによって指定されるサブシステム設定とソケットバインディングを継承します。**server-two** という名前のサーバーインスタンスも **main-server-group** に関連付けられていますが、**server-one** によって使用されるポートの値と競合しないようにソケットバインディングの **port-offset** の値も定義します。**server-three** という名前のサーバーインスタンスは **other-server-group** に関連付けられ、そのグループの設定を使用します。また、 **port-offset** の値も定義し、ホストコントローラーの起動時にこのサーバーが起動しないように **auto-start** を **false** に設定します。

サーバーの設定は、管理 CLI を使用するか、管理コンソールの **Runtime** タブから行います。

サーバーの追加
以下の管理 CLI コマンドを実行すると、サーバーを追加できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:add(group=SERVER_GROUP_NAME)
```

サーバーの更新
以下の管理 CLI コマンドを実行すると、サーバー属性を更新できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:write-
attribute(name=ATTRIBUTE_NAME,value=VALUE)
```

サーバーの削除
以下の管理 CLI コマンドを実行すると、サーバーを削除できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:remove
```

サーバーの属性
サーバーには以下の属性が必要です。

- **name**: サーバーの名前。
- **group**: ドメインモデルからのサーバーグループの名前。

サーバーには以下の任意の属性が含まれます。

- **auto-start**: ホストコントローラーの起動時にこのサーバーが起動されるかどうか。
- **socket-binding-group**: このサーバーが属するソケットバインディンググループ。
- **socket-binding-port-offset**: このサーバーのソケットバイディンググループによって提供されたポート値に追加されるオフセット。
- **update-auto-start-with-server-status**: サーバーの状態で **auto-start** 属性を更新します。
- **interface**: サーバーで使用できる完全に指定された名前付きのネットワークインターフェイスのリスト。
- **jvm**: このサーバーの JVM 設定。宣言されていない場合、設定は親のサーバーグループまたはホストから継承されます。
- **path**: 名前付きのファイルシステムパスのリスト。

- **system-property**: このサーバーに設定するシステムプロパティーのリスト。

8.4.3. サーバーの起動と停止

管理コンソールから起動、停止、リロードなどの操作をサーバーで実行するには、**Runtime** タブに移動して適切なホストまたはサーバーグループを選択します。

管理 CLI を使用してこれらの操作を実行する場合は以下のコマンドを参照してください。

サーバーの起動
特定のホストで 1 台のサーバーを起動できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:start
```

指定のサーバーグループのサーバーをすべて起動できます。

```
/server-group=SERVER_GROUP_NAME:start-servers
```

サーバーの停止
特定のホストで 1 台のサーバーを停止できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:stop
```

指定のサーバーグループのサーバーをすべて停止できます。

```
/server-group=SERVER_GROUP_NAME:stop-servers
```

サーバーのリロード
特定のホストで 1 台のサーバーをリロードできます。

```
/host=HOST_NAME/server-config=SERVER_NAME:reload
```

指定のサーバーグループのサーバーをすべてリロードできます。

```
/server-group=SERVER_GROUP_NAME:reload-servers
```

サーバーの Kill
指定のサーバーグループのすべてのサーバープロセスを kill することができます。

```
/server-group=SERVER_GROUP_NAME:kill-servers
```

8.5. ドメインコントローラーの検出およびフェイルオーバー

マネージドドメインの設定時、ドメインコントローラーに接続するために必要な情報を使用して各ホストコントローラーを設定する必要があります。JBoss EAP では、ドメインコントローラーを検索する複数のオプション を使用して各ホストコントローラーを設定できます。ホストコントローラーは、適切なオプションが見つかるまでオプションのリストを繰り返し処理します。

プライマリードメインコントローラーに問題がある場合は、バックアップホストコントローラーを ドメインコントローラーに昇格 できます。これにより、昇格後にホストコントローラーを新しいドメインコントローラーへ自動的にフェイルオーバーできます。

8.5.1. ドメイン検出オプションの設定

以下は、ドメインコントローラー検索の複数のオプションを持つホストコントローラーを設定する方法の例になります。

例: 複数のドメインコントローラーオプションを持つホストコントローラー

```
<domain-controller>
  <remote security-realm="ManagementRealm">
    <discovery-options>
      <static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"
host="172.16.81.100" port="${jboss.domain.master.port:9990}"/>
      <static-discovery name="backup" protocol="${jboss.domain.master.protocol:remote}"
host="172.16.81.101" port="${jboss.domain.master.port:9990}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

静的検出オプションには、以下の必須属性が含まれます。

name
: このドメインコントローラー検出オプションの名前。

host
: リモートドメインコントローラーのホスト名。

port
: リモートドメインコントローラーのポート。

上記の例では、最初の検出オプションが指定されることが予想されます。2 つ目のオプションはフフェイルオーバーの状況で使用される可能性があります。

8.5.2. キャッシュされたドメイン設定を用いたホストコントローラーの起動

--cached-dc オプションを使用すると、ドメインコントローラーへ接続していなくてもホストコントローラーを起動できますが、ホストコントローラーは前もってドメインコントローラーからドメイン設定をローカルでキャッシュする必要があります。**--cached-dc** オプションを使用してホストコントローラーを起動すると、ドメインコントローラーからホストコントローラーのドメイン設定がキャッシュされます。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml --cached-dc
```

これにより、このホストコントローラーがドメインコントローラーに接続せずに現在のサーバーを一時的に管理するために必要な情報が含まれる **EAP_HOME/domain/configuration/** ディレクトリーに **domain.cached-remote.xml** ファイルが作成されます。

注記

デフォルトでは、**--cached-dc** オプションを使用すると、このホストコントローラーによって使用される設定のみがキャッシュされます。そのため、ドメイン全体のドメインコントローラーには昇格されません。 ドメイン設定全体をキャッシュしてホストコントローラーをドメインコントローラーとする場合は、ドメイン設定のキャッシュ を参照してください。

--cached-dc を使用してこのホストコントローラーを起動したときにドメインコントローラーを利用できない場合、ホストコントローラーは **domain.cached-remote.xml** ファイルに保存されたキャッシュされた設定の使用を開始します。このファイルが存在しないとホストコントローラーは起動しないことに注意してください。

この状態の間、ホストコントローラーはドメイン設定を編集できませんが、サーバーを開始したり、デプロイメントを管理することはできます。

ホストコントローラーがキャッシュされた設定を使用して起動されると、ホストコントローラーは継続してドメインコントローラーへの再接続を試みます。ドメインコントローラーが使用できるようになると、ホストコントローラーは自動的にドメインコントローラーに再接続し、ドメイン設定を同期化します。変更された設定によっては、変更の反映のにホストコントローラーのリロードが必要になることがあります。この場合、警告がホストコントローラーのログに記録されます。

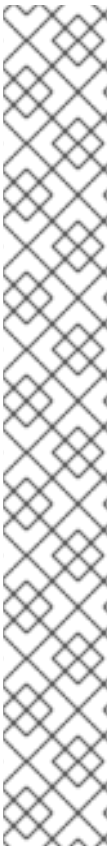
8.5.3. ホストコントローラーがドメインコントローラーとして動作するよう昇格

主要のドメインコントローラーに問題が発生した場合に、ホストコントローラーがドメインコントローラーとして動作するよう昇格できます。ホストコントローラーを 昇格 する前に、ドメインコントローラーから ドメイン設定をローカルでキャッシュ する必要があります。

ドメイン設定のキャッシュ
ドメインコントローラーとして使用したいホストコントローラーに **--backup** オプションを使用します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml --backup
```

これにより、ドメイン設定全体のコピーが含まれる **EAP_HOME/domain/configuration/** ディレクトリーに **domain.cached-remote.xml** ファイルが作成されます。この設定は、ホストコントローラーがドメインコントローラーとして動作するよう再設定された場合に使用されます。



注記

ignore-unused-configuration 属性は、特定のホストがキャッシュする設定の範囲を判断するために使用されます。**true** を指定すると、このホストコントローラーに関係する設定のみがキャッシュされ、ドメインコントローラーとして引き継ぐことはできません。**false** を指定すると、ドメイン設定全体がキャッシュされます。

The **--backup** 引数はデフォルトでこの属性の値を **false** とし、ドメイン全体をキャッシュします。しかし、この属性を **host.xml** ファイルで設定した場合は、その値が使用されます。

また、**--cached-dc** オプションのみを使用して、ドメイン設定のコピーを作成できますが、**host.xml** で **ignore-unused-configuration** の値を **false** と設定し、ドメイン全体をキャッシュする必要があります。以下に例を示します。

```
<domain-controller>
  <remote username="$local" security-realm="ManagementRealm" ignore-unused-
configuration="false">
    <discovery-options>
      ...
    </discovery-options>
  </remote>
</domain-controller>
```

ホストコントローラーをドメインコントローラーに昇格

1. 元のドメインコントローラーが停止した状態であることを確認します。
2. 管理 CLI を使用して、新しいドメインコントローラーとなるホストコントローラーへ接続します。
3. 以下のコマンドを実行してホストコントローラーを設定し、新しいドメインコントローラーとして動作するようにします。

```
/host=backup:write-attribute(name=domain-controller.local, value={})
```

4. 以下のコマンドを実行し、ホストコントローラーをリロードします。

```
reload --host=HOST_NAME
```

このホストコントローラーがドメインコントローラーとして動作するようになります。

8.6. マネージドドメインの設定

8.6.1. 1台のマシンでマネージドドメインを設定

jboss.domain.base.dir プロパティーを使用すると1台のマシンで複数のホストコントローラーを実行できます。

重要

複数の JBoss EAP ホストコントローラーを1台のマシン上でシステムサービスとして設定することはサポートされません。

1. ドメインコントローラーの **EAP_HOME**/**domain** ディレクトリーをコピーします。

```
$ cp -r EAP_HOME/domain /path/to/domain1
```

2. ホストコントローラーの **EAP_HOME**/**domain** ディレクトリーをコピーします。

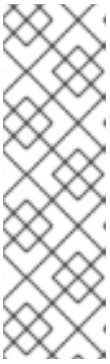
```
$ cp -r EAP_HOME/domain /path/to/host1
```

3. /**path**/**to**/**domain1** を使用してドメインコントローラーを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-master.xml -
Djboss.domain.base.dir=/path/to/domain1
```

4. /**path**/**to**/**host1** を使用してホストコントローラーを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -
Djboss.domain.base.dir=/path/to/host1 -Djboss.domain.master.address=IP_ADDRESS -
Djboss.management.http.port=PORT
```



注記

ホストコントローラーの起動時に、**jboss.domain.master.address** プロパティーを使用してドメインコントローラーのアドレスを指定する必要があります。

さらに、このホストコントローラーはドメインコントローラーと同じマシンで実行されているため、ドメインコントローラーの管理インターフェイスと競合しないように管理インターフェイスを変更する必要があります。このコマンドは **jboss.management.http.port** プロパティーを設定します。

このように起動された各インスタンスは、ベースインストールディレクトリー (例: **EAP_HOME/modules/**) のその他のリソースを共有しますが、**jboss.domain.base.dir** によって指定されたディレクトリーからドメイン設定を使用します。

8.6.2. 2 台のマシンでマネージドドメインを設定

注記

この例の実行には、ファイアウォールの設定が必要になることがあります。

1 台がドメインコントローラーでもう 1 台がホストである 2 台のマシンでマネージドドメインを作成できます。詳細は、ドメインコントローラー を参照してください。

- **IP1** = ドメインコントローラーの IP アドレス (マシン 1)
- **IP2** = ホストの IP アドレス (マシン 2)

2 台のマシンでマネージドドメインを作成

1. マシン 1 での作業
   a. ホストがドメインコントローラーによって認証されるよう、管理ユーザーを追加します。**add-user.sh** スクリプトを使用してホストコントローラー **HOST_NAME** の管理ユーザーを追加します。最後の質問で **yes** を指定し、提供される秘密の値を書き留めてください。この秘密の値はホストコントローラーの設定で使用され、以下のように表示されます。

   ```
   <secret value="SECRET_VALUE" />
   ```

   b. ドメインコントローラーを起動します。
   専用のドメインコントローラー用に事前設定された **host-master.xml** 設定ファイルを指定します。さらに、**jboss.bind.address.management** プロパティーを設定し、ドメインコントローラーが他のマシンにも表示されるようにします。

   ```
   $ EAP_HOME/bin/domain.sh --host-config=host-master.xml -
   Djboss.bind.address.management=IP1
   ```

2. マシン 2 での作業
   a. ユーザー認証情報でホスト設定を更新します。
   EAP_HOME/domain/configuration/host-slave.xml を編集し、ホスト名 **HOST_NAME** と秘密の値 **SECRET_VALUE** を設定します。

   ```
   <host xmlns="urn:jboss:domain:8.0" name="HOST_NAME">
   ```

```
<management>
  <security-realms>
    <security-realm name="ManagementRealm">
      <server-identities>
        <secret value="SECRET_VALUE" />
      </server-identities>
      ...
```

b. ホストコントローラーを起動します。
スレーブホストコントローラー用に事前設定された **host-slave.xml** 設定ファイルを指定します。さらに、**jboss.domain.master.address** プロパティーを設定してドメインコントローラーに接続し、**jboss.bind.address** プロパティーを設定してホストコントローラーバインドアドレスを設定します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml  -
Djboss.domain.master.address=IP1  -Djboss.bind.address=IP2
```

起動時に **--controller** パラメーターを使用してドメインコントローラーアドレスを指定し、管理 CLI からドメインを管理できるようになります。

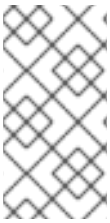
```
$ EAP_HOME/bin/jboss-cli.sh --connect --controller=IP1
```

また、**http://IP1:9990** で管理コンソールからドメインを管理することもできます。

8.7. MANAGING JBOSS EAP 7.4 バージョン

最新バージョンの JBoss EAP は、以前のバージョンの JBoss EAP を実行するサーバーおよびホストを管理できます。JBoss EAP の最新バージョンの管理については、以下のセクションを参照してください。

- JBoss EAP の以前のマイナーリリースを管理するよう JBoss EAP 7.4 ドメインコントローラーを設定

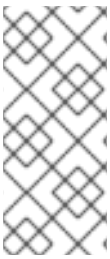


注記

Red Hat では、JBoss EAP 6.x を実行するホストとサーバーを管理する JBoss EAP 7.4 ドメインコントローラーを非推奨にしました。この非推奨のサポート機能は、JBoss EAP の次のメジャーリリースであるバージョン 8.0 で完全に削除されます。JBoss EAP 8.0 は、JBoss EAP 7.4 で実行されているホストとサーバーのみをサポートします。

8.7.1. JBoss EAP 6 インスタンスを管理するよう JBoss EAP 7.x ドメインコントローラーを設定

ホストとサーバーが JBoss EAP 6.2 以降を実行している場合、JBoss EAP 7.4 ドメインコントローラーは、JBoss EAP 6 を実行しているホストとサーバーを管理できます。



注記

JBoss EAP 7.0 ドメインコントローラーが異なるパッチリリースの JBoss EAP 7.0 ホストを管理する場合は、JBoss EAP 7.0 ドメインコントローラーは設定を変更する必要はありません。しかし、JBoss EAP 7.0 ドメインコントローラーは、管理するホストコントローラーと同じパッチリリースバージョンまたはそれ以降のバージョンを実行している必要があります。

JBoss EAP 7 の管理対象ドメインで JBoss EAP 6 インスタンスを正常に管理するには、以下の作業を行います。

1. Boss EAP 6 の設定を JBoss EAP 7 ドメインコントローラーに追加
2. JBoss EAP 6 プロファイルの動作の更新
3. JBoss EAP 6 サーバーのサーバーグループを設定します。
4. JBoss EAP 6 インスタンスが JBoss EAP 7 の更新を取得しないようにする

これらの作業が終了したら、管理 CLI を使用して JBoss EAP 7 ドメインコントローラーから JBoss EAP 6 サーバーおよび設定を管理できます。JBoss EAP 6 ホストは、バッチ処理などの JBoss EAP 7 の新機能を利用できないことに注意してください。

警告

管理コンソールは、最新バージョンの JBoss EAP に対して最適化されているため、管理コンソールを使用して JBoss EAP 6 のホスト、サーバー、およびプロファイルを更新しないでください。JBoss EAP 7 のマネージドドメインから JBoss EAP 6 の設定を管理する場合は、代わりに 管理 CLI を使用してください。

8.7.1.1. Boss EAP 6 の設定を JBoss EAP 7 ドメインコントローラーに追加

ドメインコントローラーが JBoss EAP 6 サーバーを管理できるようにするには、JBoss EAP 7 のドメイン設定に JBoss EAP 6 の設定詳細を提供する必要があります。これには、JBoss EAP 6 のプロファイル、ソケットバインディンググループ、およびサーバーグループを JBoss EAP 7 の **domain.xml** 設定ファイルにコピーします。

JBoss EAP 7 の設定と名前が競合する場合は、リソースの名前を変更する必要があります。さらに、正しく動作するようにするため、追加の 調整 を行う必要もあります。

以下の手順では、JBoss EAP 6 の **default** プロファイル、**standard-sockets** ソケットバインディンググループ、および **main-server-group** サーバーグループが使用されます。

1. JBoss EAP 7 の **domain.xml** 設定ファイルを編集します。このファイルをバックアップしてから編集することが推奨されます。
2. 該当する JBoss EAP 6 のプロファイルを JBoss EAP 7 の **domain.xml** ファイルにコピーします。
   この手順では、JBoss EAP 6 の **default** プロファイルをコピーし、名前を **eap6-default** に変更したことを仮定します。

   JBoss EAP 7 の domain.xml

```
<profiles>
  ...
  <profile name="eap6-default">
    ...
  </profile>
</profiles>
```

3. このプロファイルによって使用される必要なエクステンションを追加します。
JBoss EAP 6 のプロファイルが JBoss EAP 7 には存在しないサブシステムを使用する場合は、JBoss EAP ドメイン設定に適切なエクステンションを追加する必要があります。

JBoss EAP 7 の domain.xml

```
<extensions>
  ...
  <extension module="org.jboss.as.configadmin"/>
  <extension module="org.jboss.as.threads"/>
  <extension module="org.jboss.as.web"/>
<extensions>
```

4. 該当する JBoss EAP 6 のソケットバインディンググループを JBoss EAP 7 の **domain.xml** ファイルにコピーします。
この手順では、JBoss EAP 6 の **standard-sockets** ソケットバインディンググループをコピーし、名前を **eap6-standard-sockets** に変更したことを仮定します。

JBoss EAP 7 の domain.xml

```
<socket-binding-groups>
  ...
  <socket-binding-group name="eap6-standard-sockets" default-interface="public">
    ...
  </socket-binding-group>
</socket-binding-groups>
```

5. 該当する JBoss EAP 6 のサーバーグループを JBoss EAP 7 の **domain.xml** ファイルにコピーします。
この手順では、JBoss EAP 6 の **main-server-group** サーバーグループをコピーし、名前を **eap6-main-server-group** に変更したことを仮定します。また、このサーバーグループを更新して、JBoss EAP 6 のプロファイル **eap6-default** と JBoss EAP 6 のソケットバインディンググループ **eap6-standard-sockets** を使用するようにする必要があります。

JBoss EAP 7 の domain.xml

```
<server-groups>
  ...
  <server-group name="eap6-main-server-group" profile="eap6-default">
    ...
    <socket-binding-group ref="eap6-standard-sockets"/>
  </server-group>
</server-groups>
```

8.7.1.2. JBoss EAP 6 プロファイルの動作の更新

JBoss EAP のバージョンや必要な動作に応じて、JBoss EAP 6 インスタンスによって使用されるプロファイルを追加更新する必要があります。既存の JBoss EAP 6 インスタンスが使用するサブシステムや設定に応じて、追加の変更が必要になる場合があります。

JBoss EAP 7 ドメインコントローラーを起動して管理 CLI を起動し、以下の更新を実行します。これらの例では、JBoss EAP 6 のプロファイルが **eap6-default** であることを仮定します。

- **bean-validation** サブシステムを削除します。

JBoss EAP 7 では、bean バリデーション機能が **ee** サブシステムから独自の **bean-validation** サブシステムに移されました。JBoss EAP 7 ドメインコントローラーがレガシーの **ee** サブシステムを発見した場合、新しい **bean-validation** サブシステムを追加します。しかし、JBoss EAP 6 のホストはこのサブシステムを認識しないため、削除する必要があります。

JBoss EAP 7 のドメインコントローラー CLI

```
/profile=eap6-default/subsystem=bean-validation:remove
```

- CDI 1.0 の挙動を設定します。
これは、JBoss EAP 6 サーバーに JBoss EAP 7 で使用されるより新しいバージョンの CDI の挙動ではなく、CDI 1.0 の挙動を適用する場合のみ必要です。CDI 1.0 の挙動を提供する場合は、**weld** サブシステムに以下の更新を追加します。

JBoss EAP 7 のドメインコントローラー CLI

```
/profile=eap6-default/subsystem=weld:write-attribute(name=require-bean-descriptor,value=true)

/profile=eap6-default/subsystem=weld:write-attribute(name=non-portable-mode,value=true)
```

- JBoss EAP 6.2 のデータソース統計を有効にします。
これは、プロファイルが JBoss EAP 6.2 サーバーによって使用される場合のみ必要です。JBoss EAP 6.3 には **statistics-enabled** 属性が導入され、デフォルトでは統計を収集しない **false** に設定されます。しかし、JBoss EAP 6.2 では統計を収集しました。このプロファイルが JBoss EAP 6.2 のホストとそれ以降のバージョンの JBoss EAP を実行するホストによって使用される場合、ホストによって動作が異なることになり、これは許可されません。そのため、JBoss EAP 6.2 ホスト向けのプロファイルは、データソースに以下の変更を追加する必要があります。

JBoss EAP 7 のドメインコントローラー CLI

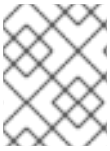
```
/profile=eap6-default/subsystem=datasources/data-source=ExampleDS:write-attribute(name=statistics-enabled,value=true)
```

8.7.1.3. JBoss EAP 6 サーバーのサーバーグループの設定

サーバーグループの名前を変更した場合は、JBoss EAP 6 のホスト設定を更新し、JBoss EAP 7 の設定に指定された新しいサーバーグループを使用する必要があります。この例では、JBoss EAP 7 の **domain.xml** に指定された **eap6-main-server-group** サーバーグループを使用します。

JBoss EAP 6 の host-slave.xml

```
<servers>
   <server name="server-one" group="eap6-main-server-group"/>
   <server name="server-two" group="eap6-main-server-group">
      <socket-bindings port-offset="150"/>
   </server>
</servers>
```



注記

ホストは、実行している JBoss EAP よりも新しいバージョンで導入された機能や設定を使用できません。

8.7.1.4. JBoss EAP 6 インスタンスが JBoss EAP 7 の更新を取得しないようにする

マネージドドメインのドメインコントローラーは、設定の更新をホストコントローラーに転送します。**host-exclude** 設定を使用して、特定のバージョンが認識できないようにするリソースを指定する必要があります。事前設定された **host-exclude** のオプションは **EAP62**、**EAP63**、**EAP64**、および **EAP64z** です。 ご使用の JBoss EAP 6 のバージョンに該当するものを選択します。

host-exclude 設定の **active-server-groups** 属性は、特定のバージョンによって使用されるサーバーグループのリストを指定します。指定のバージョンのホストは、指定されたサーバーグループとそれらに関連するプロファイル、ソケットバインディンググループ、およびデプロイメントリソースを利用できますが、それ以外は認識しません。

以下の例では、JBoss EAP のバージョンが 6.4.z であることを仮定し、JBoss EAP 6 のサーバーグループ **eap6-main-server-group** をアクティブなサーバーグループとして追加します。

JBoss EAP 7 のドメインコントローラー CLI

```
/host-exclude=EAP64z:write-attribute(name=active-server-groups,value=[eap6-main-server-group])
```

必要な場合は、**active-socket-binding-groups** 属性を使用して、サーバーによって使用される追加のソケットバインディンググループを指定します。これは、**active-server-groups** に指定されたサーバーグループと関連していないソケットバインディンググループのみに必要です。

8.7.2. JBoss EAP の以前のマイナーリリースを管理するよう JBoss EAP 7.4 ドメインコントローラーを設定

JBoss EAP 7.4 ドメインコントローラーは、以前の JBoss EAP のマイナーリリースからの実行中のホストおよびサーバーを管理できます。

JBoss EAP 7.4 の管理対象ドメインで JBoss EAP 7.3 インスタンスを正常に管理するには、以下の作業を行います。

1. Boss EAP 7.3 の設定を JBoss EAP 7.4 ドメインコントローラーに追加する 。
2. JBoss EAP 7.3 サーバーのサーバーグループを設定する。
3. JBoss EAP 7.3 インスタンスが JBoss EAP 7.4 の更新を取得しないようにする 。

これらの作業の完了後に、管理 CLI を使用して JBoss EAP 7.4 ドメインコントローラーから JBoss EAP 7.3 サーバーおよび設定を管理できます。

警告

管理コンソールは最新バージョンの JBoss EAP に対して最適化されているため、CLI を使用して JBoss EAP 7.4 のマネージドドメインから JBoss EAP 7.3 の設定を管理する必要があります。管理コンソールを使用して JBoss EAP 7.3 ホスト、サーバー、およびプロファイルを更新しないでください。

8.7.2.1. Boss EAP 7.3 の設定を JBoss EAP 7.4 ドメインコントローラーに追加

ドメインコントローラーが JBoss EAP 7.3 サーバーを管理できるようにするには、JBoss EAP 7.4 のドメイン設定に JBoss EAP 7.3 の設定詳細を提供する必要があります。これには、JBoss EAP 7.3 のプロファイル、ソケットバインディンググループ、およびサーバーグループを JBoss EAP 7.4 の **domain.xml** 設定ファイルにコピーします。

JBoss EAP 7.4 設定のリソース名と競合する場合は、リソースの名前を変更する必要があります。

以下の手順では、JBoss EAP 7.3 の **default** プロファイル、**standard-sockets** ソケットバインディンググループ、および **main-server-group** サーバーグループが使用されます。

前提条件

- JBoss EAP 7.3 の **default** プロファイルをコピーして名前を **eap73-default** に変更している。
- JBoss EAP 7.3 の **standard-sockets** ソケットバインディンググループをコピーして名前を **eap73-standard-sockets** に変更している。
- JBoss EAP 7.3 の **main-server-group** サーバーグループをコピーして名前を **eap73-main-server-group** に変更している。
  - サーバーグループを更新して、JBoss EAP 7.3 のプロファイル **eap73-default** と JBoss EAP 7.3 のソケットバインディンググループ **eap73-standard-sockets** を使用するようにしている。

手順

1. JBoss EAP 7.4 の **domain.xml** 設定ファイルを編集します。

重要

ファイルを編集する前に、JBoss EAP 7.4 の **domain.xml** 設定ファイルをバックアップします。

2. 該当する JBoss EAP 7.3 のプロファイルを JBoss EAP 7.4 の **domain.xml** ファイルにコピーします。以下に例を示します。

```
<profiles>
  ...
  <profile name="eap73-default">
    ...
  </profile>
</profiles>
```

3. 該当する JBoss EAP 7.3 のソケットバインディンググループを JBoss EAP 7.4 の **domain.xml** ファイルにコピーします。以下に例を示します。

```
<socket-binding-groups>
  ...
  <socket-binding-group name="eap73-standard-sockets" default-interface="public">
    ...
  </socket-binding-group>
</socket-binding-groups>
```

4. 該当する JBoss EAP 7.3 のサーバーグループを JBoss EAP 7.4 の **domain.xml** ファイルにコピーします。

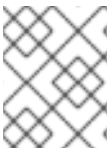
```
<server-groups>
  ...
  <server-group name="eap73-main-server-group" profile="eap73-default">
    ...
    <socket-binding-group ref="eap73-standard-sockets"/>
  </server-group>
</server-groups>
```

8.7.2.2. JBoss EAP 7.3 サーバーのサーバーグループの設定

サーバーグループの名前を変更した場合は、JBoss EAP 7.3 のホスト設定を更新し、JBoss EAP 7.4 の設定に指定された新しいサーバーグループを使用する必要があります。この例では、JBoss EAP 7.4 の **domain.xml** に指定された **eap73-main-server-group** サーバーグループを使用します。

JBoss EAP 7.3 の host-slave.xml

```
<servers>
    <server name="server-one" group="eap73-main-server-group"/>
    <server name="server-two" group="eap73-main-server-group">
        <socket-bindings port-offset="150"/>
    </server>
</servers>
```

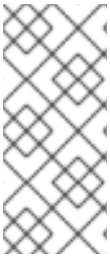


注記

ホストは、実行している JBoss EAP よりも新しいバージョンで導入された機能や設定を使用できません。

8.7.2.3. JBoss EAP 7.3 インスタンスが JBoss EAP 7.4 の更新を取得しないようにする

JBoss EAP 7.3 のホストが JBoss EAP 7.4 固有の設定やリソースを受け取らないように、マネージドドメインコントローラーは、設定更新をホストコントローラーに転送します。これらのリソースを無視するように、JBoss EAP 7.3 ホストを設定する必要があります。これには、JBoss EAP 7.3 のホストで **ignore-unused-configuration** 属性を設定します。



注記

また、**host-exclude** 設定を使用して、特定のバージョンの JBoss EAP を実行するホストが特定のリソースを認識しないよう、ドメインコントローラーに指示することもできます。**host-exclude** 設定の使用例については、JBoss EAP 6 インスタンスが JBoss EAP 7 の更新を取得しないようにする を参照してください。JBoss EAP 7.3 の場合、 **host-exclude** のオプションとして **EAP73** を使用します。

JBoss EAP 7.3 ホストコントローラーのリモートドメインコントローラーへの接続設定で、**ignore-unused-configuration** 属性を **true** に設定します。

JBoss EAP 7.3 の host-slave.xml

```
<domain-controller>
  <remote security-realm="ManagementRealm" ignore-unused-configuration="true">
    <discovery-options>
      <static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}" port="${jboss.domain.master.port:9990}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

この設定では、このホストは使用するサーバーグループと、それらのサーバーグループに関連するプロファイル、ソケットバインディンググループ、およびデプロイメントリソースのみを利用でます。その他のリソースはすべて無視されます。

8.8. JBOSS EAP プロファイルの管理

8.8.1. プロファイル

JBoss EAP は、プロファイルを使用してサーバーが使用できるサブシステムを整理します。プロファイルは、利用可能なサブシステムと各サブシステムの特定の設定で設定されます。プロファイルのサブシステムの数が多いと、サーバーの機能が多くなります。プロファイルのサブシステムが集中的で数が少ないと、機能が少なくなりますが、フットプリントも少なくなります。

JBoss EAP にはほとんどのユースケースに対応する事前定義されたプロファイルが 5 つあります。

default

logging、**security**、**datasources**、**infinispan**、**webservices**、**ee**、**ejb3**、**transactions** など、一般的に使用されるサブシステムが含まれます。

ha

default プロファイルで提供されるサブシステムと、高可用性向けの **jgroups** および **modcluster** サブシステムが含まれます。

full

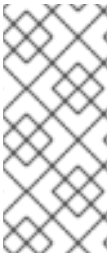
default プロファイルで提供されるサブシステムと、 **messaging-activemq** および **iiop-openjdk** サブシステムが含まれます。

full-ha

full プロファイルで提供されるサブシステムと、高可用性向けの **jgroups** および **modcluster** サブシステムが含まれます。

load-balancer

ビルトインの mod_cluster フロントエンドロードバランサーを使用して他の JBoss EAP インスタンスの負荷を分散するために必要な最低限のサブシステムが含まれます。



注記

JBoss EAP は、既存プロファイルの設定からサブシステムを削除して、エクステンションを無効にしたり、ドライバーやその他のサービスを手作業でアンロードしたりする機能を提供します。ただし、ほとんどの場合、これは必要ありません。JBoss EAP は必要時にサブシステムを動的にロードするため、サーバーまたはアプリケーションがサブシステムを使用しないと、そのサブシステムはロードされません。

既存のプロファイルが必要な機能を提供しない場合、JBoss EAP はカスタムプロファイルを定義する機能も提供します。

8.8.2. プロファイルのクローン

JBoss EAP では、既存のプロファイルをクローンしてマネージドドメインで新しいプロファイルを作成することができます。クローンした既存プロファイルの設定およびサブシステムのコピーが作成されます。

管理 CLI を使用してプロファイルをクローンするには、クローンするプロファイルに **clone** 操作を実行します。

```
/profile=full-ha:clone(to-profile=cloned-profile)
```

管理コンソールからプロファイルをクローンするには、クローンするプロファイルを選択し、**Clone** をクリックします。

8.8.3. 階層的なプロファイルの作成

マネージドドメインでは、プロファイルの階層を作成できます。これにより、他のプロファイルが継承できる共通のエクステンションが含まれるベースプロファイルを作成できます。

マネージドドメインは **domain.xml** の複数のプロファイルを定義します。複数のプロファイルが特定のサブシステムで同じ設定を使用する場合、複数のプロファイルで設定せずに、1 つのプロファイルで設定を行うことができます。親プロファイルの値はオーバーライドできません。

さらに、各プロファイルは他に依存しなくてもすむ必要があります。要素やサブシステムが参照される場合、参照されるプロファイルに定義する必要があります。

管理 CLI を使用して **list-add** 操作を実行し、含めるプロファイルを指定すると、プロファイルに階層の別のプロファイルを含めることができます。

```
/profile=new-profile:list-add(name=includes, value=PROFILE_NAME)
```

第9章 JVM の設定

Java Virtual Machine (JVM) の設定は、スタンドアロンの JBoss EAP サーバーとマネージドドメインの JBoss EAP サーバーでは異なります。

スタンドアロン JBoss EAP サーバーインスタンスでは、起動時にサーバー起動プロセスが JVM 設定を JBoss EAP サーバーに渡します。これは、JBoss EAP を起動する前にコマンドラインから宣言できます。 また、管理コンソールの **Configuration** にある **System Properties** 画面を使用して宣言することもできます。

マネージドドメインでは、JVM の設定は **host.xml** および **domain.xml** 設定ファイルで宣言され、ホスト、サーバーグループ、またはサーバーレベルで設定できます。

注記

システムプロパティーは、起動中に JBoss EAP モジュール (ロギングマネージャーなど) が使用する **JAVA_OPTS** で設定する必要があります。

9.1. スタンドアロンサーバーの JVM 設定

スタンドアロン JBoss EAP サーバーインスタンスの JVM 設定は、サーバーの起動前に **JAVA_OPTS** 環境変数を設定すると、起動時に宣言できます。

Linux で **JAVA_OPTS** 環境変数を設定する例は次のとおりです。

```
$ export JAVA_OPTS="-Xmx1024M"
```

Microsoft Windows 環境でも同じ設定を使用できます。

```
set JAVA_OPTS="Xmx1024M"
```

この代わりに、JVM に渡すオプションの例が含まれる **EAP_HOME/bin** フォルダーの **standalone.conf** ファイル (Windows Server の場合は **standalone.conf.bat**) に JVM 設定を追加することもできます。

JAVA_OPTS 環境変数を設定する以外に、次に挙げる代替方法のいずれかを使用してシステムプロパティーを設定できます。

- 以下のコマンドを実行します。

```
$ EAP_HOME/bin/standalone.sh -Dmyproperty=value
```

- JBoss プロファイル設定ファイル **standalone*.xml** または **domain.xml** を編集します。

警告

システムプロパティーが複数の方法で設定されている場合、JBoss プロファイル設定ファイル **standalone*.xml** または **domain.xml** の値が他の値を上書きするため、JBoss EAP の起動時に問題が発生する可能性があります。たとえば、**JAVA_OPTS** 環境変数と JBoss プロファイル設定ファイルでシステム設定を定義した場合、JBoss プロファイル設定の値は **JAVA_OPTS** の値を上書きします。

9.2. 管理対象ドメインの JVM 設定

JBoss EAP マネージドドメインでは、複数のレベルで JVM の設定を定義できます。特定ホストのカスタム JVM 設定を定義し、それらの設定をサーバーグループまたは個別のサーバーインスタンスに適用できます。

デフォルトでは、サーバーグループおよび各サーバーは親から JVM 設定を継承しますが、各レベルで JVM 設定をオーバーライドすることもできます。

注記

domain.conf の JVM 設定 (Windows Server の場合は **domain.conf.bat**) は JBoss EAP ホストコントローラーの Java プロセスに適用されます。 ホストコントローラーによって制御される個別の JBoss EAP サーバーインスタンスには適用されません。

9.2.1. ホストコントローラーの JVM 設定の定義

ホストコントローラーの JVM 設定を定義し、これらの設定をサーバーグループまたは各サーバーに適用することができます。JBoss EAP には **default** の JVM 設定が含まれますが、以下の管理 CLI コマンドを実行すると、カスタム JVM 設定およびオプションがある **production_jvm** という名前の JVM 設定が新たに作成されます。

```
/host=HOST_NAME/jvm=production_jvm:add(heap-size=2048m, max-heap-size=2048m, max-permgen-size=512m, stack-size=1024k, jvm-options=["-XX:-UseParallelGC"])
```

使用できるすべてのオプションの説明は マネージドドメインの JVM 設定属性 を参照してください。

また、JBoss EAP 管理コンソールで JVM 設定を作成および編集するには、**Runtime → Hosts** と選択し、ホストを選択して 表示 をクリックし、**JVMs** タブを選択します。

これらの設定は **host.xml** の **<jvm>** タグ内に保存されます。

9.2.2. JVM 設定のサーバーグループへの適用

サーバーグループの作成時に、グループのすべてのサーバーが使用する JVM 設定を指定できます。以下の管理 CLI コマンドを実行すると、前の例 で示された **production_jvm** JVM 設定を使用する groupA という名前のサーバーグループが作成されます。

```
/server-group=groupA:add(profile=default, socket-binding-group=standard-sockets)
/server-group=groupA/jvm=production_jvm:add
```

サーバーグループのすべてのサーバーは、**production_jvm** から JVM 設定を継承します。

サーバーグループレベルで特定の JVM 設定をオーバーライドすることもできます。たとえば、別のヒープサイズを設定するには、以下のコマンドを使用できます。

```
/server-group=groupA/jvm=production_jvm:write-attribute(name=heap-size,value="1024m")
```

上記コマンドの実行後、サーバーグループ **groupA** は **production_jvm** から JVM 設定を継承しますが、ヒープサイズの値は **1024m** にオーバーライドされます。

使用できるすべてのオプションの説明は マネージドドメインの JVM 設定属性 を参照してください。

また、JBoss EAP 管理コンソールでサーバーグループの JVM 設定を編集するには、**Runtime** → **Server Groups** と選択し、サーバーグループを選択して 表示 をクリックし、**JVMs** タブを選択します。

サーバーグループの設定は **domain.xml** に保存されます。

9.2.3. JVM 設定の個別のサーバーへの適用

デフォルトでは、個別の JBoss EAP サーバーインスタンスは属するサーバーグループの JVM 設定を継承します。しかし、継承した設定をホストコントローラーからの別の JVM 設定定義でオーバーライドしたり、特定の JVM 設定をオーバーライドすることもできます。

たとえば、以下のコマンドは 前の例で示したサーバーグループ の JVM 定義をオーバーライドし、**server-one** の JVM 設定を **default** の JVM 定義に設定します。

```
/host=HOST_NAME/server-config=server-one/jvm=default:add
```

また、サーバーグループと同様に、サーバーレベルで特定の JVM 設定をオーバーライドすることもできます。たとえば、別のヒープサイズを設定するには、以下のコマンドを使用できます。

```
/host=HOST_NAME/server-config=server-one/jvm=default:write-attribute(name=heap-size,value="1024m")
```

使用できるすべてのオプションの説明は マネージドドメインの JVM 設定属性 を参照してください。

また、JBoss EAP 管理コンソールでサーバー JVM 設定を作成編集するには、**Runtime** → **Hosts** と選択し、ホストを選択して 表示 をサーバー上でクリックし、**JVMs** タブを選択します。

各サーバーのこれらの設定は **host.xml** に保存されます。

9.3. JVM 状態の表示

ヒープおよびスレッドの使用状況など、スタンドアロンまたはマネージドドメインサーバーの JVM リソースの状態は管理コンソールから表示できます。統計はリアルタイムでは表示されませんが、**Refresh** をクリックすると JVM リソースの最新の概要を表示できます。

スタンドアロン JBoss EAP サーバーの JVM の状態を表示するには、以下を行います。

- **Runtime** タブに移動し、サーバーを選択して、状態 を選択します。

マネージドドメインでの JBoss EAP サーバーの JVM の状態を表示するには、以下を行います。

- **Runtime** → **Hosts** と選択し、ホストとサーバーを選択して 状態 を選択します。

以下のヒープ使用情報が表示されます。

Max

メモリー管理に使用できるメモリーの最大量。

Used

使用されたメモリーの量。

Committed

JVM が使用するために確保されたメモリー量。

JVM のアップタイムやスレッドの使用状況などの他の情報も表示されます。

9.4. JVM の調整

JVM のパフォーマンスを最適化するための情報は、**Performance Tuning Guide**の JVM Tuning を参照してください。

第10章 MAIL サブシステム

10.1. MAIL サブシステムの設定

mail サブシステムを使用すると、JBoss EAP でメールセッションを設定でき、JNDI を使用してこれらのセッションをアプリケーションにインジェクトできます。また、Jakarta EE の **@MailSessionDefinition** および **@MailSessionDefinitions** アノテーションを使用する設定もサポートします。

アプリケーションで使用する SMTP サーバーの設定

1. 以下の CLI コマンドを使用して SMTP サーバーとアウトバウンドソケットバインディングを設定します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-smtp:add(host=localhost, port=25)
```

```
/subsystem=mail/mail-session=mySession:add(jndi-name=java:jboss/mail/MySession)
```

```
/subsystem=mail/mail-session=mySession/server=smtp:add(outbound-socket-binding-ref=my-smtp, username=user, password=pass, tls=true)
```

2. アプリケーション内で設定されたメールセッションを呼び出します。

```
@Resource(lookup="java:jboss/mail/MySession")
private Session session;
```

メールセッションおよびサーバーの設定で使用できる属性の完全リストは Mail サブシステムの属性 を参照してください。

10.2. カスタムトランスポートの設定

POP3 や IMAP などの標準のメールサーバーを使用している場合、メールサーバーには定義できる属性のセットがあります。これらの属性の一部は必須です。最も重要な属性はアウトバウンドメールソケットバインディングの参照である **outbound-socket-binding-ref** で、ホストアドレスとポート番号で定義されます。

outbound-socket-binding-ref の定義は、負荷分散の目的でホスト設定に複数のホストを使用するユーザーにとっては最も効率的なソリューションではない場合があります。標準の Jakarta Mail は負荷分散のために複数のホストを使用するホスト設定をサポートしません。そのため、複数のホストを使用するこの設定を持つユーザーはカスタムメールトランスポートを実装する必要があります。カスタムメールトランスポートは **outbound-socket-binding-ref** を必要とせず、カスタムのホストプロパティー形式を許可します。

カスタムのメールトランスポートは管理 CLI から設定できます。

1. 新しいメールセッションを追加し、JNDI 名を指定します。

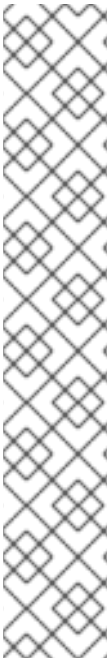
```
/subsystem=mail/mail-session=mySession:add(jndi-name=java:jboss/mail/MySession)
```

2. アウトバウンドソケットバインディングを追加し、ホストとポートを指定します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-smtp-binding:add(host=localhost, port=25)
```

3. SMTP サーバーを追加し、アウトバウンドソケットバインディング、ユーザー名、およびパスワードを指定します。

```
/subsystem=mail/mail-session=mySession/server=smtp:add(outbound-socket-binding-ref=my-smtp-binding, username=user, password=pass, tls=true)
```



注記

同様の手順で POP3 または IMAP サーバーを設定できます。

POP3 サーバー

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-pop3-binding:add(host=localhost, port=110)
/subsystem=mail/mail-session=mySession/server=pop3:add(outbound-socket-binding-ref=my-pop3-binding, username=user, password=pass)
```

IMAP サーバー

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-imap-binding:add(host=localhost, port=143)
/subsystem=mail/mail-session=mySession/server=imap:add(outbound-socket-binding-ref=my-imap-binding, username=user, password=pass)
```

カスタムサーバーを使用するには、アウトバウンドソケットバインディングを指定せずにカスタムメールサーバーを作成します。カスタムメールサーバーのプロパティー定義でホスト情報を指定できます。以下に例を示します。

```
/subsystem=mail/mail-session=mySession/custom=myCustomServer:add(username=user,password=pass, properties={"host" => "myhost", "my-property" =>"value"})
```

カスタムプロトコルを定義する場合、ピリオド (.) が含まれるプロパティー名は完全修飾名と見なされ、直接渡されます。**my-property** など、他の形式は **mail.server-name.my-property** という形式に変換されます。

以下の XML は、カスタムサーバーが含まれるメール設定の例になります。

```
<subsystem xmlns="urn:jboss:domain:mail:3.0">
  <mail-session name="default" jndi-name="java:jboss/mail/Default">
      <smtp-server outbound-socket-binding-ref="mail-smtp"/>
  </mail-session>
   <mail-session name="myMail" from="user.name@domain.org" jndi-name="java:/Mail">
      <smtp-server password="password" username="user" tls="true" outbound-socket-binding-ref="mail-smtp"/>
      <pop3-server outbound-socket-binding-ref="mail-pop3"/>
      <imap-server password="password" username="nobody" outbound-socket-binding-ref="mail-imap"/>
   </mail-session>
```

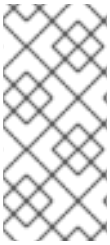
```
    <mail-session name="custom" jndi-name="java:jboss/mail/Custom" debug="true">
      <custom-server name="smtp" password="password" username="username">
        <property name="host" value="mail.example.com"/>
      </custom-server>
    </mail-session>
    <mail-session name="custom2" jndi-name="java:jboss/mail/Custom2" debug="true">
      <custom-server name="pop3" outbound-socket-binding-ref="mail-pop3">
        <property name="custom-prop" value="some-custom-prop-value"/>
      </custom-server>
    </mail-session>
  </subsystem>
```

10.3. パスワードに認証情報ストアを使用

mail サブシステムでクリアテキストのパスワードを提供する他に、認証情報ストアを使用してパスワードを提供することもできます。**elytron** サブシステムを使用すると、認証情報ストアを作成してパスワードをセキュアに保存し、JBoss EAP 全体でパスワードを使用することができます。認証情報ストアの作成および使用に関する詳細は、**How to Configure Server Security**の Credential Store を参照してください。

管理 CLI を使用した認証情報ストアの使用

```
/subsystem=mail/mail-session=mySession/server=smtp:add(outbound-socket-binding-ref=my-smtp-binding, username=user, credential-reference={store=exampleCS, alias=mail-session-pw}, tls=true)
```



注記

以下は、クリアテキストパスワードを使用する **credential-reference** 属性を指定する方法の例になります。

```
credential-reference={clear-text="MASK-Ewcyuqd/nP9;A1B2C3D4;351"}
```

管理コンソールを使用した認証情報ストアの使用

1. 管理コンソールにアクセスします。詳細は 管理コンソール を参照してください。
2. **Configuration → Subsystems → Mail** と選択します。
3. 該当するメールセッションを選択し、表示 をクリックします。
4. **Server** を選択し、該当するメールセッションサーバーを選択します。**Credential Reference** タブで認証情報リファレンスを設定でき、 **Attributes** タブでその他の属性を編集できます。

第11章 JBOSS EAP を用いたロギング

JBoss EAP は、EAP での内部使用とデプロイされたアプリケーションによる使用の両方で設定可能なロギング機能を提供します。**logging** サブシステムは JBoss LogManager を基盤とし、JBoss Logging だけでなくサードパーティーアプリケーションのロギングフレームワークを複数サポートします。

11.1. サーバーロギング

11.1.1. サーバーロギング

デフォルトでは、すべての JBoss EAP ログエントリーは **server.log** ファイルに書き込みされます。このファイルの場所は操作するモードによって異なります。

- スタンドアロンサーバーの場合: **EAP_HOME/standalone/log/server.log**
- マネージドドメインの場合: **EAP_HOME/domain/servers/SERVER_NAME/log/server.log**

このファイルはサーバーログとも呼ばれます。詳細は ルートロガー を参照してください。

11.1.2. 起動時のロギング

起動中に JBoss EAP は Java 環境と各サービスの起動に関する情報をログに記録します。このログは、トラブルシューティングに役に立ちます。デフォルトでは、すべてのログエントリーが サーバーログ に書き込まれます。

起動時のロギング設定は **logging.properties** 設定ファイルに指定されます。これは、JBoss EAP **logging** が開始され、継承するまでアクティブになります。このファイルの場所は操作するモードによって異なります。

- スタンドアロンサーバーの場合: **EAP_HOME/standalone/configuration/logging.properties**
- マネージドドメインの場合:
  ドメインコントローラーおよびサーバーごとに 1 つの **logging.properties** ファイルがあります。
  - ドメインコントローラー: **EAP_HOME/domain/configuration/logging.properties**
  - サーバー: **EAP_HOME/domain/servers/SERVER_NAME/data/logging.properties**

警告

logging.properties ファイルは、直接編集する必要があるユースケース以外では直接編集しないことが推奨されます。直接編集する前に、Red Hat カスタマーポータル でサポートケースを作成することが推奨されます。

logging.properties ファイルに手動で行った変更は起動時に上書きされます。

11.1.2.1. 起動エラーの表示

JBoss EAP をトラブルシューティングする場合、最初に行うべきことの 1 つは、起動時に発生したエラーをチェックすることです。提供された情報を使用して原因を診断し、解決します。起動時のエラーをトラブルシューティングする際にサポートが必要な場合はサポートケースを作成してください。

起動時のエラーを表示する方法は 2 つあり、どちらも利点があります。1 つは **server.log** ファイルを確認する方法で、もう 1 つは **read-boot-errors** 管理 CLI コマンドを使用してブートエラーを確認する方法になります。

サーバーログファイルの確認
server.log ファイルを開いて起動中に発生したエラーを確認します。

この方法では、各エラーメッセージおよび関連するメッセージを確認でき、エラーが発生した理由の詳細を知ることができます。また、エラーメッセージをプレーンテキスト形式で表示することもできます。

1. ファイルビューアーで **server.log** を開きます。
2. ファイルの最後に移動します。
3. 最後の起動シーケンスの開始を示す **WFLYSRV0049** メッセージ ID を後方検索します。
4. ログのその位置から **ERROR** を前方検索します。各検索一致箇所には、エラーの説明が示され、関連するモジュールがリストされます。

以下は、**server.log** ログファイルのエラー説明の例です。

```
2016-03-16 14:32:01,627 ERROR [org.jboss.msc.service.fail] (MSC service thread 1-7) MSC000001:
Failed to start service jboss.undertow.listener.default: org.jboss.msc.service.StartException in service
jboss.undertow.listener.default: Could not start http listener
        at org.wildfly.extension.undertow.ListenerService.start(ListenerService.java:142)
        at
org.jboss.msc.service.ServiceControllerImpl$StartTask.startService(ServiceControllerImpl.java:1948)
        at org.jboss.msc.service.ServiceControllerImpl$StartTask.run(ServiceControllerImpl.java:1881)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)
Caused by: java.net.BindException: Address already in use
        ...
```

管理 **CLI** からのブートエラーの読み取り
サーバーが起動しても起動中にエラーが報告された場合、**read-boot-errors** 管理 CLI コマンドを使用してエラーを確認できます。

この方法では、サーバーのファイルシステムにアクセスする必要がありません。したがって、エラーを監視する担当者がファイルシステムアクセスを持ってない場合に役に立ちます。これは CLI コマンドであるため、スクリプトで使用できます。 たとえば、複数の JBoss EAP インスタンスを起動し、起動時に発生したエラーをチェックするスクリプトを記述できます。

次の管理 CLI コマンドを実行します。

```
/core-service=management:read-boot-errors
```

起動中に発生したすべてのエラーがリストされます。

```
{
    "outcome" => "success",
```

```
    "result" => [
        {
            "failed-operation" => {
                "operation" => "add",
                "address" => [
                    ("subsystem" => "undertow"),
                    ("server" => "default-server"),
                    ("http-listener" => "default")
                ]
            },
            "failure-description" => "{\"WFLYCTL0080: Failed services\" =>
{\"jboss.undertow.listener.default\" => \"org.jboss.msc.service.StartException in service
jboss.undertow.listener.default: Could not start http listener
    Caused by: java.net.BindException: Address already in use\"}",
            "failed-services" => {"jboss.undertow.listener.default" =>
"org.jboss.msc.service.StartException in service jboss.undertow.listener.default: Could not start http
listener
    Caused by: java.net.BindException: Address already in use"}
        }
        ...
    ]
}
```

11.1.3. ガベッジコレクションのロギング

ガベッジコレクションロギングは、すべてのガベッジコレクションのアクティビティーをプレーンテキストのログファイルに記録します。これらのログファイルは診断を行うのに便利です。ガベッジコレクションロギングは、IBM の Java Development Kit を除くすべてのサポート対象設定の JBoss EAP スタンドアロンサーバーではデフォルトで有効になっています。

ガベッジコレクションログの場所は **EAP_HOME/standalone/log/gc.log.DIGIT.current** です。ガベッジコレクションのログは 3 MB ずつに制限され、最大 5 つのファイルがローテーションされます。

トラブルシューティングに便利で、オーバーヘッドは最低限であるため、ガベッジコレクションのロギングを有効にすることが強く推奨されます。しかし、スタンドアロンサーバーのガベッジコレクションのロギングを無効にする場合は、サーバーを起動する前に **GC_LOG** 変数を **false** に設定します。以下に例を示します。

```
$ export GC_LOG=false
$ EAP_HOME/bin/standalone.sh
```

11.1.4. デフォルトのログファイルの場所

以下のログファイルは、デフォルトのロギング設定に対して作成されます。デフォルトの設定では、periodic ログハンドラーを使用してサーバーログファイルが書き込まれます。

表11.1 スタンドアロンサーバーのデフォルトログファイル

ログファイル	説明
EAP_HOME/standalone/log/server.log	サーバー起動メッセージを含むサーバーログメッセージが含まれます。

ログファイル	説明
EAP_HOME/standalone/log/gc.log.DIGIT.current	ガベッジコレクションの詳細が含まれます。

表11.2 マネージドドメイン用のデフォルトログファイル

ログファイル	説明
EAP_HOME/domain/log/host-controller.log	ホストコントローラーの起動に関連するログメッセージが含まれます。
EAP_HOME/domain/log/process-controller.log	プロセスコントローラーの起動に関連するログメッセージが含まれます。
EAP_HOME/domain/servers/**SERVER_NAME**/log/server.log	サーバー起動メッセージを含む、名前付きサーバーのログメッセージが含まれます。

11.1.5. サーバーのデフォルトロケールの設定

JVM プロパティーを適切な起動設定ファイルに設定すると、デフォルトのロケールを設定できます。起動設定ファイルは、スタンドアロンサーバーの場合は **EAP_HOME/bin/standalone.conf**、マネージドドメインの場合は **EAP_HOME/bin/domain.conf** になります。

注記

Windows Server の場合、JBoss EAP 起動設定ファイルは **standalone.conf.bat** と **domain.conf.bat** になります。

国際化または現地化されたログメッセージはこのデフォルトロケールを使用します。JBoss EAP**Development Guide**の 国際化されたログメッセージの作成 に関する情報を参照してください。

言語の設定
言語を指定するには、**JAVA_OPTS** 変数を使用して **user.language** プロパティーを設定します。たとえば、以下の行を起動設定ファイルに追加し、フランス語のロケールを設定します。

```
JAVA_OPTS="$JAVA_OPTS -Duser.language=fr"
```

国際化および現地化されたログメッセージがフランス語で出力されるようになります。

言語および国の設定
言語の他に、**user.country** プロパティーを設定して国を指定することもできます。たとえば、以下の行を起動設定ファイルに追加すると、ポルトガル語のロケールがブラジルに設定されます。

```
JAVA_OPTS="$JAVA_OPTS -Duser.language=pt -Duser.country=BR"
```

国際化および現地化されたログメッセージがブラジルポルトガル語で出力されるようになります。

org.jboss.logging.locale プロパティーを使用したサーバーロケールの設定
org.jboss.logging.locale プロパティーを使用すると、Boss EAP からのメッセージや JBoss EAP が所有する依存関係からのメッセージなど、JBoss Logging を使用してログに記録されたメッセージのロ

ケールを上書きできます。Jakarta Server Faces などの他の依存関係ではロケールを上書きできません。

JBoss EAP サーバーをシステムデフォルト以外のロケールで起動するには、操作モードに応じて **EAP_HOME/bin/standalone.conf** または **EAP_HOME/bin/domain.conf** ファイルを編集し、以下のコマンドを追加して必要なロケールの JVM パラメーターを設定します。プロパティーの値は BCP 47 形式で指定する必要があります。たとえば、ブラジルポルトガル語を設定する場合は pt-BR を使用します。

```
JAVA_OPTS="$JAVA_OPTS -Dorg.jboss.logging.locale=pt-BR"
```

11.2. ログファイルの表示

サーバーやアプリケーションログの確認は、診断エラー、パフォーマンスの問題、およびその他の問題に対応するために重要です。サーバーのファイルシステムで直接ログを表示したいユーザーもいます。直接ファイルシステムにアクセスできないユーザーやグラフィカルインターフェイスを使用したいユーザーは JBoss EAP の管理コンソールからログを表示することができます。また、管理 CLI を使用してログを表示することもできます。

管理インターフェイスの 1 つからログにアクセスするには、サーバーの **jboss.server.log.dir** プロパティーによって指定されたディレクトリーに存在し、File、Periodic rotating、Size rotating、または Periodic size rotating ログハンドラーとして定義される必要があります。RBAC ロール割り当ても考慮されるため、管理コンソールまたは管理 CLI にログインしたユーザーはアクセス権限を持つログのみ表示できます。

管理コンソールからのログの表示

管理コンソールから直接ログを表示できます。

1. **Runtime** タブを選択し、該当するサーバーを選択します。
2. ログファイル を選択し、リストからログファイルを選択します。
3. 表示 をクリックしてログの内容を表示および検索するか、ドロップダウンリストから ダウンロード を選択してログファイルをローカルファイルシステムにダウンロードします。

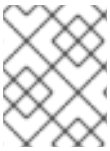
警告

管理コンソールのログビューアーは、100MB 以上のログファイルなどの非常に大きなログファイルを表示するテキストエディターの代わりとして使用するものではありません。15MB を超えるログファイルを開こうとすると、確認を求められます。管理コンソールで非常に大きなファイルを開くと、ブラウザーがクラッシュする可能性があるため、大きなログファイルは常にローカルにダウンロードし、テキストエディターで開くようにしてください。

管理 CLI からのログの表示

read-log-file コマンドを使用すると管理 CLI からログファイルの内容を取得できます。デフォルトでは、指定されたログファイルの最後の **10** 行が表示されます。

```
/subsystem=logging/log-file=LOG_FILE_NAME:read-log-file
```



注記

マネージドドメインでは、このコマンドの前に **/host=HOST_NAME/server=SERVER_NAME** を追加してください。

以下のパラメーターを使用するとログの出力をカスタマイズできます。

encoding
ファイルの読み取りに使用される文字エンコーディング。

lines
ファイルから読み取る行数。**-1** はログの行すべてを取得します。デフォルトは **10** です。

skip
読み取る前にスキップする行数。デフォルトは **0** です。

tail
ファイルの最後から読み取るかどうか。デフォルト値は **true** です。

たとえば、以下の管理 CLI コマンドは **server.log** ログファイルの最初の **5** 行を読み取ります。

```
/subsystem=logging/log-file=server.log:read-log-file(lines=5,tail=false)
```

このコマンドを実行すると以下が出力されます。

```
{
    "outcome" => "success",
    "result" => [
        "2016-03-24 08:49:26,612 INFO  [org.jboss.modules] (main) JBoss Modules version 1.5.1.Final-redhat-1",
        "2016-03-24 08:49:26,788 INFO  [org.jboss.msc] (main) JBoss MSC version 1.2.6.Final-redhat-1",
        "2016-03-24 08:49:26,863 INFO  [org.jboss.as] (MSC service thread 1-7) WFLYSRV0049: JBoss EAP 7.0.0.GA (WildFly Core 2.0.13.Final-redhat-1) starting",
        "2016-03-24 08:49:27,973 INFO  [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0039: Creating http management service using socket-binding (management-http)",
        "2016-03-24 08:49:27,994 INFO  [org.xnio] (MSC service thread 1-1) XNIO version 3.3.4.Final-redhat-1"
    ]
}
```

11.3. LOGGING サブシステム

JBoss EAP の **logging** サブシステムは ログカテゴリー および ログハンドラー のシステムを使用して設定されます。ログカテゴリーはキャプチャーするメッセージを定義します。ログハンドラーは、ディスクへの書き込みやコンソールへの送信など、これらのメッセージの対応方法を定義します。

ロギングプロファイル は、一意な名前が付けられたロギング設定の作成や、他のロギング設定に関係しないアプリケーションへの割り当てを可能にします。ロギングプロファイルの設定は、メインの **logging** サブシステムとほぼ同じです。

11.3.1. ルートロガー

JBoss EAP のルートロガーは、ログカテゴリーによってキャプチャーされないサーバーへ送信されたすべてのログメッセージ (指定のログレベル以上) をキャプチャーします。

デフォルトでは、ルートロガーはコンソールおよび周期ログハンドラーを使用するように設定されます。周期ログハンドラーは、**server.log** ファイルへ書き込むよう設定されます。このファイルはサーバーログとも呼ばれます。

詳細は ルートロガーの設定 を参照してください。

11.3.2. ログカテゴリー

ログカテゴリーは、キャプチャーするログメッセージのセットと、メッセージを処理する 1 つ以上のログハンドラーを定義します。

キャプチャーするログメッセージは、指定された元の Java パッケージとログレベルによって定義されます。そのパッケージのクラスおよびそのログレベル以上のメッセージがログカテゴリーによってキャプチャーされ、指定のログハンドラーに送信されます。

注記

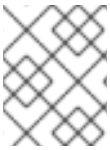
通常、ログカテゴリーは Java パッケージとクラス名ですが、**Logger.getLogger(LOGGER_NAME)** メソッドによって指定された名前をすべて使用できます。

ログカテゴリーは、独自のハンドラーの代わりにルートロガーのログハンドラーを任意で使用することができます。

詳細は ログカテゴリーの設定 を参照してください。

11.3.3. ログハンドラー

ログハンドラーはキャプチャーしたメッセージの記録方法を定義します。使用できるログハンドラーの種類は **console**、**file**、**periodic**、**size**、**periodic size**、**syslog**、 **custom**、および **async** です。



注記

ログハンドラーは、アクティブにするために少なくとも 1 つのロガーに追加する必要があります。

ログハンドラーの種類

Console

Console ログハンドラーは、ログメッセージをホストオペレーティングシステムの標準出力 (**stdout**) または標準エラー (**stderr**) ストリームに書き込みます。これらのメッセージは、JBoss EAP がコマンドラインプロンプトから実行された場合に表示されます。オペレーティングシステムで標準出力または標準エラーストリームをキャプチャーするように設定されていない限り、Console ログハンドラーからのメッセージは保存されません。

File

File ログハンドラーは、ログメッセージを指定のファイルに書き込みます。

Periodic

Periodic ログハンドラーは、指定した時間が経過するまで、ログメッセージを指定ファイルに書き込みます。その時間が経過した後、指定のタイムスタンプが追記されてファイルの名前が変更され、 ハンドラーは元の名前で新規作成されたログファイルに書き込みを継続します。

Size

Size ログハンドラーは、指定したファイルが指定したサイズに達するまで、そのファイルにログメッセージを書き込みます。ファイルが指定したサイズに達すると、数値の接尾辞が付いて名前が変更され、ハンドラーは元の名前で新規作成されたログファイルに書き込みを継続します。各サイズログハンドラーは、この方式で保管されるファイルの最大数を指定する必要があります。

Periodic Size

Periodic Size ログハンドラーは、ファイルが指定のサイズに達するまで、または指定の期間が経過するまで、ログメッセージを名前の付いたファイルに書き込みます。その後、ファイルの名前が変更され、ハンドラーは元の名前で新規作成されたログファイルに書き込みを継続します。
これは Periodic と Size ログハンドラーの組み合わせで、組み合わされた属性をサポートします。

Syslog

syslog ハンドラーは、リモートのロギングサーバーへメッセージを送信するために使用できます。これにより、複数のアプリケーションが同じサーバーにログメッセージを送信でき、そのサーバーですべてのログメッセージを解析できます。

Socket

ソケットログハンドラーを使用して、ログメッセージをソケット上でリモートロギングサーバーに送信できます。ソケットは TCP または UDP ソケットになります。

Custom

カスタムログハンドラーにより、実装された新たなタイプのログハンドラーを設定することができます。カスタムハンドラーは、**java.util.logging.Handler** を拡張する Java クラスとして実装し、モジュール内に格納する必要があります。Log4J アペンダーをカスタムログハンドラーとして使用することもできます。

Async

Async ログハンドラーは、 単一または複数のログハンドラーを対象とする非同期動作を提供するラッパーログハンドラーです。Async ログハンドラーは、待ち時間が長かったり、ネットワークファイルシステムへのログファイルの書き込みなどにパフォーマンス上の問題があるログハンドラーに対して有用です。

各ログハンドラーの設定に関する詳細は、ログハンドラーの設定 の項を参照してください。

11.3.4. ログレベル

ログレベルとは、ログメッセージの性質と重大度を示す列挙値です。特定のログメッセージのレベルは、そのメッセージを送信するために選択したロギングフレームワークの適切なメソッドを使用して開発者が指定できます。

JBoss EAP は、サポートされるアプリケーションロギングフレームワークによって使用されるすべてのログレベルをサポートします。最も一般的に使用されるログレベルは、ログレベルの低い順に **TRACE**、**DEBUG**、**INFO**、**WARN**、**ERROR** および **FATAL** となります。

ログレベルはログカテゴリーとログハンドラーによって使用され、それらが担当するメッセージを限定します。各ログレベルには、他のログレベルに対して相対的な順番を示す数値が割り当てられています。ログカテゴリーとハンドラーにはログレベルが割り当てられ、そのレベル以上のログメッセージのみを処理します。たとえば、**WARN** レベルのログハンドラーは、**WARN**、**ERROR**、および **FATAL** のレベルのメッセージのみを記録します。

サポート対象のログレベル

ログのレベル	Value	説明

ログのレベル	Value	説明
ALL	Integer.MIN_VALUE	すべてのログメッセージを提供します。
FINEST	300	-
FINER	400	-
TRACE	400	**TRACE** レベルのログメッセージは、実行中のアプリケーションの状態に関する詳細な情報を提供し、デバッグ時のみキャプチャーされます。
DEBUG	500	**DEBUG** レベルのログメッセージは、個々の要求またはアプリケーションアクティビティーの進捗状況を示し、通常はデバッグ時のみキャプチャーされます。
FINE	500	-
CONFIG	700	-
INFO	800	**INFO** レベルのログメッセージはアプリケーションの全体的な進捗状況を示します。多くの場合、アプリケーションの起動、シャットダウン、およびその他の主要なライフサイクルイベントに使用されます。
WARN	900	**WARN** レベルのログメッセージはエラーではないが、理想的とは見なされない状況を示します。**WARN** ログメッセージは将来エラーが発生する可能性のある状況を示すことがあります。
WARNING	900	-
ERROR	1000	**ERROR** レベルのログメッセージは、現在のアクティビティーや要求の完了を妨げる可能性があるが、アプリケーションの実行を妨げない発生済みエラーを示します。
SEVERE	1000	-
FATAL	1100	**FATAL** レベルのログメッセージは重大なサービス障害やアプリケーションのシャットダウンを引き起こしたり、JBoss EAP のシャットダウンを引き起こしたりする可能性があるイベントを示します。
OFF	Integer.MAX_VALUE	ログメッセージを表示しません。

注記

ALL は、最低ログレベルであり、すべてのログレベルのメッセージを含みます。ロギングの量は最も多くなります。

FATAL は、最大ログレベルであり、そのレベルのメッセージのみを含みます。ロギングの量は最も少なくなります。

11.3.5. ログフォーマッター

フォーマッターはログメッセージのフォーマットに使用されます。**named-formatter** 属性を使用するとフォーマッターをログハンドラーに割り当てできます。ログハンドラー設定の詳細は ログハンドラーの設定 を参照してください。

logging サブシステムには 4 種類のフォーマッターが含まれます。

- パターンフォーマッター
- JSON フォーマッター
- XML フォーマッター
- カスタムフォーマッター

パターンフォーマッター
パターンフォーマッターは、ログメッセージをプレーンテキストでフォーマットするために使用されます。ログハンドラーの **named-formatter** 属性としてフォーマットを使用する他に、あらかじめフォーマッターリソースを作成する必要なく **formatter** 属性として使用することもできます。パターン構文に関する詳細は パターンフォーマッターのフォーマット文字 を参照してください。

パターンフォーマッターの設定方法は パターンフォーマッターの設定 を参照してください。

JSON フォーマッター
JSON フォーマッターは、ログメッセージを JSON 形式でフォーマットするために使用されます。

JSON フォーマッターの設定方法は JSON フォーマッターの設定 を参照してください。

XML フォーマッター
XML ログフォーマッターは、ログメッセージを XML 形式でフォーマットするために使用されます。

XML フォーマッターの設定方法は XML フォーマッターの設定 を参照してください。

カスタムフォーマッター
ハンドラーと使用するカスタムフォーマッターです。ほとんどのログレコードは printf 形式でフォーマットされることに注意してください。メッセージを適切にフォーマットするには、**org.jboss.logmanager.ExtLogRecord#getFormattedMessage()** の呼び出しが必要になることがあります。

カスタムログフォーマッターの設定方法は Custom ログフォーマッターの設定 を参照してください。

11.3.6. フィルター式

フィルター式は **filter-spec** 属性を使用して設定され、さまざまな基準に基づいてログメッセージを記録するために使用されます。フィルターチェックは、常に未フォーマットの raw メッセージに対して行われます。ロガーまたはハンドラーのフィルターを含めることができますが、ハンドラーに配置されたフィルターよりもロガーフィルターが優先されます。

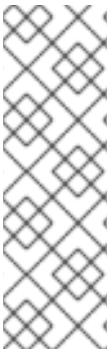
注記

ルートロガーに対して指定された **filter-spec** は他のロガーによって継承されません。ハンドラーごとに **filter-spec** を指定する必要があります。

表11.3 ロギングのフィルター式

フィルター式	説明
accept	すべてのログメッセージを許可します。
deny	すべてのログメッセージを拒否します。
not[filter expression]	単一のフィルター式の逆の値を返します。以下に例を示します。 **not(match("WFLY"))**
all[filter expression]	フィルター式のコンマ区切りリストから連結された値を返します。以下に例を示します。 **all(match("WFLY"),match("WELD"))**
any[filter expression]	フィルター式のコンマ区切りリストから1つの値を返します。以下に例を示します。 **any(match("WFLY"),match("WELD"))**
levelChange[level]	指定のレベルでログレコードを更新します。以下に例を示します。 **levelChange(WARN)**
levels[levels]	レベルのコンマ区切りリストにあるレベルの1つでログメッセージをフィルターします。以下に例を示します。 **levels(DEBUG,INFO,WARN,ERROR)**
levelRange[minLevel,maxLevel]	指定されたレベル範囲内でログメッセージをフィルターします。**[** および **]** は、含まれるレベルを示すために使用されます。**(** および **)** は除外されるレベルを示すために使用されます。以下に例を示します。 ● **levelRange[INFO,ERROR]** ○ 最小レベルは **INFO** 以上で、最大レベルは **ERROR** 以下でなければなりません。 ● **levelRange[DEBUG,ERROR)** ○ 最小レベルは **DEBUG** 以上で、最大レベルは **ERROR** 未満でなければなりません。

フィルター式	説明
match["pattern"]	提供される正規表現を使用してログメッセージをフィルターします。以下に例を示します。 **match("WFLY\d+")**
substitute["pattern","replacement value"]	最初にパターン (最初の引数) と一致した値を代替テキスト (2 番目の引数) に置き換えるフィルター。以下に例を示します。 **substitute("WFLY","EAP")**
substituteAll["pattern","replacement value"]	パターン (最初の引数) と一致したすべての値を代替テキスト (2 番目の引数) に置き換えるフィルター。以下に例を示します。 **substituteAll("WFLY","EAP")**



注記

管理 CLI を使用してフィルター式を設定する場合、値が文字列として正しく処理されるよう、フィルターテキストのコンマと引用符を必ずエスケープしてください。コンマと引用符の前にバックスラッシュ (\) を付け、式全体を引用符で囲む必要があります。以下は **substituteAll("WFLY","YLFW")** を適切にエスケープした例になります。

```
/subsystem=logging/console-handler=CONSOLE:write-attribute(name=filter-spec,
value="substituteAll(\"WFLY\"\,\"YLFW\")")
```

11.3.7. 暗黙的なロギングの依存関係

JBoss EAP の **logging** サブシステムはデフォルトで暗黙的なロギング API 依存関係をデプロイメントに追加します。**add-logging-api-dependencies** 属性を使用すると、この暗黙的な依存関係をデプロイメントに追加するかどうかを制御できます。 この属性はデフォルトでは **true** に設定されています。

管理 CLI を使用して **add-logging-api-dependencies** 属性を **false** に設定すると、暗黙的なロギング API 依存関係がデプロイメントに追加されないようになります。

```
/subsystem=logging:write-attribute(name=add-logging-api-dependencies, value=false)
```

logging サブシステムの暗黙的な依存関係については、JBoss EAP**Development Guide**の Implicit Module Dependencies の項を参照してください。

11.4. ログカテゴリーの設定

ここでは、管理 CLI を使用してログカテゴリーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Categories** を選択するとログカテゴリーを設定することができます。

ログカテゴリーを設定するために実行する主なタスクは次のとおりです。

- 新しいログカテゴリーの追加。

- ログカテゴリーの設定。
- ログハンドラーのログカテゴリーへの割り当て。

重要

ロギングプロファイルにログカテゴリーを設定する場合、 コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

ログカテゴリーの追加

ログカテゴリー名は、元の Java パッケージによって定義されます。ログレベルなどのその他の設定に準拠する限り、そのパッケージのクラスからのメッセージはキャプチャーされます。

```
/subsystem=logging/logger=LOG_CATEGORY:add
```

ログカテゴリーの設定

必要性に応じて、以下のログカテゴリー属性を 1 つ以上設定する必要がある場合があります。利用できるログカテゴリー属性の完全リストと説明は、ログカテゴリーの属性 を参照してください。

- ログレベルを設定します。
  ログカテゴリーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、ログレベル を参照してください。

  ```
  /subsystem=logging/logger=LOG_CATEGORY:write-attribute(name=level,value=LEVEL)
  ```

- このカテゴリーがルートロガーのログハンドラーを使用するかどうかを設定します。
  デフォルトでは、ログカテゴリーは独自のハンドラーと、ルートロガーのハンドラーを使用します。ログカテゴリーが割り当てられたハンドラーのみを使用する必要がある場合は **use-parent-handlers** 属性を **false** に設定します。

  ```
  /subsystem=logging/logger=LOG_CATEGORY:write-attribute(name=use-parent-handlers,value=USE_PARENT_HANDLERS)
  ```

- フィルター式を設定します。
  ログカテゴリーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

  ```
  /subsystem=logging/logger=LOG_CATEGORY:write-attribute(name=filter-spec,value=FILTER_EXPRESSION)
  ```

  利用可能なフィルター式の詳細は フィルター式 の項を参照してください。

ハンドラーの割り当て

ログハンドラーをログカテゴリーに割り当てます。

```
/subsystem=logging/logger=LOG_CATEGORY:add-handler(name=LOG_HANDLER_NAME)
```

ログカテゴリーの削除
ログカテゴリーは **remove** 操作で削除できます。

```
/subsystem=logging/logger=LOG_CATEGORY:remove
```

11.5. ログハンドラーの設定

ログハンドラーはキャプチャーしたメッセージの記録方法を定義します。以下の項を参照して必要なログハンドラーのタイプを設定してください。

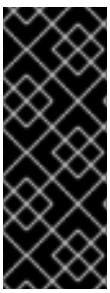
- Console ログハンドラー
- File ログハンドラー
- Periodic rotating ログハンドラー
- Size rotating ログハンドラー
- Periodic size rotating ログハンドラー
- Syslog ハンドラー
- File ログハンドラー
- Custom ログハンドラー
- Async ログハンドラー

11.5.1. Console ログハンドラーの設定

ここでは、管理 CLI を使用して Console ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Handler → Console Handler** と選択すると Console ログハンドラーを設定することができます。

Console ログハンドラーを設定するために実行する主なタスクは次のとおりです。

- 新しい Console ログハンドラーの追加。
- Console ログハンドラーの設定。
- Console ログハンドラーのロガーへの割り当て。



重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

Console ログハンドラーの追加

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:add
```

Console ログハンドラーの設定
必要性に応じて、以下の Console ログハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できる Console ログハンドラー属性の完全リストと説明は、Console ログハンドラーの属性 を参照してください。

- ログレベルを設定します。
  ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、ログレベル を参照してください。

  ```
  /subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
  ```

- ターゲットを設定します。
  ハンドラーのターゲットを設定します。値は **System.out**、**System.err**、**console** のいずれかになります。デフォルト **System.out** です。

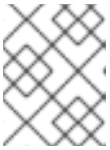
  ```
  /subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=target,value=TARGET)
  ```

- エンコーディングを設定します。
  ハンドラーのエンコーディングを設定します (例: **utf-8**)。

  ```
  /subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
  ```

- ログフォーマッターを設定します。
  ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。**FORMAT** の値は必ず引用符で囲んでください。

  ```
  /subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
  ```

  注記

  保存されたフォーマッター を参照する場合は named-formatter 属性を使用します。

- 自動フラッシュを設定します。
  毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

  ```
  /subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
  ```

- フィルター式を設定します。
  ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-
attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は フィルター式 の項を参照してください。

Console ログハンドラーのロガーへの割り当て
ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Console ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=CONSOLE_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Console ログハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=CONSOLE_HANDLER_NAME)
```

Console ログハンドラーの削除
ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

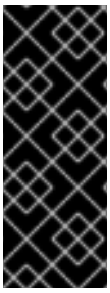
```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:remove
```

11.5.2. File ログハンドラーの設定

ここでは、管理 CLI を使用して File ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Handler → File Handler** と選択すると、File ログハンドラーを設定することができます。

File ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- 新しい File ログハンドラーの追加。
- File ログハンドラーの設定。
- File ログハンドラーのロガーへの割り当て。



重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

File ログハンドラーの追加
File ログハンドラーを追加する場合、**path** および **relative-to** 属性で設定される **file** 属性を使用してファイルパスを指定する必要があります。**path** 属性を使用して名前を含むログのファイルパスを設定します (例: **my-log.log**)。オプションで **relative-to** 属性を使用すると **path** が名前付きのパスと相対的になるよう設定できます (例: **jboss.server.log.dir**)。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:add(file={path=FILE_PATH,relative-
to=RELATIVE_TO_PATH})
```

File ログハンドラーの設定
必要性に応じて、以下の File ログハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できる File ログハンドラー属性の完全リストと説明は、File ログハンドラーの属性 を参照してください。

- ログレベルを設定します。
  ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、ログレベル を参照してください。

  ```
  /subsystem=logging/file-handler=FILE_HANDLER_NAME:write-
  attribute(name=level,value=LEVEL)
  ```

- 追加動作を設定します。
  デフォルトでは、サーバーが再起動されたときに JBoss EAP はログメッセージを同じファイルに追加します。サーバーの再起動時にファイルを上書きする場合は **append** 属性を **false** に設定します。

  ```
  /subsystem=logging/file-handler=FILE_HANDLER_NAME:write-
  attribute(name=append,value=APPEND)
  ```

- エンコーディングを設定します。
  ハンドラーのエンコーディングを設定します (例: **utf-8**)。

  ```
  /subsystem=logging/file-handler=FILE_HANDLER_NAME:write-
  attribute(name=encoding,value=ENCODING)
  ```

- ログフォーマッターを設定します。
  ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。**FORMAT** の値は必ず引用符で囲んでください。

  ```
  /subsystem=logging/file-handler=FILE_HANDLER_NAME:write-
  attribute(name=formatter,value=FORMAT)
  ```

  注記

  保存されたフォーマッター を参照する場合は named-formatter 属性を使用します。

- 自動フラッシュを設定します。
  毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

  ```
  /subsystem=logging/file-handler=FILE_HANDLER_NAME:write-
  attribute(name=autoflush,value=AUTO_FLUSH)
  ```

- フィルター式を設定します。
  ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な

FILTER_EXPRESSION 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=filter-spec,
value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は フィルター式 の項を参照してください。

File ログハンドラーのロガーへの割り当て
ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは File ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=FILE_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに File ログハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=FILE_HANDLER_NAME)
```

File ログハンドラーの削除
ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:remove
```

11.5.3. Periodic rotating ログハンドラーの設定

ここでは、管理 CLI を使用して Periodic rotating ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Handler → Periodic Handler** と選択すると、Periodic ログハンドラーを設定することができます。

Periodic ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- 新しい Periodic ログハンドラーの追加。
- Periodic ログハンドラーの設定。
- Periodic ログハンドラーのロガーへの割り当て

重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

Periodic ログハンドラーの追加
Periodic ログハンドラーを追加する場合、 **path** および **relative-to** 属性で設定される **file** 属性を使用してファイルパスを指定する必要があります。**path** 属性を使用して名前を含むログのファイルパスを設定します (例: **my-log.log**)。オプションで **relative-to** 属性を使用すると **path** が名前付きのパスと相対

的になるよう設定できます (例: **jboss.server.log.dir**)。

また、**suffix** 属性を使用してローテーションしたログの接尾辞を設定する必要もあります。これは、**.yyyy-MM-dd-HH** のように **java.text.SimpleDateFormat** が認識できる形式でなければなりません。ローテーションの周期はこの接尾辞を基に自動的に算出されます。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:add(file=
{path=FILE_PATH,relative-to=RELATIVE_TO_PATH},suffix=SUFFIX)
```

Periodic ログハンドラーの設定
必要性に応じて、以下の Periodic ログハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できる Periodic ログハンドラー属性の完全リストと説明は、Periodic ログハンドラーの属性 を参照してください。

- ログレベルを設定します。
  ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、ログレベル を参照してください。

  ```
  /subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-
  attribute(name=level,value=LEVEL)
  ```

- 追加動作を設定します。
  デフォルトでは、サーバーが再起動されたときに JBoss EAP はログメッセージを同じファイルに追加します。サーバーの再起動時にファイルを上書きする場合は **append** 属性を **false** に設定します。

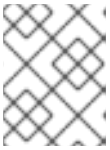
  ```
  /subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-
  attribute(name=append,value=APPEND)
  ```

- エンコーディングを設定します。
  ハンドラーのエンコーディングを設定します (例: **utf-8**)。

  ```
  /subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-
  attribute(name=encoding,value=ENCODING)
  ```

- ログフォーマッターを設定します。
  ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。**FORMAT** の値は必ず引用符で囲んでください。

  ```
  /subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-
  attribute(name=formatter,value=FORMAT)
  ```

  注記

  保存されたフォーマッター を参照する場合は named-formatter 属性を使用します。

- 自動フラッシュを設定します。
  毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- フィルター式を設定します。
  ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

  ```
  /subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
  ```

  利用可能なフィルター式の詳細は フィルター式 の項を参照してください。

Periodic ログハンドラーのロガーへの割り当て
ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Periodic ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=PERIODIC_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Periodic ログハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=PERIODIC_HANDLER_NAME)
```

Periodic ログハンドラーの削除
ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

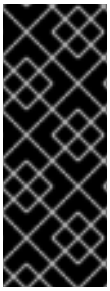
```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:remove
```

11.5.4. Size rotating ログハンドラーの設定

ここでは、管理 CLI を使用して Size rotating ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Handler → Size Handler** と選択すると、Size ログハンドラーを設定することができます。

Size ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- 新しい Size ログハンドラーの追加。
- Size ログハンドラーの設定。
- Size ログハンドラーのロガーへの割り当て



重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

Size ログハンドラーの追加
Size ログハンドラーを追加する場合、**path** および **relative-to** 属性で設定される **file** 属性を使用してファイルパスを指定する必要があります。**path** 属性を使用して名前を含むログのファイルパスを設定します (例: **my-log.log**)。オプションで **relative-to** 属性を使用すると **path** が名前付きのパスと相対的になるよう設定できます (例: **jboss.server.log.dir**)。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:add(file=
{path=FILE_PATH,relative-to=RELATIVE_TO_PATH})
```

Size ログハンドラーの設定
必要性に応じて、以下の Size ログハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できる Size ログハンドラー属性の完全リストと説明は、Size ログハンドラーの属性 を参照してください。

- ログレベルを設定します。
  ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、ログレベル を参照してください。

  ```
  /subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-
  attribute(name=level,value=LEVEL)
  ```

- ローテーションされるログの接尾辞を設定します。
  接尾辞の文字列を設定します。これは **yyyy-MM-dd-HH** のように **java.text.SimpleDateFormat** が認識できる形式でなければなりません。ローテーションの周期はこの接尾辞を基に自動的に算出されます。

  ```
  /subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-
  attribute(name=suffix, value=SUFFIX)
  ```

- ローテーションサイズを設定します。
  ファイルの最大サイズを設定します。この値を超えるとファイルがローテーションされます。デフォルトは 2 メガバイトを意味する **2m** です。

  ```
  /subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-
  attribute(name=rotate-size, value=ROTATE_SIZE)
  ```

- 保持するバックアップログの最大数の設定
  保持するバックアップの数を設定します。デフォルト値は **1** です。

  ```
  /subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-
  attribute(name=max-backup-index, value=MAX_BACKUPS)
  ```

- 起動時にログをローテーションするかどうかを設定します。

デフォルトでは、サーバーの再起動時に新しいログファイルは作成されません。サーバーの再起動時にログをローテーションするには、**true** に設定します。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=rotate-on-boot, value=ROTATE_ON_BOOT)
```

- 追加動作を設定します。
デフォルトでは、サーバーが再起動されたときに JBoss EAP はログメッセージを同じファイルに追加します。サーバーの再起動時にファイルを上書きする場合は **append** 属性を **false** に設定します。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=append,value=APPEND)
```

- エンコーディングを設定します。
ハンドラーのエンコーディングを設定します (例: **utf-8**)。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
```

- ログフォーマッターを設定します。
ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。**FORMAT** の値は必ず引用符で囲んでください。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
```

注記

保存されたフォーマッター を参照する場合は named-formatter 属性を使用します。

- 自動フラッシュを設定します。
毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- フィルター式を設定します。
ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は フィルター式 の項を参照してください。

Size ログハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Size ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=SIZE_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Size ログハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=SIZE_HANDLER_NAME)
```

Size ログハンドラーの削除
ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

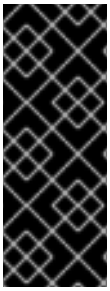
```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:remove
```

11.5.5. Periodic Size rotating ログハンドラーの設定

ここでは、管理 CLI を使用して Periodic Size rotating ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Handler → Periodic Size Handler** と選択すると、Periodic Size ログハンドラーを設定することができます。

Periodic Size ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- 新しい Periodic Size ログハンドラーの追加。
- Periodic Size ログハンドラーの設定。
- Periodic Size ログハンドラーのロガーへの割り当て



重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

Periodic Size ログハンドラーの追加
Periodic Size ログハンドラーを追加する場合、 **path** および **relative-to** 属性で設定される **file** 属性を使用してファイルパスを指定する必要があります。**path** 属性を使用して名前を含むログのファイルパスを設定します (例: **my-log.log**)。オプションで **relative-to** 属性を使用すると **path** が名前付きのパスと相対的になるよう設定できます (例: **jboss.server.log.dir**)。

また、**suffix** 属性を使用してローテーションしたログの接尾辞を設定する必要もあります。これは、**.yyyy-MM-dd-HH** のように **java.text.SimpleDateFormat** が認識できる形式でなければなりません。ローテーションの周期はこの接尾辞を基に自動的に算出されます。

```
/subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:add(file={path=FILE_PATH,relative-to=RELATIVE_TO_PATH},suffix=SUFFIX)
```

-

Periodic Size ログハンドラーの設定
必要性に応じて、以下の Periodic Size ログハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できる Periodic Size ログハンドラー属性の完全リストと説明は、Periodic Size ログハンドラーの属性 を参照してください。

- ログレベルを設定します。
  ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、ログレベル を参照してください。

  ```
  /subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
  ```

- ローテーションサイズを設定します。
  ファイルの最大サイズを設定します。この値を超えるとファイルがローテーションされます。デフォルトは 2 メガバイトを意味する **2m** です。

  ```
  /subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=rotate-size,value=ROTATE_SIZE)
  ```

- 保持するバックアップログの最大数の設定
  保持するバックアップの数を設定します。デフォルト値は **1** です。

  ```
  /subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=max-backup-index,value=MAX_BACKUPS)
  ```

- 起動時にログをローテーションするかどうかを設定します。
  デフォルトでは、サーバーの再起動時に新しいログファイルは作成されません。サーバーの再起動時にログをローテーションするには、**true** に設定します。

  ```
  /subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=rotate-on-boot,value=ROTATE_ON_BOOT)
  ```

- 追加動作を設定します。
  デフォルトでは、サーバーが再起動されたときに JBoss EAP はログメッセージを同じファイルに追加します。サーバーの再起動時にファイルを上書きする場合は **append** 属性を **false** に設定します。

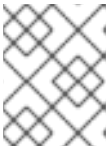
  ```
  /subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=append,value=APPEND)
  ```

- エンコーディングを設定します。
  ハンドラーのエンコーディングを設定します (例: **utf-8**)。

  ```
  /subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
  ```

- ログフォーマッターを設定します。
  ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。**FORMAT** の値は必ず引用符で囲んでください。

  ```
  /subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
  ```

  注記

  保存されたフォーマッター を参照する場合は named-formatter 属性を使用します。

- 自動フラッシュを設定します。
  毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

  ```
  /subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
  ```

- フィルター式を設定します。
  ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

  ```
  /subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
  ```

  利用可能なフィルター式の詳細は フィルター式 の項を参照してください。

Periodic Size ログハンドラーのロガーへの割り当て
ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Periodic Size ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=PERIODIC_SIZE_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Periodic Size ログハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=PERIODIC_SIZE_HANDLER_NAME)
```

Periodic Size ログハンドラーの削除
ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

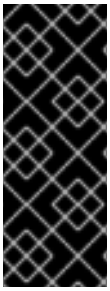
```
/subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:remove
```

11.5.6. Syslog ハンドラーの設定

ここでは、管理 CLI を使用して Syslog ハンドラーを設定する方法を説明します。Syslog ハンドラーを使用すると、Syslog プロトコル (RFC-3164 または RFC-5424) をサポートするリモートロギングサーバーにメッセージを送信できます。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Handler → Syslog Handler** と選択すると、Syslog ハンドラーを設定することができます。

Syslog ハンドラーを設定するために実行する主なタスクは以下のとおりです。

- 新しい Syslog ハンドラーの追加。
- Syslog ハンドラーの設定。
- Syslog ハンドラーのロガーへの割り当て



重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

Syslog ハンドラーの追加

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:add
```

Syslog ハンドラーの設定
必要性に応じて、以下の Syslog ハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できる Syslog ハンドラー属性の完全リストと説明は、Syslog ハンドラーの属性 を参照してください。

- ハンドラーのログレベルを設定します。デフォルトのレベルは **ALL** です。利用できるオプションは、ログレベル を参照してください。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- ログに記録するアプリケーションの名前を設定します。デフォルトの名前は **java** です。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=app-name,value=APP_NAME)
```

- Syslog サーバーのアドレスを設定します。デフォルトのアドレスは **localhost** です。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=server-address,value=SERVER_ADDRESS)
```

- syslog サーバーのポートを設定します。デフォルトのポートは **514** です。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=port,value=PORT)
```

- RFC 仕様の定義どおりに syslog 形式を設定します。デフォルトの形式は **RFC5424** です。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-
attribute(name=syslog-format,value=SYSLOG_FORMAT)
```

- syslog ペイロードのメッセージをフォーマットするために **named-formatter** 属性を指定します。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-
attribute(name=named-formatter, value=FORMATTER_NAME)
```

Syslog ハンドラーのロガーへの割り当て
ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Syslog ハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=SYSLOG_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Syslog ハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=SYSLOG_HANDLER_NAME)
```

Syslog ハンドラーの削除
ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:remove
```

11.5.7. Socket ログハンドラーの設定

ここでは、管理 CLI を使用して Socket ログハンドラーを設定する方法を説明します。ソケットは TCP または UDP ソケットになります。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Handler → Size Handler** と選択すると、Size ログハンドラーの設定も可能になります。

注記

サーバーが管理者専用モードで起動された場合、ログメッセージは破棄されます。

Socket ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- ソケットバインディングを追加します。
- ログフォーマッターを追加します。
- Socket ログハンドラーを追加 し、設定を行います。
- Socket ログハンドラーをロガーに割り当てます。

重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

ソケットバインディングの追加
remote-destination-outbound-socket-binding または **local-destination-outbound-socket-binding** を使用する ソケットバインディング として定義します。

```
/socket-binding-group=SOCKET_BINDING_GROUP/remote-destination-outbound-socket-binding=SOCKET_BINDING_NAME:add(host=HOST, port=PORT)
```

ログフォーマッターの設定
JSON フォーマッターなど、使用する ログフォーマッター を定義します。

```
/subsystem=logging/json-formatter=FORMATTER:add
```

Socket ログハンドラーの追加
Socket ログハンドラーを追加する場合、使用するソケットバインディングとフォーマッターを指定する必要があります。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:add(outbound-socket-binding-ref=SOCKET_BINDING_NAME,named-formatter=FORMATTER)
```

Socket ログハンドラーの設定
必要性に応じて、以下の Socket ログハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できる Socket ログハンドラー属性の完全リストと説明は、Socket ログハンドラーの属性 を参照してください。

- プロトコルを設定します。
  使用するプロトコルを設定します。デフォルトは **TCP** です。

  ```
  /subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=protocol,value=PROTOCOL)
  ```

- ログレベルを設定します。
  ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、ログレベル を参照してください。

  ```
  /subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
  ```

注記

サーバーの起動中、Socket ログハンドラーによって処理されるログメッセージは、ソケットバインディングが設定され、**logging** サブシステムが初期化されるまでキューに置かれます。ログレベルが **TRACE** や **DEBUG** などの低レベルに設定されている場合、起動時に大量のメモリーが消費されることがあります。

- エンコーディングを設定します。
  ハンドラーのエンコーディングを設定します (例: **utf-8**)。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
```

- 自動フラッシュを設定します。
  毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- フィルター式を設定します。
  ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

  利用可能なフィルター式の詳細は フィルター式 の項を参照してください。

Socket ログハンドラーのロガーへの割り当て
ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Socket ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=SOCKET_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Socket ログハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=SOCKET_HANDLER_NAME)
```

Socket ログハンドラーの削除
ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:remove
```

SSL/TLS を使用したソケット上でのログメッセージの送信
次の手順は、**SSL_TCP** プロトコルを使用してソケット上でログメッセージを送信するように Socket ログハンドラーを設定する方法の例を示しています。この例では、Socket ログハンドラーによって使用される **elytron** サブシステムのキーストア、トラストマネージャー、およびクライアント SSL コンテキストを設定します。ルートロガーからのログメッセージは指定のソケット上で送信され、JSON フォーマッターによってフォーマットされます。

Elytron コンポーネントの設定に関する詳細は、JBoss EAP**How to Configure Server Security**の Elytron Subsystem を参照してください。

1. Elytron を設定します。

a. キーストアを追加します。

```
/subsystem=elytron/key-store=log-server-ks:add(path=/path/to/keystore.jks, type=JKS,
credential-reference={clear-text=mypassword})
```

b. トラストマネージャーを追加します。

```
/subsystem=elytron/trust-manager=log-server-tm:add(key-store=log-server-ks)
```

c. クライアント SSL コンテキストを追加します。

```
/subsystem=elytron/client-ssl-context=log-server-context:add(trust-manager=log-server-
tm, protocols=["TLSv1.2"])
```

2. ソケットバインディングを追加します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=log-
server:add(host=localhost, port=4560)
```

3. JSON フォーマッターを追加します。

```
/subsystem=logging/json-formatter=json:add
```

4. Socket ログハンドラーを追加します。

```
/subsystem=logging/socket-handler=log-server-handler:add(named-formatter=json,
level=INFO, outbound-socket-binding-ref=log-server, protocol=SSL_TCP, ssl-context=log-
server-context)
```

5. ログハンドラーをルートロガーに割り当てます。

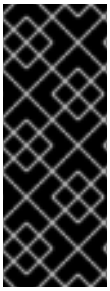
```
/subsystem=logging/root-logger=ROOT:add-handler(name=log-server-handler)
```

11.5.8. Custom ログハンドラーの設定

ここでは、管理 CLI を使用してカスタムログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Handler → Custom Handler** と選択すると、カスタムログハンドラーを設定することができます。

カスタムログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- 新しいカスタムログハンドラーの追加。
- カスタムログハンドラーの設定。
- カスタムログハンドラーのロガーへの割り当て。

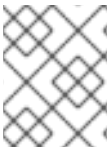


重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

Custom ログハンドラーの追加
カスタムログハンドラーを追加する場合、ハンドラーの Java クラスとハンドラーが含まれる JBoss EAP モジュールを指定する必要があります。クラスは **java.util.logging.Handler** を拡張する必要があります。



注記

すでに、カスタムロガーが含まれる モジュールが作成 されている必要があります。 作成されていないと、このコマンドの実行に失敗します。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:add(class=CLASS_NAME,module=MODULE_NAME)
```

Custom ログハンドラーの設定
必要性に応じて、以下のカスタムログハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できるカスタムログハンドラー属性の完全リストと説明は、Custom ログハンドラーの属性 を参照してください。

- ログレベルを設定します。
  ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、ログレベル を参照してください。

  ```
  /subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
  ```

- プロパティーを設定します。
  ログハンドラーに必要なプロパティーを設定します。setter メソッドを使用してプロパティーにアクセスできなければなりません。

  ```
  /subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-attribute(name=properties.PROPERTY_NAME,value=PROPERTY_VALUE)
  ```

- エンコーディングを設定します。
  ハンドラーのエンコーディングを設定します (例: **utf-8**)。

  ```
  /subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
  ```

- ログフォーマッターを設定します。
  ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。**FORMAT** の値は必ず引用符で囲んでください。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
```

注記

保存されたフォーマッター を参照する場合は named-formatter 属性を使用します。

- フィルター式を設定します。
  ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

  ```
  /subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
  ```

  利用可能なフィルター式の詳細は フィルター式 の項を参照してください。

Custom ログハンドラーのロガーへの割り当て
ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは、カスタムログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=CUSTOM_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Custom ログハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=CUSTOM_HANDLER_NAME)
```

Custom ログハンドラーの削除
ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:remove
```

11.5.9. Async ログハンドラーの設定

ここでは、管理 CLI を使用して Async ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Handler → Async Handler** と選択すると、Async ログハンドラーを設定することができます。

Async ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- 新しい Async ログハンドラーの追加。
- サブハンドラーの Async ログハンドラーへの追加。
- Async ログハンドラーの設定。
- Async ログハンドラーのロガーへの割り当て

重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

Async ログハンドラーの追加
Async ログハンドラーを追加するときにキューの長さを指定する必要があります。これは、キューに保持できるログリクエストの最大数のことです。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:add(queue-length=QUEUE_LENGTH)
```

サブハンドラーの追加
1 つ以上のハンドラーを Async ログハンドラーのサブハンドラーとして追加できます。ハンドラーが設定に存在しないと、このコマンドの実行に失敗するため注意してください。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:add-handler(name=HANDLER_NAME)
```

Async ログハンドラーの設定
必要性に応じて、以下の Async ログハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できる Async ログハンドラー属性の完全リストと説明は、Async ログハンドラーの属性 を参照してください。

- ログレベルを設定します。
  ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、ログレベル を参照してください。

  ```
  /subsystem=logging/async-handler=ASYNC_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
  ```

- オーバーフローアクションを設定します。
  オーバーフローが発生したときに行うアクションを設定します。デフォルトの値は **BLOCK** で、キューが満杯になるとスレッドがブロックされます。この値を **DISCARD** に変更すると、キューが満杯になったときにログメッセージが破棄されます。

  ```
  /subsystem=logging/async-handler=ASYNC_HANDLER_NAME:write-attribute(name=overflow-action,value=OVERFLOW_ACTION)
  ```

- フィルター式を設定します。
  ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

  ```
  /subsystem=logging/async-handler=ASYNC_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
  ```

  利用可能なフィルター式の詳細は フィルター式 の項を参照してください。

Async ログハンドラーのロガーへの割り当て
ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Async ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=ASYNC_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Async ログハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-handler(name=ASYNC_HANDLER_NAME)
```

Async ログハンドラーの削除
ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーに割り当てられている場合は削除できません。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:remove
```

11.6. ルートロガーの設定

ルートロガーは、ログカテゴリーによってキャプチャーされないサーバーへ送信されたすべてのログメッセージ (指定のログレベル以上) をキャプチャーします。

ここでは、管理 CLI を使用してルートロガーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、表示 をクリックして **Root Logger** を選択すると、ルートロガーを設定することができます。

ルートロガーの設定

重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

1. ログハンドラーをルートロガーへ割り当てます。
   ログハンドラーを追加します。

   ```
   /subsystem=logging/root-logger=ROOT:add-handler(name=LOG_HANDLER_NAME)
   ```

   ログハンドラーの削除

   ```
   /subsystem=logging/root-logger=ROOT:remove-handler(name=LOG_HANDLER_NAME)
   ```

2. ログレベルを設定します。

   ```
   /subsystem=logging/root-logger=ROOT:write-attribute(name=level,value=LEVEL)
   ```

使用できるルートロガー属性とその説明の完全リストは、ルートロガーの属性 を参照してください。

11.7. ログフォーマッターの設定

ログフォーマッターはハンドラーでのログメッセージの形式を定義します。**logging** サブシステムでは以下のログフォーマッターを設定できます。

- パターンフォーマッター
- JSON ログフォーマッター
- XML ログフォーマッター
- カスタムログフォーマッター

11.7.1. パターンフォーマッターの設定

ログハンドラーすべてで使用できる名前付きパターンフォーマッターを作成して、ログメッセージをフォーマットすることができます。

重要

ロギングプロファイルにこのログフォーマッターを設定する場合、 コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

パターンフォーマッターの作成
パターンフォーマッターを定義するとき、ログメッセージのフォーマットに使用するパターン文字列を指定します。パターン構文の詳細は、パターンフォーマッターのフォーマット文字 を参照してください。

```
/subsystem=logging/pattern-formatter=PATTERN_FORMATTER_NAME:add(pattern=PATTERN)
```

たとえば、デフォルトの設定はサーバーログへのロギングメッセージのログフォーマッター文字列として **%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** を使用します。これにより、以下のようにフォーマットされるログメッセージが作成されます。

```
2016-03-18 15:49:32,075 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin console listening on http://127.0.0.1:9990
```

また、カラーマップを定義してログレベルごとに色を割り当てることもできます。形式は **LEVEL:COLOR** のコンマ区切りリストです。

- 有効なレベル: **finest**、 **finer**、 **fine**、 **config**、 **trace**、 **debug**、 **info**、 **warning**、 **warn**、 **error**、 **fatal**、 **severe**
- 有効な色: **black**、 **green**、 **red**、 **yellow**、 **blue**、 **magenta**、 **cyan**、 **white**、 **brightblack**、**brightred**、 **brightgreen**、 **brightblue**、 **brightyellow**、 **brightmagenta**、 **brightcyan**、 **brightwhite**

```
/subsystem=logging/pattern-formatter=PATTERN_FORMATTER_NAME:write-attribute(name=color-map,value="LEVEL:COLOR,LEVEL:COLOR")
```

管理コンソールを使用してパターンログフォーマッターを設定することもできます。

1. ブラウザーで管理コンソールを開きます。
2. **Configuration → Subsystems → Logging** と選択します。
3. **Configuration** を選択し、表示 をクリックします。
4. **Formatter** を選択し、**Pattern Formatter** オプションを選択します。

11.7.2. カスタムログフォーマッターの設定

JSON 形式でログメッセージをフォーマットする JSON ログフォーマッターを作成できます。

重要

ロギングプロファイルにこのログフォーマッターを設定する場合、 コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

JSON ログフォーマッターの追加

```
/subsystem=logging/json-formatter=JSON_FORMATTER_NAME:add(pretty-print=true, exception-output-type=formatted)
```

これにより、以下のようにフォーマットされるログメッセージが作成されます。

```
{
    "timestamp": "2018-10-18T13:53:43.031-04:00",
    "sequence": 62,
    "loggerClassName": "org.jboss.as.server.logging.ServerLogger_$logger",
    "loggerName": "org.jboss.as",
    "level": "INFO",
    "message": "WFLYSRV0025: JBoss EAP 7.4.0.GA (WildFly Core 15.0.2.Final-redhat-00001) started in 5227ms - Started 317 of 556 services (343 services are lazy, passive or on-demand),
    "threadName": "Controller Boot Thread",
    "threadId": 22,
    "mdc": {
    },
    "ndc": "",
    "hostName": "localhost.localdomain",
    "processName": "jboss-modules.jar",
    "processId": 7461
}
```

Logstash JSON ログフォーマッターの追加

注記

JSON ログフォーマッター出力キーを変更し、静的メタデータを追加できます。JSON ログフォーマッターの主な目的は、ログメッセージを JSON 形式でフォーマットすることです。Logstash はこの JSON 出力を消費し、**@timestamp** および **@version** のフィールドを検索します。以下の例では、Logstash のメッセージをフォーマットする JSON ログフォーマッターを作成します。

```
/subsystem=logging/json-formatter=logstash:add(exception-output-type=formatted, key-overrides=[timestamp="@timestamp"], meta-data=[@version=1])
```

JSON フォーマッター属性は以下の説明どおりに使用できます。

- **key-overrides** 属性は、定義されたキーの名前をオーバーライドするために使用できます。
- **exception-output-type** 属性を **formatted** に設定すると、例外をオブジェクトとしてフォーマットできます。
- **exception-output-type** 属性を **detailed** に設定すると、例外スタックトレースを含めることができます。
- **exception-output-type** 属性を **detailed-and-formatted** に設定すると、例外をオブジェクトとしてフォーマットし、スタックトレースを含めることができます。
- **meta-data** 属性を使用すると、メタデータをログレコードに追加できます。

JSON フォーマッター属性の詳細は JSON ログフォーマッター属性 を参照してください。

管理コンソールを使用して JSON ログフォーマッターを設定することもできます。

1. ブラウザーで管理コンソールを開きます。
2. **Configuration → Subsystems → Logging** と選択します。
3. **Configuration** を選択し、表示 をクリックします。
4. **Formatter** を選択し、**JSON Formatter** オプションを選択します。

11.7.3. XML ログフォーマッターの設定

XML 形式でログメッセージをフォーマットする XML ログフォーマッターを作成できます。

重要

ロギングプロファイルにこのログフォーマッターを設定する場合、 コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

XML ログフォーマッターの追加

```
/subsystem=logging/xml-formatter=XML_FORMATTER_NAME:add(pretty-print=true, exception-output-type=detailed-and-formatted)
```

これにより、以下のようにフォーマットされるログメッセージが作成されます。

```
<record>
  <timestamp>2018-10-18T13:55:53.419-04:00</timestamp>
  <sequence>62</sequence>
  <loggerClassName>org.jboss.as.server.logging.ServerLogger_$logger</loggerClassName>
  <loggerName>org.jboss.as</loggerName>
  <level>INFO</level>
  <message>WFLYSRV0025: {ProductCurrentVersionExamples} (WildFly Core 10.0.0.Final-redhat-20190924) started in 6271ms - Started 495 of 679 services (331 services are lazy, passive or on-demand)</message>
  <threadName>Controller Boot Thread</threadName>
  <threadId>22</threadId>
  <mdc>
  </mdc>
  <ndc></ndc>
  <hostName>localhost.localdomain</hostName>
  <processName>jboss-modules.jar</processName>
  <processId>7790</processId>
</record>
```

キーオーバーライド **XML** ログフォーマッターの追加

```
/subsystem=logging/xml-formatter=XML_FORMATTER_NAME:add(pretty-print=true, print-namespace=true, namespace-uri="urn:custom:1.0", key-overrides={message=msg, record=logRecord, timestamp=date}, print-details=true)
```

XML フォーマッター属性は以下の説明どおりに使用できます。

- **key-overrides** 属性は、定義されたキーの名前をオーバーライドするために使用できます。
- **exception-output-type** 属性を **formatted** に設定すると、例外をオブジェクトとしてフォーマットできます。
- **exception-output-type** 属性を **detailed** に設定すると、例外スタックトレースを含めることができます。
- **exception-output-type** 属性を **detailed-and-formatted** に設定すると、例外をオブジェクトとしてフォーマットし、スタックトレースを含めることができます。
- **meta-data** 属性を使用すると、メタデータをログレコードに追加できます。

XML フォーマッター属性の詳細は XML ログフォーマッター属性 を参照してください。

管理コンソールを使用して XML ログフォーマッターを設定することもできます。

1. ブラウザーで管理コンソールを開きます。
2. **Configuration** → **Subsystems** → **Logging** と選択します。
3. **Configuration** を選択し、表示 をクリックします。
4. **Formatter** を選択し、**XML Formatter** オプションを選択します。

11.7.4. Custom ログフォーマッターの設定

ログハンドラーすべてで使用できるカスタムログフォーマッターを作成して、ログメッセージをフォーマットすることができます。

ここでは、管理 CLI を使用してカスタムログフォーマッターを設定する方法を説明します。

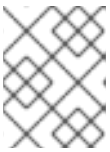
Custom ログフォーマッターの設定

重要

ロギングプロファイルにこのログフォーマッターを設定する場合、 コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/** になります。

さらに、マネージドドメインで実行している場合はコマンドの前に **/profile=PROFILE_NAME** を付けます。

1. カスタムログフォーマッターを追加します。
   カスタムログフォーマッターを追加する場合、フォーマッターの Java クラスとフォーマッターが含まれる JBoss EAP モジュールを指定する必要があります。クラスは **java.util.logging.Formatter** を拡張する必要があります。

   注記

   すでに、カスタムフォーマッターが含まれるモジュールが作成されている必要があります。 作成されていないと、このコマンドの実行に失敗します。

   ```
   /subsystem=logging/custom-formatter=CUSTOM_FORMATTER_NAME:add(class=CLASS_NAME,
   module=MODULE_NAME)
   ```

2. ログフォーマッターに必要なプロパティーを設定します。
   setter メソッドを使用してプロパティーにアクセスできなければなりません。

   ```
   /subsystem=logging/custom-formatter=CUSTOM_FORMATTER_NAME:write-
   attribute(name=properties.PROPERTY_NAME,value=PROPERTY_VALUE)
   ```

3. カスタムフォーマッターをログハンドラーに割り当てます。
   以下の管理 CLI コマンドは、Periodic Rotating ファイルハンドラーによって使用されるカスタムフォーマッターを割り当てます。

   ```
   /subsystem=logging/periodic-rotating-file-handler=FILE_HANDLER_NAME:write-
   attribute(name=named-formatter, value=CUSTOM_FORMATTER_NAME)
   ```

カスタム **XML** フォーマッターの例
以下の例は、カスタム XML フォーマッターを設定します。**org.jboss.logmanager** モジュールに提供される **java.util.logging.XMLFormatter** クラスを使用し、Console ログハンドラーに割り当てます。

```
/subsystem=logging/custom-formatter=custom-xml-
formatter:add(class=java.util.logging.XMLFormatter, module=org.jboss.logmanager)
/subsystem=logging/console-handler=CONSOLE:write-attribute(name=named-formatter,
value=custom-xml-formatter)
```

このフォーマッターを使用するログメッセージは以下のようにフォーマットされます。

```
<record>
  <date>2016-03-23T12:58:13</date>
  <millis>1458752293091</millis>
  <sequence>93963</sequence>
  <logger>org.jboss.as</logger>
  <level>INFO</level>
  <class>org.jboss.as.server.BootstrapListener</class>
  <method>logAdminConsole</method>
  <thread>22</thread>
  <message>WFLYSRV0051: Admin console listening on http://%s:%d</message>
  <param>127.0.0.1</param>
  <param>9990</param>
</record>
```

管理コンソールを使用したカスタムログフォーマッターの設定
管理コンソールを使用してカスタムログフォーマッターを設定することもできます。

1. ブラウザーで管理コンソールを開きます。
2. **Configuration → Subsystems → Logging** と選択します。
3. **Configuration** を選択し、表示 をクリックします。
4. **Formatter** を選択し、**Custom Formatter** オプションを選択します。

11.8. アプリケーションのロギング

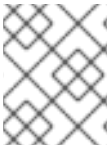
アプリケーションのロギングは、JBoss EAP の **logging** サブシステムを使用するか、デプロイメントごとに設定できます。

ログメッセージの取得に JBoss EAP ログカテゴリーおよびハンドラーを使用する方法は Logging サブシステム を参照してください。

サポートされるアプリケーションロギングフレームワークやデプロイメントごとのロギング設定など、アプリケーションロギングの詳細は JBoss EAP**Development Guide**の Logging の章を参照してください。

11.8.1. デプロイメントごとのロギング

デプロイメントごとのロギングを使用すると、開発者はアプリケーションのロギング設定を事前に設定できます。アプリケーションがデプロイされると、定義された設定に従ってロギングが開始されます。この設定によって作成されたログファイルにはアプリケーションの動作に関する情報のみが含まれます。



注記

デプロイメントごとのロギング設定が行われない場合、すべてのアプリケーションとサーバーには **logging** サブシステムの設定が使用されます。

この方法では、システム全体のロギングを使用する利点と欠点があります。利点は、JBoss EAP インスタンスの管理者がサーバーロギング以外のロギングを設定する必要がないことです。欠点は、デプロイメントごとのロギング設定はサーバーの起動時に読み取り専用であるため、実行時に変更できないこと

です。

アプリケーションでデプロイメントごとのロギングを使用する手順については、JBoss EAP**Development Guide**の Add Per-deployment Logging to an Application を参照してください。

11.8.1.1. デプロイメントごとのロギングの無効化

以下の方法の 1 つを使用するとデプロイメントごとのロギングを無効にできます。

- **use-deployment-logging-config** 属性を **false** に設定します。
  use-deployment-logging-config 属性は、デプロイメントがデプロイメントごとにロギングに対してスキャンされるかどうかを制御します。デフォルトは **true** です。デプロイメントごとのロギングを無効にするにはこの属性を **false** に設定します。

  ```
  /subsystem=logging:write-attribute(name=use-deployment-logging-config,value=false)
  ```

- **jboss-deployment-structure.xml** ファイルを使用して **logging** サブシステムを除外します。
  手順については、JBoss EAP**Development Guide**の Exclude a Subsystem from a Deployment を参照してください。

11.8.2. ロギングプロファイル

ロギングプロファイルは、デプロイされたアプリケーションに割り当てることができる独立したロギング設定のセットです。通常の **logging** サブシステム同様にロギングプロファイルはハンドラー、カテゴリー、およびルートロガーを定義できますが、他のプロファイルや主要な **logging** サブシステムを参照できません。設定が容易である点でロギングプロファイルは **logging** サブシステムと似ています。

ロギングプロファイルを使用すると、管理者は他のロギング設定に影響を与えずに 1 つ以上のアプリケーションに固有するロギング設定を作成することができます。各プロファイルはサーバー設定で定義されるため、ロギング設定を変更しても影響を受けるアプリケーションを再デプロイする必要はありません。

各ロギングプロファイルには以下の項目を設定できます。

- 一意な名前。この値は必須です。
- 任意の数のログハンドラー。
- 任意の数のログカテゴリー。
- 最大 1 つのルートロガー。

アプリケーションでは **Logging-Profile** 属性を使用して、**MANIFEST.MF** ファイルで使用するロギングプロファイルを指定できます。

11.8.2.1. ロギングプロファイルの設定

ロギングプロファイルは、ログハンドラー、カテゴリー、およびルートロガーで設定できます。ロギングプロファイルの設定には、**logging** サブシステムの設定と同じ構文を使用しますが、以下の点が異なります。

- ルート設定パスが **/subsystem=logging/logging-profile=NAME** になります。
- ロギングプロファイルに他のロギングプロファイルを追加できません。

- **logging** サブシステムには、ロギングプロファイルに使用できない以下の属性があります。
  - **add-logging-api-dependencies**
  - **use-deployment-logging-config**

ロギングプロファイルの作成および設定
以下の手順では、管理 CLI を使用してロギングプロファイルを作成し、ファイルハンドラーとロガーカテゴリーを設定します。管理コンソールでは **Configuration → Subsystems → Logging → Logging Profiles** と選択するとロギングプロファイルを設定することができます。

1. ロギングプロファイルを作成します。

```
/subsystem=logging/logging-profile=PROFILE_NAME:add
```

2. ファイルハンドラーを作成します。

```
/subsystem=logging/logging-profile=PROFILE_NAME/file-handler=FILE_HANDLER_NAME:add(file={path=>"LOG_NAME.log", "relative-to"=>"jboss.server.log.dir"})
```

```
/subsystem=logging/logging-profile=PROFILE_NAME/file-handler=FILE_HANDLER_NAME:write-attribute(name="level", value="DEBUG")
```

ファイルハンドラー属性のリストは、File ログハンドラーの属性 を参照してください。

3. ロガーカテゴリーを作成します。

```
/subsystem=logging/logging-profile=PROFILE_NAME/logger=CATEGORY_NAME:add(level=TRACE)
```

ログカテゴリー属性のリストは、ログカテゴリーの属性 を参照してください。

4. ファイルハンドラーをカテゴリーに割り当てます。

```
/subsystem=logging/logging-profile=PROFILE_NAME/logger=CATEGORY_NAME:add-handler(name="FILE_HANDLER_NAME")
```

この後、アプリケーションによって使用されるロギングプロファイルを **MANIFEST.MF** ファイルに設定できます。詳細は、JBoss EAP**Development Guide**の Specify a Logging Profile in an Application を参照してください。

11.8.2.2. ロギングプロファイル設定の例

この例は、ロギングプロファイルとそれを使用するアプリケーションの設定を表しています。管理 CLI コマンド、結果となる XML、およびアプリケーションの **MANIFEST.MF** が示されています。

ロギングプロファイルの例には次のような特徴があります。

- 名前は **accounts-app-profile** です。
- ログカテゴリーは **com.company.accounts.ejbs** です。
- ログレベルは **TRACE** です。

- ログハンドラーは、**ejb-trace.log** ファイルを使用するファイルハンドラーです。

管理 CLI セッション

```
/subsystem=logging/logging-profile=accounts-app-profile:add

/subsystem=logging/logging-profile=accounts-app-profile/file-handler=ejb-trace-file:add(file=
{path=>"ejb-trace.log", "relative-to"=>"jboss.server.log.dir"})

/subsystem=logging/logging-profile=accounts-app-profile/file-handler=ejb-trace-file:write-
attribute(name="level", value="DEBUG")

/subsystem=logging/logging-profile=accounts-app-
profile/logger=com.company.accounts.ejbs:add(level=TRACE)

/subsystem=logging/logging-profile=accounts-app-profile/logger=com.company.accounts.ejbs:add-
handler(name="ejb-trace-file")
```

XML 設定

```
<logging-profiles>
  <logging-profile name="accounts-app-profile">
    <file-handler name="ejb-trace-file">
      <level name="DEBUG"/>
      <file relative-to="jboss.server.log.dir" path="ejb-trace.log"/>
    </file-handler>
    <logger category="com.company.accounts.ejbs">
      <level name="TRACE"/>
      <handlers>
        <handler name="ejb-trace-file"/>
      </handlers>
    </logger>
  </logging-profile>
</logging-profiles>
```

アプリケーションの MANIFEST.MF ファイル

```
Manifest-Version: 1.0
Logging-Profile: accounts-app-profile
```

11.8.3. デプロイメントロギング設定の表示

以下の管理 CLI コマンドを使用すると、特定のデプロイメントのロギング設定に関する情報を取得できます。

```
/deployment=DEPLOYMENT_NAME/subsystem=logging/configuration=CONFIG:read-resource
```

デプロイメントのロギング設定値である **CONFIG** には、以下の 3 つの値の 1 つを指定します。

- デプロイメントが **logging**logging サブシステム を使用する場合は **default** を指定します。これにより、**logging** サブシステムの設定が出力されます。

- デプロイメントが **logging** サブシステムに定義されている ロギングプロファイル を使用する場合は、**profile-PROFILE_NAME** を指定します。これにより、ロギングプロファイルの設定が出力されます。
- 使用される設定ファイルへのパス (例: **myear.ear/META-INF/logging.properties**) を指定します。

たとえば、以下の管理 CLI コマンドは、特定のデプロイメントによって使用される **MYPROFILE** ロギングプロファイルの設定を表示します。

```
/deployment=mydeployment.war/subsystem=logging/configuration=profile-MYPROFILE:read-resource(recursive=true,include-runtime=true)
```

以下の情報が出力されます。

```
{
    "outcome" => "success",
    "result" => {
        "error-manager" => undefined,
        "filter" => undefined,
        "formatter" => {
            "MYFORMATTER" => {
                "class-name" => "org.jboss.logmanager.formatters.PatternFormatter",
                "module" => undefined,
                "properties" => {"pattern" => "%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n"}
            }
        },
        "handler" => {
            "MYPERIODIC" => {
                "class-name" => "org.jboss.logmanager.handlers.PeriodicRotatingFileHandler",
                "encoding" => undefined,
                "error-manager" => undefined,
                "filter" => undefined,
                "formatter" => "MYFORMATTER",
                "handlers" => [],
                "level" => "ALL",
                "module" => undefined,
                "properties" => {
                    "append" => "true",
                    "autoFlush" => "true",
                    "enabled" => "true",
                    "suffix" => ".yyyy-MM-dd",
                    "fileName" => "EAP_HOME/standalone/log/deployment.log"
                }
            }
        },
        "logger" => {"MYCATEGORY" => {
            "filter" => undefined,
            "handlers" => [],
            "level" => "DEBUG",
            "use-parent-handlers" => true
        }},
        "pojo" => undefined
    }
}
```

また、再帰的な **read-resource** 操作を使用して、ロギング設定やデプロイメントに関する他の情報を取得することができます。

```
/deployment=DEPLOYMENT_NAME/subsystem=logging:read-resource(include-runtime=true,
recursive=true)
```

11.9. LOGGING サブシステムの調整

logging サブシステムのパフォーマンスを監視および最適化するための情報は、**Performance Tuning Guide**の Logging Subsystem Tuning の項を参照してください。

第12章 データソース管理

12.1. JBOSS EAP データソース

JDBC

JDBC API は、Java アプリケーションがデータベースにアクセスする方法を定義する基準です。アプリケーションは JDBC ドライバーを参照するデータソースを設定します。その後、データベースではなくドライバーに対してアプリケーションを記述できます。ドライバーはコードをデータベース言語に変換します。そのため、適切なドライバーがインストールされていればアプリケーションをサポートされるデータベースで使用できます。

詳細は JDBC の仕様 を参照してください。

サポートされているデータベース

JBoss EAP 7 によってサポートされる JDBC 対応データベースのリストは、JBoss EAP 7 でサポートされる設定 を参照してください。

データソースタイプ

リソースの一般的なタイプには、データソースと **XA** データソースの 2 つのタイプがあります。

非 **XA** データソース

トランザクションを使用しないアプリケーション、または単一のデータベースでトランザクションを使用するアプリケーションに使用されます。

XA データソース

複数のデータベースまたはある XA トランザクションの一部として他の XA リソースを使用するアプリケーションによって使用されます。XA データソースを使用すると追加のオーバーヘッドが発生します。

JBoss EAP 管理インターフェイスを使用してデータソースを作成するときに、使用するデータソースのタイプを指定します。

ExampleDS データソース

JBoss EAP には、データソースの定義方法を実証するために提供されるデータソース設定例 **ExampleDS** が含まれています。このデータソースは、 H2 データベースを使用します。 H2 データベースはライトウェイトなリレーショナルデータベース管理システムで、アプリケーションを迅速に構築できる開発者向けの機能を提供します。

警告

ExampleDS データソースと H2 データベースは本番環境で使用 しないでください。これは、アプリケーションのテストおよび構築に必要なすべての標準をサポートする非常に小さい自己完結型のデータソースであり、本番稼働用として堅牢またはスケーラブルではありません。

12.2. JDBC ドライバー

JBoss EAP でアプリケーションが使用するデータソースを定義する前に、最初に適切な JDBC ドライバーをインストールする必要があります。

12.2.1. コアモジュールとしての JDBC ドライバーのインストール

JDBC ドライバーをコアモジュールとしてインストールするには、最初に JDBC ドライバーをコアモジュールとして追加 し、datasources サブシステムで **JDBC** ドライバーを登録 する必要があります。

12.2.1.1. JDBC ドライバーをコアモジュールとして追加

以下の手順に従うと、管理 CLI を使用して JDBC ドライバーをコアモジュールとしてインストールすることができます。

1. JDBC ドライバーをダウンロードします。
   データベースのベンダーから適切な JDBC ドライバーをダウンロードします。一般的なデータベースの JDBC ドライバーをダウンロードできる場所については、JDBC ドライバーのダウンロードできる場所 を参照してください。

   JDBC ドライバーの JAR ファイルが ZIP または TAR アーカイブ内に含まれている場合は、必ずそのアーカイブをデプロイメントしてください。

2. JBoss EAP サーバーを起動します。

3. 管理 CLI を起動します。

   ```
   $ EAP_HOME/bin/jboss-cli.sh
   ```

4. **module add** 管理 CLI コマンドを使用して新しいコアモジュールを追加します。

   ```
   [disconnected /] module add --name=MODULE_NAME --resources=PATH_TO_JDBC_JAR
   --dependencies=DEPENDENCIES
   ```

   例

   次のコマンドは、MySQL JDBC ドライバーモジュールを追加します。

   ```
   [disconnected /] module add --name=com.mysql --resources=/path/to/mysql-connector-java-
   8.0.12.jar --dependencies=javax.transaction.api,sun.jdk,ibm.jdk,javaee.api,javax.api
   ```

   例

   管理 CLI を起動して新しいコアモジュールを 1 ステップで追加するには、次のコマンドを使用します。

   ```
   $ EAP_HOME/bin/jboss-cli.sh --command="module add --name=MODULE_NAME --
   resources=PATH_TO_JDBC_JAR --dependencies=DEPENDENCIES"
   ```

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

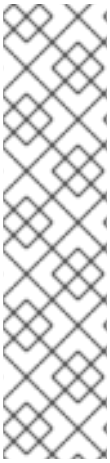
module --help を実行すると、このコマンドを使用したモジュールの追加および削除の詳細を表示できます。

次に、アプリケーションデータソースによって参照されるよう、JDBC ドライバーとして登録する必要があります。

12.2.1.2. JDBC ドライバーの登録

ドライバーが コアモジュールとしてインストール されたら、以下の管理 CLI コマンドを使用して JDBC ドライバーとして登録する必要があります。マネージドドメインを実行している場合は、コマンドの前に **/profile=PROFILE_NAME** を付けてください。

```
/subsystem=datasources/jdbc-driver=DRIVER_NAME:add(driver-name=DRIVER_NAME,driver-module-name=MODULE_NAME,driver-xa-datasource-class-name=XA_DATASOURCE_CLASS_NAME, driver-class-name=DRIVER_CLASS_NAME)
```



注記

driver-class-name パラメーターは、JDBC ドライバー jar が **/META-INF/services/java.sql.Driver** ファイルで複数の jar を定義する場合のみ必要です。

たとえば、MySQL 5.1.36 JDBC ドライバー JAR の **/META-INF/services/java.sql.Driver** ファイルは、以下の 2 つのクラスを定義します。

- com.mysql.cj.jdbc.Driver
- com.mysql.fabric.jdbc.FabricMySQLDriver

この場合、**driver-class-name=com.mysql.cj.jdbc.Driver** で渡します。

たとえば、以下のコマンドは MySQL JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-name=mysql,driver-module-name=com.mysql,driver-xa-datasource-class-name=com.mysql.cj.jdbc.MysqlXADataSource, driver-class-name=com.mysql.cj.jdbc.Driver)
```

アプリケーションのデータソースが JDBC ドライバーを参照できる状態になります。

12.2.2. JDBC ドライバーの JAR デプロイメントとしてのインストール

管理 CLI または管理コンソールを使用して JDBC ドライバーを JAR デプロイメントとしてインストールできます。JDBC 4 に対応するドライバーは、自動的に認識され、デプロイメント時に JDBC ドライバーとしてインストールされます。

以下の手順は、管理 CLI を使用した JDBC ドライバーのインストール方法になります。

注記

JDBC ドライバーを コアモジュール としてインストールする方法が推奨されます。

1. JDBC ドライバーをダウンロードします。
   データベースのベンダーから適切な JDBC ドライバーをダウンロードします。一般的なデータベースの JDBC ドライバーをダウンロードできる場所については、JDBC ドライバーのダウンロードできる場所 を参照してください。

   JDBC ドライバーの JAR ファイルが ZIP または TAR アーカイブ内に含まれている場合は、必ずそのアーカイブをデプロイメントしてください。

2. JDBC ドライバーが JDBC 4 に対応していない場合は、JDBC ドライバー JAR を JDBC 4 対応に更新 の手順を参照してください。

3. JAR を JBoss EAP にデプロイします。

   ```
   deploy PATH_TO_JDBC_JAR
   ```

   注記

   マネージドドメインでは、適切なサーバーグループを指定します。

   たとえば、以下のコマンドは MySQL JDBC ドライバーをデプロイします。

   ```
   deploy /path/to/mysql-connector-java-8.0.12.jar
   ```

   JBoss EAP サーバーログにメッセージが表示され、データソースを定義するときに使用されるデプロイされたドライバーの名前が表示されます。

   ```
   WFLYJCA0018: Started Driver service with driver-name = mysql-connector-java-8.0.12.jar
   ```

アプリケーションのデータソースが JDBC ドライバーを参照できる状態になります。

JDBC ドライバー JAR を JDBC 4 対応に更新
JDBC ドライバー JAR が JDBC 4 に対応していない場合、以下の手順に従ってデプロイ可能にすることができます。

1. 空の一時ディレクトリーを作成します。
2. **META-INF** サブディレクトリーを作成します。
3. **META-INF/services** サブディレクトリーを作成します。
4. **META-INF/services/java.sql.Driver** ファイルを作成し、JDBC ドライバーの完全修飾クラス名を示す 1 行を追加します。

たとえば、MySQL JDBC ドライバーでは以下の行を追加します。

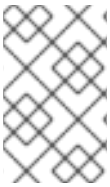
```
com.mysql.cj.jdbc.Driver
```

5. JAR コマンドラインツールを使用して、この新しいファイルを JAR に追加します。

```
jar \-uf jdbc-driver.jar META-INF/services/java.sql.Driver
```

12.2.3. JDBC ドライバーをダウンロードできる場所

下表は、JBoss EAP で使用される一般的なデータベースの JDBC ドライバーをダウンロードできる場所を示しています。



注記

これらのリンク先は他社の Web サイトであるため、Red Hat は管理しておらず、積極的に監視も行っていません。ご使用のデータベースの最新ドライバーについては、データベースベンダーのドキュメントおよび Web サイトを確認してください。

表12.1 JDBC ドライバーをダウンロードできる場所

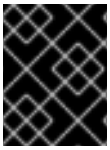
ベンダー	ダウンロード場所
MySQL	http://www.mysql.com/products/connector/
PostgreSQL	http://jdbc.postgresql.org/
Oracle	http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html
IBM	http://www-01.ibm.com/support/docview.wss?uid=swg21363866
Sybase	jConnect JDBC ドライバーは、SAP ASE インストールの SDK の一部です。現在、このドライバーのみをダウンロードできるサイトはありません。
Microsoft	http://msdn.microsoft.com/data/jdbc/

12.2.4. ベンダー固有クラスへのアクセス

場合によっては、アプリケーションが JDBC API の一部ではないベンダー固有の機能を使用する必要があることがあります。このような場合、そのアプリケーションで依存関係を宣言してベンダー固有の API にアクセスすることができます。

警告

これは高度な使用法です。JDBC API に含まれない機能を必要とするアプリケーションのみこれを実装します。



重要

このプロセスは、再認証メカニズムを使用し、ベンダー固有のクラスにアクセスする場合に必要です。

MANIFEST.MF ファイルまたは **jboss-deployment-structure.xml** ファイルを使用するとアプリケーションの依存関係を定義できます。

JDBC ドライバーをコアモジュールとしてインストール していない場合は、インストールしてください。

MANIFEST.MF ファイルの使用

1. アプリケーションの **META-INF/MANIFEST.MF** ファイルを編集します。
2. **Dependencies** 行を追加し、モジュール名を指定します。
   たとえば、以下の行は **com.mysql** モジュールを依存関係として宣言します。

```
Dependencies: com.mysql
```

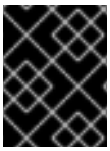
jboss-deployment-structure.xml ファイルの使用

1. アプリケーションの **META-INF/** または **WEB-INF/** フォルダーで **jboss-deployment-structure.xml** というファイルを作成します。
2. **dependencies** 要素を使用してモジュールを指定します。
   たとえば、以下の **jboss-deployment-structure.xml** ファイル例は **com.mysql** モジュールを依存関係として宣言します。

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="com.mysql"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

以下のコード例は MySQL API にアクセスします。

```
import java.sql.Connection;
...
Connection c = ds.getConnection();
if (c.isWrapperFor(com.mysql.jdbc.Connection.class)) {
    com.mysql.jdbc.Connection mc = c.unwrap(com.mysql.jdbc.Connection.class);
}
```



重要

接続は IronJacamar コンテナーによって制御されるため、ベンダー固有の API ガイドラインに従ってください。

12.3. データソースの作成

データソースは管理コンソールまたは管理 CLI を使用して作成できます。

JBoss EAP 7 では、**enabled** 属性などのデータソース属性値を式で使用することができます。設定で式を使用する場合の詳細は、プロパティーの置換 の項を参照してください。

12.3.1. 非 XA データソースの作成

管理 CLI または管理コンソールを使用して、非 XA データソース作成できます。

管理コンソールを使用した非 **XA** データソースの定義

1. スタンドアロンモードまたはドメインモードで、データソースに移動します。
   - スタンドアロンモードでは、次のナビゲーションを使用します。
     Configuration → Subsystems → Datasources & Drivers → Datasources
   - ドメインモードで次のナビゲーションを使用します。
     Configuration → Profiles → full → Datasources & Drivers → Datasources
2. 追加 (+) ボタンをクリックし、**Add Datasource** を選択します。
3. データソースタイプを選択できる **Add Datasource** ウィザードが表示されたら、**Next** をクリックします。これにより、データベースのテンプレートが作成されます。ウィザードの以下のページには、選択したデータソースに固有する値が自動的に入力されています。これにより、データソースの作成プロセスが簡単になります。
4. データソースの作成を終了する前に、**Test Connection** ページで接続をテストできます。
5. 詳細を確認し、**Finish** をクリックしてデータソースを作成します。

管理 **CLI** を使用した非 **XA** データソースの定義
data-source add 管理 CLI コマンドを使用すると、非 XA データソースを定義できます。

1. JDBC ドライバーをコアモジュールとしてインストールおよび登録していない場合は、コアモジュールとしての JDBC ドライバーのインストール を参照してインストールと登録を行ってください。
2. 適切な引数の値を指定し、**data-source add** コマンドを使用してデータソースを定義します。

```
data-source add --name=DATASOURCE_NAME --jndi-name=JNDI_NAME --driver-name=DRIVER_NAME --connection-url=CONNECTION_URL --user-name=USER_NAME --password=PASSWORD
```

注記

マネージドドメインでは、**--profile=PROFILE_NAME** 引数を指定する必要があります。

これらのパラメーター値については、以下の データソースパラメーター の項を参照してください。

詳細な例は、サポート対象データベースの データソース設定例 を参照してください。

データソースのパラメーター

jndi-name

データソースの JNDI 名は、**java:/** または **java:jboss/** で始まる必要があります。たとえば、**java:jboss/datasources/ExampleDS** になります。

driver-name

ドライバー名の値は、JDBC ドライバーがコアモジュールまたは JAR デプロイメントとしてインストールされたかによって異なります。

1. コアモジュールでは、ドライバー名の値は登録時に指定した JDBC ドライバーの名前になります。

2. JAR デプロイメントでは、**/META-INF/services/java.sql.Driver** ファイルに 1 つのクラスのみがある場合はドライバー名が JAR の名前になります。複数のクラスがリストされている場合は値が **JAR_NAME** + "_" + **DRIVER_CLASS_NAME** + "_" + **MAJOR_VERSION** + "_" + **MINOR_VERSION** (例: **mysql-connector-java-5.1.36-bin.jar_com.mysql.cj.jdbc.Driver_5_1**) になります。
また、JDBC JAR がデプロイされると JBoss EAP サーバーログにドライバー名がリストされます。

> WFLYJCA0018: Started Driver service with driver-name = mysql-connector-java-5.1.36-bin.jar_com.mysql.cj.jdbc.Driver_5_1

connection-url

サポートされるデータベースの接続 URL 形式の詳細は、データソース接続 URL のリストを参照してください。

利用できるデータソース属性の完全リストは、データソース属性 を参照してください。

user-name

新しいデータソース接続を作成するときに使用するユーザー名。

password

新しいデータソース接続を作成するときに使用するパスワード。

12.3.2. XA データソースの作成

管理 CLI または管理コンソールを使用して XA データソースを作成できます。

管理コンソールを使用した **XA** データソースの定義

1. スタンドアロンモードまたはドメインモードで、データソースに移動します。

   - スタンドアロンモードでは、次のナビゲーションを使用します。
     Configuration → Subsystems → Datasources & Drivers → Datasources

   - ドメインモードで次のナビゲーションを使用します。
     Configuration → Profiles → full → Datasources & Drivers → Datasources

2. 追加 (+) ボタンをクリックし、**XA** データソースを追加 を選択します。

3. データソースタイプを選択できる **XA** データソースを追加 ウィザードが表示されたら、**Next** をクリックします。これにより、データベースのテンプレートが作成されます。ウィザードの以下のページには、選択したデータソースに固有する値が自動的に入力されています。これにより、データソースの作成プロセスが簡単になります。

4. データソースの作成を終了する前に、**Test Connection** ページで接続をテストできます。

5. 詳細を確認し、**Finish** をクリックしてデータソースを作成します。

管理 **CLI** を使用した **XA** データソースの定義
xa-data-source add 管理 CLI コマンドを使用すると XA データソースを定義できます。

注記

マネージドドメインでは、使用するプロファイルを指定する必要があります。管理 CLI コマンドの形式に応じて、コマンドの前に **/profile=PROFILE_NAME** を付けるか、**--profile=PROFILE_NAME** 引数に渡します。

1. JDBC ドライバーをコアモジュールとしてインストールおよび登録していない場合は、コアモジュールとしての JDBC ドライバーのインストール を参照してインストールと登録を行ってください。

2. 適切な引数の値を指定し、**xa-data-source add** コマンドを使用してデータソースを定義します。

```
xa-data-source add --name=XA_DATASOURCE_NAME --jndi-name=JNDI_NAME --driver-name=DRIVER_NAME --xa-datasource-class=XA_DATASOURCE_CLASS --xa-datasource-properties=
{"ServerName"=>"HOST_NAME","DatabaseName"=>"DATABASE_NAME"}
```

   これらのパラメーター値については、以下の データソースパラメーター の項を参照してください。

3. **XA** データソースプロパティー を設定します。
   XA データソースを定義するときに最低でも 1 つの **XA** データソースプロパティーが必要になります。XA データソースプロパティーがないと、前のステップでデータソースを追加するときにエラーが発生します。XA データソースを定義するときに設定しなかったプロパティーは後で個別に設定することができます。

   a. サーバー名を設定します。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME/xa-datasource-properties=ServerName:add(value=HOST_NAME)
```

   b. データベース名を設定します。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME/xa-datasource-properties=DatabaseName:add(value=DATABASE_NAME)
```

詳細な例は、サポート対象データベースの データソース設定例 を参照してください。

データソースのパラメーター

jndi-name

データソースの JNDI 名は、**java:/** または **java:jboss/** で始まる必要があります。たとえば、**java:jboss/datasources/ExampleDS** になります。

driver-name

ドライバー名の値は、JDBC ドライバーがコアモジュールまたは JAR デプロイメントとしてインストールされたかによって異なります。

1. コアモジュールでは、ドライバー名の値は登録時に指定した JDBC ドライバーの名前になります。

2. JAR デプロイメントでは、**/META-INF/services/java.sql.Driver** ファイルに 1 つのクラスのみがある場合はドライバー名が JAR の名前になります。複数のクラスがリストされている場合は値が **JAR_NAME** + "_" + **DRIVER_CLASS_NAME** + "_" + **MAJOR_VERSION** + "_" + **MINOR_VERSION** (例: **mysql-connector-java-5.1.36-bin.jar_com.mysql.jdbc.Driver_5_1**) になります。
また、JDBC JAR がデプロイされると JBoss EAP サーバーログにドライバー名がリストされます。

> WFLYJCA0018: Started Driver service with driver-name = mysql-connector-java-5.1.36-bin.jar_com.mysql.cj.jdbc.Driver_5_1

xa-datasource-class

JDBC ドライバーの **javax.sql.XADataSource** クラスの実装に対する XA データソースクラスを指定します。

xa-datasource-properties

XA データソースを定義するときに最低でも 1 つの **XA** データソースプロパティーが必要になります。XA データソースプロパティーがないと、追加するときにエラーが発生します。 XA データソースの定義後にプロパティーを追加することもできます。

利用できるデータソース属性の完全リストは、データソース属性 を参照してください。

12.4. データソースの編集

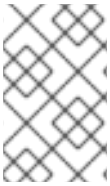
データソースは、管理コンソールまたは管理 CLI を使用して設定できます。

JBoss EAP 7 では、**enabled** 属性などのデータソース属性値を式で使用することができます。設定で式を使用する場合の詳細は、プロパティーの置換 の項を参照してください。

12.4.1. 非 XA データソースの編集

非 XA データソース設定は **data-source** 管理 CLI コマンドを使用して更新できます。スタンドアロンモードまたはドメインモードの管理コンソールから **datasource** 属性を更新することもできます。

- スタンドアロンモードでは、**Configuration → Subsystems → Datasources & Drivers → Datasources** に移動します。
- ドメインモードでは、**Configuration → Profiles → full → Datasources & Drivers → Datasources** に移動します。



注記

非 XA データソースは Jakarta Transactions トランザクションと統合できます。データソースを Jakarta Transactions と統合する場合、必ず **jta** パラメーターを **true** に設定してください。

データソースの設定を更新する例

データソースの設定は、以下の管理 CLI コマンドを使用して更新できます。

```
data-source --name=DATASOURCE_NAME --ATTRIBUTE_NAME=ATTRIBUTE_VALUE
```

注記

マネージドドメインでは、**--profile=PROFILE_NAME** 引数を指定する必要があります。

変更を反映するのにサーバーのリロードが必要になる場合があります。

12.4.2. XA データソースの編集

XA データソース設定は **xa-data-source** 管理 CLI コマンドを使用して更新できます。スタンドアロンモードまたはドメインモードの管理コンソールから **datasource** 属性を更新することもできます。

- スタンドアロンモードでは、**Configuration → Subsystems → Datasources & Drivers → Datasources** に移動します。
- ドメインモードでは、**Configuration → Profiles → full → Datasources & Drivers → Datasources** に移動します。

XA データソースの更新例

- XA データソースの設定は、以下の管理 CLI コマンドを使用して更新できます。

```
xa-data-source --name=XA_DATASOURCE_NAME -
-ATTRIBUTE_NAME=ATTRIBUTE_VALUE
```

注記

マネージドドメインでは、**--profile=PROFILE_NAME** 引数を指定する必要があります。

XA データソースプロパティーの追加例

- 以下の管理 CLI コマンドを使用すると **XA** データソースプロパティーを追加できます。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME/xa-datasource-
properties=PROPERTY:add(value=VALUE)
```

注記

マネージドドメインでは、このコマンドの前に **/profile=PROFILE_NAME** を追加する必要があります。

変更を反映するのにサーバーのリロードが必要になる場合があります。

12.5. データソースの削除

データソースは管理コンソールまたは管理 CLI を使用して削除できます。

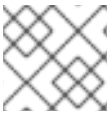
12.5.1. 非 XA データソースの削除

非 XA データソースは **data-source remove** 管理 CLI コマンドを使用して削除できます。スタンドアロンモードまたはドメインモードの管理コンソールを使用して、データソースを削除することもできます。

- スタンドアロンモードでは、**Configuration → Subsystems → Datasources & Drivers → Datasources** に移動します。
- ドメインモードでは、**Configuration → Profiles → full → Datasources & Drivers → Datasources** に移動します。

次のコマンドを使用して非 XA データソースを削除します。

```
data-source remove --name=DATASOURCE_NAME
```



注記

マネージドドメインでは、**--profile=PROFILE_NAME** 引数を指定する必要があります。

データソースの削除後にサーバーのリロードが必要になります。

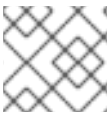
12.5.2. XA データソースの削除

XA データソースは **xa-data-source remove** 管理 CLI コマンドを使用して削除できます。スタンドアロンモードまたはドメインモードの管理コンソールを使用して、データソースを削除することもできます。

- スタンドアロンモードでは、**Configuration → Subsystems → Datasources & Drivers → Datasources** に移動します。
- ドメインモードでは、**Configuration → Profiles → full → Datasources & Drivers → Datasources** に移動します。

次のコマンドを使用して XA データソースを削除します。

```
xa-data-source remove --name=XA_DATASOURCE_NAME
```



注記

マネージドドメインでは、**--profile=PROFILE_NAME** 引数を指定する必要があります。

XA データソースの削除後にサーバーのリロードが必要になります。

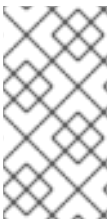
12.6. データソース接続のテスト

管理 CLI または管理コンソールを使用してデータソースの接続をテストし、設定が適切であることを検証できます。

管理 CLI を使用したデータソース接続のテスト
以下の管理 CLI コマンドを実行すると、データソースの接続をテストできます。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:test-connection-in-pool
```

-



注記

マネージドドメインでは、このコマンドの前に **/host=HOST_NAME/server=SERVER_NAME** を追加する必要があります。XA データソースをテストする場合は、 **data-source=DATASOURCE_NAME** を **xa-data-source=XA_DATASOURCE_NAME** に置き換えてください。

管理コンソールを使用したデータソース接続のテスト
管理コンソールで データソースの追加 ウィザードを使用する場合、データソースの作成前に接続をテストできます。ウィザードの テスト接続 画面で テスト接続 ボタンをクリックします。

データソースが追加されたら、次の手順を使用して接続をテストできます。

1. スタンドアロンモードでは、**Configuration → Subsystems → Datasources & Drivers → Datasources** に移動します。ドメインモードでは、**Configuration → Profiles → full → Datasources & Drivers → Datasources** に移動します。
2. データソースを選択します。
3. ドロップダウンリストから **Test Connection** を選択します。

12.7. データソース接続のフラッシュ

以下の管理 CLI コマンドを使用して、データベースの接続をフラッシュします。

注記

マネージドドメインでは、コマンドの前に **/host=HOST_NAME/server=SERVER_NAME** を追加する必要があります。

- プールの接続をすべてフラッシュします。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:flush-all-connection-in-pool
```

- プールの接続をすべて正常にフラッシュします。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:flush-gracefully-connection-in-pool
```

  サーバーは、接続がアイドル状態なるまで待ってから接続をフラッシュします。

- プールのアイドル状態の接続をすべてフラッシュします。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:flush-idle-connection-in-pool
```

- プールの無効な接続をすべてフラッシュします。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:flush-invalid-connection-in-pool
```

サーバーは、データベース接続の検証 で説明した **valid-connection-checker-class-name** または check-valid-connection-sql 検証メカニズムなどによって無効と判断されたすべての接続をフラッシュします。

管理コンソールを使用して接続をフラッシュすることもできます。**Runtime** タブでサーバーを選択し、**Datasources** を選択してデータソースを指定します。 ドロップダウンメニューを使用してアクションを選択します。

12.8. XA データソースのリカバリー

XA データソースは、XA グローバルトランザクションに参加できるデータソースです。XA グローバルトランザクションはトランザクションマネージャーによって調整され、1 つのトランザクションで複数のリソースにまたがる可能性があります。参加者の 1 つが変更のコミットに失敗した場合、他の参加者がトランザクションをアボートし、トランザクション発生前の状態にリストアします。これにより、一貫性を保持し、データの損失や破損を防ぎます。

XA リカバリーは、リソースやトランザクションの参加者がクラッシュしたり利用できない状態になっても、トランザクションの影響を受けるすべてのリソースが更新またはロールバックされるようにするプロセスです。XA リカバリーはユーザーが関与せずに行われます。

各 XA リソースにはその設定に関連するリカバリーモジュールが必要になります。リカバリーモジュールは、リカバリーの実行中に実行されるコードです。JBoss EAP は JDBC XA リソースのリカバリーモジュールを自動的に登録します。カスタムのリカバリーコードを実装する場合は XA データソースでカスタムモジュールを登録できます。リカバリーモジュールは **com.arjuna.ats.jta.recovery.XAResourceRecovery** クラスを拡張する必要があります。

12.8.1. XA リカバリーの設定

ほとんどの JDBC では、リカバリーモジュールがリソースに自動的に関連付けられます。この場合は、リカバリーモジュールがリソースに接続してリカバリーを実行することを許可するオプションのみを設定する必要があります。

以下の表は、XA リカバリーに固有する XA データソースパラメーターを表しています。これらの設定属性はデータソースの作成中または作成後に設定できます。これらは管理コンソールまたは管理 CLI を使用して設定できます。XA データソースのせって の設定に関する詳細は、 XA データソースの編集を参照してください。

表12.2 XA リカバリーのデータソースパラメーター

属性	説明
recovery-username	リカバリーでリソースに接続するために使用するユーザー名。指定しないと、データソースのセキュリティー設定が使用されます。
recovery-password	リカバリーのリソースへの接続に使用するパスワード。指定しないと、データソースのセキュリティー設定が使用されます。
recovery-security-domain	リカバリーでリソースに接続するために使用するセキュリティードメイン。
recovery-plugin-class-name	カスタムのリカバリーモジュールを使用する必要がある場合は、この属性をモジュールの完全修飾クラス名に設定します。モジュールはクラス **com.arjuna.ats.jta.recovery.XAResourceRecovery** を拡張する必要があります。

属性	説明
recovery-plugin-properties	プロパティーを設定する必要があるカスタムのリカバリーモジュールを使用する場合は、この属性をプロパティーに対するコンマ区切りの **KEY=VALUE** ペアのリストに設定します。

XA リカバリーの無効化

複数の XA データソースが同じ物理データベースに接続する場合、通常 XA リカバリーは XA データソースの 1 つのみに設定する必要があります。

以下の管理 CLI コマンドを使用して XA データソースのリカバリーを無効にします。

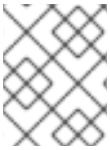
```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME:write-attribute(name=no-recovery,value=true)
```

12.8.2. ベンダー固有の XA リカバリー

ベンダー固有の設定
一部のデータベースは、JBoss EAP トランザクションマネージャーによって管理される XA トランザクションに対応するために特定の設定が必要になります。詳細な最新情報については、データベースベンダーの資料を参照してください。

MySQL

特別な設定は必要ありません。詳細は MySQL のドキュメントを参照してください。



注記

自動化された XA リカバリーの場合には、MySQL 8 以降では特別な設定が必要です。詳細は、JBoss EAP 7.4 でサポートされる設定 を参照してください。

PostgreSQL および Postgres Plus Advanced Server

PostgreSQL による XA トランザクションの処理を可能にするには、**max_prepared_transactions** 設定パラメーターを **0** よりも大きい値または **max_connections** 以上の値に変更します。

Oracle

必ず Oracle ユーザー **USER** がリカバリーに必要なテーブルにアクセスできるようにしてください。

```
GRANT SELECT ON sys.dba_pending_transactions TO USER;
GRANT SELECT ON sys.pending_trans$ TO USER;
GRANT SELECT ON sys.dba_2pc_pending TO USER;
GRANT EXECUTE ON sys.dbms_xa TO USER;
```

Oracle ユーザーに適切なパーミッションがないと、以下のようなエラーが表示される可能性があります。

```
WARN  [com.arjuna.ats.jta.logging.loggerI18N] [com.arjuna.ats.internal.jta.recovery.xarecovery1]
Local XARecoveryModule.xaRecovery  got XA exception javax.transaction.xa.XAException,
XAException.XAER_RMERR
```

Microsoft SQL Server

詳細は、http://msdn.microsoft.com/en-us/library/aa342335.aspx を含む Microsoft SQL Server のドキュメントを参照してください。

IBM DB2

特別な設定は必要ありません。詳細は IBM DB2 のドキュメントを参照してください。

Sybase

Sybase は、XA トランザクションがデータベース上で有効であることを想定します。XA トランザクションはデータベース設定が正しくないと動作しません。**enable xact coordination** パラメーターは、Adaptive Server トランザクションコーディネーションサービスを有効または無効にします。このパラメーターを有効にすると、リモート Adaptive Server データの更新が、確実に元のトラザクションでコミットまたはロールバックされるようになります。
トラザクションコーディネーションを有効にするには、以下を使用します。

```
sp_configure 'enable xact coordination', 1
```

MariaDB

特別な設定は必要ありません。詳細は MariaDB のドキュメントを参照してください。

既知の問題
ここで取り上げる既知の問題は、JBoss EAP 7 でサポートされる特定のデータベースおよび JDBC ドライバーバージョンの XA トランザクションの処理に関する問題になります。サポートされるデータベースの最新情報は、JBoss Enterprise Application Platform (EAP) 7 でサポートされる設定 を参照してください。

MySQL

MySQL は XA トランザクションを完全に処理できません。クライアントと MySQL の接続が切断されると、トランザクションに関する情報がすべて失われます。詳細は MySQL のバグ を参照してください。この問題は MuSQL 5.7 で修正されました。

PostgreSQL および Postgres Plus Advanced Server

2 フェーズコミット (2PC) のコミットフェーズ中にネットワークの障害が発生すると、JDBC ドライバーによって **XAER_RMERR** XAException エラーコードが返されます。このエラーは、トランザクションマネージャーでリカバリー不可能な重大なイベントが発生したことを示しますが、トランザクションはデータベース側で **in-doubt** 状態を維持し、ネットワーク接続の回復後に簡単に修正できます。適切な戻りコードは **XAER_RMFAIL** または **XAER_RETRY** になります。誤ったエラーコードにより、トランザクションが JBoss EAP 側で **Heuristic** 状態のままになり、データベースでロックが保持されるため、手動の介入が必要になります。詳細は PostgreSQL のバグ を参照してください。
1 フェーズコミットの最適化が使用されたときに接続の障害が発生した場合、JDBC ドライバーは **XAER_RMERR** を返しますが、適切な戻りコードは **XAER_RMFAIL** になります。そのため、1 フェーズコミット中にデータベースがデータをコミットし、同時に接続が切断されると、クライアントにはトラザクションがロールバックされたと伝えられるため、データの不整合が発生する場合があります。

Postgres Plus JDBC ドライバーは、Postgres Plus Server に存在するすべての準備済みトランザクションの XID を返すため、XID が属するデータベースを判断する方法がありません。JBoss EAP で複数のデータソースを同じデータベースに定義すると、in-doubt トランザクションリカバリーが誤ったアカウントで実行される可能性があります。 この場合、リカバリーに失敗します。

Oracle

一部のユーザー認証情報で設定されたデータソースを使用してリカバリーマネージャーがリカバリーを呼び出すと、JDBC ドライバーはデータベースインスタンスのすべてのユーザーに属する XID を返します。JDBC ドライバーは他のユーザーに属する XID をリカバリーしようとするため、例外 **ORA-24774: cannot switch to specified transaction** が発生します。
この問題を回避するには、リカバリーデータソース設定で認証情報が使用されるユーザーに **FORCE ANY TRANSACTION** 権限を付与します。特権の設定に関する詳細は http://docs.oracle.com/database/121/ADMIN/ds_txnman.htm#ADMIN12259 を参照してください。

Microsoft SQL Server

2 フェーズコミット (2PC) のコミットフェーズ中にネットワークの障害が発生すると、JDBC ドライバーによって **XAER_RMERR** XAException エラーコードが返されます。このエラーは、トランザクションマネージャーでリカバリー不可能な重大なイベントが発生したことを示しますが、トランザクションはデータベース側で **in-doubt** 状態を維持し、ネットワーク接続の回復後に簡単に修正できます。適切な戻りコードは **XAER_RMFAIL** または **XAER_RETRY** になります。誤ったエラーコードにより、トランザクションが JBoss EAP 側で **Heuristic** 状態のままになり、データベースでロックが保持されるため、手動の介入が必要になります。詳細は Microsoft SQL Server の問題レポート を参照してください。
1 フェーズコミットの最適化が使用されたときに接続の障害が発生した場合、JDBC ドライバーは **XAER_RMERR** を返しますが、適切な戻りコードは **XAER_RMFAIL** になります。そのため、1 フェーズコミット中にデータベースがデータをコミットし、同時に接続が切断されると、クライアントにはトラザクションがロールバックされたと伝えられるため、データの不整合が発生する場合があります。

IBM DB2

1 フェーズコミット中に接続の障害が発生した場合、JDBC ドライバーは **XAER_RMERR** を返しますが、適切な戻りコードは **XAER_RMFAIL** です。そのため、1 フェーズコミット中にデータベースがデータをコミットし、同時に接続が切断されると、クライアントにはトラザクションがロールバックされたと伝えられるため、データの不整合が発生する場合があります。

Sybase

2 フェーズコミット (2PC) のコミットフェーズ中にネットワークの障害が発生すると、JDBC ドライバーによって **XAER_RMERR** XAException エラーコードが返されます。このエラーは、トランザクションマネージャーでリカバリー不可能な重大なイベントが発生したことを示しますが、トランザクションはデータベース側で **in-doubt** 状態を維持し、ネットワーク接続の回復後に簡単に修正できます。適切な戻りコードは **XAER_RMFAIL** または **XAER_RETRY** になります。誤ったエラーコードにより、トランザクションが JBoss EAP 側で **Heuristic** 状態のままになり、データベースでロックが保持されるため、手動の介入が必要になります。
1 フェーズコミットの最適化が使用されたときに接続の障害が発生した場合、JDBC ドライバーは **XAER_RMERR** を返しますが、適切な戻りコードは **XAER_RMFAIL** になります。そのため、1 フェーズコミット中にデータベースがデータをコミットし、同時に接続が切断されると、クライアントにはトラザクションがロールバックされたと伝えられるため、データの不整合が発生する場合があります。

Sybase トランザクションブランチが準備済み状態になる前に、Sybase 15.7 または 16 データベースへの挿入が含まれる XA トランザクションが失敗すると、XA トランザクションの繰り返しや同じプライマリーキーでの同じレコードの挿入に失敗し、エラー **com.sybase.jdbc4.jdbc.SybSQLException: Attempt to insert duplicate key row** が発生します。この例外は、元の終了していない Sybase トランザクションブランチがロールバックするまで発生します。

MariaDB

MariaDB は XA トランザクションを完全に処理できません。クライアントと MariaDB の接続が切断されると、トランザクションに関する情報がすべて失われます。

MariaDB Galera クラスター

MariaDB Galera クラスターでは XA トランザクションはサポートされません。

12.9. データベース接続の検証

データベースのメンテナンス、ネットワークの問題、またはその他の障害により、JBoss EAP からデータベースへの接続が失われることがあります。このような状況から回復するために、データソースのデータベース接続検証を有効にすることができます。

データベース接続の検証を設定するには、検証発生時を定義する検証タイミングメソッド、検証の実行方法を決定する検証メカニズム、および例外の処理方法を定義する例外ソーターを指定します。

1. 検証タイミングメソッドを 1 つ選択します。

validate-on-match

validate-on-match メソッドが **true** に設定されている場合は、データ接続が、次の手順で指定された検証メカニズムを使用して接続プールからチェックアウトされるたびに検証されます。
接続が有効でない場合は、警告がログに書き込まれ、プール内の次の接続が取得されます。このプロセスは、有効な接続が見つかるまで続行します。プール内の各接続を繰り返し処理しない場合は、**use-fast-fail** オプションを使用できます。有効な接続がプールにない場合は、新しい接続が作成されます。接続の作成に失敗すると、例外が要求元アプリケーションに返されます。

background-validation

background-validation メソッドを **true** に設定すると、使用前にバックグラウンドスレッドで接続が周期的に検証されます。検証の頻度は **background-validation-millis** プロパティーによって指定されます。**background-validation-millis** のデフォルト値は **0** で、無効になっています。
以下を考慮して **background-validation-millis** プロパティーの値を決定してください。

- この値は **idle-timeout-minutes** 設定とは違う値に設定してください。
- 値が小さいほどプールの検証頻度が高くなり、より迅速に無効な接続がプールから削除されます。
- 値が小さいほど使用されるデータベースリソースが多くなります。値が大きいほど接続検証チェックの頻度が低くなり、データベースリソースの使用量が減りますが、無効な接続が検出されない期間が長くなります。

注記

次の例に示すように、これらの検証方法は同時に使用できません。

- **validate-on-match** が **true** に設定されている場合には、**background-validation** を **false** に設定する必要があります。
- **background-validation** が **true** に設定されている場合は、**validate-on-match** を **false** に設定する必要があります。

これらの検証方法の比較マトリックスについては 検証タイミング方法の比較 を参照してください。

2. 検証メカニズムを 1 つ選択します。

valid-connection-checker-class-name

検証メカニズムとして **valid-connection-checker-class-name** を使用することが推奨されます。これは、使用中のデータベースの接続を検証するために使用される接続チェッカークラスを指定します。JBoss EAP は以下の接続チェッカーを提供します。

- **org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLReplicationValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.novendor.JDBC4ValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.novendor.NullValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker**

check-valid-connection-sql

check-valid-connection-sql を使用して、接続の検証に使用する SQL ステートメントを提供します。
以下は、Oracle の接続を検証するために使用する SQL ステートメントの例になります。

```
select 1 from dual
```

以下は、MySQL または PostgreSQL の接続を検証するために使用する SQL ステートメントの例になります。

```
select 1
```

3. 例外ソータークラス名を設定します。
例外が致命的とマークされた場合、接続はトランザクションに参加していてもすぐに閉じられます。致命的な接続例外を適切に検出およびクリーンアップするには、例外ソータークラスオプションを使用します。データソースタイプに適切な JBoss EAP 例外ソーターを選択します。

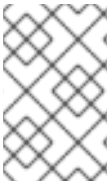
- **org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter**
- **org.jboss.jca.adapters.jdbc.extensions.informix.InformixExceptionSorter**
- **org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter**
- **org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter**
- **org.jboss.jca.adapters.jdbc.extensions.novendor.NullExceptionSorter**
- **org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter**

- **org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter**
- **org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter**

12.10. データソースセキュリティー

データソースセキュリティーとは、データソース接続のパスワードを暗号化したり分かりにくくすることを言います。これらのパスワードはプレーンテキストで設定ファイルに保存できますが、セキュリティーリスクが高くなります。

データソースセキュリティーに使用できるメソッドは複数あります。以下には、各メソッドの例が含まれています。



注記

従来のセキュリティーサブシステムを使用する場合は、データソース接続の認証と承認に LDAP を使用するようにセキュリティードメインを設定できます。セキュリティードメインの設定に関する詳細は、セキュリティードメインの設定 を参照してください。

セキュリティードメインを使用したデータソースのセキュア化
セキュリティードメインを使用してデータソースをセキュアにするには、以下の手順に従います。

1. 新しいセキュリティードメインを作成します。

```
/subsystem=security/security-domain=DsRealm:add(cache-type=default)
/subsystem=security/security-domain=DsRealm/authentication=classic:add(login-modules=
[{code=ConfiguredIdentity,flag=required,module-options={userName=sa,
principal=sa, password=sa}}])
```

   データソースのセキュリティードメインが定義されます。以下の XML の抜粋は、CLI コマンドを呼び出した結果です。

```
<security-domain name="DsRealm" cache-type="default">
 <authentication>
  <login-module code="ConfiguredIdentity" flag="required">
   <module-option name="userName" value="sa"/>
   <module-option name="principal" value="sa"/>
   <module-option name="password" value="sa"/>
  </login-module>
 </authentication>
</security-domain>
```

2. 新しいデータソースを追加します。

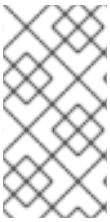
```
data-source add --name=securityDs
--jndi-name=java:jboss/datasources/securityDs
--connection-url=jdbc:h2:mem:test;DB_CLOSE_DELAY=-1 --driver-name=h2
--new-connection-sql="select current_user()"
```

3. データソースにセキュリティードメインを設定します。

```
data-source --name=securityDs --security-domain=DsRealm
```

4. 変更を有効にするには、サーバーをリロードします。

```
reload
```



注記

複数のデータソースでセキュリティードメインを使用している場合は、セキュリティードメインでキャッシュを無効にします。これには、**cache-type** 属性の値を **none** に設定するか、この属性を削除します。ただし、キャッシュが必要な場合は、各データソースに個別のセキュリティードメインを使用します。

以下の XML の抜粋は、**DsRealm** で保護されたデータソースを示しています。

```
<datasources>
  <datasource jndi-name="java:jboss/datasources/securityDs"
    pool-name="securityDs">
    <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
      <driver>h2</driver>
      <new-connection-sql>select current_user()</new-connection-sql>
      <security>
        <security-domain>DsRealm</security-domain>
      </security>
    </datasource>
</datasources>
```

セキュリティードメインの使用に関する詳細は、アイデンティティー管理の設定方法 を参照してください。

パスワード Vault を使用したデータソースのセキュア化

パスワード vault を使用してデータソースを保護するには、以下の手順を使用します。

1. ExampleDS データソースのパスワード vault を設定します。

```
data-source --name=ExampleDS
--
password=${VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmEtMWNlMDMyNDdmNmI2TElORV9CUkVBS3ZhdWx0}
```

2. サーバーをリロードして、変更を実装します。

```
reload
```

以下の XML セキュリティー要素は、パスワード vault でセキュア化された ExampleDS データソースに追加されます。

```
<security>
  <user-name>admin</user-name>

<password>${VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmEtMWNlMDMyNDdmNmI2TElORV9CUkVBS3ZhdWx0}</password>
</security>
```

パスワード vault の使用に関する詳細は、JBoss EAP**How to Configure Server Security**の Password Vault の項を参照してください。

認証情報ストアを使用したデータソースのセキュア化

認証情報ストアを使用してパスワードを提供することもできます。**elytron** サブシステムを使用すると、認証情報ストアを作成してパスワードをセキュアに保存し、JBoss EAP 全体でパスワードを使用することができます。認証情報ストアの作成および使用に関する詳細は、JBoss EAP**How to Configure Server Security**の Credential Store を参照してください。

認証情報ストア参照を ExampleDS に追加

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=credential-reference,value=
{store=exampleCS, alias=example-ds-pw})
```

認証コンテキストを使用したデータソースのセキュア化

Elytron 認証コンテキストを使用して、ユーザー名とパスワードを提供することもできます。

以下の手順にしたがって、データソースセキュリティーの認証コンテキストを設定および使用します。

1. **password** と **user-name** を削除します。

```
/subsystem=datasources/data-source=ExampleDS:undefine-attribute(name=password)
/subsystem=datasources/data-source=ExampleDS:undefine-attribute(name=user-name)
```

2. データソースの Elytron セキュリティーを有効にします。

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=elytron-
enabled,value=true)

reload
```

3. 認証情報の **authentication-configuration** を作成します。
認証設定には、接続時にデータソースに使用させる認証情報が含まれています。以下の例は、認証情報ストアへの参照を使用しますが、Elytron セキュリティードメインを使用することもできます。

```
/subsystem=elytron/authentication-configuration=exampleAuthConfig:add(authentication-
name=sa,credential-reference={clear-text=sa})
```

4. **authentication-context** を作成します。

```
/subsystem=elytron/authentication-context=exampleAuthContext:add(match-rules=
[{authentication-configuration=exampleAuthConfig}])
```

5. 認証コンテキストを使用するよう、データソースを更新します。
以下の例は、認証コンテキストを使用するよう、**ExampleDS** を更新します。

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=authentication-
context,value=exampleAuthContext)

reload
```

注記

authentication-context 属性が設定されておらず、**elytron-enabled** 属性が **true** に設定されている場合、JBoss EAP は認証に現在のコンテキストを使用します。

Kerberos を使用したデータソースの保護

kerberos 認証を使用してデータソースを保護にするには、以下の設定が必要です。

- Kerberos がデータベースサーバーに設定されている。
- JBoss EAP ホストサーバーには、データベースサーバーのキータブエントリーがある。

kerberos を使用してデータソースを保護にするには、以下を実行します。

1. Kerberos を使用するよう JBoss EAP を設定

```
/system-property=java.security.krb5.conf:add(value="/path/to/krb5.conf")
/system-property=sun.security.krb5.debug:add(value="false")
/system-property=sun.security.spnego.debug:add(value="false")
```

デバッグを行う場合は、**sun.security.krb5.debug** と **sun.security.spnego.debug** の値を **true** に変更します。本番環境では、値を **false** に設定することが推奨されます。

2. セキュリティーを設定します。
データソースを保護にするには、レガシーセキュリティーまたは Elytron セキュリティーを使用できます。

- レガシーのセキュリティーで kerberos を使用するには、以下の手順に従います。

a. 期限切れのチケットをキャッシュから定期的に削除するように infinispan キャッシュを設定します。

```
batch
/subsystem=infinispan/cache-container=security:add(default-cache=auth-cache)
/subsystem=infinispan/cache-container=security/local-cache=auth-cache:add()
/subsystem=infinispan/cache-container=security/local-cache=auth-
cache/expiration=EXPIRATION:add(lifespan=3540000,max-idle=3540000)
/subsystem=infinispan/cache-container=security/local-cache=auth-
cache/memory=object:add(size=1000)
run-batch
```

以下の属性はチケットの有効期限を定義します。

- **lifespan**: KDC から新しい証明書を要求する間隔 (ミリ秒単位)。KDC で定義される lifespan 条件よりも小さい値になるように、**lifespan** 属性 の値を設定します。
- **max-idle**: 未使用の場合に、有効なチケットがキャッシュから削除される間隔 (ミリ秒単位)。
- **max-entries**: キャッシュに保持する kerberos チケットの最大コピー数。この値は、データソースで設定された接続の数に対応します。

b. セキュリティードメインを作成します。

```
batch
```

```
/subsystem=security/security-domain=KerberosDatabase:add(cache-type=infinispan)
/subsystem=security/security-domain=KerberosDatabase/authentication=classic:add
/subsystem=security/security-
domain=KerberosDatabase/authentication=classic/login-
module="KerberosDatabase-
Module":add(code="org.jboss.security.negotiation.KerberosLoginModule",module="org.
jboss.security.negotiation",flag=required, module-options={ "debug" => "false",
"storeKey" => "false", "useKeyTab" => "true", "keyTab" => "/path/to/eap.keytab",
"principal" => "PRINCIPAL@SERVER.COM", "doNotPrompt" => "true",
"refreshKrb5Config" => "true", "isInitiator" => "true", "addGSSCredential" => "true",
"credentialLifetime" => "-1"})
run-batch
```

- SQL サーバーに Microsoft JDBC ドライバーを使用する場合は、**module-options** に属性と **"wrapsphinx" ⇒ "true"** の値を追加します。
- デバッグするには、**module-options** の **debug** 属性の値を **true** に変更します。

- Elytron で kerberos を使用するには、以下を行います。

a. Elytron で kerberos ファクトリーを設定します。

```
/subsystem=elytron/kerberos-security-factory=krbsf:add(debug=false,
principal=PRINCIPAL@SERVER.COM, path=/path/to/keytab, request-lifetime=-1,
obtain-kerberos-ticket=true, server=false)
```

- デバッグを行う場合は、属性および **debug = true** の値を追加します。
対応している属性のリストは、**How to Configure Server Security**の Kerberos Security Factory Attributes を参照してください。

b. kerberos ファクトリーを使用するよう認証設定を作成します。

```
/subsystem=elytron/authentication-configuration=kerberos-conf:add(kerberos-
security-factory=krbsf)
```

c. authentication-context を作成します。

```
/subsystem=elytron/authentication-context=ds-context:add(match-rules=
[{authentication-configuration=kerberos-conf}])
```

3. データソースを kerberos で保護します。

- レガシーのセキュリティーを使用する場合:

a. セキュリティードメインを使用するようにデータソースを設定します。

```
/subsystem=datasources/data-source=KerberosDS:add(connection-url="URL", min-
pool-size=0, max-pool-size=10, jndi-name="java:jboss/datasource/KerberosDS",
driver-name=<jdbc-driver>.jar, security-domain=KerberosDatabase, allow-multiple-
users=false, pool-prefill=false, pool-use-strict-min=false, idle-timeout-minutes=2)
```

b. ベンダー固有の接続プロパティーを設定します。

```
/subsystem=datasources/data-source=KerberosDS/connection-properties=
<connection-property-name>:add(value="(<kerberos-value>)")
```

例: Oracle データベースの接続プロパティー。

```
/subsystem=datasources/data-source=KerberosDS/connection-properties=oracle.net.authentication_services:add(value="(KERBEROS5)")
```

- Elytron を使用する場合:

  a. 認証コンテキストを使用するようにデータソースを設定します。

  ```
  /subsystem=datasources/data-source=KerberosDS:add(connection-url="URL", min-pool-size=0, max-pool-size=10, jndi-name="java:jboss/datasource/KerberosDS", driver-name=<jdbc-driver>.jar, elytron-enabled=true, authentication-context=ds-context, allow-multiple-users=false, pool-prefill=false, pool-use-strict-min=false, idle-timeout-minutes=2)
  ```

  b. ベンダー固有の接続プロパティーを設定します。

  ```
  /subsystem=datasources/data-source=KerberosDS/connection-properties=<connection-property-name>:add(value="(<kerberos-value>)")
  ```

  例: Oracle データベースの接続プロパティー

  ```
  /subsystem=datasources/data-source=KerberosDS/connection-properties=oracle.net.authentication_services:add(value="(KERBEROS5)")
  ```

  kerberos 認証を使用する場合は、データソースに以下の属性と値を使用することが推奨されます。

  - **pool-prefill=false**
  - **pool-use-strict-min=false**
  - **idle-timeout-minutes**

対応している属性のリストは、データソースの属性 を参照してください。

12.11. データソースの統計

データソースの統計収集が 有効化 されている場合、データソースの ランタイム統計を表示 できます。

12.11.1. データソース統計の有効化

データソース統計は、デフォルトでは有効になっていません。データソース統計の収集は、 管理 CLI または 管理コンソール を使用して有効にできます。

管理 **CLI** を使用したデータソース統計の有効化
以下の管理 CLI コマンドは、**ExampleDS** データソースの統計の収集を有効にします。

注記

マネージドドメインでは、このコマンドの前に **/profile=PROFILE_NAME** を付けます。

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=statistics-
enabled,value=true)
```

変更を反映するためにサーバーをリロードします。

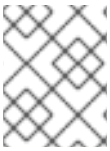
管理コンソール使用したデータソース統計の有効化
以下の手順にしたがって管理コンソールを使用し、統計の収集を有効にします。

1. スタンドアロンモードまたはドメインモードで、データソースに移動します。
   - スタンドアロンモードでは、次のナビゲーションを使用します。
     Configuration → Subsystems → Datasources & Drivers → Datasources
   - ドメインモードで次のナビゲーションを使用します。
     Configuration → Profiles → full → Datasources & Drivers → Datasources
2. データソースを選択し、**View** をクリックします。
3. **Attributes** タブ下の **Edit** をクリックします。
4. **Statistics Enabled** フィールドを **ON** に設定し、**Save** をクリックします。変更の反映にはリロードが必要であることを伝えるポップアップが表示されます。
5. サーバーをリロードします。
   - スタンドアロンサーバーの場合は、ポップアップの **Reload** ボタンをクリックしてサーバーをリロードします。
   - マネージドドメインの場合は、ポップアップの **Topology** リンクをクリックします。**Topology** タブで該当するサーバーを選択し、**Reload** ドロップダウンオプションを選択してサーバーをリロードします。

12.11.2. データソース統計の表示

管理 CLI または 管理コンソール を使用してデータソースのランタイム統計を表示できます。

管理 **CLI** を使用したデータソース統計の表示
以下の管理 CLI コマンドは、**ExampleDS** データソースのコアプールの統計を取得します。



注記

マネージドドメインでは、コマンドの前に
/host=HOST_NAME/server=SERVER_NAME を追加します。

```
/subsystem=datasources/data-source=ExampleDS/statistics=pool:read-resource(include-
runtime=true)
{
    "outcome" => "success",
    "result" => {
        "ActiveCount" => 1,
        "AvailableCount" => 20,
        "AverageBlockingTime" => 0L,
        "AverageCreationTime" => 122L,
        "AverageGetTime" => 128L,
        "AveragePoolTime" => 0L,
```

```
        "AverageUsageTime" => 0L,
        "BlockingFailureCount" => 0,
        "CreatedCount" => 1,
        "DestroyedCount" => 0,
        "IdleCount" => 1,
        ...
}
```

以下の管理 CLI コマンドは、**ExampleDS** データソースの **JDBC** の統計を取得します。

```
/subsystem=datasources/data-source=ExampleDS/statistics=jdbc:read-resource(include-
runtime=true)
{
    "outcome" => "success",
    "result" => {
        "PreparedStatementCacheAccessCount" => 0L,
        "PreparedStatementCacheAddCount" => 0L,
        "PreparedStatementCacheCurrentSize" => 0,
        "PreparedStatementCacheDeleteCount" => 0L,
        "PreparedStatementCacheHitCount" => 0L,
        "PreparedStatementCacheMissCount" => 0L,
        "statistics-enabled" => true
    }
}
```

注記

統計はラインタイム情報であるため、必ず **include-runtime=true** 引数を指定してください。

利用可能な統計の詳細リストは、データソースの統計 を参照してください。

管理コンソールを使用したデータソース統計の表示
管理コンソールからデータソースの統計を表示するには、**Runtime** タブで **Datasources** サブシステムを選択し、データソースを選択してから 表示 をクリックします。

利用可能な統計の詳細リストは、データソースの統計 を参照してください。

12.12. データソースの調整

datasources サブシステムのパフォーマンスを監視および最適化するための情報は、**Performance Tuning Guide**の Datasource and Resource Adapter Tuning の項を参照してください。

12.13. キャパシティーポリシー

JBoss EAP は、データソースを含む Jakarta Connectors デプロイメントのキャパシティーポリシーの定義をサポートします。キャパシティーポリシーは、キャパシティーのインクリメントと呼ばれるプールの物理接続の作成方法と、キャパシティーのデクリメントと呼ばれる破棄方法を定義します。デフォルトのポリシーは、キャパシティーのインクリメントではリクエストごとに 1 つの接続を作成し、キャパシティーのデクリメントではアイドル状態のタイムアウトがスケジュールされたときにタイムアウトするとすべての接続が破棄されるよう設定されます。

キャパシティーポリシーを設定するには、キャパシティーインクリメンタークラスかキャパシティーデクリメンタークラスのいずれか、両方を指定する必要があります。

例: キャパシティーポリシーの定義

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=capacity-incrementer-class,
value="org.jboss.jca.core.connectionmanager.pool.capacity.SizeIncrementer")

/subsystem=datasources/data-source=ExampleDS:write-attribute(name=capacity-decrementer-class,
value="org.jboss.jca.core.connectionmanager.pool.capacity.SizeDecrementer")
```

指定したキャパシティーインクリメンターまたはデクリメンタークラスにプロパティーを設定することもできます。

例: キャパシティーポリシーのプロパティー設定

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=capacity-incrementer-
properties.size, value=2)

/subsystem=datasources/data-source=ExampleDS:write-attribute(name=capacity-decrementer-
properties.size, value=2)
```

MaxPoolSize インクリメンターポリシー

クラス名: **org.jboss.jca.core.connectionmanager.pool.capacity.MaxPoolSizeIncrementer**

MaxPoolSize インクリメンターポリシーは、リクエストごとにプールを最大サイズまでインクリメントします。このポリシーは、常時利用できる接続を最大数維持したい場合に便利です。

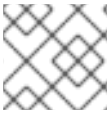
Size インクリメンターポリシー

クラス名: **org.jboss.jca.core.connectionmanager.pool.capacity.SizeIncrementer**

Size インクリメンターポリシーは、リクエストごとにプールを指定の接続数までインクリメントします。このポリシーは、次のリクエストにも接続が必要であることを予想する場合に追加の接続数でインクリメントしたい場合に便利です。

表12.3 Size ポリシープロパティー

名前	説明
Size	作成される接続数



注記

これはデフォルトのインクリメントポリシーで、**size** の値は 1 になります。

Watermark インクリメンターポリシー

クラス名: **org.jboss.jca.core.connectionmanager.pool.capacity.WatermarkIncrementer**

Watermark インクリメンターポリシーは、リクエストごとにプールを指定の接続数までインクリメントします。このポリシーは、常時プールに指定数の接続を維持したい場合に便利です。

表12.4 Watermark ポリシープロパティー

名前	説明
Watermark	接続数のウォーターマークレベル

MinPoolSize デクリメンターポリシー
クラス名: **org.jboss.jca.core.connectionmanager.pool.capacity.MinPoolSizeDecrementer**

MinPoolSize デクリメンターポリシーは、リクエストごとにのプールを最小サイズまでデクリメントします。このポリシーは、各アイドルタイムアウトリクエストの後に接続の数を制限したい場合に便利です。プールは先入れ先出し (FIFO) で操作します。

Size デクリメンターポリシー
クラス名: **org.jboss.jca.core.connectionmanager.pool.capacity.SizeDecrementer**

Size デクリメンターポリシーは、アイドルタイムアウトリクエストごとにプールを指定の接続数までデクリメントします。

表12.5 Size ポリシープロパティー

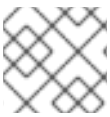
名前	説明
Size	破棄されるべき接続の数

このポリシーは、プールの使用度が徐々に減少することが予想されるためアイドルタイムアウトリクエストごとの追加接続数をデクリメントしたい場合に便利です。

プールは先入れ先出し (FIFO) で操作します。

TimedOut デクリメンターポリシー
クラス名: **org.jboss.jca.core.connectionmanager.pool.capacity.TimedOutDecrementer**

TimedOut デクリメンターポリシーは、アイドルタイムアウトリクエストごとにタイムアウトした接続をすべてプールから削除します。プールは先入れ後出し (FILO) で操作します。



注記

このポリシーはデフォルトのデクリメントポリシーです。

TimedOut/FIFO デクリメンターポリシー
クラス名: **org.jboss.jca.core.connectionmanager.pool.capacity.TimedOutFIFODecrementer**

TimedOutFIFO デクリメンターポリシーは、アイドルタイムアウトリクエストごとにタイムアウトした接続をすべてプールから削除します。プールは先入れ先出し (FIFO) で操作します。

Watermark デクリメンターポリシー
クラス名: **org.jboss.jca.core.connectionmanager.pool.capacity.WatermarkDecrementer**

Watermark デクリメンターポリシーは、アイドルタイムアウトリクエストごとにプールを指定の接続数までデクリメントします。このポリシーは、常時プールに指定数の接続を維持したい場合に便利です。プールは先入れ先出し (FIFO) で操作します。

表12.6 Watermark ポリシープロパティー

名前	説明
Watermark	接続数のウォーターマークレベル

12.14. エンリストメントトレース

XAResource インスタンスのエンリストメント中に発生するエラーを特定できるようにするために、エンリストメントトレースを記録することができます。エンリストメントトレースが有効である場合、必要時に正確なスタックトレースを作成できるように **jca** サブシステムはプール操作ごとに例外オブジェクトを作成しますが、これにはパフォーマンスのオーバーヘッドが発生します。

JBoss EAP 7.1 より、エンリストメントトレースはデフォルトで無効になっています。管理 CLI を使用してデータソースのエンリストメントトレースを有効にするには、**enlistment-trace** 属性を **true** に設定します。

非 XA データソースのエンリストメントトレースを有効にします。

```
data-source --name=DATASOURCE_NAME --enlistment-trace=true
```

XA データソースのエンリストメントトレースを有効にします。

```
xa-data-source --name=XA_DATASOURCE_NAME --enlistment-trace=true
```

警告

エンリストメントトレースを有効にするとパフォーマンスに影響することがあります。

12.15. データソース設定例

12.15.1. MySQL データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる MySQL データソースの設定例になります。

例: MySQL データソース例

```
<datasources>
  <datasource jndi-name="java:jboss/MySqlDS" pool-name="MySqlDS">
    <connection-url>jdbc:mysql://localhost:3306/jbossdb</connection-url>
    <driver>mysql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
```

```
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="mysql" module="com.mysql">
      <driver-class>com.mysql.cj.jdbc.Driver</driver-class>
      <xa-datasource-class>com.mysql.cj.jdbc.MysqlXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

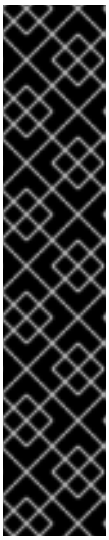
例: **MySQL JDBC** ドライバー**module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-8.0.12.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MySQL JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.mysql --resources=/path/to/mysql-connector-java-8.0.12.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. MySQL JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-name=mysql,driver-module-
name=com.mysql,driver-xa-datasource-class-name=com.mysql.cj.jdbc.MysqlXADataSource,
driver-class-name=com.mysql.cj.jdbc.Driver)
```

3. MySQL データソースを追加します。

```
data-source add --name=MySqlDS --jndi-name=java:jboss/MySqlDS --driver-name=mysql --
connection-url=jdbc:mysql://localhost:3306/jbossdb --user-name=admin --password=admin --
validate-on-match=true --background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter
```

12.15.2. MySQL XA データソースの例

以下は、XA データソースプロパティー、基本のセキュリティー、およびバリデーションオプションが含まれる MySQL XA データソースの設定例になります。

例: **MySQL XA** データソースの設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/MySqlXADS" pool-name="MySqlXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      mysqldb
    </xa-datasource-property>
    <driver>mysql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="mysql" module="com.mysql">
      <driver-class>com.mysql.cj.jdbc.Driver</driver-class>
      <xa-datasource-class>com.mysql.cj.jdbc.MysqlXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

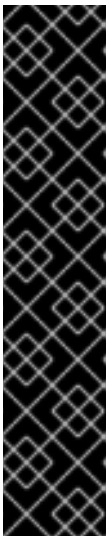
例: **MySQL JDBC** ドライバー**module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-8.0.12.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MySQL JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.mysql --resources=/path/to/mysql-connector-java-8.0.12.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. MySQL JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-name=mysql,driver-module-
name=com.mysql,driver-xa-datasource-class-name=com.mysql.cj.jdbc.MysqlXADataSource,
driver-class-name=com.mysql.cj.jdbc.Driver)
```

3. MySQL XA データソースを追加します。

```
xa-data-source add --name=MySqlXADS --jndi-name=java:jboss/MySqlXADS --driver-
name=mysql --user-name=admin --password=admin --validate-on-match=true --background-
validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter --xa-
datasource-properties={"ServerName"=>"localhost","DatabaseName"=>"mysqldb"}
```

12.15.3. PostgreSQL データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる PostgreSQL データソースの設定例になります。

例: **PostgreSQL** データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/PostgresDS" pool-name="PostgresDS">
    <connection-url>jdbc:postgresql://localhost:5432/postgresdb</connection-url>
    <driver>postgresql</driver>
    <security>
        <user-name>admin</user-name>
        <password>admin</password>
    </security>
    <validation>
        <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker"/>
        <validate-on-match>true</validate-on-match>
        <background-validation>false</background-validation>
        <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="postgresql" module="com.postgresql">
        <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

例: **PostgreSQL JDBC** ドライバーの **module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.postgresql">
  <resources>
    <resource-root path="postgresql-42.x.y.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

上記の例で、**42.x.y** をドライバーバージョン番号に置き換えてください。

関連情報

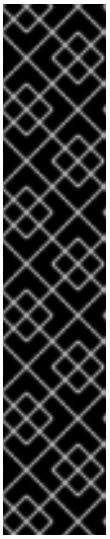
- JDBC ドライバーおよびインストール方法については、JDBC ドライバー を参照してください。
- JBoss EAP 7.4 の一部としてテストされた JDBC ドライバーに関する詳細は、JBoss EAP 7.4 Supported Configurations を参照してください。

管理 **CLI** コマンドの例
以下の CLI コマンドを使用して、PostgreSQL JDBC ドライバーおよび PostgreSQL データソースを JDBC API に追加できます。

1. PostgreSQL JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.postgresql --resources=/path/to/postgresql-42.x.y.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```

上記の例で、**42.x.y** をドライバーバージョン番号に置き換えてください。



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. PostgreSQL JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=postgresql:add(driver-name=postgresql,driver-module-
name=com.postgresql,driver-xa-datasource-class-
name=org.postgresql.xa.PGXADataSource)
```

3. PostgreSQL データソースを追加します。

```
data-source add --name=PostgresDS --jndi-name=java:jboss/PostgresDS --driver-
name=postgresql --connection-url=jdbc:postgresql://localhost:5432/postgresdb --user-
name=admin --password=admin --validate-on-match=true --background-validation=false --
valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker
--exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter
```

12.15.4. PostgreSQL XA データソースの例

以下は、XA データソースプロパティー、基本のセキュリティー、およびバリデーションオプションが含まれる PostgreSQL XA データソースの設定例になります。

例: **PostgreSQL XA** データソースの例

```
<datasources>
  <xa-datasource jndi-name="java:jboss/PostgresXADS" pool-name="PostgresXADS">
    <xa-datasource-property name="ServerName">
      localhost
```

```
    </xa-datasource-property>
    <xa-datasource-property name="PortNumber">
      5432
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      postgresdb
    </xa-datasource-property>
    <driver>postgresql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="postgresql" module="com.postgresql">
      <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

例: **PostgreSQL JDBC** ドライバーの **module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.postgresql">
  <resources>
    <resource-root path="postgresql-42.x.y.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

上記の例で、**42.x.y** をドライバーバージョン番号に置き換えてください。

関連情報

- JDBC ドライバーおよびインストール方法については、JDBC ドライバー を参照してください。
- JBoss EAP 7.4 の一部としてテストされた JDBC ドライバーに関する詳細は、JBoss EAP 7.4 Supported Configurations を参照してください。

管理 **CLI** コマンドの例

以下の CLI コマンドを使用して、PostgreSQL JDBC ドライバーおよび PostgreSQL データソースを JDBC API に追加できます。

1. PostgreSQL JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.postgresql --resources=/path/to/postgresql-42.x.y.jar --dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```

上記の例で、**42.x.y** をドライバーバージョン番号に置き換えてください。

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. PostgreSQL JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=postgresql:add(driver-name=postgresql,driver-module-name=com.postgresql,driver-xa-datasource-class-name=org.postgresql.xa.PGXADataSource)
```

3. PostgreSQL XA データソースを追加します。

```
xa-data-source add --name=PostgresXADS --jndi-name=java:jboss/PostgresXADS --driver-name=postgresql --user-name=admin --password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter --xa-datasource-properties={"ServerName"=>"localhost","PortNumber"=>"5432","DatabaseName"=>"postgresdb"}
```

12.15.5. Oracle データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる Oracle データソースの設定例になります。

例: **Oracle** データソースの例

```
<datasources>
  <datasource jndi-name="java:jboss/OracleDS" pool-name="OracleDS">
    <connection-url>jdbc:oracle:thin:@localhost:1521:XE</connection-url>
    <driver>oracle</driver>
```

```
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="oracle" module="com.oracle">
      <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

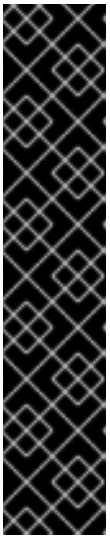
例: Oracle JDBC ドライバーの **module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
    <resources>
        <resource-root path="ojdbc7.jar"/>
    </resources>
    <dependencies>
      <module name="javaee.api"/>
      <module name="sun.jdk"/>
      <module name="ibm.jdk"/>
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Oracle JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.oracle --resources=/path/to/ojdbc7.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. Oracle JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=oracle:add(driver-name=oracle,driver-module-name=com.oracle,driver-xa-datasource-class-name=oracle.jdbc.xa.client.OracleXADataSource)
```

3. Oracle データソースを追加します。

```
data-source add --name=OracleDS --jndi-name=java:jboss/OracleDS --driver-name=oracle --connection-url=jdbc:oracle:thin:@localhost:1521:XE --user-name=admin --password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter
```

12.15.6. Oracle XA データソースの例

重要

Oracle XA データソースにアクセスするユーザーは、以下の設定を適用しないと XA リカバリーが適切に操作しません。値 **user** は、JBoss EAP から Oracle に接続するために定義されたユーザーです。

- **GRANT SELECT ON sys.dba_pending_transactions TO user;**
- **GRANT SELECT ON sys.pending_trans$ TO user;**
- **GRANT SELECT ON sys.dba_2pc_pending TO user;**
- **GRANT EXECUTE ON sys.dbms_xa TO user;**

以下は、XA データソースプロパティー、基本のセキュリティー、およびバリデーションオプションが含まれる Oracle XA データソースの設定例になります。

例: **Oracle XA** データソースの設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/OracleXADS" pool-name="OracleXADS">
```

```
    <xa-datasource-property name="URL">
      jdbc:oracle:thin:@oracleHostName:1521:orcl
    </xa-datasource-property>
    <driver>oracle</driver>
    <xa-pool>
      <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="oracle" module="com.oracle">
      <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

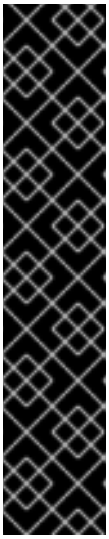
例: Oracle JDBC ドライバーの **module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
    <resources>
        <resource-root path="ojdbc7.jar"/>
    </resources>
    <dependencies>
      <module name="javaee.api"/>
      <module name="sun.jdk"/>
      <module name="ibm.jdk"/>
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Oracle JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.oracle --resources=/path/to/ojdbc7.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. Oracle JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=oracle:add(driver-name=oracle,driver-module-name=com.oracle,driver-xa-datasource-class-name=oracle.jdbc.xa.client.OracleXADataSource)
```

3. Oracle XA データソースを追加します。

```
xa-data-source add --name=OracleXADS --jndi-name=java:jboss/OracleXADS --driver-name=oracle --user-name=admin --password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter --same-rm-override=false --xa-datasource-properties={"URL"=>"jdbc:oracle:thin:@oracleHostName:1521:orcl"}
```

12.15.7. Oracle RAC データソースの例

これは、接続情報、基本的なセキュリティー、および検証オプションを使用した Real Application Cluster (RAC) データソース設定の例です。

例: Oracle RAC データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/OracleDS" pool-name="OracleDS">
    <connection-url>jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=host1)(PORT=1521))(ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521))</connection-url>
    <driver>oracle</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
```

```
        <background-validation>false</background-validation>
        <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="oracle" module="com.oracle">
        <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

例: Oracle JDBC ドライバーの module.xml ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
    <resources>
        <resource-root path="ojdbc7.jar"/>
    </resources>
    <dependencies>
      <module name="javaee.api"/>
      <module name="sun.jdk"/>
      <module name="ibm.jdk"/>
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

管理 CLI コマンドの例

この設定例を実現するには、以下の管理 CLI コマンドを使用します。

1. Oracle JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.oracle --resources=/path/to/ojdbc7.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```

2. Oracle JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=oracle:add(driver-name=oracle,driver-module-
name=com.oracle,driver-xa-datasource-class-
name=oracle.jdbc.xa.client.OracleXADataSource)
```

3. Oracle RAC データソースを追加します。

```
data-source add --name=OracleDS --jndi-name=java:jboss/OracleDS --driver-name=oracle -
-connection-url="jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=on)(ADDRESS=
(PROTOCOL=TCP)(HOST=host1)(PORT=1521))(ADDRESS=(PROTOCOL=TCP)
(HOST=host2)(PORT=1521))" --user-name=admin --password=admin --validate-on-
match=true --background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter
```

関連情報

- Oracle データソースの設定の詳細は、Oracle データソース を参照してください。

12.15.8. Oracle RAC XA データソースの例

以下は、XA データソースプロパティー、基本のセキュリティー、およびバリデーションオプションが含まれる RAC データソースの設定例になります。

例: Oracle RAC XA データソース設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/OracleXADS" pool-name="OracleXADS">
    <xa-datasource-property name="URL">
      jdbc:oracle:thin:@(DESCRIPTION=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)
(HOST=host1)(PORT=1521))(ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521))
    </xa-datasource-property>
    <driver>oracle</driver>
    <xa-pool>
      <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="oracle" module="com.oracle">
      <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

例: Oracle JDBC ドライバーの module.xml ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
    <resources>
        <resource-root path="ojdbc7.jar"/>
    </resources>
    <dependencies>
      <module name="javaee.api"/>
      <module name="sun.jdk"/>
      <module name="ibm.jdk"/>
      <module name="javax.api"/>
```

```
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI コマンドの例

この設定例を実現するには、以下の管理 CLI コマンドを使用します。

1. Oracle JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.oracle --resources=/path/to/ojdbc7.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```

2. Oracle JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=oracle:add(driver-name=oracle,driver-module-
name=com.oracle,driver-xa-datasource-class-
name=oracle.jdbc.xa.client.OracleXADataSource)
```

3. Oracle RAC XA データソースを追加します。

```
xa-data-source add --name=OracleXADS --jndi-name=java:jboss/OracleXADS --driver-
name=oracle --user-name=admin --password=admin --validate-on-match=true --
background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter --same-rm-
override=false --xa-datasource-properties={"URL"=>"jdbc:oracle:thin:@(DESCRIPTION=
(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)(HOST=host1)(PORT=1521))
(ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521))"}
```

12.15.9. Microsoft SQL Server データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる Microsoft SQL Server データソースの設定例になります。

例: **Microsoft SQL Server** データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/MSSQLDS" pool-name="MSSQLDS">
    <connection-url>jdbc:sqlserver://localhost:1433;DatabaseName=MyDatabase</connection-url>
    <driver>sqlserver</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter"/>
    </validation>
  </datasource>
```

```
  <drivers>
    <driver name="sqlserver" module="com.microsoft">
      <xa-datasource-class>com.microsoft.sqlserver.jdbc.SQLServerXADataSource</xa-datasource-
class>
    </driver>
  </drivers>
</datasources>
```

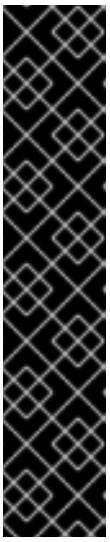
例: **Microsoft SQL Server JDBC** ドライバーの**module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.microsoft">
  <resources>
    <resource-root path="sqljdbc42.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
    <module name="javax.xml.bind.api"/>
  </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Microsoft SQL Server の JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.microsoft --resources=/path/to/sqljdbc42.jar --
dependencies=javax.api,javax.transaction.api,javax.xml.bind.api,javaee.api,sun.jdk,ibm.jdk
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. Microsoft SQL Server の JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=sqlserver:add(driver-name=sqlserver,driver-module-
name=com.microsoft,driver-xa-datasource-class-
name=com.microsoft.sqlserver.jdbc.SQLServerXADataSource)
```

3. Microsoft SQL Server データソースを追加します。

```
data-source add --name=MSSQLDS --jndi-name=java:jboss/MSSQLDS --driver-
name=sqlserver --connection-
url=jdbc:sqlserver://localhost:1433;DatabaseName=MyDatabase --user-name=admin --
password=admin --validate-on-match=true --background-validation=false --valid-connection-
checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter
```

12.15.10. Microsoft SQL Server XA データソースの例

以下は、XA データソースプロパティー、基本のセキュリティー、およびバリデーションオプションが含まれる Microsoft SQL Server XA データソースの設定例になります。

例: **Microsoft SQL Server XA** データソースの設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/MSSQLXADS" pool-name="MSSQLXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      mssqldb
    </xa-datasource-property>
    <xa-datasource-property name="SelectMethod">
      cursor
    </xa-datasource-property>
    <driver>sqlserver</driver>
    <xa-pool>
      <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="sqlserver" module="com.microsoft">
      <xa-datasource-class>com.microsoft.sqlserver.jdbc.SQLServerXADataSource</xa-datasource-
class>
    </driver>
  </drivers>
</datasources>
```

例: **Microsoft SQL Server JDBC** ドライバーの**module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.microsoft">
  <resources>
    <resource-root path="sqljdbc42.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
    <module name="javax.xml.bind.api"/>
  </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Microsoft SQL Server の JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.microsoft --resources=/path/to/sqljdbc42.jar --
dependencies=javax.api,javax.transaction.api,javax.xml.bind.api,javaee.api,sun.jdk,ibm.jdk
```

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. Microsoft SQL Server の JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=sqlserver:add(driver-name=sqlserver,driver-module-
name=com.microsoft,driver-xa-datasource-class-
name=com.microsoft.sqlserver.jdbc.SQLServerXADataSource)
```

3. Microsoft SQL Server XA データソースを追加します。

```
xa-data-source add --name=MSSQLXADS --jndi-name=java:jboss/MSSQLXADS --driver-
name=sqlserver --user-name=admin --password=admin --validate-on-match=true --
background-validation=false --background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker --
exception-sorter-class-
```

```
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter --same-rm-override=false --xa-datasource-properties={"ServerName"=>"localhost","DatabaseName"=>"mssqldb","SelectMethod"=>"cursor"}
```

12.15.11. IBM DB2 データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる IBM DB2 データソースの設定例になります。

例: IBM DB2 データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/DB2DS" pool-name="DB2DS">
    <connection-url>jdbc:db2://localhost:50000/ibmdb2db</connection-url>
    <driver>ibmdb2</driver>
    <pool>
      <min-pool-size>0</min-pool-size>
      <max-pool-size>50</max-pool-size>
    </pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="ibmdb2" module="com.ibm">
       <xa-datasource-class>com.ibm.db2.jcc.DB2XADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

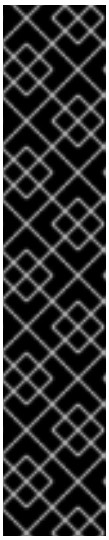
例: IBM DB2 JDBC ドライバーの**module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm">
  <resources>
    <resource-root path="db2jcc4.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. IBM DB2 の JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.ibm --resources=/path/to/db2jcc4.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. IBM DB2 の JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=ibmdb2:add(driver-name=ibmdb2,driver-module-
name=com.ibm,driver-xa-datasource-class-name=com.ibm.db2.jcc.DB2XADataSource)
```

3. IBM DB2 データソースを追加します。

```
data-source add --name=DB2DS --jndi-name=java:jboss/DB2DS --driver-name=ibmdb2 --
connection-url=jdbc:db2://localhost:50000/ibmdb2db --user-name=admin --password=admin
--validate-on-match=true --background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker --exception-
sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter --min-
pool-size=0 --max-pool-size=50
```

12.15.12. IBM DB2 XA のデータソースの例

以下は、XA データソースプロパティー、基本のセキュリティー、およびバリデーションオプションが含まれる IBM DB2 XA データソースの設定例になります。

例: **IBM DB2 XA** データソースの設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/DB2XADS" pool-name="DB2XADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      ibmdb2db
    </xa-datasource-property>
    <xa-datasource-property name="PortNumber">
```

```
      50000
    </xa-datasource-property>
    <xa-datasource-property name="DriverType">
      4
    </xa-datasource-property>
    <driver>ibmdb2</driver>
    <xa-pool>
      <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <recovery>
      <recover-plugin class-name="org.jboss.jca.core.recovery.ConfigurableRecoveryPlugin">
        <config-property name="EnableIsValid">
          false
        </config-property>
        <config-property name="IsValidOverride">
          false
        </config-property>
        <config-property name="EnableClose">
          false
        </config-property>
      </recover-plugin>
    </recovery>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="ibmdb2" module="com.ibm">
        <xa-datasource-class>com.ibm.db2.jcc.DB2XADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

例: IBM DB2 JDBC ドライバーの**module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm">
  <resources>
    <resource-root path="db2jcc4.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
```

```
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. IBM DB2 の JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.ibm --resources=/path/to/db2jcc4.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. IBM DB2 の JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=ibmdb2:add(driver-name=ibmdb2,driver-module-
name=com.ibm,driver-xa-datasource-class-name=com.ibm.db2.jcc.DB2XADataSource)
```

3. IBM DB2 XA データソースを追加します。

```
xa-data-source add --name=DB2XADS --jndi-name=java:jboss/DB2XADS --driver-
name=ibmdb2 --user-name=admin --password=admin --validate-on-match=true --
background-validation=false --background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker --exception-
sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter --same-
rm-override=false --recovery-plugin-class-
name=org.jboss.jca.core.recovery.ConfigurableRecoveryPlugin --recovery-plugin-properties=
{"EnableIsValid"=>"false","IsValidOverride"=>"false","EnableClose"=>"false"} --xa-
datasource-properties=
{"ServerName"=>"localhost","DatabaseName"=>"ibmdb2db","PortNumber"=>"50000","DriverT
ype"=>"4"}
```

12.15.13. Sybase データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる Sybase データソースの設定例になります。

例: **Sybase** データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/SybaseDB" pool-name="SybaseDB">
    <connection-url>jdbc:sybase:Tds:localhost:5000/DATABASE?
JCONNECT_VERSION=6</connection-url>
    <driver>sybase</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="sybase" module="com.sybase">
      <xa-datasource-class>com.sybase.jdbc4.jdbc.SybXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

例: Sybase JDBC ドライバーの**module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.sybase">
    <resources>
      <resource-root path="jconn4.jar"/>
    </resources>
    <dependencies>
      <module name="javaee.api"/>
      <module name="sun.jdk"/>
      <module name="ibm.jdk"/>
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Sybase JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.sybase --resources=/path/to/jconn4.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. Sybase JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=sybase:add(driver-name=sybase,driver-module-name=com.sybase,driver-xa-datasource-class-name=com.sybase.jdbc4.jdbc.SybXADataSource)
```

3. Sybase データソースを追加します。

```
data-source add --name=SybaseDB --jndi-name=java:jboss/SybaseDB --driver-name=sybase --connection-url=jdbc:sybase:Tds:localhost:5000/DATABASE?JCONNECT_VERSION=6 --user-name=admin --password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter
```

12.15.14. Sybase XA データソースの例

以下は、XA データソースプロパティー、基本のセキュリティー、およびバリデーションオプションが含まれる Sybase XA データソースの設定例になります。

例: **Sybase XA** データソースの設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/SybaseXADS" pool-name="SybaseXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      mydatabase
    </xa-datasource-property>
    <xa-datasource-property name="PortNumber">
      4100
    </xa-datasource-property>
    <xa-datasource-property name="NetworkProtocol">
      Tds
    </xa-datasource-property>
    <driver>sybase</driver>
    <xa-pool>
```

```
      <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="sybase" module="com.sybase">
      <xa-datasource-class>com.sybase.jdbc4.jdbc.SybXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

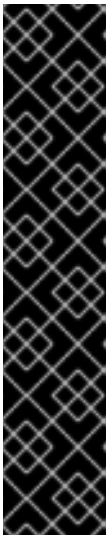
例: Sybase JDBC ドライバーの**module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.sybase">
    <resources>
      <resource-root path="jconn4.jar"/>
    </resources>
    <dependencies>
      <module name="javaee.api"/>
      <module name="sun.jdk"/>
      <module name="ibm.jdk"/>
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Sybase JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.sybase --resources=/path/to/jconn4.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. Sybase JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=sybase:add(driver-name=sybase,driver-module-name=com.sybase,driver-xa-datasource-class-name=com.sybase.jdbc4.jdbc.SybXADataSource)
```

3. Sybase XA データソースを追加します。

```
xa-data-source add --name=SybaseXADS --jndi-name=java:jboss/SybaseXADS --driver-name=sybase --user-name=admin --password=admin --validate-on-match=true --background-validation=false --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter --same-rm-override=false --xa-datasource-properties={"ServerName"=>"localhost","DatabaseName"=>"mydatabase","PortNumber"=>"4100","NetworkProtocol"=>"Tds"}
```

12.15.15. MariaDB データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる MariaDB データソースの設定例になります。

例: MariaDB データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/MariaDBDS" pool-name="MariaDBDS">
    <connection-url>jdbc:mariadb://localhost:3306/jbossdb</connection-url>
    <driver>mariadb</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
```

```
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="mariadb" module="org.mariadb">
      <driver-class>org.mariadb.jdbc.Driver</driver-class>
      <xa-datasource-class>org.mariadb.jdbc.MySQLDataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

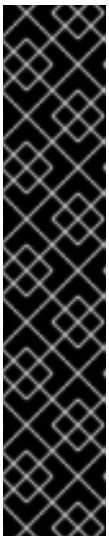
例: MariaDB JDBC ドライバーの **module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="org.mariadb">
  <resources>
    <resource-root path="mariadb-java-client-3.3.0.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="org.slf4j"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MariaDB JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=org.mariadb --resources=/path/to/mariadb-java-client-3.3.0.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api,org.slf4j
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. MariaDB JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mariadb:add(driver-name=mariadb,driver-module-name=org.mariadb,driver-xa-datasource-class-name=org.mariadb.jdbc.MySQLDataSource,driver-class-name=org.mariadb.jdbc.Driver)
```

3. MariaDB データソースを追加します。

```
data-source add --name=MariaDBDS --jndi-name=java:jboss/MariaDBDS --driver-name=mariadb --connection-url=jdbc:mariadb://localhost:3306/jbossdb --user-name=admin --password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter
```

12.15.16. MariaDB XA データソースの例

以下は、XA データソースプロパティー、基本のセキュリティー、およびバリデーションオプションが含まれる MariaDB XA データソースの設定例になります。

例: **MariaDB XA** データソースの設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/MariaDBXADS" pool-name="MariaDBXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      mariadbdb
    </xa-datasource-property>
    <driver>mariadb</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="mariadb" module="org.mariadb">
      <driver-class>org.mariadb.jdbc.Driver</driver-class>
      <xa-datasource-class>org.mariadb.jdbc.MySQLDataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

例: **MariaDB JDBC** ドライバーの **module.xml** ファイル

```
<?xml version="1.0" ?>
```

```
<module xmlns="urn:jboss:module:1.1" name="org.mariadb">
  <resources>
    <resource-root path="mariadb-java-client-3.3.0.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="org.slf4j"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MariaDB JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=org.mariadb --resources=/path/to/mariadb-java-client-3.3.0.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api,org.slf4j
```

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. MariaDB JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mariadb:add(driver-name=mariadb,driver-module-
name=org.mariadb,driver-xa-datasource-class-name=org.mariadb.jdbc.MySQLDataSource,
driver-class-name=org.mariadb.jdbc.Driver)
```

3. MariaDB XA データソースを追加します。

```
xa-data-source add --name=MariaDBXADS --jndi-name=java:jboss/MariaDBXADS --driver-
name=mariadb --user-name=admin --password=admin --validate-on-match=true --
background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter --xa-
datasource-properties={"ServerName"=>"localhost","DatabaseName"=>"mariadbdb"}
```

12.15.17. MariaDB Galera Cluster データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる MariaDB Galera Cluster データソースの設定例になります。

警告

MariaDB Galera Cluster は XA トランザクションをサポートしません。

例: **MariaDB Galera Cluster** データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/MariaDBGaleraClusterDS" pool-
name="MariaDBGaleraClusterDS">
    <connection-url>jdbc:mariadb://192.168.1.1:3306,192.168.1.2:3306/jbossdb</connection-url>
    <driver>mariadb</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="mariadb" module="org.mariadb">
      <driver-class>org.mariadb.jdbc.Driver</driver-class>
      <xa-datasource-class>org.mariadb.jdbc.MySQLDataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

例: **MariaDB JDBC** ドライバーの **module.xml** ファイル

```
<?xml version='1.0' encoding='UTF-8'?>
<module xmlns="urn:jboss:module:1.1" name="org.mariadb">
  <resources>
    <resource-root path="mariadb-java-client-3.3.0.jar"/>
  </resources>
  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="org.slf4j"/>
```

```
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MariaDB JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=org.mariadb --resources=/path/to/mariadb-java-client-3.3.0.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api,org.slf4j
```

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. MariaDB JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mariadb:add(driver-name=mariadb,driver-module-
name=org.mariadb,driver-xa-datasource-class-name=org.mariadb.jdbc.MySQLDataSource,
driver-class-name=org.mariadb.jdbc.Driver)
```

3. MariaDB Galera Cluster データソースを追加します。

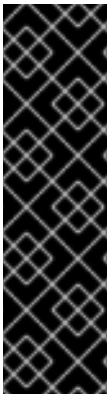
```
data-source add --name=MariaDBGaleraClusterDS --jndi-
name=java:jboss/MariaDBGaleraClusterDS --driver-name=mariadb --connection-
url=jdbc:mariadb://192.168.1.1:3306,192.168.1.2:3306/jbossdb --user-name=admin --
password=admin --validate-on-match=true --background-validation=false --valid-connection-
checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter
```

第13章 AGROAL でのデータソース管理

13.1. AGROAL データソースサブシステム

新しい **datasources-agroal** サブシステムが JBoss EAP 7.2 に導入されました。これは、Agroal プロジェクトによる、新しい高パフォーマンスな直接接続プールです。現在の Jakarta Connectors ベースの **datasources** サブシステムの代わりに使用することができます。

datasources-agroal サブシステムはデフォルトでは使用できません。 この新しい実装を 有効 にして使用する必要があります。



重要

datasources-agroal サブシステムはテクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

13.2. AGROAL データソースサブシステムの有効化

datasources-agroal サブシステムは、デフォルトの JBoss EAP 設定では有効になっていません。以下の管理 CLI コマンドを使用して有効にします。

1. 拡張を追加します。

```
/extension=org.wildfly.extension.datasources-agroal:add
```

2. サブシステムを追加します。

```
/subsystem=datasources-agroal:add
```

3. 変更を反映するためにサーバーをリロードします。

```
reload
```

これにより、サーバー設定ファイルに以下の XML が追加されます。

```
<server xmlns="urn:jboss:domain:8.0">
  <extensions>
    ...
    <extension module="org.wildfly.extension.datasources-agroal"/>
    ...
  </extensions>
  ...
  <subsystem xmlns="urn:jboss:domain:datasources-agroal:1.0"/>
  ...
</server>
```

13.3. AGROAL データソースに対するコアモジュールとしての JDBC ドライバーのインストール

JBoss EAP でアプリケーションが使用する Agroal データソースを定義する前に、最初に適切な JDBC ドライバーをインストールする必要があります。

Agroal データソースに対して JDBC ドライバーをコアモジュールとしてインストールするには、最初に JDBC ドライバーをコアモジュールとして追加 し、datasources-agroal サブシステムで **JDBC** ドライバーを登録 する必要があります。

13.3.1. JDBC ドライバーをコアモジュールとして追加

以下の手順に従うと、管理 CLI を使用して JDBC ドライバーをコアモジュールとしてインストールすることができます。

1. JDBC ドライバーをダウンロードします。
   データベースのベンダーから適切な JDBC ドライバーをダウンロードします。一般的なデータベースの JDBC ドライバーをダウンロードできる場所については、JDBC ドライバーのダウンロードできる場所 を参照してください。

   JDBC ドライバーの JAR ファイルが ZIP または TAR アーカイブ内に含まれている場合は、必ずそのアーカイブをデプロイメントしてください。

2. JBoss EAP サーバーを起動します。

3. 管理 CLI を起動します。

   ```
   $ EAP_HOME/bin/jboss-cli.sh
   ```

4. **module add** 管理 CLI コマンドを使用して新しいコアモジュールを追加します。

   ```
   [disconnected /] module add --name=MODULE_NAME --resources=PATH_TO_JDBC_JAR
   --dependencies=DEPENDENCIES
   ```

   例

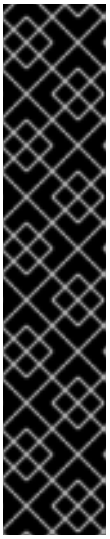
   次のコマンドは、MySQL JDBC ドライバーモジュールを追加します。

   ```
   [disconnected /] module add --name=com.mysql --resources=/path/to/mysql-connector-java-
   8.0.12.jar --dependencies=javax.transaction.api,sun.jdk,ibm.jdk,javaee.api,javax.api
   ```

   例

   管理 CLI を起動して新しいコアモジュールを 1 ステップで追加するには、次のコマンドを使用します。

   ```
   $ EAP_HOME/bin/jboss-cli.sh --command="module add --name=MODULE_NAME --
   resources=PATH_TO_JDBC_JAR --dependencies=DEPENDENCIES"
   ```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

module --help を実行すると、このコマンドを使用したモジュールの追加および削除の詳細を表示できます。

次に、アプリケーションデータソースによって参照されるよう、JDBC ドライバーとして登録する必要があります。

13.3.2. Agroal データソースの JDBC ドライバーの登録

ドライバーが コアモジュールとしてインストール されたら、以下の管理 CLI コマンドを使用して JDBC ドライバーとして登録する必要があります。マネージドドメインを実行している場合は、コマンドの前に **/profile=PROFILE_NAME** を付けてください。

```
/subsystem=datasources-
agroal/driver=DRIVER_NAME:add(module=MODULE_NAME,class=CLASS_NAME)
```

CLASS_NAME は、非 XA データソースの場合は **java.sql.Driver** または **javax.sql.DataSource**、XA データソースの場合は **javax.sql.XADataSource** を実装する完全修飾クラス名である必要があります。

13.4. AGROAL 非 XA データソースの設定

重要

Agroal データソースの使用は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

13.4.1. Agroal データソースの作成

以下の管理 CLI コマンドは Agroal データソースを作成します。マネージドドメインを実行している場合は、コマンドの前に **/profile=PROFILE_NAME** を付けてください。

```
/subsystem=datasources-agroal/datasource=DATASOURCE_NAME:add(jndi-name=JNDI_NAME,connection-factory={driver=DRIVER_NAME,url=URL},connection-pool={max-size=MAX_POOL_SIZE})
```

利用できる Agroal データソース属性の完全リストは、Agroal データソース属性 を参照してください。

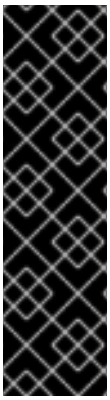
Agroal データソース設定の例は、MySQL Agroal データソースの例 を参照してください。

13.4.2. Agroal データソースの削除

以下の管理 CLI コマンドは Agroal データソースを削除します。マネージドドメインを実行している場合は、コマンドの前に **/profile=PROFILE_NAME** を付けてください。

```
/subsystem=datasources-agroal/datasource=DATASOURCE_NAME:remove
```

13.5. AGROAL XA データソースの設定



重要

Agroal データソースの使用は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

13.5.1. Agroal XA データソースの作成

以下の管理 CLI コマンドは Agroal XA データソースを作成します。マネージドドメインを実行している場合は、コマンドの前に **/profile=PROFILE_NAME** を付けてください。

```
/subsystem=datasources-agroal/xa-datasource=XA_DATASOURCE_NAME:add(jndi-name=JNDI_NAME,connection-factory={driver=DRIVER_NAME,connection-properties={ServerName=HOST_NAME,PortNumber=PORT,DatabaseName=DATABASE_NAME}},connection-pool={max-size=MAX_POOL_SIZE})
```

利用できる Agroal データソース属性の完全リストは、Agroal データソース属性 を参照してください。

Agroal XA データソース設定の例は、MySQL Agroal XA データソースの例 を参照してください。

13.5.2. Agroal XA データソースの削除

以下の管理 CLI コマンドは Agroal XA データソースを削除します。マネージドドメインを実行している場合は、コマンドの前に **/profile=PROFILE_NAME** を付けてください。

```
/subsystem=datasources-agroal/xa-datasource=DATASOURCE_NAME:remove
```

13.6. AGROAL データソースの設定例

13.6.1. MySQL Agroal データソースの例

以下は、接続設定および基本のセキュリティー設定が含まれる、MySQL Agroal データソースの設定例になります。

例: MySQL Agroal データソース設定

```
<subsystem xmlns="urn:jboss:domain:datasources-agroal:1.0">
    <datasource name="ExampleAgroalDS" jndi-name="java:jboss/datasources/ExampleAgroalDS">
        <connection-factory driver="mysql" url="jdbc:mysql://localhost:3306/jbossdb" username="admin"
password="admin"/>
        <connection-pool max-size="30"/>
    </datasource>
    <drivers>
        <driver name="mysql" module="com.mysql" class="com.mysql.cj.jdbc.Driver"/>
    </drivers>
</subsystem>
```

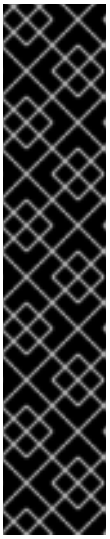
例: MySQL JDBC ドライバー **module.xml** ファイル

```
<?xml version='1.0' encoding='UTF-8'?>

<module xmlns="urn:jboss:module:1.1" name="com.mysql">

    <resources>
        <resource-root path="mysql-connector-java-8.0.12.jar"/>
    </resources>

    <dependencies>
      <module name="javaee.api"/>
      <module name="sun.jdk"/>
      <module name="ibm.jdk"/>
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MySQL JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.mysql --resources=/path/to/mysql-connector-java-8.0.12.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```



重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. MySQL JDBC ドライバーを登録します。

```
/subsystem=datasources-
agroal/driver=mysql:add(module=com.mysql,class=com.mysql.cj.jdbc.Driver)
```

3. MySQL データソースを追加します。

```
/subsystem=datasources-agroal/datasource=ExampleAgroalDS:add(jndi-
name=java:jboss/datasources/ExampleAgroalDS,connection-factory=
{driver=mysql,url=jdbc:mysql://localhost:3306/jbossdb,username=admin,password=admin},conn
ection-pool={max-size=30})
```

13.6.2. MySQL Agroal XA データソースの例

以下は、XA データソースプロパティーおよび基本のセキュリティー設定が含まれる、MySQL Agroal XA データソースの設定例になります。

例: **MySQL Agroal XA** データソース設定

```
<subsystem xmlns="urn:jboss:domain:datasources-agroal:1.0">
    <xa-datasource name="ExampleAgroalXADS" jndi-
name="java:jboss/datasources/ExampleAgroalXADS">
        <connection-factory driver="mysqlXA" username="admin" password="admin">
            <connection-properties>
                <property name="ServerName" value="localhost"/>
                <property name="PortNumber" value="3306"/>
                <property name="DatabaseName" value="jbossdb"/>
            </connection-properties>
        </connection-factory>
        <connection-pool max-size="30"/>
    </xa-datasource>
    <drivers>
        <driver name="mysqlXA" module="com.mysql" class="com.mysql.cj.jdbc.MysqlXADataSource"/>
    </drivers>
</subsystem>
```

例: **MySQL JDBC** ドライバー**module.xml** ファイル

```
<?xml version='1.0' encoding='UTF-8'?>

<module xmlns="urn:jboss:module:1.1" name="com.mysql">

  <resources>
    <resource-root path="mysql-connector-java-8.0.12.jar"/>
  </resources>

  <dependencies>
    <module name="javaee.api"/>
    <module name="sun.jdk"/>
    <module name="ibm.jdk"/>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 **CLI** コマンドの例
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MySQL JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.mysql --resources=/path/to/mysql-connector-java-8.0.12.jar --
dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、マネージドドメインでの使用や、リモートによる管理 CLI への接続時には適していません。本番環境ではモジュールを手作業で 追加 および 削除 してください。

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

2. MySQL XA JDBC ドライバーを登録します。

```
/subsystem=datasources-
agroal/driver=mysqlXA:add(module=com.mysql,class=com.mysql.cj.jdbc.MysqlXADataSource)
```

3. MySQL XA データソースを追加します。

```
/subsystem=datasources-agroal/xa-datasource=ExampleAgroalXADS:add(jndi-
name=java:jboss/datasources/ExampleAgroalXADS,connection-factory=
{driver=mysqlXA,connection-properties=
{ServerName=localhost,PortNumber=3306,DatabaseName=jbossdb},username=admin,passwo
rd=admin},connection-pool={max-size=30})
```

第14章 TRANSACTIONS サブシステムの設定

transactions サブシステムは、タイムアウト値、トランザクションロギング、統計の収集、TS を使用するかどうかなど、トランザクションマネージャー (TM) オプション の設定を可能にします。JBoss EAP は、Narayana フレームワークを使用してトランザクションサービスを提供します。このフレームワークは、Jakarta Transactions、JTS、Web Service トランザクションなどの標準ベースの幅広いトランザクションプロトコルのサポートを利用します。

詳細は、Managing Transactions on JBoss EAP を参照してください。

第15章 ORB 設定

15.1. COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

Common Object Request Broker Architecture (CORBA) は、アプリケーションとサービスが複数の互換性がない言語で記述され、異なるプラットフォームでホストされる場合でも、アプリケーションとサービスが連携することを可能にする標準です。CORBA のリクエストは Object Request Broker (ORB) というサーバーサイドコンポーネントが仲介します。JBoss EAP は、Open JDK ORB コンポーネントを用いて ORB インスタンスを提供します。

ORB は JTS トランザクションに対して内部的に使用され、ユーザー独自のアプリケーションが使用することもできます。

注記

Object Transaction Service (OTS) は、CORBA の一部を形成するクロスプラットフォームサービスです。OTS 仕様は、オブジェクト管理グループにより維持されます。JTS はトランザクションマネージャーの構築の仕様で、JTS は OTS 仕様に基づいて設計されました。

CORBA およびそのコンポーネントの詳細は、Common Object Request Broker Architecture を参照してください。

15.2. JTS の ORB の設定

JBoss EAP のデフォルトインストールでは、トランザクションの ORB (Object Request Broker: オブジェクトリクエストブローカー) のサポートが無効になっています。管理 CLI または管理コンソールを使用して **iiop-openjdk** サブシステムの ORB を設定できます。

注記

iiop-openjdk サブシステムを利用できるのは、マネージドドメインで **full** または **full-ha** プロファイルを使用している場合や、スタンドアロンサーバーの **standalone-full.xml** または **standalone-full-ha.xml** 設定ファイルを使用している場合です。

iiop-openjdk サブシステムで使用できる設定オプションのリストは、IIOP サブシステムの属性 を参照してください。

管理 CLI を使用した ORB の設定
管理 CLI を使用して ORB を設定できます。これは、JTS と使用するために行う ORB の最低限の設定です。

以下のマネージドドメインの管理 CLI コマンドは、**full** プロファイルを使用して設定できます。必要な場合は設定に応じてプロファイルを変更してください。スタンドアロンサーバーを使用している場合は、コマンドの **/profile=full** 部分を省略してください。

セキュリティーインターセプターの有効化
値を **identity** に設定し、**security** 属性を有効にします。

```
/profile=full/subsystem=iiop-openjdk:write-attribute(name=security,value=identity)
```

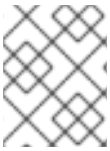
IIOP サブシステムでのトランザクションの有効化

JTS に対して ORB を有効にするには、**transactions** 属性の値をデフォルトの **spec** ではなく **full** に設定します。

```
/profile=full/subsystem=iiop-openjdk:write-attribute(name=transactions, value=full)
```

Transactions サブシステムでの JTS の有効化

```
/profile=full/subsystem=transactions:write-attribute(name=jts,value=true)
```



注記

JTS をアクティベートするにはリロードでは不十分なため、サーバーを再起動する必要があります。

管理コンソールを使用した ORB の設定

1. 管理コンソールの上部で、**Configuration** タブを選択します。マネージドドメインを使用する場合は、変更するプロファイルを選択する必要があります。
2. **Subsystems → IIOP (OpenJDK)** と選択し、表示 をクリックします。
3. 編集 をクリックし、必要に応じて属性を変更します。
4. 保存 をクリックして変更を保存します。

15.3. ELYTRON サブシステムで SSL/TLS を使用するよう IIOP を設定

SSL/TLS/ を使用するよう **iiop-openjdk** サブシステムを設定し、クライアントとサーバー間の通信をセキュア化することができます。**elytron** サブシステムおよびレガシーの **security** サブシステムは、**iiop-openjdk** サブシステムや JBoss EAP 内の他のサブシステムに SSL/TLS を設定するために必要なコンポーネントを提供します。以下の手順にしたがって、SSL/TLS に **elytron** サブシステムを使用するよう **iiop-openjdk** サブシステムを設定します。

1. 以下の管理 CLI コマンドを使用して、**iiop-openjdk** サブシステムの現在のレガシー SSL/TLS 設定を表示します。

```
/subsystem=iiop-openjdk:read-attribute(name=security-domain)
{
    "outcome" => "success",
    "result" => "iiopSSLSecurityDomain"
}
```

iiop-openjdk サブシステムは、SSL/TLS にレガシーの **security** サブシステムか **elytron** サブシステムのいずれかを使用する必要があります。両方のサブシステムを同時に使用することはできません。上記のコマンドは、 **iiop-openjdk** サブシステムが SSL/TLS の処理にレガシーのセキュリティードメインを使用していることを示しています。SSL/TLS に **elytron** サブシステムを使用するよう **iiop-openjdk** サブシステムを設定する前に、以下の参照を削除する必要があります。

```
/subsystem=iiop-openjdk:undefine-attribute(name=security-realm)
```

iiop-openjdk の **security-domain** 属性が定義されていない場合は、手順を続行できません。

2. **server-ssl-context** を作成します。

iiop-openjdk サブシステムで SSL/TLS を使用するには、**server-ssl-context** を定義する必要があります。JBoss EAP は、サーバーへの SSL/TLS 接続を確立するときに、**server-ssl-context** によって提供される設定を使用します。**server-ssl-context** の作成に関する詳細は、**How to Configure Server Security**の Enable One-way SSL/TLS for Applications using the Elytron Subsystem を参照してください。

3. **client-ssl-context** を作成します。
iiop-openjdk サブシステムで SSL/TLS を使用するには、**client-ssl-context** を定義する必要があります。JBoss EAP は、クライアントへの SSL/TLS 接続を確立するときに、**client-ssl-context** によって提供される設定を使用します。**client-ssl-context** の作成に関する詳細は、**How to Configure Server Security**の Using a client-ssl-context を参照してください。

4. **client-ssl-context** および **server-ssl-context** を使用するよう、**iiop-openjdk** サブシステムを設定します。

例: client-ssl-context および server-ssl-context の設定

```
batch

/subsystem=iiop-openjdk:write-attribute(name=client-ssl-context,value=iiopClientSSC)

/subsystem=iiop-openjdk:write-attribute(name=server-ssl-context,value=iiopServerSSC)

run-batch

reload
```

5. **iiop-openjdk** サブシステムを接続先および接続元とする接続の設定
以下の属性を調整すると、**iiop-openjdk** サブシステムを接続先および接続元とする接続を確立するときに SSL/TLS 接続を要求するかどうかを示すことができます。

- **iiop-openjdk** サブシステムで SSL のサポートを有効にするには、**support-ssl** を **true** に設定します。デフォルトは **false** です。
- **iiop-openjdk** サブシステムからの SSL/TLS 接続を要求するには、**client-requires-ssl** を **true** に設定します。デフォルトは **false** です。
- **iiop-openjdk** サブシステムへの SSL/TLS 接続を要求するには、**server-requires-ssl** を **true** に設定します。デフォルトは **false** です。これを **true** に設定すると、非 SSL IIOP ソケットへの接続をブロックすることに注意してください。
- **socket-binding** を調整するには、**ssl-socket-binding** を希望のバインディングに設定します。デフォルトは **iiop-ssl** です。

例: IIOP を接続先および接続元とする SSL/TLS 接続の設定

```
/subsystem=iiop-openjdk:write-attribute(name=support-ssl,value=true)

/subsystem=iiop-openjdk:write-attribute(name=client-requires-ssl,value=true)

/subsystem=iiop-openjdk:write-attribute(name=server-requires-ssl,value=true)

/subsystem=iiop-openjdk:write-attribute(name=ssl-socket-binding,value=iiop-ssl)

reload
```

第16章 JAKARTA CONNECTORS 管理

16.1. JAKARTA CONNECTORS について

Jakarta Connectors は、外部の異種 EIS (Enterprise Information System) に対して Jakarta EE システムの標準アーキテクチャーを定義します。EIS の例として、 エンタープライズリソースプランニング (ERP) システム、メインフレームトランザクション処理 (TP)、データベース、メッセージングシステムなどが挙げられます。リソースアダプター は Jakarta Connectors アーキテクチャーを実装するコンポーネントです。

Jakarta Connectors 1.7 は以下を管理するための機能を提供します。

- connections
- transactions
- security
- life-cycle
- work instances
- transaction inflow
- message inflow

16.2. リソースアダプター

リソースアダプターは、Jakarta Connectors 仕様を使用して Jakarta EE アプリケーションとエンタープライズ情報システム (EIS) との間の通信を提供するデプロイ可能な Jakarta EE コンポーネントです。EIS ベンダーの製品と Jakarta EE アプリケーションの統合を容易にするため、リソースアダプターは通常 EIS ベンダーによって提供されます。

エンタープライズ情報システムは、組織内における他のあらゆるソフトウェアシステムのことです。例としては、エンタープライズリソースプランニング (ERP) システム、データベースシステム、電子メールサーバー、商用メッセージングシステムなどが挙げられます。

リソースアダプターは、JBoss EAP にデプロイできる Resource Adapter Archive (RAR) ファイルでパッケージ化されます。また、RAR ファイルは、Enterprise Archive (EAR) デプロイメントにも含めることができます。

リソースアダプター自体は、IronJacamar プロジェクトによって提供される **resource-adapters** サブシステム内に定義されます。

16.3. JCA サブシステムの設定

jca サブシステムは、Jakarta Connectors およびリソースアダプターデプロイメントの一般的な設定を制御します。管理コンソールまたは管理 CLI を使用して **jca** サブシステムを設定できます。

設定する主な **jca** サブシステム要素は次のとおりです。

- アーカイブバリデーション
- Bean バリデーション

- ワークマネージャー
- 分散ワークマネージャー
- ブートストラップコンテキスト
- キャッシュ接続マネージャー

管理コンソールでの **jca** の設定

管理コンソールで **jca** サブシステムを設定するには、**Configuration → Subsystems → JCA** と選択し、表示 をクリックします。次に、該当するタブを選択します。

- **Configuration**: キャッシュ接続マネージャー、アーカイブバリデーション、および Bean バリデーションの設定が含まれます。これらの項目は独自のタブに含まれます。設定を変更するには、タブを開き、**Edit** リンクをクリックします。
- **Bootstrap Context**: 設定されたブートストラップコンテキストのリストが含まれます。新しいブートストラップコンテキストオブジェクトを追加、削除、および設定できます。各ブートストラップコンテキストをワークマネージャーに割り当てる必要があります。
- **Workmanager**: 設定されたワークマネージャーのリストが含まれます。新しいワークマネージャーを追加および削除でき、スレッドプールをここで設定できます。各ワークマネージャーは短時間実行されるスレッドプールを 1 つ持つことができ、任意で長時間実行されるスレッドプールを 1 つ持つことができます。
  スレッドプール属性を設定するには、選択したワークマネージャーの **Thread Pools** をクリックします。

管理 **CLI** での **jca** システムの設定

/subsystem=jca アドレスから管理 CLI で **jca** サブシステムを設定できます。マネージドドメインでは、コマンドの前に **/profile=PROFILE_NAME** を追加する必要があります。

注記

以下のセクションの表には、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-jca_5_0.xsd** のスキーマ定義ファイルで確認してください。

アーカイブバリデーション

デプロイメントユニットでアーカイブバリデーションを実行するかどうかを決定します。以下の表はアーカイブバリデーションに設定できる属性を表しています。

表16.1 アーカイブバリデーション属性

属性	デフォルト値	説明
enabled	true	アーカイブバリデーションが有効であるかどうかを指定します。
fail-on-error	true	アーカイブバリデーションのエラーレポートによってデプロイメントが失敗するかどうかを指定します

属性	デフォルト値	説明
fail-on-warn	false	アーカイブバリデーションの警告レポートによってデプロイメントが失敗するかどうかを指定します。

アーカイブバリデーションが有効な状態で、アーカイブが Jakarta Connectors 仕様を正しく実装しない場合、デプロイメント中に問題を説明するエラーメッセージが表示されます。以下に例を示します。

```
Severity: ERROR
Section: 19.4.2
Description: A ResourceAdapter must implement a "public int hashCode()" method.
Code: com.mycompany.myproject.ResourceAdapterImpl

Severity: ERROR
Section: 19.4.2
Description: A ResourceAdapter must implement a "public boolean equals(Object)" method.
Code: com.mycompany.myproject.ResourceAdapterImpl
```

アーカイブバリデーションが指定されていない場合は、アーカイブバリデーションが指定されているとみなされ、**enabled** 属性のデフォルトが **true** に設定されます。

Bean Validation

この設定は Bean の検証が実行されるかどうかを決定します。この仕様の詳細は、Jakarta Bean Validation 仕様 を参照してください。以下の表は Bean バリデーションに設定できる属性について表しています。

表16.2 Bean バリデーションの属性

属性	デフォルト値	説明
enabled	true	Bean バリデーションが有効であるかどうかを指定します。

Bean バリデーションが指定されていない場合は、Bean バリデーションが指定されているとみなされ、**enabled** 属性のデフォルトが **true** に設定されます。

ワークマネージャー

ワークマネージャーには次の 2 種類があります。

デフォルトワークマネージャー
: デフォルトのワークマネージャーおよびそのスレッドプール。

カスタムワークマネージャー
: カスタムワークマネージャーの定義およびそのスレッドプール。

ワークマネージャーに設定できる属性は下表のとおりです。

表16.3 ワークマネージャーの属性

属性	説明
name	ワークマネージャーの名前を指定します。

属性	説明
elytron-enabled	この属性は、**workmanager** の Elytron セキュリティーを有効にします。

ワークマネージャーには以下の子要素もあります。

表16.4 ワークマネージャーの子要素

子要素	説明
short-running-threads	標準の Work インスタンスのスレッドプール。ワークマネージャーごとに短時間実行されるスレッドプールが 1 つあります。
long-running-threads	**LONG_RUNNING** ヒントを設定する Jakarta Connectors 1.7 Work インスタンスのスレッドプール。各ワークマネージャーはオプションの長期スレッドプールを 1 つ持てます。

ワークマネージャーのスレッドプールに設定できる属性は下表のとおりです。

表16.5 スレッドプールの属性

属性	説明
allow-core-timeout	コアスレッドがタイムアウトするかどうかを決定するブール値の設定。デフォルト値は **false** です。
core-threads	コアスレッドプールのサイズ。スレッドプールの最大サイズ以下である必要があります。
handoff-executor	タスクが受け入れられない場合、タスクを委譲するエグゼキューター。指定されていない場合、受け入れられないタスクは警告なしに破棄されます。
keepalive-time	ワーク実行後にスレッドプールが保持される期間を指定します。
max-threads	スレッドプールの最大サイズ。
name	スレッドプールの名前を指定します。
queue-length	キューの最大長。
thread-factory	スレッドファクトリーへの参照。

分散ワークマネージャー

分散ワークマネージャーは、別のワークマネージャーインスタンスでワークの実行スケジュールを変更できるワークマネージャーです。

以下の管理 CLI コマンドの例は、分散ワークマネージャーを設定します。スタンドアロンサーバーの **standalone-ha.xml** または **standalone-full-ha.xml** 設定ファイルなど、高可用性を提供する設定を使用する必要があることに注意してください。

例: 分散ワークマネージャーの設定

```
batch
/subsystem=jca/distributed-workmanager=myDistWorkMgr:add(name=myDistWorkMgr)
/subsystem=jca/distributed-workmanager=myDistWorkMgr/short-running-
threads=myDistWorkMgr:add(queue-length=10,max-threads=10)
/subsystem=jca/bootstrap-
context=myCustomContext:add(name=myCustomContext,workmanager=myDistWorkMgr)
run-batch
```

注記

short-running-threads 要素の名前は **distributed-workmanager** 要素の名前と同じである必要があります。

以下の表は、分散ワークマネージャーに設定できる属性を表しています。

表16.6 分散ワークマネージャーの属性

属性	説明
elytron-enabled	ワークマネージャーの Elytron セキュリティーを有効にします。
name	分散ワークマネージャーの名前。
policy	ワークインスタンスをいつ再分散するかを決定します。使用できる値は次のとおりです。 • **NEVER** - ワークインスタンスを別のノードに分散しません。 • **ALWAYS** - 常にワークインスタンスを別のノードに分散します。 • **WATERMARK** - 現在のノードで使用できる空きのワーカースレッドの数に応じてワーカーインスタンスを別のノードに分散します。
policy-options	ポリシーのキーバリューペアのオプションリスト。**WATERMARK** ポリシーを使用する場合、**watermark** ポリシーオプションを使用して、ワークが分散される空きスレッドの数を指定します。以下に例を示します。 `/subsystem=jca/distributed-workmanager=myDistWorkMgr:write-attribute(name=policy-options,value={watermark=3})`

属性	説明
selector	ワークインスタンスを再分散するネットワークのノードを決定します。使用できる値は次のとおりです。 • **FIRST_AVAILABLE** - リストの最初に利用できるノードを選択します。 • **PING_TIME** - ping の時間が最も短いノードを選択します。 • **MAX_FREE_THREADS** - 空きワーカースレッドの数が最も多いノードを選択します。
selector-options	セレクターのキーバリューペアのオプションリスト。

分散ワークマネージャーには以下の子要素もあります。

表16.7 分散ワークマネージャーの子要素

子要素	説明
long-running-threads	**LONG_RUNNING** ヒントを設定するワークインスタンスのスレッドプール。各分散ワークマネージャーはオプションで長時間実行スレッドプールを持つことができます。
short-running-threads	標準ワークインスタンスのスレッドプール。各分散ワークマネージャーには短期間実行スレッドプールがある必要があります。

ブートストラップコンテキスト

カスタムのブートストラップコンテキストを定義するために使用されます。以下の表は、ブートストラップコンテキストに設定できる属性を表しています。

表16.8 ブートストラップコンテキストの属性

属性	説明
name	ブートストラップコンテキストの名前を指定します。
workmanager	このコンテキストに使用するワークマネージャーの名前を指定します。

キャッシュ接続マネージャー

トランザクションの接続における接続のデバッグおよび Lazy Enlistment のサポートに使用され、アプリケーションによって使用およびリリースされるかどうかを追跡します。以下の表はキャッシュ接続マネージャーに設定できる属性を表しています。

表16.9 キャッシュ接続マネージャーの属性

属性	デフォルト値	説明

属性	デフォルト値	説明
debug	false	接続を明示的に閉じるため、障害時に警告を出力します。
error	false	接続を明示的に閉じるため、障害時に例外が発生します。
ignore-unknown-connections	false	未知の接続はキャッシュされないことを指定します。
install	false	キャッシュされた接続マネージャーバルブおよびインターセプターを有効または無効にします。

16.4. リソースアダプターの設定

16.4.1. リソースアダプターのデプロイ

リソースアダプターは管理 CLI または管理コンソールを使用して他のデプロイメントと同様にデプロイできます。また、スタンドアロンサーバー実行時にアーカイブをデプロイメントディレクトリーにコピーして、デプロイメントスキャナーによって検出されるようにすることもできます。

管理 **CLI** を使用したリソースアダプターのデプロイ
リソースアダプターをスタンドアロンサーバーへデプロイするには、以下の管理 CLI コマンドを実行します。

```
deploy /path/to/resource-adapter.rar
```

リソースアダプターをマネージドドメインのすべてのサーバーグループにデプロイするには、以下の管理 CLI コマンド を入力します。

```
deploy /path/to/resource-adapter.rar --all-server-groups
```

管理コンソールを使用したリソースアダプターのデプロイ

1. 管理コンソールにログインし、**Deployments** タブをクリックします。
2. 追加 (+) ボタンをクリックします。マネージドドメインでは最初に **Content Repository** を選択する必要があります。
3. デプロイメントのアップロード オプションを選択します。
4. リソースアダプターアーカイブを閲覧し、次へ をクリックします。
5. アップロードを確認してから 完了をクリックします。
6. マネージドドメインでは、デプロイメントを該当するサーバーグループにデプロイし、デプロイメントを有効にします。

デプロイメントスキャナーを使用したリソースアダプターのデプロイ
リソースアダプターを手作業でスタンドアロンサーバーにデプロイするには、リソースアダプターアーカイブをサーバーデプロイメントディレクトリー (例: **EAP_HOME/standalone/deployments/**) にコピーします。これにより、デプロイメントスキャナーによって検出され、デプロイされます。

注記

このオプションはマネージドドメインでは使用できません。管理コンソールまたは管理 CLI を使用してリソースアダプターをサーバーグループにデプロイする必要があります。

16.4.2. リソースアダプターの設定

管理インターフェイスを使用してリソースアダプターを設定できます。以下の例は、管理 CLI を使用してリソースアダプターを設定する方法を示しています。サポートされるプロパティーやその他の重要な情報は、リソースアダプターベンダーのマニュアルを参照してください。

リソースアダプター設定の追加
リソースアダプター設定を追加します。

```
/subsystem=resource-adapters/resource-adapter=eis.rar:add(archive=eis.rar, transaction-support=XATransaction)
```

リソースアダプターの設定
必要に応じて以下を設定します。

- **config-properties** を設定します。
  server 設定プロパティーを追加します。

  ```
  /subsystem=resource-adapters/resource-adapter=eis.rar/config-properties=server:add(value=localhost)
  ```

  port 設定プロパティーを追加します。

  ```
  /subsystem=resource-adapters/resource-adapter=eis.rar/config-properties=port:add(value=9000)
  ```

- **admin-objects** を設定します。
  管理オブジェクトを追加します。

  ```
  /subsystem=resource-adapters/resource-adapter=eis.rar/admin-objects=aoName:add(class-name=com.acme.eis.ra.EISAdminObjectImpl, jndi-name=java:/eis/AcmeAdminObject)
  ```

  管理オブジェクト設定プロパティーを設定します。

  ```
  /subsystem=resource-adapters/resource-adapter=eis.rar/admin-objects=aoName/config-properties=threshold:add(value=10)
  ```

- **connection-definitions** を設定します。
  管理接続ファクトリーの接続定義を追加します。

  ```
  /subsystem=resource-adapters/resource-adapter=eis.rar/connection-definitions=cfName:add(class-name=com.acme.eis.ra.EISManagedConnectionFactory, jndi-name=java:/eis/AcmeConnectionFactory)
  ```

  管理接続ファクトリーの設定プロパティーを設定します。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/connection-definitions=cfName/config-properties=name:add(value=Acme Inc)
```

エンリストメントトレースを記録するかどうかを設定します。エンリストメントトレースの記録を有効にするには、**enlistment-trace** 属性を **true** に設定します。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/connection-definitions=cfName:write-attribute(name=enlistment-trace,value=true)
```

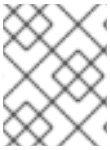
警告

エンリストメントトレースを有効にすると、トランザクションエンリストメント中のエラーを容易に追跡できますが、パフォーマンスに影響します。

リソースアダプターの利用可能なすべてのオプションについては、リソースアダプターの属性 を参照してください。

リソースアダプターのアクティベート
リソースアダプターをアクティベートします。

```
/subsystem=resource-adapters/resource-adapter=eis.rar:activate
```



注記

リソースアダプターのキャパシティーポリシーを定義することも可能です。詳細は キャパシティーポリシー の項を参照してください。

16.4.3. Elytron サブシステムを使用するようリソースアダプターを設定

IronJacamar では、サーバーとリソースアダプターの間で 2 種類の通信が発生します。

その通信の 1 つは、サーバーがリソースアダプター接続を開いた場合に発生します。仕様によって定義されたとおり、コンテナー管理のサインオンでセキュアにすることができます。これには、接続を開いたときにプリンシパルと認証情報で JAAS サブジェクトをリソースアダプターへ伝搬する必要があります。このサインオンは、Elytron に託せます。

IronJacamar はセキュリティーインフローをサポートします。このメカニズムは、ワークをワークマネージャーに提出するときや、同じ JBoss EAP インスタンスにあるエンドポイントにメッセージを配信するときに、リソースアダプターがセキュリティー情報を確立できるようにします。

コンテナー管理のサインオン
Elytron でコンテナー管理のサインオンを実現するには、**elytron-enabled** 属性を **true** に設定する必要があります。これにより、リソースアダプターへのすべての接続が Elytron によってセキュア化されます。

```
/subsystem=resource-adapters/resource-adapter=RAR_NAME/connection-definitions=FACTORY_NAME:write-attribute(name=elytron-enabled,value=true)
```

resource-adapters サブシステムの **elytron-enabled** 属性を **true** に設定すると、管理 CLI を使用して **elytron-enabled** 属性を設定できます。デフォルトではこの属性は **false** に設定されています。

authentication-context 属性は、サインオンの実行に使用される Elytron 認証コンテキストの名前を定義します。

Elytron の **authentication-context** 属性には 1 つ以上の **authentication-configuration** 要素を含めることができ、使用を希望する認証情報が含まれます。

authentication-context 属性が設定されていない場合、JBoss EAP は現在の **authentication-context** を使用します。 これは、接続を開く呼び出し元コードによって使用される **authentication-context** です。

例: authentication-configuration の作成

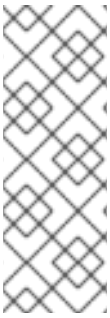
```
/subsystem=elytron/authentication-configuration=exampleAuthConfig:add(authentication-name=sa,credential-reference={clear-text=sa})
```

例: 上記設定を使用した authentication-context の作成

```
/subsystem=elytron/authentication-context=exampleAuthContext:add(match-rules=[{authentication-configuration=exampleAuthConfig}])
```

セキュリティーインフロー
また、ワークマネージャーによって実行される予定のワークを提出するときに、リソースマネージャーはセキュリティー認証情報を流入 (インフロー) することもできます。セキュリティーインフローは、実行前にワークがワーク自体を認証できるようにします。認証に成功すると、提出されたワークは結果となる認証コンテキスト下で実行されます。認証に失敗すると、ワークの実行は拒否されます。

Elytron セキュリティーインフローを有効にするには、リソースアダプターのワークマネージャーを設定するときに **wm-elytron-security-domain** 属性を設定します。Elytron は指定のドメインを元に認証を実行します。



注記

リソースアダプターのワークマネージャーが Elytron セキュリティードメイン **wm-elytron-security-domain** を使用するよう設定された場合、参照されたワークマネージャーの **elytron-enabled** 属性を **true** に設定する必要があります。

```
/subsystem=jca/workmanager=customWM:add(name=customWM, elytron-enabled=true)
```

注記

wm-elytron-security-domain の代わりに **wm-security-domain** 属性が使用された場合、セキュリティーインフローはレガシーの **security** サブシステムによって実行されます。

以下の **jca** サブシステムの設定例では、**ra-with-elytron-security-domain** というリソースアダプターの設定を確認できます。このリソースアダプターは、Elytron セキュリティードメインの **wm-realm** を使用するようワークマネージャーセキュリティーを設定します。

```
<subsystem xmlns="urn:jboss:domain:jca:5.0">
```

```
    <archive-validation enabled="true" fail-on-error="true" fail-on-warn="false"/>
    <bean-validation enabled="true"/>
    <default-workmanager>
        <short-running-threads>
            <core-threads count="50"/>
            <queue-length count="50"/>
            <max-threads count="50"/>
            <keepalive-time time="10" unit="seconds"/>
        </short-running-threads>
        <long-running-threads>
            <core-threads count="50"/>
            <queue-length count="50"/>
            <max-threads count="50"/>
            <keepalive-time time="10" unit="seconds"/>
        </long-running-threads>
    </default-workmanager>
    <workmanager name="customWM">
        <elytron-enabled>true</elytron-enabled>
        <short-running-threads>
            <core-threads count="20"/>
            <queue-length count="20"/>
            <max-threads count="20"/>
        </short-running-threads>
    </workmanager>
    <bootstrap-contexts>
        <bootstrap-context name="customContext" workmanager="customWM"/>
    </bootstrap-contexts>
    <cached-connection-manager/>
</subsystem>
```

その後、ワークマネージャーは **resource-adapter** サブシステムからブートストラップコンテキストを使用して参照されます。

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:5.0">
    <resource-adapters>
        <resource-adapter id="ra-with-elytron-security-domain">
            <archive>
                ra-with-elytron-security-domain.rar
            </archive>
            <bootstrap-context>customContext</bootstrap-context>
            <transaction-support>NoTransaction</transaction-support>
            <workmanager>
                <security>
                    <elytron-security-domain>wm-realm</elytron-security-domain>
                    <default-principal>wm-default-principal</default-principal>
                    <default-groups>
                        <group>
                            wm-default-group
                        </group>
                    </default-groups>
                </security>
            </workmanager>
        </resource-adapter>
    </resource-adapters>
</subsystem>
```

例: セキュリティードメインの設定

```
/subsystem=elytron/properties-realm=wm-properties-realm:add(users-properties={path=/security-dir/users.properties, plain-text=true}, groups-properties={path=/security-dir/groups.properties})

/subsystem=elytron/simple-role-decoder=wm-role-decoder:add(attribute=groups)

/subsystem=elytron/constant-permission-mapper=wm-permission-mapper:add(permissions=[{class-name="org.wildfly.security.auth.permission.LoginPermission"}])

/subsystem=elytron/security-domain=wm-realm:add(default-realm=wm-properties-realm, permission-mapper=wm-permission-mapper, realms=[{role-decoder=wm-role-decoder, realm=wm-properties-realm}])
```

Work クラスは、指定のドメイン下で Elytron の認証の認証情報を提供するロールがあります。これには **javax.resource.spi.work.WorkContextProvider** を実装する必要があります。

```
public interface WorkContextProvider {
  /**
   * Gets an instance of <code>WorkContexts</code> that needs to be used
   * by the <code>WorkManager</code> to set up the execution context while
   * executing a <code>Work</code> instance.
   *
   * @return an <code>List</code> of <code>WorkContext</code> instances.
   */
  List<WorkContext> getWorkContexts();
}
```

このインターフェイスは、**Work** クラスが **WorkContext** を使用して、ワークが実行されるコンテキストの一部の機能を設定できるようにします。その機能の 1 つがセキュリティーインフローです。これについては、**List<WorkContext> getWorkContexts** メソッドは **javax.resource.spi.work.SecurityContext** を提供する必要があります。このコンテキストは、Jakarta Authentication によって定義される **javax.security.auth.callback.Callback** オブジェクトを使用します。この仕様の詳細は、Jakarta Authentication 仕様 を参照してください。

例: コンテキストを使用したコールバックの作成

```
public class ExampleWork implements Work, WorkContextProvider {

    private final String username;
    private final String role;

    public MyWork(TestBean bean, String username, String role) {
        this.principals = null;
        this.roles = null;
        this.bean = bean;
        this.username = username;
        this.role = role;
    }

    public List<WorkContext> getWorkContexts() {
        List<WorkContext> l = new ArrayList<>(1);
        l.add(new MySecurityContext(username, role));
        return l;
    }
```

```
    public void run() {
        ...
    }

    public void release() {
        ...
    }

    public class ExampleSecurityContext extends SecurityContext {

        public void setupSecurityContext(CallbackHandler handler, Subject executionSubject, Subject
serviceSubject) {
            try {
                List<javax.security.auth.callback.Callback> cbs = new ArrayList<>();
                cbs.add(new CallerPrincipalCallback(executionSubject, new SimplePrincipal(username)));
                cbs.add(new GroupPrincipalCallback(executionSubject, new String[]{role}));
                handler.handle(cbs.toArray(new javax.security.auth.callback.Callback[cbs.size()]));
            } catch (Throwable t) {
                throw new RuntimeException(t);
            }
        }
    }
```

上記の例では、**ExampleWork** は **WorkContextProvider** インターフェイスを実装し、**ExampleSecurityContext** を提供します。そのコンテキストは、ワークの実行時に Elytron によって認証されるセキュリティー情報の提供に必要なコールバックを作成します。

16.4.4. IBM MQ リソースアダプターのデプロイおよび設定

JBoss EAP の**Configuring Messaging**には、IBM MQ リソースアダプターのデプロイメント の手順が記載されています。

16.4.5. 汎用 Jakarta Messaging リソースアダプターのデプロイおよび設定

JBoss EAP の Configuring Messaging には、汎用 **Jakarta Messaging** リソースアダプターの設定の手順が記載されています。

16.5. 管理接続プールの設定

JBoss EAP は **ManagedConnectionPool** インターフェイスの 3 つの実装を提供します。

org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreConcurrentLinkedQueueManagedConnectionPool

これは、JBoss EAP 7 のデフォルトの接続プールで、追加設定がない場合のパフォーマンスが最も優れています。

org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreArrayListManagedConnectionPool

過去の JBoss EAP バージョンではデフォルトの接続プールでした。

org.jboss.jca.core.connectionmanager.pool.mcp.LeakDumperManagedConnectionPool

この接続プールはデバッグの目的でのみ使用され、シャットダウンやプールのフラッシュ時にリークが報告されます。

以下の管理 CLI コマンドを使用して、データソースの管理接続プール実装を設定できます。

```
/subsystem=datasources/data-source=DATA_SOURCE:write-
attribute(name=mcp,value=MCP_CLASS)
```

以下の管理 CLI コマンドを使用して、リソースアダプターの管理接続プール実装を設定できます。

```
/subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-
definitions=CONNECTION_DEFINITION:write-attribute(name=mcp,value=MCP_CLASS)
```

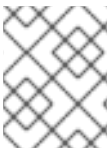
以下の管理 CLI コマンドを使用して、メッセージングサーバーの管理接続プール実装を設定できます。

```
/subsystem=messaging-activemq/server=SERVER/pooled-connection-
factory=CONNECTION_FACTORY:write-attribute(name=managed-connection-
pool,value=MCP_CLASS)
```

16.6. 接続統計の表示

/deployment=NAME.rar サブツリーから定義された接続の統計を読み取りできます。これは、**standalone.xml** または **domain.xml** 設定に定義されていなくても、すべての RAR の統計にアクセスできるようにするためです。

```
/deployment=NAME.rar/subsystem=resource-adapters/statistics=statistics/connection-
definitions=java\:\/testMe:read-resource(include-runtime=true)
```



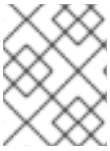
注記

統計はすべてランタイムのみの情報であるため、必ず **include-runtime=true** 引数を指定してください。

利用できる統計については、リソースアダプターの統計 を参照してください。

16.7. リソースアダプター接続のフラッシュ

以下の管理 CLI コマンドを使用して、リソースアダプターの接続をフラッシュします。



注記

マネージドドメインでは、コマンドの前に **/host=HOST_NAME/server=SERVER_NAME** を追加する必要があります。

- プールの接続をすべてフラッシュします。

  ```
  /subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-
  definitions=CONNECTION_DEFINITION:flush-all-connection-in-pool
  ```

- プールの接続をすべて正常にフラッシュします。

  ```
  /subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-
  definitions=CONNECTION_DEFINITION:flush-gracefully-connection-in-pool
  ```

  サーバーは、接続がアイドル状態なるまで待ってから接続をフラッシュします。

- プールのアイドル状態の接続をすべてフラッシュします。

  ```
  /subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-definitions=CONNECTION_DEFINITION:flush-idle-connection-in-pool
  ```

- プールの無効な接続をすべてフラッシュします。

  ```
  /subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-definitions=CONNECTION_DEFINITION:flush-invalid-connection-in-pool
  ```

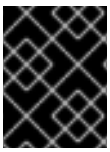
  サーバーは無効であると判断したすべての接続をフラッシュします。

16.8. RESOURCE ADAPTERS サブシステムの調整

resource-adapters サブシステムのパフォーマンスを監視および最適化するための情報は、**Performance Tuning Guide**の Datasource and Resource Adapter Tuning の項を参照してください。

第17章 WEB サーバーの設定 (UNDERTOW)

17.1. UNDERTOW サブシステムの概要



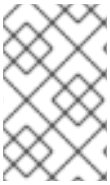
重要

JBoss EAP 7 では、JBoss EAP 6 の **web** サブシステムの代わりに **undertow** サブシステムが使用されます。

undertow サブシステムは、Web サーバーおよびサーブレットコンテナーの設定を可能にします。Jakarta Servlet 4.0 Specification や WebSocket を実装します。HTTP のアップグレードや、サーブレットデプロイメントでの高パフォーマンスな非ブロッキングハンドラーの使用もサポートします。**undertow** サブシステムは、mod_cluster をサポートする高パフォーマンスなリバースプロキシーとして動作することも可能です。

undertow サブシステム内で設定する主なコンポーネントは 5 つあります。

- バッファーキャッシュ
- server
- サーブレットコンテナー
- handlers
- フィルター



注記

JBoss EAP には、これらの各コンポーネントの設定を更新する機能がありますが、デフォルト設定は妥当なパフォーマンスの設定を提供し、ほとんどのユースケースに適しています。

デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <http-invoker security-realm="ApplicationRealm"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
```

```
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
</subsystem>
```

重要

また、**undertow** サブシステムは **io** サブシステムに依存して XNIO ワーカーやバッファープールを提供します。**io** サブシステムは個別に設定され、ほとんどの場合で最適なパフォーマンスを得られるデフォルト設定を提供します。

注記

JBoss EAP 6 の **web** サブシステムと比較すると、JBoss EAP 7 の **undertow** サブシステムでは HTTP メソッドのデフォルト動作 が異なります。

Undertow サブシステムでの Elytron の使用

web アプリケーションがデプロイされると、そのアプリケーションが必要なセキュリティードメインの名前が特定されます。これは、デプロイメント内からで、デプロイメントにセキュリティードメインがない場合は **undertow** サブシステムで定義された **default-security-domain** が仮定されます。デフォルトでは、セキュリティードメインがレガシー security サブシステムで定義された **PicketBox** にマップすることが仮定されます。しかし、アプリケーションが必要なセキュリティードメインの名前から適切な Elytron 設定にマッピングする **undertow** サブシステムに **application-security-domain** リソースを追加することができます。

例: マッピングの追加

```
/subsystem=undertow/application-security-domain=ApplicationDomain:add(security-
domain=ApplicationDomain)
```

マッピングの追加に成功すると、以下のような結果が出力されます。

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0" ... default-security-domain="other">
...
    <application-security-domains>
        <application-security-domain name="ApplicationDomain" security-domain="ApplicationDomain"/>
    </application-security-domains>
...
</subsystem>
```

注記

- この時点でデプロイメントがすでにデプロイされている場合は、アプリケーションサーバーをリロードして、アプリケーションセキュリティードメインマッピングを有効にする必要があります。
- 現在の Web サービス/Elytron 統合では、Web サービスエンドポイントと Elytron セキュリティードメイン名をセキュアにするために指定されたセキュリティードメインの名前が同じである必要があります。

この簡単な例は、**BASIC**、**CLIENT_CERT**、**DIGEST**、**FORM** など、デプロイメントがサーブレット仕様内で定義された標準の HTTP メカニズムを使用する場合に適しています。この場合、認証は **ApplicationDomain** セキュリティードメインに対して実行されます。このフォームは、アプリケー

ションが認証メカニズムを使用せず、代わりにプログラムによる認証を使用したり、デプロイメントに関連する **SecurityDomain** を取得して直接使用したりする場合にも適しています。

例: 高度なマッピング

```
/subsystem=undertow/application-security-domain=MyAppSecurity:add(http-authentication-factory=application-http-authentication)
```

高度なマッピングに成功すると、以下のような結果が出力されます。

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0" ... default-security-domain="other">
...
    <application-security-domains>
        <application-security-domain name="MyAppSecurity" http-authentication-factory="application-http-authentication"/>
    </application-security-domains>
...
</subsystem>
```

このような設定では、セキュリティードメインを参照する代わりに、**http-authentication-factory** が参照されます。これは、認証メカニズムのインスタンスの取得に使用されるファクトリーで、セキュリティードメインと関連付けられます。

カスタム HTTP 認証メカニズムを使用する場合や、プリンシパルトランスフォーマー、認証情報ファクトリー、メカニズムレルムなどのメカニズムに追加の設定を定義する必要がある場合など、**http-authentication-factory** 属性を参照する必要があります。また、**Servlet** 仕様に記載されている 4 つのメカニズム以外を使用する場合は、**http-authentication-factory** 属性を参照した方がよいでしょう。

高度なマッピングが使用される場合、別の設定オプション **override-deployment-config** を使用できます。参照された **http-authentication-factory** は認証メカニズムの完全セットを返すことができます。デフォルトでは、アプリケーションによってリクエストされたメカニズムと一致するするためにフィルターされます。このオプションが **true** に設定された場合、ファクトリーによって提供されたメカニズムは、アプリケーションによってリクエストされたメカニズムをオーバーライドします。

application-security-domain リソースには追加のオプション **enable-jacc** もあります。これを **true** に設定すると、このマッピングに一致するすべてのデプロイメントに対して Jakarta Authorization が有効になります。

ランタイム情報

application-security-domain マッピングが使用されている場合、このマッピングに対してデプロイメントが想定どおり一致していることを再度確認すると便利です。リソースが **include-runtime=true** で読み取りされた場合、マッピングに関連するデプロイメントは以下のように表示されます。

```
/subsystem=undertow/application-security-domain=MyAppSecurity:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "enable-jacc" => false,
        "http-authentication-factory" => undefined,
        "override-deployment-config" => false,
        "referencing-deployments" => ["simple-webapp.war"],
        "security-domain" => "ApplicationDomain",
```

```
        "setting" => undefined
    }
}
```

この出力の **referencing-deployments** 属性は、**simple-webapp.war** デプロイメントがマッピングを使用してデプロイされたことを示しています。

17.2. バッファーキャッシュの設定

バッファーキャッシュは静的リソースをキャッシュするために使用されます。JBoss EAP では複数のキャッシュを設定でき、デプロイメントによる参照が可能であるため、デプロイメントごとに異なるキャッシュサイズを使用することができます。バッファーは固定サイズで、リージョンで割り当てられます。使用領域の合計は、バッファーサイズ、リージョンごとのバッファー数、およびリージョンの最大数を掛けて算出できます。バッファーキャッシュのデフォルトサイズは 10MB です。

JBoss EAP はデフォルトで単一のキャッシュを提供します。

デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    ...
</subsystem>
```

既存のバッファーキャッシュの更新

既存のバッファーキャッシュを更新するには、以下を指定します。

```
/subsystem=undertow/buffer-cache=default/:write-attribute(name=buffer-size,value=2048)
```

```
reload
```

新規バッファーキャッシュの作成

新しいバッファーキャッシュを作成するには、以下を指定します。

```
/subsystem=undertow/buffer-cache=new-buffer:add
```

バッファーキャッシュの削除

バッファーキャッシュを削除するには、以下を指定します。

```
/subsystem=undertow/buffer-cache=new-buffer:remove
```

```
reload
```

バッファーキャッシュの設定に使用できる属性の完全リストは、Undertow サブシステムの属性 の項を参照してください。

17.3. バイトバッファープールの設定

Undertow バイトバッファープールは、プールされた NIO **ByteBuffer** インスタンスの割り当てに使用されます。すべてのリスナーにバイトバッファープールがあり、各リスナーに異なるバッファープールおよびワーカーを使用できます。バイトバッファープールは異なるサーバーインスタンス間で共有できま

す。

これらのバッファーは IO 操作に使用され、バッファーサイズはアプリケーションのパフォーマンスに大きく影響します。通常、ほとんどのサーバーでは 16 K が理想のサイズになります。

既存のバイトバッファープールの更新
既存のバイトバッファープールを更新するには、以下を指定します。

```
/subsystem=undertow/byte-buffer-pool=myByteBufferPool:write-attribute(name=buffer-size,value=1024)
```

```
reload
```

バイトバッファープールの削除
新しいバイトバッファープールを作成するには、以下を指定します。

```
/subsystem=undertow/byte-buffer-pool=newByteBufferPool:add
```

バイトバッファープールの削除
バイトバッファープールを削除するには、以下を指定します。

```
/subsystem=undertow/byte-buffer-pool=newByteBufferPool:remove
```

```
reload
```

バイトバッファープールの設定に使用できる属性の完全リストは バイトバッファープールの属性 を参照してください。

17.4. サーバーの設定

サーバーは Undertow のインスタンスを表し、複数の要素で設定されます。

- host
- http-listener
- https-listener
- ajp-listener

ホスト要素は仮想ホスト設定を提供し、3 つのリスナーはそのタイプの接続を Undertow インスタンスに提供します。

サーバーのデフォルトの動作では、サーバーの起動中にリクエストをキューに置きます。デフォルトの動作を変更するには、ホストで **queue-requests-on-start** 属性を使用します。この属性がデフォルトの **true** に設定されている場合、サーバーの起動時に到達するリクエストはサーバーの準備が整うまで保留されます。この属性が **false** に設定された場合、サーバーが完全に起動する前に到達したリクエストは、デフォルトの応答コードで拒否されます。

属性の値に関わらず、サーバーが完全に起動するまでリクエストの処理は開始されません。

管理コンソールを使用して **queue-requests-on-start** 属性を設定するには、**Configuration → Subsystems → Web (Undertow) → Server** と選択した後、サーバーを選択して 表示 をクリックし、**Hosts** タブを選択します。マネージドドメイでは、設定するプロファイルを指定する必要がありま

す。

注記

複数のサーバーを設定できるため、デプロイメントやサーバーを完全に分離できます。これは、マルチテナント環境などで便利です。

JBoss EAP はデフォルトでサーバーを提供します。

デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <http-invoker security-realm="ApplicationRealm"/>
        </host>
    </server>
    ...
</subsystem>
```

以下の例は、管理 CLI を使用してサーバーを設定する方法を示しています。管理コンソールを使用してサーバーを設定する場合は、**Configuration → Subsystems → Web (Undertow) → Server** と選択します。

既存のサーバーの更新

既存のサーバーを更新するには、以下を設定します。

```
/subsystem=undertow/server=default-server:write-attribute(name=default-host,value=default-host)
```

```
reload
```

新規サーバーの作成

新規サーバーを作成するには、以下を指定します。

```
/subsystem=undertow/server=new-server:add
```

```
reload
```

サーバーの削除

サーバーを削除するには、以下を指定します。

```
/subsystem=undertow/server=new-server:remove
```

```
reload
```

サーバーの設定に使用できる属性の完全リストは、Undertow サブシステムの属性 の項を参照してください。

17.4.1. アクセスロギング

定義する各ホストにアクセスロギングを設定できます。

利用できるアクセスロギングオプションは、標準のアクセスロギングとコンソールアクセスロギングの 2 つです。

アクセスロギングに必要な追加の処理はシステムパフォーマンスに影響を与える可能性があることに注意してください。

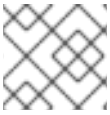
17.4.1.1. 標準のアクセスロギング

標準のアクセスログは、ログエントリーをログファイルに書き込みます。

デフォルトでは、ログファイルは standalone/log/access_log.log ディレクトリーに保存されます。

標準のアクセスログを有効にするには、アクセスログデータを取得するホストに access-log 設定を追加します。以下の CLI コマンドは、デフォルトの JBoss EAP サーバーのデフォルトのホストの設定を示しています。

```
/subsystem=undertow/server=default-server/host=default-host/setting=access-log:add
```



注記

標準のアクセスログを有効にしたら、サーバーをリロードする必要があります。

デフォルトでは、アクセスログレコードには以下のデータが含まれます。

- リモートホスト名
- リモート論理ユーザー名 (常に -)
- 認証されたリモートユーザー
- Common Log Format 形式のリクエストの日時
- 要求の最初の行
- 応答の HTTP ステータスコード
- HTTP ヘッダーを除く、送信済みバイト数。

この一連のデータは共通のパターンとして定義されます。組み合わせた別のパターンも使用できます。一般的なパターンでログ記録されているデータのほかに、組み合わせたパターンには、受信ヘッダーからの閲覧元およびユーザーエージェントが含まれます。

ログに記録されるデータは、**pattern** 属性を使用して変更できます。以下の CLI コマンドは、組み合わせたパターンを使用ための **pattern** 属性の更新を示しています。

```
/subsystem=undertow/server=default-server/host=default-host/setting=access-log:write-
attribute(name=pattern,value="combined"
```

注記

pattern 属性を更新した後は、サーバーをリロードする必要があります。

表17.1 利用可能なパターン

パターン	説明
%a	リモート IP アドレス
%A	ローカル IP アドレス
%b	送信済みバイト数 (HTTP ヘッダーまたは **-** を除く (バイトが送信されなかった場合))
%B	送信済みバイト数 (HTTP ヘッダーを除く)
%h	リモートホスト名
%H	要求プロトコル
%l	**identd** からのリモート論理ユーザー名 (常に **-** を返します。これは、Apache アクセスログの互換性のために含まれています)
%m	要求メソッド
%p	ローカルポート
%q	クエリー文字列 (**?** 文字を除く)
%r	要求の最初の行
%s	応答の HTTP ステータスコード
%t	Common Log Format 形式の日時
%u	認証されたリモートユーザー
%U	要求された URL パス
%v	ローカルサーバー名
%D	要求を処理するのにかかった時間 (ミリ秒単位)
%T	要求を処理するのにかかった時間 (秒単位)

パターン	説明
%l	現在の要求スレッド名 (後でスタックトレースと比較できます)
common	**%h %l %u %t "%r" %s %b**
combined	**%h %l %u %t "%r" %s %b "%{i,Referer}" "%{i,User-Agent}"**

cookie、受信ヘッダーおよび応答ヘッダー、またはセッションから情報を書き込むこともできます。これは、Apache 構文に基づきます。

- **%{i,xxx}** (受信ヘッダーの場合)
- **%{o,xxx}** (送信応答ヘッダーの場合)
- **%{c,xxx}** (特定のクッキーの場合)
- **%{r,xxx}** (**xxx** は **ServletRequest** の属性)
- **%{s,xxx}** (**xxx** は **HttpSession** の属性)

このログには、追加の設定オプションも利用できます。詳細は、付録の access-log 属性を参照してください。

17.4.1.2. コンソールアクセスロギング

コンソールのアクセスログは、JSON データとして構造化された標準出力 (stdout) にデータを書き込みます。

各アクセスログレコードは、1 行のデータです。ログ集約システムにより、このデータを取得して処理できます。

コンソールアクセスロギングを設定するには、アクセスログデータをキャプチャーしたいホストに console-access-log 設定を追加します。以下の CLI コマンドは、デフォルトの JBoss EAP サーバーのデフォルトのホストの設定を示しています。

```
/subsystem=undertow/server=default-server/host=default-host/setting=console-access-log:add
```

デフォルトでは、コンソールアクセスログレコードには以下のデータが含まれます。

表17.2 デフォルトのコンソールアクセスログデータ

ログデータフィールド名	説明
eventSource	リクエストのイベントのソース
hostName	要求を処理した JBoss EAP ホスト
bytesSent	JBoss EAP サーバーがリクエストへの応答として送信したバイト数

ログデータフィールド名	説明
DateTime	要求が JBoss EAP サーバーによって処理された日時 JBoss EAP サーバーがリクエストを処理した日時
remoteHost	要求の発信先のマシンの IP アドレス
remoteuser	リモート要求に関連付けられたユーザー名
requestLine	送信された要求
responseCode	JBoss EAP サーバーによって返された HTTP 応答コード

デフォルトプロパティーはログ出力に常に含まれます。**attributes** 属性を使用して、デフォルトのログデータのラベルを変更できます。場合によっては、データ設定を変更することも可能です。**attributes** 属性を使用して、さらにログデータを出力に追加することもできます。

表17.3 利用可能なコンソールアクセスログデータ

ログデータフィールド名	説明	形式
authentication-type	要求に関連付けられたユーザーの認証に使用される認証タイプデフォルトラベル: authenticationType。このキーオプションを使用して、このプロパティーのラベルを変更します。	authentication-type{} authentication-type={key="authType"}
bytes-sent	リクエストに対して返されるバイト数。HTTP ヘッダーを除く。デフォルトラベル: bytesSent。このキーオプションを使用して、このプロパティーのラベルを変更します。	bytes-sent={} bytes-sent={key="sent-bytes"}

ログデータフィールド名	説明	形式
date-time	要求が受信および処理された日時。デフォルトラベル: dateTime。このキーオプションを使用して、このプロパティーのラベルを変更します。date-format を使用して、日付レコードのフォーマットに使用するパターンを定義します。このパターンは Java SimpleDateFormatter パターンである必要があります。date-format オプションが定義されている場合は、time-zone オプションを使用して日付や時間データのフォーマットに使用するタイムゾーンを指定します。この値は有効な java.util.TimeZone である必要があります。	date-time={key="<keyname>", date-format="<date-time format>"} date-time={key="@timestamp", date-format="yyyy-MM-dd'T'HH:mm:sSSS"}
host-and-port	リクエストによってクエリーされたホストおよびポート。デフォルトラベル: hostAndPort。このキーオプションを使用して、このプロパティーのラベルを変更します。	host-and-port{} host-and-port={key="port-host"}
local-ip	ローカル接続の IP アドレス。このキーオプションを使用して、このプロパティーのラベルを変更します。デフォルトラベル: localIp。このキーオプションを使用して、このプロパティーのラベルを変更します。	local-ip{} local-ip{key="localIP"}
local-port	ローカル接続のポート。デフォルトラベル: localPort。このキーオプションを使用して、このプロパティーのラベルを変更します。	local-port{} local-port{key="LocalPort"}
local-server-name	要求を処理したローカルサーバー名。デフォルトラベル: localServerName。このキーオプションを使用して、このプロパティーのラベルを変更します。	local-server-name {} local-server-name {key=LocalServerName}

ログデータフィールド名	説明	形式
path-parameter	リクエストに含まれる 1 つ以上のパスまたは URI パラメーター。name プロパティーは、交換値を解決するために使用される名前のコンマ区切りリストです。key-prefix プロパティーを使用してキーを一意にします。key-prefix が指定されている場合は、出力の各 path パラメーター名の前に接頭辞が追加されます。	path-parameter{names={store,section}} path-parameter{names={store,section}, key-prefix="my-"}
predicate	述語コンテキストの名前。name プロパティーは、交換値を解決するために使用される名前のコンマ区切りリストです。key-prefix プロパティーを使用してキーを一意にします。key-prefix が指定されている場合は、出力の各 path パラメーター名の前に接頭辞が追加されます。	predicate{names={store,section}} predicate{names={store,section}, key-prefix="my-"}
query-parameter	リクエストに含まれる 1 つ以上のパラメーターまたはクエリーパラメーター。names プロパティーは、交換値を解決するために使用される名前のコンマ区切りリストです。key-prefix プロパティーを使用してキーを一意にします。key-prefix が指定されている場合は、出力の各 path パラメーター名の前に接頭辞が追加されます。	query-parameter{names={store,section}} query-parameter{names={store,section}, key-prefix="my-"}
query-string	リクエストのクエリー文字列。デフォルトラベル: localPort。このキーオプションを起用して、このプロパティーのラベルを変更します。include-question-mark プロパティーを使用して、クエリー文字列に疑問符を含めるかどうかを指定します。デフォルトでは、疑問符は含まれていません。	query-string{} query-string{key="QueryString", include-question-mark="true"}
relative-path	要求の相対パス。デフォルトラベル: relativePath。このキーオプションを使用して、このプロパティーのラベルを変更します。	relative-path{} relative-path{key="RelativePath"}

ログデータフィールド名	説明	形式
remote-host	リモートホスト名デフォルトラベル: remotehost。このキーオプションを使用して、このプロパティーのラベルを変更します。	remote-host{} remote-host{key="RemoteHost"}
remote-ip	リモート IP アドレス。デフォルトラベル: remoteIp。このキーオプションを使用して、このプロパティーのラベルを変更します。この難読化プロパティーを使用して、出力ログレコードの IP アドレスを難読化します。デフォルト値は false です。	remote-ip{} remote-ip{key="RemoteIP", obfuscated="true"}
remote-user	認証されたリモートユーザーデフォルトラベル: remoteuser。このキーオプションを使用して、このプロパティーのラベルを変更します。	remote-user{} remote-user{key="RemoteUser"}
request-header	要求ヘッダーの名前。構造化されたデータのキーはヘッダーの名前です。その値は名前付きヘッダーの値になります。names プロパティーは、交換値を解決するために使用される名前のコンマ区切りリストです。key-prefix プロパティーを使用してキーを一意にします。key-prefix が指定されている場合には、出力の要求ヘッダー名の前に接頭辞が追加されます。	request-header{names={store,section}} request-header{names={store,section}, key-prefix="my-"}
request-line	リクエストの行。デフォルトラベル: requestLine。このキーオプションを使用して、このプロパティーのラベルを変更します。	request-line{} request-line{key="Request-Line"}
request-method	要求メソッドデフォルトラベル: requestMethod。このキーオプションを使用して、このプロパティーのラベルを変更します。	request-method{} request-method{key="RequestMethod"}
request-path	リクエストの相対パス。デフォルトラベル: requestPath。このキーオプションを使用して、このプロパティーのラベルを変更します。	request-path{} request-path{key="RequestPath"}

ログデータフィールド名	説明	形式
request-protocol	要求のプロトコル。デフォルトラベル: requestProtocol。このキーオプションを使用して、このプロパティーのラベルを変更します。	request-protocol{} request-protocol{key="RequestProtocol"}
request-scheme	要求の URI スキーム。デフォルトラベル: requestScheme。このキーオプションを使用して、このプロパティーのラベルを変更します。	request-scheme{} request-scheme{key="RequestScheme"}
request-url	元の要求 URI。クライアントで指定した場合、ホスト名、プロトコルなどが含まれます。デフォルトラベル: requestUrl。このキーオプションを使用して、このプロパティーのラベルを変更します。	request-url{} request-url{key="RequestURL"}
resolved-path	解決したパス。デフォルトラベル: resolvedPath。このキーオプションを使用して、このプロパティーのラベルを変更します。	resolved-path{} resolved-path{key="ResolvedPath"}
response-code	応答コード。デフォルトラベル: responseCode。このキーオプションを使用して、このプロパティーのラベルを変更します。	response-code{} response-code{key="ResponseCode"}
response-header	応答ヘッダーの名前。構造化されたデータのキーはヘッダーの名前です。その値は名前付きヘッダーの値になります。names プロパティーは、交換値を解決するために使用される名前のコンマ区切りリストです。key-prefix プロパティーを使用してキーを一意にします。key-prefix が指定されている場合には、出力の要求ヘッダー名の前に接頭辞が追加されます。	response-header{names={store,section}} response-header{names={store,section}, key-prefix="my-"}
response-reason-phrase	応答コードのテキスト理由。デフォルトラベル: responseReasonPhrase。このキーオプションを使用してこのプロパティーのラベルを変更します。	response-reason-phrase{} response-reason-phrase{key="ResponseReasonPhrase"}

ログデータフィールド名	説明	形式
response-time	リクエストの処理に使われる時間。デフォルトラベル: responseTime。このキーオプションを使用して、このプロパティーのラベルを変更します。デフォルトの時間単位はミリ秒 (MILLISECONDS) です。利用可能な時間の単位は以下のとおりです。* NANOSECONDS * MICROSECONDS * MILLISECONDS * SECONDS	response-time{} response-time{key="ResponseTime", time-unit=SECONDS}
secure-exchange	交換がセキュアであったかどうかを示します。デフォルトラベル: secureExchange。このキーオプションを使用して、このプロパティーのラベルを変更します。	secure-exchange{} secure-exchange{key="SecureExchange"}
ssl-cipher	要求の SSL 暗号。デフォルトラベル: sslCipher。このオプションを使用して、このプロパティーのラベルを変更します。	ssl-cipher{} ssl-cipher{key="SSLCipher"}
ssl-client-cert	要求の SSL クライアント証明書。デフォルトラベル: sslclientcert。このキーオプションを使用して、このプロパティーのラベルを変更します。	ssl-client-cert{} ssl-client-cert{key="SSLClientCert"}
ssl-session-id	要求の SSL セッション ID。デフォルトラベル: sslSessionId。このキーオプションを使用して、このプロパティーのラベルを変更します。	ssl-session-id{} stored-response
要求に対する、保存した応答。デフォルトラベル: storedResponse。このキーオプションを使用して、このプロパティーのラベルを変更します。	stored-response{} stored-response{key="StoredResponse"}	thread-name
現在のスレッドのスレッド名。デフォルトラベル: threadName。このキーオプションを使用して、このプロパティーのラベルを変更します。	thread-name{} thread-name{key="ThreadName"}	transport-protocol

metadata 属性を使用して、アクセスログレコードに追加する追加の任意のデータを設定できます。**metadata** 属性の値は、アクセスログレコードに追加するデータを定義する key:value ペアのセットです。ペアのこの値は管理モデルの式になります。管理モデル式は、サーバーの起動またはリロード時

に解決されます。キーと値のペアはコンマで区切られます。

以下の CLI コマンドは、追加のログデータ、ログデータのカスタマイズ、追加のメタデータなど、複雑なコンソールログ設定の例を示しています。

```
/subsystem=undertow/server=default-server/host=default-host/setting=console-access-log:add(metadata={"@version"="1", "qualifiedHostName"=${jboss.qualified.host.name:unknown}}, attributes={bytes-sent={}, date-time={key="@timestamp", date-format="yyyy-MM-dd'T'HH:mm:ssSSS"}, remote-host={}, request-line={}, response-header={key-prefix="responseHeader", names=["Content-Type"]}, response-code={}, remote-user={}})
```

結果として生成されるアクセスログレコードは以下の追加の JSON データに類似しています (注意: 以下の出力例は、読みやすさを考慮してフォーマットされています。実際の記録では、すべてのデータが単一の行として出力されます)。

```
{
    "eventSource":"web-access",
    "hostName":"default-host",
    "@version":"1",
    "qualifiedHostName":"localhost.localdomain",
    "bytesSent":1504,
    "@timestamp":"2019-05-02T11:57:37123",
    "remoteHost":"127.0.0.1",
    "remoteUser":null,
    "requestLine":"GET / HTTP/2.0",
    "responseCode":200,
    "responseHeaderContent-Type":"text/html"
}
```

以下のコマンドは、コンソールアクセスログを有効にした後にのログデータへの更新を示しています。

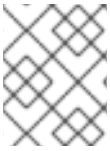
```
/subsystem=undertow/server=default-server/host=default-host/setting=console-access-log:write-attribute(name=attributes,value={bytes-sent={}, date-time={key="@timestamp", date-format="yyyy-MM-dd'T'HH:mm:ssSSS"}, remote-host={}, request-line={}, response-header={key-prefix="responseHeader", names=["Content-Type"]}, response-code={}, remote-user={}})
```

以下のコマンドは、コンソールアクセスログを有効にした後にカスタムメタデータへの更新を示しています。

```
/subsystem=undertow/server=default-server/host=default-host/setting=console-access-log:write-attribute(name=metadata,value={"@version"="1", "qualifiedHostName"=${jboss.qualified.host.name:unknown}})
```

17.5. サーブレットコンテナーの設定

サーブレットコンテナーは、すべてのサーブレット、 Jakarta Server Pages、およびソケット関連の設定 (セッションに関連する設定を含む) を提供します。ほとんどのサーバーにはサーブレットコンテナーが 1 つだけ必要ですが、**servlet-container** 要素を追加すると複数のサーブレットコンテナーを設定することができます。サーブレットコンテナーが複数設定されていると、複数のデプロイメントを異なる仮想ホストの同じコンテキストパスにデプロイできるなど、一部の動作を有効にすることができます。



注記

サーブレットコンテナーによって提供される設定の多くは、デプロイされたアプリケーションが **web.xml** ファイルを使用して個別にオーバーライドできます。

JBoss EAP はデフォルトでサーブレットコンテナーを提供します。

デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0">
  <buffer-cache name="default"/>
  <server name="default-server">
    ...
  </server>
  <servlet-container name="default">
     <jsp-config/>
     <websockets/>
  </servlet-container>
...
</subsystem>
```

以下の例は、管理 CLI を使用してサーブレットコンテナーを設定する方法を示しています。管理コンソールを使用してサーブレットコンテナーを設定する場合は、**Configuration → Subsystems → Web (Undertow) → Servlet Container** と選択します。

既存のサーブレットコンテナーの更新
既存のサーブレットコンテナーを更新するには、以下を指定します。

```
/subsystem=undertow/servlet-container=default:write-attribute(name=ignore-flush,value=true)
```

```
reload
```

新規サーブレットコンテナーの作成
新規のサーブレットコンテナーを作成するには、以下を指定します。

```
/subsystem=undertow/servlet-container=new-servlet-container:add
```

```
reload
```

サーブレットコンテナーの削除
サーブレットコンテナーを削除するには、以下を指定します。

```
/subsystem=undertow/servlet-container=new-servlet-container:remove
```

```
reload
```

サーブレットコンテナーの設定に使用できる属性の完全リストは、Undertow サブシステムの属性 の項を参照してください。

17.6. サーブレット拡張の設定

サーブレット拡張は、サーブレットデプロイメントプロセスへのフックや、サーブレットデプロイメン

トの変更を可能にします。これは、追加の認証メカニズムをデプロイメントに追加する必要がある場合や、ネイティブ Undertow ハンドラーをサーブレットデプロイメントの一部として使用する必要がある場合などに便利です。

カスタムサーブレット拡張を作成するには、**io.undertow.servlet.ServletExtension** インターフェイスを実装した後、実装クラスの名前をデプロイメントの **META-INF/services/io.undertow.servlet.ServletExtension** ファイルに追加する必要があります。さらに、**ServletExtension** 実装のコンパイルされたクラスファイルを含める必要もあります。Undertow がサーブレットをデプロイすると、**deployments** クラスローダーからすべてのサービスをロードし、それらの **handleDeployment** メソッドを呼び出します。

デプロイメントの完全かつ変更可能な記述が含まれる Undertow **DeploymentInfo** 構造は、このメソッドに渡されます。この構造を変更して、デプロイメントの内容を変更することができます。

DeploymentInfo 構造は、組み込み API によって使用される構造と同じあるため、**ServletExtension** は Undertow を組み込みモードで使用したときと同じ柔軟性を持ちます。

17.7. ハンドラーの設定

JBoss EAP では、2 つのタイプのハンドラーを設定できます。

- ファイルハンドラー
- リバースプロキシーハンドラー

ファイルハンドラーは静的ファイルに対応します。各ファイルハンドラーは仮想ホストの場所にアタッチされている必要があります。リバースプロキシーハンドラーによって、JBoss EAP は高パフォーマンスなリバースプロキシーとして機能することができます。

JBoss EAP はデフォルトでファイルハンドラーを提供します。

デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    <server name="default-server">
        ...
    </server>
    <servlet-container name="default">
        ...
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
</subsystem>
```

静的リソースに WebDAV を使用
過去のバージョンの JBoss EAP では、**web** サブシステムで WebDAV を使用して (**WebdavServlet** 経由) 静的リソースをホストし、追加の HTTP メソッドを有効にしてこれらのファイルへのアクセスや操作を実行できました。JBoss EAP 7 では、ファイルハンドラーを経由した静的ファイルの対応メカニズムは **undertow** サブシステムによって提供されますが、**undertow** サブシステムは WebDAV をサポートしません。JBoss EAP 7 で WebDAV を使用する場合は、カスタムの WebDav サーブレットを記述してください。

既存のファイルハンドラーの更新

既存のファイルハンドラーを更新するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/file=welcome-content:write-attribute(name=case-sensitive,value=true)
```

```
reload
```

新規ファイルハンドラーの作成
新規のファイルハンドラーを作成するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/file=new-file-handler:add(path="${jboss.home.dir}/welcome-content")
```

警告

ファイルハンドラーの **path** を直接ディレクトリーではなくファイルに設定した場合、そのファイルハンドラーを参照する **location** 要素の最後にフォワードスラッシュ (/) を付けないでください。最後にフォワードスラッシュが付くと、**404 - Not Found** が返されます。

ファイルハンドラーの削除
ファイルハンドラーを削除するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/file=new-file-handler:remove
```

```
reload
```

ハンドラーの設定に使用できる属性の完全リストは、Undertow サブシステムの属性 の項を参照してください。

17.8. フィルターの設定

フィルターはリクエストの一部の変更を可能にし、述語を使用してフィルターの実行時を制御できます。フィルターの一般的なユースケースには、ヘッダーの設定や GZIP 圧縮などがあります。

注記

フィルターの機能は、JBoss EAP 6 で使用されたグローバルバルブと同等です。

以下のタイプのフィルターを定義できます。

- custom-filter
- error-page
- expression-filter

- gzip
- mod-cluster
- request-limit
- response-header
- rewrite

以下の例は、管理 CLI を使用してフィルターを設定する方法を示しています。管理コンソールを使用してフィルターを設定する場合は、**Configuration → Subsystems → Web (Undertow) → Filters** と選択します。

既存のフィルターの更新
既存のフィルターを更新するには、以下を指定します。

```
/subsystem=undertow/configuration=filter/response-header=myHeader:write-attribute(name=header-value,value="JBoss-EAP")
```

```
reload
```

新規のフィルターの作成
新規のフィルターを作成するには、以下を指定します。

```
/subsystem=undertow/configuration=filter/response-header=new-response-header:add(header-name=new-response-header,header-value="My Value")
```

フィルターの削除
フィルターを削除するには、以下を指定します。

```
/subsystem=undertow/configuration=filter/response-header=new-response-header:remove
```

```
reload
```

フィルターの設定に使用できる属性の完全リストは Undertow サブシステムの属性 の項を参照してください。

17.8.1. buffer-request ハンドラーの設定

クライアントまたはブラウザーからのリクエストは、ヘッダーとボディーの 2 つで設定されます。通常の場合、ヘッダーとボディーの間に遅延がない状態で JBoss EAP に送信されます。しかし、ヘッダーが最初に送信され、その数秒後にボディーが送信されると、完全なリクエストの送信に遅延が発生します。このような場合、JBoss EAP にスレッドが作成され、完全なリクエストの実行を待機していることを示す **waiting** が表示されます。

リクエストのヘッダーおよびボディーの送信による遅延は、**buffer-request** ハンドラーを使用して修正できます。**buffer-request** ハンドラーは、ワーカースレッドに割り当てする前に、非ブロッキング IO スレッドからリクエストの消費を試みます。追加された **buffer-request** ハンドラーがない場合、スレッドは直接ワーカースレッドに割り当てられます。しかし、**buffer-request** ハンドラーが追加されると、ワーカースレッドに割り当てする前に、ハンドラーは IO スレッドを使用してブロックせずにバッファー処理できる量のデータを読み取ろうとします。

以下の管理 CLI コマンドを使用して **buffer-request** ハンドラーを設定できます。

```
/subsystem=undertow/configuration=filter/expression-filter=buf:add(expression="buffer-request(buffers=1)")

/subsystem=undertow/server=default-server/host=default-host/filter-ref=buf:add
```

処理できるバッファーリクエストのサイズには制限があります。この制限は、以下の式のとおり、バッファーサイズとバッファー合計数の組み合わせで決定されます。

Total_size = num_buffers × buffer_size

この式の説明は次のとおりです。

- **Total_size** は、リクエストがワーカースレッドに送信される前にバッファー処理されるデータのサイズになります。
- **num_buffers** はバッファーの数になります。バッファーの数は、ハンドラーの **buffers** パラメーターによって設定されます。
- **buffer_size** は各バッファーのサイズになります。バッファーサイズは **io** サブシステムで設定され、デフォルトではリクエストごとに 16KB になります。

警告

メモリー不足になる可能性があるため、サイズが大きすぎるバッファーリクエストは設定しないでください。

17.8.2. SameSite 属性の設定

SameSite 属性を使用して、Cookie のアクセシビリティー (同じサイト内で Cookie にアクセスできるかどうか) を定義します。この属性により、ブラウザーはクロスサイトリクエストで Cookie を送信しないため、クロスサイトフォージェリー攻撃を阻止できます。

undertow サブシステムの **SameSiteCookieHandler** を使用して、Cookie の **SameSite** 属性を設定できます。この設定では、アプリケーションコードを変更する必要はありません。

次の表には、**SameSiteCookieHandler** パラメーターの詳細が記載されています。

表17.4 SameSiteCookieHandler パラメーター

パラメーター名	存在	説明
add-secure-for-none	任意	このパラメーターは、**SameSite** 属性モードが **None** の場合に、Cookie に **Secure** 属性を追加します。デフォルト値は **true** です。
case-sensitive	任意	このパラメーターは、**cookie-pattern** が大文字と小文字を区別するかどうかを示します。デフォルト値は **true** です。

パラメーター名	存在	説明
cookie-pattern	任意	このパラメーターは、Cookie 名の regex パターンを受け入れます。このパラメーターが指定されていない場合、属性 **SameSite=<specified-mode>** がすべての Cookie に追加されます。
enable-client-checker	任意	このパラメーターは、クライアントアプリケーションが **SameSite=None** 属性と互換性がないかどうかを確認します。デフォルト値は **true** です。 このデフォルト値を使用し、**SameSite** 属性モードを **None** 以外の値に設定すると、パラメーターは検証を無視します。 互換性のないクライアントの問題を防ぐため、このパラメーターは **SameSite** 属性モードを **None** に設定することをスキップします。そのため、何も影響はありません。互換性のあるクライアントからのリクエストの場合、パラメーターは想定どおりに **SameSite** 属性モードを **None** に設定します。
mode	必須	このパラメーターは **SameSite** 属性モードを指定します。これは **Strict**、**Lax**、または **None** に設定できます。 クロスサイトリクエストフォージェリー攻撃に対するセキュリティーを向上させるために、一部のブラウザーではデフォルトの **SameSite** 属性モードを **Lax** に設定します。詳細は、**関連情報** セクションを参照してください。

SameSiteCookieHandler は、**cookie-pattern** に一致する Cookie に、または **cookie-pattern** が指定されていない場合はすべての Cookie に、属性 **SameSite=<specified-mode>** を追加します。属性 **SameSite=<specified-mode>** には、ユーザーが置換した変数、つまり **<specified-mode>** が含まれています。**cookie-pattern** は、**case-sensitive** に設定された値に従って照合されます。

ブラウザーの **SameSite** 属性を設定する前に、次の点を考慮してください。

- アプリケーションを確認して、Cookie に **SameSite** 属性が必要かどうか、またそれらの Cookie を保護する必要があるか確認します。
- すべての Cookie に対して **SameSite** 属性モードを **None** に設定すると、アプリケーションが攻撃を受けやすくなります。

式フィルターを使用して SameSiteCookieHandler を設定する手順

expression-filter を使用してサーバー上で **SameSiteCookieHandler** を設定するには、次の手順を実行します。

1. 次のコマンドを使用し、**SameSiteCookieHandler** を使用して新しい **expression-filter** を作成します。

```
/subsystem=undertow/configuration=filter/expression-filter=addSameSiteLax:add(expression="path-prefix('/mypathprefix') -> samesite-cookie(Lax)")
```

2. 次のコマンドを使用して、**undertow** Web サーバーの **expression-filter** を有効にします。

```
/subsystem=undertow/server=default-server/host=default-host/filter-ref=addSameSiteLax:add
```

設定ファイルを追加して SameSiteCookieHandler を設定する手順

undertow-handlers.conf ファイルを追加してアプリケーションに **SameSiteCookieHandler** を設定するには、次の手順を実行します。

1. WAR の WEB-INF ディレクトリーに **undertow-handlers.conf** ファイルを追加します。
2. **undertow-handlers.conf** ファイルに、特定の **SameSiteCookieHandler** パラメーターを指定して次のコマンドを追加します。

```
samesite-cookie(mode=<mode>)
```

mode パラメーターの有効な値は、**Strict**、**Lax**、または **None** です。上記のコマンドを使用すると、**cookie-pattern**、**case-sensitive**、**enable-client-checker**、または **add-secure-for-none** などの他の **SameSiteCookieHandler** パラメーターを設定することもできます。

関連情報

- Chromium サイトに関する情報
- Chrome サイトの情報
- Mozilla サイトの情報
- Microsoft サイトの情報
- IETF サイトの RFC に関する情報

17.9. デフォルトの WELCOME WEB アプリケーションの設定

JBoss EAP には、デフォルトではポート **8080** のルートコンテキストで表示されるデフォルトの **Welcome** アプリケーションが含まれます。

Undertow には、Welcome コンテンツに対応するデフォルトのサーバーが事前設定されています。

デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
    ...
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <http-invoker security-realm="ApplicationRealm"/>
```

```
        </host>
    </server>
    ...
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
</subsystem>
```

デフォルトのサーバー **default-server** にはデフォルトのホスト **default-host** が設定されています。デフォルトのホストは、**welcome-content** ファイルハンドラーで **<location>** 要素を使用して、サーバーのルートへのリクエストを処理するよう設定されています。**welcome-content** ハンドラーは **path** プロパティーに指定された場所でコンテンツを処理します。

このデフォルトの **Welcome** アプリケーションは、独自の Web アプリケーションで置き換えることができます。これは、以下の 2 つのいずれかの方法で設定できます。

- **welcome-content** ファイルハンドラーの変更
- **default-web-module** の変更

Welcome コンテンツを無効に することもできます。

welcome-content ファイルハンドラーの変更

1. 新しいデプロイメントを参照する、既存の **welcome-content** ファイルハンドラーのパスを変更します。

```
/subsystem=undertow/configuration=handler/file=welcome-content:write-attribute(name=path,value="/path/to/content")
```

注記

または、サーバーのルートにより使用される異なるファイルハンドラーを作成することもできます。

```
/subsystem=undertow/configuration=handler/file=NEW_FILE_HANDLER:add(path="/path/to/content")
/subsystem=undertow/server=default-server/host=default-host/location=\/:write-attribute(name=handler,value=NEW_FILE_HANDLER)
```

2. 変更を反映するためにサーバーをリロードします。

```
reload
```

default-web-module の変更

1. デプロイされた Web アプリケーションをサーバーのルートにマップします。

```
/subsystem=undertow/server=default-server/host=default-host:write-attribute(name=default-web-module,value=hello.war)
```

2. 変更を反映するためにサーバーをリロードします。

```
reload
```

-

デフォルトの Welcome Web アプリケーションの無効化

1. **default-host** の **location** エントリー (/) を削除して welcome アプリケーションを無効にします。

```
/subsystem=undertow/server=default-server/host=default-host/location=\/:remove
```

2. 変更を反映するためにサーバーをリロードします。

```
reload
```

17.10. HTTPS の設定

Web アプリケーションの HTTPS 設定に関する詳細は、**How to Configure Server Security**の Configure One-way and Two-way SSL/TLS for Applications を参照してください。

JBoss EAP 管理インターフェイスと使用するための HTTPS 設定に関する詳細は、**How to Configure Server Security**の How to Secure the Management Interfaces を参照してください。

17.11. HTTP セッションタイムアウトの設定

HTTP セッションタイムアウトは、HTTP セッションの無効を宣言するために必要な非アクティブな期間を定義します。たとえば、HTTP セッションを作成する JBoss EAP にデプロイされたアプリケーションにユーザーがアクセスしたとします。HTTP セッションタイムアウト後に同じユーザーが同じアプリケーションに再度アクセスしようとすると、元の HTTP セッションは無効化され、ユーザーは新しい HTTP セッションの作成を強制されます。これにより、永続化されなかったデータを損失したり、ユーザーを再認証する必要がある場合があります。

HTTP セッションタイムアウトは、アプリケーションの **web.xml** ファイルに設定されますが、デフォルトの HTTP セッションタイムアウトは JBoss EAP 内で指定できます。サーバーのタイムアウト値はデプロイされたすべてのアプリケーションに適用されますが、アプリケーションの **web.xml** はサーバーの値をオーバーライドします。

サーバーの値は、**undertow** サブシステムの **servlet-container** セクションにある **default-session-timeout** プロパティーに指定されます。**default-session-timeout** の値は分単位で指定され、デフォルトは **30** です。

デフォルトのセッションタイムアウトの設定
default-session-timeout を設定するには、以下を指定します。

```
/subsystem=undertow/servlet-container=default:write-attribute(name=default-session-timeout,
value=60)
```

```
reload
```

17.12. HTTP のみのセッション管理クッキーの設定

セッション管理クッキーは、JavaScript などの HTTP API および非 HTTP API の両方によってアクセスされます。JBoss EAP は **HttpOnly** ヘッダーを **Set-Cookie** 応答ヘッダーの一部としてクライアント (通常はブラウザー) に送信します。サポートされるブラウザーでこのヘッダーを有効にすると、非 HTTP API を経由してセッション管理クッキーへアクセスしないようにブラウザーに通知します。セッ

ション管理クッキーを HTTP API のみに制限すると、クロスサイトスクリプティングの攻撃よるセッションクッキーの窃盗のリスクを軽減することができます。この動作を有効にするには、**http-only** 属性を **true** に設定する必要があります。

重要

HttpOnly ヘッダーを使用しても、単にブラウザーに通知を行うだけで、クロスサイトスクリプティングによる攻撃を実際に防ぐわけではありません。この動作を反映するには、ブラウザーも **HttpOnly** をサポートしている必要があります。

重要

HttpOnly 属性を使用すると制限がセッション管理クッキーのみに適用され、その他のブラウザークッキーには適用されません。

http-only 属性は **undertow** サブシステムの 2 カ所で設定されます。

- セッションクッキー設定としてサーブレットコンテナーで設定
- 単一のサインオンプロパティーとしてサーバーのホストセクションで設定

host-only をサーブレットコンテナーセッションクッキーに設定
host-only プロパティーをサーブレットコンテナーセッションクッキーに設定するには、以下を指定します。

```
/subsystem=undertow/servlet-container=default/setting=session-cookie:add
```

```
/subsystem=undertow/servlet-container=default/setting=session-cookie:write-attribute(name=http-only,value=true)
```

```
reload
```

host-only をホストシングルサインオンに設定
host-only プロパティーをホストシングルサインオンに設定するには、以下を指定します。

```
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:add
```

```
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:write-attribute(name=http-only,value=true)
```

```
reload
```

17.13. HTTP/2 の設定

Undertow では、HTTP/2 標準を使用できます。この標準は、ヘッダーの圧縮と多くのストリームの多重化を同じ TCP 接続で行い、待ち時間を削減します。さらに、リクエストの前にサーバーがリソースをクライアントにプッシュできる機能も提供するため、ページのロードがより速くなります。

HTTP/2 は、HTTP/2 標準をサポートするクライアントとブラウザーでのみ機能することに注意してください。

重要

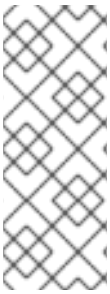
最新のブラウザーは、**h2** と呼ばれるセキュアな TLS 接続上の HTTP/2 を強制します。**h2c** と呼ばれるプレーン HTTP 上の HTTP/2 はサポートしないことがあります。**h2c** で HTTP/2 を使用するように JBoss EAP を設定することが可能です。つまり、HTTPS を使用せずに HTTP のアップグレードでプレーン HTTP のみを使用します。この場合、Undertow の HTTP リスナーで HTTP/2 を有効にします。

```
/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=enable-http2,value=true)
```

HTTP/2 を使用するよう Undertow を設定するには、**enable-http2** 属性を **true** に設定し、HTTP/2 を使用するよう Undertow の HTTPS リスナーを有効にします。

```
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=enable-http2,value=true)
```

HTTPS リスナーの情報や、Web アプリケーションで HTTPS を使用するよう Undertow を設定する方法については、**How to Configure Server Security**の Configure One-way and Two-way SSL/TLS for Applications を参照してください。



注記

elytron サブシステムで HTTP/2 を使用するには、Undertow の **https-listener** にある設定済みの **ssl-context** が変更可能として設定される必要があります。これには、適切な **server-ssl-context** の **wrap** 属性を **false** に設定します。デフォルトでは **wrap** 属性は **false** に設定されます。これは、ALPN に関する **ssl-context** の変更を行うために Undertow で必要になります。提供された **ssl-context** が書き込み可能でない場合、ALPN は使用できず、接続は HTTP/1.1 にフォールバックします。

HTTP/2 使用時の ALPN サポート

セキュアな TLS 接続上で HTTP/2 を使用する場合、ALPN TLS プロトコル拡張をサポートする TLS スタックが必要になります。このスタックの取得はインストールされた JDK によって異なります。

- Java 8 を使用する場合、ALPN 実装は Java 内部に依存して直接 JBoss EAP に導入されます。そのため、ALPN 実装は Oracle および OpenJDK でのみ動作します。IBM Java では動作しません。Red Hat は、ALPN 機能を実装する OpenSSL ライブラリーとともに、OpenSSL プロバイダーからの ALPN TLS プロトコル拡張サポートを JBoss EAP で使用することを推奨します。OpenSSL プロバイダーの ALPN TLS プロトコル拡張サポートを使用すると、パフォーマンスが向上します。

- Java 9 より、JDK は ALPN をネイティブでサポートするようになりましたが、Java 9 以上を使用する場合でも OpenSSL プロバイダーからの ALPN TLS プロトコル拡張サポートを使用するとパフォーマンスが向上されるはずです。

ALPN TLS プロトコル拡張サポートを取得するための OpenSSL のインストール手順は、JBoss Core Services からの OpenSSL のインストール を参照してください。標準のシステム OpenSSL は Red Hat Enterprise Linux 8 でサポートされており、追加の JBoss Core Services OpenSSL は必要ありません。

OpenSSL がインストールされたら、OpenSSL を使用するよう JBoss EAP を設定 にある手順にしたがいます。

HTTP/2 が使用されていることを検証

Undertow が HTTP/2 を使用していることを検証するには、Undertow からのヘッダーを確認する必要

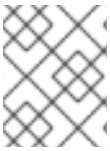
があります。https を使用して JBoss EAP インスタンスに移動し (例: https://localhost:8443)、ブラウザーの開発者ツールを使用してヘッダーを確認します。Google Chrome などの一部のブラウザーは、HTTP/2 の使用時には **:path**、**:authority**、**:method**、**:scheme** などの HTTP/2 擬似ヘッダーを表示します。Firefox や Safari などの他のブラウザーは、ヘッダーの状態またはバージョンを **HTTP/2.0** と表示します。

17.14. REQUESTDUMPING ハンドラーの設定

RequestDumping ハンドラーである **io.undertow.server.handlers.RequestDumpingHandler** は、JBoss EAP 内で Undertow によって処理されるリクエストとその応答オブジェクトの詳細をログに記録します。

重要

このハンドラーはデバッグに便利ですが、機密情報がログに記録される可能性があります。この点に留意してこのハンドラーを有効にしてください。



注記

RequestDumping ハンドラーは、JBoss EAP 6 の **RequestDumperValve** の代わりに使用されます。

RequestDumping ハンドラーは、JBoss EAP のサーバーレベルまたは個別のアプリケーション内のいずれかで設定できます。

17.14.1. サーバーでの RequestDumping ハンドラーの設定

RequestDumping ハンドラーは式フィルターとして設定する必要があります。**RequestDumping** ハンドラーを式フィルターとして設定するには、以下を行う必要があります。

RequestDumping ハンドラーで新しい式フィルターを作成する

```
/subsystem=undertow/configuration=filter/expression-
filter=requestDumperExpression:add(expression="dump-request")
```

Undertow Web サーバーで式フィルターを有効にする

```
/subsystem=undertow/server=default-server/host=default-host/filter-
ref=requestDumperExpression:add
```

重要

このように **RequestDumping** ハンドラーを式フィルターとして有効にすると、Undertow Web サーバーによって処理されるすべてのリクエストおよびそれらの応答がログに記録されます。

特定 URL に対して RequestDumping ハンドラーを設定する
すべてのリクエストをログに記録する他に、特定の URL のリクエストやそれらの応答のみをログに記録するために式フィルターを使用することもできます。これには、**path**、**path-prefix**、**path-suffix** などの述語を式に使用します。たとえば、**/myApplication/test** へのリクエストとそれらの応答をすべてログに記録するには、式フィルターの作成時に式 **"dump-request"** の代わりに **"path(/myApplication/test) -> dump-request"** を使用します。これにより、**/myApplication/test** に完全一致するパスを持つリクエストのみが **RequestDumping** ハンドラーに送られます。

17.14.2. アプリケーション内での RequestDumping ハンドラーの設定

サーバーで **RequestDumping** ハンドラーを設定する他に、個別のアプリケーション内で設定することもできます。これにより、ハンドラーの範囲がそのアプリケーションのみに制限されます。**RequestDumping** ハンドラーは **WEB-INF/undertow-handlers.conf** で設定する必要があります。

指定のアプリケーションのすべてのリクエストとそれらの応答をログに記録するよう **WEB-INF/undertow-handlers.conf** で **RequestDumping** ハンドラーを設定するには、以下の式を **WEB-INF/undertow-handlers.conf** に追加します。

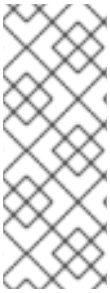
例: WEB-INF/undertow-handlers.conf

```
dump-request
```

指定のアプリケーション内での特定 URL のリクエストやそれらの応答のみをログに記録するよう、**WEB-INF/undertow-handlers.conf** で **RequestDumping** ハンドラーを設定するには、**path**、**path-prefix**、**path-suffix** などの述語を式に使用します。たとえば、アプリケーションの **test** へのリクエストとそれらの応答をすべてログに記録するには、**path** 述語が含まれる以下の式を使用できます。

例: WEB-INF/undertow-handlers.conf

```
path(/test) -> dump-request
```



注記

path、**path-prefix**、**path-suffix** などの述語をアプリケーションの **WEB-INF/undertow-handlers.conf** に定義された式で使用する場合、使用する値はアプリケーションのコンテキストルートからの相対値になります。たとえば、アプリケーションのコンテキストルートは **myApplication** で、式 **path(/test) -> dump-request** が **WEB-INF/undertow-handlers.conf** に設定されている場合、**/myApplication/test** へのリクエストとそれらの応答のみがログに記録されます。

17.15. クッキーセキュリティーの設定

secure-cookie ハンドラーを使用して、サーバーとクライアント間の接続上で作成されたクッキーのセキュリティーを強化できます。この場合、クッキーが設定される接続がセキュアであるとみなされると、クッキーの **secure** 属性が **true** に設定されます。

リスナーを設定したり、HTTPS を使用すると、接続をセキュアにすることができます。**secure-cookie** ハンドラーを設定するには、**undertow** サブシステムの **expression-filter** を定義します。詳細は、フィルターの設定 を参照してください。

secure-cookie ハンドラーが使用されている場合、セキュアな接続上で設定されたクッキーは暗黙的にセキュアとして設定され、セキュアでない接続上で送信されることはありません。

17.16. UNDERTOW サブシステムの調整

undertow サブシステムのパフォーマンスを最適化するための情報は、Performance Tuning Guideの Undertow Subsystem Tuning を参照してください。

第18章 REMOTING の設定

18.1. REMOTING サブシステムについて

remoting サブシステムは、ローカルおよびリモートサービスのインバウンドおよびアウトバウンド接続の設定を可能にします。

JBoss Remoting プロジェクトには、endpoint、connector、http-connector、および一連のローカルおよびリモート接続 URI などの設定可能な要素が含まれています。

ほとんどのユースケースでは、**remoting** サブシステムを設定する必要がない場合があります。アプリケーションにカスタム connector を使用する場合は、**remoting** サブシステムを設定する必要があります。

Jakarta Enterprise Beans などの、リモーティングクライアントとして動作するアプリケーションには特定のコネクターに接続するための別の設定が必要になります。

Remoting サブシステムのデフォルト設定

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
    <endpoint/>
    <http-connector name="http-remoting-connector" connector-ref="default" security-
realm="ApplicationRealm"/>
</subsystem>
```

remoting サブシステムで使用できる属性の完全リストは、**Remoting** サブシステムの属性 を参照してください。

remoting エンドポイント

リモーティングエンドポイントは、**io** サブシステムによって宣言および設定される XNIO ワーカーを使用します。

リモーティングエンドポイントの設定方法の詳細は、エンドポイントの設定 を参照してください。

connector

connector は、JBoss Remoting プロジェクトの主要な remoting 設定要素であり、外部クライアントが特定のポートでサーバーに接続できるようにするために使用されます。**connector** を介してサーバーに接続する必要があるクライアントは、サーバーを参照する URL で Remoting **remote** プロトコルを使用する必要があります (例:remote://localhost:4447)。

複数の connector を設定できます。各 connector は、いくつかのサブ要素と、**socket-binding** や **ssl-context** などの他のいくつかの属性を備えた **<connector>** 要素で設定されます。

いくつかの JBoss EAP サブシステムは、デフォルトの connector を使用できます。カスタム connector の要素と属性の設定は、アプリケーションによって異なります。詳細は Red Hat グローバルサポートサービスまでお問い合わせください。

connector の設定方法の詳細は、connector の設定 を参照してください。

http-connector

http-connector 要素は、特別なコネクター設定要素です。外部クライアントは、この要素を使用し、**undertow** の HTTP アップグレード機能を使用してサーバーに接続できます。

この設定では、クライアントは最初に HTTP プロトコルを使用してサーバーとの接続を確立し、次に同じ接続を介して **remote** プロトコルを使用します。これにより、**undertow** のデフォルトポート 8080

など、異なるプロトコルを使用するクライアントが同じポート経由で接続できるようになります。同じポートを介して接続すると、サーバー上で開いているポートの数を減らすことができます。

HTTP アップグレードによるサーバーへの接続が必要なクライアントは、暗号化されていない接続の場合は remoting **remote+http** プロトコルを使用するか、暗号化された接続の場合は remoting **remote+https** プロトコルを使用する必要があります。

アウトバウンド接続
3 つのタイプのアウトバウンド接続を指定することができます。

- URI によって指定される アウトバウンド接続
- ソケットなどのローカルリソースに接続する ローカルアウトバウンド接続
- リモートリソースに接続し、セキュリティーレルムを使用して認証を行う リモートアウトバウンド接続

追加の設定
リモーティングは、ネットワークインターフェイスや IO ワーカーなどの **remoting** サブシステム外部で設定された複数の要素に依存します。

詳細は、リモーティングの追加設定 を参照してください。

18.2. エンドポイントの設定

重要

JBoss EAP 6 では、ワーカースレッドプールは直接 **remoting** サブシステムで設定されていました。JBoss EAP 7 では、リモーティング **endpoint** 設定が **io** サブシステムからワーカーを参照します。

JBoss EAP は 以下の **endpoint** 設定をデフォルトで提供します。

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
  <endpoint/>
  ...
</subsystem>
```

既存のエンドポイント設定の更新

```
/subsystem=remoting/configuration=endpoint:write-attribute(name=authentication-retries,value=2)
```

```
reload
```

新規エンドポイント設定の作成

```
/subsystem=remoting/configuration=endpoint:add
```

エンドポイント設定の削除

```
/subsystem=remoting/configuration=endpoint:remove
```

```
reload
```

- エンドポイント設定で使用できる属性の完全リストは、エンドポイントの設定 を参照してください。

18.3. コネクターの設定

コネクターはリモーティングに関する主な設定要素で、追加設定のサブ要素が複数含まれます。

既存のコネクター設定の更新

```
/subsystem=remoting/connector=new-connector:write-attribute(name=socket-binding,value=my-socket-binding)
```

```
reload
```

新規コネクターの作成

```
/subsystem=remoting/connector=new-connector:add(socket-binding=my-socket-binding)
```

コネクターの削除

```
/subsystem=remoting/connector=new-connector:remove
```

```
reload
```

コネクターの設定に使用できる属性の完全リストは Remoting サブシステムの属性 の項を参照してください。

18.4. HTTP コネクターの設定

HTTP コネクターは、HTTP アップグレードベースのリモーティングコネクターの設定を提供します。JBoss EAP はデフォルトで次の **http-connector** 設定を提供します。

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
    ...
    <http-connector name="http-remoting-connector" connector-ref="default" security-realm="ApplicationRealm"/>
</subsystem>
```

デフォルトでは、この HTTP コネクターは **undertow** サブシステムに設定される **default** という名前の HTTP リスナーに接続します。詳細は、Web サーバーの設定 (Undertow) を参照してください。

既存の HTTP コネクター設定の更新

```
/subsystem=remoting/http-connector=new-connector:write-attribute(name=connector-ref,value=new-connector-ref)
```

```
reload
```

新規 HTTP コネクターの作成

```
/subsystem=remoting/http-connector=new-connector:add(connector-ref=default)
```

HTTP コネクターの削除

```
/subsystem=remoting/http-connector=new-connector:remove
```

HTTP コネクターの設定に使用できる属性の完全リストは、コネクターの属性 を参照してください。

18.5. アウトバウンド接続の設定

アウトバウンド接続は、URI によって完全に指定される汎用のリモーティングアウトバウンド接続です。

既存のアウトバウンド接続の更新

```
/subsystem=remoting/outbound-connection=new-outbound-connection:write-attribute(name=uri,value=http://example.com)
```

新規アウトバウンド接続の作成

```
/subsystem=remoting/outbound-connection=new-outbound-connection:add(uri=http://example.com)
```

アウトバウンド接続の削除

```
/subsystem=remoting/outbound-connection=new-outbound-connection:remove
```

アウトバウンド接続の設定に使用できる属性の完全リストは、アウトバウンド接続の属性 を参照してください。

18.6. リモートアウトバウンド接続の設定

リモートアウトバウンド接続は、プロトコル、アウトバウンドソケットバインディング、ユーザー名、およびセキュリティーレルムによって指定されます。プロトコルは **remote**、**http-remoting**、**https-remoting** のいずれかになります。

既存のリモートアウトバウンド接続の更新

```
/subsystem=remoting/remote-outbound-connection=new-remote-outbound-connection:write-attribute(name=outbound-socket-binding-ref,value=outbound-socket-binding)
```

新規リモートアウトバウンド接続の作成

```
/subsystem=remoting/remote-outbound-connection=new-remote-outbound-connection:add(outbound-socket-binding-ref=outbound-socket-binding)
```

リモートアウトバウント接続の削除

```
/subsystem=remoting/remote-outbound-connection=new-remote-outbound-connection:remove
```

リモートアウトバウンド接続の設定に使用できる属性の完全リストは、リモートアウトバウンド接続の属性 を参照してください。

18.7. ローカルアウトバウンド接続の設定

ローカルアウトバウンド接続はプロトコルが **local** のリモーティングアウトバウンド接続で、アウトバウントソケットバインディングのみによって指定されます。

既存のローカルアウトバウンド接続の更新

```
/subsystem=remoting/local-outbound-connection=new-local-outbound-connection:write-attribute(name=outbound-socket-binding-ref,value=outbound-socket-binding)
```

新規ローカルアウトバウンド接続の作成

```
/subsystem=remoting/local-outbound-connection=new-local-outbound-connection:add(outbound-socket-binding-ref=outbound-socket-binding)
```

ローカルアウトバウンド接続の削除

```
/subsystem=remoting/local-outbound-connection=new-local-outbound-connection:remove
```

ローカルアウトバウンド接続の設定に使用できる属性の完全リストは、ローカルアウトバウンド接続の属性 を参照してください。

18.8. リモーティングの追加設定

remoting サブシステム外部に接続されるリモーティング要素が複数あります。

IO ワーカー

以下のコマンドを使用してリモーティングの IO ワーカーを設定します。

```
/subsystem=remoting/configuration=endpoint:write-attribute(name=worker, value=WORKER_NAME)
```

IO ワーカーの設定方法に関する詳細は ワーカーの設定 を参照してください。

ネットワークインターフェイス

remoting サブシステムによって使用されるネットワークインターフェイスは **public** インターフェイスです。このインターフェイスは他のサブシステムによっても使用されるため、変更する場合は十分注意してください。

```
<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}"/>
  </interface>
  <interface name="unsecure">
    <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}"/>
  </interface>
</interfaces>
```

マネージドドメインでは、**public** インターフェイスはホストごとに **host.xml** ファイルで定義されます。

ソケットバインディング

remoting サブシステムによって使用されるデフォルトのソケットバインディングはポート **8080** にバインドされます。
ソケットバインディングおよびソケットバインディンググループの詳細は、ソケットバインディング を参照してください。

セキュアなトランスポート設定

リモーティングトランスポートはクライアントの要求があれば STARTTLS を使用してセキュアな接続 (HTTPS、Secure Servlet など) を使用します。セキュアな接続とセキュアでない接続の両方で同じソケットバインディングまたはネットワークポートが使用されるため、サーバー側に追加の設定をする必要はありません。クライアントは必要に応じてセキュアなトランスポートまたはセキュアでないトランスポートを要求します。Jakarta Enterprise Beans、ORB、および Java Messaging Service プロバイダーなどのリモーティングを使用する JBoss EAP のコンポーネントはデフォルトでセキュアなインターフェイスを使用します。

警告

STARTTLS はクライアントの要求があればセキュアな接続を有効にしますが、セキュアでない接続がデフォルトになります。本質的に、StartTLS は攻撃者がクライアントの要求を妨害し、要求を編集してセキュアでない接続を要求する中間者攻撃の対象になりやすい欠点があります。セキュアでない接続が適切なフォールバックである場合を除き、クライアントがセキュアな接続を取得できなかったときに適切に失敗するよう記述する必要があります。

第19章 IO サブシステムの設定

19.1. IO サブシステムの概要

io サブシステムは、Undertow や Remoting などの他のサブシステムによって使用される XNIO ワーカー と バッファープール を定義します。これらのワーカーやバッファープールは、**io** サブシステムの以下のコンポーネント内で定義されます。

IO サブシステムのデフォルト設定

```
<subsystem xmlns="urn:jboss:domain:io:3.0">
    <worker name="default"/>
    <buffer-pool name="default"/>
</subsystem>
```

19.2. ワーカーの設定

ワーカーは XNIO ワーカーインスタンスです。XNIO ワーカーインスタンスは、IO およびワーカースレッドの管理や SSL サポートなどの機能を提供する Java NIO API の抽象化レイヤーです。JBoss EAP はデフォルトで **default** という単一のワーカーを提供しますが、複数のワーカーを定義できます。

既存のワーカーの更新
既存のワーカーを更新するには、以下を指定します。

```
/subsystem=io/worker=default:write-attribute(name=io-threads,value=10)
```

```
reload
```

新規ワーカーの作成
新規ワーカーを作成するには、以下を指定します。

```
/subsystem=io/worker=newWorker:add
```

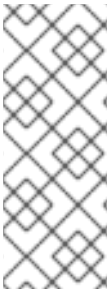
ワーカーの削除
ワーカーを削除するには、以下を指定します。

```
/subsystem=io/worker=newWorker:remove
```

```
reload
```

ワーカーの設定に使用できる属性の完全リストは IO サブシステムの属性 の項を参照してください。

19.3. バッファープールの設定



注記

IO バッファープールは非推奨となっていますが、現行リリースではデフォルトとして設定されています。バッファープールはプールされた NIO バッファーインスタンスです。バッファーサイズの変更はアプリケーションのパフォーマンスに大きく影響します。通常、ほとんどのサーバーでは 16K が理想のバッファーサイズになります。Undertow バイトバッファープールの設定の詳細は、JBoss EAP設定ガイドの Undertow バイトバッファープール を参照してください。

既存のバッファープールの更新
既存のバッファープールを更新するには、以下を指定します。

```
/subsystem=io/buffer-pool=default:write-attribute(name=direct-buffers,value=true)
```

```
reload
```

バッファープールの作成
新しいバッファープールを作成するには、以下を指定します。

```
/subsystem=io/buffer-pool=newBuffer:add
```

バッファープールの削除
バッファープールを削除するには、以下を指定します。

```
/subsystem=io/buffer-pool=newBuffer:remove
```

```
reload
```

バッファープールの設定に使用できる属性の完全リストは IO サブシステムの属性 の項を参照してください。

19.4. IO サブシステムの調整

io サブシステムのパフォーマンスを監視および最適化するための情報は、**Performance Tuning Guide**の IO Subsystem Tuning の項を参照してください。

第20章 WEB サービスの設定

JBoss EAP では、管理コンソールまたは管理 CLI を使用して **webservices** システム経由でデプロイされた Web サービスの動作を設定できます。パブリッシュされたエンドポイントアドレスやハンドラーチェーンを設定できます。また、Web サービスのランタイム統計収集を有効にすることもできます。

詳細は、JBoss EAP**Developing Web Services Applications**の Configuring the Web Services Subsystem を参照してください。

第21章 JAKARTA SERVER FACES 設定

21.1. JAKARTA SERVER FACES の複数の JAKARTA SERVER FACES 実装

jsf サブシステムでは、複数の Jakarta Server Faces 実装を同じ JBoss EAP サーバーインスタンスにインストールできます。Jakarta Server Faces 仕様 2.3 以降を実装する Sun Mojarra または Apache MyFaces のバージョンをインストールできます。

21.1.1. Jakarta Server Faces 実装のインストール

以下の手順は、新しい Jakarta Server Faces 実装を手作業でインストールし、デフォルトの実装にする方法になります。

1. Jakarta Server Faces 実装 JAR ファイルを追加 します。
2. Jakarta Server Faces API JAR ファイルを追加 します。
3. Jakarta Server Faces インジェクション JAR ファイルを追加 します。
4. MyFaces をインストールする場合は **commons-digester** JAR ファイルを追加 します。
5. デフォルトの Jakarta Server Faces 実装 を設定します。

Jakarta Server Faces 実装 JAR ファイルの追加

1. Jakarta Server Faces 実装の **EAP_HOME/modules/** ディレクトリーに適切なディレクトリー構造を作成します。

```
$ cd EAP_HOME/modules/
$ mkdir -p com/sun/jsf-impl/IMPL_NAME-VERSION
```

注記

たとえば、**IMPL_NAME-VERSION** を、Jakarta Server Faces 仕様 2.3 以降をサポートする Mojarra のバージョンに置き換えます。

2. Jakarta Server Faces 実装 JAR ファイルを **IMPL_NAME-VERSION/** サブディレクトリーにコピーします。
3. **IMPL_NAME-VERSION/** サブディレクトリーで、この Mojarra テンプレート またはこの MyFaces テンプレート と似た **module.xml** ファイルを作成します。テンプレートを使用する場合は、置き換え可能な変数に適切な値を適用してください。

Jakarta Server Faces API JAR ファイルの追加

1. Jakarta Server Faces 実装の **EAP_HOME/modules/** ディレクトリーに適切なディレクトリー構造を作成します。

```
$ cd EAP_HOME/modules/
$ mkdir -p javax/faces/api/IMPL_NAME-VERSION
```

2. Jakarta Server Faces API JAR ファイルを **IMPL_NAME-VERSION/** サブディレクトリーにコピーします。

3. **IMPL_NAME-VERSION/** サブディレクトリーで、この Mojarra テンプレート またはこの MyFaces テンプレート と似た **module.xml** ファイルを作成します。テンプレートを使用する場合は、置き換え可能な変数に適切な値を適用してください。

Jakarta Server Faces インジェクション JAR ファイルの追加

1. Jakarta Server Faces 実装の **EAP_HOME/modules/** ディレクトリーに適切なディレクトリー構造を作成します。

```
$ cd EAP_HOME/modules/
$ mkdir -p org/jboss/as/jsf-injection/IMPL_NAME-VERSION
```

2. Patching and Upgrading ガイドの指示に従い、お使いの JBoss EAP インスタンスの最新の累積パッチをダウンロードします。次に、以下のいずれかの手順を実行します。

   - パッチ更新をサーバーに適用していない場合は、**EAP_HOME/modules/system/layers/base/org/jboss/as/jsf-injection/main/** から **IMPL_NAME-VERSION/** サブディレクトリーに **wildfly-jsf-injection** と **weld-core-jsf** JAR ファイルをコピーします。

   - パッチ更新をサーバーに適用した場合は、最新のパッチ更新ディレクトリーから **IMPL_NAME-VERSION/** サブディレクトリーに **wildfly-jsf-injection** と **weld-core-jsf** JAR ファイルをコピーします。たとえば、**EAP_HOME/modules/system/layers/base/.overlays/layer-base-jboss-eap-7.4.z.CP/org/jboss/as/jsf-injection** などです。**z** は最新のバージョン番号です。

3. **IMPL_NAME-VERSION/** サブディレクトリーで、この Mojarra テンプレート またはこの MyFaces テンプレート と似た **module.xml** ファイルを作成します。テンプレートを使用する場合は、置き換え可能な変数に適切な値を適用してください。

MyFaces の commons-digester JAR ファイルの追加

1. **commons-digester** JAR の **EAP_HOME/modules/** ディレクトリーに適切なディレクトリー構造を作成します。

```
$ cd EAP_HOME/modules/
$ mkdir -p org/apache/commons/digester/main
```

2. **commons-digester** JAR ファイルをダウンロードし、**main/** サブディレクトリーにコピーします。

3. **main/** サブディレクトリーで、この テンプレート と似た **module.xml** ファイルを作成します。テンプレートを使用する場合は、置き換え可能な変数に適切な値を適用してください。

デフォルトの Jakarta Server Faces 実装の設定

1. 以下の管理 CLI コマンドを実行して、新しい Jakarta Server Faces 実装をデフォルト実装として設定します。

```
/subsystem=jsf:write-attribute(name=default-jsf-impl-slot,value=IMPL_NAME-VERSION)
```

2. JBoss EAP サーバーを再起動し、変更を反映します。

21.1.2. マルチ Jakarta Server Faces 実装サポート

JBoss EAP 7.4 には、単一の Jakarta Server Faces 実装である Mojarra をベースとした Jakarta Server Faces 2.3 実装が含まれています。

マルチ Jakarta Server Faces を使用すると、複数の Jakarta Server Faces 実装およびバージョンを同じ JBoss EAP サーバーにインストールできます。この目的は、Jakarta Server Faces 実装、MyFaces または Mojarra のいずれか、ならびに Faces 2.1 以上、Jakarta Server Faces 2.3 以上のバージョンをすべて使用できるようにすることです。マルチ Jakarta Server Faces は、コンテナーと完全統合された実装を提供するため、より効率的なアノテーション処理、ライフサイクル処理、およびその他の統合の利点を実現できます。

21.1.2.1. マルチ Jakarta Server Faces 実装の仕組み

各 Jakarta Server Faces バージョンごとに新しいスロットが **com.sun.jsf-impl**、**javax.faces.api**、および **org.jboss.as.jsf-injection** 下のモジュールパスに作成されることで、マルチ Jakarta Server Faces は機能します。**jsf** サブシステムが開始されると、モジュールパスをスキャンしてインストールされた Jakarta Server Faces 実装をすべて見つけます。**jsf** サブシステムが指定されたコンテキストパラメーターが含まれる web アプリケーションをデプロイすると、スロットが作成されたモジュールをデプロイメントに追加します。

たとえば、MyFaces 2.2.12 がサーバー上にインストールされていることを仮定して、Jakarta Server Faces アプリケーションは MyFaces 2.2.12 を使用すべきであることを示すには、以下のコンテキストパラメーターを追加する必要があります。

```
<context-param>
  <param-name>org.jboss.jbossfaces.JSF_CONFIG_NAME</param-name>
  <param-value>myfaces-2.2.12</param-value>
</context-param>
```

21.1.2.2. デフォルトの Jakarta Server Faces 実装の変更

マルチ Jakarta Server Faces 機能では、**jsf** サブシステムに **default-jsf-impl-slot** 属性が含まれています。この属性を使用すると、以下の手順のようにデフォルトの Jakarta Server Faces 実装を変更することができます。

1. **write-attribute** コマンドを使用して、**default-jsf-impl-slot** 属性の値をアクティブな Jakarta Server Faces 実装の 1 つに設定します。

   ```
   /subsystem=jsf:write-attribute(name=default-jsf-impl-slot,value=JSF_IMPLEMENTATION)
   ```

2. 変更を反映するために、JBoss EAP サーバーを再起動します。

   ```
   reload
   ```

インストールされている Jakarta Server Faces 実装を確認するには、**list-active-jsf-impls** 操作を実行します。

```
/subsystem=jsf:list-active-jsf-impls
{
    "outcome" => "success",
    "result" => [
        "myfaces-2.1.12",
        "mojarra-2.2.0-m05",
```

```
        "main"
    ]
}
```

21.2. DOCTYPE 宣言の拒否

以下の管理 CLI コマンドを使用すると、Jakarta Server Faces デプロイメントの **DOCTYPE** 宣言を拒否することができます。

```
/subsystem=jsf:write-attribute(name=disallow-doctype-decl,value=true)
reload
```

特定の Jakarta Server Faces デプロイメントの設定をオーバーライドするには、デプロイメントの **web.xml** ファイルに **com.sun.faces.disallowDoctypeDecl** コンテキストパラメーターを追加します。

```
<context-param>
  <param-name>com.sun.faces.disallowDoctypeDecl</param-name>
  <param-value>false</param-value>
</context-param>
```

第22章 バッチアプリケーションの設定

JBoss EAP 7 は Jakarta Batch をサポートします。バッチアプリケーションを実行するための環境を設定し、**batch-jberet** サブシステムを使用してバッチジョブを管理できます。

バッチアプリケーションの開発に関する詳細は、JBoss EAP**Development Guide**の Jakarta Batch Application Development を参照してください。

22.1. BATCH ジョブの設定

JBeret 実装を基にした **batch-jberet** サブシステムを使用してバッチジョブを設定できます。

デフォルトの **batch-jberet** サブシステム設定は、インメモリージョブリポジトリーとデフォルトのスレッドプールの設定を定義します。

```
<subsystem xmlns="urn:jboss:domain:batch-jberet:2.0">
    <default-job-repository name="in-memory"/>
    <default-thread-pool name="batch"/>
    <job-repository name="in-memory">
        <in-memory/>
    </job-repository>
    <thread-pool name="batch">
        <max-threads count="10"/>
        <keepalive-time time="30" unit="seconds"/>
    </thread-pool>
</subsystem>
```

デフォルトでは、サーバーの一時停止中に停止したバッチジョブはサーバーの再開時に再度開始されます。**restart-jobs-on-resume** プロパティーを **false** に設定すると **STOPPED** 状態のジョブをそのままにすることができます。

```
/subsystem=batch-jberet:write-attribute(name=restart-jobs-on-resume,value=false)
```

バッチ ジョブリポジトリー および スレッドプール を設定することもできます。

22.1.1. バッチジョブリポジトリーの設定

本項では、管理 CLI を使用してバッチジョブ情報を保存するインメモリーおよび JDBC ジョブリポジトリーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Batch (JBeret)** と選択し、表示 をクリックして左側のメニューで **In Memory** または **JDBC** のいずれかを選択すると、ジョブリポジトリーを設定することができます。

インメモリージョブリポジトリーの追加
バッチジョブ情報をメモリーに保存するジョブリポジトリーを追加できます。

```
/subsystem=batch-jberet/in-memory-job-repository=REPOSITORY_NAME:add
```

JDBC ジョブリポジトリー
バッチジョブ情報をデータベースに保存するジョブリポジトリーを追加できます。データソースの名前を指定してデータベースに接続する必要があります。

```
/subsystem=batch-jberet/jdbc-job-repository=REPOSITORY_NAME:add(data-
source=DATASOURCE)
```

デフォルトのジョブリポジトリーの設定
インメモリーまたは JDBC ジョブリポジトリーをバッチアプリケーションのデフォルトのジョブリポジトリーとして設定できます。

```
/subsystem=batch-jberet:write-attribute(name=default-job-repository,value=REPOSITORY_NAME)
```

サーバーをリロードする必要があります。

```
reload
```

22.1.2. バッチスレッドプールの設定

本項では、管理 CLI を使用してバッチジョブに使用するスレッドプールとスレッドファクトリーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Batch (JBeret)** と選択し、表示 をクリックして左側のメニューで **Thread Factory** または **Thread Pool** のいずれかを選択すると、スレッドプールとスレッドファクトリーを設定できます。

スレッドプールの設定
スレッドプールを追加するときに **max-threads** を指定する必要があります。 パーティションのジョブが想定どおりに実行されるように 2 つのスレッドが予約されているため、**3** よりも大きい値を常に設定してください。

1. スレッドプールを追加します。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:add(max-threads=10)
```

2. 必要な場合は **keepalive-time** の値を設定します。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:write-
attribute(name=keepalive-time,value={time=60,unit=SECONDS})
```

スレッドファクトリーの使用

1. スレッドファクトリーを追加します。

```
/subsystem=batch-jberet/thread-factory=THREAD_FACTORY_NAME:add
```

2. スレッドファクトリーの属性を設定します。
   - **group-name** - このスレッドファクトリーに作成するスレッドグループの名前。
   - **priority** - 作成されたスレッドの優先度。
   - **thread-name-pattern** - スレッドの名前の作成に使用されるテンプレート。以下のパターンを使用できます。
     - **%%** - パーセント記号
     - **%t** - ファクトリーごとのスレッドシーケンス番号
     - **%g** - グローバルスレッドシーケンス番号
     - **%f** - ファクトリーシーケンス番号
     - **%i** - スレッド ID

3. スレッドファクトリーをスレッドプールに割り当てます。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:write-attribute(name=thread-factory,value=THREAD_FACTORY_NAME)
```

サーバーをリロードする必要があります。

```
reload
```

デフォルトスレッドプールの設定
別のスレッドプールをデフォルトのスレッドプールとして設定できます。

```
/subsystem=batch-jberet:write-attribute(name=default-thread-pool,value=THREAD_POOL_NAME)
```

サーバーをリロードする必要があります。

```
reload
```

スレッドプールの統計表示
read-resource 管理 CLI 操作を使用するとバッチスレッドプールのランタイム情報を表示できます。このランタイム情報を表示するには **include-runtime=true** パラメーターを使用する必要があります。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "active-count" => 0,
        "completed-task-count" => 0L,
        "current-thread-count" => 0,
        "keepalive-time" => undefined,
        "largest-thread-count" => 0,
        "max-threads" => 15,
        "name" => "THREAD_POOL_NAME",
        "queue-size" => 0,
        "rejected-count" => 0,
        "task-count" => 0L,
        "thread-factory" => "THREAD_FACTORY_NAME"
    }
}
```

管理コンソールの Runtime タブで Batch サブシステムを選択して、バッチスレッドプールのランタイム情報を表示することもできます。

22.2. バッチジョブの管理

デプロイメントの **batch-jberet** サブシステムリソースを使用すると、バッチジョブを起動、停止、および再起動でき、実行詳細を表示することもできます。バッチジョブは 管理 CLI または 管理コンソール から管理できます。

管理 CLI からのバッチジョブの管理
バッチジョブの再開
STOPPED または **FAILED** 状態のジョブを再開するには、実行 ID を指定し、任意でバッチジョブの再開時に使用するプロパティーを指定します。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:restart-job(execution-
id=EXECUTION_ID,properties={PROPERTY=VALUE})
```

実行 ID はジョブインスタンスが最後に実行された ID である必要があります。

バッチジョブの開始
バッチジョブを開始するには、ジョブ XML ファイルを指定し、任意でバッチジョブの再開時に使用するプロパティーを指定します。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:start-job(job-xml-
name=JOB_XML_NAME,properties={PROPERTY=VALUE})
```

バッチジョブの停止
実行中のバッチジョブを停止するには、実行 ID を指定します。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:stop-job(execution-
id=EXECUTION_ID)
```

バッチジョブ実行詳細の表示
バッチジョブ実行の詳細を表示することができます。このランタイム情報を表示するには、 **read-resource** 操作で **include-runtime=true** パラメーターを使用する必要があります。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:read-resource(recursive=true,include-
runtime=true)
{
    "outcome" => "success",
    "result" => {"job" => {"import-file" => {
        "instance-count" => 2,
        "running-executions" => 0,
        "execution" => {
            "2" => {
                "batch-status" => "COMPLETED",
                "create-time" => "2016-04-11T22:03:12.708-0400",
                "end-time" => "2016-04-11T22:03:12.718-0400",
                "exit-status" => "COMPLETED",
                "instance-id" => 58L,
                "last-updated-time" => "2016-04-11T22:03:12.719-0400",
                "start-time" => "2016-04-11T22:03:12.708-0400"
            },
            "1" => {
                "batch-status" => "FAILED",
                "create-time" => "2016-04-11T21:57:17.567-0400",
                "end-time" => "2016-04-11T21:57:17.596-0400",
                "exit-status" => "Error : org.hibernate.exception.ConstraintViolationException: could not
execute statement",
                "instance-id" => 15L,
                "last-updated-time" => "2016-04-11T21:57:17.597-0400",
                "start-time" => "2016-04-11T21:57:17.567-0400"
            }
        }
    }}}
}
```

管理コンソールからのバッチジョブの管理

管理コンソールからバッチジョブを管理するには、**Runtime** タブでサーバーを選択し、**Batch (JBeret)** を選択してリストからジョブを選びます。

バッチジョブの再開
実行を選択して **Restart** をクリックし、**STOPPED** ジョブを再開します。

バッチジョブの開始
ジョブを選択してドロップダウンメニューで **Start** を選択し、バッチジョブの新たな実行を開始します。

バッチジョブの停止
実行を選択して **Stop** をクリックし、バッチジョブの実行を停止します。

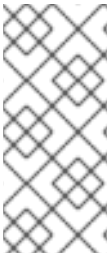
バッチジョブ実行詳細の表示
表にリストされる各実行のジョブ実行詳細が表示されます。

22.3. バッチジョブのセキュリティー設定

batch-jberet サブシステムを設定すると、Elytron セキュリティードメインで バッチジョブを実行することができます。これにより、バッチジョブをセキュアに一時停止でき、同じセキュアなアイデンティティーで再開できます。たとえば、**batch-jberet** サブシステムを使用して、バッチジョブを開始するためにセキュアな RESTful エンドポイントが作成されます。RESTful エンドポイントと **batch-jberet** サブシステムの両方が同じセキュリティードメインを使用してセキュア化された場合、または **batch-jberet** セキュリティードメインが RESTful エンドポイントのセキュリティードメインドメインを信頼した場合、このように開始されたバッチジョブはセキュアに一時停止され、同じセキュアなアイデンティティーによって再開されます。

以下の管理 CLI コマンドを使用して **security-domain** 属性を更新し、バッチジョブのセキュリティーを設定します。

```
/subsystem=batch-jberet:write-attribute(name=security-domain, value=ExampleDomain)

reload
```



注記

バッチジョブには、**org.wildfly.extension.batch.jberet.deployment.BatchPermission** パーミッションが必要です。これは、**javax.batch.operations.JobOperator** に対応する **start**、**stop**、**restart**、**abandon** および **read** を提供します。**default-permission-mapper** マッパーは **org.wildfly.extension.batch.jberet.deployment.BatchPermission** パーミッションを提供します。

第23章 NAMING サブシステムの設定

23.1. NAMING サブシステム

naming サブシステムは JBoss EAP の JNDI 実装を提供します。このサブシステムを設定して、 グローバル JNDI 名前空間のエントリーをバインド することができます。さらに、このサブシステムを設定して リモート JNDI をアクティブまたは非アクティブ にすることもできます。

以下は、すべての要素と属性が指定された **naming** サブシステムの XML 設定例になります。

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
    <bindings>
        <simple name="java:global/simple-integer-binding" value="100" type="int" />
        <simple name="java:global/jboss.org/docs/url" value="https://docs.jboss.org"
type="java.net.URL" />
        <object-factory name="java:global/foo/bar/factory" module="org.foo.bar"
class="org.foo.bar.ObjectFactory" />
        <external-context name="java:global/federation/ldap/example"
class="javax.naming.directory.InitialDirContext" cache="true">
            <environment>
                <property name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory" />
                <property name="java.naming.provider.url" value="ldap://ldap.example.com:389" />
                <property name="java.naming.security.authentication" value="simple" />
                <property name="java.naming.security.principal" value="uid=admin,ou=system" />
                <property name="java.naming.security.credentials" value="secret" />
            </environment>
        </external-context>
        <lookup name="java:global/new-alias-name" lookup="java:global/original-name" />
    </bindings>
    <remote-naming/>
</subsystem>
```

23.2. グローバルバインディングの設定

naming サブシステムは、エントリーを **java:global**、**java:jboss**、または **java** グローバル JNDI 名前空間へバインドできるようにしますが、標準のポータブルな **java:global** 名前空間を使用することが推奨されます。

グローバルバインディングは **naming** サブシステムの **<bindings>** 要素で設定されます。以下の 4 種類のバインディングがサポートされます。

- シンプルバインディング
- オブジェクトファクトリーバインディング
- 外部コンテキストバインディング
- バインディングルックアップエイリアス

シンプルバインディングの設定
simple XML 設定要素は、プリミティブまたは **java.net.URL** エントリーにバインドします。

- **name** 属性は必須で、エントリーのターゲット JNDI 名を指定します。

- **value** 属性は必須で、エントリーの値を定義します。
- 任意の **type** 属性はエントリーの値の型を指定し、デフォルトは **java.lang.String** になります。**java.lang.String** の他に、**int** または **java.lang.Integer**、および **java.net.URL** などのプリミティブ型や対応するオブジェクトラッパークラスを指定できます。

以下に、シンプルバインディングを作成する管理 CLI コマンドの例を示します。

```
/subsystem=naming/binding=java\:global\/simple-integer-binding:add(binding-type=simple, type=int,
value=100)
```

結果の XML 設定

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
    <bindings>
        <simple name="java:global/simple-integer-binding" value="100" type="int"/>
    </bindings>
    <remote-naming/>
</subsystem>
```

以下のコマンドを使用してバインディングを削除します。

```
/subsystem=naming/binding=java\:global\/simple-integer-binding:remove
```

バインディングオブジェクトファクトリー

object-factory XML 設定要素は **javax.naming.spi.ObjectFactory** エントリーをバインドします。

- **name** 属性は必須で、エントリーのターゲット JNDI 名を指定します。
- **class** 属性は必須で、オブジェクトファクトリーの Java タイプを定義します。
- **module** 属性は必須で、オブジェクトファクトリーの Java クラスをロードできる JBoss Module ID を指定します。
- 任意の **environment** 子要素は、カスタム環境をオブジェクトファクトリーに提供するために使用できます。

以下に、オブジェクトファクトリーバインディングを作成する管理 CLI コマンドの例を示します。

```
/subsystem=naming/binding=java\:global\/foo\/bar\/factory:add(binding-type=object-factory,
module=org.foo.bar, class=org.foo.bar.ObjectFactory, environment=[p1=v1, p2=v2])
```

結果の XML 設定

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
    <bindings>
        <object-factory name="java:global/foo/bar/factory" module="org.foo.bar"
class="org.foo.bar.ObjectFactory">
            <environment>
                <property name="p1" value="v1" />
                <property name="p2" value="v2" />
            </environment>
```

```
      </object-factory>
    </bindings>
  </subsystem>
```

以下のコマンドを使用してバインディングを削除します。

```
/subsystem=naming/binding=java\:global\/foo\/bar\/factory:remove
```

外部コンテンツのバインド

LDAP コンテキストなどの外部 JNDI コンテキストのフェデレーションは、**external-context** XML 設定要素を使用して実行されます。

- **name** 属性は必須で、エントリーのターゲット JNDI 名を指定します。
- **class** 属性は必須で、フェデレートされたコンテキストの作成に使用される Java 初期ネーミングコンテキストタイプを示します。このようなタイプには、単一の環境マップ引数を持つコンストラクターが必要なことに注意してください。
- 任意の **module** 属性は、外部 JNDI コンテキストが必要とするすべてのクラスをロードできる JBoss Module ID を指定します。
- オプションの **cache** 属性は外部コンテキストインスタンスをキャッシュする必要があるかどうかを示し、デフォルトは **false** になります。
- 任意の **environment** 子要素は、外部コンテキストを検索するために必要なカスタム環境を提供するために使用されます。

以下に、外部コンテキストバインディングを作成する管理 CLI コマンドの例を示します。

```
/subsystem=naming/binding=java\:global\/federation\/ldap\/example:add(binding-type=external-
context, cache=true, class=javax.naming.directory.InitialDirContext, module=org.jboss.as.naming,
environment=[java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url="ldap://ldap.example.com:389", java.naming.security.authentication=simple,
java.naming.security.principal="uid=admin,ou=system", java.naming.security.credentials=secret])
```

結果の XML 設定

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
  <bindings>
    <external-context name="java:global/federation/ldap/example" module="org.jboss.as.naming"
class="javax.naming.directory.InitialDirContext" cache="true">
      <environment>
        <property name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
        <property name="java.naming.provider.url" value="ldap://ldap.example.com:389"/>
        <property name="java.naming.security.authentication" value="simple"/>
        <property name="java.naming.security.principal" value="uid=admin,ou=system"/>
        <property name="java.naming.security.credentials" value="secret"/>
      </environment>
    </external-context>
  </bindings>
</subsystem>
```

以下のコマンドを使用してバインディングを削除します。

```
/subsystem=naming/binding=java\:global\/federation\/ldap\/example:remove
```

注記

JNDI プロバイダーのリソースルックアップが **lookup(Name)** メソッドを適切に実装しないと、javax.naming.InvalidNameException: Only support CompoundName names エラーが発生することがあります。

以下のプロパティーを追加せずに、外部コンテキスト環境が **lookup(String)** メソッドを使用するよう指定すると、この問題を回避できる可能性がありますが、パフォーマンスが劣化します。

```
<property name="org.jboss.as.naming.lookup.by.string" value="true"/>
```

ルックアップエイリアスのバインド
lookup 要素を使用すると、既存のエントリーを追加の名前またはエイリアスにバインドできます。

- **name** 属性は必須で、エントリーのターゲット JNDI 名を指定します。
- **lookup** 属性は必須で、ソース JNDI 名を示します。

以下に、既存のエントリーをエイリアスにバインドする管理 CLI コマンドの例を示します。

```
/subsystem=naming/binding=java\:global\/new-alias-name:add(binding-type=lookup,
lookup=java\:global\/original-name)
```

結果の XML 設定

```
<lookup name="java:global/new-alias-name" lookup="java:global/original-name" />
```

以下のコマンドを使用してバインディングを削除します。

```
/subsystem=naming/binding=java\:global\/c:remove
```

23.3. JNDI バンディングの動的な変更

JBoss EAO 7.1 には、サーバーのリロードや再起動を強制せずに JNDI バインディングを動的に変更する機能が追加されました。この機能は、バージョンの更新、テストの要件、またはアプリケーション機能の更新によってネットワークサービスのエンドポイントが動的に設定される場合に便利です。

JNDI バインディングを更新するには、**rebind** 操作を使用します。**rebind** 操作は **add** 操作と同じ引数を取ります。このコマンドは、**external-context** バンディングタイプ以下のすべてのバインディングタイプで動作します。外部コンテキストバインディングは、モジュラーサービスコンテナー (MSC) の状態に影響する追加の依存関係を必要とするため、サービスを再起動せずに外部コンテキストバインディングを再起動することはできません。

以下のコマンドは、シンプルバインディングの設定 の例で定義した JNDI バインディングを動的に変更します。

```
/subsystem=naming/binding=java\:global\/simple-integer-binding:rebind(binding-type=simple,
type=int, value=200)
```

naming サブシステムでグローバルバインディングを設定する方法に関する詳細は、グローバルバインディングの設定 を参照してください。

23.4. リモート JNDI インターフェイスの設定

リモート JNDI インターフェイスは、クライアントがリモート JBoss EAP インスタンスでエントリーをルックアップできるようにします。**naming** サブシステムを設定すると、このインターフェイスをアクティブまたは非アクティブにすることができます (デフォルトではアクティブになります)。リモート JNDI インターフェイスは **<remote-naming>** 要素を使用して設定されます。

以下の管理 CLI コマンドを使用して、リモート JNDI インターフェイスをアクティブまたは非アクティブにします。

```
/subsystem=naming/service=remote-naming:add
```

以下の管理 CLI コマンドを使用して、リモート JNDI インターフェイスを非アクティブにします。

```
/subsystem=naming/service=remote-naming:remove
```

注記

リモート JNDI 上では **java:jboss/exported** コンテキスト内のエントリーのみにアクセスできます。

第24章 高可用性の設定

24.1. 高可用性

JBoss EAP はデプロイされた Jakarta EE アプリケーションの可用性を保証するために以下の高可用性サービスを提供します。

ロードバランシング

複数のサーバーにワークロードを分散し、サービスが大量のリクエストを処理できるようにします。リクエストが大量に発生しても、クライアントはサービスからタイムリーに応答を受け取ることができます。

フェイルオーバー

ハードウェアやネットワークの障害が発生してもクライアントのサービスへのアクセスが中断しないようにします。サービスに障害が発生すると、別のクラスターメンバーがクライアントのリクエストを引き継ぎ、処理が続行されます。

クラスタリングはこれらすべての機能を包括する言葉です。クラスターのメンバーを設定すると、ロードバランシングと呼ばれるワークロードの共有や、フェイルオーバーと呼ばれる他のクラスターメンバーの障害時におけるクライアント処理の引き継ぎを行うことができます。

注記

選択した JBoss EAP の操作モード (スタンドアロンサーバーまたはマネージドドメイン) によってサーバーの管理方法が異なることに注意してください。操作モードに関係なく JBoss EAP で高可用性サービスを設定できます。

JBoss EAP は、さまざまなコンポーネントを使用した異なるレベルの高可用性をサポートします。高可用性を実現できるランタイムのコンポーネントおよびアプリケーションの一部は以下のとおりです。

- アプリケーションサーバーのインスタンス
- 内部 JBoss Web Server、Apache HTTP Server、Microsoft IIS、または Oracle iPlanet Web Server と併用される Web アプリケーション
- ステートフルおよびステートレスセッション Jakarta Enterprise Beans
- シングルサインオン (SSO) メカニズム
- HTTP セッション
- Jakarta Messaging サービスおよびメッセージ駆動型 Bean (MDB)
- シングルトン MSC サービス
- シングルトンデプロイメント

JBoss EAP では、**jgroups**、**infinispan**、および **modcluster** サブシステムによってクラスタリングが使用できるようになります。**ha** および **full-ha** プロファイルではこれらのシステム有効になっています。JBoss EAP では、これらのサービスは必要に応じて起動およびシャットダウンしますが、分散可能と設定されたアプリケーションがサーバーにデプロイされた場合のみ起動します。

アプリケーションを分散可能とする 方法は、JBoss EAP**Development Guide**を参照してください。

24.2. JGROUPS を用いたクラスター通信

24.2.1. JGroups

JGroups は信頼できるメッセージングのためのツールキットで、お互いにメッセージを送信するノードを持つクラスターを作成するために使用できます。

jgroups サブシステムは JBoss EAP で高可用性サービスのグループ通信サポートを提供します。これにより、名前付きのチャネルおよびプロトコルスタックを設定でき、チャネルのランタイム統計を表示することもできます。**jgroups** サブシステムは高可用性の機能を提供する設定を使用する場合に使用できます (マネージドドメインでは **ha** や **full-ha** プロファイル、スタンドアロンサーバーは **standalone-ha.xml** や **standalone-full-ha.xml** 設定ファイルなど)。

JBoss EAP には 2 つの JGroups スタックが事前に設定されています。

udp
: クラスターのノードは UDP (User Datagram Protocol) マルチキャストを使用してお互いに通信します。これはデフォルトのスタックです。

tcp
: クラスターのノードは TCP (Transmission Control Protocol) を使用してお互いに通信します。

注記

TCP はエラーのチェック、パケットの順番、および輻輳制御を処理するため、TCP のオーバーヘッドは UDP よりも大きく、速度も遅くなると考えられます。JGroups は UDP のこれらの機能を処理しますが、TCP はこれらの機能を独自で処理します。信頼できないネットワークや輻輳度の高いネットワークで JGroups を使用する場合やマルチキャストが使用できない場合は、TCP を選択するとよいでしょう。

事前設定されたスタックを使用できますが、システムの要件に合うように独自に定義をすることもできます。使用できるプロトコルとそれらの属性については、以下の項を参照してください。

- JGroups サブシステムの属性
- JGroups Protocols

24.2.2. デフォルトの JGroups チャネルが TCP を使用するよう設定

デフォルトでは、クラスターノードは **ee** JGroups チャネルに設定された **udp** プロトコルスタックを使用して通信します。

```
<channels default="ee">
  <channel name="ee" stack="udp"/>
</channels>
<stacks>
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp"/>
    <protocol type="PING"/>
    ...
  </stack>
  <stack name="tcp">
    <transport type="TCP" socket-binding="jgroups-tcp"/>
    <protocol type="MPING" socket-binding="jgroups-mping"/>
```

```
    ...
  </stack>
</stacks>
```

マルチキャストを使用した TCP の設定

一部のネットワークでは TCP のみを使用できます。以下の管理 CLI コマンドを使用して、**ee** チャネルが事前設定された **tcp** スタックを使用するようにします。

```
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=tcp)
```

このデフォルトの **tcp** スタックは、IP マルチキャストを使用して初期クラスターメンバーシップを検出する **MPING** プロトコルを使用します。

マルチキャストを使用しない TCP の設定

マルチキャストが好ましくない場合、またはセキュリティーポリシーで許可されていない場合は、TCP を使用するようにデフォルトのプロトコルスタックを変更できます。マルチキャストなしで TCP ベースのクラスタリングを設定するには、以下の手順を実行します。

1. 以下のコマンドを実行して、**ee** チャネルが JGroups サブシステムで事前設定された **tcp** スタックを使用するように切り替えます。

```
<channel name="ee" stack="tcp" cluster="ejb"/>
```

2. クラスターノードの名前を設定します。
   - スタンドアロン設定モードでは、次のいずれかの手順を実行します。
     - 以下のコマンドを実行します。

```
<server xmlns="urn:jboss:domain:8.0" name="node_1">
```

     - インスタンスの起動時にシステムプロパティー **jboss.node.name** に一意の名前を指定します。
   - ドメインモードでは、クラスターサーバーはサーバーのタグの **host-*.xml** ファイルに一覧表示されます。デフォルト設定では、以下のサーバー名を指定します。このサーバー名は、必要に応じて編集できます。

```
<servers>
   <server name="server-one" group="main-server-group"/>
   <server name="server-two" group="other-server-group">
      <socket-bindings port-offset="150"/>
   </server>
</servers>
```

3. 他のクラスターメンバーを検出するには、以下のいずれかのプロトコルを選択します。
   - **TCPGOSSIP**: このプロトコルは、外部のゴシップルーターサービスを使用してクラスターのメンバーを検出します。これには、追加のプロセスの設定および管理が必要ですが、個々の EAP インスタンスが相互のクラスターメンバーを一覧表示しないようにすることができます。このプロトコルは、クラスターメンバーが頻繁に変更される場合に役立ちます。詳細は TCPPING を参照してください。
   - **TCPPING**: このプロトコルは静的クラスターメンバーシップリストを定義し、各ノードが

潜在的なクラスターメンバーの一覧を表示する必要があります。このプロトコルは、クラスターメンバーアドレスが認識されており、頻繁に変更されない場合に推奨されます。詳細は、TCPGOSSIP を参照してください。

24.2.3. TCPPING の設定

この手順は **TCPPING** プロトコルを使用する新しい JGroups スタックを作成し、静的クラスターメンバーシップのリストを定義します。ベーススクリプトは、**tcpping** スタックを作成し、この新しいスタックを使用するようデフォルトの **ee** チャネルを設定します。このスクリプトの管理 CLI コマンドは環境に合わせてカスタマイズする必要があり、バッチで処理されます。

1. 以下のスクリプトをテキストエディターにコピーし、ローカルファイルシステムに保存します。

```
# Define the socket bindings
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=jgroups-host-a:add(host=HOST_A,port=7600)
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=jgroups-host-b:add(host=HOST_B,port=7600)
batch
# Add the tcpping stack
/subsystem=jgroups/stack=tcpping:add
/subsystem=jgroups/stack=tcpping/transport=TCP:add(socket-binding=jgroups-tcp)
/subsystem=jgroups/stack=tcpping/protocol=TCPPING:add(socket-bindings=[jgroups-host-a,jgroups-host-b])
/subsystem=jgroups/stack=tcpping/protocol=MERGE3:add
/subsystem=jgroups/stack=tcpping/protocol=FD_SOCK:add
/subsystem=jgroups/stack=tcpping/protocol=FD_ALL:add
/subsystem=jgroups/stack=tcpping/protocol=VERIFY_SUSPECT:add
/subsystem=jgroups/stack=tcpping/protocol=pbcast.NAKACK2:add
/subsystem=jgroups/stack=tcpping/protocol=UNICAST3:add
/subsystem=jgroups/stack=tcpping/protocol=pbcast.STABLE:add
/subsystem=jgroups/stack=tcpping/protocol=pbcast.GMS:add
/subsystem=jgroups/stack=tcpping/protocol=MFC:add
/subsystem=jgroups/stack=tcpping/protocol=FRAG3:add
# Set tcpping as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=tcpping)
run-batch
reload
```

定義されたプロトコルの順番が重要になることに注意してください。また、**add-index** の値を **add** コマンドに渡すと、特定のインデックスでプロトコルを挿入できます。インデックスはゼロベースであるため、以下の管理 CLI コマンドは **UNICAST3** プロトコルを 7 つ目のプロトコルとして追加します。

```
/subsystem=jgroups/stack=tcpping/protocol=UNICAST3:add(add-index=6)
```

2. 環境に合わせてスクリプトを変更します。

- マネージドドメインで実行している場合は、/**subsystem=jgroups** コマンドの前に /**profile=PROFILE_NAME** を追加し、更新するプロファイルを指定する必要があります。
- 以下のプロパティーを環境に合わせて調整します。

- **socket-bindings**: ウェルノウンとして見なされ、最初のメンバーシップの検索に利用できるホストとポートの組み合わせのコンマ区切りリスト。ソケットバインディングの定義に関する詳細は、ソケットバインディングの設定 を参照してください
- **initial_hosts**: ウェルノウンとして見なされ、最初のメンバーシップの検索に使用できる **HOST[PORT]** という構文を使用したホストとポートの組み合わせのコンマ区切りリスト (例: **host1[1000],host2[2000]**)。
- **port_range**: このプロパティーは、 **initial_hosts** ポート範囲を指定した値の分拡張するために使用されます。たとえば、**initial_hosts** を **host1[1000],host2[2000]** に設定し、**port_range** を **1** に設定した場合 **initial_hosts** 設定は **host1[1000],host1[1001],host2[2000],host2[2001]** に拡張されます。このプロパティーは **initial_hosts** プロパティーと併用した場合のみ有効です。

3. スクリプトファイルを管理 CLI に渡してスクリプトを実行します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/SCRIPT_NAME
```

TCPPING スタックが使用できるようになり、ネットワークの通信に TCP が使用されます。

24.2.3.1. スタンドアロンモードでの TCPPING の設定

この手順では、スタンドアロンモードでクラスター化されたアプリケーションの TCP スタックおよびノードを設定するのに役立ちます。

手順

1. JGroups サブシステムで、デフォルトのスタックを **udp** から **tcp** に変更します。

```
<channel name="ee" stack="tcp" cluster="ejb"/>
```

2. デフォルトの MPING プロトコルの代わりに TCPPING プロトコルを使用するように TCP スタックを設定します。以下のコードでは、**initial_hosts** プロパティーはクラスター内の全ノードのリストに相関し、**7600** は設定および環境に応じて異なるデフォルトの **jgroups-tcp** ポートを示します。

```
<stack name="tcp">
  <transport type="TCP" socket-binding="jgroups-tcp"/>
  <protocol type="TCPPING">
    <property name="initial_hosts">192.168.1.5[7600],192.168.1.9[7600]</property>
    <property name="port_range">0</property>
  </protocol>
  <protocol type="MERGE3"/>
  <protocol type="FD_SOCK" socket-binding="jgroups-tcp-fd"/>
  <protocol type="FD_ALL"/>
  <protocol type="VERIFY_SUSPECT"/>
  <protocol type="pbcast.NAKACK2"/>
  <protocol type="UNICAST3"/>
  <protocol type="pbcast.STABLE"/>
  <protocol type="pbcast.GMS"/>
  <protocol type="MFC"/>
  <protocol type="FRAG3"/>
</stack>
```

注記

initial_hosts に設定されたポート番号 **7600** は、**jgroups-tcp** ソケットバインディング定義で定義されたポート番号と同じでなければなりません。ソケットバインディングに port-offset 機能を使用する場合は、**initial_hosts** のオフセットの後に同じ値を指定する必要があります。

3. JGroups コンポーネントによって使用されるプライベートインターフェイスの IP アドレスを設定します。IP アドレスは、 **initial_hosts** で指定されている IP アドレスのいずれかに関連付ける必要があります。

```
<interface name="private">
  <inet-address value="${jboss.bind.address.private:192.168.1.5}"/>
</interface>
```

4. クラスター内の他のノードを設定するには、上記の手順を繰り返します。ノードが設定されたら、各ノードを起動して、クラスター化されたアプリケーションをデプロイします。

検証

- ログを確認して、ノードが稼働しているかどうかを確認できます。

```
INFO  [org.infinispan.remoting.transport.jgroups.JGroupsTransport] (thread-2,ee,node_1)
ISPN000094: Received new cluster view for channel server: [node_1|1] (2) [node_1, node_2]
INFO  [org.infinispan.remoting.transport.jgroups.JGroupsTransport] (thread-2,ee,node_1)
ISPN000094: Received new cluster view for channel web: [node_1|1] (2) [node_1, node_2]
```

24.2.3.2. ドメインモードでの TCPPING の設定

この手順では、ドメインモードでクラスター化されたアプリケーションの TCP スタックおよびノードを設定するのに役立ちます。

手順

1. 複数のクラスターに同じプロファイルを使用する場合は、システムプロパティーの値を **initial_hosts** に設定します。

```
<protocol type="TCPPING">
  <property name="initial_hosts">${jboss.cluster.tcp.initial_hosts}</property>

Set the system property at the `server-group` level:
<server-groups>
  <server-group name="a-server-group" profile="ha">
  <socket-binding-group ref="ha-sockets"/>
  <system-properties>
    <property name="jboss.cluster.tcp.initial_hosts"
value="192.168.1.5[7600],192.168.1.9[7600]" />
  </system-properties>
```

2. ホストコントローラーの XML 設定内でプライベートインターフェイスの IP アドレスを設定します。プライベートインターフェイスの IP アドレスは、 **initial_hosts** で指定されている IP アドレスのいずれかに関連付ける必要があります。

```
<interfaces>
```

```
    ....
  <interface name="private">
    <inet-address value="${jboss.bind.address.private:192.168.1.5}"/>
  </interface>
</interfaces>
```

検証

- ログを確認して、ノードが稼働しているかどうかを確認できます。

```
INFO  [org.infinispan.remoting.transport.jgroups.JGroupsTransport] (thread-2,ee,node_1)
ISPN000094: Received new cluster view for channel server: [node_1|1] (2) [node_1, node_2]
INFO  [org.infinispan.remoting.transport.jgroups.JGroupsTransport] (thread-2,ee,node_1)
ISPN000094: Received new cluster view for channel web: [node_1|1] (2) [node_1, node_2]
```

24.2.4. TCPGOSSIP の設定

この手順は、**TCPGOSSIP** プロトコルを使用する新しい JGroups スタックを作成し、外部ゴシップルーターを使用してクラスターのメンバーを検索します。ベーススクリプトは、**tcpgossip** スタックを作成し、この新しいスタックを使用するようデフォルトの **ee** チャネルを設定します。このスクリプトの管理 CLI コマンドは環境に合わせてカスタマイズする必要があり、バッチで処理されます。

1. 以下のスクリプトをテキストエディターにコピーし、ローカルファイルシステムに保存します。

```
# Define the socket bindings
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=jgroups-host-a:add(host=HOST_A,port=13001)
batch
# Add the tcpgossip stack
/subsystem=jgroups/stack=tcpgossip:add
/subsystem=jgroups/stack=tcpgossip/transport=TCP:add(socket-binding=jgroups-tcp)
/subsystem=jgroups/stack=tcpgossip/protocol=TCPGOSSIP:add(socket-bindings=[jgroups-host-a])
/subsystem=jgroups/stack=tcpgossip/protocol=MERGE3:add
/subsystem=jgroups/stack=tcpgossip/protocol=FD_SOCK:add
/subsystem=jgroups/stack=tcpgossip/protocol=FD_ALL:add
/subsystem=jgroups/stack=tcpgossip/protocol=VERIFY_SUSPECT:add
/subsystem=jgroups/stack=tcpgossip/protocol=pbcast.NAKACK2:add
/subsystem=jgroups/stack=tcpgossip/protocol=UNICAST3:add
/subsystem=jgroups/stack=tcpgossip/protocol=pbcast.STABLE:add
/subsystem=jgroups/stack=tcpgossip/protocol=pbcast.GMS:add
/subsystem=jgroups/stack=tcpgossip/protocol=MFC:add
/subsystem=jgroups/stack=tcpgossip/protocol=FRAG3:add
# Set tcpgossip as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=tcpgossip)
run-batch
reload
```

定義されたプロトコルの順番が重要になることに注意してください。また、**add-index** の値を **add** コマンドに渡すと、特定のインデックスでプロトコルを挿入できます。インデックスはゼロベースであるため、以下の管理 CLI コマンドは **UNICAST3** プロトコルを 7 つ目のプロトコルとして追加します。

```
/subsystem=jgroups/stack=tcpgossip/protocol=UNICAST3:add(add-index=6)
```

2. 環境に合わせてスクリプトを変更します。

   - マネージドドメインで実行している場合は、**/subsystem=jgroups** コマンドの前に **/profile=PROFILE_NAME** を追加し、更新するプロファイルを指定する必要があります。
   - 以下のプロパティーを環境に合わせて調整します。
     - **socket-bindings**: ウェルノウンとして見なされ、最初のメンバーシップの検索に利用できるホストとポートの組み合わせのコンマ区切りリスト。ソケットバインディングの定義に関する詳細は、ソケットバインディングの設定 を参照してください
     - **initial_hosts**: ウェルノウンとして見なされ、最初のメンバーシップの検索に使用できる **HOST[PORT]** という構文を使用したホストとポートの組み合わせのコンマ区切りリスト (例: **host1[1000],host2[2000]**)。
     - **port_range**: このプロパティーは、 **initial_hosts** ポート範囲を指定した値の分拡張するために使用されます。たとえば、**initial_hosts** を **host1[1000],host2[2000]** に設定し、**port_range** を **1** に設定した場合 **initial_hosts** 設定は **host1[1000],host1[1001],host2[2000],host2[2001]** に拡張されます。このプロパティーは **initial_hosts** プロパティーと併用した場合のみ有効です。
     - **reconnect_interval**: 接続が切断されたスタブがゴシップルーターへの再接続を試みる間隔 (ミリ秒単位)。
     - **sock_conn_timeout**: ソケット作成の最大時間。デフォルトは **1000** ミリ秒です。
     - **sock_read_timeout**: 読み取りがブロックされる最大時間 (ミリ秒単位)。**0** を値として指定すると無制限にブロックされます。

3. スクリプトファイルを管理 CLI に渡してスクリプトを実行します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/SCRIPT_NAME
```

TCPGOSSIP スタックが使用できるようになり、ネットワークの通信に TCP が使用されます。このスタックはゴシップルーターと使用するように設定されるため、JGroups メンバーは他のクラスターメンバーを見つけることができます。

24.2.5. JDBC_PING の設定

JDBC_PING プロトコルを使用して、クラスターでメンバーシップを管理および検出できます。

JDBC_PING はデータソースに指定されたデータベースを使用して、クラスターのメンバーをリスト表示します。

1. クラスターメンバーシップの管理に使用するデータベースに接続するデータソースを作成します。
2. 以下のスクリプトをテキストエディターにコピーし、ローカルファイルシステムに保存します。

```
batch
# Add the JDBC_PING stack
/subsystem=jgroups/stack=JDBC_PING:add
/subsystem=jgroups/stack=JDBC_PING/transport=TCP:add(socket-binding=jgroups-tcp)
```

```
/subsystem=jgroups/stack=JDBC_PING/protocol=JDBC_PING:add(data-
source=ExampleDS)
/subsystem=jgroups/stack=JDBC_PING/protocol=MERGE3:add
/subsystem=jgroups/stack=JDBC_PING/protocol=FD_SOCK:add
/subsystem=jgroups/stack=JDBC_PING/protocol=FD_ALL:add
/subsystem=jgroups/stack=JDBC_PING/protocol=VERIFY_SUSPECT:add
/subsystem=jgroups/stack=JDBC_PING/protocol=pbcast.NAKACK2:add
/subsystem=jgroups/stack=JDBC_PING/protocol=UNICAST3:add
/subsystem=jgroups/stack=JDBC_PING/protocol=pbcast.STABLE:add
/subsystem=jgroups/stack=JDBC_PING/protocol=pbcast.GMS:add
/subsystem=jgroups/stack=JDBC_PING/protocol=MFC:add
/subsystem=jgroups/stack=JDBC_PING/protocol=FRAG3:add
# Set JDBC_PING as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=JDBC_PING)
run-batch
reload
```

定義されたプロトコルの順番が重要になることに注意してください。また、**add-index** の値を **add** コマンドに渡すと、特定のインデックスでプロトコルを挿入できます。インデックスはゼロベースであるため、以下の管理 CLI コマンドは **UNICAST3** プロトコルを 7 つ目のプロトコルとして追加します。

```
/subsystem=jgroups/stack=JDBC_PING/protocol=UNICAST3:add(add-index=6)
```

3. 環境に合わせてスクリプトを変更します。
   - マネージドドメインで実行している場合は、**/subsystem=jgroups** コマンドの前に **/profile=PROFILE_NAME** を追加し、更新するプロファイルを指定する必要があります。
   - **ExampleDS**を、手順 1 で定義したデータソースの名前に置き換えます。
4. スクリプトファイルを管理 CLI に渡してスクリプトを実行します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/SCRIPT_NAME
```

JDBC_PING スタックが使用できるようになり、ネットワークの通信に TCP が使用されます。

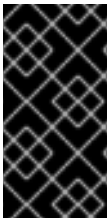
関連情報
JDBC_PING: https://developer.jboss.org/wiki/JDBCPING

データソースの作成: JBoss EAP データソース

24.2.6. JGroups のネットワークインターフェイスへのバインディング

デフォルトでは、JGroups は **private** ネットワークインターフェイスのみへバインドし、デフォルト設定でローカルホストへ示されます。クラスタリングのトラフィックはパブリックネットワークインターフェイスで公開するべきではないため、セキュリティー上の理由で JGroups は JBoss EAP の起動中に指定された **-b** 引数によって定義されたネットワークインターフェイスにはバインドしません。

ネットワークインターフェイスの設定方法については ネットワークおよびポート設定 の章を参照してください。



重要

セキュリティー上の理由で、JGroups はパブリックではないネットワークインターフェイスのみにバインドされる必要があります。また、パフォーマンス上の理由で JGroups トラフィックのネットワークインターフェイスは専用の VLAN (Virtual Local Area Network) の一部にすることが推奨されます。

24.2.7. クラスターのセキュア化

クラスターをセキュアに実行するには、複数の課題に対応する必要があります。

- 承認されていないノードがクラスターに参加しないようにします。これは 認証 を要求して対処します。
- クラスターメンバーがクラスターメンバー以外と通信しないようにします。これは メッセージの暗号化 で対処します。

24.2.7.1. 認証の設定

JGroups の認証は **AUTH** プロトコルによって実行されます。認証されたノードのみがクラスターに参加できるようにすることが目的です。

該当のサーバー設定ファイルに **AUTH** プロトコルと適切なプロパティー設定を追加します。**AUTH** プロトコルは **pbcast.GMS** プロトコルの直前に設定する必要があります。

以下は、**AUTH** を異なる承認トークンと使用する例になります。

AUTH とシンプルトークン

```
...
<protocol type="pbcast.STABLE"/>
<auth-protocol type="AUTH">
  <plain-token>
    <shared-secret-reference clear-text="my_password"/>
  </plain-token>
</auth-protocol>
<protocol type="pbcast.GMS"/>
...
```

AUTH とダイジェストアルゴリズムトークン
この形式は、MD5 や SHA-2 など、どのダイジェストアルゴリズムでも使用できます。JBoss EAP 7.4 でのデフォルトのダイジェストアルゴリズムは SHA-256 で、これは JVM によるサポートに必要な最強のダイジェストアルゴリズムです。多くの JVM は、追加で SHA-512 を実装します。

```
...
<protocol type="pbcast.STABLE"/>
<auth-protocol type="AUTH">
  <digest-token algorithm="SHA-512">
    <shared-secret-reference clear-text="my_password"/>
  </digest-token>
</auth-protocol>
<protocol type="pbcast.GMS"/>
...
```

AUTH と X509 トークン

この例は、**elytron** サブシステムで新しいキーストアを作成し、JGroups の **AUTH** 設定で参照します。

1. キーストアを作成します。

```
$ keytool -genkeypair -alias jgroups_key -keypass my_password -storepass my_password -
storetype jks -keystore jgroups.keystore -keyalg RSA
```

2. 管理 CLI を使用して、キーストアを **elytron** サブシステムに追加します。

```
/subsystem=elytron/key-store=jgroups-token-
store:add(type=jks,path=/path/to/jgroups.keystore,credential-reference={clear-
text=my_password}, required=true)
```

3. JGroups のスタック定義で **AUTH** を設定してキーストアを使用するようにします。

```
...
<protocol type="pbcast.STABLE"/>
<auth-protocol type="AUTH">
  <cipher-token algorithm="RSA" key-alias="jgroups_key" key-store="jgroups-token-store">
    <shared-secret-reference clear-text="my_password"/>
    <key-credential-reference clear-text="my_password"/>
  </cipher-token>
</auth-protocol>
<protocol type="pbcast.GMS"/>
...
```

24.2.7.2. 暗号化の設定

JGroups はクラスターのメンバーが共有する秘密鍵を使用してメッセージを暗号化します。送信元は共有する秘密鍵を使用してメッセージを暗号化し、受信先は同じ秘密鍵を使用してメッセージを復号化します。対称暗号化 は **SYM_ENCRYPT** プロトコルを使用して設定され、ノードは共有のキーストアを使用して秘密鍵を取得します。非対称暗号化 は **ASYM_ENCRYPT** プロトコルを使用して設定され、ノードは **AUTH** を使用して認証された後にクラスターのコーディネーターから秘密鍵を取得します。

対称暗号化の使用
SYM_ENCRYPT を使用するには、各ノードの JGroups 設定で参照されるキーストアを設定する必要があります。

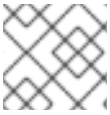
1. キーストアを作成します。
   以下のコマンドでは、**VERSION** を適切な JGroups JAR バージョンに置き換え、 **PASSWORD** をキーストアパスワードに置き換えます。

```
$ java -cp EAP_HOME/modules/system/layers/base/org/jgroups/main/jgroups-VERSION.jar
org.jgroups.demos.KeyStoreGenerator --alg AES --size 128 --storeName
defaultStore.keystore --storepass PASSWORD --alias mykey
```

   これにより、JGroups 設定で参照される **defaultStore.keystore** ファイルが生成されます。

2. キーストアが生成されたら、2 つの方法の 1 つを使用して **SYM_PROTOCOL** で定義されます。

   - キーストアを 直接設定に 定義します。
   - Elytron サブシステムを使用 してキーストアを参照します。



注記

SYM_ENCRYPT を使用する場合、**AUTH** の設定は任意です。

キーストアを直接参照して対称暗号化を使用

1. **jgroups** サブシステムで **SYM_ENCRYPT** プロトコルを設定します。
該当するサーバー設定ファイルに、**SYM_ENCRYPT** プロトコルと適切なプロパティー設定を追加します。このプロトコルは、以前作成されたキーストアを参照します。**SYM_ENCRYPT** プロトコルは **pbcast.NAKACK2** の直前に設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:jgroups:6.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <protocol type="SYM_ENCRYPT">
        <property name="provider">SunJCE</property>
        <property name="sym_algorithm">AES</property>
        <property name="encrypt_entire_message">true</property>
        <property name="keystore_name">/path/to/defaultStore.keystore</property>
        <property name="store_password">PASSWORD</property>
        <property name="alias">mykey</property>
      </protocol>
      <protocol type="pbcast.NAKACK2"/>
      <protocol type="UNICAST3"/>
      <protocol type="pbcast.STABLE"/>
      <protocol type="pbcast.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG3"/>
    </stack>
  </stacks>
</subsystem>
```

Elytron と対称暗号化の使用

1. 管理 CLI を使用して、対称暗号化を使用して 作成された **defaultStore.keystore** を参照するキーストアを elytron サブシステムに作成します。

```
/subsystem=elytron/key-store=jgroups-
keystore:add(path=/path/to/defaultStore.keystore,credential-reference={clear-
text=PASSWORD},type=JCEKS)
```

2. **SYM_ENCRYPT** プロトコルと適切なプロパティー設定を **jgroups** サブシステムに追加します。以下の設定どおり、**SYM_ENCRYPT** プロトコルは **pbcast.NAKACK2** プロトコルの直前に設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:jgroups:6.0">
  <stacks>
    <stack name="udp">
```

```
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <encrypt-protocol type="SYM_ENCRYPT" key-alias="mykey" key-store="jgroups-
keystore">
        <key-credential-reference clear-text="PASSWORD"/>
        <property name="provider">SunJCE</property>
        <property name="encrypt_entire_message">true</property>
      </encrypt-protocol>
      <protocol type="pbcast.NAKACK2"/>
      <protocol type="UNICAST3"/>
      <protocol type="pbcast.STABLE"/>
      <protocol type="pbcast.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG3"/>
    </stack>
  </stacks>
</subsystem>
```

注記

上記の例はクリアテキストのパスワードを使用しますが、認証情報ストアを定義して設定ファイル外部にパスワードを定義することもできます。認証情報ストアの設定に関する詳細は、**How to Configure Server Security**の Credential Store を参照してください。

非対称暗号化の使用
ASYM_ENCRYPT を使用するには **AUTH** プロトコルを定義する必要があります。AUTH プロトコルを **jgroups** サブシステムで設定する手順は、認証の設定 を参照してください。

ASYM_ENCRYPT は 2 つの方法の 1 つで設定されます。

- 秘密鍵を生成し、設定で直接 参照します。
- キーストアを作成し、Elytron サブシステムを使用して 参照します。

秘密鍵を生成して非対象暗号化を使用

1. **jgroups** サブシステムで **ASYM_ENCRYPT** プロトコルを設定します。
   該当のサーバー設定ファイルに **ASYM_ENCRYPT** プロトコルと適切なプロパティー設定を追加します。以下の設定どおり、**ASYM_ENCRYPT** プロトコルは **pbcast.NAKACK2** プロトコルの直前に設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:jgroups:6.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK"/>
      <protocol type="FD_ALL"/>
```

```
      <protocol type="VERIFY_SUSPECT"/>
      <protocol type="ASYM_ENCRYPT">
        <property name="encrypt_entire_message">true</property>
        <property name="sym_keylength">128</property>
        <property name="sym_algorithm">AES/ECB/PKCS5Padding</property>
        <property name="asym_keylength">512</property>
        <property name="asym_algorithm">RSA</property>
      </protocol>
      <protocol type="pbcast.NAKACK2"/>
      <protocol type="UNICAST3"/>
      <protocol type="pbcast.STABLE"/>
      <!-- Configure AUTH protocol here -->
      <protocol type="pbcast.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG3"/>
    </stack>
  </stacks>
</subsystem>
```

Elytron と非対称暗号化の使用

1. キーペアが含まれるキーストアを作成します。以下のコマンドは、**mykey** というエイリアスでキーストアを作成します。

```
$ keytool -genkeypair -alias mykey -keyalg RSA -keysize 1024 -keystore
defaultKeystore.keystore -dname "CN=localhost" -keypass secret -storepass secret
```

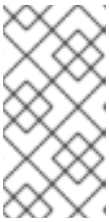
2. 管理 CLI を使用して、**defaultStore.keystore** を参照するキーストアを **elytron** サブシステムに作成します。

```
/subsystem=elytron/key-store=jgroups-
keystore:add(path=/path/to/defaultStore.keystore,credential-reference={clear-
text=PASSWORD},type=JCEKS)
```

3. **ASYM_ENCRYPT** プロトコルと適切なプロパティー設定を **jgroups** サブシステムに追加します。以下の設定どおり、**ASYM_ENCRYPT** プロトコルは **pbcast.NAKACK2** プロトコルの直前に設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:jgroups:6.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <encrypt-protocol type="ASYM_ENCRYPT" key-alias="mykey" key-store="jgroups-
keystore">
        <key-credential-reference clear-text="secret" />
        <property name="encrypt_entire_message">true</property>
      </encrypt-protocol>
      <protocol type="pbcast.NAKACK2"/>
      <protocol type="UNICAST3"/>
```

```
      <protocol type="pbcast.STABLE"/>
      <!-- Configure AUTH protocol here -->
      <protocol type="pbcast.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG3"/>
    </stack>
  </stacks>
</subsystem>
```



注記

上記の例はクリアテキストのパスワードを使用しますが、認証情報ストアを定義して設定ファイル外部にパスワードを定義することもできます。認証情報ストアの設定に関する詳細は、**How to Configure Server Security**の Credential Store を参照してください。

24.2.8. JGroups スレッドプールの設定

jgroups サブシステムには **default**、**internal**、**oob**、および **timer** スレッドプールが含まれます。これらのプールはすべての JGroups スタックに対して設定でき、ローカルノードに設定された Bean やその他のプールには影響しません。JGroup スレッドプールは、クラスター通信をサポートするために使用されます。

以下の表は、各スレッドプールに設定できる属性とデフォルト値を表しています。

スレッドプール名	説明	keepalive-time	max-threads	min-threads	queue-length
default	このプールは、帯域外としてマークされていない受信メッセージを処理するために使用されます。	60000L	300	20	100
internal	このプールは、EAP のメンテナンスに必要な内部プロセスを処理するために使用されます。	60000L	4	2	100

スレッドプール名	説明	keepalive-time	max-threads	min-threads	queue-length
oob	このプールは、受信 OOB (out of band) メッセージを処理する場合に使用されます。	60000L	300	20	0
timer	このプールは、タイムバウンドスケジューラーメッセージを処理するために使用されます。	5000L	4	2	500

以下の構文を使用して、管理 CLI で JGroups スレッドプールを設定します。

```
/subsystem=jgroups/stack=STACK_TYPE/transport=TRANSPORT_TYPE/thread-pool=THREAD_POOL_NAME:write-attribute(name=ATTRIBUTE_NAME, value=ATTRIBUTE_VALUE)
```

以下は、 **udp** スタックの **default** スレッドプールで **max-threads** の値を **500** に設定する管理 CLI コマンドの例になります。

```
/subsystem=jgroups/stack=udp/transport=UDP/thread-pool=default:write-attribute(name="max-threads", value="500")
```

24.2.9. JGroups の送受信バッファーの設定

バッファーサイズ警告の解決
デフォルトでは、JGroups は特定の送受信バッファー値で設定されています。しかし、可能なバッファーサイズがオペレーティングシステムによって制限され、JBoss EAP が設定されたバッファー値を使用できないことがあります。このような場合、以下と似た警告が JBoss EAP のログに記録されます。

```
WARNING [org.jgroups.protocols.UDP] (ServerService Thread Pool -- 68)
JGRP000015: the send buffer of socket DatagramSocket was set to 640KB, but the OS only allocated 212.99KB.
This might lead to performance problems. Please set your max send buffer in the OS correctly (e.g. net.core.wmem_max on Linux)
WARNING [org.jgroups.protocols.UDP] (ServerService Thread Pool -- 68)
JGRP000015: the receive buffer of socket DatagramSocket was set to 20MB, but the OS only allocated 212.99KB.
This might lead to performance problems. Please set your max receive buffer in the OS correctly (e.g. net.core.rmem_max on Linux)
```

これに対応するには、オペレーティングシステムのマニュアルでバッファーサイズを増やす方法を参照

してください。Red Hat Enterprise Linux システムの場合は、root ユーザーとして **/etc/sysctl.conf** を編集し、システムの再起動後も維持されるバッファーサイズの最大値を設定します。以下に例を示します。

```
# Allow a 25MB UDP receive buffer for JGroups
net.core.rmem_max = 26214400
# Allow a 1MB UDP send buffer for JGroups
net.core.wmem_max = 1048576
```

/etc/sysctl.conf を編集後、**sysctl -p** を実行して変更を反映します。

JGroups バッファーサイズの設定
以下のトランスポートプロパティーを UDP および TCP JGroups スタックに設定すると、JBoss EAP が使用する JGroups バッファーサイズを設定できます。

UDP スタック

- **ucast_recv_buf_size**
- **ucast_send_buf_size**
- **mcast_recv_buf_size**
- **mcast_send_buf_size**

TCP スタック

- **recv_buf_size**
- **send_buf_size**

JGroups バッファーサイズは、管理コンソールまたは管理 CLI を使用して設定できます。

以下の構文を使用して、管理 CLI で JGroups バッファーサイズプロパティーを設定します。

```
/subsystem=jgroups/stack=STACK_NAME/transport=TRANSPORT/property=PROPERTY_NAME:add(value=BUFFER_SIZE)
```

以下は、**tcp** スタックで **recv_buf_size** プロパティーを **20000000** に設定する管理 CLI コマンドの例になります。

```
/subsystem=jgroups/stack=tcp/transport=TRANSPORT/property=recv_buf_size:add(value=20000000)
```

JGroups バッファーサイズは管理コンソールを使用して設定することもできます。 **Configuration** タブで **JGroups** サブシステムを選択し、表示 をクリックして **Stack** タブを選択し、該当するスタックを選びます。 **Transport** をクリックし、 **Properties** フィールドを編集します。

24.2.10. JGroups サブシステムの調整

jgroups サブシステムのパフォーマンスを監視および最適化するための情報は、**Performance Tuning Guide**の JGroups Subsystem Tuning の項を参照してください。

24.2.11. JGroups トラブルシューティング

24.2.11.1. ノードがクラスターを形成しない

マシンで IP マルチキャストが正しくセットアップされていることを確認します。JBoss EAP には、IP マルチキャストのテストに使用できる **McastReceiverTest** と **McastSenderTest** の 2 つのテストプログラムが含まれています。

ターミナルで **McastReceiverTest** を開始します。

```
$ java -cp EAP_HOME/bin/client/jboss-client.jar org.jgroups.tests.McastReceiverTest -mcast_addr
230.11.11.11 -port 5555
```

別のターミナルウインドウで **McastSenderTest** を開始します。

```
$ java -cp EAP_HOME/bin/client/jboss-client.jar org.jgroups.tests.McastSenderTest -mcast_addr
230.11.11.11 -port 5555
```

特定のネットワークインターフェイスカード (NIC) をバインドする場合は、**-bind_addr YOUR_BIND_ADDRESS** を使用します。**YOUR_BIND_ADDRESS** はバインドする NIC の IP アドレスに置き換えます。送信側と受信側の両方にこのパラメーターを使用します。

McastSenderTest ターミナルウインドウで入力すると **McastReceiverTest** ウインドウに出力が表示されます。表示されない場合は以下の手順に従います。

- 送信側のコマンドに **-ttl VALUE** を追加して、マルチキャストパケットの TTL (time-to-live) を増やします。このテストプログラムによって使用されるデフォルトの値は **32** で、**VALUE** は **255** 以下である必要があります。
- マシンに複数のインターフェイスがある場合は、適切なインターフェイスを使用していることを確認します。
- システム管理者に連絡し、マルチキャストが選択したインターフェイスで動作することを確認します。

クラスターの各マシンでマルチキャストが適切に動作したら、送信側と受信側を別々のマシンに配置し、テストを繰り返します。

24.2.11.2. 障害検出での不明なハートビートの原因

ハートビートの確認が一定時間 (**T**) 受信されないと、障害検出 (FD) によってクラスターメンバーが原因として疑われることがあります。T は **timeout** および **max_tries** によって定義されます。

たとえば、ノード A、B、C、および D のクラスターがあり、A が B、B が C、C が D、D が A を ping する場合、以下のいずれかの理由で C が疑われます。

- B または C が CPU の使用率が 100% の状態で **T** 秒よりも長く稼働している場合。この場合、C がハートビート確認を B に送信しても CPU の使用率が 100% であるため B が確認を処理できないことがあります。
- B または C がガベッジコレクションを実行している場合、上記と同じ結果になります。
- 上記 2 件の組み合わせ。
- ネットワークによるパケットの損失が発生する場合。通常、ネットワークに大量のトラフィックがあり、スイッチがパケットを破棄すると発生します (通常は最初にブロードキャスト、次に IP マルチキャスト、そして最後に TCP パケットが破棄されます)。

- B または C がコールバックを処理する場合。C が処理に **T** + 1 秒かかるリモートメソッド呼び出しをチャネル上で受信した場合、C はハートビートを含む他のメッセージを処理できません。そのため、B はハートビート確認を受信せず、C が疑われます。

24.3. INFINISPAN

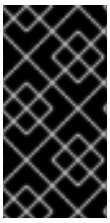
24.3.1. Infinispan

Infinispan は Java データグリッドプラットフォームで、キャッシュされたデータの管理に Jakarta Persistence 2.2 準拠のキャッシュインターフェイスを提供します。

Infinispan の機能や設定オプションの詳細は Infinispan のドキュメント を参照してください。

infinispan サブシステムは JBoss EAP のキャッシュサポートを提供します。名前付きのキャッシュコンテナーやキャッシュのランタイムメトリックスを設定および表示できます。

高可用性の機能を提供する設定を使用する場合 (マネージドドメインでは **ha** や **full-ha** プロファイル、スタンドアロンサーバーは **standalone-ha.xml** や **standalone-full-ha.xml** 設定ファイル)、**infinispan** サブシステムはキャッシング、状態のレプリケーション、および状態分散サポートを提供します。高可用性でない設定では、**infinispan** サブシステムはローカルのキャッシュサポートを提供します。



重要

Infinispan は JBoss EAP 7.4 の公開モジュールです。アプリケーション用の **infinispan** サブシステムを使用して、新しいキャッシュコンテナーまたはキャッシュを作成および使用できます。また、Infinispan API の使用は、アプリケーションによってサポートされています。

24.3.2. キャッシュコンテナー

キャッシュコンテナーはサブシステムによって使用されるキャッシュのリポジトリーです。各キャッシュコンテナーは、使用されるデフォルトのキャッシュを定義します。

JBoss EAP 7 は以下のデフォルト Infinispan キャッシュコンテナーを定義します。

- **server** (シングルトンキャッシング)
- **web** (Web セッションクラスタリング)
- **ejb** (ステートフルセッション Bean クラスタリング)
- **hibernate** (エンティティーキャッシング)

例: Infinispan のデフォルト設定

```
<subsystem xmlns="urn:jboss:domain:infinispan:7.0">
  <cache-container name="server" aliases="singleton cluster" default-cache="default"
module="org.wildfly.clustering.server">
    <transport lock-timeout="60000"/>
    <replicated-cache name="default">
      <transaction mode="BATCH"/>
    </replicated-cache>
  </cache-container>
  <cache-container name="web" default-cache="dist" module="org.wildfly.clustering.web.infinispan">
    <transport lock-timeout="60000"/>
```

```
    <distributed-cache name="dist">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <file-store/>
    </distributed-cache>
  </cache-container>
  <cache-container name="ejb" aliases="sfsb" default-cache="dist"
module="org.wildfly.clustering.ejb.infinispan">
    <transport lock-timeout="60000"/>
    <distributed-cache name="dist">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <file-store/>
    </distributed-cache>
  </cache-container>
  <cache-container name="hibernate" default-cache="local-query" module="org.hibernate.infinispan">
    <transport lock-timeout="60000"/>
    <local-cache name="local-query">
      <object-memory size="1000"/>
      <expiration max-idle="100000"/>
    </local-cache>
    <invalidation-cache name="entity">
      <transaction mode="NON_XA"/>
      <object-memory size="1000"/>
      <expiration max-idle="100000"/>
    </invalidation-cache>
    <replicated-cache name="timestamps" mode="ASYNC"/>
  </cache-container>
</subsystem>
```

各キャッシュコンテナーで定義されたデフォルトのキャッシュに注目してください。この例では、**web** キャッシュコンテナーは **dist** 分散キャッシュをデフォルトとして定義します。そのため、Web セッションのクラスタリングでは **dist** キャッシュが使用されます。

デフォルトキャッシュの変更やキャッシュの追加に関する詳細は、キャッシュコンテナーの設定 を参照してください。

24.3.2.1. キャッシュコンテナーの設定

キャッシュコンテナーやキャッシュ属性は、管理コンソールまたは管理 CLI を使用して設定できます。

警告

設定の他のコンポーネントが参照する可能性があるため、キャッシュまたはキャッシュコンテナーの名前を変更しないでください。

24.3.2.1.1. 管理コンソールを使用したキャッシュの設定

管理コンソールの **Configuration** タブで **Infinispan** サブシステムを選択するとキャッシュやキャッシュコンテナーを設定できます。マネージドドメインでは設定するプロファイルを選択してください。

- キャッシュコンテナーを追加します。
  Cache Container の横にある追加 (+) ボタンをクリックし、Cache Container を追加 を選択して新しいキャッシュコンテナーの設定を入力します。

- キャッシュコンテナー設定を更新します。
  該当するキャッシュコンテナーを選択し、表示 を選択します。必要に応じてキャッシュコンテナーの設定を行います。

- キャッシュコンテナートランスポート設定を更新します。
  該当するキャッシュコンテナーを選択し、表示 を選択します。Transport タブで、必要に応じてキャッシュコンテナートランスポートを設定します。

- キャッシュを設定します。
  該当するキャッシュコンテナーを選択し、表示 を選択します。キャッシュタブ (Replicated Cache など) でキャッシュを追加、更新、および削除できます。

24.3.2.1.2. 管理 CLI を使用したキャッシュの設定

管理 CLI を使用してキャッシュおよびキャッシュコンテナーを設定できます。マネージドドメインではコマンドの前に **/profile=PROFILE_NAME** を追加して更新するプロファイルを指定する必要があります。

- キャッシュコンテナーを追加します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER:add
```

- レプリケートされたキャッシュを追加します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/replicated-cache=CACHE:add(mode=MODE)
```

- キャッシュコンテナーのデフォルトキャッシュを設定します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER:write-attribute(name=default-cache,value=CACHE)
```

- レプリケートされたキャッシュのバッチ処理を設定します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/replicated-cache=CACHE/component=transaction:write-attribute(name=mode,value=BATCH)
```

以下の例は、**concurrent** 分散キャッシュを **web** キャッシュコンテナーに追加する方法を表しています。このキャッシュ設定はデフォルトキャッシュのロック制約を緩和し、複数の同時リクエストが同じ web セッションに同時にアクセスできるようにします。これは、ロックのない読み取りを許可し、排他的ロックをより頻繁により短期間で取得することで実現されます。

以下の CLI コマンドを使用して **concurrent** 分散キャッシュを **web** キャッシュコンテナーに追加し、デフォルトのキャッシュにします。

```
batch
/subsystem=infinispan/cache-container=web/distributed-cache=concurrent:add
/subsystem=infinispan/cache-container=web:write-attribute(name=default-cache,value=concurrent)
/subsystem=infinispan/cache-container=web/distributed-cache=concurrent/store=file:add
run-batch
```

これにより、サーバーが以下のように設定されます。

```
<cache-container name="web" default-cache="concurrent"
module="org.wildfly.clustering.web.infinispan">
  ...
  <distributed-cache name="concurrent">
     <file-store/>
  </distributed-cache>
</cache-container>
```

24.3.2.1.3. デフォルト Jakarta Enterprise Beans キャッシュコンテナーの変更

以下のようにキャッシュコンテナーを **ejb3** サブシステムで使用できます。

- Jakarta Enterprise Beans セッション bean のパッシベーションをサポートするには、**infinispan** サブシステムに定義された **ejb** キャッシュコンテナーを使用してセッションを保存できます。
- サーバー上でクラスター化されたデプロイメントに接続するリモート Jakarta Enterprise Beans クライアントの場合、クラスタートポロジーの情報をこれらのクライアントに提供して、これらのクライアントが対話するノードに障害が発生したときにクラスターの別のノードにフェイルオーバーできるようにする必要があります。

パッシベーションやトポロジー情報の提供をサポートするデフォルトのキャッシュコンテナー **ejb** を変更する場合や、その名前を変更する場合、以下の例のように **cache-container** 属性を **passivation-stores** 要素に追加し、**cluster** 属性を **remote** 要素に追加する必要があります。独自に使用するために新しいキャッシュを追加する場合は、このような変更を加える必要はありません。

```
<subsystem xmlns="urn:jboss:domain:ejb3:5.0">
    <passivation-stores>
        <passivation-store name="infinispan" cache-container="ejb-cltest" max-size="10000"/>
    </passivation-stores>

    <remote cluster="ejb-cltest" connectors="http-remoting-connector" thread-pool-name="default"/>
</subsystem>

<subsystem xmlns="urn:jboss:domain:infinispan:7.0">
    ...
    <cache-container name="ejb-cltest" aliases="sfsb" default-cache="dist"
module="org.wildfly.clustering.ejb.infinispan">
</subsystem>
```

24.3.2.1.4. Infinispan サブシステムから Jakarta EE アプリケーションにリソースを注入する

@Resource アノテーションを使用して、キャッシュなどの Infinispan リソースを Infinispan サブシステムからアプリケーションに注入できます。次の例は、**@Resource** アノテーションを使用して Jakarta EE アプリケーションにキャッシュを注入する方法を示しています。

```
@Resource(lookup = "java:jboss/infinispan/cache/foo/bar")
private org.infinispan.Cache<Integer, Object> cache;
```

上記の例では、**foo** はキャッシュコンテナーの名前であり、**bar** は注入するキャッシュの名前です。

EAP は注入されたリソースのライフサイクルを管理するため、アプリケーションはキャッシュやキャッシュマネージャーなどのリソースを管理する必要がありません。

注記

リソースを手動で作成すると、EAP ではなくアプリケーションがそれらのリソースを管理します。

次の例は、Infinispan サブシステムから、さまざまなリソースをアプリケーションに注入する方法を示しています。

デフォルトのキャッシュを注入する例

キャッシュコンテナーのデフォルトキャッシュを Infinispan サブシステムからアプリケーションに注入するには、次のコマンドを使用します。

```
@Resource(lookup = "java:jboss/infinispan/cache/foo/default")
```

埋め込みキャッシュマネージャーの注入例

埋め込みキャッシュマネージャーを注入して、アプリケーションが新しいキャッシュ設定とキャッシュを作成できるようにするには、次のコマンドを使用します。

```
@Resource(lookup = "java:jboss/infinispan/container/foo")
private org.infinispan.manager.EmbeddedCacheManager manager;
```

キャッシュ設定の注入例

Infinispan サブシステム内で定義されたキャッシュ設定は、アプリケーションがそれらのキャッシュ設定に明示的に依存しない限り、常にインストールされていたり使用できたりするわけではありません。Infinispan サブシステムからアプリケーションにキャッシュ設定を注入するには、**@Resource** アノテーションを使用します。

- 次の例では、**@Resource** アノテーションを使用して **foo** コンテナーのキャッシュ設定を注入します。

  ```
  @Resource(lookup = "java:jboss/infinispan/configuration/foo/bar")
  private org.infinispan.config.Configuration config;
  ```

- 次の例では、**@Resource** アノテーションを使用して、**foo** コンテナーのデフォルトのキャッシュ設定を注入します。

  ```
  @Resource(lookup = "java:jboss/infinispan/configuration/foo/default")
  private org.infinispan.config.Configuration config;
  ```

24.3.2.1.5. Hibernate キャッシュコンテナーのエビクション機能

hibernate キャッシュコンテナーのエビクション機能は、メモリーからキャッシュエントリーを削除します。この機能は、サブシステムのメモリー負荷を軽減するのに役立ちます。

size 属性は、キャッシュエントリーのエビクションの開始前に保存されるキャッシュエントリーの最大数を設定します。

例: エビクション機能

-

```
<cache-container name="hibernate" default-cache="local-query" module="org.hibernate.infinispan">
  <transport lock-timeout="60000"/>
  <local-cache name="local-query">
    <object-memory size="1000"/>
    <expiration max-idle="100000"/>
```

エビクションはメモリー内でのみ実行されることに注意してください。キャッシュストアは、情報の永続的な損失を防ぐためにエビクトされたキャッシュエントリーを保持します。エビクション 機能についての詳細は、Infinispan ユーザーガイドの エビクションおよびデータコンテナー セクションを参照してください。

24.3.2.1.6. Hibernate キャッシュコンテナーの有効期限機能

hibernate キャッシュコンテナーの有効期限機能は、クラスター化された操作です。したがって、クラスター化されたキャッシュを使用すると、期限切れのキャッシュエントリーがすべてのクラスターメンバーから削除されます。詳細は、Infinispan ユーザーガイドの 有効期限 セクションを参照してください。

24.3.3. クラスタリングモード

JBoss EAP で Infinispan を使用すると、2 つの方法でクラスタリングを設定できます。ご使用のアプリケーションに最も適した方法は要件によって異なります。各モードでは可用性、一貫性、信頼性、およびスケーラビリティーのトレードオフが発生します。クラスタリングモードを選択する前に、ネットワークで最も重要な点を特定し、これらの要件のバランスを取ることが必要となります。

キャッシュモード

Replication (レプリケーション)

Replication (レプリケーション) モードはクラスターの新しいインスタンスを自動的に検出し、追加します。これらのインスタンスに加えられた変更は、クラスター上のすべてのノードにレプリケートされます。ネットワークでレプリケートされる情報量が多いため、通常 Replication モードは小型のクラスターでの使用に最も適しています。UDP マルチキャストを使用するよう Infinispan を設定すると、ネットワークトラフィックの輻輳をある程度軽減できます。

Distribution (分散)

Distribution (分散) モードでは、Infinispan はクラスターを線形にスケールできます。Distribution モードは一貫性のあるハッシュアルゴリズムを使用して、クラスター内で新しいノードを配置する場所を決定します。保持する情報のコピー数 (または所有者数) は設定可能です。保持するコピー数、データの永続性、およびパフォーマンスにはトレードオフが生じます。保持するコピー数が多いほどパフォーマンスへの影響が大きくなりますが、サーバーの障害時にデータを損失する可能性は低くなります。ハッシュアルゴリズムは、メタデータのマルチキャストや保存を行わずにエントリーを配置し、ネットワークトラフィックを軽減します。
クラスターサイズが 6-8 ノードを超える場合は Distribution モードをキャッシュストラテジーとして使用することを考慮してください。Distribution モードでは、データはクラスター内のすべてのノードではなくノードのサブセットのみに分散されます。

Scattered (散在)

Scattered (散在) モードは、一貫したハッシュアルゴリズムを使用して所有者を判断する Distribution モードと似ています。しかし、所有者は 2 名のメンバーに限定され、オリジネーター (指定のセッションのリクエストを受信するノード) は常にロックを調整する所有者やキャッシュエントリーの更新を想定します。2 名の所有者を持つ分散キャッシュが頻繁に 2 つの RPC 呼び出しを使用できる場合、Scattered モードで使用されるキャッシュ書き込みアルゴリズムは、書き込み操作の結果が単一の RPC 呼び出しになることを保証します。これは、ロードバランサーのフェイルオー

バーがプライマリー以外の所有者やバックアップノードにトラフィックを向ける可能性があるため、分散 Web セッションに便利です。これにより、競合の可能性を下げ、クラスタートポロジーの変更後にパフォーマンスを向上することが可能になります。
Scattered モードは、トランザクションや L1 キャッシュをサポートしません。しかし、一貫したハッシュによると所有者ではない場合でも、指定エントリーのキャッシュの書き込みを開始するノードは一定の期間は継続してそのエントリーの読み取りを継続する、バイアス読み取り (Biased read) をサポートします。バイアス読み取りと L1 キャッシュの設定属性は異なりますが、L1 キャッシュと同様の効果があります。

同期および非同期のレプリケーション
レプリケーションは同期または非同期モードで実行でき、選択するモードは要件やアプリケーションモードによって異なります。

重要

JBoss EAP 7.1 以降、セッションレプリケーションには常に同期 (**SYNC**) キャッシュモードを使用する必要があります。**sync** はデフォルトのキャッシュモードです。セッションレプリケーションと適切なキャッシュモードの詳細は、How to configure and tune the session replication for EAP を参照してください。

同期のレプリケーション

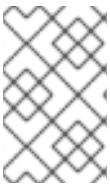
同期のレプリケーションでは、レプリケーションプロセスはユーザーのリクエストを処理するのと同じスレッドで実行されます。セッションレプリケーションは、完了した応答がクライアントに返送された後に開始され、スレッドはレプリケーションの完了後のみに解放されます。同期レプリケーションはクラスターの各ノードからの応答を必要とするため、ネットワークトラフィックに影響を与えます。ただし、クラスターのすべてのノードへ確実に変更が加えられる利点があります。

非同期のレプリケーション

非同期のレプリケーションでは、Infinispan はスレードプールを使用してバックグラウンドでレプリケーションを実行します。送信元はクラスターの他のノードからの返答を待ちません。しかし、同じセッションを読み取るキャッシュはその前のレプリケーションが完了するまでブロックされるため、陳腐データは読み取られません。レプリケーションは時間ベースまたはキューのサイズによって引き起こされます。失敗したレプリケーションはログに書き込まれ、リアルタイムで通知されません。

24.3.3.1. キャッシュモードの設定

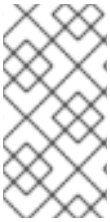
管理 CLI を使用してデフォルトキャッシュを変更できます。



注記

ここでは、デフォルトが Distribution モードである Web セッションキャッシュの設定に固有する手順を説明します。手順と管理 CLI コマンドは、他のキャッシュコンテナー向けに簡単に調整できます。

Replication キャッシュモードへの変更
Web セッションキャッシュのデフォルトの JBoss EAP 7 設定には、レプリケートされたキャッシュ **repl** が含まれていません。最初にこのキャッシュを追加する必要があります。



注記

以下はスタンドアロンサーバーの管理 CLI コマンドになります。マネージドドメインで実行している場合は、/**subsystem=infinispan** コマンドの前に /**profile=PROFILE_NAME** を追加し、更新するプロファイルを指定する必要があります。

1. レプリケーションキャッシュ **repl** を追加し、デフォルトのキャッシュとして設定します。

```
batch
/subsystem=infinispan/cache-container=web/replicated-cache=repl:add()
/subsystem=infinispan/cache-container=web/replicated-
cache=repl/component=transaction:add(mode=BATCH)
/subsystem=infinispan/cache-container=web/replicated-
cache=repl/component=locking:add(isolation=REPEATABLE_READ)
/subsystem=infinispan/cache-container=web/replicated-cache=repl/store=file:add
/subsystem=infinispan/cache-container=web:write-attribute(name=default-cache,value=repl)
run-batch
```

2. サーバーをリロードします。

```
reload
```

Distribution キャッシュモードへの変更
Web セッションキャッシュのデフォルトの JBoss EAP 7 設定にはすでに **dist** 分散キャッシュが含まれています。

注記

以下はスタンドアロンサーバーの管理 CLI コマンドになります。マネージドドメインで実行している場合は、/**subsystem=infinispan** コマンドの前に /**profile=PROFILE_NAME** を追加し、更新するプロファイルを指定する必要があります。

1. デフォルトのキャッシュを **dist** 分散キャッシュに変更します。

```
/subsystem=infinispan/cache-container=web:write-attribute(name=default-cache,value=dist)
```

2. 分散キャッシュの所有者数を設定します。以下のコマンドは所有者の数を **5** に設定します。デフォルトは **2** です。

```
/subsystem=infinispan/cache-container=web/distributed-cache=dist:write-
attribute(name=owners,value=5)
```

3. サーバーをリロードします。

```
reload
```

Scattered キャッシュモードの変更
web セッションキャッシュのデフォルトの JBoss EAP 設定には **scattered-cache** が含まれていません。以下の例は、scattered キャッシュを追加し、デフォルトのキャッシュとして設定する管理 CLI コマンドを示しています。

注記

以下は、HA プロファイルを使用するスタンドアロンサーバー向けの管理 CLI コマンドになります。マネージドドメインで実行している場合は、**/subsystem=infinispan** コマンドの前に **/profile=PROFILE_NAME** を追加し、更新するプロファイルを指定する必要があります。

1. 読み取りのバイアスライフスパンがデフォルトの web セッションタイムアウト値 (30 分) と同じになるよう、scattered キャッシュを作成します。

```
/subsystem=infinispan/cache-container=web/scattered-cache=scattered:add(bias-lifespan=1800000)
```

2. **scattered** をデフォルトキャッシュとして設定します。

```
/subsystem=infinispan/cache-container=web:write-attribute(name=default-cache,value=scattered)
```

これにより、サーバーが以下のように設定されます。

```
<cache-container name="web" default-cache="scattered" module="org.wildfly.clustering.web.infinispan">
    ...
    <scattered-cache name="scattered" bias-lifespan="1800000"/>
    ...
</cache-container>
```

24.3.3.2. キャッシュストラテジーのパフォーマンス

SYNC キャッシュストラテジーを使用すると、レプリケーションが完了するまでリクエストが完了しないため、レプリケーションのコストを簡単に評価でき、直接応答時間に影響します。

ASYNC キャッシュストラテジーの応答時間は **SYNC** キャッシュストラテジーよりも短いと思われがちですが、一定の状況下でのみ短くなります。**ASYNC** キャッシュストラテジーの評価はより困難ですが、リクエスト間の時間がキャッシュ操作を完了できるほど長い場合はパフォーマンスが **SYNC** よりも良くなります。これは、レプリケーションのコストが応答時間に即影響しないためです。

同じセッションのリクエストの作成が早すぎると、先行リクエストからのレプリケーションが完了するまで待機しなければならないため、先行リクエストのレプリケーションコストが後続リクエストの前に移動します。応答の受信直後に後続リクエストが送信され、リクエストが急速に発生する場合、**ASYNC** キャッシュストラテジーのパフォーマンスは **SYNC** よりも劣化します。そのため、同じセッションのリクエストの間には、**SYNC** キャッシュストラテジーのパフォーマンスが **ASYNC** キャッシュストラテジーよりも良くなる期間のしきい値があります。実際の使用状態では、通常同じセッションのリクエストが立て続けに受信されることはありませんが、一般的にリクエストの間に数秒程度の時間が存在します。その代わりに、通常、要求間に数秒以上の時間が経過します。この場合、**ASYNC** キャッシュストラテジーが適切なデフォルトで、応答時間が早くなります。

24.3.4. 状態遷移

状態遷移は、基本的なデータグリッドとクラスター化されたキャッシュ機能の両方です。状態遷移がない状態では、ノードがクラスターに追加されたり、クラスターから削除されるとデータが失われます。

状態遷移は、キャッシュメンバーシップの変更に応じて、キャッシュの内部状態を調整します。この変更は、ノードがクラスターに参加または退出するとき、2 つ以上のクラスターパーティションがマージ

するとき、またはこれらのイベントの組み合わせが発生した後に自動的に行われます。新しいキャッシュは、以下のキャッシュのモードを基にして、最大量の状態を受け取る必要があるため、新たに開始されたキャッシュの最初の状態遷移は最も多くのリソースが必要となります。

timeout 属性を使用すると、新たに開始されたキャッシュが状態を受け取る待ち時間を制御することができます。**timeout** 属性が正の値である場合、キャッシュはすべての初期状態を受け取るまで待機した後、リクエストに対応できるようになります。状態遷移が指定時間内 (デフォルトは **240000** ミリ秒) に完了しなかった場合、キャッシュによってエラーが発生し、開始がキャンセルされます。**timeout** を **0** に設定するとキャッシュは即座に利用可能になり、バックグラウンド操作中に最初の状態を受け取ります。最初の状態遷移が完了するまで、キャッシュが受け取っていないキャッシュエントリーのリクエストは、リモートノードから取得する必要があります。

timeout 属性を **0** に設定するには、以下のコマンドを実行します。

```
/subsystem=infinispan/cache-container=server/CACHE_TYPE=CACHE/component=state-transfer:write-attribute(name=timeout,value=0)
```

状態遷移の挙動はキャッシュのモードによって決まります。

- Replicated (レプリケート) モードでは、キャッシュに参加する新規ノードは既存ノードからキャッシュの状態をすべて受け取ります。ノードがクラスターから退出すると、状態遷移はありません。
- Distribution (分散) モードでは、新しいノードは既存のノードから状態の一部のみを受け取り、既存のノードは一貫したハッシュの決定どおりに、各キーの **owners** コピーを維持するために状態の一部を削除します。ノードがクラスターから退出する場合、分散キャッシュはそのノードに保存されたキーの追加コピーを作成し、各キーの所有者が存続するようにする必要デフォルトがあります。
- Invalidation (インバリデーション) モードでは、最初の状態推移は Replicated モードと似ていますが、ノードが同じ状態でいられる保証がないことが唯一の違いです。ノードがクラスターから退出すると、状態遷移はありません。

デフォルトでは、状態遷移はインメモリーおよび永続状態の両方を遷移しますが、設定で無効にすることもできます。状態遷移が無効になっている場合、**ClusterLoader** を設定しないと、データをキャッシュにロードせずにノードがキーの所有者またはバックアップ所有者になります。さらに、Distribution モードで状態遷移が無効になっていると、キャッシュでキーのコピーが **owners** コピーよりも少なくなることがあります。

24.3.5. Infinispan スレッドプールの設定

infinispan サブシステムには **async-operations**、**expiration**、**listener**、**persistence**、**remote-command**、**state-transfer**、および **transport** スレッドプールが含まれます。これらのプールはすべての Infinispan キャッシュコンテナーに設定できます。

以下の表は、**infinispan** サブシステムの各スレッドプールに設定できる属性とデフォルト値を表しています。

スレッドプール名	keepalive-time	max-threads	min-threads	queue-length
async-operations	60000L	25	25	1000
expiration	60000L	1	該当なし	該当なし

スレッドプール名	keepalive-time	max-threads	min-threads	queue-length
listener	60000L	1	1	100000
persistence	60000L	4	1	0
remote-command	60000L	200	1	0
state-transfer	60000L	60	1	0
transport	60000L	25	25	100000

以下の構文を使用して、管理 CLI で Infinispan スレッドプールを設定します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER_NAME/thread-pool=THREAD_POOL_NAME:write-attribute(name=ATTRIBUTE_NAME, value=ATTRIBUTE_VALUE)
```

以下は、 **server** キャッシュコンテナーの **persistence** スレッドプールで **max-threads** の値を **10** に設定する管理 CLI コマンドの例になります。

```
/subsystem=infinispan/cache-container=server/thread-pool=persistence:write-attribute(name="max-threads", value="10")
```

24.3.6. Infinispan の統計

監視目的で、Infinispan キャッシュやキャッシュコンテナーに関する実行時統計を有効にできます。パフォーマンス上の理由で、統計の収集はデフォルトでは無効になっています。

統計収集は、各キャッシュコンテナー、キャッシュ、または両方に対して有効にできます。各キャッシュの統計オプションはキャッシュコンテナーのオプションをオーバーライドします。キャッシュコンテナーの統計収集を無効または有効にすると、独自の設定が明示的に指定されている場合以外はそのコンテナーのすべてのキャッシュが設定を継承します。

24.3.6.1. Infinispan 統計の有効化

警告

Infinispan の統計を有効にすると、**infinispan** サブシステムのパフォーマンスに影響する可能性があります。統計は必要な場合のみ有効にしてください。

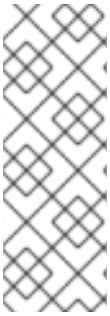
管理コンソールまたは管理 CLI を使用すると Infinispan の統計収集を有効または無効にできます。管理コンソールでは、**Configuration** タブで **Infinispan** サブシステム選択してキャッシュまたはキャッシュコンテナーを選択し、**Statistics Enabled** 属性を編集します。管理 CLI を使用する場合は以下のコマンドを実行して統計を有効にします。

キャッシュコンテナーの統計収集を有効にします。サーバーをリロードする必要があります。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER:write-attribute(name=statistics-enabled,value=true)
```

キャッシュの統計収集を有効にします。サーバーをリロードする必要があります。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/CACHE_TYPE=CACHE:write-attribute(name=statistics-enabled,value=true)
```



注記

以下のコマンドを使用すると、キャッシュの **statistics-enabled** 属性の定義が解除され、キャッシュコンテナーの **statistics-enabled** 属性の設定を継承します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/CACHE_TYPE=CACHE:undefine-attribute(name=statistics-enabled)
```

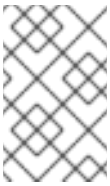
24.3.7. Infinispan パーティションの処理

Infinispan クラスターは、データが保存される複数のノードで構築されます。複数のノードに障害が発生した場合のデータの損失を防ぐため、Infinispan は複数のノードで同じデータをコピーします。このレベルのデータ冗長性は **owners** 属性を使用して設定されます。設定されたノードの数未満のノードが同時にクラッシュしても、Infinispan はデータのコピーを利用できます。

しかし、クラスターから大量のノードが消滅すると最悪の事態を招く可能性があります。

スプリットブレイン

スプリットブレインはクラスターを独立して動作する 2 つ以上のパーティションまたはサブクラスターに分割します。このような場合、異なるパーティションから読み書きする複数のクライアントが同じキャッシュエントリーの異なるバージョンを見ることになり、多くのアプリケーションにとって問題となります。



注記

スプリットブレインの発生を軽減する方法には、冗長化ネットワークや IP ボンディング などがあります。 しかし、これらの方法は問題発生のリードタイムを削減するだけです。

複数ノードの連続クラッシュ

複数のノード (所有者の数) が連続してクラッシュし、Infinispan がクラッシュ間の状態を適切に調整する時間がない場合、結果として部分的なデータの損失が発生します。

スプリットブレインや複数ノードの連続クラッシュが原因で、不適切なデータがユーザーに返されないようにすることが大切です。

24.3.7.1. スプリットブレイン

スプリットブレインが発生した場合、各ネットワークパーティションが独自の JGroups ビューをインストールし、他のパーティションからノードを削除します。パーティションはお互いを認識しないため、クラスターが 2 つ以上のパーティションに分割されたかどうかを直接判断することはできません。

そのため、明示的な脱退メッセージを送信せずに、1 つ以上のノードが JGroups クラスターから消滅した場合にクラスターが分割されたと判断します。

パーティション処理が無効の場合、各パーティションは継続して独立したクラスターとして機能します。各パーティションはデータの一部のみを認識できる可能性があり、競合する更新をキャッシュに書き込む可能性があります。

パーティション処理が有効の場合、スプリットを検出したときに各パーティションは即座にリバランスを行わず、degrade モードにするかどうかを最初にチェックします。

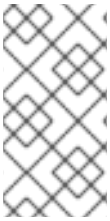
- 1 つ以上のセグメントがすべての所有者を失った場合 (最後に行ったリバランスが完了した後に指定した所有者の数以上が脱退した場合)、パーティションは degrade モードになります。
- 最新の安定したトポロジーでパーティションに単純多数のノード (floor(numNodes/2) + 1) が含まれない場合も、パーティションは degrade モードになります。
- その他の場合は、パーティションは通常通り動作し、リバランスを開始します。

安定したトポロジーは、リバランス操作が終了するたびに更新され、コーディネーターによって他のリバランスが必要ないと判断された場合に毎回更新されます。これらのルールは、1 つのパーティションが available モードを維持し、他のパーティションが degraded モードになるようにします。

パーティションが degraded モードの場合、完全に所有されたキーへのアクセスのみを許可します。

- このパーティション内のノード上のコピーをすべて持つエントリーのリクエスト (読み取りおよび書き込み) は許可されます。
- 消滅したノードによって完全所有または一部所有されたエントリーのリクエストは **AvailabilityException** によって拒否されます。

これにより、パーティションが同じキーに異なる値を書き込めないようにし (キャッシュの一貫性を保つ)、さらにパーティションが他のパーティションで更新されたキーを読み取れないようにします (陳腐データをなくす)。



注記

2 つのパーティションは分離して開始できます。これらのパーティションはマージされなければ不整合なデータを読み書きできます。 将来的に、この状況に対処できるカスタムの可用性ストラテジーが許可される可能性があります (例: 特定のノードがクラスターの一部であるかを確認、外部のマシンにアクセスできるかどうかを確認など)。

24.3.7.2. パーティション処理の設定

現在、パーティションの処理はデフォルトで無効になっています。パーティションの処理を有効にするには、以下の管理 CLI コマンドを使用します。

```
/subsystem=infinispan/cache-container=web/distributed-cache=dist/component=partition-handling:write-attribute(name=enabled, value=true)
```

24.3.8. リモートキャッシュコンテナーの設定

マネージドドメインのすべてのサーバーグループにリモートキャッシュを設定する必要があります。**statistics-enabled** 属性により、指定の **remote-cache-container** および関連するランタイムキャッシュについてのメトリックのコレクションを有効にできます。

24.3.8.1. リモートキャッシュコンテナーの作成

マネージドドメインの各サーバーグループには、一意のリモートキャッシュが必要です。このキャッシュは、同じデータグリッドに所属することができます。そのため、ユーザーは、サーバーグループのソケットバインディングを定義し、ソケットバインディングをリモートキャッシュコンテナーに関連付けることにより、すべてのサーバーグループのリモートキャッシュを設定する必要があります。

手順

1. **socket-binding** を定義します。 クラスターの各リモート Red Hat Data Grid インスタンスの必要に応じて、コマンドを繰り返し実行します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=SOCKET_BINDING:add(host=HOSTNAME,port=PORT)
```

2. 新規作成されたソケットバインディングを参照する **remote-cache-container** を定義します。

```
batch
/subsystem=infinispan/remote-cache-container=CACHE_CONTAINER:add(default-remote-cluster=data-grid-cluster)
/subsystem=infinispan/remote-cache-container=CACHE_CONTAINER/remote-cluster=data-grid-cluster:add(socket-bindings=[SOCKET_BINDING,SOCKET_BINDING_2,...])
run-batch
```

24.3.8.2. リモートキャッシュコンテナーの統計の有効化

statistics-enabled 属性により、指定の **remote-cache-container** および関連するランタイムキャッシュについてのメトリックのコレクションが有効になります。

- foo という **remote-cache-container** 場合は、次の操作を使用して統計を有効にします。

```
/subsystem=infinispan/remote-cache-container=foo:write-attribute(name=statistics-enabled, value=true)
```

- foo という **remote-cache-container** については、以下のメトリックがランタイム時に表示されます。

```
/subsystem=infinispan/remote-cache-container=foo:read-attribute(name=connections)
/subsystem=infinispan/remote-cache-container=foo:read-attribute(name=active-connections)
/subsystem=infinispan/remote-cache-container=foo:read-attribute(name=idle-connections)
```

- これらのメトリックの説明は、**remote-cache-container** に read-resource-description 操作を実行します。

```
/subsystem=infinispan/remote-cache-container=foo:read-resource-description
```

- 以下のメトリックは、選択したデプロイメントによって使用されるリモートキャッシュに固有のものです。

```
/subsystem=infinispan/remote-cache-container=foo/remote-cache=bar.war:read-resource(include-runtime=true, recursive=true)
{
    "average-read-time" : 1,
```

```
    "average-remove-time" : 0,
    "average-write-time" : 2,
    "hits" : 9,
    "misses" : 0,
    "near-cache-hits" : 7,
    "near-cache-invalidations" : 8,
    "near-cache-misses" : 9,
    "near-cache-size" : 1,
    "removes" : 0,
    "time-since-reset" : 82344,
    "writes" : 8
}
```

- これらのメトリックの説明は、リモートキャッシュに対して read-resource-description 操作を実行します。

```
/subsystem=infinispan/remote-cache-container=foo/remote-cache=bar.war:read-resource-description
```

- これらのメトリックの一部は計算値 (例: average- *) であり、ヒットやミスなどのその他は集計値です。この集計メトリックは、以下の操作でリセットできます。

```
/subsystem=infinispan/remote-cache-container=foo/remote-cache=bar.war:reset-statistics()
```

24.3.9. HTTP セッションの Red Hat Data Grid への外部化

注記

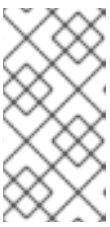
この機能を使用するには Red Hat Data Grid のサブスクリプションが必要です。

Red Hat Data Grid は、HTTP セッションなどの JBoss EAP のアプリケーション固有データの外部キャッシュコンテナーとして使用できます。これにより、アプリケーションとは独立したデータレイヤーのスケーリングが可能になり、さまざまなドメインに存在する可能性がある異なる JBoss EAP クラスターが同じ Red Hat Data Grid クラスターからデータにアクセスできるようになります。また、他のアプリケーションは Red Hat Data Grid によって提供されたキャッシュと対話できます。

以下の例は、HTTP セッションを外部化する方法を説明しています。これは、JBoss EAP のスタンドアロンインスタンスとマネージドドメインの両方に適用されます。

1. **remote-cache-container** を作成します。詳細は、リモートキャッシュコンテナーの設定 を参照してください。

2. HotRod ストアを設定します。HotRod ストアは、JBoss EAP サーバーによって作成された各キャッシュに対して専用のリモートキャッシュを 1 つ使用します。通常、以下の CLI スクリプトのように、JBoss EAP サーバーで 1 つのインバリデーション キャッシュが使用されます。



注記

Red Hat JDG サーバーでリモートキャッシュを手作業で設定する必要があります。これらのキャッシュに推奨される設定は、楽観的ロックを持つトランザクション分散モードです。キャッシュ名はデプロイメントファイル名に対応する必要があります (例: **test.war**)。

リモートキャッシュコンテナーが設定されたら、**hotrod** ストアを設定して既存のストアを置き換えることができます。以下の CLI スクリプトは、インバリデーションキャッシュとともにセッションをオフロードする典型的なユースケースを示しています。

```
batch
/subsystem=infinispan/cache-container=web/invalidation-cache=CACHE_NAME:add()
/subsystem=infinispan/cache-container=web/invalidation-
cache=CACHE_NAME/store=hotrod:add(remote-cache-
container=CACHE_CONTAINER,fetch-
state=false,purge=false,passivation=false,shared=true)
/subsystem=infinispan/cache-container=web/invalidation-
cache=CACHE_NAME/component=transaction:add(mode=BATCH)
/subsystem=infinispan/cache-container=web/invalidation-
cache=CACHE_NAME/component=locking:add(isolation=REPEATABLE_READ)
/subsystem=infinispan/cache-container=web:write-attribute(name=default-
cache,value=CACHE_NAME)
run-batch
```

このスクリプトは新しいインバリデーションキャッシュを設定します。作成後、セッションデータはパフォーマンスの目的でキャッシュに維持され、復元の目的でストアに書き込まれます。

@Resource アノテーションを使用すると、HotRod クライアントを直接 Jakarta EE アプリケーションにインジェクトすることができます。**@Resource** アノテーションは **hotrod-client.properties** ファイルで、クラスパスの設定プロパティーを検索します。

```
@Resource(lookup = "java:jboss/infinispan/remote-container/web-sessions")
private org.infinispan.client.hotrod.RemoteCacheContainer client;
```

例: hotrod-client.properties ファイル

```
infinispan.client.hotrod.transport_factory =
org.infinispan.client.hotrod.impl.transport.tcp.TcpTransportFactory
infinispan.client.hotrod.server_list = 127.0.0.1:11222
infinispan.client.hotrod.marshaller =
org.infinispan.commons.marshall.jboss.GenericJBossMarshaller
infinispan.client.hotrod.async_executor_factory =
org.infinispan.client.hotrod.impl.async.DefaultAsyncExecutorFactory
infinispan.client.hotrod.default_executor_factory.pool_size = 1
infinispan.client.hotrod.default_executor_factory.queue_size = 10000
infinispan.client.hotrod.hash_function_impl.1 =
org.infinispan.client.hotrod.impl.consistenthash.ConsistentHashV1
infinispan.client.hotrod.tcp_no_delay = true
infinispan.client.hotrod.ping_on_startup = true
infinispan.client.hotrod.request_balancing_strategy =
org.infinispan.client.hotrod.impl.transport.tcp.RoundRobinBalancingStrategy
infinispan.client.hotrod.key_size_estimate = 64
infinispan.client.hotrod.value_size_estimate = 512
infinispan.client.hotrod.force_return_values = false

## below is connection pooling config

maxActive=-1
maxTotal = -1
maxIdle = -1
```

```
whenExhaustedAction = 1
timeBetweenEvictionRunsMillis=120000
minEvictableIdleTimeMillis=300000
testWhileIdle = true
minIdle = 1
```

リモートキャッシュコンテナーのセキュア化
SSL を使用してリモート Red Hat Data Grid インスタンスへの通信をセキュアにすることが可能です。これを行うには、JBoss EAP インスタンスで **remote-cache-container** を設定し、Red Hat Data Grid インスタンスで hotrod コネクターを調整してアクティブなセキュリティーレルムを使用するようにします。

1. JBoss EAP で **client-ssl-context** を作成します。他の elytron コンポーネントの生成など、**client-ssl-context** 作成のその他の詳細は、JBoss EAP**How to Configure Server Security**の Using a client-ssl-context を参照してください。

```
/subsystem=elytron/client-ssl-context=CLIENT_SSL_CONTEXT:add(key-manager=KEY_MANAGER,trust-manager=TRUST_MANAGER)
```

2. クライアント SSL コンテキストを使用するよう、リモートキャッシュコンテナーを設定します。

```
/subsystem=infinispan/remote-cache-container=CACHE_CONTAINER/component=security:write-attribute(name=ssl-context,value=CLIENT_SSL_CONTEXT)
```

3. リモート Red Hat Data Grid インスタンスをセキュアにします。各インスタンスの必要に応じて手順を繰り返します。

   a. **client-ssl-context** で使用されるキーストアをリモート Red Hat Data Grid インスタンスにコピーします。

   b. **ApplicationRealm** を設定して、このキーストアを使用するようにします。

```
/core-service=management/security-realm=ApplicationRealm/server-identity=ssl:add(keystore-path="KEYSTORE_NAME",keystore-relative-to="jboss.server.config.dir",keystore-password="KEYSTORE_PASSWORD")
```

   c. hotrod connector を調整して、このセキュリティーレルムを示すようにします。

```
/subsystem=datagrid-infinispan-endpoint/hotrod-connector=hotrod-connector/encryption=ENCRYPTION:add(require-ssl-client-auth=false,security-realm="ApplicationRealm")
```

   d. リモート Red Hat Data Grid インスタンスをリロードします。

```
reload
```

24.3.10. リモートストアを使用した HTTP セッションの Red Hat Data Grid への外部化

注記

この機能を使用するには Red Hat Data Grid のサブスクリプションが必要です。

この手順は、セッションを外部化する旧式の方法になります。JBoss EAP 7.2 には、**elytron** サブシステムと統合する HotRod プロトコルをベースとするカスタムの最適化されたキャッシュストアが導入されました。**HTTP** セッションの **JBoss Data Grid** への外部化 の説明にしたがって、新しい hotrod ストアを使用することが推奨されます。

注記

分散可能なアプリケーションごとに完全に新しいキャッシュを作成する必要があります。新しいキャッシュは **web** などの既存のキャッシュコンテナーに作成できます。

HTTP セッションを外部化するには、以下を行います。

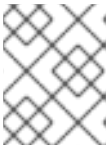
1. ネットワーク情報を **socket-binding-group** に追加することにより、リモート Red Hat Data Grid サーバーの場所を定義します。

   例: リモートソケットバインディングの追加

   ```
   /socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-rhdg-server1:add(host=RHDGHostName1, port=11222)

   /socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-rhdg-server2:add(host=RHDGHostName2, port=11222)
   ```

   結果の XML

   ```
   <socket-binding-group name="standard-sockets" ... >
     ...
     <outbound-socket-binding name="remote-rhdg-server1">
       <remote-destination host="RHDGHostName1" port="11222"/>
     </outbound-socket-binding>
     <outbound-socket-binding name="remote-rhdg-server2">
       <remote-destination host="RHDGHostName2" port="11222"/>
     </outbound-socket-binding>
   </socket-binding-group>
   ```

   注記

   各 Red Hat Data Grid サーバーにリモートソケットバインディングを設定する必要があります。

2. リモートキャッシュコンテナーが JBoss EAP の **infinispan** サブシステムで定義されているようにしてください。 以下の例では、**remote-store** 要素の **cache** 属性によって、リモート Red Hat Data Grid サーバーのキャッシュ名が定義されます。
   マネージドドメインで実行している場合は、コマンドの前に **/profile=PROFILE_NAME** を追加してください。

   例: リモートキャッシュコンテナーの追加

   ```
   /subsystem=infinispan/cache-container=web/invalidation-cache=rhdg:add(mode=SYNC)

   /subsystem=infinispan/cache-container=web/invalidation-cache=rhdg/component=locking:write-attribute(name=isolation,value=REPEATABLE_READ)
   ```

```
/subsystem=infinispan/cache-container=web/invalidation-
cache=rhdg/component=transaction:write-attribute(name=mode,value=BATCH)

/subsystem=infinispan/cache-container=web/invalidation-
cache=rhdg/store=remote:add(remote-servers=["remote-rhdg-server1","remote-rhdg-
server2"], cache=default, socket-timeout=60000, passivation=false, purge=false,
shared=true)
```

結果の XML

```
<subsystem xmlns="urn:jboss:domain:infinispan:7.0">
  ...
  <cache-container name="web" default-cache="dist"
module="org.wildfly.clustering.web.infinispan" statistics-enabled="true">
    <transport lock-timeout="60000"/>
    <invalidation-cache name="rhdg" mode="SYNC">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <remote-store cache="default" socket-timeout="60000" remote-servers="remote-rhdg-
server1 remote-rhdg-server2" passivation="false" purge="false" shared="true"/>
    </invalidation-cache>
    ...
  </cache-container>
</subsystem>
```

3. キャッシュ情報をアプリケーションの **jboss-web.xml** ファイルに追加します。以下の例では、**web** はキャッシュコンテナーの名前で、**rhdg** はこのコンテナーにある適切なキャッシュの名前になります。

例: jboss-web.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-web_10_0.xsd"
           version="10.0">
    <replication-config>
        <replication-granularity>SESSION</replication-granularity>
        <cache-name>web.rhdg</cache-name>
    </replication-config>
</jboss-web>
```

24.4. JBOSS EAP をフロントエンドロードバランサーとして設定

バックエンド JBoss EAP サーバーへリクエストをプロキシーするフロントエンドロードバランサーとして動作するよう JBoss EAP と **undertow** サブシステムを設定できます。Undertow は非同期 IO を使用するため、リクエストに関与するスレッドは接続用の IO スレッドのみになります。同じスレッドがバックエンドサーバーへの接続に使用されます。

以下のプロトコルを使用できます。

- HTTP/1 および HTTP/2 (**h2c**) をサポートするプレーンテキスト上の HTTP (**http**)

- HTTP/1 および HTTP/2 (**h2c**) をサポートするセキュアの接続上の HTTP (**http**)
- AJP (**ajp**)

静的ロードバランサー を定義して設定でバックエンドホストを指定するか、mod_cluster フロントエンド を使用してホストを動的に更新します。

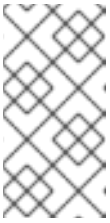
HTTP/2 を使用するよう Undertow を設定する手順は、HTTP/2 の設定 を参照してください。

24.4.1. mod_cluster を使用して Undertow をロードバランサーとして設定

組み込みの mod_cluster フロントエンドロードバランサーを使用して、他の EAP インスタンスを負荷分散できます。

この手順では、マネージドドメインを実行し、以下が設定済みであることを前提としています。

- ロードバランサーとして動作する JBoss EAP サーバー。
  - このサーバーは、**load-balancer-sockets** ソケットバインディンググループにバインドされる **load-balancer** プロファイルを使用します。



注記

このサーバーをフロントエンドロードバランサーとして使用するため、**load-balancer** プロファイルには、ソケットバインディング、mod-cluster Undertow フィルター、およびデフォルトホストでのフィルターへの参照がすでに事前設定されています。

- バックエンドサーバーとして動作する 2 つの JBoss EAP サーバー。
  - これらのサーバーはクラスターで実行され、**ha-sockets** ソケットバインディンググループにバインドされる **ha** プロファイルを使用します。
- バックエンドサーバーにデプロイされた負荷分散される分散可能なアプリケーション。

mod_cluster フロントエンドロードバランサーの設定
以下の手順は、マネージドドメインのサーバーを負荷分散しますが、手順を変更するとスタンドアロンサーバーのセットに適用することができます。ご使用の環境に応じて管理 CLI コマンドの値を変更してください。

1. mod_cluster アドバタイズセキュリティーキーを設定します。
   アドバタイズセキュリティーキーを追加すると、ロードバランサーとサーバーが検出中に認証されます。

   以下の管理 CLI コマンドを使用して、mod_cluster アドバタイズセキュリティーキーを設定します。

   ```
   /profile=ha/subsystem=modcluster/proxy=default:write-attribute(name=advertise-security-key,
   value=mypassword)
   ```

2. mod_cluster ロードバランサーのセキュリティーキーを更新します。
   以下の管理 CLI コマンドを使用して、mod_cluster フィルターのセキュリティーキーを設定します。

```
/profile=load-balancer/subsystem=undertow/configuration=filter/mod-cluster=load-balancer:write-attribute(name=security-key,value=mypassword)
```

重要

mod_cluster によって使用される管理およびアドバタイズソケットバインディングが内部ネットワークのみで公開され、パブリック IP アドレスで公開されないことが推奨されます。

ロードバランサーである JBoss EAP サーバーが 2 つのバックエンドである JBoss EAP サーバーを負荷分散できるようになります。

複数の mod_cluster 設定
mod_cluster サブシステムは名前の付いたプロキシー設定を複数サポートし、デフォルトでない **undertow** サーバーをリバースプロキシーに登録できます。さらに、単一のアプリケーションサーバーノードを異なるグループのプロキシーサーバーに登録することができます。

以下の例は、**ajp-listener**、サーバー、および **undertow** サーバーへのホストを追加します。また、アドバタイズのメカニズムを使用してホストを登録する新しい **mod_cluster** 設定も追加します。

```
/socket-binding-group=standard-sockets/socket-binding=ajp-other:add(port=8010)
/subsystem=undertow/server=other-server:add
/subsystem=undertow/server=other-server/ajp-listener=ajp-other:add(socket-binding=ajp-other)
/subsystem=undertow/server=other-server/host=other-host:add(default-web-module=root-other.war)
/subsystem=undertow/server=other-server/host=other-host
/location=other:add(handler=welcome-content)
/subsystem=undertow/server=other-server/host=other-host:write-attribute(name=alias,value=[localhost]))

/socket-binding-group=standard-sockets/socket-binding=modcluster-other:add(multicast-address=224.0.1.106,multicast-port=23364)
/subsystem=modcluster/proxy=other:add(advertise-socket=modcluster-other,balancer=other-balancer,connector=ajp-other)

reload
```

24.4.2. ロードバランサーでのランク付けされたセッションアフィニティーの有効化

distributable-web サブシステムで複数の順序付けされたルートを持つセッションアフィニティーを設定するには、ランク付けされたセッションアフィニティーをロードバランサーで有効にする必要があります。**distributable-web** サブシステムと各種アフィニティーオプションの詳細は、JBoss EAP**Development Guide**の The distributable-web subsystem for Distributable Web Session Configurations を参照してください。

ノードルートを分離するデフォルトの区切り文字は **.** です。別の値が必要な場合は、**affinity** リソースの **delimiter** 属性を設定できます。

手順

1. ロードバランサーに対して、ランク付けされたセッションアフィニティーを有効にします。

```
/subsystem=undertow/configuration=filter/mod-cluster=load-balancer/affinity=ranked:add
```

2. オプション: **affinity** リソースの **delimiter** 属性を設定します。

```
/subsystem=undertow/configuration=filter/mod-cluster=load-balancer/affinity=ranked:write-attribute(name=delimiter,value=':')
```

24.4.3. Undertow を静的ロードバランサーとして設定

Undertow の静的ロードバランサーを設定するには、**undertow** サブシステムでプロキシーハンドラーを設定する必要があります。Undertow でプロキシーハンドラーを設定するには、静的ロードバランサーとして動作する JBoss EAP インスタンスで以下を行う必要があります。

1. リバースプロキシーハンドラーを追加します。
2. 各リモートホストのアウトバウンドソケットバインディングを定義します。
3. 各リモートホストをリバースプロキシーハンドラーへ追加します。
4. リバースプロキシーの場所を追加します。

以下の例は、JBoss EAP インスタンスを静的ロードバランサーとして設定する方法を示しています。JBoss EAP インスタンスは **lb.example.com** にあり、2 つの追加サーバーである **server1.example.com** と **server2.example.com** との間で負荷分散を行います。ロードバランサーは **/app** に逆プロキシーを行い、AJP プロトコルを使用します。

1. リバースプロキシーハンドラーを追加するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-handler:add
```

2. 各リモートホストのアウトバウンドソケットバインディングを定義するには、以下を指定します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-host1/:add(host=server1.example.com, port=8009)

/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-host2/:add(host=server2.example.com, port=8009)
```

3. 各リモートホストをリバースプロキシーハンドラーに追加するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-handler/host=host1:add(outbound-socket-binding=remote-host1, scheme=ajp, instance-id=myroute1, path=/test)

/subsystem=undertow/configuration=handler/reverse-proxy=my-handler/host=host2:add(outbound-socket-binding=remote-host2, scheme=ajp, instance-id=myroute2, path=/test)
```

4. リバースプロキシーの場所を追加するには、以下を指定します。

```
/subsystem=undertow/server=default-server/host=default-host/location=\/test:add(handler=my-handler)
```

lb.example.com:8080/app にアクセスすると、**server1.example.com** および **server2.example.com** からプロキシーされた内容が表示されるようになります。

24.5. 外部 WEB サーバーのプロキシーサーバーとしての使用

JBoss EAP は、外部 Web サーバーの設定に応じて、サポートされる HTTP、HTTPS、または AJP プロトコルを使用して外部 Web サーバーからリクエストを許可できます。

各 Web サーバーのサポートされる HTTP コネクターの詳細は、HTTP コネクターの概要 を参照してください。使用する Web サーバーと HTTP コネクターを決定したら、適切なコネクターの設定情報の項を参照してください。

- Apache HTTP Server の場合は、mod_cluster、mod_jk、または mod_proxy の項を参照してください。
- Microsoft IIS の場合は、ISAPI コネクター の項を参照してください。
- Oracle iPlanet Web Server の場合は、NSAPI コネクター の項を参照してください。

HTTP コネクターのサポートされる設定に関する最新情報は、JBoss EAP でサポートされる設定 を参照してください。

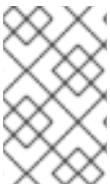
また、JBoss EAP が 外部 Web サーバーからのリクエストを許可 するよう設定されている必要があります。

24.5.1. HTTP コネクターの概要

JBoss EAP には、Apache HTTP Server、Microsoft IIS、Oracle iPlanet などの外部 Web サーバーや Undertow より構築された負荷分散およびクラスタリングメカニズムを使用する機能があります。JBoss EAP はコネクターを使用して Web サーバーと通信します。これらのコネクターは JBoss EAP の **undertow** サブシステム内に設定されます。

Web サーバーには、HTTP リクエストが JBoss EAP ノードにルーティングされる方法を制御するソフトウェアモジュールが含まれています。これらのモジュールの挙動や設定方法はモジュールごとに異なります。モジュールは、複数の JBoss EAP ノード全体でワークロードを分散したり、障害発生時にワークロードを他のサーバーに移動したりするために設定されます。

JBoss EAP は複数のコネクターをサポートします。使用中の Web サーバーや必要な機能に応じてコネクターを選択します。以下の表を参照して、サポートされる設定と、JBoss EAP と互換性のある HTTP コネクターの機能を比較してください。



注記

JBoss EAP 7 をマルチプラットフォームロードバランサーとして使用する場合は mod_cluster を使用して Undertow をロードバランサーとして設定 を参照してください。

HTTP コネクターのサポートされる設定に関する最新情報は、JBoss EAP でサポートされる設定 を参照してください。

表24.1 HTTP コネクターでサポートされる設定

コネクター	Web Server	サポート対象オペレーティングシステム	サポート対象プロトコル

コネクター	Web Server	サポート対象オペレーティングシステム	サポート対象プロトコル
mod_cluster	Red Hat JBoss Core Services の Apache HTTP Server、Red Hat JBoss Web Server の Apache HTTP Server、JBoss EAP (Undertow)	Red Hat Enterprise Linux、Microsoft Windows Server、Oracle Solaris	HTTP、HTTPS、AJP、WebSocket
mod_jk	Red Hat JBoss Core Services の Apache HTTP Server、Red Hat JBoss Web Server の Apache HTTP Server	Red Hat Enterprise Linux、Microsoft Windows Server、Oracle Solaris	AJP
mod_proxy	Red Hat JBoss Core Services の Apache HTTP Server、Red Hat JBoss Web Server の Apache HTTP Server	Red Hat Enterprise Linux、Microsoft Windows Server、Oracle Solaris	HTTP、HTTPS、AJP
ISAPI コネクター	Microsoft IIS	Microsoft Windows Server	AJP
NSAPI コネクター	Oracle iPlanet Web Server	Oracle Solaris	AJP

表24.2 HTTP コネクターの機能

コネクター	スティッキーセッションのサポート	デプロイメント状態への適合
mod_cluster	○	可。アプリケーションのデプロイメントとアンデプロイメントを検出し、アプリケーションがそのサーバーにデプロイされたかどうかに基づいて、サーバーにクライアント要求を送信するかどうかを動的に決定します。
mod_jk	○	不可。 アプリケーションの状態に関係なく、コンテナーが利用可能な限り、クライアント要求をコンテナーに送信します。
mod_proxy	○	不可。 アプリケーションの状態に関係なく、コンテナーが利用可能な限り、クライアント要求をコンテナーに送信します。
ISAPI コネクター	○	不可。 アプリケーションの状態に関係なく、コンテナーが利用可能な限り、クライアント要求をコンテナーに送信します。
NSAPI コネクター	○	不可。 アプリケーションの状態に関係なく、コンテナーが利用可能な限り、クライアント要求をコンテナーに送信します。

24.5.2. Apache HTTP Server

スタンドアロン Apache HTTP Server バンドルは、Red Hat JBoss Core Services の個別ダウンロードとして利用できるようになりました。これにより、インストールと設定が容易になり、更新の一貫性が保たれます。

24.5.2.1. Apache HTTP Server のインストール

Apache HTTP Server のインストールに関する詳細は、JBoss Core Services の Apache HTTP Server Installation Guide を参照してください。

24.5.3. 外部 Web サーバーからのリクエストの許可

JBoss EAP に AJP、HTTP、 HTTPS などの適切なプロトコルハンドラーが設定されていれば、プロキシーサーバーからのリクエストを許可するための特別な設定は必要ありません。

プロキシーサーバーが mod_jk、mod_proxy、ISAPI、または NSAPI を使用する場合、リクエストを JBoss EAP に送信し、JBoss EAP は応答を返信します。mod_cluster の場合、リクエストをどこにルーティングするかを判断できるようにするため、JBoss EAP が現在の負荷、アプリケーションライフサイクルイベント、ヘルス状態などの情報をプロキシーサーバーへ送信できるようネットワークを設定する必要もあります。mod_cluster プロキシーサーバーの設定に関する詳細は mod_cluster HTTP コネクター を参照してください。

JBoss EAP 設定の更新
以下の手順では、例で使用されているプロトコルやポートを実際に設定する必要があるプロトコルやポートに置き換えてください。

1. Undertow の **instance-id** 属性を設定します。
   外部 Web サーバーは **instance-id** を使用してコネクター設定の JBoss EAP インスタンスを識別します。以下の管理 CLI コマンドを使用して、Undertow で **instance-id** 属性を設定します。

   ```
   /subsystem=undertow:write-attribute(name=instance-id,value=node1)
   ```

   上記の例では、外部 Web サーバーは現在の JBoss EAP インスタンスを **node1** として識別します。

2. 必要なリスナーを Undertow に追加します。
   外部 Web サーバーが JBoss EAP に接続するには、Undertow にリスナーが必要です。各プロトコルにはソケットバインディングに関連する独自のリスナーが必要です。

   注記

   プロトコルやポート設定によってはこの手順は必要でないことがあります。HTTP リスナーはデフォルトの JBoss EAP 設定すべてに設定されており、**ha** または **full-ha** プロファイルを使用している場合、AJP リスナーは設定されています。

   デフォルトのサーバー設定を確認すると、必要なリスナーが設定済みであるかどうかを確認できます。

   ```
   /subsystem=undertow/server=default-server:read-resource
   ```

   リスナーを Undertow に追加するには、ソケットバインディングが必要です。ソケットバインディングは、サーバーまたはサーバーグループによって使用されるソケットバインディンググ

ループに追加されます。以下の管理 CLI コマンドを使用すると **ajp** ソケットバインディングが追加され、ポート **8009** と **standard-sockets** ソケットバインディンググループにバインドされます。

```
/socket-binding-group=standard-sockets/socket-binding=ajp:add(port=8009)
```

以下の管理 CLI コマンドを使用すると **ajp** ソケットバインディングを使用して **ajp** リスナーが Undertow に追加されます。

```
/subsystem=undertow/server=default-server/ajp-listener=ajp:add(socket-binding=ajp)
```

24.6. MOD_CLUSTER HTTP コネクター

mod_cluster コネクターは Apache HTTP Server ベースのロードバランサーです。通信チャネルを使用して、リクエストを Apache HTTP Server からアプリケーションサーバーノードのセットの 1 つに転送します。

他のコネクターと比べて mod_cluster コネクターには複数の利点があります。

- mod_cluster Management Protocol (MCMP) は、JBoss EAP サーバーと mod_cluster が有効な Apache HTTP Server との間の追加的な接続です。HTTP メソッドのカスタムセットを使用してサーバー側の負荷分散係数およびライフサイクルイベントを Apache HTTP Server サーバーに返信するために JBoss EAP サーバーによって使用されます。
- mod_cluster がある Apache HTTP Server を動的に設定すると、手動設定を行わずに JBoss EAP サーバーが負荷分散に参加できます。
- JBoss EAP は、mod_cluster がある Apache HTTP Server に依存せずに負荷分散係数の計算を行います。これにより、負荷分散メトリックが他のコネクターよりも正確になります。
- mod_cluster コネクターにより、アプリケーションライフサイクルを細かく制御できるようになります。各 JBoss EAP サーバーは Web アプリケーションコンテキストライフサイクルイベントを Apache HTTP Server サーバーに転送し、指定コンテキストのルーティングリクエストを開始または停止するよう通知します。これにより、リソースを使用できないことが原因の HTTP エラーがエンドユーザーに表示されないようになります。
- AJP、HTTP、または HTTPS トランスポートを使用できます。

modcluster サブシステムの特定の設定オプションに関する詳細は、ModCluster サブシステムの属性 を参照してください。

24.6.1. Apache HTTP Server の mod_cluster の設定

JBoss Core Services Apache HTTP Server をインストールする場合や JBoss Web Server を使用する場合、mod_cluster モジュールはすでに含まれており、デフォルトでロードされます。

注記

JBoss Web Server バージョン 3.1.0 より、Apache HTTP Server は配布されないようになりました。

以下の手順を参照し、お使いの環境に合った mod_cluster モジュールを設定します。

注記

Red Hat のお客様は Red Hat カスタマーポータルにある Load Balancer Configuration Tool を使用して mod_cluster やその他のコネクターに最適な設定テンプレートを迅速に生成することもできます。このツールを使用するにはログインする必要があります。

mod_cluster の設定
Apache HTTP Server には、mod_cluster モジュールをロードし、基本設定を提供する mod_cluster 設定ファイル **mod_cluster.conf** がすでに含まれています。このファイルに指定されている IP アドレス、ポート、およびその他の設定 (以下参照) は必要に応じて変更できます。

```
# mod_proxy_balancer should be disabled when mod_cluster is used
LoadModule proxy_cluster_module modules/mod_proxy_cluster.so
LoadModule cluster_slotmem_module modules/mod_cluster_slotmem.so
LoadModule manager_module modules/mod_manager.so
LoadModule advertise_module modules/mod_advertise.so

MemManagerFile cache/mod_cluster

<IfModule manager_module>
  Listen 6666
  <VirtualHost *:6666>
    <Directory />
      Require ip 127.0.0.1
    </Directory>
    ServerAdvertise on
    EnableMCPMReceive
    <Location /mod_cluster_manager>
      SetHandler mod_cluster-manager
      Require ip 127.0.0.1
   </Location>
  </VirtualHost>
</IfModule>
```

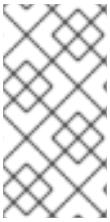
Apache HTTP Server サーバーはロードバランサーとして設定でき、JBoss EAP で実行されている **modcluster** サブシステムと動作します。JBoss EAP が mod_cluster を認識するよう mod_cluster ワーカーノードを設定 する必要があります。

mod_cluster のアドバタイズを無効にし、代わりに静的プロキシーリストを設定する場合は mod_cluster のアドバタイズの無効化 を参照してください。Apache HTTP Server で使用できる mod_cluster 設定オプションの詳細は、Apache HTTP Server の mod_cluster ディレクティブ を参照してください。

mod_cluster の設定に関する詳細は、JBoss Web ServerHTTP Connectors and Load Balancing Guide の**Configure Load Balancing Using Apache HTTP Server and mod_cluster**を参照してください。

24.6.2. mod_cluster のアドバタイズの無効化

デフォルトでは、**modcluster** サブシステムのバランサーはマルチキャスト UDP を使用して可用性をバックグラウンドワーカーにアドバタイズします。アドバタイズを無効にし、代わりにプロキシーリストを使用する場合は、以下の手順に従ってください。



注記

以下の手順の管理 CLI コマンドは、マネージドドメインで **full-ha** プロファイルを使用していることを前提としています。**full-ha** 以外のプロファイルを使用している場合は、コマンドに適切なプロファイル名を使用してください。スタンドアロンサーバーを実行している場合は **/profile=full-ha** を削除してください。

1. Apache HTTP Server 設定を変更します。
   httpd.conf Apache HTTP Server 設定ファイルを編集します。 **EnableMCPMReceive** ディレクティブを使用して、MCPM リクエストをリッスンする仮想ホストに以下の更新を加えてください。

   a. サーバーアドバタイズメントを無効にするディレクティブを追加します。
      ServerAdvertise ディレクティブを **Off** に設定し、サーバーのアドバタイズを無効にします。

      ```
      ServerAdvertise Off
      ```

   b. アドバタイズの頻度を無効にします。
      設定に **AdvertiseFrequency** が指定されている場合は **#** 文字を使用してコメントアウトします。

      ```
      # AdvertiseFrequency 5
      ```

   c. MCPM メッセージの受信機能を有効にします。
      Web サーバーがワーカーノードから MCPM メッセージを受信できるようにするため、必ず **EnableMCPMReceive** ディレクティブが存在するようにしてください。

      ```
      EnableMCPMReceive
      ```

2. **modcluster** サブシステムでアドバタイズを無効にします。
   以下の管理 CLI コマンドを使用してアドバタイズを無効にします。

   ```
   /profile=full-ha/subsystem=modcluster/proxy=default:write-
   attribute(name=advertise,value=false)
   ```

   重要

   必ず次のステップでプロキシーのリストを提供してください。プロキシーのリストが空であるとアドバタイズが無効になりません。

3. JBoss EAP の **modcluster** サブシステムにプロキシーのリストを提供します。
   アドバタイズが無効になっていると **modcluster** サブシステムは自動的にプロキシーを検出できないため、プロキシーのリストを提供する必要があります。

   最初に、適切なソケットバインディンググループにアウトバウンドソケットバインディングを定義します。

   ```
   /socket-binding-group=full-ha-sockets/remote-destination-outbound-socket-
   binding=proxy1:add(host=10.33.144.3,port=6666)
   /socket-binding-group=full-ha-sockets/remote-destination-outbound-socket-
   binding=proxy2:add(host=10.33.144.1,port=6666)
   ```

次に、プロキシーを mod_cluster 設定に追加します。

```
/profile=full-ha/subsystem=modcluster/proxy=default:list-add(name=proxies,value=proxy1)
/profile=full-ha/subsystem=modcluster/proxy=default:list-add(name=proxies,value=proxy2)
```

Apache HTTP Server のバランサーがその存在をワーカーノードにアドバタイズしなくなり、UDP マルチキャストが使用されないようになります。

24.6.3. mod_cluster ワーカーノードの設定

mod_cluster ワーカーノードは、JBoss EAP サーバーで設定されます。このサーバーは、スタンドアロンサーバーまたはマネージドドメインのサーバーグループの一部になります。個別のプロセスが JBoss EAP 内で実行され、クラスターのワーカーノードをすべて管理します。これはマスターと呼ばれます。

マネージドドメインのワーカーノードは、サーバーグループ全体で同じ設定を共有します。スタンドアロンサーバーとして実行しているワーカーノードは個別に設定されます。設定手順は同じです。

- スタンドアロンサーバーは **standalone-ha** または **standalone-full-ha** プロファイルで起動する必要があります。
- マネージドドメインのサーバーグループは **ha** または **full-ha** プロファイルを使用し、 **ha-sockets** または **full-ha-sockets** ソケットバインディンググループを使用する必要があります。JBoss EAP にはこれらの要件を満たし、クラスターが有効になっている other-server-group というサーバーグループが含まれます。

ワーカーノードの設定
この手順の管理 CLI コマンドは、**full-ha** プロファイルのマネージドドメインを使用していることを前提としています。スタンドアロンサーバーを実行している場合は、コマンドの **/profile=full-ha** の部分を削除してください。

1. ネットワークインターフェイスの設定
   デフォルトでは、ネットワークインターフェイスがすべて **127.0.0.1** に設定されます。スタンドアロンサーバーまたはサーバーグループ内の 1 つまたは複数のサーバーをホストする各物理ホストでは、インターフェイスが他のサーバーが見つけることができるパブリック IP アドレスを使用するよう設定する必要があります。

   以下の管理 CLI コマンドを使用して、ご使用の環境に合うよう **management**、**public**、および **unsecure** インターフェイスの外部 IP アドレスを変更します。コマンドの **EXTERNAL_IP_ADDRESS** をホストの実際の外部 IP アドレスに置き換えてください。

   ```
   /interface=management:write-attribute(name=inet-address,value="${jboss.bind.address.management:EXTERNAL_IP_ADDRESS}")
   /interface=public:write-attribute(name=inet-address,value="${jboss.bind.address.public:EXTERNAL_IP_ADDRESS}")
   /interface=unsecure:write-attribute(name=inet-address,value="${jboss.bind.address.unsecure:EXTERNAL_IP_ADDRESS}")
   ```

   サーバーをリロードします。

   ```
   reload
   ```

2. ホスト名を設定します。

マネージドドメインに参加する各ホストに一意なホスト名を設定します。この名前はスレーブ全体で一意になる必要があり、スレーブがクラスターを識別するために使用されるため、使用する名前を覚えておくようにしてください。

a. 適切な **host.xml** 設定ファイルを使用して JBoss EAP のスレーブホストを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml
```

b. 以下の管理 CLI コマンドを使用して、一意なホスト名を設定します。この例では、**slave1** が新しいホスト名として使用されます。

```
/host=EXISTING_HOST_NAME:write-attribute(name=name,value=slave1)
```

ホスト名の設定に関する詳細は、ホスト名の設定 を参照してください。

3. ドメインコントローラーに接続する各ホストを設定します。

注記

このステップはスタンドアロンサーバーには適用されません。

新たに設定されたホストがマネージドドメインに参加する必要がある場合、ローカル要素を削除し、ドメインコントローラーを示すリモート要素ホスト属性を追加する必要があります。

a. 適切な **host.xml** 設定ファイルを使用して JBoss EAP のスレーブホストを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml
```

b. 以下の管理 CLI コマンドを使用して、ドメインコントローラーを設定します。

```
/host=SLAVE_HOST_NAME:write-remote-domain-
controller(host=DOMAIN_CONTROLLER_IP_ADDRESS,port=${jboss.domain.master.po
rt:9990},security-realm="ManagementRealm")
```

これにより、host-slave.xml ファイルの XML が次のように変更されます。

```
<domain-controller>
    <remote host="DOMAIN_CONTROLLER_IP_ADDRESS"
port="${jboss.domain.master.port:9990}" security-realm="ManagementRealm"/>
</domain-controller>
```

詳細は、ホストコントローラーの設定を 参照してください。

4. 各スレーブホストの認証を設定します。
各スレーブサーバーには、ドメインコントローラーまたはスタンドアロンマスターの ManagementRealm で作成されたユーザー名とパスワードが必要です。ドメインコントローラーまたはスタンドアロンマスターで、各ホストに対して **EAP_HOME/bin/add-user.sh** コマンドを実行します。スレーブのホスト名と一致するユーザー名で、各ホストの管理ユーザーを追加します。

秘密の値が提供されるようにするため、必ず最後の質問 (Is this new user going to be used for one AS process to connect to another AS process?) に **yes** と返答してください。

例: add-user.sh スクリプト出力 (抜粋)

```
$ EAP_HOME/bin/add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): a

Username : slave1
Password : changeme
Re-enter Password : changeme
What groups do you want this user to belong to? (Please enter a comma separated list, or
leave blank for none)[  ]:
About to add user 'slave1' for realm 'ManagementRealm'
Is this correct yes/no? yes
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for
server to server {JEB} calls.
yes/no? yes
To represent the user add the following to the server-identities definition <secret
value="SECRET_VALUE" />
```

ここで出力された Base64 でエンコードされた秘密の値 **SECRET_VALUE** をコピーします。この値は次のステップで使用することがあります。

詳細は、JBoss EAP**How To Configure Server Security**の Adding a User to the Master Domain Controller を参照してください。

5. 新しい認証を使用するようスレーブホストのセキュリティーレルムを変更します。
サーバー設定に秘密の値を指定する方法、認証情報ストアまたは vault からパスワードを取得する方法、またはパスワードをシステムプロパティーとして渡す方法のいずれかでパスワードを指定できます。

   - 管理 CLI を使用して、サーバー設定ファイルに Base64 でエンコードされたパスワード値を指定します。
   以下の管理 CLI コマンドを使用して秘密の値を指定します。必ず **SECRET_VALUE** を前のステップの add-user 出力から返された秘密の値に置き換えてください。

     ```
     /host=SLAVE_HOST_NAME/core-service=management/security-
     realm=ManagementRealm/server-identity=secret:add(value="SECRET_VALUE")
     ```

     サーバーをリロードする必要があります。**--host** 引数はスタンドアロンサーバーには適用されません。

     ```
     reload --host=HOST_NAME
     ```

     詳細は、JBoss EAP**How To Configure Server Security**の Configuring the Slave Controllers to Use the Credential を参照してください。

   - ホストを設定し、認証情報ストアからパスワードを取得します。
   秘密の値を認証情報ストアに保存した場合、以下のコマンドを使用してサーバーの秘密の値が認証情報ストアからの値になるよう設定できます。

```
/host=SLAVE_HOST_NAME/core-service=management/security-realm=ManagementRealm/server-identity=secret:add(credential-reference={store=STORE_NAME,alias=ALIAS}
```

サーバーをリロードする必要があります。**--host** 引数はスタンドアロンサーバーには適用されません。

```
reload --host=HOST_NAME
```

詳細は、JBoss EAP**How To Configure Server Security**の Credential Store を参照してください。

- ホストを設定し、vault からパスワードを取得します。
  a. **EAP_HOME/bin/vault.sh** スクリプトを使用してマスクされたパスワードを生成します。以下のような **VAULT::secret::password::VAULT_SECRET_VALUE** 形式の文字列が生成されます。

```
VAULT::secret::password::ODVmYmJjNGMtZDU2ZC00YmNlLWE4ODMtZjQ1NWNmNDU4ZDc1TElORV9CUkVBS3ZhdWx0.
```

注記

vault でパスワードを作成する場合、Base64 エンコードではなくプレーンテキストで指定する必要があります。

  b. 以下の管理 CLI コマンドを使用して秘密の値を指定します。必ず **VAULT_SECRET_VALUE** を前のステップで生成したマスクされたパスワードに置き換えてください。

```
/host=master/core-service=management/security-realm=ManagementRealm/server-identity=secret:add(value="${VAULT::secret::password::VAULT_SECRET_VALUE}")
```

サーバーをリロードする必要があります。**--host** 引数はスタンドアロンサーバーには適用されません。

```
reload --host=HOST_NAME
```

詳細は、JBoss EAP **How To Configure Server Security**の Password Vault を参照してください。

- システムプロパティーとしてパスワードを指定します。
  次の例は、**server.identity.password** をパスワードのシステムプロパティー名として使用します。
  a. サーバー設定ファイルでパスワードのシステムプロパティーを指定します。
  以下の管理 CLI コマンドを使用して、システムプロパティーを使用する秘密のアイデンティティーを設定します。

```
/host=SLAVE_HOST_NAME/core-service=management/security-realm=ManagementRealm/server-identity=secret:add(value="${server.identity.password}")
```

サーバーをリロードする必要があります。**--host** 引数はスタンドアロンサーバーには適用されません。

```
reload --host=master
```

b. サーバーの起動時にシステムプロパティーのパスワードを設定します。
server.identity.password システムプロパティーを設定するには、このプロパティーをコマンドライン引数として渡すか、プロパティーファイルで渡します。

i. プレーンテキストのコマンドライン引数として渡します。
サーバーを起動し、**server.identity.password** プロパティーを渡します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -
Dserver.identity.password=changeme
```

警告

パスワードはプレーンテキストで入力する必要があります。**ps -ef** コマンドを実行すると、このパスワードを確認できます。

ii. プロパティーファイルでプロパティーを設定します。
プロパティーファイルを作成し、キーバリューペアをプロパティーファイルに追加します。 例を以下に示します。

```
server.identity.password=changeme
```

警告

パスワードはプレーンテキストで、このプロパティーファイルにアクセスできるユーザーはパスワードを確認できます。

コマンドライン引数を使用してサーバーを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml --
properties=PATH_TO_PROPERTIES_FILE
```

6. サーバーを再起動します。
ホスト名をユーザー名として使用し、暗号化された文字列をパスワードとして使用してスレーブがマスターに対して認証されます。

スタンドアロンサーバーまたはマネージドドメインのサーバーグループ内のサーバーが mod_cluster ワーカーノードとして設定されます。クラスター化されたアプリケーションをデプロイする場合、セッ

ションはフェイルオーバーのためにすべてのクラスターサーバーに複製され、外部 Web サーバーまたはロードバランサーからのリクエストを許可できます。クラスターの各ノードは、デフォルトで自動検出を使用して他のノードを検出します。

24.6.4. mod_cluster の fail_on_status パラメーターの設定

fail_on_status パラメーターは、クラスターのワーカーノードによって返されるとそのノードの失敗を意味する HTTP ステータスコードをリストします。ロードバランサーはその後のリクエストをクラスターの別のワーカーノードに送信します。失敗したワーカーノードは、ロードバランサーに **STATUS** メッセージを送信するまで **NOTOK** の状態になります。

fail_on_status パラメーターはロードバランサーの **httpd** 設定ファイルに設定する必要があります。**fail_on_status** の HTTP ステータスコードが複数ある場合はコンマで区切って指定します。以下の例は、HTTP ステータスコード **203** および **204** を **fail_on_status** に指定します。

例: fail_on_status の設定

```
ProxyPass / balancer://MyBalancer stickysession=JSESSIONID|jsessionid nofailover=on
failonstatus=203,204
ProxyPassReverse / balancer://MyBalancer
ProxyPreserveHost on
```

24.6.5. クラスター間のトラフィックの移行

JBoss EAP を使用して新しいクラスターを作成した後、アップグレードプロセスの一部として以前のクラスターから新しいクラスターへトラフィックを移行できます。ここでは、停止時間やダウンライムを最小限に抑えてトラフィックを移行する方法について説明します。

- 新しいクラスターの設定。このクラスターを **ClusterNEW** と呼びます。
- 不要となった古いクラスターの設定。このクラスターを **ClusterOLD** と呼びます。

クラスターのアップグレードプロセス - ロードバランシンググループ

1. 前提条件に従って、新しいクラスターを設定します。
2. **ClusterNEW** および **ClusterOLD** の両方で、設定オプション **sticky-session** をデフォルト設定の **true** に設定してください。このオプションを有効にすると、クラスターのクラスターノードへの新しいリクエストはすべてそのクラスターノードに送信されます。

   ```
   /profile=full-ha/subsystem=modcluster/proxy=default:write-attribute(name=sticky-
   session,value=true)
   ```

3. **ClusterOLD** のすべてのクラスターノードは **ClusterOLD** ロードバランシンググループのメンバーであることを仮定し、**load-balancing-group** を **ClusterOLD** に設定します。

   ```
   /profile=full-ha/subsystem=modcluster/proxy=default:write-attribute(name=load-balancing-
   group,value=ClusterOLD)
   ```

4. **mod_cluster** ワーカーノードの設定の手順に従って ClusterNEW のノードを個別に mod_cluster 設定に追加します。また、この手順を使用してロードバランシンググループを **ClusterNEW** に設定します。
   この時点で、以下の簡易例に似た出力が **mod_cluster-manager** コンソールに表示されます。

```
            mod_cluster/<version>

   LBGroup ClusterOLD: [Enable Nodes]  [Disable Nodes]  [Stop Nodes]
      Node node-1-jvmroute (ajp://node1.oldcluster.example:8009):
         [Enable Contexts]  [Disable Contexts]  [Stop Contexts]
         Balancer: qacluster, LBGroup: ClusterOLD, Flushpackets: Off, ..., Load: 100
         Virtual Host 1:
            Contexts:
               /my-deployed-application-context, Status: ENABLED Request: 0 [Disable]
[Stop]

      Node node-2-jvmroute (ajp://node2.oldcluster.example:8009):
         [Enable Contexts]  [Disable Contexts]  [Stop Contexts]
         Balancer: qacluster, LBGroup: ClusterOLD, Flushpackets: Off, ..., Load: 100
         Virtual Host 1:
            Contexts:
               /my-deployed-application-context, Status: ENABLED Request: 0 [Disable]
[Stop]


   LBGroup ClusterNEW: [Enable Nodes]  [Disable Nodes]  [Stop Nodes]
      Node node-3-jvmroute (ajp://node3.newcluster.example:8009):
         [Enable Contexts]  [Disable Contexts]  [Stop Contexts]
         Balancer: qacluster, LBGroup: ClusterNEW, Flushpackets: Off, ..., Load: 100
         Virtual Host 1:
            Contexts:
               /my-deployed-application-context, Status: ENABLED Request: 0 [Disable]
[Stop]

      Node node-4-jvmroute (ajp://node4.newcluster.example:8009):
         [Enable Contexts]  [Disable Contexts]  [Stop Contexts]
         Balancer: qacluster, LBGroup: ClusterNEW, Flushpackets: Off, ..., Load: 100
         Virtual Host 1:
            Contexts:
               /my-deployed-application-context, Status: ENABLED Request: 0 [Disable]
[Stop]
```

5. **ClusterOLD** グループ内に古いアクティブなセッションがあり、新しいセッションは **ClusterOLD** または **CLusterNEW** グループ内に作成されます。次に、現在アクティブなクライアントのセッションにエラーが発生しないようにクラスターノードを停止するため、**ClusterOLD** グループ全体を無効にします。
mod_cluster-manager Web コンソールで LBGroup **ClusterOLD** の **Disable Nodes** リンクをクリックします。

   この後、すでに確立されたセッションに属するリクエストのみが **ClusterOLD** ロードバランシンググループのメンバーにルーティングされます。新しいクライアントのセッションは **ClusterNEW** グループのみに作成されます。**ClusterOLD** グループ内にアクティブなセッションがなくなったら、そのメンバーを安全に削除できます。

注記

Stop Nodes を使用すると、即座にこのドメインへのリクエストのルーティングを停止するようロードバランサーが指示されます。これにより、別のロードバランシンググループへのフェイルオーバーが強制され、**ClusterNEW** と **ClusterOLD** の間にセッションレプリケーションがない場合はクライアントへのセッションデータが損失する原因となります。

デフォルトのロードバランシンググループ
mod_cluster-manager コンソールの LBGroup を確認して、現在の **ClusterOLD** 設定にロードバランシンググループの設定が含まれていない場合でも、**ClusterOLD** ノードの無効化を利用できます。この場合、各 **ClusterOLD** ノードの **Disable Contexts** をクリックします。これらのノードのコンテンツは無効化され、アクティブなセッションがなくなったら削除することができます。新しいクライアントのセッションは、有効なコンテンツを持つノードのみに作成されます (この例では **ClusterOLD** メンバー)。

管理 **CLI** の使用
mod_cluster-manager web コンソールを使用する他に、JBoss EAP 管理 CLI を使用して特定のコンテキストを停止または無効化することもできます。

コンテキストの停止

```
/host=master/server=server-one/subsystem=modcluster:stop-context(context=/my-deployed-application-context, virtualhost=default-host, waittime=0)
```

waittime を **0** に設定してタイムアウトがない状態でコンテキストを停止すると、すべてのリクエストのルーティングを即座に停止するようバランサーに指示を出し、利用できる他のコンテキストへのフェイルオーバーを強制します。

waittime 引数を使用してタイムアウト値を設定すると、このコンテキストでは新しいセッションは作成されませんが、既存のセッションが完了するか、指定のタイムアウト値を経過するまで、既存のセッションはこのノードに引き続き転送されます。**waittime** 引数のデフォルト値は **10** 秒です。

コンテキストの無効化

```
/host=master/server=server-one/subsystem=modcluster:disable-context(context=/my-deployed-application-context, virtualhost=default-host)
```

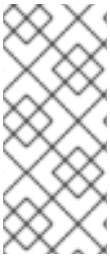
コンテキストを無効にすると、バランサーがこのコンテキストで新しいセッションを作成しないよう指示します。

24.7. APACHE MOD_JK HTTP コネクター

Apache mod_jk は、互換性の維持を目的に提供される HTTP コネクターです。

JBoss EAP は Apache HTTP プロキシーサーバーからのワークロードを許可します。プロキシーサーバーは Web フロントエンドからのクライアントリクエストを許可し、ワークを参加する JBoss EAP サーバーへ渡します。スティッキーセッションが有効な場合、同じクライアントリクエストは常に同じ JBoss EAP サーバーに送信されます (同じサーバーが使用できない場合を除く)。

mod_jk は AJP 1.3 プロトコルを介して通信します。**mod_cluster** または **mod_proxy** には他のプロトコルを使用できます。詳細は HTTP コネクターの概要 を参照してください。



注記

mod_cluster は mod_jk よりも高度なロードバランサーで、推奨される HTTP コネクターです。mod_cluster は mod_jk のすべての機能と、それ以外の追加機能を提供します。JBoss EAP の mod_cluster HTTP コネクターとは違い、Apache mod_jk HTTP コネクターはサーバーまたはサーバーグループのデプロイメントの状態を認識せず、ワークの送信先に順応できません。

詳細は、Apache mod_jk ドキュメント を参照してください。

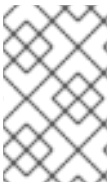
24.7.1. Apache HTTP Server での mod_jk の設定

JBoss Core Services Apache HTTP Server のインストール時または JBoss Web Server の使用時に mod_jk モジュールである **mod_jk.so** はすでに含まれていますが、デフォルトではロードされません。

注記

JBoss Web Server バージョン 3.1.0 より、Apache HTTP Server は配布されないようになりました。

以下の手順に従って、Apache HTTP Server の mod_jk をロードおよび設定します。この手順では、Apache HTTP Server の **httpd**/ ディレクトリーがカレントディレクトリーであることを前提としていますが、このディレクトリーはプラットフォームによって異なります。ご使用のプラットフォームに対応するインストール手順は、JBoss Core Services の **Apache HTTP Server Installation Guide** を参照してください。



注記

Red Hat のお客様は Red Hat カスタマーポータルにある Load Balancer Configuration Tool を使用して mod_jk やその他のコネクターに最適な設定テンプレートを迅速に生成することもできます。このツールを使用するにはログインする必要があります。

1. mod_jk モジュールを設定します。

注記

mod_jk 設定ファイルの例は **conf.d/mod_jk.conf.sample** にあります。独自のファイルを作成せずにこのファイルを使用するには、**.sample** 拡張子を削除し、必要に応じて内容を変更します。

conf.d/mod_jk.conf という新しいファイルを作成します。以下の設定をファイルに追加し、必要に応じて内容を変更します。

```
# Load mod_jk module
# Specify the filename of the mod_jk lib
LoadModule jk_module modules/mod_jk.so

# Where to find workers.properties
JkWorkersFile conf.d/workers.properties

# Where to put jk logs
JkLogFile logs/mod_jk.log
```

```
# Set the jk log level [debug/error/info]
JkLogLevel info

# Select the log format
JkLogStampFormat  "[%a %b %d %H:%M:%S %Y]"

# JkOptions indicates to send SSK KEY SIZE
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories

# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"

# Mount your applications
JkMount /application/* loadbalancer

# Add shared memory.
# This directive is present with 1.2.10 and
# later versions of mod_jk, and is needed for
# for load balancing to work properly
JkShmFile logs/jk.shm

# Add jkstatus for managing runtime data
<Location /jkstatus/>
    JkMount status
    Require ip 127.0.0.1
</Location>
```

注記

JkMount ディレクティブは、Apache HTTP Server が mod_jk モジュールに転送する必要がある URL を指定します。ディレクティブの設定に基づき、mod_jk は受信した URL を適切なワーカーに送信します。直接静的コンテンツに対応し、Java アプリケーションのロードバランサーのみを使用するには、URL パスは **/application/*** である必要があります。mod_jk をロードバランサーとして使用するには、値 /* を使用してすべての URL を mod_jk に転送します。

一般的な mod_jk の設定の他に、このファイルは **mod_jk.so** モジュールをロードするよう指定し、**workers.properties** ファイルの場所を定義します。

2. mod_jk ワーカーノードを設定します。

注記

ワーカー設定ファイルの例は **conf.d/workers.properties.sample** にあります。独自のファイルを作成せずにこのファイルを使用するには、**.sample** 拡張子を削除し、必要に応じて内容を変更します。

conf.d/workers.properties という新しいファイルを作成します。以下の設定をファイルに追加し、必要に応じて内容を変更します。

```
# Define list of workers that will be used
# for mapping requests
worker.list=loadbalancer,status
```

```
# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=node1.mydomain.com
worker.node1.type=ajp13
worker.node1.ping_mode=A
worker.node1.lbfactor=1

# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host=node2.mydomain.com
worker.node2.type=ajp13
worker.node2.ping_mode=A
worker.node2.lbfactor=1

# Load-balancing behavior
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1

# Status worker for managing load balancer
worker.status.type=status
```

mod_jk **workers.properties** ファイルの構文の詳細およびその他の高度な設定オプションの詳細は、mod_jk ワーカープロパティー を参照してください。

3. 任意で JKMountFile ディレクティブを指定します。
mod-jk.conf の JKMount ディレクティブの他に、mod_jk に転送される複数の URL パターンが含まれるファイルを指定できます。

   a. **uriworkermap.properties** ファイルを作成します。

   注記

   URI ワーカーマップ設定ファイルの例は **conf.d/uriworkermap.properties.sample** にあります。独自のファイルを作成せずにこのファイルを使用するには、**.sample** 拡張子を削除し、必要に応じて内容を変更します。

   conf.d/uriworkermap.properties という新しいファイルを作成します。以下の例のように、一致する各 URL パターンの行を追加します。

   ```
   # Simple worker configuration file
   /*=loadbalancer
   ```

   b. **uriworkermap.properties** ファイルを示すよう、設定を更新します。
   conf.d/mod_jk.conf の最後に以下を追加します。

   ```
   # Use external file for mount points.
   # It will be checked for updates each 60 seconds.
   # The format of the file is: /url=worker
   # /examples/*=loadbalancer
   JkMountFile conf.d/uriworkermap.properties
   ```

mod_jk の設定に関する詳細は、 JBoss Web ServerHTTP Connectors and Load Balancing Guide の**Configuring Apache HTTP Server to Load mod_jk**を参照してください。

24.7.2. JBoss EAP が mod_jk と通信するよう設定

JBoss EAP の **undertow** サブシステムは、外部 Web サーバーからのリクエストを許可し、外部 Web サーバーへ返答を返送するために、リスナーを指定する必要があります。mod_jk は AJP プロトコルを使用するため、AJP リスナーを設定する必要があります。

デフォルトの高可用性設定の 1 つ (**ha** または **full-ha**) を使用している場合は、AJP リスナーはすでに設定されています。

手順は 外部 Web サーバーからのリクエストの許可 を参照してください。

24.8. APACHE MOD_PROXY HTTP コネクター

Apache mod_proxy は、AJP、HTTP、および HTTPS プロトコルを介して接続をサポートする HTTP コネクターです。 mod_proxy は負荷分散された設定と負荷分散されていない設定が可能で、スティッキーセッションをサポートします。

mod_proxy モジュールを使用するには、使用するプロトコルに応じて JBoss EAP の **undertow** サブシステムに HTTP、HTTPS または AJP リスナーを設定する必要があります。

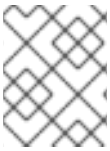
注記

mod_cluster は mod_proxy よりも高度なロードバランサーで、推奨される HTTP コネクターです。mod_cluster は mod_proxy のすべての機能と、それ以外の追加機能を提供します。JBoss EAP の mod_cluster HTTP コネクターとは違い、Apache mod_proxy HTTP コネクターはサーバーまたはサーバーグループのデプロイメントの状態を認識せず、ワークの送信先に順応できません。

詳細は Apache mod_proxy ドキュメント を参照してください。

24.8.1. Apache HTTP Server での mod_proxy の設定

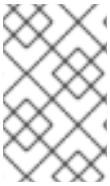
JBoss Core Services Apache HTTP Server をインストールする場合や JBoss Web Server を使用する場合、mod_proxy モジュールはすでに含まれており、デフォルトでロードされます。



注記

JBoss Web Server バージョン 3.1.0 より、Apache HTTP Server は配布されないようになりました。

基本の ロードバランシング または 非ロードバランシング プロキシーを設定するには、以下のセクションを参照してください。この手順では、Apache HTTP Server の **httpd**/ ディレクトリーがカレントディレクトリーであることを前提としていますが、このディレクトリーはプラットフォームによって異なります。ご使用のプラットフォームに対応するインストール手順は、JBoss Core Services の **Apache HTTP Server Installation Guide** を参照してください。また、この手順は JBoss EAP の **undertow** サブシステムに必要な HTTP リスナーが設定されていることを前提としています。



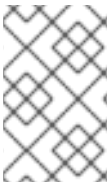
注記

Red Hat のお客様は Red Hat カスタマーポータルにある Load Balancer Configuration Tool を使用して mod_proxy やその他のコネクターに最適な設定テンプレートを迅速に生成することもできます。このツールを使用するにはログインする必要があります。

非ロードバランシングプロキシーの追加
以下の設定を、他の **<VirtualHost>** ディレクティブの直下にある **conf/httpd.conf** ファイルに追加します。値を設定に適切な値に置き換えます。

```
<VirtualHost *:80>
# Your domain name
ServerName YOUR_DOMAIN_NAME

ProxyPreserveHost On

# The IP and port of JBoss
# These represent the default values, if your httpd is on the same host
# as your JBoss managed domain or server

ProxyPass / http://localhost:8080/
ProxyPassReverse / http://localhost:8080/

# The location of the HTML files, and access control information
DocumentRoot /var/www
<Directory /var/www>
Options -Indexes
Order allow,deny
Allow from all
</Directory>
</VirtualHost>
```

ロードバランシングプロキシーの追加



注記

デフォルトの Apache HTTP Server は mod_cluster との互換性がないため、**mod_proxy_balancer.so** モジュールが無効になっています。この作業を行うには、このモジュールをロードし、mod_cluster を無効にする必要があります。

mod_proxy をロードバランサーとして使用し、ワークを複数の JBoss EAP インスタンスに送信するには、以下の設定を **conf/httpd.conf** ファイルに追加する必要があります。IP アドレスの例は以下のようになります。ご使用の環境に適切な値に置き換えてください。

```
<Proxy balancer://mycluster>

Order deny,allow
Allow from all

# Add each JBoss Enterprise Application Server by IP address and port.
# If the route values are unique like this, one node will not fail over to the other.
BalancerMember http://192.168.1.1:8080 route=node1
BalancerMember http://192.168.1.2:8180 route=node2
</Proxy>
```

```
<VirtualHost *:80>
 # Your domain name
 ServerName YOUR_DOMAIN_NAME

 ProxyPreserveHost On
 ProxyPass / balancer://mycluster/

 # The location of the HTML files, and access control information DocumentRoot /var/www
 <Directory /var/www>
  Options -Indexes
  Order allow,deny
  Allow from all
 </Directory>

</VirtualHost>
```

上記の例はすべて HTTP プロトコルを使用して通信します。適切な mod_proxy モジュールをロードすれば AJP または HTTPS プロトコルを使用することもできます。詳細は Apache mod_proxy ドキュメント を参照してください。

スティッキーセッションの有効化
スティッキーセッション を使用すると、クライアントリクエストが特定の JBoss EAP ワーカーに送信された場合に、ワーカーが利用不可能にならない限り、後続のリクエストがすべて同じワーカーに送信されます。これは、ほとんどの場合で推奨される動作です。

mod_proxy のスティッキーセッションを有効にするには、**stickysession** パラメーターを **ProxyPass** ステートメントに追加します。

```
ProxyPass / balancer://mycluster stickysession=JSESSIONID
```

追加のパラメーターを **lbmethod** や **nofailover** などの **ProxyPass** ステートメントに指定できます。使用可能なパラメーターの詳細は、Apache mod_proxy ドキュメント を参照してください。

24.8.2. JBoss EAP が mod_proxy と通信するよう設定

JBoss EAP の **undertow** サブシステムは、外部 Web サーバーからのリクエストを許可し、外部 Web サーバーへ返答を返送するために、リスナーを指定する必要があります。使用するプロトコルによっては、リスナーを設定する必要があることがあります。

HTTP リスナーは JBoss EAP のデフォルト設定に設定されます。デフォルトの高可用性設定である **ha** または **full-ha** のいずれかを使用している場合は、AJP リスナーも事前設定されています。

手順は 外部 Web サーバーからのリクエストの許可 を参照してください。

24.9. MICROSOFT ISAPI コネクター

Internet Server API (ISAPI) は、Microsoft のインターネット情報サービス (IIS) などの Web サーバー用の Digital Server 拡張やフィルターを書き込むために使用される API のセットです。**ISAPI_redirect.dll** は IIS 向けに調整された **mod_jk** の拡張機能です。**ISAPI_redirect.dll** を使用すると、JBoss EAP インスタンスをワーカーノードとしてロードバランサーとして設定できます。

注記

Windows Server および IIS のサポートされる設定については、JBoss Enterprise Application Platform (EAP) 7 でサポートされる設定 を参照してください。

24.9.1. Microsoft IIS が ISAPI コネクターを使用するよう設定

Red Hat カスタマーポータルから ISAPI コネクターをダウンロードします。

1. ブラウザーを開き、Red Hat カスタマーポータルで JBoss の Software Downloads ページにログインします。
2. **Product** ドロップダウンメニューから **Web Connectors** を選択します。
3. **Version** ドロップダウンメニューで最新バージョンの JBoss Core Services を選択します。
4. リストで **Red Hat JBoss Core Services ISAPI Connector**を見つけ、**Download** リンクをクリックします。
5. アーカイブを抽出し、**sbin** ディレクトリーの内容をサーバーの場所にコピーします。以下の手順は、内容が **C:\connectors** にコピーされたことを前提としています。

IIS マネージャー (IIS 7) を使用して IIS リディレクターを設定するには、以下を行います。

1. **Start → Run**とクリックして IIS マネージャーを開き、**inetmgr** と入力します。
2. 左側のツリービューペインで **IIS 7** をデプロイメントします。
3. **ISAPI and CGI Registrations** をダブルクリックし、新しいウインドウで開きます。
4. **Actions** ペインで **Add** をクリックします。**Add ISAPI or CGI Restriction** ウインドウが開きます。
5. 以下の値を指定します。
   - **ISAPI or CGI Path**: **C:\connectors\isapi_redirect.dll**
   - **Description**: **jboss**
   - **Allow extension path to execute** チェックボックスを選択します。
6. **OK** をクリックして **Add ISAPI or CGI Restriction** ウインドウを閉じます。
7. JBoss ネイティブ仮想ディレクトリーの定義
   - **Default Web Site** を右クリックし、**Add Virtual Directory** をクリックします。**Add Virtual Directory** ウインドウが開きます。
   - 以下の値を指定して仮想ディレクトリーを追加します。
     - **Alias**: **jboss**
     - **Physical Path**: **C:\connectors**
   - **OK** をクリックして値を保存し、**Add Virtual Directory** ウインドウを閉じます。
8. JBoss ネイティブ ISAPI リダイレクトフィルターの定義
   - ツリービューペインで **Sites → Default Web Site** とデプロイメントします。

- ツリービューペインで Sites > Default Web Site をクリックします。
- **ISAPI Filters** をダブルクリックします。**ISAPI Filters Features** ビューが表示されます。
- **Actions** ペインで Add をクリックします。**Add ISAPI Filter** ウインドウが表示されます。
- 以下の値を **Add ISAPI Filter** ウインドウに指定します。
  - Filter name: **jboss**
  - Executable: **C:\connectors\isapi_redirect.dll**
- **OK** をクリックして値を保存し、**Add ISAPI Filter** ウインドウを閉じます。

9. ISAPI-dll ハンドラーの有効化
   - ツリービューペインの **IIS 7** をダブルクリックします。**IIS 7 Home Features View**が開きます。
   - **Handler Mappings** をダブルクリックします。**Handler Mappings Features View** が表示されます。
   - **Group by** コンボボックスで **State** を選択します。**Handler Mappings** が **Enabled and Disabled Groups** に表示されます。
   - **ISAPI-dll** を見つけます。**Disabled** グループにある場合は右クリックし、**Edit Feature Permissions** を選択します。
   - 以下のパーミッションを有効にします。
     - Read
     - Script
     - Execute
   - **OK** をクリックして値を保存し、**Edit Feature Permissions** ウインドウを閉じます。

これで、ISAPI コネクターを使用するよう Microsoft IIS が設定されます。

24.9.2. ISAPI コネクターがクライアントリクエストを JBoss EAP に送信するよう設定

このタスクでは、JBoss EAP サーバーのグループが ISAPI コネクターからのリクエストを受け入れるよう設定します。ロードバランシングまたは高可用性フェイルオーバーの設定は含まれません。

この設定は IIS サーバーで行われ、外部 Web サーバーからのリクエストを許可 するよう JBoss EAP が設定されていることを前提としています。また、IIS への完全な管理者アクセスが必要で、IIS が ISAPI コネクターを使用するよう設定 されている必要があります。

プロパティーファイルの作成およびリダイレクトの設定

1. ログ、プロパティーファイル、およびロックファイルを格納するディレクトリーを作成します。
   以下の手順では、ディレクトリー **C:\connectors** の使用を前提としています。異なるディレクトリーを使用する場合は、適切に手順を変更してください。
2. **isapi_redirect.properties** ファイルを作成します。

C:\connectors\isapi_redirect.properties という新しいファイルを作成します。このファイルに次の内容をコピーします。

```
# Configuration file for the ISAPI Connector
# Extension uri definition
extension_uri=/jboss/isapi_redirect.dll

# Full path to the log file for the ISAPI Connector
log_file=c:\connectors\isapi_redirect.log

# Log level (debug, info, warn, error or trace)
log_level=info

# Full path to the workers.properties file
worker_file=c:\connectors\workers.properties

# Full path to the uriworkermap.properties file
worker_mount_file=c:\connectors\uriworkermap.properties

#Full path to the rewrite.properties file
rewrite_rule_file=c:\connectors\rewrite.properties
```

rewrite.properties ファイルを使用しない場合は、行の先頭に **#** 文字を記入して最後の行をコメントアウトします。

3. **uriworkermap.properties** ファイルの作成
uriworkermap.properties ファイルには、デプロイされたアプリケーション URL と、それらへの要求を処理するワーカー間のマッピングが含まれます。以下のサンプルファイルはファイルの構文を示しています。**uriworkermap.properties** ファイルを **C:\connectors** に格納します。

```
# images and css files for path /status are provided by worker01
/status=worker01
/images/*=worker01
/css/*=worker01

# Path /web-console is provided by worker02
# IIS (customized) error page is used for http errors with number greater or equal to 400
# css files are provided by worker01
/web-console/*=worker02;use_server_errors=400
/web-console/css/*=worker01

# Example of exclusion from mapping, logo.gif won't be displayed
# !/web-console/images/logo.gif=*

# Requests to /app-01 or /app-01/something will be routed to worker01
/app-01|/*=worker01

# Requests to /app-02 or /app-02/something will be routed to worker02
/app-02|/*=worker02
```

4. **workers.properties** ファイルを作成します。
workers.properties ファイルには、ワーカーラベルとサーバーインスタンス間のマッピング定義が含まれます。このファイルは、 Apache mod_jk ワーカープロパティー 設定で使用される同じファイルの構文を使用します。

以下は **workers.properties** ファイルの例になります。ワーカー名、**worker01**、および **worker02** は、JBoss EAP の **undertow** サブシステムで設定 された **instance-id** に一致する必要があります。

このファイルを **C:\connectors** ディレクトリーに格納してください。

```
# An entry that lists all the workers defined
worker.list=worker01, worker02

# Entries that define the host and port associated with these workers

# First JBoss EAP server definition, port 8009 is standard port for AJP in EAP
worker.worker01.host=127.0.0.1
worker.worker01.port=8009
worker.worker01.type=ajp13

# Second JBoss EAP server definition
worker.worker02.host=127.0.0.100
worker.worker02.port=8009
worker.worker02.type=ajp13
```

5. **rewrite.properties** ファイルを作成します。
rewrite.properties ファイルには、特定のアプリケーションの単純な URL 書き換えルールが含まれます。以下の例で示されているように、書き換えられたパスは名前と値のペアを使用して指定されます。このファイルを **C:\connectors** ディレクトリーに格納してください。

```
#Simple example
# Images are accessible under abc path
/app-01/abc/=/app-01/images/
```

6. **net stop** および **net start** コマンドを使用して IIS サーバーを再起動します。

```
C:\> net stop was /Y
C:\> net start w3svc
```

アプリケーションごとに、設定した特定の JBoss EAP サーバーにクライアントリクエストを送信するよう IIS サーバーが設定されます。

24.9.3. ISAPI コネクターがクライアントリクエストを複数の JBoss EAP サーバーで分散するよう設定

この設定は、指定する JBoss EAP サーバー全体でクライアントリクエストを分散します。この設定は IIS サーバーで行われ、外部 Web サーバーからのリクエストを許可 するよう JBoss EAP が設定されていることを前提としています。また、IIS への完全な管理者アクセスが必要で、IIS が ISAPI コネクターを使用するよう設定 されている必要があります。

複数のサーバー間でのクライアント要求の分散

1. ログ、プロパティーファイル、およびロックファイルを格納するディレクトリーを作成します。
以下の手順では、ディレクトリー **C:\connectors** の使用を前提としています。異なるディレクトリーを使用する場合は、適切に手順を変更してください。

2. **isapi_redirect.properties** ファイルを作成します。

C:\connectors\isapi_redirect.properties という新しいファイルを作成します。このファイルに次の内容をコピーします。

```
# Configuration file for the ISAPI Connector
# Extension uri definition
extension_uri=/jboss/isapi_redirect.dll

# Full path to the log file for the ISAPI Connector
log_file=c:\connectors\isapi_redirect.log

# Log level (debug, info, warn, error or trace)
log_level=info

# Full path to the workers.properties file
worker_file=c:\connectors\workers.properties

# Full path to the uriworkermap.properties file
worker_mount_file=c:\connectors\uriworkermap.properties

#OPTIONAL: Full path to the rewrite.properties file
rewrite_rule_file=c:\connectors\rewrite.properties
```

rewrite.properties ファイルを使用しない場合は、行の先頭に **#** 文字を記入して最後の行をコメントアウトします。

3. **uriworkermap.properties** ファイルを作成します。
uriworkermap.properties ファイルには、デプロイされたアプリケーション URL と、それらへの要求を処理するワーカー間のマッピングが含まれます。以下のサンプルファイルは負荷分散が設定されたファイルの構文を示しています。ワイルドカード (*) はさまざまな URL サブディレクトリーのすべてのリクエストを router という名前のロードバランサーに送信します。ロードバランサーの設定は次のステップで説明します。

   uriworkermap.properties ファイルを **C:\connectors** に格納します。

```
# images, css files, path /status and /web-console will be
# provided by nodes defined in the load-balancer called "router"
/css/*=router
/images/*=router
/status=router
/web-console|/*=router

# Example of exclusion from mapping, logo.gif won't be displayed
# !/web-console/images/logo.gif=*

# Requests to /app-01 and /app-02 will be routed to nodes defined
# in the load-balancer called "router"
/app-01|/*=router
/app-02|/*=router

# mapping for management console, nodes in cluster can be enabled or disabled here
/jkmanager|/*=status
```

4. **workers.properties** ファイルを作成します。

workers.properties ファイルには、ワーカーラベルとサーバーインスタンス間のマッピング定義が含まれます。このファイルは、 Apache mod_jk ワーカープロパティー 設定で使用される同じファイルの構文を使用します。

以下は **workers.properties** ファイルの例になります。ロードバランサーはファイルの末尾付近に設定され、ワーカー **worker01** および **worker02** で設定されます。これらのワーカーは、JBoss EAP の **undertow** サブシステム で設定された **instance-id** に一致する必要があります。

このファイルを **C:\connectors** ディレクトリーに格納してください。

```
# The advanced router LB worker
worker.list=router,status

# First EAP server definition, port 8009 is standard port for AJP in EAP
#
# lbfactor defines how much the worker will be used.
# The higher the number, the more requests are served
# lbfactor is useful when one machine is more powerful
# ping_mode=A – all possible probes will be used to determine that
# connections are still working

worker.worker01.port=8009
worker.worker01.host=127.0.0.1
worker.worker01.type=ajp13
worker.worker01.ping_mode=A
worker.worker01.socket_timeout=10
worker.worker01.lbfactor=3

# Second EAP server definition
worker.worker02.port=8009
worker.worker02.host=127.0.0.100
worker.worker02.type=ajp13
worker.worker02.ping_mode=A
worker.worker02.socket_timeout=10
worker.worker02.lbfactor=1

# Define the LB worker
worker.router.type=lb
worker.router.balance_workers=worker01,worker02

# Define the status worker for jkmanager
worker.status.type=status
```

5. **rewrite.properties** ファイルを作成します。
rewrite.properties ファイルには、特定のアプリケーションの単純な URL 書き換えルールが含まれます。以下の例で示されているように、書き換えられたパスは名前と値のペアを使用して指定されます。このファイルを **C:\connectors** ディレクトリーに格納してください。

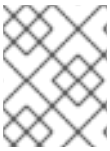
```
#Simple example
# Images are accessible under abc path
/app-01/abc/=/app-01/images/
Restart the IIS server.
```

```
Restart your IIS server by using the net stop and net start commands.
C:\> net stop was /Y
C:\> net start w3svc
```

IIS サーバーは、**workers.properties** ファイルで参照された JBoss EAP サーバーにクライアントリクエストを送信し、サーバー間で負荷を **1:3** の比率で分散するよう設定されます。この比率は、各サーバーに割り当てられた負荷分散係数 **lbfactor** から派生します。

24.10. ORACLE NSAPI コネクター

Netscape Server API (NSAPI) は、拡張機能をサーバーに実装するために Oracle iPlanet Web Server (旧名 Netscape Web Server) によって提供される API です。これらの拡張機能はサーバープラグインと呼ばれます。NSAPI コネクターは、Oracle iPlanet Web Server 向けに調整された mod_jk の拡張機能である **nsapi_redirector.so** で使用されます。NSAPI コネクターを使用すると、JBoss EAP インスタンスをワーカーノード、Oracle iPlanet Web Server をロードバランサーとして設定できます。



注記

Solaris および Oracle iPlanet Web Server のサポートされる設定については、JBoss Enterprise Application Platform (EAP) 7 でサポートされる設定 を参照してください。

24.10.1. iPlanet Web Server が NSAPI コネクターを使用するよう設定

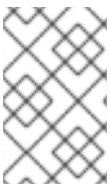
前提条件

- ワーカーとして動作する各サーバーに JBoss EAP がインストールおよび設定されます。

Red Hat カスタマーポータルから NSAPI コネクターをダウンロードします。

1. ブラウザーを開き、Red Hat カスタマーポータルで JBoss の Software Downloads ページにログインします。
2. **Product** ドロップダウンメニューから **Web Connectors** を選択します。
3. **Version** ドロップダウンメニューで最新バージョンの JBoss Core Services を選択します。
4. システムのプラットフォームとアーキテクチャーに対応する **Red Hat JBoss Core Services NSAPI Connector** を見つけ、**Download** リンクをクリックします。
5. **lib/** または **lib64/** ディレクトリーにある **nsapi_redirector.so** ファイルを、**IPLANET_CONFIG/lib/** または **IPLANET_CONFIG/lib64/** ディレクトリーにデプロイメントします。

NSAPI コネクターを設定します。



注記

これらの手順では、**IPLANET_CONFIG** は Oracle iPlanet の設定ディレクトリーを意味します (通常 **/opt/oracle/webserver7/config/** になります)。Oracle iPlanet 設定ディレクトリーが異なる場合は、適切に手順を変更してください。

1. サーブレットマッピングを無効にします。

IPLANET_CONFIG/default.web.xml ファイルを開き、**Built In Server Mappings**という見出しのセクションを見つけます。次の 3 つのサーブレットを XML コメント文字 (**<!--** および **-->**) で囲み、これらのサーブレットへのマッピングを無効にします。

- default
- invoker
- jsp

以下の設定例は、無効にされたマッピングを示しています。

```
<!-- ============== Built In Servlet Mappings =============== -->
<!-- The servlet mappings for the built in servlets defined above. -->
<!-- The mapping for the default servlet -->
<!--servlet-mapping>
 <servlet-name>default</servlet-name>
 <url-pattern>/</url-pattern>
</servlet-mapping-->
<!-- The mapping for the invoker servlet -->
<!--servlet-mapping>
 <servlet-name>invoker</servlet-name>
 <url-pattern>/servlet/*</url-pattern>
</servlet-mapping-->
<!-- The mapping for the Jakarta Server Pages servlet -->
<!--servlet-mapping>
 <servlet-name>jsp</servlet-name>
 <url-pattern>*.jsp</url-pattern>
</servlet-mapping-->
```

ファイルを保存し、終了します。

2. iPlanet Web Server が NSAPI コネクターモジュールをロードするよう設定します。
IPLANET_CONFIG/magnus.conf ファイルの最後に次の行を追加し、設定に合わせてファイルパスを変更します。これらの行は、**nsapi_redirector.so** モジュールと、ワーカーおよびそのプロパティーがリストされた **workers.properties** ファイルの場所を定義します。

```
Init fn="load-modules" funcs="jk_init,jk_service" shlib="/lib/nsapi_redirector.so" shlib_flags="
(global|now)"

Init fn="jk_init" worker_file="IPLANET_CONFIG/connectors/workers.properties"
log_level="info" log_file="IPLANET_CONFIG/connectors/nsapi.log"
shm_file="IPLANET_CONFIG/connectors/tmp/jk_shm"
```

上記の設定は 32 ビットアーキテクチャー向けです。64 ビット Solaris を使用している場合は、文字列 **lib/nsapi_redirector.so** を **lib64/nsapi_redirector.so** に変更します。

ファイルを保存し、終了します。

3. NSAPI コネクターを設定します。
負荷分散のない基本設定または負荷分散設定向けに NSAPI コネクターを設定できます。以下のいずれかのオプションを選択します。 その後、設定が完了します。

- NSAPI コネクターがクライアントリクエストを JBoss EAP に送信するよう設定
- NSAPI コネクターがクライアントリクエストを複数の JBoss EAP サーバーで分散するよう設定

24.10.2. NSAPI コネクターがクライアントリクエストを JBoss EAP に送信するよう設定

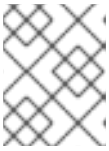
このタスクでは、NSAPI コネクターが負荷分散またはフェイルオーバーなしでクライアントリクエストを JBoss EAP サーバーにリダイレクトするよう設定します。リダイレクトは、デプロイメントごとに (つまり、URL ごとに) 行われます。

重要

このタスクを続行するには、NSAPI コネクターが設定 されている必要があります。

基本的な **HTTP** コネクターの設定

1. JBoss EAP サーバーにリダイレクトする URL パスを定義します。



注記

IPLANET_CONFIG/obj.conf では、前の行から継続する行以外は、行の最初にスペースを挿入しないでください。

IPLANET_CONFIG/obj.conf ファイルを編集します。**<Object name="default">**で始まるセクションを見つけ、一致する各 URL パターンを次のサンプルファイルで示された形式で追加します。文字列 **jknsapi** は、次の手順で定義される HTTP コネクターを示します。例は、ワイルドカードを使用したパターン一致を示しています。

```
<Object name="default">
[...]
NameTrans fn="assign-name" from="/status" name="jknsapi"
NameTrans fn="assign-name" from="/images(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/css(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/nc(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/jmx-console(|/*)" name="jknsapi"
</Object>
```

2. 各パスを提供するワーカーを定義します。
IPLANET_CONFIG/obj.conf ファイルの編集を続行します。編集したセクションの終了タグのすぐ後に、**</Object>** を追加します。

```
<Object name="jknsapi">
ObjectType fn=force-type type=text/plain
Service fn="jk_service" worker="worker01" path="/status"
Service fn="jk_service" worker="worker02" path="/nc(/*)"
Service fn="jk_service" worker="worker01"
</Object>
```

上記の例は、URL パス **/status** へのリクエストを **worker01** という名前のワーカーにリダイレクトし、**/nc/** 以下のすべての URL パスを **worker02** という名前のワーカーにリダイレクトします。3 番目の行は、前の行で一致しない **jknsapi** オブジェクトに割り当てられたすべての URL が **worker01** に提供されることを示しています。

ファイルを保存し、終了します。

3. ワーカーとその属性を定義します。

IPLANET_CONFIG/connectors/ ディレクトリーに **workers.properties** というファイルを作成します。以下の内容をそのファイルに貼り付けし、お使いの環境に合わせて変更します。

```
# An entry that lists all the workers defined
worker.list=worker01, worker02

# Entries that define the host and port associated with these workers
worker.worker01.host=127.0.0.1
worker.worker01.port=8009
worker.worker01.type=ajp13

worker.worker02.host=127.0.0.100
worker.worker02.port=8009
worker.worker02.type=ajp13
```

workers.properties ファイルは Apache mod_jk と同じ構文を使用します。

ファイルを保存し、終了します。

4. iPlanet Web Server の再起動
以下のコマンドを実行し、iPlanet Web Server を再起動します。

```
IPLANET_CONFIG/../bin/stopserv
IPLANET_CONFIG/../bin/startserv
```

iPlanet Web Server が、JBoss EAP のデプロイメントに設定した URL へクライアントリクエストを送信します。

24.10.3. NSAPI コネクターがクライアントリクエストを複数の JBoss EAP サーバーで分散するよう設定

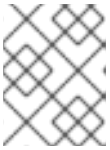
このタスクは、負荷分散の設定でクライアントリクエストを JBoss EAP サーバーに送信するよう NSAPI コネクターを設定します。

重要

このタスクを続行するには、NSAPI コネクターが設定 されている必要があります。

ロードバランシングのコネクターの設定

1. JBoss EAP サーバーにリダイレクトする URL パスを定義します。



注記

IPLANET_CONFIG/obj.conf では、前の行から継続する行以外は、行の最初にスペースを挿入しないでください。

IPLANET_CONFIG/obj.conf ファイルを編集します。**<Object name="default">** で始まるセクションを見つけ、一致する各 URL パターンを次のサンプルファイルで示された形式で追加します。文字列 **jknsapi** は、次の手順で定義される HTTP コネクターを示します。例は、ワイルドカードを使用したパターン一致を示しています。

```
<Object name="default">
```

```
[...]
NameTrans fn="assign-name" from="/status" name="jknsapi"
NameTrans fn="assign-name" from="/images(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/css(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/nc(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/jmx-console(|/*)" name="jknsapi"
NameTrans fn="assign-name" from="/jkmanager/*" name="jknsapi"
</Object>
```

2. 各パスを提供するワーカーを定義します。
IPLANET_CONFIG/obj.conf ファイルの編集を続行します。前の手順で変更したセクションの終了タグ (**</Object>**) のすぐ後に、以下の新しいセクションを追加し、必要に応じて変更します。

```
<Object name="jknsapi">
ObjectType fn=force-type type=text/plain
Service fn="jk_service" worker="status" path="/jkmanager(/*)"
Service fn="jk_service" worker="router"
</Object>
```

この **jksnapi** オブジェクトは、**default** オブジェクトの **name="jksnapi"** マッピングにマップされた各パスを提供するために使用されるワーカーノードを定義します。**/jkmanager/*** に一致する URL 以外のすべてが、**router** という名前のワーカーにリダイレクトされます。

3. ワーカーとその属性を定義します。
workers.properties という名前のファイルを **IPLANET_CONFIG/connector/** で作成します。以下の内容をそのファイルに貼り付けし、お使いの環境に合わせて変更します。

```
# The advanced router LB worker
# A list of each worker
worker.list=router,status

# First JBoss EAP server
# (worker node) definition.
# Port 8009 is the standard port for AJP
#

worker.worker01.port=8009
worker.worker01.host=127.0.0.1
worker.worker01.type=ajp13
worker.worker01.ping_mode=A
worker.worker01.socket_timeout=10
worker.worker01.lbfactor=3

# Second JBoss EAP server
worker.worker02.port=8009
worker.worker02.host=127.0.0.100
worker.worker02.type=ajp13
worker.worker02.ping_mode=A
worker.worker02.socket_timeout=10
worker.worker02.lbfactor=1

# Define the load-balancer called "router"
worker.router.type=lb
worker.router.balance_workers=worker01,worker02
```

```
# Define the status worker
worker.status.type=status
```

workers.properties ファイルは Apache mod_jk と同じ構文を使用します。

ファイルを保存し、終了します。

4. iPlanet Web Server 7.0 を再起動します。

```
IPLANET_CONFIG/../bin/stopserv
IPLANET_CONFIG/../bin/startserv
```

iPlanet Web Server は、設定した URL パターンを負荷分散設定の JBoss EAP サーバーにリダイレクトします。

付録A 参考資料

A.1. サーバーランタイム引数

アプリケーションサーバーの起動スクリプトは実行時に引数とスイッチを受け入れます。そのため、**standalone.xml**、**domain.xml**、および **host.xml** 設定ファイルに定義されていない他の設定でサーバーを起動できます。

他の設定には、ソケットバインディングの代替セットを持つサーバーの起動や 2 次設定が含まれていることがあります。

help スイッチ **-h** または **--help** を起動時に渡すと、利用可能なパラメーターのリストを使用できます。

表A.1 ランタイムスイッチおよび引数

引数またはスイッチ	操作モード	説明
--admin-only	Standalone	サーバーの実行タイプを **ADMIN_ONLY** に設定します。これにより管理インターフェイスが開かれ、管理リクエストが許可されますが、他のランタイムサービスは起動されず、エンドユーザーのリクエストは許可されません。この代わりに **--start-mode=admin-only** を使用することが推奨されます。
--admin-only	Domain	ホストコントローラーの実行タイプを **ADMIN_ONLY** に設定します。 これにより管理インターフェイスが開かれ、管理リクエストが許可されますが、サーバーは起動しません。 ホストコントローラーがドメインのマスターである場合はスレーブホストコントローラーからの受信接続が許可されます。
-b=<value>、 -b <value>	Standalone、Domain	パブリックインターフェイスのバインドアドレスを設定するために使用される **jboss.bind.address** システムプロパティーを設定します。値の指定がない場合は、デフォルトで **127.0.0.1** が指定されます。他のインターフェイスにバインドアドレスを設定するには **-b<interface>=<value>** エントリーを確認します。
-b<interface>=<value>	Standalone、Domain	システムプロパティー **jboss.bind.address.<interface>** を指定の値に設定します。**-bmanagement=IP_ADDRESS** など。
--backup	Domain	このホストがドメインコントローラーではない場合でも永続ドメイン設定のコピーを保持します。
-c=<config>、 -c <config>	Standalone	使用するサーバー設定ファイルの名前。デフォルトは **standalone.xml** です。

引数またはスイッチ	操作モード	説明
-c=<config>、-c <config>	Domain	使用するサーバー設定ファイルの名前。デフォルトは **domain.xml** です。
--cached-dc	Domain	ホストがドメインコントローラーではなく、起動時にドメインコントローラーに接続できない場合、ローカルでキャッシュされたドメイン設定のコピーを使用してブートします。
--debug [<port>]	Standalone	オプションの引数を用いてデバッグモードを有効にし、ポートを指定します。起動スクリプトがサポートする場合のみ動作します。
-D<name>[=<value>]	Standalone、Domain	システムプロパティーを設定します。
--domain-config=<config>	Domain	使用するサーバー設定ファイルの名前。デフォルトは **domain.xml** です。
--git-repo	Standalone	サーバー設定データの管理および永続化に使用される Git リポジトリーの場所。これは、ローカルで保存する場合は **local** を指定し、リモートの場合はリモートリポジトリーへの URL を指定します。
--git-branch	Standalone	使用する Git リポジトリーのブランチまたはタグ名。ブランチまたはタグ名が存在**しない**と作成されないため、この引数には既存のブランチまたはタグ名を指定する必要があります。タグ名を使用する場合、リポジトリーを detached HEAD 状態にし、今後のコミットがブランチにアタッチされないようにします。タグ名は読み取り専用で、通常複数のノード全体で設定をレプリケートする必要があるときに使用されます。
--git-auth	Standalone	Elytron 設定ファイルへの URL には、リモート Git リポジトリーへの接続時に使用される認証情報が含まれています。この引数は、Git リポジトリーに認証が必要な場合に必要となります。Elytron は SSH をサポートしません。したがって、パスワードなしでの秘密鍵を使用したデフォルトの SSH 認証のみがサポートされます。この引数は **local** リポジトリーとは使用されません。
-h、--help	Standalone、Domain	ヘルプメッセージを表示し、終了します。
--host-config=<config>	Domain	使用するホスト設定ファイルの名前。デフォルトは **host.xml** です。

引数またはスイッチ	操作モード	説明
--interprocess-hc-address=<address>	Domain	ホストコントローラーがプロセスコントローラーからの通信をリッスンしなければならないアドレス。
--interprocess-hc-port=<port>	Domain	ホストコントローラーがプロセスコントローラーからの通信をリッスンしなければならないポート。
--master-address=<address>	Domain	システムプロパティー **jboss.domain.master.address** を指定の値に設定します。デフォルトのスレーブホストコントローラー設定では、マスターホストコントローラーのアドレスを設定するために使用されます。
--master-port=<port>	Domain	システムプロパティー **jboss.domain.master.port** を指定の値に設定します。デフォルトのスレーブホストコントローラー設定では、マスターホストコントローラーによるネイティブ管理の通信で使用されるポートを設定するために使用されます。
--read-only-server-config=<config>	Standalone	使用するサーバー設定ファイルの名前。元のファイルは上書きされないため、**--server-config** および **-c** とは異なります。
--read-only-domain-config=<config>	Domain	使用するドメイン設定ファイルの名前。最初のファイルは上書きされないため、**--domain-config** および **-c** とは異なります。
--read-only-host-config=<config>	Domain	使用するホスト設定ファイルの名前。最初のファイルは上書きされないため、**--host-config** とは異なります。
-P=<url>、-P <url>、--properties=<url>	Standalone、Domain	該当する URL からシステムプロパティーをロードします。
--pc-address=<address>	Domain	プロセスコントローラーが制御するプロセスからの通信をリッスンするアドレス。
--pc-port=<port>	Domain	プロセスコントローラーが制御するプロセスからの通信をリッスンするポート。
-S<name>[=<value>]	Standalone	セキュリティープロパティーを設定します。
-secmgr	Standalone、Domain	セキュリティーマネージャーがインストールされた状態でサーバーを実行します。

引数またはスイッチ	操作モード	説明
--server-config=<config>	Standalone	使用するサーバー設定ファイルの名前。デフォルトは **standalone.xml** です。
--start-mode=<mode>	Standalone	サーバーの起動モードを設定します。このオプションは、 **--admin-only** と併用できません。有効な値は以下のとおりです。 ● **normal**: サーバーは通常どおりに起動します。 ● **admin-only**: サーバーは管理インターフェイスのみを開き、管理リクエストを許可しますが、他のランタイムサービスは起動せず、エンドユーザーのリクエストを許可しません。 ● **suspend**: サーバーは中断モードで起動され、再開するまでリクエストに対処しません。
-u=<value>、 -u <value>	Standalone、 Domain	設定ファイルの socket-binding 要素のマルチキャストアドレスを設定するために使用される **jboss.default.multicast.address** システムプロパティーを設定します。値の指定がない場合はデフォルトで **230.0.0.4** が指定されます。
-v、 -V、 --version	Standalone、 Domain	アプリケーションサーバーのバージョンを表示し、終了します。

警告

JBoss EAP に同梱される設定ファイルは、スイッチ (**-b**、**-u** など) を処理するよう設定されます。スイッチによって制御されるシステムプロパティーを使用しないよう設定ファイルを変更した場合は、実行するコマンドにスイッチを追加しても効果はありません。

A.2. RPM サービス設定ファイル

JBoss EAP の RPM インストールには、ZIP またはインストーラーインストールよりも 2 つ多い設定ファイルが含まれています。これらのファイルは、JBoss EAP の起動環境を指定するために、サービス初期化スクリプトによって使用されます。これらのサービス設定ファイルの場所は、Red Hat Enterprise Linux 6 と Red Hat Enterprise Linux 7 以降では異なります。

重要

Red Hat Enterprise Linux 7 以降では、RPM サービス設定ファイルは **systemd** を使用してロードされるため、変数式は拡張されません。

表A.2 Red Hat Enterprise Linux 6 の RPM 設定ファイル

File	説明
/etc/sysconfig/eap7-standalone	Red Hat Enterprise Linux 6 のスタンドアロン JBoss EAP サーバーに固有する設定
/etc/sysconfig/eap7-domain	Red Hat Enterprise Linux 6 でマネージドドメインとして実行されている JBoss EAP に固有する設定

表A.3 Red Hat Enterprise Linux 7 以降の RPM 設定ファイル

File	説明
/etc/opt/rh/eap7/wildfly/eap7-standalone.conf	Red Hat Enterprise Linux 7 以降のスタンドアロン JBoss EAP サーバーに固有する設定
/etc/opt/rh/eap7/wildfly/eap7-domain.conf	Red Hat Enterprise Linux 7 以降でマネージドドメインとして実行されている JBoss EAP に固有する設定

A.3. RPM サービス設定プロパティー

以下の表は、JBoss EAP RPM サービスで使用できる設定プロパティーと、そのデフォルト値のリストになります。

注記

同じ名前のプロパティーが RPM サービス設定ファイル (例: **/etc/sysconfig/eap7-standalone**) と JBoss EAP 起動設定ファイル (例: **EAP_HOME/bin/standalone.conf**) にある場合、JBoss EAP 起動設定ファイルのプロパティーの値が優先されます。このようなプロパティーの 1 つが **JAVA_HOME** です。

表A.4 RPM サービス設定プロパティー

プロパティー	説明
JAVA_HOME	Java Runtime Environment がインストールされたディレクトリー。 デフォルト値: **/usr/lib/jvm/jre**
JAVAPTH	Java 実行可能ファイルがインストールされたパス。 デフォルト値: **$JAVA_HOME/bin**

プロパティー	説明
WILDFLY_STARTUP_WAIT	start または restart コマンドを受け取った後にサーバーが正常に起動されたことを確認するまで、初期化スクリプトが待機する秒数。このプロパティーは、Red Hat Enterprise Linux 6 のみに適用されます。 デフォルト値: **60**
WILDFLY_SHUTDOWN_WAIT	stop または restart コマンドの受信時、続行する前に初期化スクリプトがサーバーのシャットダウンを待機する秒数。このプロパティーは、Red Hat Enterprise Linux 6 のみに適用されます。 デフォルト値: **20**
WILDFLY_CONSOLE_LOG	CONSOLE ログハンドラーがリダイレクトされるファイル。 デフォルト値: スタンドアロンサーバーの場合は **/var/opt/rh/eap7/log/wildfly/standalone/console.log**、マネージドドメインの場合は **/var/opt/rh/eap7/log/wildfly/domain/console.log**
WILDFLY_SH	JBoss EAP サーバーを起動するために使用されるスクリプト。 デフォルト値: スタンドアロンサーバーの場合は **/opt/rh/eap7/root/usr/share/wildfly/bin/standalone.sh**、マネージドドメインの場合は **/opt/rh/eap7/root/usr/share/wildfly/bin/domain.sh**
WILDFLY_SERVER_CONFIG	使用するサーバー設定ファイル。 このプロパティーにはデフォルト値がありません。開始時に **standalone.xml** または **domain.xml** を定義できます。
WILDFLY_HOST_CONFIG	マネージドドメインでは、このプロパティーによってユーザーはホスト設定ファイル (**host.xml** など) を指定できます。デフォルトとして設定されている値はありません。
WILDFLY_MODULEPATH	JBoss EAP モジュールディレクトリーのパス。 デフォルト値: **/opt/rh/eap7/root/usr/share/wildfly/modules**
WILDFLY_BIND	パブリックインターフェイスのバインドアドレスを設定するために使用される **jboss.bind.address** システムプロパティーを設定します。値の指定がない場合、**0.0.0.0** がデフォルトとして使用されます。
WILDFLY_OPTS	起動時に含む追加の引数。以下に例を示します。 -Dorg.wildfly.openssl.path=**PATH_TO_OPENSSL_LIBS**

A.4. JBOSS EAP サブシステムの概要

以下の表は、JBoss EAP のサブシステムを簡単に説明します。

表A.5 JBoss EAP サブシステム

JBoss EAP サブシステム	説明
batch-jberet	バッチアプリケーションを実行する環境を設定 し、バッチジョブを管理 します。
bean-validation	Java オブジェクトデータを検証するために bean バリデーション を設定します。
core-management	サーバーライフサイクルイベント のリスナーを登録し、設定の変更 を追跡します。
datasources	データソース を作成および設定し、JDBC データベースドライバー を管理します。
deployment-scanner	アプリケーションがデプロイする特定の場所を監視するために デプロイメントスキャナー を設定します。
ee	グローバルモジュール の定義、記述子ベースのプロパティー置換 の有効化、およびデフォルトバインディングの設定など、Jakarta EE プラットフォームで一般的な機能を設定します。
ejb3	セッション Bean やメッセージ駆動型 Bean を含む Jakarta Enterprise Beans を設定します。 **ejb3** サブシステムの詳細は、JBoss EAP の **Developing Jakarta Enterprise Beans Applications** を参照してください。
elytron	サーバーおよびアプリケーションのセキュリティーを設定します。 **elytron** サブシステムの詳細は JBoss EAP の **Security Architecture** を参照してください。
iiop-openjdk	JTS トランザクションの CORBA (Common Object Request Broker Architecture) サービスおよびセキュリティーを含むその他の ORB サービス を設定します。JBoss EAP 6 ではこの機能は **jacorb** サブシステムに含まれていました。
infinispan	JBoss EAP の高可用性サービスの キャッシング 機能を設定します。
io	他のサブシステムによって使用される ワーカー および バッファープール を定義します。
jaxrs	Jakarta RESTful Web Services アプリケーションのデプロイメントおよび機能を有効にします。
jca	Jakarta Connectors コンテナー およびリソースアダプターデプロイメントの一般設定を行います。

JBoss EAP サブシステム	説明
jdr	トラブルシューティングに役立つ診断データの収集を有効にします。JBoss EAP のサブスクライバーはサポートをリクエストするときにこの情報を Red Hat に提供できます。
jgroups	クラスターのサーバーがお互いに対話するためのプロトコルスタックと 通信メカニズム を設定します。
jmx	リモート JJakarta Management のアクセスを設定します。
jpa	Jakarta Persistence 2.2 のコンテナー管理の要件を管理し、永続ユニットの定義、アノテーション、および記述子のデプロイを可能にします。 **jpa** サブシステムの詳細は JBoss EAP の **Development Guide** を参照してください。
jsf	Jakarta Server Faces 実装を管理します。
jsr77	Jakarta 管理仕様 いによって定義されている Jakarta EE 管理機能を提供します。
logging	ログカテゴリー および ログハンドラー を介してシステムおよびアプリケーションレベルのロギングを設定します。
mail	メールサーバーの属性 と カスタムメールトランスポート を設定して、JBoss EAP へデプロイされたアプリケーションがメールを送信できるメールサービスを作成します。
messaging-activemq	統合メッセージングプロバイダーである Artemis の Jakarta Messaging 宛先、接続ファクトリー、およびその他の設定を設定します。JBoss EAP 6 では、メッセージング機能は **messaging** サブシステムに含まれていました。 **messaging-activemq** サブシステムの詳細は、JBoss EAP の **Configuring Messaging** を参照してください。
metrics	管理モデルおよび Java 仮想マシン (JVM)MBean からのベースメトリックを表示します。JBoss EAP には **microprofile-smallrye-metrics** サブシステムが含まれなくなったため、アプリケーションメトリクスが利用できなくなりました。
health	JBoss EAP ランタイムのヘルスチェックを公開します。JBoss EAP には **microprofile-smallrye-health** サブシステムが含まれなくなったため、アプリケーションのヘルスチェックは利用できなくなりました。
modcluster	サーバー側の mod_cluster ワーカーノード を設定します。

JBoss EAP サブシステム	説明
naming	エントリーをグローバル JNDI 名前空間にバインドし、リモート JNDI インターフェイスを設定します。
picketlink-federation	PicketLink SAML ベースのシングルサインオン (SSO) を設定します。 **picketlink-federation** サブシステムの詳細は JBoss EAP の How To Set Up SSO with SAML v2 を参照してください。
picketlink-identity-management	PicketLink アイデンティティー管理サービスを設定します。このサブシステムはサポート対象外です。
pojo	過去のバージョンの JBoss EAP でサポートされたように、JBoss Microcontainer サービスが含まれるアプリケーションのデプロイメントを有効にします。
remoting	ローカルおよび リモートサービス のインバウンドおよびアウトバウンド接続を設定します。
discovery	discovery サブシステムは、現在内部サブシステム使用のみで、公開 API では使用できません。
request-controller	サーバーを正常に一時停止およびシャットダウン するよう設定します。
resource-adapters	Jakarta Connectors 仕様を使用して Jakarta EE アプリケーションおよび EIS (Enterprise Information System) 間の通信を行うために リソースアダプター を設定および維持します。
rts	REST-AT のサポート対象外の実装。
sar	過去のバージョンの JBoss EAP でサポートされたように、MBean サービスが含まれる SAR アーカイブのデプロイメントを有効にします。
security	アプリケーションのセキュリティーを設定するレガシーな方法。 **security** サブシステムの詳細は、JBoss EAP の Security Architecture を参照してください。
security-manager	Java Security Manager によって使用される Java セキュリティーポリシーを設定します。 **security-manager** サブシステムの詳細は JBoss EAP の How to Configure Server Security を参照してください。
singleton	シングルトンポリシーを定義して、シングルトンデプロイメントの動作を設定したり、シングルトン MSC サービスを作成したりします。 **singleton** サブシステムの詳細は JBoss EAP の Development Guide を参照してください。

JBoss EAP サブシステム	説明
transactions	タイムアウト値やトランザクションロギングなどのトランザクションマネージャー (TM) のオプションと、JTS (Java Transaction Service) を使用するかどうかを設定します。 **transactions** サブシステムの詳細は、Managing Transactions on JBoss EAP を参照してください。
undertow	JBoss EAP の web サーバー およびサーブレットコンテナーを設定します。JBoss EAP 6 では、この機能は **web** サブシステムに含まれていました。
webservices	パブリッシュされたエンドポイントアドレスおよびエンドポイントハンドラーチェーンを設定します。 また、Web サービスプロバイダーのホスト名、ポート、および WSDL アドレスも設定します。 **webservices** サブシステムの詳細は JBoss EAP の Developing Web Services Applications を参照してください。
weld	JBoss EAP の Jakarta Contexts and Dependency Injection 機能を設定します。
xts	トランザクションの Web サービスの調整を設定します。

A.5. ADD-USER ユーティリティー引数

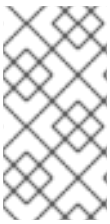
以下の表は、**add-user.sh** または **add-user.bat** スクリプトで使用できる引数を示しています。 これらのスクリプトはデフォルトの認証のプロパティーファイルに新しいユーザーを追加するためのユーティリティーです。

表A.6 add-user コマンド引数

コマンドライン引数	説明
-a	アプリケーションレルムでユーザーを作成します。省略した場合、デフォルトでは管理レルムでユーザーが作成されます。
-dc <value>	プロパティーファイルが含まれるドメイン設定ディレクトリー。省略した場合、デフォルトのディレクトリーは **EAP_HOME/domain/configuration/** になります。
-sc <value>	プロパティーファイルが含まれる代替のスタンドアロンサーバー設定ディレクトリー。省略した場合、デフォルトのディレクトリーは **EAP_HOME/standalone/configuration/** になります。
-up、--user-properties <value>	代替のユーザープロパティーファイルの名前。絶対パスを使用でき、代替の設定ディレクトリーを指定する **-sc** または **-dc** 引数と共に使用されるファイル名を使用することもできます。

コマンドライン引数	説明
-g、--group <value>	このユーザーに割り当てるグループのコンマ区切りリスト。
-gp、--group-properties <value>	代替のグループプロパティーファイルの名前。絶対パスを使用でき、代替の設定ディレクトリーを指定する **-sc** または **-dc** 引数と共に使用されるファイル名を使用することもできます。
-p、--password <value>	ユーザーのパスワード。
-u、--user <value>	ユーザーの名前。ユーザー名には、以下の文字のみを使用できます。 文字の数と順番は自由です。 ● 英数字 (a-z、A-Z、0-9) ● ダッシュ (-)、ピリオド (.)、コンマ (,)、アットマーク (@) ● バックスラッシュ (\) ● 等号 (=)
-r、--realm <value>	管理インターフェイスをセキュアにするために使用されるレルムの名前。省略した場合、デフォルト値は **ManagementRealm** です。
-s、--silent	コンソールへ出力せずに **add-user** スクリプトを実行します。
-e、--enable	ユーザーを有効にします。
-d、--disable	ユーザーを無効にします。
-cw、--confirm-warning	対話モードで自動的に警告を確認します。
-h、--help	**add-user** スクリプトの使用情報を表示します。
-ds、 --display-secret	非対話モードで秘密の値を出力します。

A.6. 管理監査ロギング属性



注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-config_5_0.xsd** のスキーマ定義ファイルで確認してください。

表A.7 管理監査ロギング: ロガー属性

属性	説明
enabled	監査ロギングが有効になっているかどうか。
log-boot	操作がサーバーの起動時にログに記録されるかどうか。
log-read-only	設定を変更しない操作またはランタイムサービスがログに記録されるかどうか。

表A.8 管理監査ロギング: ログフォーマッター属性

属性	説明
compact	**true** の場合、JSON を 1 行でフォーマットします。新しい行が含まれる値が存在する可能性があるため、レコード全体を 1 行にするのが重要な場合は、 **escape-new-line** または **escape-control-characters** を **true** に設定します。
date-format	**java.text.SimpleDateFormat** が理解するように使用するデータ形式。**include-date** が **false** に設定されている場合は無視されます。
date-separator	日付と他のフォーマットされたログメッセージのセパレーター。**include-date** が **false** に設定されている場合は無視されます。
escape-control-characters	**true** の場合、すべての制御文字 (10 進値が **32** を超える ASCII エントリー) を 8 進の ASCII コードでエスケープします。たとえば、新しい行は **#012** になります。**true** の場合、**escape-new-line=false** をオーバーライドします。
escape-new-line	**true** の場合、新しい行を 8 進の ASCII コードでエスケープします (**#012**)。
include-date	フォーマットされたログレコードに日付が含まれるかどうか。

表A.9 管理監査ロギング: ファイルハンドラー属性

属性	説明
disabled-due-to-failure	ロギングの失敗によりこのハンドラーが無効になったかどうか (読み取り専用)。
failure-count	ハンドラーが初期化された後に発生したロギング失敗数 (読み取り専用)。
formatter	ログメッセージのフォーマットに使用される JSON フォーマッター。
max-failure-count	このハンドラーを無効化する前の最大ロギング失敗数。

属性	説明
path	監査ログファイルのパス。
relative-to	以前指定された別のパスの名前、またはシステムによって提供される標準的なパスの 1 つ。**relative-to** が指定された場合、**path** 属性の値は、この属性によって指定されたパスへの相対値としてみなされます。
rotate-at-startup	サーバーの起動時に古いログファイルをローテーションするかどうか。

表A.10 管理監査ロギング: Syslog ハンドラー属性

属性	説明
app-name	**RFC-5424** のセクション 6.2.5 で定義された syslog レコードに追加するアプリケーション名。指定されない場合、デフォルト値は製品の名前になります。
disabled-due-to-failure	ロギングの失敗によりこのハンドラーが無効になったかどうか (読み取り専用)。
facility	**RFC-5424** のセクション 6.2.1 と **RFC-3164** のセクション 4.1.1 で定義された syslog ロギングに使用する機能。
failure-count	ハンドラーが初期化された後に発生したロギング失敗数 (読み取り専用)。
formatter	ログメッセージのフォーマットに使用される JSON フォーマッター。
max-failure-count	このハンドラーを無効化する前の最大ロギング失敗数。
max-length	許可される、ヘッダーを含むログメッセージの最大長 (バイト単位)。未定義の場合、デフォルト値は **1024** バイト (**syslog-format** が **RFC3164** の場合) または **2048** バイト (**syslog-format** が **RFC5424** の場合) になります。
protocol	syslog ハンドラーに使用するプロトコル。**udp**、**tcp**、または **tls** のいずれか 1 つである必要があります。
syslog-format	syslog 形式: **RFC5424** または **RFC3164**.
truncate	ヘッダーを含むメッセージの長さ (バイト単位) が **max-length** 属性の値よりも大きい場合、そのメッセージを省略するかどうか。**false** に設定すると、メッセージが分割され、同じヘッダー値で送信されます。

注記

syslog サーバーごとに実装が異なるため、すべての設定をすべての syslog サーバーに適用できるとは限りません。テストは **rsyslog** syslog 実装を使用して実行されています。

次の表には、高度な属性のみがリストされています。各属性は設定パラメーターを持ち、一部の属性は個設定パラメーターを持ちます。

A.7. インターフェイス属性

注記

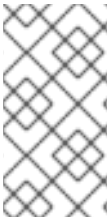
この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-config_5_0.xsd** のスキーマ定義ファイルで確認してください。

表A.11 インターフェイス属性と値

インターフェイス要素	説明
any	インターフェイスの選択基準の一部は、最低でも基準のネストされたセットの 1 つ (すべてとは限らない) を満たす必要があることを示す要素。
any-address	このインターフェイスを使用するソケットをワイルドカードアドレスにバインドする必要があることを示す空の要素。**java.net.preferIPv4Stack** システムプロパティーが true に設定されていない限り、IPv6 ワイルドカードアドレス (**::**) が使用されます。true に設定された場合は、IPv4 ワイルドカードアドレス (**0.0.0.0**) が使用されます。ソケットがデュアルスタックマシンの IPv6 anylocal アドレスにバインドされた場合は、IPv6 および IPv4 トラフィックを受け入れることができます。 IPv4 (IPv4 マッピング) anylocal アドレスにバインドされた場合は、IPv4 トラフィックのみを受け入れることができます。
inet-address	IPv6 または IPv4 のドット区切り表記の IP アドレス、または IP アドレスに解決できるホスト名。
link-local-address	インターフェイスの選択基準の一部として、関連付けられたアドレスがリンクローカルであるかどうかを示す空の要素。
loopback	インターフェイスの選択基準の一部として、ループバックインターフェイスであるかどうかを示す空の要素。
loopback-address	マシンのループバックインターフェイスで実際には設定できないループバックアドレス。IP アドレスが関連付けられた NIC が見つからない場合であっても該当する値が使用されるため、inet-address タイプとは異なります。

インターフェイス要素	説明
multicast	インターフェイスの選択基準の一部として、マルチキャストをサポートするかどうかを示す空の要素。
name	インターフェイスの名前。
nic	ネットワークインターフェイスの名前 (eth0、eth1、lo など)。
nic-match	使用できるインターフェイスを見つけるために、マシンで利用可能なネットワークインターフェイスの名前を検索する正規表現。
not	インターフェイスの選択基準の一部は、基準のネストされたセットを満たしてはならないことを示す要素。
point-to-point	インターフェイスの選択基準の一部として、ポイントツーポイントインターフェイスであるかどうかを示す空の要素。
public-address	インターフェイスの選択基準の一部として、公開されたルーティング可能なアドレスを持つかどうかを示す空の要素。
site-local-address	インターフェイスの選択基準の一部として、関連付けられたアドレスがサイトローカルであるかどうかを示す空の要素。
subnet-match	**スラッシュ表記法**で記述されたネットワーク IP アドレスとアドレスのネットワーク接頭辞のビット数 (例: **192.168.0.0/16**)。
up	インターフェイスの選択基準の一部として、現在稼動しているかどうかを示す空の要素。
virtual	インターフェイスの選択基準の一部として、仮想インターフェイスであるかどうかを示す空の要素。

A.8. ソケットバインディング属性



注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-config_5_0.xsd** のスキーマ定義ファイルで確認してください。

以下の表は、3 種類のソケットバインディングそれぞれに設定できる属性を表しています。

- socket-binding
- remote-destination-outbound-socket-binding
- local-destination-outbound-socket-binding

表A.12 インバウンドソケットバインディング **(socket-binding)** の属性

属性	説明
client-mappings	このソケットバインディングのクライアントマッピングを指定します。このソケットへ接続するクライアントは、希望のアウトバウンドインターフェイスと一致するマッピングに指定された宛先アドレスを使用する必要があります。これにより、ネットワークアドレスの変換を使用する高度なネットワークトポロジーまたは複数のネットワークインターフェイスにバインディングを持つ高度なネットワークトポロジーが機能します。各マッピングは宣言された順序で評価される必要があり、最初に一致したマッピングを使用して宛先が決定されます。
fixed-port	ソケットグループの他のソケットに数値のオフセットが適用された場合でもポートの値を固定したままにするかどうか。
interface	ソケットがバインドされる必要があるインターネットの名前、またはマルチキャストソケットの場合はリッスンするインターフェイス。宣言されたインターフェイスの1つである必要があります。定義されないと、エンクロージングソケットバインディンググループからの **default-interface** の値が使用されます。
multicast-address	ソケットがマルチキャストトラフィックを受信するマルチキャストアドレス。指定しないと、ソケットがマルチキャストを受信するよう設定されません。
multicast-port	ソケットがマルチキャストトラフィックを受信するポート。**multicast-address** が設定されている場合はこれも設定する必要があります。
name	ソケットの名前。ソケット設定情報にアクセスする必要があるサービスは、この名前を使用してソケット設定情報を探します。必須の属性です。
port	ソケットがバインドされる必要があるポートの番号。サーバーによってポートオフセットが適用され、ポートの値がすべて増加または減少される場合、この値は上書きされることに注意してください。

表A.13 リモートアウトバウンドソケットバインディング **(remote-destination-outbound-socket-binding)** の属性

属性	説明
fixed-source-port	数値のオフセットポートがソケットグループの別のアウトバウンドソケットに適用される場合、ポートの値を固定すべきかどうか。
host	このアウトバウンドソケットが接続するリモート宛先のホスト名または IP アドレス。
port	アウトバウンドソケットが接続すべきリモート宛先のポート番号。
source-interface	アウトバウンドソケットのソースアドレスに使用されるインターフェイスの名前。
source-port	アウトバウンドソケットのソースポートとして使用されるポート番号。

表A.14 ローカルアウトバウンドソケットバインディング (**local-destination-outbound-socket-binding**) の属性

属性	説明
fixed-source-port	数値のオフセットポートがソケットグループの別のアウトバウンドソケットに適用される場合、ポートの値を固定すべきかどうか。
socket-binding-ref	このアウトバウンドソケットが接続するポートを決定するために使用されるローカルソケットバインディングの名前。
source-interface	アウトバウンドソケットのソースアドレスに使用されるインターフェイスの名前。
source-port	アウトバウンドソケットのソースポートとして使用されるポート番号。

A.9. デフォルトのソケットバインディング

以下の表は、各ソケットバインディンググループのデフォルトのソケットバインディングを示しています。

- standard-sockets
- ha-sockets
- full-sockets
- full-ha-sockets
- load-balancer-sockets

表A.15 standard-sockets

ソケットバインディング	ポート	説明
ajp	8009	Apache JServ プロトコル。HTTP クラスタリングおよび負荷分散に使用します。
http	8080	デプロイされた Web アプリケーションのデフォルトポート。
https	8443	デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。
management-http	9990	管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993	管理レイヤーを用いた HTTPS 通信に使用されます。
txn-recovery-environment	4712	Jakarta Transactions リカバリーマネージャー。

ソケットバインディング	ポート	説明
txn-status-manager	4713	Jakarta Transactions / JTS トランザクションマネージャー。

表A.16 ha-sockets

ソケットバインディング	ポート	マルチキャストポート	説明
ajp	8009		Apache JServ プロトコル。HTTP クラスタリングおよび負荷分散に使用します。
http	8080		デプロイされた Web アプリケーションのデフォルトポート。
https	8443		デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。
jgroups-mping		45700	マルチキャスト。HA クラスターでの初期メンバーシップの検出に使用されます。
jgroups-tcp	7600		TCP を使用した、HA クラスター内でのユニキャストピア検出。
jgroups-udp	55200	45688	UDP を使用した、HA クラスター内でのマルチキャストピア検出。
management-http	9990		管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993		管理レイヤーを用いた HTTPS 通信に使用されます。
modcluster		23364	JBoss EAP と HTTP ロードバランサー間の通信に対するマルチキャストポート。
txn-recovery-environment	4712		Jakarta Transactions リカバリーマネージャー。
txn-status-manager	4713		Jakarta Transactions / JTS トランザクションマネージャー。

表A.17 full-sockets

ソケットバインディング	ポート	説明
ajp	8009	Apache JServ プロトコル。HTTP クラスタリングおよび負荷分散に使用します。
http	8080	デプロイされた Web アプリケーションのデフォルトポート。
https	8443	デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。
iiop	3528	JTS トランザクションおよび他の ORB 依存サービス用の CORBA サービス。
iiop-ssl	3529	SSL 暗号化 CORBA サービス。
management-http	9990	管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993	管理レイヤーを用いた HTTPS 通信に使用されます。
txn-recovery-environment	4712	Jakarta Transactions リカバリーマネージャー。
txn-status-manager	4713	Jakarta Transactions / JTS トランザクションマネージャー。

表A.18 full-ha-sockets

名前	ポート	マルチキャストポート	説明
ajp	8009		Apache JServ プロトコル。HTTP クラスタリングおよび負荷分散に使用します。
http	8080		デプロイされた Web アプリケーションのデフォルトポート。
https	8443		デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。
iiop	3528		JTS トランザクションおよび他の ORB 依存サービス用の CORBA サービス。
iiop-ssl	3529		SSL 暗号化 CORBA サービス。
jgroups-mping		45700	マルチキャスト。HA クラスターでの初期メンバーシップの検出に使用されます。

名前	ポート	マルチキャストポート	説明
jgroups-tcp	7600		TCP を使用した、HA クラスター内でのユニキャストピア検出。
jgroups-udp	55200	45688	UDP を使用した、HA クラスター内でのマルチキャストピア検出。
management-http	9990		管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993		管理レイヤーを用いた HTTPS 通信に使用されます。
modcluster		23364	JBoss EAP と HTTP ロードバランサー間の通信に対するマルチキャストポート。
txn-recovery-environment	4712		Jakarta Transactions リカバリーマネージャー。
txn-status-manager	4713		Jakarta Transactions / JTS トランザクションマネージャー。

表A.19 load-balancer-sockets

名前	ポート	マルチキャストポート	説明
http	8080		デプロイされた Web アプリケーションのデフォルトポート。
https	8443		デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。
management-http	9990		管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993		管理レイヤーを用いた HTTPS 通信に使用されます。
mcmp-management	8090		ライフサイクルイベントを送信する MCMP (Mod-Cluster Management Protocol) 接続のポート。
modcluster		23364	JBoss EAP と HTTP ロードバランサー間の通信に対するマルチキャストポート。

A.10. モジュールコマンド引数

以下の引数は、**module add** 管理 CLI コマンドに渡すことができます。

表A.20 モジュールコマンド引数

引数	説明
--absolute-resources	この引数を使用して、**module.xml** ファイルから参照するファイルシステムの絶対パスのリストを指定します。指定されるファイルはモジュールディレクトリーにはコピーされません。 区切り文字の詳細は **--resource-delimiter** を参照してください。
--allow-nonexistent-resources	この引数を使用して、**--resources** によって指定された存在しないリソースの空のディレクトリーを作成します。存在しないリソースがある場合、この引数を使用しないと **module add** コマンドの実行に失敗します。
--dependencies	この引数を使用して、このモジュールが依存するモジュール名のコンマ区切りリストを提供します。
--export-dependencies	この引数を使用して、エクスポートされた依存関係を指定します。 module add --name=com.mysql --resources=/**path**/**to**/mysql-connector-java-8.0.12.jar --export-dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
--main-class	この引数を使用して、モジュールのメインメソッドを宣言する完全修飾クラス名を指定します。
--module-root-dir	デフォルトの **EAP_HOME**/**modules**/ ディレクトリーの代わりに、外部の JBoss EAP モジュールディレクトリーを定義した場合は、この引数を使用します。 module add --module-root-dir=/**path**/**to**/my-external-modules/ --name=com.mysql --resources=/**path**/**to**/mysql-connector-java-8.0.12.jar --dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
--module-xml	この引数を使用して、この新規モジュールに使用する **module.xml** へのファイルシステムパスを提供します。ます。この引数を指定しないと、**module.xml** ファイルがモジュールディレクトリーに生成されます。
--name	この引数を使用して、追加するモジュールの名前を提供します。この引数は必須です。

引数	説明
--properties	この引数を使用して、モジュールプロパティーを定義する **PROPERTY_NAME=PROPERTY_VALUE** ペアのコンマ区切りリストを提供します。
--resource-delimiter	この引数を使用して、**--resources** または **absolute-resources** 引数に提供されたリソースリストの、ユーザー定義のファイルパス区切り文字を設定します。設定されていない場合、ファイルパスの区切り文字は Linux ではコロン (**:**) 、Windows ではセミコロン (**;**) になります。
--resources	この引数を使用して、ファイルシステムパスのリストを提供してこのモジュールのリソースを指定します。ファイルは直接このモジュールディレクトリーにコピーされ、その **module.xml** ファイルから参照されます。ディレクトリーへのパスを提供した場合、ディレクトリーとその内容はモジュールディレクトリーにコピーされます。シンボリックリンクは保持されず、リンクされたリソースはモジュールディレクトリーにコピーされます。**--absolute-resources** または **--module-xml** が提供されている場合を除き、この引数は必須です。 区切り文字の詳細は **--resource-delimiter** を参照してください。
--slot	この引数を使用して、デフォルトの **main** スロット以外のスロットにモジュールを追加します。 `module add --name=com.mysql --slot=8.0 --resources=/path/to/mysql-connector-java-8.0.12.jar --dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api`

A.11. デプロイメントスキャナーマーカーファイル

マーカーファイルは、JBoss EAP サーバーインスタンスのデプロイメントディレクトリー内でアプリケーションの状態をマーク付けするためにデプロイメントスキャナーによって使用されます。マーカーファイルの名前はデプロイメントの名前と同じで、ファイル接尾辞はアプリケーションのデプロイメントの状態を示します。

たとえば、**test-application.war** のデプロイメントには **test-application.war.deployed** という名前のマーカーファイルがあります。

以下の表は、使用できるマーカーファイルタイプとそれらの意味を表しています。

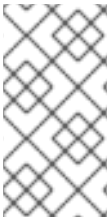
表A.21 マーカーファイルのタイプ

ファイル名接尾辞	生成	説明
.deployed	システムによる生成	コンテンツがデプロイされたことを示します。このファイルが削除された場合、コンテンツはアンデプロイされます。

ファイル名接尾辞	生成	説明
.dodeploy	ユーザーによる生成	コンテンツをデプロイまたは再デプロイする必要があることを意味します。
.failed	システムによる生成	デプロイメントの失敗を示します。マーカーファイルには、失敗の原因に関する情報が含まれます。マーカーファイルが削除された場合、コンテンツは再度自動デプロイの対象になります。
.isdeploying	システムによる生成	デプロイ中であることを示します。デプロイの完了後、このマーカーファイルは削除されます。
.isundeploying	システムによる生成	**.deployed** ファイルの削除によってトリガーされ、コンテンツのアンデプロイ中であることを示します。デプロイの完了後、このマーカーファイルは削除されます。
.pending	システムによる生成	デプロイメントスキャナーはコンテンツをデプロイする必要性を認識しているにも関わらず、現在問題によって自動デプロイメントが実行されないことを意味します (例: コンテンツのコピー中である場合)。このマーカーはグローバルデプロイメントロードブロックとして機能するため、スキャナーはこのマーカーファイルが存在する間、**あらゆる**コンテンツのデプロイおよびアンデプロイをサーバーに指示しません。
.skipdeploy	ユーザーによる生成	アプリケーションの自動デプロイを無効にします。デプロイメント形式のコンテンツの自動デプロイメントを一時的にブロックする方法として役に立ち、不完全なコンテンツの編集がプッシュされないようにします。スキャナーは zip 形式のコンテンツに対する処理中の変更を検出し、完了するまで待機しますが、zip 形式のコンテンツとともに使用できます。
.undeployed	システムによる生成	コンテンツがアンデプロイされたことを示します。このマーカーファイルを削除しても、コンテンツの再デプロイメントには影響ありません。

A.12. デプロイメントスキャナーの属性

デプロイメントスキャナーには設定可能な以下の属性が含まれます。



注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/jboss-as-deployment-scanner_2_0.xsd** のスキーマ定義ファイルで確認してください。

表A.22 デプロイメントスキャナーの属性

名前	デフォルト	説明
auto-deploy-exploded	false	**.dodeploy** マーカーファイルなしで、デプロイメント形式のコンテンツの自動デプロイメントを許可します。基本的な開発シナリオに対してのみ推奨され、開発者またはオペレーティングシステムによる変更中に、デプロイメント形式のアプリケーションデプロイメントが行われないようにします。
auto-deploy-xml	true	**.dodeploy** マーカーファイルなしでの XML コンテンツの自動デプロイメントを許可します。
auto-deploy-zipped	true	**.dodeploy** マーカーファイルなしでの zip 形式のコンテンツの自動デプロイメントを許可します。
deployment-timeout	600	デプロイメントスキャナーでデプロイメントをキャンセルするまでのデプロイメント試行許可時間 (秒単位)。
path	deployments	スキャンされる実際のファイルシステムパス。**relative-to** 属性が指定されない限り、絶対パスとして処理され、値はそのパスの相対パスとして処理されます。
relative-to	jboss.server.base.dir	サーバー設定の パス として定義されるファイルシステムパスへの参照。
runtime-failure-causes-rollback	false	デプロイメントのランタイム障害が原因で、スキャン操作の一部としてそのデプロイメントやその他すべてのデプロイメント (関係しないデプロイメントの可能性あり) のロールバックが発生するかどうか。
scan-enabled	true	**scan-interval** により起動時にアプリケーションの自動スキャンを許可します。
scan-interval	5000	変更に対してレポジトリーがスキャンされる間隔 (ミリ秒単位)。**1** 未満の値を設定すると、初期設定時のみスキャンが行われます。

A.13. マネージドドメインの JVM 設定属性

以下の JVM 設定オプションは、ホスト、サーバーグループ、またはサーバーレベルでマネージドドメインに設定できます。一部の属性の有効な値は JVM によって異なることに注意してください。詳細は、JDK ベンダーのドキュメントを参照してください。

注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-config_5_0.xsd** のスキーマ定義ファイルで確認してください。

表A.23 JVM 設定属性

属性	説明
agent-lib	Java エージェントライブラリーを指定する **-agentlib java** オプションの値を設定します。
agent-path	Java エージェントパスを指定する **-agentpath java** オプションの値を設定します。
debug-enabled	デバッグを有効にするかどうか。この属性は、サーバーレベルの JVM 設定のみに適用されます。
debug-options	デバッグが有効な場合に JVM オプションを指定します。この属性は、サーバーレベルの JVM 設定のみに適用されます。
env-classpath-ignored	**CLASSPATH** 環境変数を無視するかどうか。
environment-variables	キーバリューペアの環境変数を指定します。
heap-size	JVM によって割り当てられる初期のヒープサイズを指定する **-Xms** オプションの値を設定します。
java-agent	Java エージェントを指定する **-javaagent java** オプションの値を設定します。
java-home	**JAVA_HOME** 変数の値を設定します。
jvm-options	必要な追加の JVM オプションを指定します。
launch-command	サーバープロセスの起動に使用される **java** コマンドの前に付けるオペレーティングシステムレベルのコマンドを指定します。たとえば、**sudo** コマンドを使用して、別のユーザーとして Java プロセスを実行できます。
max-heap-size	JVM によって割り当てられる最大ヒープサイズを指定する **-Xmx** オプションの値を設定します。
max-permgen-size	永久生成の最大サイズを設定します。**非推奨: JVM は個別の永久生成領域を提供しないようになりました。**
permgen-size	永久生成の初期サイズを設定します。**非推奨: JVM は個別の永久生成領域を提供しないようになりました。**
stack-size	JVM スタックサイズを指定する **-Xss** オプションの値を設定します。
type	使用中の JVM を提供したベンダーを指定します。**ORACLE**、**IBM**、**SUN** および **OTHER** を指定できます。

A.14. MAIL サブシステムの属性

以下の表は、メールセッション と以下のメールサーバータイプ用の mail サブシステムの属性を表しています。

- imap
- pop3
- smtp
- custom

注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-mail_3_0.xsd** のスキーマ定義ファイルで確認してください。

表A.24 メールセッションの属性

属性	説明
debug	Jakarta メールのデバッグを有効にするかどうか。
from	送信時に送信元が設定されていない場合のデフォルトの送信元 (from) アドレス。
jndi-name	メールセッションのバインド先の JNDI 名。

表A.25 IMAP メールサーバーの属性

属性	説明
credential-reference	サーバー上で認証される、認証情報ストアからの認証情報。
outbound-socket-binding-ref	メールサーバーのアウトバウンドソケットバインディングへの参照。
password	サーバー上で認証するパスワード。
ssl	サーバーが SSL を必要とするかどうか。
tls	サーバーに TLS が必要であるかどうか。
username	サーバー上で認証するユーザー名。

表A.26 POP3 メールサーバーの属性

属性	説明
credential-reference	サーバー上で認証される、認証情報ストアからの認証情報。
outbound-socket-binding-ref	メールサーバーのアウトバウンドソケットバインディングへの参照。
password	サーバー上で認証するパスワード。
ssl	サーバーが SSL を必要とするかどうか。
tls	サーバーに TLS が必要であるかどうか。
username	サーバー上で認証するユーザー名。

表A.27 SMTP メールサーバーの属性

属性	説明
credential-reference	サーバー上で認証される、認証情報ストアからの認証情報。
outbound-socket-binding-ref	メールサーバーのアウトバウンドソケットバインディングへの参照。
password	サーバー上で認証するパスワード。
ssl	サーバーが SSL を必要とするかどうか。
tls	サーバーに TLS が必要であるかどうか。
username	サーバー上で認証するユーザー名。

表A.28 カスタムメールサーバーの属性

属性	説明
credential-reference	サーバー上で認証される、認証情報ストアからの認証情報。
outbound-socket-binding-ref	メールサーバーのアウトバウンドソケットバインディングへの参照。
password	サーバー上で認証するパスワード。
properties	このサーバーの Jakarta Mail プロパティー
ssl	サーバーが SSL を必要とするかどうか。

属性	説明
tls	サーバーに TLS が必要であるかどうか。
username	サーバー上で認証するユーザー名。

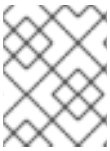
A.15. ルートロガーの属性

注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/jboss-as-logging_3_0.xsd** のスキーマ定義ファイルで確認してください。

表A.29 ルートロガーの属性

属性	説明
filter	簡単なフィルタータイプを定義します。**filter-spec が導入されたため非推奨になりました。**
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** は、パターンに**一致しない**ログエントリーを除外するフィルターを定義します。
handlers	ルートロガーによって使用されるログハンドラーのリスト。
level	ルートロガーが記録するログメッセージの最低レベル。



注記

ルートロガーに指定された **filter-spec** は他のハンドラーによって継承されません。ハンドラーごとに **filter-spec** を指定する必要があります。

A.16. ログカテゴリーの属性

注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/jboss-as-logging_6_0.xsd** のスキーマ定義ファイルで確認してください。

表A.30 ログカテゴリーの属性

属性	説明
category	ログメッセージがキャプチャーされるログカテゴリー。
filter	簡単なフィルタータイプを定義します。**filter-spec が導入されたため非推奨になりました。**
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** はパターンに一致しないフィルターを定義します。
handlers	ロガーに関連付けられたログハンドラーのリスト。
level	ログカテゴリーが記録するログメッセージの最低レベル。
use-parent-handlers	**true** に設定した場合、割り当てられた他のハンドラーだけでなく、ルートロガーのログハンドラーを使用します。

A.17. ログハンドラーの属性

注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/jboss-as-logging_6_0.xsd** のスキーマ定義ファイルで確認してください。

表A.31 Console ログハンドラーの属性

属性	説明
autoflush	**true** に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	**true** に設定された場合、ハンドラーが有効になり、通常とおり機能します。**false** に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
filter	簡単なフィルタータイプを定義します。**filter-spec が導入されたため非推奨になりました。**
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。

属性	説明
level	ログハンドラーが記録するログメッセージの最低レベル。
name	ログハンドラーの名前。**ハンドラーのアドレスに名前が含まれるため非推奨になりました。**
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
target	ログハンドラーの出力が送信されるシステム出力ストリーム。以下の1つになります。 • **System.err**: ログハンドラーの出力はシステムエラーストリームに送信されます。 • **System.out**: ログハンドラーの出力は標準出力ストリームに送信されます。 • **console**: ログハンドラーの出力は **java.io.PrintWriter** クラスに送信されます。

表A.32 File ログハンドラーの属性

属性	説明
append	**true** に設定された場合、このハンドラーが書き込んだすべてのメッセージがファイル (すでに存在する場合) に追加されます。**false** に設定された場合、アプリケーションサーバーが起動されるたびに、新しいファイルが作成されます。
autoflush	**true** に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	**true** に設定された場合、ハンドラーが有効になり、通常とおり機能します。**false** に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
file	このログハンドラーの出力が書き込まれるファイルを表すオブジェクト。このオブジェクトには、**relative-to** と **path** の2つの設定プロパティーが含まれます。
filter	簡単なフィルタータイプを定義します。**filter-spec が導入されたため非推奨になりました。**
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。

属性	説明
name	ログハンドラーの名前。**ハンドラーのアドレスに名前が含まれるため非推奨になりました。**
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。

表A.33 Periodic ログハンドラーの属性

属性	説明
append	**true** に設定された場合、このハンドラーが書き込んだすべてのメッセージがファイル (すでに存在する場合) に追加されます。**false** に設定された場合、アプリケーションサーバーが起動されるたびに、新しいファイルが作成されます。
autoflush	**true** に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	**true** に設定された場合、ハンドラーが有効になり、通常とおり機能します。**false** に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
file	このログハンドラーの出力が書き込まれるファイルを表すオブジェクト。このオブジェクトには、**relative-to** と **path** の 2 つの設定プロパティーが含まれます。
filter	簡単なフィルタータイプを定義します。**filter-spec が導入されたため非推奨になりました。**
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。
name	ログハンドラーの名前。**ハンドラーのアドレスに名前が含まれるため非推奨になりました。**
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
suffix	この文字列はローテーションログに追加される接尾辞に含まれます。**suffix** の書式は、ドット (**.**) の後に **SimpleDateFormat** クラスが解析できる日付の文字列を追加したものです。

表A.34 Size ログハンドラーの属性

属性	説明
append	**true** に設定された場合、このハンドラーが書き込んだすべてのメッセージがファイル (すでに存在する場合) に追加されます。**false** に設定された場合、アプリケーションサーバーが起動されるたびに、新しいファイルが作成されます。
autoflush	**true** に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	**true** に設定された場合、ハンドラーが有効になり、通常とおり機能します。**false** に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
file	このログハンドラーの出力が書き込まれるファイルを表すオブジェクト。このオブジェクトには、**relative-to** と **path** の 2 つの設定プロパティーが含まれます。
filter	簡単なフィルタータイプを定義します。**filter-spec が導入されたため非推奨になりました。**
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。
max-backup-index	保持するローテーションログの最大数。この数に達すると、古いログが再使用されます。デフォルト値は **1** です。 **suffix** 属性が使用された場合は、ローテーションログファイルの接尾辞がローテーションアルゴリズムに含まれます。ログファイルがローテーションされると、名前が **name**+**suffix** で始まる最も古いファイルが削除され、残りのローテーションログファイルの数値接尾辞が増分され、新しくローテーションされたログファイルに数値接尾辞 **1** が提供されます。
name	ログハンドラーの名前。**ハンドラーのアドレスに名前が含まれるため非推奨になりました。**
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
rotate-on-boot	**true** に設定されると、新しいログファイルがサーバーの起動時に作成されます。デフォルトは **false** です。

属性	説明
rotate-size	ログファイルがローテーションされる前に到達できる最大サイズです。数字に追加された単一の文字はサイズ単位を示します。バイトの場合は **b**、キロバイトの場合は **k**、メガバイトの場合は **m**、ギガバイトの場合は **g** になります。たとえば、50 メガバイトの場合は、**50m** になります。
suffix	この文字列はローテーションログに追加される接尾辞に含まれます。**suffix** の書式は、ドット (**.**) の後に **SimpleDateFormat** クラスが解析できる日付の文字列を追加したものです。

表A.35 Periodic Size ログハンドラーの属性

属性	説明
append	**true** に設定された場合、このハンドラーが書き込んだすべてのメッセージがファイル (すでに存在する場合) に追加されます。**false** に設定された場合、アプリケーションサーバーが起動されるたびに、新しいファイルが作成されます。
autoflush	**true** に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	**true** に設定された場合、ハンドラーが有効になり、通常とおり機能します。**false** に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
file	このログハンドラーの出力が書き込まれるファイルを表すオブジェクト。このオブジェクトには、**relative-to** と **path** の 2 つの設定プロパティーが含まれます。
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。
max-backup-index	保持するローテーションログの最大数。この数に達すると、古いログが再使用されます。デフォルト値は **1** です。 **suffix** 属性が使用された場合は、ローテーションログファイルの接尾辞がローテーションアルゴリズムに含まれます。ログファイルがローテーションされると、名前が **name**+**suffix** で始まる最も古いファイルが削除され、残りのローテーションログファイルの数値接尾辞が増分され、新しくローテーションされたログファイルに数値接尾辞 **1** が提供されます。
name	ログハンドラーの名前。**ハンドラーのアドレスに名前が含まれるため非推奨になりました。**

属性	説明
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
rotate-on-boot	**true** に設定されると、新しいログファイルがサーバーの起動時に作成されます。デフォルトは **false** です。
rotate-size	ログファイルがローテーションされる前に到達できる最大サイズです。数字に追加された単一の文字はサイズ単位を示します。バイトの場合は **b**、キロバイトの場合は **k**、メガバイトの場合は **m**、ギガバイトの場合は **g** になります。たとえば、50 メガバイトの場合は、**50m** になります。
suffix	この文字列はローテーションログに追加される接尾辞に含まれます。**suffix** の書式は、ドット (**.**) の後に **SimpleDateFormat** クラスが解析できる日付の文字列を追加したものです。

表A.36 syslog ハンドラーの属性

属性	説明
app-name	メッセージを RFC5424 形式でフォーマットするときに使用されるアプリケーション名。デフォルトのアプリケーション名は **java** です。
enabled	**true** に設定された場合、ハンドラーが有効になり、通常とおり機能します。**false** に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
facility	RFC-5424 および RFC-3164 によって定義される機能。
hostname	メッセージ送信元のホストの名前。たとえば、アプリケーションサーバーが実行されているホストの名前になります。
level	ログハンドラーが記録するログメッセージの最低レベル。
port	syslog サーバーがリッスンしているポート。
server-address	syslog サーバーのアドレス。
syslog-format	RFC 仕様にしたがってログメッセージをフォーマットします。
named-formatter	syslog ペイロードのメッセージをフォーマットします。この属性を使用すると、必要に応じてメッセージをカスタマイズできます。

表A.37 Socket ログハンドラーの属性

属性	説明
autoflush	毎回書き込み後に自動的にフラッシュを行うかどうか。
block-on-reconnect	**true** に設定すると、再接続の試行時に書き込みメソッドがブロックされます。非同期ハンドラーの使用時のみ **true** に設定することが推奨されます。
enabled	**true** に設定された場合、ハンドラーが有効になり、通常とおり機能します。**false** に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	このハンドラーによって使用される文字エンコーディング。
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** はパターンに一致しないフィルターを定義します。
level	ログハンドラーが記録するログメッセージの最低レベル。
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
outbound-socket-binding-ref	ソケット接続のアウトバウンドソケットバインディングへの参照。
protocol	ソケットが通信すべきプロトコル。使用できる値は **TCP**、**UDP**、および **SSL_TCP** です。
ssl-context	定義された SSL コンテキストへの参照。これは、**protocol** が **SSL_TCP** に設定された場合のみ使用されます。

表A.38 Custom ログハンドラーの属性

属性	説明
class	使用されるロギングハンドラークラス。
enabled	**true** に設定された場合、ハンドラーが有効になり、通常とおり機能します。**false** に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
filter	簡単なフィルタータイプを定義します。**filter-spec が導入されたため非推奨になりました。**
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。

属性	説明
level	ログハンドラーが記録するログメッセージの最低レベル。
module	ロギングハンドラーが依存するモジュール。
name	ログハンドラーの名前。**ハンドラーのアドレスに名前が含まれるため非推奨になりました。**
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
properties	ロギングハンドラーに使用されるプロパティー。

表A.39 Async ログハンドラーの属性

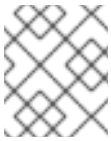
属性	説明
enabled	**true** に設定された場合、ハンドラーが有効になり、通常とおり機能します。**false** に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
filter	簡単なフィルタータイプを定義します。**filter-spec が導入されたため非推奨になりました。**
filter-spec	フィルターを定義する式の値。式 **not(match("WFLY.*"))** はパターンに一致しないフィルターを定義します。
level	ログハンドラーが記録するログメッセージの最低レベル。
name	ログハンドラーの名前。**ハンドラーのアドレスに名前が含まれるため非推奨になりました。**
overflow-action	キューの長さを超えたときにこのハンドラーがどのように応答するかを示します。これは **BLOCK** または **DISCARD** に設定できます。**BLOCK** により、キューでスペースが利用可能になるまでロギングアプリケーションが待機します。これは、非同期でないログハンドラーと同じ動作です。**DISCARD** により、ロギングアプリケーションは動作し続けますが、ログメッセージは削除されます。
queue-length	サブハンドラーが応答するときに、このハンドラーが保持するログメッセージの最大数。
subhandlers	この async ハンドラーがログメッセージを渡すログハンドラーのリスト。

A.18. ログフォーマッター属性

表A.40 パターンフォーマッターのフォーマット文字

記号	説明
%c	ロギングイベントのカテゴリー。
%p	ログエントリーのレベル (INFO、DEBUG など)
%P	ログエントリーのローカライズレベル。
%d	現在の日付/時間 (**yyyy-MM-dd HH:mm:ss,SSS** 形式)。
%r	相対時間 (ログ初期化以降のミリ秒単位の時間)。
%z	日付 (**%d**) の前に指定する必要があるタイムゾーン。例: **%z{GMT}%d{HH:mm:ss,SSS}**
%k	ログリソースキー (ログメッセージのローカリゼーションに使用)。
%m	ログメッセージ (例外トレースを含む)。
%s	単純なログメッセージ (例外トレースなし)。
%e	例外スタックトレース (拡張モジュール情報なし)。
%E	例外スタックトレース (拡張モジュール情報あり)。
%t	現在のスレッドの名前。
%n	改行文字。
%C	ログメソッドを呼び出すコードのクラス (低速)。
%F	ログメソッドを呼び出すクラスのファイル名 (低速)。
%l	ログメソッドを呼び出すコードのソースロケーション (低速)。
%L	ログメソッドを呼び出すコードの行番号 (低速)。
%M	ログメソッドを呼び出すコードのメソッド (低速)。
%x	ネスト化診断コンテキスト。
%X	メッセージ診断コンテキスト。
%%	リテラルパーセント (**%**) 記号 (エスケープ)。

表A.41 JSON ログフォーマッター属性

属性	説明
date-format	日付と時刻の形式パターン。有効な **java.time.format.DateTimeFormatter.ofPattern()** パターンである必要があります。デフォルトのパターンは オフセットのある ISO-8601 拡張形式です。
exception-output-type	ある場合はログに記録されたメッセージの原因が JSON 出力に追加される方法を示します。使用できる値は次のとおりです。 • **detailed** • **formatted** • **detailed-and-formatted**
key-overrides	JSON プロパティーのキー名のオーバーライドを許可します。
meta-data	JSON フォーマッターで使用されるメタデータを設定します。
pretty-print	フォーマット時にプリティープリントを使用すべきかどうか。
print-details	詳細を出力するかどうか。詳細には、ソースクラス名、ソールファイル名、ソースメソッド名、ソースモジュール名、ソースモジュールバージョン、およびソース行番号が含まれます。  **注記** 値は呼び出し元から取得されるため、詳細の出力はリソースを多く使用する可能性があります。
record-delimiter	レコードの最後を示すために使用される値。null を設定すると、レコードの最後に区切り文字が使用されません。デフォルト値は改行です。
zone-id	日付と時刻をフォーマットためのゾーン ID です。定義のない場合はシステムのデフォルトが使用されます。

表A.42 XML ログフォーマッター属性

属性	説明
date-format	日付と時刻の形式パターン。有効な **java.time.format.DateTimeFormatter.ofPattern()** パターンである必要があります。デフォルトのパターンは オフセットのある ISO-8601 拡張形式です。

属性	説明
exception-output-type	ある場合はログに記録されたメッセージの原因が XML 出力に追加される方法を示します。使用できる値は次のとおりです。 • **detailed** • **formatted** • **detailed-and-formatted**
key-overrides	XML プロパティーのキー名のオーバーライドを許可します。
meta-data	XML 形式で使用されるメタデータを設定します。プロパティーは各ログメッセージに追加されます。
namespace-uri	print-namespace 属性が true の場合、各レコードに使用される名前空間 URI を設定します。定義されている namespace-uri がなく、オーバーライドされたキーがある場合、**print-namespace** 属性が true に設定されている場合でも書き込まれるネームスペースはありません。
pretty-print	フォーマット時にプリティープリントを使用すべきかどうか。
print-details	詳細を出力するかどうか。詳細には、ソースクラス名、ソールファイル名、ソースメソッド名、ソースモジュール名、ソースモジュールバージョン、およびソース行番号が含まれます。  **注記** 値は呼び出し元から取得されるため、詳細の出力はリソースを多く使用する可能性があります。
record-delimiter	レコードの最後を示すために使用される値。null の場合、レコードの最後に区切り文字が使用されません。デフォルト値は改行です。
zone-id	日付と時刻をフォーマットためのゾーン ID です。定義のない場合はシステムのデフォルトが使用されます。

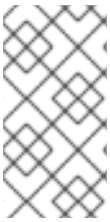
A.19. データソース接続 URL

表A.43 データソース接続 URL

データソース	接続 URL
IBM DB2	jdbc:db2://**SERVER_NAME**:**PORT**/**DATABASE_NAME**
MariaDB	jdbc:mariadb://**SERVER_NAME**:**PORT**/**DATABASE_NAME**

データソース	接続 URL
MariaDB Galera クラスター	jdbc:mariadb://**SERVER_NAME**:**PORT**,**SERVER_NAME**:**PORT**/**DATABASE_NAME**
Microsoft SQL Server	jdbc:sqlserver://**SERVER_NAME**:**PORT**;DatabaseName=**DATABASE_NAME**
MySQL	jdbc:mysql://**SERVER_NAME**:**PORT**/**DATABASE_NAME**
Oracle	jdbc:oracle:thin:@**SERVER_NAME**:**PORT**:**ORACLE_SID**
PostgreSQL	jdbc:postgresql://**SERVER_NAME**:**PORT**/**DATABASE_NAME**
Sybase	jdbc:sybase:Tds:**SERVER_NAME**:**PORT**/**DATABASE_NAME**

A.20. データソースの属性



注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/jboss_as_logging_5_0.xsd** のスキーマ定義ファイルで確認してください。

表A.44 データソースの属性

属性	データソースタイプ	説明
allocation-retry	非 XA、XA	例外が発生する前に接続の割り当てを試行する回数。デフォルトは **0** で、初回の割り当て失敗で例外が発生します。
allocation-retry-wait-millis	非 XA、XA	接続の割り当てを再試行する間隔 (ミリ秒単位)。デフォルトは **0** ミリ秒です。
allow-multiple-users	非 XA、XA	複数のユーザーが **getConnection(user, password)** メソッドを使いデータソースへアクセスするかどうか、およびこの挙動が内部プールタイプによるものであるかどうか。
authentication-context	非 XA、XA	プールの接続を区別するために使用される **javax.security.auth.Subject** を定義する Elytron 認証コンテキスト。

属性	データソースタイプ	説明
background-validation	非 XA、XA	接続がバックグラウンドスレッドで検証されるべきか、使用前に検証されるか。通常バックグラウンド検証は **validate-on-match** とともに使用されません。使用されると冗長チェックが行われます。バックグラウンド検証では、検証時とクライアントへ渡す時の間で接続が不良になる可能性があるため、アプリケーションはこの可能性に対応する必要があります。
background-validation-millis	非 XA、XA	バックグラウンド検証を実行する頻度 (ミリ秒単位)。
blocking-timeout-wait-millis	非 XA、XA	例外が発生する前に接続を待機している間にブロックする最大時間 (ミリ秒単位)。この時間を超過すると、例外が発生します。 これは、接続のロックを待っている間のみブロックし、新規接続の作成に長時間要している場合は例外は発生しません。
capacity-decrementer-class	非 XA、XA	プールの接続をデクリメントするポリシーを定義するクラス。
capacity-decrementer-properties	非 XA、XA	プールの接続をデクリメントするポリシーを定義するクラスにインジェクトされるプロパティー。
capacity-incrementer-class	非 XA、XA	プールの接続をインクリメントするポリシーを定義するクラス。
capacity-incrementer-properties	非 XA、XA	プールの接続をインクリメントするポリシーを定義するクラスにインジェクトされるプロパティー。
check-valid-connection-sql	非 XA、XA	プール接続の妥当性を確認する SQL ステートメント。これは、プールから管理接続を取得するときに呼び出される場合があります。
connectable	非 XA、XA	CMR の使用を有効にします。 これは、ローカルリソースが信頼して XA トランザクションに参加できることを意味します。

属性	データソースタイプ	説明
connection-listener-class	非 XA、XA	**org.jboss.jca.adapters.jdbc.spi.listener.ConnectionListener** を拡張するクラス名を指定します。このクラスは、接続がアプリケーションまたはプールに返される前にアクションを実行するために接続のアクティベーションおよびパッシベーションをリッスンします。コアモジュールとしての JDBC ドライバーのインストール のとおりに、2 つのリソース JAR を使用して 1 つのモジュールで指定されたクラスを JDBC ドライバーとバンドルするか、グローバルモジュールの定義 のとおりに、指定されたクラスを個別のグローバルモジュールでバンドルする必要があります。
connection-listener-property	非 XA、XA	**connection-listener-class** で指定されたクラスにインジェクトされるプロパティー。インジェクトされるプロパティーは JavaBeans の慣例に対応します。たとえば、**foo** という名前のプロパティーを指定する場合、接続リスナークラスには **String** を引数として許可するメソッド **setFoo** が必要です。
connection-properties	非 XA のみ	**Driver.connect(url, props)** メソッドに渡す任意の文字列名と値のペアである接続プロパティー。
connection-url	非 XA のみ	JDBC ドライバーの接続 URL。
credential-reference	非 XA、XA	データソース上で認証される、認証情報ストアからの認証情報。
datasource-class	非 XA のみ	JDBC データソースクラスの完全修飾名。
driver-class	非 XA のみ	JDBC ドライバークラスの完全修飾名。
driver-name	非 XA、XA	データソースが使用する JDBC ドライバーを定義します。インストールされたドライバーに一致するシンボリック名になります。ドライバーが JAR としてデプロイされた場合、名前はデプロイメントの名前になります。
elytron-enabled	非 XA、XA	接続の認証の処理に Elytron セキュリティーを有効にします。指定されたコンテキストがない場合、使用される Elytron の **authentication-context** が現在のコンテキストになります。詳細は **authentication-context** を参照してください。
enabled	非 XA、XA	データソースを有効にするかどうか。

属性	データソースタイプ	説明
enlistment-trace	非 XA、XA	エンリストメントトレースを記録するかどうか。デフォルトは **false** です。
exception-sorter-class-name	非 XA、XA	例外がエラーをブロードキャストする場合に検証するメソッドを提供する **org.jboss.jca.adapters.jdbc.ExceptionSorter** のインスタンス。
exception-sorter-properties	非 XA、XA	例外ソータープロパティー。

属性	データソースタイプ	説明
flush-strategy	非 XA、XA	エラーの場合にプールをフラッシュする方法を指定します。有効な値は以下のとおりです。 **FailingConnectionOnly** 問題のある接続のみが削除されます。これはデフォルト設定です。 **InvalidIdleConnections** 同じクレデンシャルを共有し、ValidatingManagedConnectionFactory.getInvalidConnections(...) メソッドによって無効として返される、問題のある接続とアイドル状態の接続が削除されます。 **IdleConnections** 同じ認証情報を共有する問題のある接続とアイドル状態の接続が削除されます。 **Gracefully** 同じ認証情報を共有する問題のある接続とアイドル状態の接続が削除されます。同じ認証情報を共有するアクティブな接続は、プールに戻された時点で破棄されます。 **EntirePool** 同じ認証情報を共有する問題のある接続、アイドル状態の接続、およびアクティブな接続が削除されます。この設定は本番環境のシステムには推奨されません。 **AllInvalidIdleConnections** ValidatingManagedConnectionFactory.getInvalidConnections(...) メソッドによって無効として返される、問題のある接続とアイドル状態の接続が削除されます。 **AllIdleConnections** 問題のある接続と、アイドル状態の接続すべてが削除されます。 **AllGracefully** 問題のある接続と、アイドル状態の接続すべてが削除されます。アクティブな接続は、プールに戻された時点で破棄されます。 **AllConnections** 問題のある接続、アイドル状態の接続すべて、およびアクティブな接続すべてが削除されます。この設定は本番環境のシステムには推奨されません。
idle-timeout-minutes	非 XA、XA	接続が閉じられるまでのアイドル最大時間 (分単位) を指定します。指定がない場合、デフォルトは **30** 分になります。実際の最大時間は、IdleRemover スキャン時間 (プールの最小 idle-timeout-minutes の半分) にも左右されます。
initial-pool-size	非 XA、XA	プールが保持する最初の接続数。

属性	データソースタイプ	説明
interleaving	XA のみ	XA 接続のインターリービングを有効にするかどうか。
jndi-name	非 XA、XA	データソースの一意の JNDI 名。
jta	非 XA のみ	Jakarta Transactions インテグレーションを有効にします。
max-pool-size	非 XA、XA	プールが保持可能な最大接続数。
mcp	非 XA、XA	**ManagedConnectionPool** 実装。例: **org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreArrayListManagedConnectionPool**
min-pool-size	非 XA、XA	プールが保持可能な最小接続数。
new-connection-sql	非 XA、XA	接続が接続プールに追加されたときに実行する SQL ステートメント。
no-recovery	XA のみ	リカバリーから接続プールが除外されるかどうか。
no-tx-separate-pool	XA のみ	各コンテキストに対して個別のサブプールを作成するかどうか。これは、Jakarta Transactions トランザクションの内部と外部の両方で XA 接続を使用できない一部の Oracle データソースで必要になることがあります。このオプションを使用すると、実際のプールがつ作成されるため、合計プールサイズが **max-pool-size** の倍になります。
pad-xid	XA のみ	Xid のパディングを行うかどうかを指定します。
password	非 XA、XA	新しい接続の作成時に使用するパスワード。
pool-fair	非 XA、XA	プールが fair であるかを定義します。この設定は、Jakarta Connectors の接続プールを管理するために使用される **Semaphore** クラスの一部で、リースする接続の順番が必要でない一部のユースケースでパフォーマンスが向上されます。
pool-prefill	非 XA、XA	プールをプレフィルするかどうか。
pool-use-strict-min	非 XA、XA	**min-pool-size** を厳格に考慮するかどうか。
prepared-statements-cache-size	非 XA、XA	LRU (Least Recently Used) キャッシュにある接続毎の準備済みステートメントの数。

属性	データソースタイプ	説明
query-timeout	非 XA、XA	クエリーのタイムアウト (秒単位)。デフォルトではタイムアウトはありません。
reauth-plugin-class-name	非 XA、XA	物理接続の再認証する再認証プラグイン実装の完全修飾クラス名。
reauth-plugin-properties	非 XA、XA	再認証プラグインのプロパティー。
recovery-authentication-context	XA のみ	プールの接続を区別するために使用される **javax.security.auth.Subject** を定義する Elytron 認証コンテキスト。
recovery-credential-reference	XA のみ	データソース上で認証される、認証情報ストアからの認証情報。
recovery-elytron-enabled	XA のみ	リカバリーに対する接続の認証の処理に Elytron セキュリティーを有効にします。指定された **authentication-context** がない場合、使用される Elytron の **authentication-context** が現在のコンテキストになります。詳細は **authentication-context** を参照してください。
recovery-password	XA のみ	リカバリーのリソースへの接続に使用するパスワード。
recovery-plugin-class-name	XA のみ	リカバリープラグイン実装の完全修飾クラス名。
recovery-plugin-properties	XA のみ	リカバリープラグインのプロパティー。
recovery-security-domain	XA のみ	リカバリーでリソースに接続するために使用するセキュリティードメイン。
recovery-username	XA のみ	リカバリーでリソースに接続するために使用するユーザー名。
same-rm-override	XA のみ	**javax.transaction.xa.XAResource.isSameRM(XAResource)** クラスが **true** または **false** を返すかどうか。
security-domain	非 XA、XA	認証処理を行う JAAS security-manager の名前。この名前は、JAAS ログイン設定の application-policy/name 属性に相関します。

属性	データソースタイプ	説明
set-tx-query-timeout	非 XA、XA	トランザクションがタイムアウトするまでの残り時間を基にクエリーのタイムアウトを設定するかどうかを指定します。トランザクションが存在しない場合は設定済みのクエリーのタイムアウトが使用されます。
share-prepared-statements	非 XA、XA	アプリケーションに提供されたラッパーがアプリケーションコードによって閉じられたときに、JBoss EAP が基盤の物理ステートメントを閉じたり終了せずに、キャッシュするかどうか。デフォルトは **false** です。
spy	非 XA、XA	JDBC レイヤーでスパイ機能を有効にします。この機能は、データソースへの JDBC トラフィックをすべてログに記録します。ロギングカテゴリーの **jboss.jdbc.spy** も **logging** サブシステムのログレベルである **DEBUG** に設定する必要があることに注意してください。
stale-connection-checker-class-name	非 XA、XA	**isStaleConnection(SQLException)** メソッドを提供する **org.jboss.jca.adapters.jdbc.StaleConnectionChecker** のインスタンス。このメソッドが **true** を返す場合、例外は **org.jboss.jca.adapters.jdbc.StaleConnectionException** でラップされます。
stale-connection-checker-properties	非 XA、XA	陳腐接続チェッカーのプロパティー。
statistics-enabled	非 XA、XA	ランタイム統計が有効になっているかどうか。デフォルトは **false** です。
track-statements	非 XA、XA	接続がプールへ返され、ステートメントが準備済みステートメントキャッシュへ返された時に、閉じられていないステートメントをチェックするかどうか。false の場合、ステートメントは追跡されません。有効な値は次のとおりです。 ● **true**: ステートメントと結果セットが追跡され、ステートメントが閉じられていない場合は警告が出力されます。 ● **false**: ステートメントと結果セットのいずれも追跡されません。 ● **nowarn**: ステートメントは追跡されますが、警告は出力されません (**デフォルト**)。

属性	データソースタイプ	説明
tracking	非 XA、XA	トランザクション境界にまたがる接続ハンドルを追跡するかどうか。
transaction-isolation	非 XA、XA	**java.sql.Connection** トランザクション分離レベル。有効な値は次のとおりです。 • **TRANSACTION_READ_UNCOMMITTED** • **TRANSACTION_READ_COMMITTED** • **TRANSACTION_REPEATABLE_READ** • **TRANSACTION_SERIALIZABLE** • **TRANSACTION_NONE**
url-delimiter	非 XA、XA	高可用性 (HA) データソースの connection-url にある URL の区切り文字。
url-property	XA のみ	**xa-datasource-property** 値の **URL** プロパティーのプロパティー。
url-selector-strategy-class-name	非 XA、XA	**org.jboss.jca.adapters.jdbc.URLSelectorStrategy** を実装するクラス。
use-ccm	非 XA、XA	キャッシュ接続マネージャーを有効にします。
use-fast-fail	非 XA、XA	true の場合、接続が無効であれば最初に接続を割り当てしようとした時点で失敗します。false の場合、プールが枯渇するまで再試行します。
use-java-context	非 XA、XA	データソースをグローバル JNDI にバインドするかどうか。
use-try-lock	非 XA、XA	内部ロックのタイムアウト値。ロックが使用できない場合に即座に失敗するのではなく、タイムアウトする前に設定された秒数間ロックの取得を試みます。**tryLock()** の代わりに **lock()** を使用します。
user-name	非 XA、XA	新しい接続の作成時に使用するユーザー名。
valid-connection-checker-class-name	非 XA、XA	**SQLException.isValidConnection(Connection e)** メソッドを提供して接続を検証する **org.jboss.jca.adaptors.jdbc.ValidConnectionChecker** の実装。接続が破棄されると例外が発生します。存在する場合、**check-valid-connection-sql** 属性が上書きされます。

属性	データソースタイプ	説明
valid-connection-checker-properties	非 XA、XA	有効な接続チェッカープロパティー。
validate-on-match	非 XA、XA	接続ファクトリーが管理された接続への一致を試みたときに接続の検証が実行されるかどうか。これは、使用前にクライアントの接続を検証する必要がある場合に使用する必要があります。通常、Validate-on-match は **background-validation** と使用しません。 使用すると冗長チェックが行われます。
wrap-xa-resource	XA のみ	**org.jboss.tm.XAResourceWrapper** インスタンスで XAResource をラップするかどうか。
xa-datasource-class	XA のみ	**javax.sql.XADataSource** 実装クラスの完全修飾名。
xa-datasource-properties	XA のみ	XA データソースプロパティーの文字列名と値のペア。
xa-resource-timeout	XA のみ	ゼロ以外の値は **XAResource.setTransactionTimeout** メソッドに渡されます。

表A.45 JDBC ドライバー属性

属性	データソースタイプ	説明
datasource-class-info	非 XA、XA	**jdbc-driver** の **datasource-class** および **xa-datasource-class** に使用できるプロパティー。**datasource-class** および **xa-datasource-class** 属性は、**javax.sql.DataSource** または **javax.sql.XADataSource** クラスを実装する完全修飾クラス名を定義します。定義されたクラスはさまざまなプロパティーのセッターを持つことができます。**datasource-class-info** 属性は、クラスに設定できるこれらのプロパティーをリストします。

A.21. データソースの統計

表A.46 コアプールの統計

名前	説明
ActiveCount	アクティブな接続の数。各接続はアプリケーションによって使用されているか、プールで使用可能な状態であるかのいずれかになります。
AvailableCount	プールの使用可能な接続の数。
AverageBlockingTime	プールの排他ロックの取得をブロックするために費やされた平均時間。値はミリ秒単位です。
AverageCreationTime	接続の作成に費やされた平均時間。値はミリ秒単位です。
AverageGetTime	接続の取得に費やされた平均時間。値はミリ秒単位です。
AveragePoolTime	接続がプールで費やした平均時間。この値はミリ秒単位です。
AverageUsageTime	接続の使用に費やされた平均時間。値はミリ秒単位です。
BlockingFailureCount	接続の取得に失敗した回数。
CreatedCount	作成された接続の数。
DestroyedCount	破棄された接続の数。
IdleCount	現在アイドル状態の接続数。
InUseCount	現在使用中の接続の数。
MaxCreationTime	接続の作成にかかった最大時間。値はミリ秒単位です。
MaxGetTime	接続取得の最大時間。値はミリ秒単位です。
MaxPoolTime	プールの接続の最大時間。値はミリ秒単位です。
MaxUsageTime	接続使用の最大時間。値はミリ秒単位です。
MaxUsedCount	使用される接続の最大数。
MaxWaitCount	同時に接続を待機する要求の最大数。
MaxWaitTime	プールの排他ロックの待機に費やされた最大時間。値はミリ秒単位です。
TimedOut	タイムアウトした接続の数。

名前	説明
TotalBlockingTime	プールの排他ロックの待機に費やされた合計時間。値はミリ秒単位です。
TotalCreationTime	接続の作成に費やされた合計時間。値はミリ秒単位です。
TotalGetTime	接続の取得に費やされた合計時間。値はミリ秒単位です。
TotalPoolTime	プールの接続によって費やされた合計時間。値はミリ秒単位です。
TotalUsageTime	接続の使用に費やされた合計時間。値はミリ秒単位です。
WaitCount	接続の取得を待つ必要のあるリクエストの数。
XACommitAverageTime	XAResource commit 呼び出しの平均時間。値はミリ秒単位です。
XACommitCount	XAResource commit 呼び出しの数。
XACommitMaxTime	XAResource commit 呼び出しの最大時間。値はミリ秒単位です。
XACommitTotalTime	すべての XAResource commit 呼び出しの合計時間。値はミリ秒単位です。
XAEndAverageTime	XAResource end 呼び出しの平均時間。値はミリ秒単位です。
XAEndCount	XAResource end 呼び出しの数。
XAEndMaxTime	XAResource end 呼び出しの最大時間。値はミリ秒単位です。
XAEndTotalTime	すべての XAResource end 呼び出しの合計時間。値はミリ秒単位です。
XAForgetAverageTime	XAResource forget 呼び出しの平均時間。値はミリ秒単位です。
XAForgetCount	XAResource forget 呼び出しの数。
XAForgetMaxTime	XAResource forget 呼び出しの最大時間。値はミリ秒単位です。
XAForgetTotalTime	すべての XAResource forget 呼び出しの合計時間。値はミリ秒単位です。
XAPrepareAverageTime	XAResource prepare 呼び出しの平均時間。値はミリ秒単位です。

名前	説明
XAPrepareCount	XAResource prepare 呼び出しの数。
XAPrepareMaxTime	XAResource prepare 呼び出しの最大時間。値はミリ秒単位です。
XAPrepareTotalTime	すべての XAResource prepare 呼び出しの合計時間。値はミリ秒単位です。
XARecoverAverageTime	XAResource recover 呼び出しの平均時間。値はミリ秒単位です。
XARecoverCount	XAResource recover 呼び出しの数。
XARecoverMaxTime	XAResource recover 呼び出しの最大時間。値はミリ秒単位です。
XARecoverTotalTime	すべての XAResource recover 呼び出しの合計時間。値はミリ秒単位です。
XARollbackAverageTime	XAResource rollback 呼び出しの平均時間。値はミリ秒単位です。
XARollbackCount	XAResource rollback 呼び出しの数。
XARollbackMaxTime	XAResource rollback 呼び出しの最大時間。値はミリ秒単位です。
XARollbackTotalTime	すべての XAResource rollback 呼び出しの合計時間。値はミリ秒単位です。
XAStartAverageTime	XAResource start 呼び出しの平均時間。値はミリ秒単位です。
XAStartCount	XAResource start 呼び出しの数。
XAStartMaxTime	XAResource start 呼び出しの最大時間。値はミリ秒単位です。
XAStartTotalTime	すべての XAResource start 呼び出しの合計時間。値はミリ秒単位です。

表A.47 JDBC の統計

名前	説明
PreparedStatementCacheAccessCount	ステートメントキャッシュがアクセスされた回数。
PreparedStatementCacheAddCount	ステートメントキャッシュに追加されたステートメントの数。

名前	説明
PreparedStatementCacheCurrentSize	ステートメントキャッシュに現在キャッシュされた準備済みおよび呼び出し可能ステートメントの数。
PreparedStatementCacheDeleteCount	キャッシュから破棄されたステートメントの数。
PreparedStatementCacheHitCount	キャッシュからのステートメントが使用された回数。
PreparedStatementCacheMissCount	ステートメント要求がキャッシュのステートメントと一致しなかった回数。

A.22. AGROAL データソースの属性

注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-agroal_1_0.xsd** のスキーマ定義ファイルで確認してください。

表A.48 Agroal データソースの属性

属性	説明
connectable	このデータソースで CMR (Commit Markable Resource) 機能を有効にするかどうか。これは非 XA データソースのみに適用されます。
jndi-name	データソースの JNDI 名 を指定します。
jta	Jakarta Transactions インテグレーションを有効にするかどうか。これは非 XA データソースのみに適用されます。
statistics-enabled	このデータソース統計を有効にするかどうか。デフォルトは **false** です。

表A.49 Agroal データソース接続ファクトリーの属性

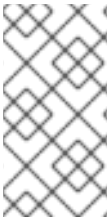
属性	説明
authentication-context	**elytron** サブシステムの認証コンテキストへの参照。
connection-properties	接続の作成時に JDBC ドライバーに渡されるプロパティー。
credential-reference	認証に使用する認証情報ストアからの認証情報。

属性	説明
driver	JDBC ドライバーへの一意な参照。
new-connection-sql	作成後に接続で実行される SQL ステートメント。
password	データベースとの基本認証に使用するパスワード。
transaction-isolation	使用する **java.sql.Connection** トランザクション分離レベルを設定します。
url	JDBC ドライバーの接続 URL。
username	データベースとの基本認証に使用するユーザー名。

表A.50 Agroal データソース接続プールの属性

属性	説明
background-validation	バックグラウンドで実行されるバリデーションの合間 (ミリ秒単位)。
blocking-timeout	例外が発生する前に接続を待機している間にブロックする最大時間 (ミリ秒単位)。
idle-removal	削除する前に接続がアイドル状態である必要がある期間 (分単位)。
initial-size	プールが保持する最初の接続数。
leak-detection	漏えいを警告する前に接続を保持しなければならない期間 (ミリ秒単位)。
max-size	プールの接続の最大数。
min-size	プールが保持すべき最小接続数。

A.23. トランザクションマネージャーの設定オプション



注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-txn_5_0.xsd** のスキーマ定義ファイルで確認してください。

表A.51 Transactions サブシステムの属性

属性	説明
default-timeout	デフォルトのトランザクションタイムアウトです。デフォルトでは **300** 秒に設定されています。トランザクションごとにプログラムで上書きできます。
enable-statistics	**statistics-enabled が導入されたため非推奨になりました。**
enable-tsm-status	アウトオブプロセスのリカバリーに使用される TSM (トランザクションステータスマネージャー) サービスを有効にするかどうか。アウトオブプロセスのリカバリーマネージャーを実行してメモリー内ではなく異なるプロセスから **ActionStatusService** にアクセスすることはサポートされないため、このオプションはサポート対象外になります。
hornetq-store-enable-async-io	**journal-store-enable-async-io が導入されたため非推奨になりました。**
jdbc-action-store-drop-table	JDBC アクションストアがテーブルをドロップするか。デフォルトは **false** です。
jdbc-action-store-table-prefix	設定された JDBC アクションストアにトランザクションログを書き込むために使用されるテーブルのオプションの接頭辞。
jdbc-communication-store-drop-table	JDBC コミュニケーションストアがテーブルをドロップするか。デフォルトは **false** です。
jdbc-communication-store-table-prefix	設定された JDBC コミュニケーションストアにトランザクションログを書き込むために使用されるテーブルのオプションの接頭辞。
jdbc-state-store-drop-table	JDBC ステートストアがテーブルをドロップするか。デフォルトは **false** です。
jdbc-state-store-table-prefix	設定された JDBC ステートストアにトランザクションログを書き込むために使用されるテーブルのオプションの接頭辞。
jdbc-store-datasource	使用される非 XA データソースの名前。データソースは **datasources** サブシステムで定義する必要があります。
journal-store-enable-async-io	ジャーナルストアに対して **AsyncIO** を有効にするかどうか。デフォルトは **false** です。この設定を有効にするにはサーバーを再起動する必要があります。
jts	Java Transaction Service (JTS) トランザクションを使用するかどうかを指定します。デフォルト値は **false** で、Jakarta Transactions トランザクションのみを使用します。
maximum-timeout	トランザクションに無制限のタイムアウトを意味する **0** がトランザクションタイムアウトと h して設定されている場合、トランザクションマネージャーは代わりにこの属性によって設定された値を使用します。デフォルトは **31536000** 秒 (365 日) です。

属性	説明
node-identifier	トランザクションマネージャーのノード識別子。このオプションが設定されていないと、サーバーの起動時に警告が表示されます。このオプションは以下の場合に必要になります。 ● JTS 対 JTS の通信 ● 2 つのトランザクションマネージャーが共有のリソースマネージャーにアクセスする場合 ● 2 つのトランザクションマネージャーが共有のオブジェクトマネージャーにアクセスする場合 リカバリー中にデータの整合性を維持する必要があるため、各トランザクションマネージャーの node-identifier は一意である必要があります。複数のノードが同じリソースマネージャーと対話したり、トランザクションオブジェクトストアを共有したりするため、node-identifier は Jakarta Transactions に対しても一意である必要があります。
object-store-path	トランザクションマネージャーオブジェクトストアがデータを格納するファイルシステムの相対または絶対パスです。デフォルトは **object-store-relative-to** パラメーターに相対する値になります。**object-store-relative-to** を空の文字列に設定すると、この値は絶対パスとして処理されます。
object-store-relative-to	ドメインモデルのグローバルなパス設定を参照します。デフォルト値は、JBoss EAP のデータディレクトリーで、**jboss.server.data.dir** プロパティーの値です。デフォルトは、マネージドドメインの場合は **EAP_HOME/domain/data/**、スタンドアロンサーバーインスタンスの場合は **EAP_HOME/standalone/data/** です。オブジェクトストアの **object-store-path** トランザクションマネージャー属性の値はこのパスに相対的です。**object-store-path** が絶対パスとして処理されるようにするには、この属性を空の文字列に設定します。
process-id-socket-binding	トランザクションマネージャーがソケットベースのプロセス ID を使用する必要がある場合に使用するソケットバインディング設定の名前。**process-id-uuid** が **true** に設定された場合は **undefined** になります。 それ以外の場合は、設定する必要があります。
process-id-socket-max-ports	トランザクションマネージャーは、各トランザクションログに対し一意の識別子を作成します。一意の識別子を生成するメカニズムは 2 種類あります。 ソケットベースのメカニズムとプロセスのプロセス識別子をベースにしたメカニズムです。 ソケットベースの識別子の場合、あるソケットを開くと、そのポート番号が識別子に使用されます。ポートがすでに使用されている場合は、空きのポートが見つかるまで次のポートがプローブされます。process-id-socket-max-ports は失敗するまでトランザクションマネージャーが試行するソケットの最大数を表します。デフォルト値は **10** です。

属性	説明
process-id-uuid	**true** に設定すると、プロセス識別子を使用して各トランザクションに一意の識別子が作成されます。そうでない場合は、ソケットベースのメカニズムが使用されます。デフォルト値は **true** です。詳細は、**process-id-socket-max-ports** を参照してください。**process-id-socket-binding** を有効にするには、**process-id-uuid** を **false** に設定します。
recovery-listener	トランザクションリカバリーのプロセスがネットワークソケットをリッスンするかどうかを指定します。デフォルトは **false** です。
socket-binding	**recovery-listener** が **true** に設定されている場合にトランザクション周期リカバリーリスナーによって使用されるソケットバインディングの名前を指定します。
statistics-enabled	統計を有効にするべきか。デフォルトは **false** です。
status-socket-binding	トランザクションステータスマネージャーに使用するソケットバインディングを指定します。この設定オプションはサポートされません。
use-hornetq-store	**use-journal-store が導入されたため非推奨になりました。**
use-jdbc-store	トランザクションログの書き込みに JDBC ストアを使用します。有効にする場合は **true**、デフォルトのログストアタイプを使用する場合は **false** を設定します。
use-journal-store	ファイルベースのストレージの代わりに Apache ActiveMQ Artemis のジャーナルストレージメカニズムをトランザクションログに使用します。デフォルトでは無効になっていますが、I/O パフォーマンスが改善されます。別々のトランザクションマネージャーで JTS トランザクションを使用することは推奨されません。このオプションの変更を反映するには **shutdown** コマンドを使用してサーバーを再起動する必要があります。

表A.52 ログストアの属性

属性	説明
expose-all-logs	すべてのログを公開するかどうか。デフォルトは **false** で、トランザクションログのサブセットのみが公開されます。
type	ロギングストアの実装タイプを指定します。デフォルトは **default** です。

表A.53 CMR (Commit Markable Resurce) の属性

属性	説明
batch-size	この CMR リソースのバッチサイズ。デフォルトは **100** です。
immediate-cleanup	この CMR リソースの即時クリーンアップを実行するかどうか。デフォルトは **true** です。
jndi-name	この CMR リソースの JNDI 名。
name	XID を格納するテーブルの名前。デフォルトは **xids** です。

A.24. IIOP サブシステムの属性

注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-iiop-openjdk_3_0.xsd** のスキーマ定義ファイルで確認してください。

表A.54 IIOP サブシステムの属性

属性	説明
add-component-via-interceptor	SSL コンポーネントが IOR インターセプターによって追加されるべきかどうかを示します。**非推奨**。
auth-method	認証方法。有効な値は **none** および **username_password** です。
authentication-context	セキュリティーイニシャライザーが **elytron** に設定されたときに使用する認証コンテキストの名前。
caller-propagation	呼び出し元の ID を SAS コンテキストで伝播する必要があるかどうかを示します。有効な値は **none** および **supported** です。
client-requires	クライアント SSL がパラメーターを必要とすることを示す値。有効な値は **None**、**ServerAuth**、**ClientAuth**、および **MutualAuth** です。**非推奨のため、代わりに client-requires-ssl を使用してください。**
client-requires-ssl	サーバーからの IIOP 接続に SSL が必要であるかどうかを示します。
client-ssl-context	クライアント側の SSL ソケットの作成に使用される SSL コンテキストの名前。

属性	説明
client-supports	クライアント SSL がサポートされるパラメーターを示す値。有効な値は **None**、**ServerAuth**、**ClientAuth**、および **MutualAuth** です。**非推奨のため、代わりに client-requires-ssl を使用してください。**
confidentiality	トランスポートに機密性の保護が必要であるかどうかを示します。有効な値は **none**、**supported**、および **required** です。**非推奨のため、代わりに server-requires-ssl を使用してください。**
detect-misordering	トランスポートに間違った順序の検出が必要であるかどうかを示します。有効な値は **none**、**supported**、および **required** です。**非推奨のため、代わりに server-requires-ssl を使用してください。**
detect-replay	トランスポートにリプレイの検出が必要であるかどうかを示します。有効な値は **none**、**supported**、および **required** です。**非推奨のため、代わりに server-requires-ssl を使用してください。**
export-corbaloc	ルートコンテキストを **corbaloc::address:port/NameService** としてエクスポートすべきかどうかを示します。
giop-version	使用される GIOP バージョン。
high-water-mark	TCP 接続キャッシュパラメーター。接続数がこの値を超えるたびに、ORB は接続を再利用しようとします。再利用される接続の数は、**number-to-reclaim** プロパティーによって指定されます。このプロパティーが設定されていない場合は、デフォルトの OpenJDK ORB が使用されます。
integrity	トランスポートに整合性の保護が必要であるかどうかを示します。有効な値は **none**、**supported**、および **required** です。**非推奨のため、代わりに server-requires-ssl を使用してください。**
number-to-reclaim	TCP 接続キャッシュパラメーター。接続数が **high-water-mark** プロパティーを超えるたびに、ORB は接続を再利用しようとします。再利用される接続の数は、このプロパティーによって指定されます。このプロパティーが設定されていない場合は、デフォルトの OpenJDK ORB が使用されます。

属性	説明
persistent-server-id	サーバーの永続 ID。永続オブジェクト参照はサーバーの多くのアクティベーションで有効であり、このプロパティーを使用してこの ID を識別します。結果として、同じサーバーの多くのアクティベーションではこのプロパティーを同じ値に設定し、同じホストで実行されている異なる複数のサーバーインスタンスに異なるサーバー ID を割り当てる必要があります。
properties	汎用キー/値のプロパティーリスト。
realm	認証サービスのレルム名。
required	認証が必要であるかどうかを示します。
root-context	ネーミングサービスのルートコンテキスト
security	セキュリティーインターセプターをインストールするかどうかを示します。有効な値は **client**、**identity**、**elytron**、および **none** です。
security-domain	SSL 接続の確立に使用されるキーストアとトラストストアを保持するセキュリティードメインの名前。
server-requires	サーバー SSL がパラメーターを必要とすることを示す値。有効な値は **None**、**ServerAuth**、**ClientAuth**、および **MutualAuth** です。**非推奨のため、代わりに server-requires-ssl を使用してください。**
server-requires-ssl	サーバーへの IIOP 接続に SSL が必要であるかどうかを示します。
server-ssl-context	サーバー側の SSL ソケットの作成に使用される SSL コンテキストの名前。
server-supports	サーバー SSL がサポートされるパラメーターを示す値。有効な値は **None**、**ServerAuth**、**ClientAuth**、および **MutualAuth** です。**非推奨のため、代わりに server-requires-ssl を使用してください。**
socket-binding	ORB ポートを指定するソケットバインディング設定名
ssl-socket-binding	ORB SSL を指定するソケットバインディング設定名
support-ssl	SSL がサポートされるかどうかを示します。

属性	説明
transactions	トランザクションインターセプターがインストールされるかどうかを示します。有効な値は **full**、**spec**、および **none** です。値が **full** の場合は JTS が有効になり、**spec** の場合は受信トランザクションコンテキストを拒否する JTS でない仕様に準拠するモードを有効にします。
trust-in-client	トランスポートの確立にクライアントの信頼が必要であるかどうかを示します。有効な値は **none**、**supported**、および **required** です。**非推奨のため、代わりに server-requires-ssl を使用してください。**
trust-in-target	トランスポートの確立にターゲットの信頼が必要であるかどうかを示します。有効な値は **none** および **supported** です。**非推奨のため、代わりに server-requires-ssl を使用してください。**

A.25. リソースアダプターの属性

以下の表はリソースアダプターの属性を示しています。

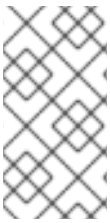
注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-resource-adapters_5_0.xsd** のスキーマ定義ファイルで確認してください。

表A.55 主な属性

属性	説明
archive	リソースアダプターアーカイブ。
beanvalidationgroups	使用する必要がある Bean バリデーショングループ。
bootstrap-context	使用する必要があるブートストラップコンテキストの一意な名前。
config-properties	カスタム定義の設定プロパティー。
module	リソースアダプターがロードされるモジュール。
statistics-enabled	ランタイム統計が有効になっているかどうか。
transaction-support	リソースアダプターのトランザクションサポートレベル。有効な値は **NoTransaction**、**LocalTransaction**、または **XATransaction** です。

属性	説明
wm-elytron-security-domain	使用される必要がある Elytron セキュリティードメインの名前を定義します
wm-security	このリソースアダプターの **wm.security** をオンまたはオフにします。false の場合、デフォルトを含む **wm-security-*** パラメーターが無視されます。
wm-security-default-groups	使用された **Subject** インスタンスに追加する必要があるデフォルトのグループリスト。
wm-security-default-principal	使用された **Subject** インスタンスに追加する必要があるデフォルトのプリンシパルリスト。
wm-security-domain	使用する必要のあるセキュリティードメインの名前。
wm-security-mapping-groups	グループマッピングのリスト。
wm-security-mapping-required	セキュリティー認証情報にマッピングが必要であるかどうかを定義します。
wm-security-mapping-users	ユーザーマッピングのリスト。



注記

リソースアダプターが **bootstrap-context** とともに、**elytron-enabled** が **true** に設定されているワークマネージャーを使用している場合、セキュリティードメインの仕様に **wm-security-domain** 属性ではなく **wm-elytron-security-domain** 属性を使用する必要があります。

表A.56 admin-objects Attributes

属性	説明
class-name	管理オブジェクトの完全修飾クラス名。
enabled	管理オブジェクトを有効にする必要があるかを指定します。
jndi-name	管理オブジェクトの JNDI 名。
use-java-context	false に設定するとオブジェクトがグローバル JNDI にバインドされます。

表A.57 connection-definitions Attributes

属性	説明
allocation-retry	接続の割り当てを再試行する回数を示します。 この回数を超えると例外が発生します。
allocation-retry-wait-millis	接続の割り当てを再試行する間隔 (ミリ秒単位)。
authentication-context	プールの接続を区別するために使用される **javax.security.auth.Subject** を定義する Elytron 認証コンテキスト。
authentication-context-and-application	プール内の接続を区別するために、**getConnection(user, pw)** や **Subject** などからのアプリケーションによって提供されたパラメーターを使用するかどうかを示します。これらのパラメーターは、設定済みの **authentication-context** を使用するときに Elytron によって認証後に提供されます。
background-validation	接続をバックグラウンドで検証するか、使用前に検証するかを指定します。値の変更後にサーバーを再起動する必要があります。
background-validation-millis	バックグラウンド検証を実行する期間 (ミリ秒単位)。値の変更後にサーバーを再起動する必要があります。
blocking-timeout-wait-millis	例外が発生する前に接続を待機している間にブロックする最大時間 (ミリ秒単位)。この時間を超過すると、例外が発生します。 これは、接続のロックを待っている間のみブロックし、新規接続の作成に長時間要している場合は例外は発生しません。
capacity-decrementer-class	プールの接続をデクリメントするポリシーを定義するクラス。
capacity-decrementer-properties	プールの接続をデクリメントするポリシーを定義するクラスにインジェクトするプロパティー。
capacity-incrementer-class	プールの接続をインクリメントするポリシーを定義するクラス。
capacity-incrementer-properties	プールの接続をインクリメントするポリシーを定義するクラスにインジェクトするプロパティー。
class-name	管理された接続ファクトリーまたは管理オブジェクトの完全修飾クラス名。
connectable	CMR の使用を有効にします。この機能により、ローカルリソースが信頼して XA トランザクションに参加できます。
elytron-enabled	接続の認証の処理に Elytron セキュリティーを有効にします。指定されたコンテキストがない場合、使用される Elytron の **authentication-context** が現在のコンテキストになります。詳細は **authentication-context** を参照してください。
enabled	リソースアダプターを有効にするべきかどうかを指定します。

属性	説明
enlistment	リソースアダプターによってサポートされる場合に lazy enlistment (レイジーエンリストメント) が使用されるべきであるかどうかを指定します。
enlistment-trace	JBoss EAP または IronJacamar がエンリストメントトレースを記録すべきかどうかを指定します。デフォルトは **false** です。
flush-strategy	エラーの場合にプールをフラッシュする方法を指定します。有効な値は以下のとおりです。 **FailingConnectionOnly** 問題のある接続のみが削除されます。これはデフォルト設定です。 **InvalidIdleConnections** 同じクレデンシャルを共有し、ValidatingManagedConnectionFactory.getInvalidConnections(...) メソッドによって無効として返される、問題のある接続とアイドル状態の接続が削除されます。 **IdleConnections** 同じ認証情報を共有する問題のある接続とアイドル状態の接続が削除されます。 **Gracefully** 同じ認証情報を共有する問題のある接続とアイドル状態の接続が削除されます。同じ認証情報を共有するアクティブな接続は、プールに戻された時点で破棄されます。 **EntirePool** 同じ認証情報を共有する問題のある接続、アイドル状態の接続、およびアクティブな接続が削除されます。この設定は本番環境のシステムには推奨されません。 **AllInvalidIdleConnections** ValidatingManagedConnectionFactory.getInvalidConnections(...) メソッドによって無効として返される、問題のある接続とアイドル状態の接続が削除されます。 **AllIdleConnections** 問題のある接続と、アイドル状態の接続すべてが削除されます。 **AllGracefully** 問題のある接続と、アイドル状態の接続すべてが削除されます。アクティブな接続は、プールに戻された時点で破棄されます。 **AllConnections** 問題のある接続、アイドル状態の接続すべて、およびアクティブな接続すべてが削除されます。この設定は本番環境のシステムには推奨されません。
idle-timeout-minutes	接続が閉じられるまでのアイドル最大時間 (分単位) を指定します。実際の最大時間は、**IdleRemover** スキャン時間 (プールの最小 **idle-timeout-minutes** の半分) に基づきます。値の変更後にサーバーを再起動する必要があります。
initial-pool-size	プールが保持する最初の接続数。
interleaving	XA 接続のインターリービングを有効にするかどうかを指定します。

属性	説明
jndi-name	接続ファクトリーの JNDI 名。
max-pool-size	プールの最大接続数。各サブプールではこの値を超える接続は作成されません。
mcp	**ManagedConnectionPool** 実装。例: **org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreArrayListManagedConnectionPool**
min-pool-size	プールの最小接続数。
no-recovery	接続プールがリカバリーから除外されるべきであるかどうかを指定します。
no-tx-separate-pool	Oracle では、XA 接続を Jakarta Transactions トランザクションの内部と外部の両方で使用することが推奨されません。この問題を回避するには、異なるコンテキストに個別のサブプールを作成します。
pad-xid	Xid のパディングを行うべきかどうかを指定します。
pool-fair	プールの使用が公正であるべきかどうかを指定します。
pool-prefill	プールをプレフィルすべきかどうかを指定します。値の変更後にサーバーを再起動する必要があります。
pool-use-strict-min	**min-pool-size** を厳格に考慮するかどうかを指定します。
recovery-authentication-context	リカバリーに使用される Elytron 認証コンテキスト。**authentication-context** の指定がない場合、現在の認証コンテキストが使用されます。
recovery-credential-reference	接続のリカバリーで認証される、認証情報ストアからの認証情報。
recovery-elytron-enabled	Elytron 認証コンテキストがリカバリーに使用されることを示します。デフォルトは **false** です。
recovery-password	リカバリーに使用されるパスワード。
recovery-plugin-class-name	リカバリープラグイン実装の完全修飾クラス名。
recovery-plugin-properties	リカバリープラグインのプロパティー。
recovery-security-domain	リカバリーに使用されるセキュリティードメイン。
recovery-username	リカバリーに使用されるユーザー名。

属性	説明
same-rm-override	**javax.transaction.xa.XAResource.isSameRM(XAResource)** が true または false を返すかどうかを無条件に設定します。
security-application	プール内の接続を区別するために、アプリケーションにより提供されたパラメーター (**getConnection(user, pw)** からなど) を使用することを指定します。
security-domain	プール内の接続を区別するために使用される **javax.security.auth.Subject** を定義するセキュリティードメイン。
security-domain-and-application	プール内の接続を区別するために、**getConnection(user, pw)** または **Subject** など、セキュリティードメインからのアプリケーションが提供するパラメーターを使用するかどうかを示します。
sharable	共有可能な接続の使用を有効にします。 サポートされる場合はレイジーアソシエーションが有効になります。
tracking	IronJacamar がトランザクション境界にまたがって接続ハンドルを追跡するかどうかを指定します。
use-ccm	キャッシュされた接続マネージャーの使用を有効にします。
use-fast-fail	**true** に設定すると、無効な場合に接続の割り当てが初回で失敗します。**false** に設定すると、プールの潜在的な接続をすべて使い果たすまで試行を継続します。
use-java-context	**false** に設定するとオブジェクトがグローバル JNDI にバインドされます。
validate-on-match	接続ファクトリーが管理された接続への一致を試みたときに接続の検証を実行すべきかどうかを指定します。通常、バックグラウンド検証の使用のみに限定されます。
wrap-xa-resource	**XAResource** インスタンスを **org.jboss.tm.XAResourceWrapper** インスタンスでラップするかどうかを指定します。
xa-resource-timeout	値は **XAResource.setTransactionTimeout()** に渡されます (秒単位)。デフォルトは **0** です。

A.26. リソースアダプターの統計

表A.58 リソースアダプターの統計

名前	説明

名前	説明
ActiveCount	アクティブな接続の数。各接続はアプリケーションによって使用されているか、プールで使用可能な状態であるかのいずれかになります。
AvailableCount	プールの使用可能な接続の数。
AverageBlockingTime	プールの排他ロックの取得をブロックするために費やされた平均時間。値はミリ秒単位です。
AverageCreationTime	接続の作成に費やされた平均時間。値はミリ秒単位です。
CreatedCount	作成された接続の数。
DestroyedCount	破棄された接続の数。
InUseCount	現在使用中の接続の数。
MaxCreationTime	接続の作成にかかった最大時間。値はミリ秒単位です。
MaxUsedCount	使用される接続の最大数。
MaxWaitCount	同時に接続を待機する要求の最大数。
MaxWaitTime	プールの排他ロックの待機に費やされた最大時間。
TimedOut	タイムアウトした接続の数。
TotalBlockingTime	プールの排他ロックの待機に費やされた合計時間。値はミリ秒単位です。
TotalCreationTime	接続の作成に費やされた合計時間。値はミリ秒単位です。
WaitCount	接続を待機する必要がある要求の数。

A.27. UNDERTOW サブシステムの属性

undertow サブシステムのさまざまな要素の属性は以下の表を参照してください。

注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-undertow_4_0.xsd** のスキーマ定義ファイルで確認してください。

- 主な属性
- アプリケーションセキュリティードメインの属性

- バッファーキャッシュの属性
- バイトバッファープールの属性
- サーブレットコンテナーの属性
- フィルターの属性
- ハンドラーの属性
- サーバーの属性

表A.59 undertow の主な属性

属性	デフォルト	説明
default-security-domain	other	Web デプロイメントによって使用されるデフォルトのセキュリティードメイン。
default-server	default-server	デプロイメントに使用するデフォルトのサーバー。
default-servlet-container	default	デプロイメントに使用するデフォルトのサーブレットコンテナー。
default-virtual-host	default-host	デプロイメントに使用するデフォルトの仮想ホスト。
instance-id	${jboss.node.name}	クラスターインスタンス ID。
obfuscate-session-route	true	サーバーのルーティング中に instance-id 値を難読化するかどうか。instance-id 値が変更されない限り、難読化されたサーバールートはサーバーの再起動後も変更されません。
statistics-enabled	false	統計を有効にするかどうか。

アプリケーションセキュリティードメインの属性

アプリケーションセキュリティードメインの属性の構造は以下のとおりです。

- **application-security-domain**
  - **setting**
    - **single-sign-on**

application-security-domain 属性

表A.60 application-security-domain 属性

属性	デフォルト	説明
enable-jacc	false	Jakarta Authorization を使用して承認を有効にします。
enable-jaspi	true	関連付けられたデプロイメントに Jakarta Authentication を有効にします。
http-authentication-factory		マップされたセキュリティードメインを参照するデプロイメントで使用される HTTP 認証ファクトリー。
integrated-jaspi	true	**integrated-jaspi** を使用するべきかどうか。Jakarta Authentication 認証時に **true** に設定すると、アイデンティティーはデプロイメントによって参照される **SecurityDomain** からロードされます。**false** に設定されると、アドホックアイデンティティーが代わりに作成されます。
override-deployment-config	false	デプロイメントの認証設定がファクトリーによってオーバーライドされるべきであるかどうか。
referencing-deployments		現在このマッピングを参照しているデプロイメント。
security-domain		デプロイメントによって使用される **SecurityDomain**。

single-sign-on 属性

表A.61 single-sign-on 属性

属性	デフォルト	説明
client-ssl-context		バックチャネルログアウト接続のセキュア化に使用される SSL コンテキストへの参照。
cookie-name	JSESSIONIDSSO	クッキーの名前。
credential-reference		プライベートキーエントリーを復号化するための認証情報参照。
domain		使用されるクッキードメイン。
http-only	false	クッキーの httpOnly 属性を設定します。

属性	デフォルト	説明
key-alias		バックチャネルログアウト接続の署名および検証に使用されるプライベートキーエントリーのエイリアス。
key-store		プライベートキーエントリーが含まれるキーストアへの参照。
path	/	クッキーのパス。
secure	false	クッキーの secure 属性を設定します。

バッファーキャッシュの属性

表A.62 buffer-cache 属性

属性	デフォルト	説明
buffer-size	1024	バッファーのサイズ。バッファーが小さいと領域をより効率的に使用できます。
buffers-per-region	1024	リージョンごとのバッファーの数。
max-regions	10	リージョンの最大数。キャッシングに使用できる最大メモリー容量を制御します。

バイトバッファープールの属性

表A.63 byte-buffer-pool 属性

属性	デフォルト	説明
buffer-size		各バッファースライスのサイズ (バイト単位)。指定のない場合は、以下のようにシステムで利用できる RAM を基にサイズが設定されます。 ● RAM が 64 MB 未満の場合は 512 バイト ● RAM が 64 - 128 MB の場合は 1024 バイト (1 KB) ● RAM が 128 MB を超える場合は 16384 バイト (16 KB) この属性のパフォーマンス調整は、JBoss EAP の**Performance Tuning Guide**の Configuring Buffer Pools を参照してください。

属性	デフォルト	説明
direct		このバッファーがダイレクトまたはヒーププールであることを示すブール値。指定のない場合は、システムで使用できる RAM を基にして値が設定されます。 ● 使用できる RAM が < 64MB の場合、値は **false** に設定されます。 ● 使用できる RAM が >= 64MB の場合、値は **true** に設定されます。 ダイレクトプールには対応するヒーププールがあることにも注意してください。
leak-detection-percent	0	漏えい検出に割り当てられるバッファーの割合 (パーセント)。
max-pool-size		プールの保持するバッファーの最大数。この制限を超えるバッファーを割り当てすることができますが、プールが満杯の場合は保持されません。
thread-local-cache-size	12	スレッドごとのキャッシュのサイズ。これは最大サイズで、スレッドが実際にバッファーを割り当てる場合にスマートサイズを使用してバッファーをスレッドにのみ保持します。

サーブレットコンテナーの属性

サーブレットコンテナーコンポーネントの構造は次のとおりです。

- servlet-container
  - mime-mapping
  - setting
    - crawler-session-management
    - jsp
    - persistent-sessions
    - session-cookie
    - websockets
  - welcome-file

servlet-container 属性

表A.64 servlet-container 属性

属性	デフォルト	説明
allow-non-standard-wrappers	false	標準のラッパークラスを拡張しないリクエストおよび応答ラッパーが使用可能であるかどうか。
default-buffer-cache	default	静的リソースのキャッシュに使用するバッファーキャッシュ。
default-cookie-version	0	アプリケーションによって作成されるクッキーに使用するデフォルトのクッキーバージョン。
default-encoding		デプロイされたすべてのアプリケーションに使用するデフォルトのエンコード。
default-session-timeout	30	コンテナーにデプロイされたすべてのアプリケーションに対するデフォルトのセッションタイムアウト (分単位)。
directory-listing		デフォルトのサーブレットにディレクトリーリスティングを有効にするかどうか。
disable-caching-for-secured-pages	true	ヘッダーを設定してセキュア化されたページのキャッシュを無効にするかどうか。無効にすると機密性の高いページが中間者によってキャッシュされる可能性があるため、セキュリティー上の問題が発生することがあります。
disable-file-watch-service	false	**true** に設定した場合、デプロイメント形式のデプロイメントの変更を監視するためにファイルウォッチサービスは使用されません。この属性は、**io.undertow.disable-file-system-watcher** システムプロパティーをオーバーライドします。
disable-session-id-reuse	false	**true** に設定した場合、未知のセッション ID は再使用されず、新しいセッション ID が生成されます。**false** に設定した場合、同じサーバー上のアプリケーション間で同じ ID を共有できるようにするため別のデプロイメントのセッションマネージャーに存在する場合のみセッション ID が再使用されます。
eager-filter-initialization	false	最初にリクエストされたときではなく、デプロイメントの開始時に filter init() を呼び出すかどうか。
ignore-flush	false	サーブレット出力ストリームでのフラッシュを無視します。ほとんどの場合でパフォーマンスに悪影響を与えます。
max-sessions		1 度にアクティブにできるセッションの最大数。

属性	デフォルト	説明
proactive-authentication	true	プロアクティブ認証を使用すべきかどうか。**true** の場合、認証情報のあるユーザーは常に認証されます。
session-id-length	30	セッション ID が長いほどセキュアになります。この値は、生成されたセッション ID の長さをバイト単位で指定します。システムは生成されたセッション ID を Base64 文字列としてエンコードし、結果をセッション ID クッキーとしてクライアントに提供します。この処理により、サーバーは最初に生成したセッション ID よりも約 33% 長いクッキー値をクライアントに送信します。たとえば、セッション ID の長さが 30 の場合、クッキーの値の長さは 40 になります。
stack-trace-on-error	local-only	エラーの発生時にスタックトレースのあるエラーページを生成するかどうか。値は **all**、**none**、および **local-only** です。
use-listener-encoding	false	リスナーで定義されたエンコードを使用します。

mime-mapping 属性

表A.65 mime-mapping 属性

属性	デフォルト	説明
value		このマッピングの mime タイプ。

crawler-session-management 属性
クローラーボット (crawler bot) に特別なセッション処理を設定します。

注記

管理 CLI を使用して **crawler-session-management** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。以下に例を示します。

```
/subsystem=undertow/servlet-container=default/setting=crawler-session-management:add
/subsystem=undertow/servlet-container=default/setting=crawler-session-management:read-resource
```

表A.66 crawler-session-management 属性

属性	デフォルト	説明
session-timeout		クローラーが所有するセッションのセッションタイムアウト (秒単位)。
user-agents		クローラーのユーザーエージェントの一致に使用される正規表現。

jsp 属性

注記

管理 CLI を使用して **jsp** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。以下に例を示します。

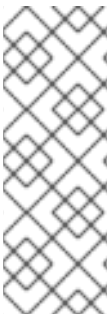
```
/subsystem=undertow/servlet-container=default/setting=jsp:read-resource
```

表A.67 jsp 属性

属性	デフォルト	説明
check-interval	0	バックグラウンドスレッドを使用して Jakarta Server Pages 更新の間隔をチェックします。ファイルシステム通知 API を使用して Jakarta Server Pages 変更通知が処理されるほとんどのデプロイメントでは効果はありません。ファイル監視サービスが無効になっている場合のみ有効です。
development	false	Jakarta Server Pages のリロードをオンザフライで有効にする開発モードを有効にします。
disabled	false	Jakarta Server Pages コンテナーを有効にします。
display-source-fragment	true	ランタイムエラーの発生時に、対応する Jakarta Server Pages ソースの断片の表示を試行します。
dump-smap	false	SMAP データをファイルに書き込みます。
error-on-use-bean-invalid-class-attribute	false	useBean で不適切なクラスを使用するときにエラーを有効にします。
generate-strings-as-char-arrays	false	String 定数を char 配列として生成します。
java-encoding	UTF8	Java ソースに使用するエンコーディングを指定します。
keep-generated	true	生成されたサーブレットを保持します。

属性	デフォルト	説明
mapped-file	true	Jakarta Server Pages ソースにマッピングします。
modification-test-interval	4	更新の 2 つのテスト間の最小時間 (秒単位)。
optimize-scriptlets	false	文字列連結の削除に Jakarta Server Pages スクリプトレットを最適化するかどうか。
recompile-on-fail	false	各リクエストで失敗した Jakarta Server Pages のコンパイルを再試行します。
scratch-dir		別のワークディレクトリーを指定します。
smap	true	SMAP を有効にします。
source-vm	1.8	コンパイルのソース VM レベル。
tag-pooling	true	タブプーリングを有効にします。
target-vm	1.8	コンパイルのターゲット VM レベル。
trim-spaces	false	生成されたサーブレットから一部の領域をトリミングします。
x-powered-by	true	x-powered-by で Jakarta Server Pages エンジンのアドバタイズを有効にします。

persistent-sessions 属性



注記

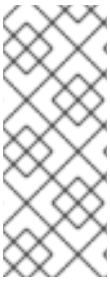
管理 CLI を使用して **persistent-sessions** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。以下に例を示します。

```
/subsystem=undertow/servlet-container=default/setting=persistent-sessions:add
/subsystem=undertow/servlet-container=default/setting=persistent-sessions:read-resource
```

表A.68 persistent-sessions 属性

属性	デフォルト	説明
path		永続セッションデータディレクトリーへのパス。null の場合、セッションがメモリーに保存されます。
relative-to		相対パスの起点となるディレクトリー。

session-cookie 属性



注記

管理 CLI を使用して **session-cookie** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。以下に例を示します。

```
/subsystem=undertow/servlet-container=default/setting=session-cookie:add
/subsystem=undertow/servlet-container=default/setting=session-cookie:read-resource
```

表A.69 session-cookie 属性

属性	デフォルト	説明
comment		クッキーのコメント。
domain		クッキーのドメイン。
http-only		クッキーが http 専用であるかどうか。
max-age		クッキーの最大有効期間。
name		クッキーの名前。
secure		クッキーがセキュアであるかどうか。

websockets 属性

注記

管理 CLI を使用して **websockets** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。以下に例を示します。

```
/subsystem=undertow/servlet-container=default/setting=websockets:read-resource
```

表A.70 websockets 属性

属性	デフォルト	説明
buffer-pool	default	websocket デプロイメントに使用するバッファープール。
deflater-level	0	DEFLATE アルゴリズムの圧縮レベルを設定します。
dispatch-to-worker	true	コールバックがワーカースレッドにディスパッチされるべきかどうか。**false** の場合、IO スレッドで実行され、より高速になりますが、ブロッキング操作を実行しないように注意する必要があります。

属性	デフォルト	説明
per-message-deflate	false	websocket のメッセージごとの圧縮拡張機能を有効にします。
worker	default	websocket デプロイメントに使用するワーカー。

welcome-file 属性
ウェルカムファイルを定義し、オプションはありません。

フィルターの属性
これらのコンポーネントは **/subsystem=undertow/configuration=filter** にあります。

custom-filter フィルター

表A.71 custom-filter 属性

属性	デフォルト	説明
class-name		HttpHandler のクラス名。
module		クラスをロードできるモジュール名。
parameters		フィルターのパラメーター。

error-page フィルター
エラーページ。

表A.72 error-page 属性

属性	デフォルト	説明
code		エラーページコード。
path		エラーページパス。

expression-filter フィルター
Undertow 式言語から解析されたフィルター。

表A.73 expression-filter 属性

属性	デフォルト	説明
expression		フィルターを定義する式。
module		フィルター定義のロードに使用するモジュール。

gzip フィルター

gzip フィルターを定義し、属性はありません。

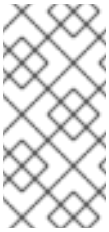
mod-cluster フィルター
mod-cluster フィルターコンポーネントの構造は次のとおりです。

- **mod-cluster**
  - **balancer**
    - **load-balancing-group**
    - **node**
      - **context**

表A.74 mod-cluster 属性

属性	デフォルト	説明
advertise-frequency	10000	ネットワーク上で mod_cluster 自体がアドバタイズする頻度 (ミリ秒単位)。
advertise-path	/	このパス以下に mod-cluster が登録されます。
advertise-protocol	http	使用中のプロトコル。
advertise-socket-binding		アドバタイズに使用されるマルチキャストグループ。
broken-node-timeout	60000	この期間の経過後に破損したノードがテーブルから削除されます。
cached-connections-per-thread	5	無期限に維持される接続の数。
connection-idle-timeout	60	接続がアイドル状態でいられる期間。この期間を経過すると接続が閉じられます。 プールサイズが設定された最小値に達すると (**cached-connections-per-thread** によって設定) 接続はタイムアウトしません。
connections-per-thread	10	IO スレッドごとにバックエンドサーバーに保持される接続の数。
enable-http2	false	ロードバランサーがバックエンド接続の HTTP/2 へのアップグレードを試行すべきかどうか。HTTP/2 がサポートされていない場合は通常どおり HTTP または HTTPS が使用されます。
failover-strategy	LOAD_BALANCED	セッションアフィニティーが実行されるノードが利用できない場合にフェイルオーバーノードの選択方法を判断する属性。

属性	デフォルト	説明
health-check-interval	10000	バックエンドノードへのヘルスチェック ping の頻度。
http2-enable-push	true	HTTP/2 接続に対してプッシュを有効にするべきかどうか。
http2-header-table-size	4096	HPACK 圧縮に使用されるヘッダーテーブルのサイズ (バイト単位)。このメモリー量が圧縮のために接続ごとに割り当てられます。より大きな値はより多くのメモリーを使用しますが、圧縮が向上されます。
http2-initial-window-size	65535	クライアントがサーバーにデータを送信できる速度を制御するフロー制御ウインドウサイズ (バイト単位)。
http2-max-concurrent-streams		単一の接続上でいつでもアクティブな状態になれる HTTP/2 の最大数。
http2-max-frame-size	16384	HTTP/2 の最大フレームサイズ (バイト単位)。
http2-max-header-list-size		サーバーが許可する用意があるリクエストヘッダーの最大サイズ (バイト単位)。
management-access-predicate		mod_cluster 管理コマンドを実行できるかどうかを判断するために受信リクエストに適用される述語。**management-socket-binding** からのリクエストの管理を制限することで、提供されるセキュリティーに追加のセキュリティーを提供します。
management-socket-binding		mod_cluster 管理ポートのソケットバインディング。mod_cluster を使用する場合、リクエストを処理するパブリックの HTTP リスナーと、mod_cluster コマンドを処理するための内部ネットワークにバインドされた HTTP リスナーの 2 つの HTTP リスナーを定義する必要があります。このソケットバインディングは内部リスナーと対応する必要があり、公的にアクセスできない必要があります。
max-ajp-packet-size	8192	AJP パケットの最大サイズ (バイト単位)。この値を大きくすると、AJP はヘッダーの量が多いリクエストおよび応答に対応できます。ロードバランサーとバックエンドサーバーで同じにする必要があります。
max-request-time	-1	バックエンドノードへのリクエストの送信にかかる最大期間。この期間を超えるとリクエストが Kill されます。

属性	デフォルト	説明
max-retries	1	リクエストの失敗時に、リクエストの再試行を実行する回数。  **注記** リクエストがべき等とみなされない場合、バックエントサーバーに送信されなかったことをプロキシーが確信できる場合のみ再試行が行われます。
request-queue-size	10	接続プールが満杯の場合にキューに置けるリクエストの数。 この数を超えるリクエストは拒否され、503 エラーが発生します。
security-key		mod_cluster グループに使用されるセキュリティーキー。すべてのメンバーが同じセキュリティーキーを使用する必要があります。
security-realm		SSL 設定を提供するセキュリティーレルム。**非推奨: ssl-context 属性を使用して設定済みの SSLContext を直接参照してください。**
ssl-context		フィルターによって使用される **SSLContext** への参照。
use-alias	false	エイリアスチェックが実行されるかどうか。
worker	default	アドバタイズ通知の送信に使用される XNIO ワーカー。

表A.75 balancer 属性

属性	デフォルト	説明
max-attempts		リクエストをバックエンドサーバーへ送信する試行回数。
sticky-session		スティッキーセッションが有効であるかどうか。
sticky-session-cookie		セッションクッキー名。
sticky-session-force		**true** の場合、リクエストがスティッキーノードへルーティングできないとエラーが返されます。 その他の場合は別のノードにルーティングされます。

属性	デフォルト	説明
sticky-session-path		スティッキーセッションクッキーのパス。
sticky-session-remove		リクエストを正しいホストへルーティングできない場合、セッションクッキーを削除します。
wait-worker		利用可能なワーカーを待つ秒数。

load-balancing-group 属性

ロードバランシンググループを定義し、オプションはありません。

表A.76 node 属性

属性	デフォルト	説明
aliases		ノードのエイリアス。
cache-connections		無期限に維持される接続の数。
elected		選択 (elected) 数。
flush-packets		受信したデータを即座にフラッシュするかどうか。
load		このノードの現在の負荷。
load-balancing-group		このノードが属するロードバランシンググループ。
max-connections		IO スレッドごとの最大接続数。
open-connections		現在開かれている接続の数。
ping		ノードの ping。
queue-new-requests		リクエストが受信され、即座に使用できるワーカーがない場合にキューに置くかどうか。
read		ノードから読み取るバイト数。
request-queue-size		リクエストキューのサイズ。
status		このノードの現在の状態。

属性	デフォルト	説明
timeout		リクエストのタイムアウト。
ttl		接続の数が **cache-connections** よりも多い場合に、閉じられる前にリクエストがない状態で接続にキープアライブが使用される時間。
uri		ロードバランサーがノードへの接続に使用するURI。
written		ノードに転送されたバイト数。

表A.77 context 属性

属性	デフォルト	説明
requests		このコンテキストに対するリクエストの数。
status		このコンテキストの状態。

request-limit フィルター

表A.78 request-limit 属性

属性	デフォルト	説明
max-concurrent-requests		同時リクエストの最大数。
queue-size		キューに置くリクエスト数。この数を超えるリクエストは拒否されます。

response-header フィルター
response-header フィルターはカスタムヘッダーの追加を可能にします。

表A.79 response-header 属性

属性	デフォルト	説明
header-name		ヘッダー名。
header-value		ヘッダーの値

rewrite フィルター

表A.80 rewrite 属性

属性	デフォルト	説明
redirect	false	再書き込みの代わりにリダイレクトが行われるかどうか。
target		ターゲットを定義する式。定数ターゲットにリダイレクトを行う場合は、値を単一引用符で囲みます。

ハンドラーの属性

これらのコンポーネントは **/subsystem=undertow/configuration=handler** にあります。

file 属性

表A.81 file 属性

属性	デフォルト	説明
cache-buffer-size	1024	バッファーのサイズ。
cache-buffers	1024	バッファーの数。
case-sensitive	true	大文字と小文字を区別してファイルを処理するかどうか。**false** (区別しない) に設定すると、基盤のファイルシステムが大文字と小文字を区別しない場合のみ動作します。
directory-listing	false	ディレクトリーのリストを有効にするかどうか。
follow-symlink	false	シンボリックリンクのフォローを有効にするかどうか。
path		ファイルハンドラーがリソースに対応する場所からのファイルシステム上のパス。
safe-symlink-paths		シンボリックリンクのターゲットとして安全なパス。

静的リソースに **WebDAV** を使用
過去のバージョンの JBoss EAP では、**web** サブシステムで WebDAV を使用して (**WebdavServlet** 経由) 静的リソースをホストし、追加の HTTP メソッドを有効にしてこれらのファイルへのアクセスや操作を実行できました。JBoss EAP 7 では、ファイルハンドラーを経由した静的ファイルの対応メカニズムは **undertow** サブシステムによって提供されますが、**undertow** サブシステムは WebDAV をサポートしません。JBoss EAP 7 で WebDAV を使用する場合は、カスタムの WebDav サーブレットを記述してください。

reverse-proxy 属性
reverse-proxy ハンドラーコンポーネントの構造は以下のとおりです。

- reverse-proxy
  - host

表A.82 reverse-proxy 属性

属性	デフォルト	説明
cached-connections-per-thread	5	無期限に維持される接続の数。
connection-idle-timeout	60	接続がアイドル状態でいられる期間。この期間を経過すると接続が閉じられます。 プールサイズが設定された最小値に達すると (cached-connections-per-thread によって設定) 接続はタイムアウトしません。
connections-per-thread	40	IO スレッドごとにバックエンドサーバーに保持される接続の数。
max-request-time	-1	プロキシーリクエストがアクティブな状態でいられる最大時間。この値を超えるとリクエストは kill されます。 デフォルトは unlimited (無制限) です。
max-retries	1	リクエストの失敗時に、リクエストの再試行を実行する回数。  **注記** リクエストがべき等とみなされない場合、バックエントサーバーに送信されなかったことをプロキシーが確信できる場合のみ再試行が行われます。
problem-server-retry	30	ダウンしたサーバーへの再接続を試みる前に待機する時間 (秒単位)。
request-queue-size	10	接続プールが満杯の場合にキューに置けるリクエストの数。 この数を超えるリクエストは拒否され、503 エラーが発生します。
session-cookie-names	JSESSIONID	セッションクッキー名のコンマ区切りリスト。通常は JSESSIONID。

表A.83 host 属性

属性	デフォルト	説明
enable-http2	false	**true** の場合、プロキシーは HTTP/2 を使用してバックエンドへの接続を試行します。サポートされていない場合は、HTTP/1.1 にフォールバックします。
instance-id		スティッキーセッションを有効にするために使用されるインスタンス ID または JVM ルート。

属性	デフォルト	説明
outbound-socket-binding		このホストのアウトバウンドソケットバインディング。
path	/	ホストがルート以外のリソースを使用する場合のオプションのパス。
scheme	http	使用されるスキームの種類。
security-realm		ホストへの接続の SSL 設定を提供するセキュリティーレルム。
ssl-context		このハンドラーによって使用される SSLContext への参照。

サーバーの属性

server コンポーネントの構造は次のとおりです。

- server
  - ajp-listener
  - host
    - filter-ref
    - location
      - filter-ref
    - setting
      - access-log
      - console-access-log
      - http-invoker
      - single-sign-on
  - http-listener
  - https-listener

server 属性

表A.84 server 属性

属性	デフォルト	説明
default-host	default-host	サーバーのデフォルトの仮想ホスト。

属性	デフォルト	説明
servlet-container	default	サーバーのデフォルトのサーブレットコンテナー。

ajp-listener 属性

表A.85 ajp-listener 属性

属性	デフォルト	説明
allow-encoded-slash	false	リクエストにエンコードされた文字 (例: **%2F**) が含まれる場合にデコードするかどうか。
allow-equals-in-cookie-value	false	引用符で囲まれていないクッキー値のエスケープされていない等号記号を許可するかどうか。引用符で囲まれていないクッキー値に等号記号が含まれないことがあります。等号記号が含まれると、等号の前で値が終了します。残りのクッキー値は破棄されます。
allow-unescaped-characters-in-url	false	URL でエスケープされていない文字を許可するかどうか。**true** に設定されていると、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を処理します。**false** に設定されている場合、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を **HTTP Bad Request 400** 応答コードで拒否します。
always-set-keep-alive	true	仕様が厳密に必要としない場合でも Connection: keep-alive ヘッダーが応答に追加されるかどうか。
buffer-pipelined-data	false	パイプライン化されたリクエストをバッファーするかどうか。
buffer-pool	default	AJP リスナーのバッファープール。
decode-url	true	**true** の場合、選択された文字エンコーディング (デフォルトでは UFT-8) を使用してパーサーが URL およびクエリーパラメーターをデコードします。false の場合はデコードされません。これにより、ハンドラーによる希望の文字セットへのデコードが可能になります。
disallowed-methods	["TRACE"]	許可されない HTTP メソッドのコンマ区切りリスト。
enabled	true	リスナーが有効であるかどうか。**非推奨: 属性を有効にすると、設定の一貫性の保持に問題が生じます。**

属性	デフォルト	説明
max-ajp-packet-size	8192	AJP パケットがサポートされる最大サイズ。変更する場合は、ロードバランサーとバックエンドサーバーで増やす必要があります。
max-buffered-request-size	16384	バッファー済みのリクエストの最大サイズ (バイト単位)。 リクエストは通常バッファーされませんが、バッファーされる最も一般的なケースが POST リクエストの SSL 再ネゴシエーションを実行する場合です。 再ネゴシエーションを実行するには、POST データを完全にバッファーする必要があります。
max-connections		同時接続の最大数。サーバー設定で値が設定されないと、同時接続の数は **Integer.MAX_VALUE** に制限されます。
max-cookies	200	解析されるクッキーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-header-size	1048576	HTTP リクエストヘッダーの最大サイズ (バイト単位)。
max-headers	200	解析されるヘッダーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-parameters	1000	解析されるパラメーターの最大数。ハッシュの脆弱性に対して保護するために使用されます。クエリーパラメーターと POST データの両方に適用されますが累積されません。たとえば、max-parameters の 2 倍をパラメーターの合計数とすることができます。
max-post-size	10485760	許可される最大 POST サイズ。
no-request-timeout	60000	接続がアイドル状態でいられる期間 (ミリ秒単位)。この期間を超えると接続がコンテナーによって閉じられます。
read-timeout		ソケットの読み取りタイムアウトを設定します (ミリ秒単位)。読み取りに成功しないまま指定の時間が経過すると、ソケットの次の読み取りによって **ReadTimeoutException** が発生します。
receive-buffer		受信バッファーサイズ。

属性	デフォルト	説明
record-request-start-time	false	リクエストの開始時間を記録し、リクエスト時間がログに記録されるようにするかどうか。パフォーマンスへの影響は小さいながら、ある程度の影響を与えます。
redirect-socket		このリスナーが SSL でないリクエストをサポートし、リクエストが一致する必要がある SSL トランスポートに対して受信された場合、リクエストをここに指定されたソケットバインディングポートに自動的にリダイレクトするかどうか。
request-parse-timeout		リクエストの解析に費やすことができる最大時間 (ミリ秒単位)。
resolve-peer-address	false	ホストの DNS ルックアップを有効にします。
scheme		リスナースキーム (HTTP または HTTPS)。デフォルトでは、スキーマは受信 AJP リクエストから取得されます。
secure	false	**true** の場合、リクエストが HTTPS を使用しなくてもこのリスナーから送信されるリクエストはセキュアであるとマーク付けされます。
send-buffer		送信バッファーサイズ。
socket-binding		AJP リスナーのソケットバインディング。
tcp-backlog		指定のバックログでサーバーを設定します。
tcp-keep-alive		実装に依存して TCP キープアライブメッセージを送信するようチャネルを設定します。
url-charset	UTF-8	URL の文字セット。
worker	default	リスナーの XNIO ワーカー。
write-timeout		ソケットの書き込みタイムアウトを設定します (ミリ秒単位)。書き込みに成功しないまま指定の時間が経過すると、ソケットの次の書き込みによって **WriteTimeoutException** が発生します。

host 属性

表A.86 host 属性

属性	デフォルト	説明
alias		ホストのエイリアスのコンマ区切りリスト。
default-response-code	404	設定した場合、サーバー上に要求されたコンテキストが存在しない場合に設定した応答コードが返信されます。
default-web-module	ROOT.war	デフォルトの Web モジュール。
disable-console-redirect	false	**true** に設定すると、**/console** リダイレクトはこのホストに対して有効になりません。
queue-requests-on-start	true	**true** に設定すると、リクエストはこのホストに対して開始時にキューに置かれます。**false** に設定された場合、代わりにデフォルトの応答コードが返されます。

filter-ref 属性

表A.87 filter-ref 属性

属性	デフォルト	説明
predicate		predicate は交換を基に true または false の決定を行う簡単な方法です。多くのハンドラーには条件によって適用される要件があり、predicate は条件を指定する一般的な方法を提供します。
priority	1	フィルターの順序を定義します。小さい数字は、同じコンテキストの他の大きな数字よりも先にサーバーがハンドラーチェーンに含まれるよう指示します。値の範囲は、フィルターが最初に処理されることを示す **1** からフィルターが最後に処理されることを示す **2147483647** までになります。

location 属性

表A.88 location 属性

属性	デフォルト	説明
handler		この場所のデフォルトのハンドラー。

filter-ref 属性

表A.89 filter-ref 属性

属性	デフォルト	説明
predicate		predicate は交換を基に true または false の決定を行う簡単な方法です。多くのハンドラーには条件によって適用される要件があり、predicate は条件を指定する一般的な方法を提供します。
priority	1	フィルターの順序を定義します。1以上を設定する必要があります。同じコンテキスト下で数字が大きいほどサーバーのハンドラーチェインでの順序が早くなるよう指示します。

access-log 属性

注記

管理 CLI を使用して **access-log** 要素を管理する場合、**host** 要素の **settings** 下で使用できます。以下に例を示します。

```
/subsystem=undertow/server=default-server/host=default-host/setting=access-log:add
/subsystem=undertow/server=default-server/host=default-host/setting=access-log:read-resource
```

表A.90 access-log 属性

属性	デフォルト	説明
directory	${jboss.server.log.dir}	ログを保存するディレクトリー。
extended	false	ログが拡張されたログファイル形式を使用するかどうか。

属性	デフォルト	説明
pattern	common	アクセスログパターン。この属性に利用できるオプションの詳細は、**Development Guide**の Provided Undertow Handlers を参照してください。  **注記** リクエストの処理にかかった時間を出力するよう **pattern** を設定した場合、該当するリスナーの **record-request-start-time** 属性も有効にする必要があります。そうしないと、アクセスログに時間が適切に記録されません。例を以下に示します。以下に例を示します。 `/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=record-request-start-time,value=true)`
predicate		リクエストをログに記録するかどうかを判断する述語。
prefix	access_log	ログファイル名の接頭辞。
relative-to		相対パスの起点となるディレクトリー。
rotate	true	アクセスログを毎日ローテーションするかどうか。
suffix	log	ログファイル名の接尾辞。
use-server-log	false	ログが個別のファイルではなくサーバーログに書き込まれるかどうか。

属性	デフォルト	説明
worker	default	ロギングに使用するワーカーの名前。

console-access-log 属性

表A.91 console-access-log 属性

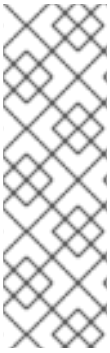
属性	デフォルト	説明
attributes	{remote-host={},remote-user={},date-time={},request-line={},response-code={},bytes-sent={}}	コンソールアクセスログ出力に含めるログデータおよびデフォルトデータ設定のカスタマイズを指定します。
include-host-name	false	JSON 構造の出力にホスト名を含めるかどうかを指定します。**true** に設定した場合、構造化データのキーは "hostName" で、その値は console-access-log が設定されたホストの名前になります。
metadata		コンソールアクセスログの出力に含めるカスタムメタデータを指定します。
predicate		リクエストをログに記録するかどうかを判断する述語。
worker	default	ロギングに使用するワーカーの名前。

http-invoker Attributes

表A.92 http-invoker Attributes

属性	デフォルト	説明
http-authentication-factory		認証に使用する HTTP 認証ファクトリー。
path	wildfly-services	サービスがインストールされるパス。
security-realm		認証に使用するレガシーのセキュリティーレルム。

single-sign-on 属性



注記

管理 CLI を使用して **single-sign-on** 要素を管理する場合は、**host** 要素の **settings** 下で使用できます。以下に例を示します。

```
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:add
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:read-resource
```

重要

配布されるシングルサインオンは、アプリケーションの観点からではこれまでのバージョンの JBoss EAP と変わりありませんが、JBoss EAP 7 では認証情報のキャッシュおよび配布の処理が異なります。JBoss EAP 7 で **ha** プロファイルを実行する場合、デフォルトではホストは関連するセッションおよび SSO クッキー情報が保存される独自の Infinispan キャッシュを持ちます。このキャッシュは Web キャッシュコンテナーのデフォルトキャッシュがベースになります。また、JBoss EAP はホストすべての個別のキャッシュ間で情報の伝搬を処理します。

表A.93 single-sign-on 属性

属性	デフォルト	説明
cookie-name	JSESSIONIDSSO	クッキーの名前。
domain		使用されるクッキードメイン。
http-only	false	クッキーの httpOnly 属性を設定します。
path	/	クッキーのパス。
secure	false	クッキーの secure 属性を設定します。

http-listener 属性

表A.94 http-listener 属性

属性	デフォルト	説明
allow-encoded-slash	false	リクエストにエンコードされた文字 (例: **%2F**) が含まれる場合にデコードするかどうか。
allow-equals-in-cookie-value	false	引用符で囲まれていないクッキー値のエスケープされていない等号記号を許可するかどうか。引用符で囲まれていないクッキー値に等号記号が含まれないことがあります。等号記号が含まれると、等号の前で値が終了します。残りのクッキー値は破棄されます。

属性	デフォルト	説明
allow-unescaped-characters-in-url	false	URL でエスケープされていない文字を許可するかどうか。**true** に設定されていると、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を処理します。**false** に設定されている場合、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を **HTTP Bad Request 400** 応答コードで拒否します。
always-set-keep-alive	true	仕様が厳密に必要としない場合でも Connection: keep-alive ヘッダーが応答に追加されるかどうか。
buffer-pipelined-data	false	パイプライン化されたリクエストをバッファーするかどうか。
buffer-pool	default	リスナーのバッファープール。
certificate-forwarding	false	証明書の転送を有効にするかどうか。有効な場合、リスナーは **SSL_CLIENT_CERT** 属性から証明書を取得します。これらのヘッダーを常に設定するようプロキシーが設定されている場合のみ有効にする必要があります。
decode-url	true	選択された文字エンコーディング (デフォルトでは UFT-8) を使用してパーサーが URL およびクエリーパラメーターをデコードするかどうか。false の場合はデコードされません。これにより、ハンドラーによる希望の文字セットへのデコードが可能になります。
disallowed-methods	["TRACE"]	許可されない HTTP メソッドのコンマ区切りリスト。
enable-http2	false	このリスナーの HTTP/2 サポートを有効にするかどうか。
enabled	true	リスナーが有効であるかどうか。**非推奨: 属性を有効にすると、設定の一貫性の保持に問題が生じます。**
http2-enable-push	true	この接続に対してサーバープッシュが有効であるかどうか。
http2-header-table-size	4096	HPACK 圧縮で使用されるヘッダーテーブルのサイズ (バイト単位)。このメモリー量が圧縮のために接続ごとに割り当てられます。より大きな値はより多くのメモリーを使用しますが、圧縮が向上されます。

属性	デフォルト	説明
http2-initial-window-size	65535	クライアントがサーバーにデータを送信できる速度を制御するフロー制御ウインドウサイズ (バイト単位)。
http2-max-concurrent-streams		単一の接続上でいつでもアクティブな状態になれる HTTP/2 の最大数。
http2-max-frame-size	16384	HTTP/2 の最大フレームサイズ (バイト単位)。
http2-max-header-list-size		サーバーが許可する用意があるリクエストヘッダーの最大サイズ。
max-buffered-request-size	16384	バッファー済みのリクエストの最大サイズ (バイト単位)。 リクエストは通常バッファーされませんが、バッファーされる最も一般的なケースが POST リクエストの SSL 再ネゴシエーションを実行する場合です。 再ネゴシエーションを実行するには、POST データを完全にバッファーする必要があります。
max-connections		同時接続の最大数。サーバー設定で値が設定されないと、同時接続の数は **Integer.MAX_VALUE** に制限されます。
max-cookies	200	解析されるクッキーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-header-size	1048576	HTTP リクエストヘッダーの最大サイズ (バイト単位)。
max-headers	200	解析されるヘッダーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-parameters	1000	解析されるパラメーターの最大数。ハッシュの脆弱性に対して保護するために使用されます。クエリーパラメーターと POST データの両方に適用されますが累積されません。たとえば、max-parameters の 2 倍をパラメーターの合計数とすることができます。
max-post-size	10485760	許可される最大 POST サイズ。
no-request-timeout	60000	接続がアイドル状態でいられる期間 (ミリ秒単位)。この期間を超えると接続がコンテナーによって閉じられます。
proxy-address-forwarding	false	x-forwarded-host および同様のヘッダーを有効にし、リモート IP アドレスおよびホスト名を設定するかどうか。

属性	デフォルト	説明
proxy-protocol	false	PROXY プロトコルを使用して接続情報を送信するかどうか。**true** に設定された場合、リスナーは The PROXY protocol Versions 1 & 2 仕様によって定義される PROXY プロトコルバージョン 1 を使用します。このオプションは、同じプロトコルをサポートするロードバランサーの背後になるリスナーにのみ有効にする必要があります。
read-timeout		ソケットの読み取りタイムアウトを設定します (ミリ秒単位)。読み取りに成功しないまま指定の時間が経過すると、ソケットの次の読み取りによって **ReadTimeoutException** が発生します。
receive-buffer		受信バッファーサイズ。
record-request-start-time	false	リクエストの開始時間を記録し、リクエスト時間がログに記録されるようにするかどうか。パフォーマンスへの影響は小さいながら、ある程度の影響を与えます。
redirect-socket		このリスナーが SSL でないリクエストをサポートし、リクエストが一致する必要がある SSL トランスポートに対して受信された場合、リクエストをここに指定されたソケットバインディングポートに自動的にリダイレクトするかどうか。
request-parse-timeout		リクエストの解析に費やすことができる最大時間 (ミリ秒単位)。
require-host-http11	false	すべての HTTP/1.1 リクエストに **Host** ヘッダーが必要になります。リクエストにこのヘッダーが含まれないと、403 エラーにより拒否されます。
resolve-peer-address	false	ホストの DNS ルックアップを有効にします。
secure	false	**true** の場合、リクエストが HTTPS を使用しなくてもこのリスナーから送信されるリクエストはセキュアであるとマーク付けされます。
send-buffer		送信バッファーサイズ。
socket-binding		リスナーのソケットバインディング。
tcp-backlog		指定のバックログでサーバーを設定します。
tcp-keep-alive		実装に依存して TCP キープアライブメッセージを送信するようチャネルを設定します。

属性	デフォルト	説明
url-charset	UTF-8	URL の文字セット。
worker	default	リスナーの XNIO ワーカー。
write-timeout		ソケットの書き込みタイムアウトを設定します (ミリ秒単位)。書き込みに成功しないまま指定の時間が経過すると、ソケットの次の書き込みによって **WriteTimeoutException** が発生します。

https-listener 属性

表A.95 https-listener 属性

属性	デフォルト	説明
allow-encoded-slash	false	リクエストにエンコードされた文字 (例: **%2F**) が含まれる場合にデコードするかどうか。
allow-equals-in-cookie-value	false	引用符で囲まれていないクッキー値のエスケープされていない等号記号を許可するかどうか。引用符で囲まれていないクッキー値に等号記号が含まれないことがあります。等号記号が含まれると、等号の前で値が終了します。残りのクッキー値は破棄されます。
allow-unescaped-characters-in-url	false	URL でエスケープされていない文字を許可するかどうか。**true** に設定されていると、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を処理します。**false** に設定されている場合、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を **HTTP Bad Request 400** 応答コードで拒否します。
always-set-keep-alive	true	仕様が厳密に必要としない場合でも Connection: keep-alive ヘッダーが応答に追加されるかどうか。
buffer-pipelined-data	false	パイプライン化されたリクエストをバッファーするかどうか。
buffer-pool	default	リスナーのバッファープール。
certificate-forwarding	false	証明書の転送を有効にすべきかどうか。有効な場合、リスナーは **SSL_CLIENT_CERT** 属性から証明書を取得します。これらのヘッダーを常に設定するようプロキシーが設定されている場合のみ有効にする必要があります。

属性	デフォルト	説明
decode-url	true	選択された文字エンコーディング (デフォルトでは UFT-8) を使用してパーサーが URL およびクエリーパラメーターをデコードするかどうか。false の場合はデコードされません。これにより、ハンドラーによる希望の文字セットへのデコードが可能になります。
disallowed-methods	["TRACE"]	許可されない HTTP メソッドのコンマ区切りリスト。
enable-http2	false	このリスナーの HTTP/2 サポートを有効にします。
enable-spdy	false	このリスナーの SPDY サポートを有効にします。**非推奨: SPDY は HTTP/2 に置き換えられました。**
enabled	true	リスナーが有効であるかどうか。**非推奨: 属性を有効にすると、設定の一貫性の保持に問題が生じます。**
enabled-cipher-suites		有効な SSL 暗号を設定します。**非推奨: SSLContext が参照される場合は、暗号スイートで設定してサポートされるようにする必要があります。**
enabled-protocols		SSL プロトコルを設定します。**非推奨: SSLContext が参照される場合は、暗号スイートで設定してサポートされるようにする必要があります。**
http2-enable-push	true	この接続に対してサーバープッシュが有効であるかどうか。
http2-header-table-size	4096	HPACK 圧縮で使用されるヘッダーテーブルのサイズ (バイト単位)。このメモリー量が圧縮のために接続ごとに割り当てられます。より大きな値はより多くのメモリーを使用しますが、圧縮が向上されます。
http2-initial-window-size	65535	クライアントがサーバーにデータを送信できる速度を制御するフロー制御ウインドウサイズ (バイト単位)。
http2-max-concurrent-streams		単一の接続上でいつでもアクティブな状態になれる HTTP/2 の最大数。
http2-max-frame-size	16384	HTTP/2 の最大フレームサイズ (バイト単位)。
http2-max-header-list-size		サーバーが許可する用意があるリクエストヘッダーの最大サイズ。

属性	デフォルト	説明
max-buffered-request-size	16384	バッファー済みのリクエストの最大サイズ (バイト単位)。 リクエストは通常バッファーされませんが、バッファーされる最も一般的なケースが POST リクエストの SSL 再ネゴシエーションを実行する場合です。 再ネゴシエーションを実行するには、POST データを完全にバッファーする必要があります。
max-connections		同時接続の最大数。サーバー設定で値が設定されないと、同時接続の数は **Integer.MAX_VALUE** に制限されます。
max-cookies	100	解析されるクッキーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-header-size	1048576	HTTP リクエストヘッダーの最大サイズ (バイト単位)。
max-headers	200	解析されるヘッダーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-parameters	1000	解析されるパラメーターの最大数。ハッシュの脆弱性に対して保護するために使用されます。クエリーパラメーターと POST データの両方に適用されますが累積されません。たとえば、max-parameters の 2 倍をパラメーターの合計数とすることができます。
max-post-size	10485760	許可される最大 POST サイズ。
no-request-timeout	60000	接続がアイドル状態でいられる期間 (ミリ秒単位)。この期間を超えると接続がコンテナーによって閉じられます。
proxy-address-forwarding	false	x-forwarded-host ヘッダー (およびその他の x-forwarded-* ヘッダー) の処理を有効にし、このヘッダー情報を使用してリモートアドレスを設定します。これらのヘッダーを設定する信頼されたプロキシーの背後でのみ使用する必要があります。 そうでないと、リモートユーザーによる IP アドレスの偽装が可能になります。
proxy-protocol	false	PROXY プロトコルを使用して接続情報を送信するかどうか。**true** に設定された場合、リスナーは The PROXY protocol Versions 1 & 2 仕様によって定義される PROXY プロトコルバージョン 1 を使用します。このオプションは、同じプロトコルをサポートするロードバランサーの背後になるリスナーにのみ有効にする必要があります。

属性	デフォルト	説明
read-timeout		ソケットの読み取りタイムアウトを設定します (ミリ秒単位)。読み取りに成功しないまま指定の時間が経過すると、ソケットの次の読み取りによって **ReadTimeoutException** が発生します。
receive-buffer		受信バッファーサイズ。
record-request-start-time	false	リクエストの開始時間を記録し、リクエスト時間がログに記録されるようにするかどうか。パフォーマンスへの影響は小さいながら、ある程度の影響を与えます。
request-parse-timeout		リクエストの解析に費やすことができる最大時間 (ミリ秒単位)。
require-host-http11	false	すべての HTTP/1.1 リクエストに Host ヘッダーが必要になります。リクエストにこのヘッダーが含まれないと、403 エラーにより拒否されます。
resolve-peer-address	false	ホストの DNS ルックアップを有効にします。
secure	false	**true** の場合、リクエストが HTTPS を使用しなくてもこのリスナーから送信されるリクエストはセキュアであると見なされます。
security-realm		リスナーのセキュリティーレルム。**非推奨: ssl-context 属性を使用して設定済みの SSLContext を直接参照してください。**
send-buffer		送信バッファーサイズ。
socket-binding		リスナーのソケットバインディング。
ssl-context		このリスナーによって使用される SSLContext への参照。
ssl-session-cache-size		アクティブな SSL セッションの最大数。**非推奨: Elytron セキュリティーコンテキスト上で設定できるようになりました。**
ssl-session-timeout		SSL セッションのタイムアウト (秒単位)。**非推奨: Elytron セキュリティーコンテキスト上で設定できるようになりました。**
tcp-backlog		指定のバックログでサーバーを設定します。

属性	デフォルト	説明
tcp-keep-alive		実装に依存して TCP キープアライブメッセージを送信するようチャネルを設定します。
url-charset	UTF-8	URL の文字セット。
verify-client	NOT_REQUESTED	SSL チャネルの希望の SSL クライアント認証モード。**非推奨: SSLContext が参照される場合は、クライアント検証の必要なモードに対して直接設定する必要があります。**
worker	default	リスナーの XNIO ワーカー。
write-timeout		ソケットの書き込みタイムアウトを設定します (ミリ秒単位)。書き込みに成功しないまま指定の時間が経過すると、ソケットの次の書き込みによって **WriteTimeoutException** が発生します。

A.28. UNDERTOW サブシステムの統計

表A.96 **ajp-listener** の統計

名前	説明
bytes-received	このリスナーによって受信されたバイト数。
bytes-sent	このリスナーによって送信されたバイト数。
error-count	このリスナーによって送信された 500 応答コードの数。
max-processing-time	このリスナーのリクエストによる最大処理時間。
processing-time	このリスナーによって処理されるすべてのリクエストの合計処理時間。
request-count	このリスナーが対応したリクエストの数。

表A.97 **http-listener** の統計

名前	説明
bytes-received	このリスナーによって受信されたバイト数。
bytes-sent	このリスナーによって送信されたバイト数。
error-count	このリスナーによって送信された 500 応答コードの数。

名前	説明
max-processing-time	このリスナーのリクエストによる最大処理時間。
processing-time	このリスナーによって処理されるすべてのリクエストの合計処理時間。
request-count	このリスナーが対応したリクエストの数。

表A.98 **https-listener** の統計

名前	説明
bytes-received	このリスナーによって受信されたバイト数。
bytes-sent	このリスナーによって送信されたバイト数。
error-count	このリスナーによって送信された 500 応答コードの数。
max-processing-time	このリスナーのリクエストによる最大処理時間。
processing-time	このリスナーによって処理されるすべてのリクエストの合計処理時間。
request-count	このリスナーが対応したリクエストの数。

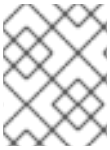
A.29. HTTP メソッドのデフォルトの動作

JBoss EAP 7.4 の **undertow** サブシステムは、これまでの JBoss EAP リリースの **web** サブシステムとは HTTP メソッドのデフォルト動作が異なります。以下の表は、JBoss EAP 7.4 のデフォルト動作を表しています。

表A.99 **HTTP** メソッドのデフォルト動作

HTTP メソッド	Jakarta Server Pages	静的 HTML	ファイルハンドラーによる静的 HTML
GET	OK	OK	OK
POST	OK	NOT_ALLOWED	OK
HEAD	OK	OK	OK
PUT	NOT_ALLOWED	NOT_ALLOWED	NOT_ALLOWED
TRACE	NOT_ALLOWED	NOT_ALLOWED	NOT_ALLOWED
DELETE	NOT_ALLOWED	NOT_ALLOWED	NOT_ALLOWED

HTTP メソッド	Jakarta Server Pages	静的 HTML	ファイルハンドラーによる静的 HTML
OPTIONS	NOT_ALLOWED	OK	NOT_ALLOWED



注記

サーブレットでは、実装によってデフォルトの動作が異なりますが、デフォルトの動作が **NOT_ALLOWED** である **TRACE** メソッドは例外です。

A.30. REMOTING サブシステムの属性

注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-remoting_4_0.xsd** のスキーマ定義ファイルで確認してください。

表A.100 remoting 属性

属性	デフォルト	説明
worker-read-threads	1	リモーティングワーカーに対して作成する読み取りスレッドの数。
worker-task-core-threads	4	リモーティングワーカーのタスクスレッドプールに対するコアスレッドの数。
worker-task-keepalive	60	コアでないリモーティングワーカーのタスクスレッドにキープアライブを使用する期間 (ミリ秒単位)。
worker-task-limit	16384	許可するリモーティングワーカータスクの最大数。この数を超えるリモーティングワーカータスクは拒否されます。
worker-task-max-threads	16	リモーティングワーカーのタスクスレッドプールに対するスレッドの最大数。
worker-write-threads	1	リモーティングワーカーに作成する書き込みスレッドの数。

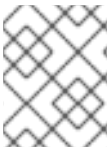
重要

remoting 要素の上記の属性は非推奨になりました。これらの属性は **io** サブシステムを使用して設定する必要があります。

表A.101 endpoint 属性

属性	デフォルト	説明
auth-realm		認証の CallbackHandler が指定されていない場合に使用する認証レルム。
authentication-retries	3	接続を閉じる前にクライアントが認証を再試行できる回数を指定します。
authorize-id		SASL 認証 ID。指定されている認証 CallbackHandler がなく、選択した SASL メカニズムがユーザー名を要求する場合に認証ユーザー名として使用されます。
buffer-region-size		割り当てられたバッファーリージョンのサイズ。
heartbeat-interval	2147483647	接続ハートビートに使用する間隔 (ミリ秒単位)。この時間アウトバウンド方向の接続がアイドル状態であると、ping メッセージが送信され、対応する応答メッセージがトリガーされます。
max-inbound-channels	40	チャネルの同時インバウンドメッセージの最大数。
max-inbound-message-size	9223372036854775807	許可されるインバウンドメッセージの最大サイズ。メッセージがこのサイズを超えると、読み取り側および書き込み側に例外が発生します。
max-inbound-messages	80	接続に対してサポートされるインバウンドチャネルの最大数。
max-outbound-channels	40	チャネルの同時アウトバウンドメッセージの最大数。
max-outbound-message-size	9223372036854775807	送信するアウトバウンドメッセージの最大サイズ。これよりも大きいサイズのメッセージは送信されません。 送信しようとすると、書き込み側に例外が発生します。
max-outbound-messages	65535	接続に対してサポートされるアウトバウンドチャネルの最大数。
receive-buffer-size	8192	接続上でこのエンドポイントが許可する最大バッファーサイズ。
receive-window-size	131072	接続チャネルの受信方向の最大ウインドウサイズ (バイト単位)。
sasl-protocol	**remote**	**SaslServer** または **SaslClient** が作成された場合、デフォルトで指定されるプロトコルは **remote** になります。この属性を使用すると、このプロトコルをオーバーライドできます。

属性	デフォルト	説明
send-buffer-size	8192	接続上でこのエンドポイントが送信する最大バッファーサイズ。
server-name		最初の案内応答で接続のサーバー側はその名前をクライアントに渡します。デフォルトでは、名前は接続のローカルアドレスから自動的に検索されますが、これを使用して名前を上書きできます。
transmit-window-size	131072	接続チャネルの送信方向の最大ウインドウサイズ (バイト単位)。
worker	**default**	使用するワーカー



注記

endpoint 要素の更新に管理 CLI を使用する場合は、**remoting** 要素の **configuration** 下で利用できます。例: **/subsystem=remoting/configuration=endpoint/**

コネクターの属性

connector コンポーネントの構造は次のとおりです。

- connector
  - property
  - security
    - sasl
      - property
      - sasl-policy
        - policy

表A.102 connector 属性

属性	デフォルト	説明
authentication-provider		**authentication-provider** 要素には、受信接続に使用する認証プロバイダーの名前が含まれます。
sasl-authentication-factory		このコネクターをセキュアにする SASL 認証ファクトリーへの参照。
sasl-protocol	**remote**	認証に使用する SASL メカニズムに渡すプロトコル。

属性	デフォルト	説明
security-realm		このコネクターの認証に使用する、関連するセキュリティーレルム。
server-name		最初のメッセージ交換で送信され、SASL ベースの認証用のサーバー名。
socket-binding		アタッチするソケットバインディングの名前。
ssl-context		このコネクターに使用する SSL コンテキストへの参照。

表A.103 property 属性

属性	デフォルト	説明
value		プロパティーの値。

セキュリティーの属性

security コンポーネントはコネクターのセキュリティーを設定できるようにしますが、直接の設定属性は含まれていません。これは、sasl などのネストされたコンポーネントを使用して設定できます。

表A.104 sasl 属性

属性	デフォルト	説明
include-mechanisms		オプションのネストされた **include-mechanisms** 要素には、許可される SASL メカニズム名のホワイトリストが含まれます。このリストにないメカニズムは許可されません。
qop		オプションのネストされた **qop** 要素には、QOP (保護品質) の値が希望順に並ぶコンマ区切りリストが含まれます。 このリストの QOP の値は次のとおりです。 ● **auth**: 認証のみ ● **auth-int**: 認証と整合性保護 ● **auth-conf**: 認証、整合性保護および機密性保護

属性	デフォルト	説明
reuse-session	false	オプションのネストされた **reuse-session** ブール値要素は、以前認証したセッション情報をサーバーが再使用するかどうかを指定します。メカニズムによってはこのような再使用をサポートしない可能性があり、その他の要因によって再使用できないこともあります。
server-auth	false	オプションのネストされた **server-auth** ブール値要素は、サーバーがクライアントに対して認証されるかどうかを指定します。メカニズムによってはこの設定をサポートしない可能性があります。
strength		オプションのネストされた **strength** 要素には希望順に並ぶ暗号強度の値のコンマ区切りリストが含まれます。 このリストの暗号強度の値は次のとおりです。 • **high** • **medium** • **low**

sasl-policy 属性

sasl-policy コンポーネントでは、利用できるメカニズムのセットを限定するために使用する任意のポリシーを指定できますが、直接の設定属性は含まれていません。policy などのネストされたコンポーネントを使用して設定できます。

表A.105 policy 属性

属性	デフォルト	説明
forward-secrecy	true	オプションのネストされた **forward-secrecy** 要素には、セッション間で Forward Secrecy を実装するメカニズムを指定するブール値が含まれます。Forward Secrecy とは、あるセッションが侵入されてもその後のセッションに侵入するための情報が自動的に提供されないことを意味します。
no-active	true	オプションのネストされた **no-active** 要素には、能動的攻撃 (辞書攻撃でない) を受けやすいメカニズムを許可するかどうかを指定するブール値が含まれます。許可する場合は **false**、拒否する場合は **true** を設定します。

属性	デフォルト	説明
no-anonymous	true	オプションのネストされた **no-anonymous** 要素には、匿名のログインを許可するメカニズムを許可するかどうかを指定するブール値が含まれます。許可する場合は **false**、拒否する場合は **true** を設定します。
no-dictionary	true	オプションのネストされた **no-dictionary** 要素には、受動的な辞書攻撃を受けやすいメカニズムを許可するかどうかを指定するブール値が含まれます。許可する場合は **false**、拒否する場合は **true** を設定します。
no-plain-text	true	オプションのネストされた **no-plain-text** 要素には、平文の受動的攻撃 (例: **PLAIN**) を受けやすいメカニズムを拒否するかどうかを指定するブール値が含まれます。許可する場合は **false**、拒否する場合は **true** を設定します。
pass-credentials	true	オプションネストされた **pass-credentials** 要素には、クライアントの認証情報を渡すメカニズムが必要であるかどうかを指定するブール値が含まれます。

HTTP コネクターの属性

http-connector コンポーネントの構造は次のとおりです。

- http-connector
  - property (connector と同じ)
  - security (connector と同じ)
    - sasl (connector と同じ)
      - property (connector と同じ)
      - sasl-policy (connector と同じ)
        - policy (connector と同じ)

表A.106 http-connector 属性

属性	デフォルト	説明
authentication-provider		**authentication-provider** 要素には、受信接続に使用する認証プロバイダーの名前が含まれます。
connector-ref		接続する **undertow** サブシステムのコネクターの名前。

属性	デフォルト	説明
sasl-authentication-factory		このコネクターをセキュアにする SASL 認証ファクトリーへの参照。
sasl-protocol	**remote**	認証に使用する SASL メカニズムに渡すプロトコル。
security-realm		このコネクターの認証に使用する、関連するセキュリティーレルム。
server-name		最初のメッセージ交換で送信され、SASL ベースの認証用のサーバー名。

アウトバウンド接続の属性

outbound-connection コンポーネントの構造は次のとおりです。

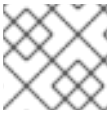
- outbound-connection
  - property

表A.107 outbound-connection 属性

属性	デフォルト	説明
uri		アウトバウンド接続の接続 URI。

表A.108 property 属性

属性	デフォルト	説明
value		プロパティーの値。



注記

上記の **property** 属性は、接続の作成中に使用される XNIO 操作に関連します。

リモートアウトバウンド接続

remote-outbound-connection コンポーネントの構造は次のとおりです。

- remote-outbound-connection
  - property (outbound-connection と同じ)

表A.109 remote-outbound-connection 属性

属性	デフォルト	説明

属性	デフォルト	説明
authentication-context		アウトバウンド接続の設定が含まれる認証コンテキストインスタンスへの参照。
outbound-socket-binding-ref		接続の宛先アドレスとポートの判断に使用される **outbound-socket-binding** の名前。
protocol	**http-remoting**	リモート接続に使用するプロトコル。デフォルトは **http-remoting** です。**非推奨: アウトバウンドセキュリティー設定は authentication-context 定義に移行する必要があります。**
security-realm		パスワードと SSL 設定の取得に使用するセキュリティーレルムへの参照。**非推奨: アウトバウンドセキュリティー設定は authentication-context 定義に移行する必要があります。**
username		リモートサーバーに対して認証する際に使用するユーザー名。**非推奨: アウトバウンドセキュリティー設定は authentication-context 定義に移行する必要があります。**

ローカルアウトバウンド接続の属性

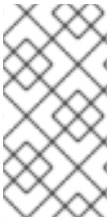
local-outbound-connection コンポーネントの構造は次のとおりです。

- local-outbound-connection
  - property (outbound-connection と同じ)

表A.110 local-outbound-connection 属性

属性	デフォルト	説明
outbound-socket-binding-ref		接続の宛先アドレスとポートの判断に使用される **outbound-socket-binding** の名前。

A.31. IO サブシステムの属性



注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-io_3_0.xsd** のスキーマ定義ファイルで確認してください。

表A.111 worker の属性

属性	デフォルト	説明
io-threads		ワーカーに作成する I/O スレッドの数。指定のない場合は、スレッドの数が CPU の数の 2 倍に設定されます。
stack-size	0	ワーカースレッドへの使用を試みるスタックサイズ (バイト単位)。
task-keepalive	60000	コアでないタスクスレッドを生存状態にするミリ秒数。
task-core-threads	2	コアタスクスレッドプールのスレッド数。
task-max-threads		ワーカータスクスレッドプールの最大スレッド数。指定のない場合は、**MaxFileDescriptorCount** Jakarta Management プロパティーを考慮して (設定されている場合)、最大スレッド数が CPU の数の 16 倍に設定されます。

表A.112 buffer-pool の属性

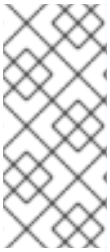
属性	デフォルト	説明
buffer-size		各バッファースライスのサイズ (バイト単位)。指定のない場合は、以下のようにシステムで利用できる RAM を基にサイズが設定されます。 ● RAM が 64 MB 未満の場合は 512 バイト ● RAM が 64 - 128 MB の場合は 1024 バイト (1 KB) ● RAM が 128 MB を超える場合は 16384 バイト (16 KB) この属性のパフォーマンス調整は、JBoss EAP の**Performance Tuning Guide**の Configuring Buffer Pools を参照してください。
buffers-per-slice		大型のバッファーをいくつのスライス (セクション) に分割するか。これは、多数の個別のバッファーに割り当てするよりもメモリーの効率がよくなります。指定のない場合、システムで利用可能な RAM を基にしてスライスの数が設定されます。 ● RAM が 128 MB 未満の場合は 10 ● RAM が 128 MB を超える場合は 20

属性	デフォルト	説明
direct-buffers		バッファープールが直接バッファーを使用するかどうか。直接バッファーをサポートしないプラットフォームがあることに注意してください。

A.32. JAKARTA SERVER FACES モジュールテンプレート

以下の例は、JBoss EAP に異なる Jakarta Server Faces バージョンをインストールする際に必要なさまざまな Jakarta Server Faces モジュールに使用されるテンプレートです。手順の詳細は、Jakarta Server Faces 実装のインストール を参照してください。

例: Mojarra Jakarta Server Faces 実装 JAR の module.xml



注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **IMPL_NAME**
- **VERSION**

```
<module xmlns="urn:jboss:module:1.8" name="com.sun.jsf-impl:IMPL_NAME-VERSION">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <dependencies>
    <module name="javax.faces.api:IMPL_NAME-VERSION"/>
    <module name="javaee.api"/>
    <module name="javax.servlet.jstl.api"/>
    <module name="org.apache.xerces" services="import"/>
    <module name="org.apache.xalan" services="import"/>
    <module name="org.jboss.weld.core"/>
    <module name="org.jboss.weld.spi"/>
    <module name="javax.xml.rpc.api"/>
    <module name="javax.rmi.api"/>
    <module name="org.omg.api"/>
  </dependencies>

  <resources>
    <resource-root path="impl-VERSION.jar"/>
  </resources>
</module>
```

例: MyFaces Jakarta Server Faces 実装 JAR の module.xml

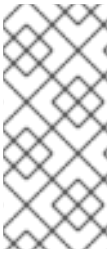
注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **IMPL_NAME**
- **VERSION**

```
<module xmlns="urn:jboss:module:1.8" name="com.sun.jsf-impl:IMPL_NAME-VERSION">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <dependencies>
    <module name="javax.faces.api:IMPL_NAME-VERSION">
      <imports>
        <include path="META-INF/**"/>
      </imports>
    </module>
    <module name="javaee.api"/>
    <module name="javax.servlet.jstl.api"/>
    <module name="org.apache.xerces" services="import"/>
    <module name="org.apache.xalan" services="import"/>

    <!-- extra dependencies for MyFaces -->
    <module name="org.apache.commons.collections"/>
    <module name="org.apache.commons.codec"/>
    <module name="org.apache.commons.beanutils"/>
    <module name="org.apache.commons.digester"/>

    <!-- extra dependencies for MyFaces 1.1
    <module name="org.apache.commons.logging"/>
    <module name="org.apache.commons.el"/>
    <module name="org.apache.commons.lang"/> -->
    <module name="javax.xml.rpc.api"/>
    <module name="javax.rmi.api"/>
    <module name="org.omg.api"/>
  </dependencies>

  <resources>
    <resource-root path="IMPL_NAME-impl-VERSION.jar"/>
  </resources>
</module>
```

例: Mojarra Jakarta Server Faces API JAR の module.xml



注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **IMPL_NAME**
- **VERSION**

```
<module xmlns="urn:jboss:module:1.8" name="javax.faces.api:IMPL_NAME-VERSION">
  <dependencies>
    <module name="com.sun.jsf-impl:IMPL_NAME-VERSION"/>
    <module name="javax.enterprise.api" export="true"/>
    <module name="javax.servlet.api" export="true"/>
    <module name="javax.servlet.jsp.api" export="true"/>
    <module name="javax.servlet.jstl.api" export="true"/>
    <module name="javax.validation.api" export="true"/>
    <module name="org.glassfish.javax.el" export="true"/>
    <module name="javax.api"/>
    <module name="javax.websocket.api"/>
  </dependencies>

  <resources>
    <resource-root path="jsf-api-VERSION.jar"/>
  </resources>
</module>
```

例: MyFaces Jakarta Server Faces API JAR の module.xml

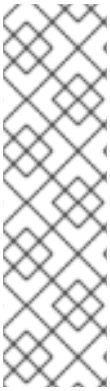
注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **IMPL_NAME**
- **VERSION**

```
<module xmlns="urn:jboss:module:1.8" name="javax.faces.api:IMPL_NAME-VERSION">
  <dependencies>
    <module name="javax.enterprise.api" export="true"/>
    <module name="javax.servlet.api" export="true"/>
    <module name="javax.servlet.jsp.api" export="true"/>
    <module name="javax.servlet.jstl.api" export="true"/>
    <module name="javax.validation.api" export="true"/>
    <module name="org.glassfish.javax.el" export="true"/>
    <module name="javax.api"/>

    <!-- extra dependencies for MyFaces 1.1
    <module name="org.apache.commons.logging"/>
    <module name="org.apache.commons.el"/>
    <module name="org.apache.commons.lang"/> -->
  </dependencies>

  <resources>
    <resource-root path="myfaces-api-VERSION.jar"/>
  </resources>
</module>
```

例: Mojarra Jakarta Server Faces インジェクション JAR の module.xml



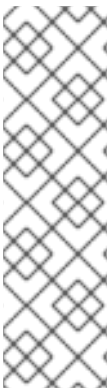
注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **IMPL_NAME**
- **VERSION**
- **INJECTION_VERSION**
- **WELD_VERSION**

```
<module xmlns="urn:jboss:module:1.8" name="org.jboss.as.jsf-injection:IMPL_NAME-VERSION">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <resources>
    <resource-root path="wildfly-jsf-injection-INJECTION_VERSION.jar"/>
    <resource-root path="weld-core-jsf-WELD_VERSION.jar"/>
  </resources>

  <dependencies>
    <module name="com.sun.jsf-impl:IMPL_NAME-VERSION"/>
    <module name="java.naming"/>
    <module name="java.desktop"/>
    <module name="org.jboss.as.jsf"/>
    <module name="org.jboss.as.web-common"/>
    <module name="javax.servlet.api"/>
    <module name="org.jboss.as.ee"/>
    <module name="org.jboss.as.jsf"/>
    <module name="javax.enterprise.api"/>
    <module name="org.jboss.logging"/>
    <module name="org.jboss.weld.core"/>
    <module name="org.jboss.weld.api"/>

    <module name="javax.faces.api:IMPL_NAME-VERSION"/>
  </dependencies>
</module>
```

例: MyFaces Jakarta Server Faces インジェクション JAR の **module.xml**



注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **IMPL_NAME**
- **VERSION**
- **INJECTION_VERSION**
- **WELD_VERSION**

```
<module xmlns="urn:jboss:module:1.8" name="org.jboss.as.jsf-injection:IMPL_NAME-VERSION">
```

```
    <properties>
        <property name="jboss.api" value="private"/>
    </properties>

    <resources>
        <resource-root path="wildfly-jsf-injection-INJECTION_VERSION.jar"/>
        <resource-root path="weld-jsf-WELD_VERSION.jar"/>
    </resources>

    <dependencies>
        <module name="com.sun.jsf-impl:IMPL_NAME-VERSION"/>
        <module name="javax.api"/>
        <module name="org.jboss.as.web-common"/>
        <module name="javax.servlet.api"/>
        <module name="org.jboss.as.jsf"/>
        <module name="org.jboss.as.ee"/>
        <module name="org.jboss.as.jsf"/>
        <module name="javax.enterprise.api"/>
        <module name="org.jboss.logging"/>
        <module name="org.jboss.weld.core"/>
        <module name="org.jboss.weld.api"/>
        <module name="org.wildfly.security.elytron"/>

        <module name="javax.faces.api:IMPL_NAME-VERSION"/>
    </dependencies>
</module>
```

例: MyFaces commons-digester JAR の module.xml

注記

テンプレートの置き換え可能な **VERSION** 変数に適切な値を使用するようにしてください。

```
<module xmlns="urn:jboss:module:1.5" name="org.apache.commons.digester">
    <properties>
        <property name="jboss.api" value="private"/>
    </properties>

    <resources>
        <resource-root path="commons-digester-VERSION.jar"/>
    </resources>

    <dependencies>
        <module name="javax.api"/>
        <module name="org.apache.commons.collections"/>
        <module name="org.apache.commons.logging"/>
        <module name="org.apache.commons.beanutils"/>
    </dependencies>
</module>
```

A.33. JGROUPS サブシステムの属性

jgroups サブシステムのさまざまな要素の属性は以下の表を参照してください。

- 主な属性
- チャネルの属性
- スタックの属性

注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/jboss-as-jgroups_5_0.xsd** のスキーマ定義ファイルで確認してください。

表A.113 jgroups の主な属性

属性	デフォルト	説明
default-channel	ee	デフォルトの JGroups チャネル。
default-stack		デフォルトの JGroups プロトコルスタック。

チャネルの属性
channel 要素の構造は次のとおりです。

- **channel**
  - **fork**
    - **protocol**
  - **protocol**

channel 属性

表A.114 channel 属性

属性	デフォルト	説明
cluster		JGroups チャネルのクラスター名。定義されていない場合はチャネルの名前が使用されます。
module	org.wildfly.clustering.server	チャネルサービスをロードするモジュール。
stack		JGroups チャネルのプロトコルスタック。
statistics-enabled	false	統計を有効にするかどうか。
stats-enabled	false	統計を有効にするかどうか。**非推奨: 代わりに statistics-enabled 属性を使用してください。**

スタックの属性

stack 要素の構造は次のとおりです。

- stack
  - protocol
  - relay
    - remote-site
  - transport
    - thread-pool

stack 属性

表A.115 stack 属性

属性	デフォルト	説明
statistics-enabled	false	スタックのすべてのプロトコルが統計を収集するかどうかを示します。

protocol 属性

一般的に使用されるプロトコルのリストは、JGroups プロトコル の項を参照してください。

表A.116 protocol 属性

属性	デフォルト	説明
module	org.jgroups	プロトコルタイプを解決するモジュール。
properties		このプロトコルのプロパティー。
statistics-enabled	false	スタック設定をオーバーライドして、このプロトコルが統計を収集するかどうかを示します。

relay 属性

表A.117 relay 属性

属性	デフォルト	説明
module	org.jgroups	プロトコルタイプを解決するモジュール。
properties		このプロトコルのプロパティー。
site		ローカルサイトの名前。

属性	デフォルト	説明
statistics-enabled	false	スタック設定をオーバーライドして、このプロトコルが統計を収集するかどうかを示します。

remote-site 属性

表A.118 remote-site 属性

属性	デフォルト	説明
channel		このリモートサイトと通信するために使用されるブリッジチャネルの名前。
cluster		このリモートサイトへのブリッジチャネルのクラスター名。**非推奨: 代わりに明示的に定義された channel を使用してください。**
stack		このリモートサイトへのブリッジを作成するスタック。**非推奨: 代わりに明示的に定義された channel を使用してください。**

transport 属性

表A.119 transport 属性

属性	デフォルト	説明
default-executor		受信メッセージを処理するスレッドプールエグゼキューター。**非推奨: 代わりに事前定義された default スレッドプールを設定してください。**
diagnostics-socket-binding		このプロトコル層の診断ソケットバインディング仕様。通信用の IP インターフェイスとポートを指定するために使用されます。
machine		このノードのマシン (ホスト) 識別子。Infinispan のトポロジー認識のコンシステントハッシュ法によって使用されます。
module	org.jgroups	プロトコルタイプを解決するモジュール。
oob-executor		受信 OOB (out of band) メッセージを処理するスレッドプールエグゼキューター。**非推奨: 代わりに事前定義された oob スレッドプールを設定してください。**
properties		このトランスポートのプロパティー。

属性	デフォルト	説明
rack		サーバーラックなど、このノードのラック識別子。Infinispan のトポロジー認識のコンシステントハッシュ法によって使用されます。
shared	false	**true** の場合、このスタックを使用するすべてのチャネルによって基盤のトランスポートが共有されます。**非推奨: 代わりにチャネルの fork を設定してください。**
site		データセンターなど、このノードのサイト識別子。Infinispan のトポロジー認識のコンシステントハッシュ法によって使用されます。
socket-binding		このプロトコル層のソケットバインディング仕様。通信用の IP インターフェイスおよびポートを指定するために使用されます。
statistics-enabled	false	スタック設定をオーバーライドして、このプロトコルが統計を収集するかどうかを示します。
thread-factory		非同期のトランスポート固有のタスクを処理するために使用するスレッドファクトリー。**非推奨: 代わりに事前定義された internal スレッドプールを設定してください。**
timer-executor		プロトコル関連のタイミングタスクを処理するスレッドプールエグゼキューター。**非推奨: 代わりに事前定義された timer スレッドプールを設定してください。**

thread-pool 属性

表A.120 thread-pool 属性

属性	デフォルト	説明
keepalive-time	5000L	アイドル時にプールスレッドの実行を継続すべき時間 (ミリ秒単位) 。指定されていない場合は、スレッドはエグゼキューターが終了するまで実行します。
max-threads	4	スレッドプールの最大サイズ。
min-threads	2	**max-threads** よりも小さいコアスレッドプールサイズ。定義されていない場合、コアスレッドプールサイズは **max-threads** と同じになります。

属性	デフォルト	説明
queue-length	500	キューの長さ。

A.34. JGROUPS PROTOCOLS

プロトコル	プロトコルタイプ	説明
ASYM_ENCRYPT	暗号化	クラスターのコーディネーターに保存されるシークレットキーを使用して、クラスターメンバー間のメッセージを暗号化します。
AUTH	認証	認証のレイヤーをクラスターメンバーに提供します。
azure.AZURE_PING	検出	Microsoft Azure の blob ストレージを使用したノード検出をサポートします。
FD_ALL	障害検出	簡単なハートビートプロトコルを基にした障害検出を提供します。
FD_SOCK	障害検出	クラスターメンバー間で作成された TCP ソケットのリングを基にした障害検出を提供します。
JDBC_PING	検出	メンバーがアドレスを書き込む共有データベースを使用してクラスターメンバーを検出します。
MERGE3	マージ	クラスターの分割が発生したときにサブクラスターをマージします。
MFC	フロー制御	送信元とすべてのクラスターメンバーとの間でマルチキャストフロー制御を提供します。
MPING	検出	IP マルチキャストでクラスターメンバーを検出します。
pbcast.GMS	グループメンバーシップ	新規メンバーのクラスター参加、既存メンバーによる離脱リクエスト、およびクラッシュしたメンバーの **SUSPECT** メッセージを含む、グループメンバーシップを処理します。
pbcast.NAKACK2	メッセージの送信	メッセージの信頼性および順序を確実にし、1 つの送信元から送信されたすべてのメッセージが送信順に受信されることを保証します。
pbcast.STABLE	メッセージの安定性	すべてのメンバーによって送信されたメッセージを削除します。
PING	検出	クラスターメンバーの動的検出のサポートによる、メンバーの最初の検出。

プロトコル	プロトコルタイプ	説明
SASL	認証	SASL メカニズムを使用してクラスターメンバーに認証のレイヤーを提供します。
SYM_ENCRYPT	暗号化	共有キーストアを使用して、クラスターメンバー間のメッセージを暗号化します。
S3_PING	検出	Amazon S3 を使用して最初のメンバーを検出します。
TCPGOSSIP	検出	外部のゴシップルーターを使用してクラスターメンバーを検出します。
TCPPING	検出	クラスターを形成するクラスターメンバーのアドレスの静的リストが含まれます。
UFC	フロー制御	送信元とすべてのクラスターメンバーとの間でユニキャストフロー制御を提供します。
UNICAST3	メッセージの送信	ユニキャストメッセージのメッセージの信頼性および順序を確実にし、1つの送信元から送信されたすべてのメッセージが送信順に受信されることを保証します。
VERIFY_SUSPECT	障害検出	疑わしいメンバーをエビクトする前に最終的に ping してそのメンバーが不能になったことを検証します。

汎用プロトコルの属性

すべてのプロトコルは以下の属性にアクセスできます。

表A.121 protocol 属性

属性	デフォルト	説明
module	org.jgroups	プロトコルタイプを解決するモジュール。
properties		このプロトコルのプロパティー。
statistics-enabled	false	統計を有効にするかどうか。

認証プロトコル

認証プロトコルは、認証を実行するために使用されます。主な役目は、認証されたメンバーのみがクラスターに参加できるようにすることです。これらのプロトコルは、**GMS** プロトコルの下に位置するため、クラスター参加のリクエストをリッスンする可能性があります。

- AUTH
- SASL

AUTH 属性

AUTH プロトコルにはその他の属性は含まれませんが、トークンが子要素として定義されている必要があります。

注記

このプロトコルを定義するとき、**protocol** 要素の代わりに **auth-protocol** 要素が使用されます。

トークンタイプ
Elytron をセキュリティーに使用する場合、以下の認証トークンの 1 つを使用することが推奨されます。これらの認証トークンは、Elytron と使用する目的で設計されており、レガシーのセキュリティー設定とは使用できない可能性があります。

表A.122 Elytron トークンタイプ

トークン	説明
cipher-token	共有の秘密が変換される認証トークン。変換に使用されるデフォルトのアルゴリズムは RSA です。
digest-token	共有の秘密が変換される認証トークン。変換に使用されるデフォルトのアルゴリズムは SHA-256 です。
plain-token	共有の秘密に追加の変換を行わない認証トークン。

以下の認証トークンは JGroups から継承されます。認証が望ましいすべての設定で使用の対象となります。

表A.123 JGroups トークンタイプ

トークン	説明
MD5Token	共有の秘密が MD5 または SHA ハッシュを使用して暗号化される認証トークン。この暗号化に使用されるデフォルトのアルゴリズムは MD5 です。
SimpleToken	共有の秘密に追加の変換を行わない認証トークン。このトークンは大文字と小文字を区別せず、文字列の一致を判断するときに大文字と小文字を考慮しません。
X509Token	共有の秘密が X509 証明書を使用して暗号化される認証トークン。

SASL の属性

表A.124 SASL の属性

属性	デフォルト	説明
client_callback_handler		ノードがクライアントとして動作するときに使用する **CallbackHandler** のクラス名。

属性	デフォルト	説明
client_name		ノードがクライアントとして動作するときに使用する名前。JAAS ログインモジュールを使用する場合、この名前はサブジェクトの取得にも使用されます。
client_password		ノードがクライアントとして動作するときに使用するパスワード。JAAS ログインモジュールを使用する場合、このパスワードはサブジェクトの取得にも使用されます。
login_module_name		SASL クライアントおよびサーバーを作成するためにサブジェクトとして使用する JAAS ログインモジュールの名前。この属性は、GSSAPI などの一部の **mech** 値でのみ必要です。
mech		SASL 認証メカニズムの名前。この名前にはローカル SASL プロバイダーがサポートするすべてのメカニズムを指定でき、JDK はデフォルトで **CRAM-MD5**、**DIGEST-MD5**、**GSSAPI**、および **NTLM** を提供します。
sasl_props		定義された **mech** のプロパティー。
server_callback_handler		ノードがサーバーとして動作するときに使用する **CallbackHandler** のクラス名。
server_name		完全修飾サーバー名。
timeout	5000	チャレンジへの応答を待つ時間 (ミリ秒単位)。

検出プロトコル

以下のプロトコルはクラスターの最初のメンバーシップを見つけるために使用され、その後現在のコーディネーターを決定するために使用できます。検出プロトコルのリストは次のとおりです。

- AZURE_PING
- JDBC_PING
- MPING
- PING
- S3_PING
- TCPGOSSIP
- TCPPING

AZURE_PING の属性

表A.125 AZURE_PING の属性

属性	デフォルト	説明
container		PING データに使用する blob コンテナーの名前。有効な DNS 名を指定する必要があります。
storage_access_key		ストレージアカウントのシークレットアクセスキー。
storage_account_name		blob コンテナーが含まれる Microsoft Azure ストレージアカウントの名前。

JDBC_PING の属性

表A.126 JDBC_PING の属性

属性	デフォルト	説明
data-source		接続および JNDI ルックアッププロパティーの代わりに使用されるデータソース参照。

注記

JDBC_PING プロトコルを定義するとき、**protocol** 要素の代わりに **jdbc-protocol** 要素が使用されます。

S3_PING の属性

表A.127 S3_PING の属性

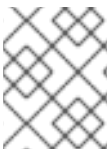
属性	デフォルト	説明
access_key		S3 バケットのアクセスに使用される Amazon S3 アクセスキー。
host	s3.amazonaws.com	S3 web サービスの宛先。
location		使用する Amazon S3 バケットの名前。バケットが存在する必要があり、一意な名前を使用する必要があります。
pre_signed_delete_url		DELETE 操作に使用する事前署名付き URL。

属性	デフォルト	説明
port	• **use_ssl** が **true** の場合は 443。 • **use_ssl** が **false** の場合は 80。	web サービスがリッスンしているポート。
pre_signed_put_url		PUT 操作に使用する事前署名付き URL。
prefix		設定され、**location** が設定されている場合、バケット名を **PREFIX-LOCATION** として定義します。設定され、指定の **PREFIX-LOCATION** にバケットが存在しない場合は、 **PREFIX** の後に無作為の UUID が続くバケット名になります。
secret_access_key		S3 バケットのアクセスに使用される Amazon S3 のシークレットアクセスキー。
use_ssl	true	ホストとポートの組み合わせと通信するときに SSL が使用されるかどうかを決定します。

TCPGOSSIP の属性

表A.128 TCPGOSSIP の属性

属性	デフォルト	説明
socket-binding		このプロトコル層のソケットバインディング仕様。**非推奨: 代わりに socket-bindings を使用してください。**
socket-bindings		このプロトコルのアウトバウンドソケットバインディング。



注記

TCPGOSSIP プロトコルを定義するとき、 **protocol** 要素の代わりに **socket-discovery-protocol** 要素が使用されます。

TCPPING の属性

表A.129 TCPPING の属性

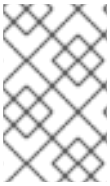
属性	デフォルト	説明
socket-binding		このプロトコル層のソケットバインディング仕様。**非推奨: 代わりに socket-bindings を使用してください。**
socket-bindings		このプロトコルのアウトバウンドソケットバインディング。

注記

TCPPING プロトコルを定義するとき、**protocol** 要素の代わりに **socket-discovery-protocol** 要素が使用されます。

暗号化プロトコル

以下のプロトコルは通信スタックをセキュアにするために使用されます。暗号化はクラスターのすべてのメンバーが持つ共有のシークレットキーを基にします。このキーは、**SYM_ENCRYPT** を使用している場合は共有のキーストアから取得され、**ASYM_ENCRYPT** を使用している場合はパブリック/プライベートキーの交換から取得されます。以下のプロトコルのいずれかを定義すると、結果となる XML に **encrypt-protocol** 要素が作成されます。



注記

ASYM_ENCRYPT を使用している場合は、同じスタックに **AUTH** プロトコルが定義されている必要があります。**SYM_ENCRYPT** を使用している場合、**AUTH** プロトコルは任意です。

- ASYM_ENCRYPT
- SYM_ENCRYPT

ASYM_ENCRYPT の属性

表A.130 ASYM_ENCRYPT の属性

属性	デフォルト	説明
key-alias		指定されたキーストアから取得される暗号化キーのエイリアス。
key-credential-reference		キーストアから暗号化キーを取得するために必要な認証情報。
key-store		暗号化キーが含まれるキーストアへの参照。

SYM_ENCRYPT の属性

表A.131 SYM_ENCRYPT の属性

属性	デフォルト	説明
key-alias		指定されたキーストアから取得される暗号化キーのエイリアス。
key-credential-reference		キーストアから暗号化キーを取得するために必要な認証情報。
key-store		暗号化キーが含まれるキーストアへの参照。

障害検出プロトコル
以下のプロトコルは、クラスターのメンバーを調査して生存の有無を判断するために使用されます。これらのプロトコルには、汎用属性以外の属性はありません。

- FD_ALL
- FD_SOCK
- VERIFY_SUSPECT

フロー制御プロトコル
以下のプロトコルは、メッセージの送信側の速度を最も遅い受信側に合わせて処理するフロー制御を行います。送信側が送信するメッセージの速度が継続して受信側よりも速い場合、受信側はメッセージをキューに置くか破棄するため、再送信が発生します。これらのプロトコルには、汎用属性以外の属性はありません。

- MFC - マルチキャストフロー制御
- UFC - ユニキャストフロー制御

グループメンバーシッププロトコル
pbcast.GMS プロトコルは、新規メンバーのクラスターへの参加、既存メンバーによるクラスターからの離脱、およびクラッシュした疑いのあるメンバーに対応します。このプロトコルには、汎用属性以外の属性はありません。

マージプロトコル
クラスターが分割された場合、**MERGE3** プロトコルによってサブクラスターが元どおりにマージされます。このプロトコルはクラスターメンバーを元どおりにマージするロールを果たしますが、クラスターの状態はマージしません。マージ状態のコールバックはアプリケーションが対応します。このプロトコルには、汎用属性以外の属性はありません。

メッセージの安定性
pbcast.STABLE プロトコルは、クラスターのすべてのメンバーが受け取ったメッセージをガベッジコレクションで処理します。このプロトコルは、ダイジェストと呼ばれるメンバーのメッセージ番号が含まれる安定したメッセージを送信します。クラスターのすべてのメンバーが他のメンバーのダイジェストを受信した後、メッセージを再送信テーブルから削除することができます。このプロトコルには、汎用属性以外の属性はありません。

信頼できるメッセージの送信
以下のプロトコルは、信頼できるメッセージの配信と、クラスターのすべてのノードに送信されたメッセージの FIFO プロパティーを提供します。信頼できる配信では、すべてのメッセージに番号が付けられ、シーケンス番号が受信されなかった場合に再送信リクエストが送信されるため、送信元が送った

メッセージを損失することはありません。これらのプロトコルには、汎用属性以外の属性はありません。

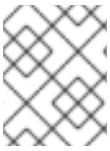
- pbcast.NAKACK2
- pbcast.UNICAST3

非推奨となったプロトコル
以下のプロトコルは非推奨となり、クラス名のみが含まれるプロトコルによって置き換えられました。たとえば、**org.jgroups.protocols.ASYM_ENCRYPT** を指定する代わりに、プロトコル名は **ASYM_ENCRYPT** になります。

- org.jgroups.protocols.ASYM_ENCRYPT
- org.jgroups.protocols.AUTH
- org.jgroups.protocols.JDBC_PING
- org.jgroups.protocols.SYM_ENCRYPT
- org.jgroups.protocols.TCPGOSSIP
- org.jgroups.protocols.TCPPING

A.35. APACHE HTTP SERVER の MOD_CLUSTER ディレクティブ

mod_cluster コネクターは Apache HTTP Server ベースのロードバランサーです。通信チャネルを使用して、リクエストを Apache HTTP Server からアプリケーションサーバーノードのセットの 1 つに転送します。mod_cluster の設定には以下のディレクティブを指定できます。



注記

mod_cluser は Apache HTTP Server に転送しなければならない URL を自動的に設定するため、**ProxyPass** ディレクティブを使用する必要はありません。

表A.132 mod_cluster ディレクティブ

ディレクティブ	説明	値
CreateBalancers	バランサーが Apache HTTP Server の VirtualHosts でどのように作成されるかを定義します。**ProxyPass /balancer://mycluster1/** のようなディレクティブを許可します。	• **0**: Apache HTTP Server に定義されるすべての VirtualHosts を作成します。 • **1**: バランサーを作成しません (バランサー名を定義するため最低でも 1 つの **ProxyPass** または **ProxyMatch** が必要です)。 • **2**: メインサーバーのみ作成します **(デフォルト)**。

ディレクティブ	説明	値
UseAlias	エイリアスがサーバー名に対応することを確認します。	• **0**: エイリアスを無視します **(デフォルト)**。 • **1**: エイリアスをチェックします。
LBstatusRecalTime	負荷分散ロジックがノードの状態を再計算する間隔 (秒単位)。	デフォルト: 5 秒
WaitBeforeRemove	削除されたノードを httpd が記憶しなくなるまでの時間 (分単位)。	デフォルト: 10 秒
ProxyPassMatch/ProxyPass	ProxyPassMatch および ProxyPass は、バックエンド URL の代わりに!を使用するとパスのリバースプロキシーを防ぐ **mod_proxy** ディレクティブです。これは、Apache HTTP Server が静的なコンテンツに対応できるようにするために使用されます。例: **ProxyPassMatch ^(/.*\.gif)$!**この例では、Apache HTTP Server は直接 .gif ファイルに対応できます。	

注記

JBoss EAP 7 のセッションのパフォーマンス最適化により、ホットスタンバイノードの設定はサポートされません。

mod_manager

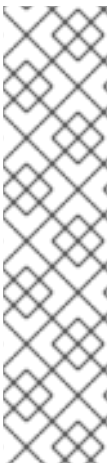
mod_manager ディレクティブのコンテキストは、指定がある場合を除きすべて VirtualHost になります。サーバー設定 コンテキストは、ディレクティブが VirtualHost 設定外になければならないことを示します。そうでない場合、エラーメッセージが表示され、Apache HTTP Server が開始しません。

表A.133 mod_manager ディレクティブ

ディレクティブ	説明	値
EnableMCPMReceive	VirtualHost がノードから **MCPM** を受信できるようにします。mod_cluster が動作するようにするため、Apache HTTP Server 設定に **EnableMCPMReceive** が含まれます。VirtualHost のアドバタイズを設定する場所に保存します。	

ディレクティブ	説明	値
MemManagerFile	設定の保存、共有メモリーまたはロックされたファイルのキー生成に **mod_manager** が使用する名前のベース名。絶対パス名である必要があります。ディレクトリーは必要な場合に作成されます。コンテキスト: server config	**$server_root/logs/**
Maxcontext	mod_cluster によってサポートされるコンテキストの最大数。コンテキスト: server config	デフォルト: **100**
Maxnode	mod_cluster によってサポートされるノードの最大数。コンテキスト: server config	デフォルト: **20**
Maxhost	mod_cluster によってサポートされるホスト (エイリアス) の最大数。バランサーの最大数も含まれます。コンテキスト: server config	デフォルト: **20**
Maxsessionid	**mod_cluster-manager** ハンドラーにアクティブなセッションの数を提供するために保存されるアクティブ **sessionid** の数。5 分以内に mod_cluster がセッションから情報を受信しないとセッションは非アクティブになります。コンテキスト: server config。このフィールドはデモおよびデバッグの目的のみで使用されます。	**0**: ロジックはアクティベートされません。
MaxMCMPMaxMessSize	他の Max ディレクティブからの MCMP メッセージの最大サイズ。	他の Max ディレクティブより計算されます。最小: **1024**
ManagerBalancerName	JBoss EAP インスタンスがバランサー名を提供しない場合に使用されるバランサーの名前。	**mycluster**
PersistSlots	ファイルのノード、エイリアス、およびコンテキストを保持するよう mod_slotmem に伝えます。コンテキスト: server config	オフ
CheckNonce	**mod_cluster-manager** ハンドラーを使用する際に **nonce** のチェックを切り替えます。	on/off、デフォルト: on - Nonce をチェック

ディレクティブ	説明	値
AllowDisplay	**mod_cluster-manager** メインページの追加表示を切り替えます。	on/off、デフォルト: off - バージョンのみを表示
AllowCmd	**mod_cluster-manager** URL を使用するコマンドを許可します。	on/off、デフォルト: on - コマンドを許可
ReduceDisplay	メインの **mod_cluster-manager** ページに表示される情報を減らし、ページ上により多くのノードを表示できるようにします。	on/off、デフォルト: off - 情報をすべて表示
SetHandler mod_cluster-manager	mod_cluster がクラスターから可視できるノードの情報を表示します。情報には一般的な情報が含まれ、追加でアクティブなセッションの数を調べます。 <Location /mod_cluster-manager> SetHandler mod_cluster-manager Require ip 127.0.0.1 </Location>	on/off、デフォルト: off



注記

httpd.conf に定義された場所にアクセスする場合:

- Transferred: バックエンドサーバーに送信された POST データに対応。
- Connected: mod_cluster の状態ページが要求されたときに処理された要求の数に対応。
- Num_sessions: mod_cluster がアクティブと報告するセッションの数に対応 (過去 5 分以内に要求があった場合)。Maxsessionid がゼロの場合、このフィールドは存在しません。 このフィールドはデモおよびデバッグの目的でのみ使用されます。

A.36. MODCLUSTER サブシステムの属性

modcluster サブシステムの構造は次のとおりです。

- proxy
  - load-provider=dynamic
    - custom-load-metric
    - load-metric

- load-provider=simple
- ssl

load-provider=dynamic リソースにより、負荷分散の動作を決定するために CPU、セッション、ヒープ、メモリー、重みなどの要素を設定できます。

load-provider=simple リソースを使用すると、静的な定数のみを factor 属性として設定できます。これは、ユーザーが受信 HTTP リクエストの負荷分散に動的なルールまたは複雑なルールを必要としない場合に役立ちます。

注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/jboss-as-mod-cluster_3_0.xsd** のスキーマ定義ファイルで確認してください。

表A.134 proxy 設定オプション

属性	デフォルト	説明
advertise	true	マルチキャストベースのアドバタイズメカニズムを有効にするかどうか。
advertise-security-key		httpd インスタンスと、httpd インスタンスからのアドバタイズをリッスンする JBoss EAP サーバーとの間の共有の秘密です。
advertise-socket		登録するリバースプロキシーのバランサーの名前。
auto-enable-contexts	true	**false** に設定された場合、コンテキストは無効としてリバースプロキシーと登録されます。**enable-context** 操作を使用するか、mod_cluster_manager コンソールを使用するとコンテキストを有効にできます。
balancer		登録するリバースプロキシーのバランサーの名前。設定されていない場合、値は **ManagerBalancerName** ディレクティブで Apache HTTP Server 側に設定され、デフォルトでは **mycluster** になります。
connector		mod_cluster のリバースプロキシーが接続する Undertow リスナーの名前。
excluded-contexts		リバースプロキシーでの登録から除外されるコンテキストのリスト。ホストの指定がない場合、**localhost** がホストと仮定されます。**ROOT** は web アプリケーションのルートコンテキストを示します。

属性	デフォルト	説明
flush-packets	false	web サーバーへのパケットのフラッシングを有効にするかどうか。
flush-wait	-1	httpd でパケットをフラッシュする前に待機する期間。最大値は **2,147,483,647**。
listener		リバースプロキシーと登録される Undertow リスナーの名前。
load-balancing-group		設定された場合、リクエストはロードバランサーの指定のロードバランシンググループに送信されます。
max-attempts	1	リバースプロキシーが指定リクエストのワーカーへの送信を試みる回数。 この回数試行した後に送信を断念します。
node-timeout	-1	ワーカーへのプロキシー接続のタイムアウト (秒単位)。mod_cluster はこの期間バックエンド応答を待ち、その経過後にエラーを返します。**node-timeout** 属性が未定義である場合、httpd **ProxyTimeout** ディレクティブが使用されます。**ProxyTimeout** が未定義である場合、httpd **Timeout** ディレクティブが使用され、デフォルトは 300 秒になります。
ping	10	ping に対する pong 返答を待つ時間 (秒単位)。
proxies		**socket-binding-group** の **outbound-socket-binding** によって定義される登録する mod_cluster のプロキシーのリスト。
proxy-list		プロキシーのリスト。形式は **HOST_NAME:PORT** で、コンマで区切られます。**非推奨になりました。proxies の使用が推奨されます。**
proxy-url	/	MCMP リクエストのベース URL。

属性	デフォルト	説明
session-draining-strategy	DEFAULT	Web アプリケーションのアンデプロイメント中に使用されるセッションドレインストラテジー。有効な値は **DEFAULT**、**ALWAYS**、または **NEVER** です。 **DEFAULT** web アプリケーションが分散可能でない場合のみ、web アプリケーションのアンデプロイ前にセッションをドレインします。 **ALWAYS** web アプリケーションが分散可能であっても、web アプリケーションのアンデプロイ前に常にセッションをドレインします。 **NEVER** web アプリケーションのアンデプロイ前にセッションをドレインしません。
load-provider=simple		動的ロードプロバイダーが存在しない場合に使用するロードプロバイダー。各クラスターメンバーに負荷係数 **1** を割り当て、負荷分散アルゴリズムを適用せずに作業を均等に分散します。
smax	-1	httpd の soft maximum アイドル接続数。
socket-timeout	20	タイムアウトの前およびプロキシーをエラーのように警告する前に、httpd プロキシーから MCMP コマンドへの応答を待つ秒数。
ssl-context		mod_cluster によって使用される **SSLContext** への参照。
status-interval	10	STATUS メッセージがアプリケーションサーバーからリバースプロキシーへ送信される秒数。**1** から **2,147,483,647** までの値が許可されます。
sticky-session	true	あるセッションの後続リクエストを可能な限り同じノードへルーティングするべきかどうか。
sticky-session-force	false	バランサーがリクエストをスタックしたノードへルーティングできない場合に、リバースプロキシーがエラーを返すかどうか。スティッキーセッションが無効な場合は無視されます。
sticky-session-remove	false	フェイルオーバー時にセッション情報を削除します。

属性	デフォルト	説明
stop-context-timeout	10	分散可能なコンテキストの場合は、コンテキストが保留中のリクエストを処理するのを待つ最大時間 (秒単位)。 分散可能でないコンテキストの場合は、コンテキストがアクティブなセッションを破棄するのを待つ最大時間 (秒単位)。
ttl	-1	smax を超えるアイドル接続の TTL (Time To Live、秒単位)。**-1** から **2,147,483,647** までの値が許可されます。
worker-timeout	-1	httpd で利用可能なワーカーによるリクエスト処理の待機時間のタイムアウト値。**-1** から **2,147,483,647** までの値が許可されます。

表A.135 load-provider=dynamic 設定オプション

属性	デフォルト	説明
decay	2	減退 (Decay)。
history	9	履歴。
initial-load	0	ノードによって報告される初期負荷です。有効な範囲は **0-100** で、**0** が最大負荷になります。 この属性は、新しく加わるノードの負荷値を段階的に増大させ、クラスターに加わる際のオーバーロードを回避するのに役立ちます。 この動作を無効にするには、値を **-1** に設定します。無効にすると、ノードは負荷値 **100** を報告し、クラスターに加わるときに負荷がないことを示します。

表A.136 custom-load-metric 属性オプション

属性	デフォルト	説明
capacity	1.0	メトリックの容量
class		カスタムメトリックのクラス名。
property		メトリックのプロパティー。
weight	1	メトリックの重さ。

表A.137 load-metric 属性オプション

属性	デフォルト	説明
capacity	1.0	メトリックの容量
property		メトリックのプロパティー。
type		メトリックのタイプ。有効な値は **cpu**、**mem**、**heap**、**sessions**、**receive-traffic**、**send-traffic**、**requests**、または **busyness** です。
weight	1	メトリックの重さ。

表A.138 ssl 属性オプション

属性	デフォルト	説明
ca-certificate-file		認証局。
ca-revocation-url		認証局の失効リスト。
certificate-key-file	${user.home}/.keystore	証明書のキーファイル。
cipher-suite		許可された暗号スイート。
key-alias		キーエイリアス。
password	changeit	パスワード。
protocol	TLS	有効な SSL プロトコル。

A.37. MOD_JK ワーカープロパティー

workers.properties ファイルは mod_jk がクライアント要求を渡すワーカーの動作を定義します。**workers.properties** ファイルは、異なるサーブレットコンテナーが存在する場所と、ワークロードをアプリケーションサーバーすべてで分散する方法を定義します。

プロパティーの一般的な構造は **worker.WORKER_NAME.DIRECTIVE** です。**WORKER_NAME** は、JBoss EAP **undertow** サブシステムで設定された **instance-id** と一致しなければならない一意な名前です。**DIRECTIVE** はワーカーに適用される設定です。

Apache mod_jk ロードバランサーの設定リファレンス

テンプレートはデフォルトのロードバランサーごとの設定を指定します。ロードバランサーの設定内でテンプレートを上書きできます。

表A.139 グローバルプロパティー

プロパティー	説明
worker.list	**mod_jk** によって使用されるワーカー名のコンマ区切りリスト。

表A.140 必須ディレクティブ

プロパティー	説明
type	ワーカーのタイプ。デフォルトのタイプは **ajp13** です。他の可能な値は **ajp14**、**lb**、**status** です。これらのディレクティブの詳細は、https://tomcat.apache.org/connectors-doc/reference/workers.html の Apache Tomcat Connectors Reference を参照してください。

表A.141 負荷分散ディレクティブ

プロパティー	説明
balance_workers	ロードバランサーが管理する必要があるワーカーノードを指定します。同じロードバランサーにディレクティブを複数回使用できます。コンマ区切りのワーカーノード名のリストで設定されます。
sticky_session	同じセッションからのリクエストを常に同じワーカーにルーティングするかどうかを指定します。デフォルトは **1** で、スティッキーセッションが有効になります。スティッキーセッションを無効にするには **0**. を設定します。すべてのリクエストが実際にステートレスである場合を除き、スティッキーセッションは通常有効にする必要があります。

表A.142 接続ディレクティブ

プロパティー	説明
host	バックエンドサーバーのホスト名または IP アドレス。バックエンドサーバーは **ajp** プロコトルスタックをサポートする必要があります。デフォルト値は **localhost** です。
port	定義されたプロトコルリクエストをリッスンしているバックエンドサーバーインスタンスのポート番号。デフォルトの値は、AJP13 ワーカーのデフォルトのリッスンポートである **8009** です。AJP14 ワーカーのデフォルト値は **8011** です。

プロパティー	説明
ping_mode	ネットワークの状態に対して接続がプローブされる条件。プローブは CPing に空の AJP13 パケットを使用し、応答で CPong を想定します。ディレクティブフラグの組み合わせを使用して条件を指定します。フラグはコンマまたはスペースで区切られません。ping_mode は C、P、I、および A の任意の組み合わせです。 ● C - Connect (接続)。サーバーへの接続後に 1 回接続をプローブします。connect_timeout の値を使用してタイムアウトを指定します。指定がないと、ping_timeout の値が使用されます。 ● P - Prepost (プレポスト)。各リクエストをサーバーに送信する前に接続をプローブします。prepost_timeout ディレクティブを使用してタイムアウトを指定します。指定がないと、ping_timeout の値が使用されます。 ● I - Interval (間隔)。connection_ping_interval で指定された間隔で接続をプローブします (指定がある場合)。指定がないと、ping_timeout の値が使用されます。 ● A - All (すべて)。すべての接続プローブを使用することを指定する CPI のショートカットです。
ping_timeout、connect_timeout、prepost_timeout、connection_ping_interval	上記の接続プローブ設定のタイムアウト値。値はミリ秒単位で指定され、**ping_timeout** のデフォルト値は **10000** です。
lbfactor	各バックエンドサーバーインスタンスの負荷分散係数を指定します。より強力なサーバーにより多くのワークロードを割り当てる場合に便利です。ワーカーにデフォルトの 3 倍の負荷を割り当てるには、**worker.my_worker.lbfactor=3** のように **3** を設定します。

以下の例は、ポート **8009** でリッスンする **node1** および **node2** の 2 つのワーカーノードの間でスティッキーセッションを用いて負荷を分散します。

例: workers.properties ファイル

```
# Define list of workers that will be used for mapping requests
worker.list=loadbalancer,status

# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=node1.mydomain.com
worker.node1.type=ajp13
worker.node1.ping_mode=A
worker.node1.lbfactor=1

# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host= node2.mydomain.com
```

```
worker.node2.type=ajp13
worker.node2.ping_mode=A
worker.node2.lbfactor=1

# Load-balancing behavior
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1

# Status worker for managing load balancer
worker.status.type=status
```

Apache mod_jk の設定の詳細は、本書の範囲外です。Apache のドキュメント を参照してください。

A.38. SECURITY MANAGER サブシステム

security-manager サブサブシステム自体には設定可能な属性はありませんが、**deployment-permissions=default** という設定可能な属性を持つ子リソースが 1 つあります。

注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-security-manager_1_0.xsd** のスキーマ定義ファイルで確認してください。

表A.143 deployment-permissions 設定オプション

属性	説明
maximum-permissions	デプロイメントまたは JAR に付与できる最大パーミッションセット。
minimum-permissions	デプロイメントまたは JAR に付与できる最小パーミッションセット。

A.39. JBOSS CORE SERVICES からの OPENSSL のインストール

JBoss Core Services OpenSSL のファイルは、ZIP または RPM ディストリビューションからインストールできます。インストールの方法に応じて、以下の手順にしたがってください。

注記

Red Hat Enterprise Linux 8 では、標準のシステム OpenSSL に対応しているため、JBoss Core Services からの OpenSSL のインストールは必要ありません。

JBoss Core Services OpenSSL ZIP ファイルディストリビューションの使用

注記

ZIP アーカイブの **libs/** ディレクトリーへのパスは、Linux の場合は **jbcs-openssl-VERSION/openssl/lib(64)**、Windows の場合は **jbcs-openssl-VERSION/openssl/bin** になります。

1. お使いのオペレーティングシステムとアーキテクチャーに該当する OpenSSL パッケージを Software Downloads ページからダウンロードします。
2. ダウンロードした ZIP ファイルをインストールディレクトリーでデプロイメントします。
3. OpenSSL ライブラリーのある場所を JBoss EAP に通知します。
これを行うには、以下の方法の 1 つを使用します。以下のコマンドでは、必ず **JBCS_OPENSSL_PATH** を JBoss Core Services OpenSSL ライブラリーへのパスに置き換えてください (例: **/opt/rh/jbcs-httpd24/root/usr/lib64**)。
   - 以下の引数を使用すると、OpenSSL パスを **standalone.conf** または **domain.conf** 設定ファイルの **JAVA_OPTS** 変数に追加できます。

     ```
     JAVA_OPTS="$JAVA_OPTS -Dorg.wildfly.openssl.path=JBCS_OPENSSL_PATH
     ```

   - 以下の管理 CLI コマンドを使用すると、OpenSSL パスを指定するシステムプロパティーを定義できます。

     ```
     /system-property=org.wildfly.openssl.path:add(value=JBCS_OPENSSL_PATH)
     ```

重要

使用する方法に関係なく、サーバーを再起動して **JAVA_OPTS** の値またはシステムプロパティーを有効にする必要があります。サーバーをリロードするだけでは有効にできません。

JBoss Core Services OpenSSL RPM ディストリビューションの使用

1. システムが JBoss Core Services チャンネルに登録されていることを確認してください。
   a. オペレーティングシステムバージョンとアーキテクチャーの JBoss Core Services の CDN リポジトリー名を決定します。
      - **RHEL 6**: jb-coreservices-1-for-rhel-6-server-rpms
      - **RHEL 7**: jb-coreservices-1-for-rhel-7-server-rpms

   b. システムでリポジトリーを有効にします。

      ```
      # subscription-manager repos --enable REPO_NAME
      ```

   c. 以下のメッセージが表示されたことを確認してください。

      ```
      Repository REPO_NAME is enabled for this system.
      ```

2. このチャンネルから OpenSSL をインストールします。

   ```
   # yum install jbcs-httpd24-openssl
   ```

3. インストールの完了後、**/opt/rh/jbcs-httpd24/root/usr/lib64** の JBCS OpenSSL ライブラリーを利用できます。 x86 アーキテクチャーでは **/opt/rh/jbcs-httpd24/root/usr/lib** になります。

4. OpenSSL ライブラリーのある場所を JBoss EAP に通知します。
これを行うには、以下の方法の 1 つを使用します。以下のコマンドでは、必ず **JBCS_OPENSSL_PATH** を JBoss Core Services OpenSSL ライブラリーへのパスに置き換えてください (例: **/opt/rh/jbcs-httpd24/root/usr/lib64**)。

   - サービス設定ファイルの **eap7-standalone** または **eap7-domain** 設定の **WILDFLY_OPTS** 変数を更新できます。

     ```
     WILDFLY_OPTS="$WILDFLY_OPTS -
     Dorg.wildfly.openssl.path=JBCS_OPENSSL_PATH"
     ```

   - 以下の管理 CLI コマンドを使用すると、OpenSSL パスを指定するシステムプロパティーを定義できます。

     ```
     /system-property=org.wildfly.openssl.path:add(value=JBCS_OPENSSL_PATH)
     ```

     重要

     使用する方法に関係なく、サーバーを再起動して **WILDFLY_OPTS** の値またはシステムプロパティーを有効にする必要があります。サーバーをリロードするだけでは有効にできません。

A.40. OPENSSL を使用するよう JBOSS EAP を設定

JBoss EAP が OpenSSL を使用するよう設定する方法は複数あります。

- **elytron** サブシステムを再設定し、OpenSSL の優先度を上げて、OpenSSL がデフォルトですべての場合で使用されるようにすることができます。

  注記

  OpenSSL は **elytron** サブシステムにインストールされていますが、デフォルトの TLS プロバイダーではありません。

  ```
  /subsystem=elytron:write-attribute(name=initial-providers, value=combined-providers)
  /subsystem=elytron:undefine-attribute(name=final-providers)

  reload
  ```

- **elytron** サブシステムでは、OpenSSL プロバイダーは **ssl-context** リソースでも指定できます。これにより、デフォルトの優先度を使用せずに、OpenSSL プロトコルを状況に応じて選択できます。
ssl-context リソースを作成し、Elytron ベースの SSL/TLS 設定で OpenSSL ライブラリーを使用するには、以下のコマンドを使用します。

  ```
  /subsystem=elytron/server-ssl-context=httpsSSC:add(key-manager=localhost-manager,
  trust-manager=ca-manager, provider-name=openssl)

  reload
  ```

- レガシーの **security** サブシステムの SSL/TLS 設定で OpenSSL ライブラリーを使用するには、以下のコマンドを使用します。

```
/core-service=management/security-realm=ApplicationRealm/server-identity=ssl:write-attribute(name=protocol,value=openssl.TLSv1.2)

reload
```

  使用可能な OpenSSL プロトコルは次のとおりです。

  - openssl.TLS
  - openssl.TLSv1
  - openssl.TLSv1.1
  - openssl.TLSv1.2

JBoss EAP は自動的にシステム上の OpenSSL ライブラリーの検索を行い、それを使用します。JBoss EAP の起動中に **org.wildfly.openssl.path** プロパティーを使用すると、カスタム OpenSSL ライブラリーの場所を指定することもできます。JBoss Core Services によって提供される OpenSSL ライブラリーのバージョン 1.0.2 以上のみがサポートされます。

OpenSSL が適切にロードされると、JBoss EAP の起動中に以下と似たメッセージが **server.log** に出力されます。

```
15:37:59,814 INFO [org.wildfly.openssl.SSL] (MSC service thread 1-7) WFOPENSSL0002 OpenSSL Version OpenSSL 1.0.2k-fips 23 Mar 2017
```

A.41. JAVA 8 向けに提供されるプラットフォームモジュール

- **java.base**: この依存関係は提供されるモジュールローダーに常に含まれます。
- **java.compiler**
- **java.datatransfer**
- **java.desktop**
- **java.instrument**
- **java.jnlp**
- **java.logging**
- **java.management**
- **java.management.rmi**
- **java.naming**
- **java.prefs**
- **java.rmi**
- **java.scripting**

- **java.se**: このモジュールエイリアスは以下のベースモジュールのセットを集約します。
  - **java.compiler**
  - **java.datatransfer**
  - **java.desktop**
  - **java.instrument**
  - **java.logging**
  - **java.management**
  - **java.management.rmi**
  - **java.naming**
  - **java.prefs**
  - **java.rmi**
  - **java.scripting**
  - **java.security.jgss**
  - **java.security.sasl**
  - **java.sql**
  - **java.sql.rowset**
  - **java.xml**
  - **java.xml.crypto**
- **java.security.jgss**
- **java.security.sasl**
- **java.smartcardio**
- **java.sql**
- **java.sql.rowset**
- **java.xml**
- **java.xml.crypto**
- **javafx.base**
- **javafx.controls**
- **javafx.fxml**
- **javafx.graphics**

- **javafx.media**
- **javafx.swing**
- **javafx.web**
- **jdk.accessibility**
- **jdk.attach**
- **jdk.compiler**
- **jdk.httpserver**
- **jdk.jartool**
- **jdk.javadoc**
- **jdk.jconsole**
- **jdk.jdi**
- **jdk.jfr**
- **jdk.jsobject**
- **jdk.management**
- **jdk.management.cmm**
- **jdk.management.jfr**
- **jdk.management.resource**
- **jdk.net**
- **jdk.plugin.dom**
- **jdk.scripting.nashorn**
- **jdk.sctp**
- **jdk.security.auth**
- **jdk.security.jgss**
- **jdk.unsupported**
- **jdk.xml.dom**

A.42. 検証タイミング方法の比較

validate-on-match と **background-validation** メソッドのさまざまな側面を比較して、データベース接続の検証の設定に適した方法を判断できます。

次の表には、検証タイミング方法の比較マトリックスが含まれています。

表A.144 検証タイミング方法の比較マトリックス

比較の側面	Validate-on-match メソッド	Background-validation メソッド
信頼性	**validate-on-match** メソッドは、各データベース接続を使用する直前に検証します。つまり、この検証では、アプリケーションで使用するためにプールからチェックアウトされた接続がテストされます。	**background-validation** メソッドは、定期的なバックグラウンド検証をしてから、検証された接続を使用するまでの間に接続が失敗する可能性があるので、信頼性が低くなります。 バックグラウンド検証メソッドが頻繁に実行される場合に、検証が行われるのは、アプリケーションで使用するように予約されていないプールでの接続のみを対象とします。つまり、こちらの検証も、使用するためにプールからチェックアウトされた接続がテストされます。
システムの使用、ネットワークのパフォーマンス、および接続の問題のタイミングと範囲に依存するパフォーマンス	長時間アイドル状態のままのシステムのユーザーは、**validate-on-match** を使用して接続を要求するときに、短いまたは長い遅延が発生する可能性が高くなります。 JDBC 4 検証メカニズムなど、より効率的な検証メカニズムを備えたシステムのユーザーは、**validate-on-match** を使用すると遅延が少ないことに気付くかもしれません。これは、システムがほとんどアイドル状態でなく、接続がタイムアウトする可能性が低い場合に当てはまります。 プール内のほとんどまたはすべての接続に影響を与える広範なサービス停止の後、**validate-on-match** が設定されたデータソースのユーザーは、接続を取得する際に遅延が発生する可能性が高くなります。これは、ユーザーが接続を待機しているときに、切断された接続が繰り返し検証され、削除されるためです。	長時間アイドル状態のままのシステムのユーザーは、**background-validation** を使用して接続を要求するときに、短いまたは長い遅延が発生する可能性が低くなります。 JDBC 4 検証メカニズムなど、より効率的な検証メカニズムを備えたシステムのユーザーは、**background-validation** を使用すると遅延が少ないことに気付くかもしれません。これは、システムがほとんどアイドル状態でなく、接続がタイムアウトする可能性が低い場合に当てはまります。 プール内のほとんどまたはすべての接続に影響を与える広範な停止の後、**background-validation** が設定されたデータソースのユーザーは、接続が繰り返し切断され、複数回再試行が行われる可能性が高くなります。

耐障害性のコーディング	アプリケーションによって接続がプールから取得された後でも、任意のタイミングで外部から接続を終了できるため、障害が発生した場合でも、**validate-on-match** を使用する場合のアプリケーションロジックは変わりません。 **validate-on-match** を使用すると、切断された接続が発生する可能性が低くなります。これは、**validate-on-match** が使用前に接続の即時検証を実行するためです。	アプリケーションによって接続がプールから取得された後でも、任意のタイミングで外部から接続を終了できるため、障害が発生した場合でも、**background-validation** を使用する場合のアプリケーションロジックは変わりません。 **background validation** を使用すると、接続が切断される可能性が高くなります。

改訂日時: 2024-02-09