Red Hat JBoss Enterprise Application Platform 7.0
設定ガイド

Red Hat JBoss Enterprise Application Platform 7.0 向け

Red Hat Customer Content Services

Red Hat JBoss Enterprise Application Platform 7.0 設定ガイド

Red Hat JBoss Enterprise Application Platform 7.0 向け

法律上の通知

Copyright © 2017 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書は、Red Hat JBoss Enterprise Application Platform 7.0 の設定に関する管理者向けの実用ガイドです。

目次

第1章 概要 **6**

第2章 JBOSS EAP の開始および停止 **7**
2.1. JBOSS EAP の開始 7
2.2. JBOSS EAP の停止 8
2.3. JBOSS EAP の管理専用モードでの実行 8
2.4. JBOSS EAP の正常な中断およびシャットダウン 10
2.5. JBOSS EAP の起動および停止 (RPM インストール) 12
2.6. POWERSHELL スクリプト 15

第3章 JBOSS EAP の管理 **16**
3.1. サブシステム、拡張、およびプロファイル 16
3.2. 管理ユーザー 16
3.3. 管理インターフェース 19
3.4. 設定データ 22
3.5. ファイルシステムパス 29
3.6. システムプロパティー 34
3.7. 管理監査ロギング 36

第4章 ネットワークおよびポート設定 **41**
4.1. インターフェース 41
4.2. ソケットバインディング 43
4.3. IPV6 アドレス 47

第5章 JBOSS EAP のセキュリティー **49**

第6章 JBOSS EAP クラスローディング **50**
6.1. モジュール 50
6.2. モジュールの依存性 51
6.3. カスタムモジュールの作成 52
6.4. グローバルモジュールの定義 54
6.5. サブデプロイメント分離の設定 55
6.6. 外部 JBOSS EAP モジュールディレクトリーの定義 55
6.7. 動的モジュールの命名規則 56

第7章 アプリケーションのデプロイ **57**
7.1. 管理 CLI を使用したアプリケーションのデプロイ 57
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 59
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 62
7.4. MAVEN を使用したアプリケーションのデプロイ 64
7.5. HTTP API を使用したアプリケーションのデプロイ 68
7.6. デプロイメントの動作のカスタマイズ 68

第8章 ドメイン管理 **75**
8.1. 管理対象ドメイン 75
8.2. ドメイン設定のナビゲート 78
8.3. 管理対象ドメインの起動 79
8.4. サーバーの管理 83
8.5. 管理対象ドメインの設定 87
8.6. JBOSS EAP プロファイルの管理 89

第9章 JVM の設定 **91**
9.1. スタンドアロンサーバーの JVM 設定 91
9.2. 管理対象ドメインの JVM 設定 91

9.3. JVM 状態の表示 93
9.4. 32 または 64 ビット JVM アーキテクチャーの指定 93

第10章 MAIL サブシステム 96
10.1. MAIL サブシステムの設定 96
10.2. カスタムトランスポートの設定 96

第11章 WEB サービスの設定 99

第12章 JBOSS EAP を用いたロギング 100
12.1. サーバーロギング 100
12.2. ログファイルの表示 103
12.3. LOGGING サブシステム 105
12.4. ログカテゴリーの設定 113
12.5. ログハンドラーの設定 115
12.6. ルートロガーの設定 133
12.7. ログフォーマッターの設定 134
12.8. アプリケーションのロギング 137

第13章 データソース管理 142
13.1. JBOSS EAP データソース 142
13.2. JDBC ドライバー 142
13.3. データソースの作成 148
13.4. データソースの編集 151
13.5. データソースの削除 152
13.6. データソース接続のテスト 153
13.7. XA データソースのリカバリー 153
13.8. データベース接続の検証 157
13.9. データソースセキュリティー 159
13.10. データソースの統計 161
13.11. キャパシティーポリシー 162
13.12. エンリストメントトレース 165
13.13. データソース設定例 165

第14章 トランザクションの設定 190
14.1. TRANSACTIONS サブシステムの設定 190
14.2. トランザクション管理 193

第15章 ORB 設定 199
15.1. COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA) 199
15.2. JTS トランザクション用 ORB の設定 199

第16章 JAVA CONNECTOR ARCHITECTURE (JCA) の管理 201
16.1. JAVA CONNECTOR ARCHITECTURE (JCA) 201
16.2. リソースアダプター 201
16.3. JCA サブシステムの設定 201
16.4. リソースアダプターの設定 205
16.5. 管理接続プールの設定 208
16.6. 接続統計の表示 209

第17章 WEB サーバーの設定 (UNDERTOW) 210
17.1. UNDERTOW サブシステムの概要 210
17.2. バッファーキャッシュの設定 211
17.3. サーバーの設定 212
17.4. サーブレットコンテナーの設定 213
17.5. ハンドラーの設定 214

17.5. ハンドラーの設定 214
17.6. フィルターの設定 215
17.7. デフォルトの WELCOME WEB アプリケーションの設定 217
17.8. HTTPS の設定 219
17.9. HTTP セッションタイムアウトの設定 219
17.10. HTTP のみのセッション管理クッキーの設定 219
17.11. HTTP/2 の設定 220
17.12. REQUESTDUMPING ハンドラーの設定 222

第18章 リモーティングの設定 225
18.1. REMOTING サブシステム 225
18.2. エンドポイントの設定 226
18.3. コネクターの設定 226
18.4. HTTP コネクターの設定 227
18.5. アウトバウンド接続の設定 228
18.6. リモートアウトバウンド接続の設定 228
18.7. ローカルアウトバウンド接続の設定 229
18.8. リモーティングの追加設定 229

第19章 IO サブシステムの設定 231
19.1. IO サブシステムの概要 231
19.2. ワーカーの設定 231
19.3. バッファープールの設定 231

第20章 バッチアプリケーションの設定 233
20.1. BATCH ジョブの設定 233
20.2. バッチジョブの管理 236

第21章 NAMING サブシステムの設定 238
21.1. NAMING サブシステム 238
21.2. グローバルバインディングの設定 238
21.3. リモート JNDI インターフェースの設定 242

第22章 高可用性の設定 243
22.1. 高可用性 243
22.2. JGROUPS を用いたクラスター通信 244
22.3. INFINISPAN 254
22.4. JBOSS EAP をフロントエンドロードバランサーとして設定 265
22.5. 外部 WEB サーバーのプロキシーサーバーとしての使用 268
22.6. MOD_CLUSTER HTTP コネクター 272
22.7. APACHE MOD_JK HTTP コネクター 283
22.8. APACHE MOD_PROXY HTTP コネクター 286
22.9. MICROSOFT ISAPI コネクター 289
22.10. ORACLE NSAPI コネクター 296

付録A リファレンス資料 302
A.1. サーバーランタイム引数 302
A.2. RPM サービス設定ファイル 305
A.3. RPM サービス設定プロパティー 306
A.4. JBOSS EAP サブシステムの概要 308
A.5. ADD-USER ユーティリティー引数 312
A.6. 管理監査ロギング属性 313
A.7. インターフェース属性 316
A.8. ソケットバインディング属性 318
A.9. デフォルトのソケットバインディング 319
A.10. デプロイメントスキャナーマーカーファイル 321

A.10. デプロイメントスキャナーマーカーファイル 321
A.11. デプロイメントスキャナーの属性 322
A.12. ルートロガーの属性 324
A.13. ログカテゴリーの属性 324
A.14. ログハンドラーの属性 325
A.15. データソース接続 URL 334
A.16. データソースのパラメーター 335
A.17. データソースの統計 343
A.18. トランザクションマネージャーの設定オプション 347
A.19. リソースアダプターの属性 351
A.20. リソースアダプターの統計 356
A.21. UNDERTOW サブシステムの属性 358
A.22. HTTP メソッドのデフォルトの動作。 391
A.23. IO サブシステムの属性 392
A.24. REMOTING サブシステムの属性 393
A.25. APACHE HTTP SERVER の MOD_CLUSTER ディレクティブ 401
A.26. MODCLUSTER サブシステムの属性 405
A.27. MOD_JK ワーカープロパティー 410
A.28. SECURITY MANAGER サブシステム 412

第1章 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。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、**インストールガイド**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、スタンドアロンサーバーと管理対象ドメインの 2 つの操作モードの 1 つで実行され、Red Hat Enterprise Linux、Windows Server、Oracle Solaris および Hewlett-Packard HP-UX でサポートされます。

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。

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。このファイルの設定はカスタマイズすることができます。

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引数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 **domain.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。このファイルの設定はカスタマイズすることができます。

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。利用できる管理対象ドメイン設定ファイルとそれらの使用方法については、管理対象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引数 の項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

管理対象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管理対象ドメインの両方で利用できます。管理対象ドメインの場合、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。

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 **--admin-only** ランタイムスイッチを使用します。

注記

ここで使用する管理 CLI コマンドは、JBoss EAP スタンドアロンサーバーの実行を仮定しています。JBoss EAP 管理対象ドメインの管理 CLI を使用する場合の詳細は JBoss EAP **Management CLI Guide** を参照してください。

JBoss EAP の管理専用モードでの起動

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

-

JBoss EAP が管理専用モードで実行されていることを確認

JBoss EAP インスタンスが管理専用モードで実行されているかを確認するには、以下を実行します。

注記

ここで使用する管理 CLI コマンドは、JBoss EAP スタンドアロンサーバーの実行を仮定しています。JBoss EAP 管理対象ドメインの管理 CLI を使用する場合の詳細は JBoss EAP **Management CLI Guide** を参照してください。

```
:read-attribute(name=running-mode)
```

JBoss EAP インスタンスが管理専用モードで実行されていると、結果は次のようになります。

```
{
  "outcome" => "success",
  "result" => "ADMIN_ONLY"
}
```

管理専用モードで実行されていない場合、結果は次のようになります。

```
{
  "outcome" => "success",
  "result" => "NORMAL"
}
```

管理 CLI からの別モードでの再起動

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だけでなく、管理 CLI を使用してサーバーをリロードして異なるモードで起動することもできます。JBoss EAP インスタンスをリロードして管理専用モードで起動するには以下を実行します。

注記

ここで使用する管理 CLI コマンドは、JBoss EAP スタンドアロンサーバーの実行を仮定しています。JBoss EAP 管理対象ドメインの管理 CLI を使用する場合の詳細は JBoss EAP **Management CLI Guide** を参照してください。

```
reload --admin-only=true
```

JBoss EAP インスタンスをリロードして通常モードで起動するには以下を実行します。

```
reload --admin-only=false
```

注記

現在の実行モードではなく、最初の実行モードをチェックするには、コマンド **`/core-service=server-environment:read-attribute(name=initial-running-mode)`** を実行します。このコマンドは **`:read-attribute(name=running-mode)`** とは異なり、JBoss EAP が起動されたときの実行モードを表示するため、現在の実行モードは表示されません。

2.4. JBOSS EAP の正常な中断およびシャットダウン

JBoss EAP は正常に中断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、中断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機する期間を指定します。サーバーが中断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中断することをロードバランサーに通知します。

EJB

`ejb3` サブシステムはすべてのリモート EJB 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。EJB 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。

EE 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、EE 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。

サーバーが中断状態である間、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、正常なシャットダウンはインバウンドのリモート分散トランザクションや新しいインバウンド JMS 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 EE バッチジョブおよび EE 同時実行タスクは、現在実行の継続が許可されます。しかし、タイムアウトウィンドウを渡す EE 同時実行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、中断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中断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. サーバーの中断

JBoss EAP 7 には、サーバーの操作を正常に中断する **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

管理インターフェースのサーバー中断に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中断および再開できます。

サーバーの中断状態のチェック

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)
```

- 管理対象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、サーバーはすべてのアクティブなリクエストが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、中断します。

- スタンドアロンサーバーを中断します。

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

- 管理対象ドメインのすべてのサーバーを中断します。

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

- 管理対象ドメインの単一のサーバーを中断します。

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

- サーバーグループのすべてのサーバーを中断します。

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

再開

resume コマンドを適切なレベル (サーバー、サーバーグループ、ドメイン全体) で使用すると、サーバーが正常な実行状態に戻り、新しいリクエストを許可できるようになります。

```
:resume
```

2.4.2. 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中断され、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、すべてのアクティブなリクエストが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(timeout=60)
```

- 管理対象ドメインのすべてのサーバーを停止します。

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

- 管理対象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(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(timeout=60)
```

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管理対象ドメイン) 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 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。

管理対象ドメインでの 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 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 管理対象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。

- 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。管理対象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

管理インターフェースを特定の 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。管理対象ドメインの場合、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 管理インターフェースを指定の 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管理対象ドメイン) 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 の場合

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

管理対象ドメインでの 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 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

PowerShell スクリプトは技術プレビューとしてのみ使用でき、サポートされません。JBoss EAP は Windows Server 2008 R2 Enterprisex86_64 および Windows Server 2012 R2 Standard x86_64 でサポートされるため、PowerShell スクリプトはバージョン 2 以降で正確に動作するように設計されています。

- デフォルトでは、Windows Server 2008 R2 Enterprisex86_64 は PowerShell バージョン 2 を使用します。これよりも新しいバージョンをインストールすることもできます。
- デフォルトでは、Windows Server 2012 R2 Standard x86_64 は PowerShell バージョン 4 を使用します。これよりも新しいバージョンをインストールすることもできます。

PowerShell スクリプトのパスは **`EAP_HOME/bin`** で、Windows でモダンなスクリプト言語を使用したいときにこのスクリプトを使用できます。

注記

スクリプトの引数は、以下のように引用符で囲む必要があります。

```
PS C:\Users\Administrator\7.0.0.ER2\jboss-eap-7.0\bin>
.\standalone.ps1 "-c=standalone-full.xml"
```

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管理対象ドメインごとに 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、管理対象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 または Web ベースの管理コンソールを使用して設定できます。これらの管理インターフェー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。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、拡張、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。

サブシステムは特定の拡張機能の設定オプションを提供します。**拡張** はサーバーのコア機能を拡張するモジュールです。拡張はデプロイメントが必要になるとロードされ、必要がなくなるとアンロードされます。

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構成されます。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。管理対象ドメインでは、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。

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。

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ースを好むユーザーは管理コンソールを使用し、コマンドラインインターフェースを好むユーザーは管理 CLI を使用するとよいでしょう。

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について説明します。このスクリプトは、既定の認証のプロパティーファイルに新規ユーザーを追加する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) などの高度な認証や承認のオプションについては、**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コマンドラインベース管理 CLI を使用して監理操作を実行することを許可されます。別のオプション (**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。

   デフォルトでは、弱い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'
```

このコマンドを実行すると、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、Add-User ユーティリティー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。

デフォルトでは、弱い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。

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は次のとおりです。

- 最小文字数
- アルファベットの最小文字数
- 数字の最小文字数
- シンボルの最大文字数
- 禁止されたパスワードのリスト (**admin** など)
- ユーザー名と一致するパスワードを許可するかどうか

3.3. 管理インターフェース

3.3.1. 管理 CLI

管理コマンドラインインターフェース (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。操作はバッチモードで実行でき、複数のタスクをグループとして実行できます。

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
```

以下のコマンドを実行して、特定コマンドのヘルプを表示します。

```
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
}
```

管理対象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

管理対象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
```

3.3.2. 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。

管理対象ドメインでは、同じドメイン内のサーバーインスタンスや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一元的に管理できます。

デフォルトの管理ポ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/App.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管理対象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。管理対象ドメインでは、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。管理対象ドメインでは、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。管理対象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。

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。

Patching (パッチ)

JBoss EAP インスタンスにパッチを適用します。

注記

更新された管理コンソールについて説明するツアーに参加するには、管理コンソールのホームページにある **Take a Tour** リンクをクリックします。

フォームフィールドの詳細を表示するには、**Need Help?** リンクをクリックします。

実行した設定アクションのメッセージ履歴を表示するには、管理コンソールの右上にある **Messages** リンクをクリックします。

3.3.2.1. 管理コンソールの有効化/無効化

管理コンソールを有効または無効にするには、**`/core-service=management/management-interface=http-interface`** リソースの **`console-enabled`** ブール値属性を設定します。ドメインモードのマスターホストの場合は、**`/host=master/core-service=management/management-interface=http-interface`** になります。

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)
```

3.3.2.2. 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. **Confirm** をクリックします。選択したロケールを使用するために Web ブラウザーが自動的に更新されます。

3.4. 設定データ

3.4.1. 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 4 つのプロファイル (**default**、**ha**、**full**、および **full-ha**) ごとに個別のファイルがあります。

表3.1 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。

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
```

3.4.2. 管理対象ドメイン設定ファイル

管理対象ドメイン設定ファイルは **EAP_HOME/domain/configuration/** ディレクトリーにあります。

表3.2 管理対象ドメイン設定ファイル

設定ファイル	目的
domain.xml	これは管理対象ドメインのメインの設定ファイルです。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の設定が含まれます (**default**、**ha**、**full**、**full-ha**)。

設定ファイル	目的
host.xml	このファイルには、管理対象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ース、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、サーバーを管理対象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 を管理対象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
```

3.4.3. 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、管理対象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。

3.4.4. 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/2015
1022-133109702standalone.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

管理対象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。

3.4.5. 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。

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。

```
/core-service=management/service=configuration-changes:add(max-
history=10)
```

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。

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

このコマンドは、各設定変更とその変更日、結果、および操作の詳細を一覧で表示します。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。

3.4.6. プロパティーの置換

JBoss EAP では、式を使用して設定のリテラル値の代わりに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** インターフェースの **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。Java 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。有効になっていると、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**
- ***-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)
```

Java 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**
- **persistence.xml**
- **application.xml**
- **web.xml**

以下の管理 CLI コマンドを使用すると、Java 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)
```

3.5. 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3.3 標準的なパス

プロパティー	説明
java.ext.dirs	Java Development Kit 拡張ディレクトリーのパス。
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管理対象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。管理対象ドメインの **`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。
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	ドメインが管理対象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。

プロパティー	説明
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。

3.5.1. 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
```

- サーバー設定ファイル (**standalone.conf** または **domain.conf**) の **JAVA_OPTS** 変数を編集します。以下に例を示します。

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

管理対象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
```

3.5.2. 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)
  ```

- 管理コンソールでファイルシステムパスを設定するには **Configuration** タブで **Paths** を選択してパスを追加、編集、および削除します。

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:3.0">
  ...
  <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>
```

3.5.3. ディレクトリーのグループ化

管理対象ドメインでは、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>
```

3.6. 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ー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 の管理対象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)
```

例を以下に示します。

```
/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。

管理対象ドメインの場合、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、**View** ボタンをクリックします。
- 管理対象ドメインの場合
  - 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、**View** 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 **View** 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。

JAVA_OPTS を使用したシステムプロパティーの設定

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。管理対象ドメインの場合は、**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。

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。例を以下に示します。

```
...
# 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。例を以下に示します。

```
...
# 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
...
```

3.7. 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、管理インターフェー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。

管理対象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)
```

管理対象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。

3.7.1. 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管理対象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。

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設定 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。

管理対象ドメイン監査ロギングの有効化

管理対象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/standalone/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。

3.7.2. 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。管理対象ドメインでは、**/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=tr
uststore: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。

   管理対象ドメインでは、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。

3.7.3. 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**で参照することをお勧めします。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お勧めします

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3.4 管理監査ログフィールド

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ースから実行されます。 - **HTTP** - 操作がドメイン HTTP インターフェー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。

フィールド名	説明
version	JBoss EAP インスタンスのバージョン番号。

第4章 ネットワークおよびポート設定

4.1. インターフェース

JBoss EAP は設定全体で名前付きインターフェースを参照します。これにより、使用ごとにインターフェー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ー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ースの詳細が異なる場合に管理対象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ー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ース名があります。**management** インターフェースは、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** インターフェースは、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** インターフェースは、標準設定の IIOP ソケットに使用されます。**private** インターフェースは、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ース設定

```
<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ースを **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** インターフェースの **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ースまたはポートを変更する場合、変更したインターフェー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ースとポートを指定するようにしてください。

4.1.2. インターフェースの設定

ネットワークインターフェースは、物理インターフェー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ー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ース選択基準は インターフェース属性を参照してください。

インターフェ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。以下にインターフェー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ースの追加

NIC 値が **eth0** であるインターフェー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ースの追加

稼働時に適切なサブネットのすべてのインターフェー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ー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>
```

インターフェース属性の更新

public インターフェースのデフォルトの **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>
```

管理対象ドメインでインターフェースをサーバーに追加

```
/host=master/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ー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、管理対象ドメインには複数のグループを含むことができます。管理対象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。

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 4 つのプロファイル (**default**、**ha**、**full**、および **full-ha**) 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ースまたはポートを変更する場合、変更したインターフェー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ー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**、および **standalone-full-ha.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>
```

管理対象ドメイン

管理対象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 4 つあります。

- **standard-sockets**
- **ha-sockets**
- **full-sockets**
- **full-ha-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="${jboss.default.multicast.address:230.0.0.4}"
multicast-port="45700"/>
        <socket-binding name="jgroups-tcp" interface="private"
port="7600"/>
        <socket-binding name="jgroups-tcp-fd" interface="private"
port="57600"/>
        <socket-binding name="jgroups-udp" interface="private"
port="55200" multicast-
address="${jboss.default.multicast.address:230.0.0.4}" multicast-
port="45688"/>
        <socket-binding name="jgroups-udp-fd" interface="private"
port="54200"/>
        <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-groups>
```

注記

管理インターフェー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。これは、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ースを使用するよう、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. 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 を使用して管理対象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、管理対象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。
   - 管理対象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ース宣言の更新

設定のデフォルトのインターフェース値は、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** インターフェースを 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ー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 自体をセキュアにするための情報は、**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をセキュアにするための情報は、**How to Configure Identity Management** を参照してください。
- Kerberos 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 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ド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、Java 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。

MySQL JDBC ドライバー **`module.xml`** ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.36-bin.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

モジュール名 (**`com.mysql`**) は、モジュールのディレクトリー構造 (スロット名 (**`main`**) を除く) に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。

各モジュールに 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。

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から派生されます。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。

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ド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。

デプロイされたアプリケーション (JAR や WAR など) 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 Java EE 7 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。エクスポートされた依存関係は、エクスポートするモジュールに依存するモジュールへ提供されます。

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-5.1.36-bin.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-5.1.36-bin.jar"/>
  </resources>
  <dependencies>
    <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するには、以下の手順に従います。

注記

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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

1. JBoss EAP サーバーを起動します。
2. 管理 CLI を起動しますが、稼働しているインスタンスへの接続に **--connect** または **-c** 引数を使用しないでください。

```
$ 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
```

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。

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、**開発ガイド**のデプロイメントへの明示的なモジュール依存関係の追加 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:write-attribute(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.prop
erties");
```

6.4. 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されるモジュールの名前を知っている必要があります。デプロイメントのモジュールの命名慣例については、動的モジュールの命名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}]
```

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。

6.5. 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。

6.6. 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。
   - 管理対象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。

6.7. 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ースや管理 CLI のコマンドラインインターフェー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 JBoss 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ー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、管理対象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. 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 **`deploy`** 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。

```
deploy /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")
```

アプリケーションが正常にデプロイされます。

アプリケーションのアンデプロイ

管理 CLI で **`undeploy`** コマンドを使用し、デプロイメント名を指定します。

- アプリケーションをアンデプロイし、デプロイメントの内容を削除します。

```
undeploy test-application.war
```

- リポジトリーからデプロイメントの内容を削除せずにアプリケーションをアンデプロイします。

```
undeploy test-application.war --keep-content
```

これは、管理コンソールからデプロイメントを無効にすることと同じです。

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")
```

アプリケーションが正常にアンデプロイされます。

デプロイメントの一覧表示

管理 CLI で **deploy -l** コマンドを使用し、すべてのデプロイメントを一覧表示します。

```
deploy -l
```

出力には、ランタイム名や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            ENABLED STATUS
test-application.war    test-application.war    true    OK
jboss-helloworld.war    jboss-helloworld.war    true    OK
```

7.1.2. 管理対象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 **deploy** 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があります。

- 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

```
deploy /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

```
deploy /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")
```

アプリケーションが管理対象ドメインの適切なサーバーグループに正常にデプロイされます。

アプリケーションのアンデプロイ

管理 CLI で **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があります。

- 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。

  ```
  undeploy test-application.war --all-relevant-server-groups
  ```

- 特定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。そのデプロイメントの他のサーバーグループに対してデプロイメントの内容をリポジトリーに保持する必要があるため、**--keep-content** パラメーターが必要になります。

  ```
  undeploy test-application.war --server-groups=other-server-group --
  keep-content
  ```

  これは、管理コンソールからデプロイメントを無効にすることと同じです。

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")
```

アプリケーションが正常にアンデプロイされます。

デプロイメントの一覧表示

管理 CLI で **deploy -l** コマンドを使用し、すべてのデプロイメントを一覧表示します。

```
deploy -l
```

出力には各デプロイメントとそのランタイム名が表示されます。

```
NAME                    RUNTIME-NAME
test-application.war    test-application.war
jboss-helloworld.war    jboss-helloworld.war
```

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ー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. 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、**新規 Deployment** ウィザードを使用してアプリケーションをデプロイします。**新規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っています。

- 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。

- 未管理のデプロイメント作成

  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。展開形式のデプロイメントは未管理としてのみサポートされることに注意してください。

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、**削除**オプションを選択してアプリケーションをアンデプロイします。これにより、デプロイメントがアンデプロイされ、コンテンツリポジトリーから削除されます。

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. 管理対象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割り当てることができます。

- 未割り当てのコンテンツ

  サーバーグループに割り当てられていないデプロイメントは**未割り当てのコンテンツ** セクションにリストされます。ここで、デプロイメントをサーバーグループに追加したり、削除したり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割り当てら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デプロイ

1. **Content Repository** で **追加**ボタンをクリックします。
2. **新規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デプロイします。
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割り当てる必要があります。

また、**Server Groups** でデプロイメントを追加すると、デプロイメントの追加、サーバーグループへの割り当て、および有効化を同時に行うことができます。

アプリケーションのサーバーグループへの割り当て

1. **未割り当てのコンテンツ** でデプロイメントを選択し、**割り当て**ボタンをクリックします。
2. このデプロイメントを割り当て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割り当ての解除

1. **Server Groups** で適切なサーバーグループを選択します。
2. サーバーグループの割り当てを解除するデプロイメントを選択し、 **割り当ての解除** をクリックします。

また、**Content Repository** でデプロイメントの **割り当ての解除** を選択すると、複数のサーバーグループへの割り当てを同時に解除することができます。

アプリケーションのアンデプロイ

1. それでもデプロイメントがサーバーグループに割り当てられている場合は、デプロイメントの割り当てを解除します。
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、デプロイメントがアンデプロイされ、コンテンツリポジトリーから削除されます。

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、**完了** 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. 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、および展開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、展開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。

展開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展開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. 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 **スタートガイド**のクイックスタートサンプルの使用 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 "jboss-helloworld.war" (runtime-
name: "jboss-helloworld.war")
WFLYUT0021: Registered web context: /jboss-helloworld
WFLYSRV0010: Deployed "jboss-helloworld.war" (runtime-name : "jboss-
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: /jboss-helloworld
WFLYSRV0028: Stopped deployment jboss-helloworld.war (runtime-name:
jboss-helloworld.war) in 27ms
WFLYSRV0009: Undeployed "jboss-helloworld.war" (runtime-name: "jboss-
helloworld.war")
```

7.4.2. 管理対象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 を使用して管理対象ドメインで 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 **スタートガイド**のクイックスタートサンプルの使用 を参照してください。

アプリケーションのデプロイ

管理対象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 "jboss-helloworld.war" (runtime-
name: "jboss-helloworld.war")
WFLYUT0021: Registered web context: /jboss-helloworld
WFLYSRV0010: Deployed "jboss-helloworld.war" (runtime-name : "jboss-
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: /jboss-helloworld
WFLYSRV0028: Stopped deployment jboss-helloworld.war (runtime-name:
jboss-helloworld.war) in 106ms
```

```
WFLYSRV0009: Undeployed "jboss-helloworld.war" (runtime-name: "jboss-
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。

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" -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}' -u USER:PASSWORD
```

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" -d '{"operation" : "composite", "address" : [],
"steps" : [{"operation" : "undeploy", "address" : {"deployment" : "test-
application.war"}},{"operation" : "remove", "address" : {"deployment" :
"test-application.war"}}],"json.pretty":1}' -u USER:PASSWORD
```

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。

7.6. 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。

管理対象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、管理対象ドメインのデプロイされたコンテンツは **EAP_HOME/domain/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 umlns="urn:jboss:1.0">
  <jboss-deployment-dependencies xmlns="urn:jboss:deployment-
dependencies:1.0">
    <dependency name="framework.ear" />
  </jboss-deployment-dependencies>
</jboss>
```

これにより、**app.ear** の前に **framework.ear** がデプロイされます。

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内容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、JAR ファイル、クラス、 JSP 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。

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。

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。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。

パラメーター

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カンマ区切りリスト。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カンマ区切りリスト。

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。

7.6.4. 操作ヘッダーの指定

管理 CLI を使用してアプリケーションをデプロイする場合、操作ヘッダーを指定するとデプロイメント操作実行の特定の側面を制御することができます。**--headers** 引数を使用すると以下の操作ヘッダーを **deploy** コマンドに渡すことができます。

- **allow-resource-service-restart** - 操作の変更を有効にするために再起動が必要なランタイムサービスを再起動するかどうか。デフォルトは **false** です。
- **blocking-timeout** - 完了プロセスのいずれかの時点で操作がブロックする最大期間 (秒単位)。この期間を超えると操作がロールバックされます。デフォルトは **300** 秒です。

- **rollback-on-runtime-failure** - 変更をランタイムサービスに適用できなかった場合に永続設定の変更を元に戻すかどうか。デフォルトは **true** です。
- **rollout** - 管理対象ドメインデプロイメントのロールアウト計画。ロールアウト計画の使用 の項を参照してください。

例を以下に示します。

```
deploy /path/to/deployment.war --headers={allow-resource-service-
restart=true}
```

セミコロン(;) を使用して複数の操作ヘッダーを区切ります。

7.6.5. ロールアウト計画の使用

ロールアウト計画

管理対象ドメインでは、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 台のサーバー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の最大数で、失敗したサーバーの台数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-failure-percentage**: **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)}
```

保存され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。

第8章 ドメイン管理

本項では、管理対象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。

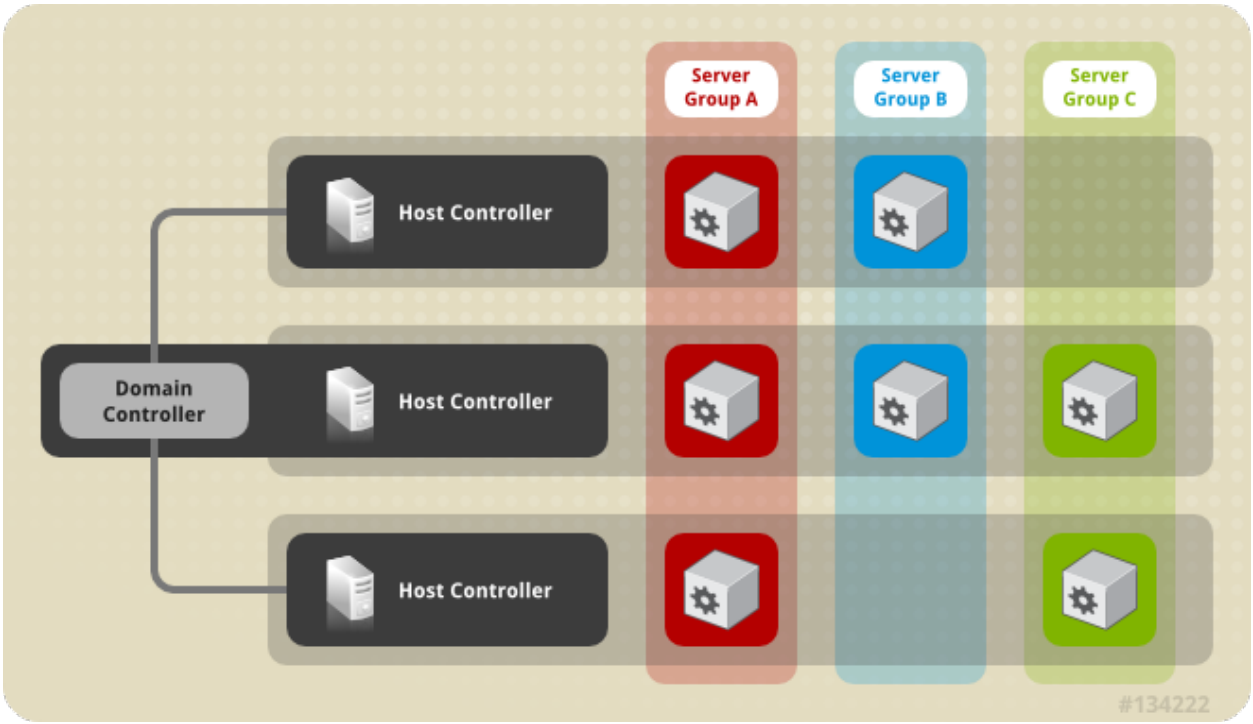
管理対象ドメインのセキュア化については、JBoss EAP **How to Configure Server Security** の Securing a Managed Domain の項を参照してください。

8.1. 管理対象ドメイン

管理対象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構成されます。



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役割は次のとおりです。

- 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

JBoss EAP 7 ドメインコントローラーは、JBoss EAP 6.2 以上を実行している JBoss EAP 6 ホストおよびサーバーを管理できます。

詳細は 管理対象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役割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 **`EAP_HOME/domain/configuration/host.xml`** ファイルから設定を読み取ります。このファイルは、ホストのファイルシステム上にある未展開の JBoss EAP インストールファイルに存在します。**`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ースの定義は **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 管理対象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。管理対象ドメインでは、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内容を共有します。

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。

ドメインは複数のサーバーグループで構成されます。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アップデート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。管理対象ドメインでは、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、規模の小さい管理対象ドメインと規模の大きい管理対象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ースを提供します。

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、およびリー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。

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割り当てすることができます。コンテンツリポジトリーのデプロイメントや、未割り当てのデプロイメントをすべて表示でき、特定のサーバーグループに割り当てられたデプロイメントを表示することもできます。

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、管理対象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ースを提供します。

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
```

管理対象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、管理対象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、管理対象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. 管理対象ドメインの起動

8.3.1. 管理対象ドメインの起動

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引数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 台のマシンで管理対象ドメインを設定
- 2 台のマシンで管理対象ドメインを設定

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 台のマシンで管理対象ドメインを設定
- 2 台のマシンで管理対象ドメインを設定

8.3.2. ドメインコントローラーの設定

ドメインコントローラーとして動作するホストの設定

<domain-controller> 宣言に **<local/>** 要素が含まれると、そのホストがドメインコントローラーとして指名されます。

```
<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ースを公開する必要があります。HTTP(S) 管理インターフェースを公開する必要はありませんが、管理コンソールへのアクセスを可能にするため、HTTP(S) 管理インターフェースを公開することが推奨されます。

```
<management-interfaces>
  <native-interface security-realm="ManagementRealm">
    <socket interface="management"
port="${jboss.management.native.port:9999}"/>
  </native-interface>
  <http-interface security-realm="ManagementRealm" http-upgrade-
enabled="true">
    <socket interface="management"
port="${jboss.management.http.port:9990}"/>
  </http-interface>
</management-interfaces>
```

EAP_HOME/domain/configuration/host-master.xml ファイルには、ドメインコントローラーとして動作するための設定が事前設定されています。

8.3.3. ホストコントローラーの設定

ドメインコントローラーへの接続

ホストコントローラー自体をドメインに登録するため、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する方法を指定する必要があります。これは **<domain-controller>** 要素に設定されます。

```
<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:9999}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

EAP_HOME/domain/configuration/host-slave.xml ファイルには、ドメインコントローラーに接続するための設定が事前設定されています。ホストコントローラーの起動時に **jboss.domain.master.address** プロパティーを提供する必要があります。

```
$ 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 台のマシンで管理対象ドメインを設定を参照してください。

8.3.3.1. ホスト名の設定

管理対象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:4.0" 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:4.0">
```

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
```

8.3.4. 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

管理対象ドメインの設定時、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。

これにより、バックアップドメインコントローラーのコンタクト情報を用いてホストコントローラーを事前設定できます。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マスターに昇格でき、昇格後にホストコントローラーを新しいマスターへ自動的にフェイルオーバーできます。

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="${jboss.domain.master.port:9999}"/>
      <static-discovery name="backup"
protocol="${jboss.domain.master.protocol:remote}" host="172.16.81.101"
port="${jboss.domain.master.port:9999}"/>
    </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ェイルオーバーの状況で使用される可能性があります。

プライマリードメインコントローラーで問題が発生した場合、**--backup** オプションで開始された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することができます。

注記

--backup オプションでホストコントローラーを開始すると、コントローラーによってドメイン設定のローカルコピーが維持されます。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。

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=HOST_NAME:write-local-domain-controller
```

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。

8.4. 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="jboss-helloworld.war" runtime-name="jboss-
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 (これには、**remoting** サブシステムが存在する必要があります)。
- **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ース。
- **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ースのリスト。
- **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
```

8.5. 管理対象ドメインの設定

8.5.1. 1 台のマシンで管理対象ドメインを設定

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。

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.native.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ースと競合しないように管理インターフェースを変更する必要があります。このコマンドは **jboss.management.native.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。

8.5.2. 2 台のマシンで管理対象ドメインを設定

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 台のマシンで管理対象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 台のマシンで管理対象ドメインを作成

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 (**<secret value="SECRET_VALUE" />**) を書き留めておきます。この秘密の値はホストコントローラーの設定で使用します。

    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. ユーザークレデンシャル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:1.6" 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。

8.6. JBOSS EAP プロファイルの管理

8.6.1. 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構成されます。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 4 つ含まれています。

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。

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が、ほとんどの場合で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。

8.6.2. 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管理対象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。

8.6.3. 階層的なプロファイルの作成

管理対象ドメインでは、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。

管理対象ドメインは **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。

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 サーバーと管理対象ドメインの 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 **System Properties** 画面を使用して宣言することもできます。

管理対象ドメインでは、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 へ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** ファイルに JVM 設定を追加することもできます。

警告

JAVA_OPTS 環境変数を設定すると、**standalone.conf** からのデフォルト値がオーバーライドされるため、JBoss EAP の起動に問題が発生する可能性があります。

9.2. 管理対象ドメインの JVM 設定

JBoss EAP 管理対象ドメインでは、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"]
)
```

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** タブを選択してから **Hosts** を選択し、編集するホストの **JVM** をクリックします。

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、異なるヒープサイズを設定するには、以下のコマンドを使用します。

```
/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。

また、**Runtime** タブを選択してから **Server Groups** を選択し、変更するサーバーグループの **View** をクリックすると、JBoss EAP 管理コンソールでサーバーグループの JVM 設定を編集できます。

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")
```

また、**Runtime** タブを選択してから **Hosts** を選択し、関連するホストを指定すると、JBoss EAP 管理コンソールでサーバーの JVM 設定を編集できます。サーバーを編集するには、サーバーを選択し、**View** をクリックします。

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管理対象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 Results** 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** タブを選択した後、**Standalone Server** を選択します。**Monitor** の列で **JVM** を選択し、**View** をクリックします。

管理対象ドメインでの 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** タブを選択した後、表示するサーバーグループとサーバーを選択します。**Monitor** 列で **JVM** を選択し、 **View** をクリックします。

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. 32 または 64 ビット JVM アーキテクチャーの指定

Hewlett-Packard HP-UX や Solaris などの一部の環境では、32 ビットまたは 64 ビット JVM の実行を指定するために **-d32** または **-d64** スイッチが使用されます。指定がない場合はデフォルトで 32 ビットが使用されます。

スタンドアロンサーバーでの 64 ビットアーキテクチャーの指定

1. **EAP_HOME/bin/standalone.conf** を開きます。
2. 以下の行を追加して、**-d64** オプションを **JAVA_OPTS** に追加します。

```
JAVA_OPTS="$JAVA_OPTS -d64"
```

管理対象ドメインでの 64 ビットアーキテクチャーの指定

管理対象ドメインを実行している場合、サーバーインスタンスの他にホストとプロセスコントローラーの 64 ビット環境を指定できます。

1. 64 ビット JVM で実行するよう、ホストおよびプロセスコントローラーを設定します。
   a. **EAP_HOME/bin/domain.conf** を開きます。
   b. 以下の行を追加して、**-d64** オプションを **JAVA_OPTS** に追加します。必ず **PROCESS_CONTROLLER_JAVA_OPTS** および **HOST_CONTROLLER_JAVA_OPTS** の設定部分の前に挿入してください。

   ```
   JAVA_OPTS="$JAVA_OPTS -d64"
   ```

   domain.conf の **JVM** オプションの例

   ```
   #
   # Specify options to pass to the Java VM.
   #
   if [ "x$JAVA_OPTS" = "x" ]; then
     JAVA_OPTS="-Xms64m -Xmx512m -XX:MaxPermSize=256m -
   Djava.net.preferIPv4Stack=true"
     JAVA_OPTS="$JAVA_OPTS -
   Djboss.modules.system.pkgs=$JBOSS_MODULES_SYSTEM_PKGS -
   Djava.awt.headless=true"
     JAVA_OPTS="$JAVA_OPTS -Djboss.modules.policy-
   permissions=true"
     JAVA_OPTS="$JAVA_OPTS -d64"
   else
     echo "JAVA_OPTS already set in environment; overriding
   default settings with values: $JAVA_OPTS"
   fi
   ```

2. 64 ビット JVM で実行するよう、サーバーインスタンスを設定します。

   適切な JVM 設定で **-d64** を JVM オプションとして追加します。以下のコマンドは、**default** の JVM 設定に追加されたことを表示します。

   ```
   /host=HOST_NAME/jvm=default:add-jvm-option(jvm-option="-d64")
   ```

-

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、Java EE 7 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;
```

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 JavaMail 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。例を以下に示します。

```
/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:2.0">
```

```
   <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>
```

第11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12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。

12.1. サーバーロギング

12.1.1. 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**
- 管理対象ドメインの場合: **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。

12.1.2. 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。この設定ファイルは JBoss EAP の **logging** サブシステムが起動し、引き継ぐ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**
- 管理対象ドメインの場合:

  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。

12.1.2.1. 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.ja
va:142)
        at
org.jboss.msc.service.ServiceControllerImpl$StartTask.startService(Serv
iceControllerImpl.java:1948)
        at
org.jboss.msc.service.ServiceControllerImpl$StartTask.run(ServiceContro
llerImpl.java:1881)
        at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.ja
va:1142)
        at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.j
ava: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"}
        }
        ...
    ]
}
```

12.1.3. 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デフォルトの場所は **`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。

```
$ export GC_LOG=false
$ EAP_HOME/bin/standalone.sh
```

12.1.4. 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、周期的な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12.1 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。
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12.2 管理対象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。

12.2. 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が、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ー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。

管理インターフェースの 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。

- **Runtime** タブを選択します。
- **Standalone Server** を選択します。管理対象ドメインで実行している場合は適切なサーバーを選択します。
- **Log Files** を選択し、**View** をクリックします。

リストからログファイルを選択すると、管理コンソールで直接ログの内容を表示および検索できます。ログファイルをローカルファイルシステムへダウンロードすることもできます。

> **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**
>
> 管理ドメインの場合では、このコマンドの前に **/host=HOST_NAME/server=SERVER_NAME** を追加します。

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"
    ]
}
```

12.3. 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。

12.3.1. 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。

12.3.2. 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他の名前を使用できます。

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。

12.3.3. 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。

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。

12.3.4. 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	値	説明
ALL	Integer.MIN_VALUE	すべてのログメッセージを提供します。

ログのレベル	値	説明
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。

ログのレベル	値	説明
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。

12.3.5. ログフォーマッター

ログフォーマッターは、そのハンドラーからのログメッセージの表示を定義します。**`java.util.logging.Formatter`** クラスを基にした構文を使用する文字列です。

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、以下のような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
```

ログフォーマッターの設定に関する詳細は 名前付きパターンフォーマッターの設定 または カスタムログフォーマッターの設定を参照してください。

ログフォーマッター文字列で使用される構文は以下の表を参照してください。

ログフォーマッター構文

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)。

記号	説明
%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。

記号	説明
%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)。

12.3.6. 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12.3 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]	フィルター式のカ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]	フィルター式のカ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"))**

フィルター式	説明
levelChange[level]	指定のレベルでログレコードを更新します。以下の例を示します。 **levelChange(WARN)**
levels[levels]	レベルのカ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。

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\")")
```

12.3.7. 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 **開発ガイド**の暗黙的なモジュール依存関係 の項を参照してください。

12.4. ログカテゴリーの設定

ここでは、管理 CLI を使用したログカテゴリーの設定方法を説明します。管理コンソールを使用してログカテゴリーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択し、 **Log Categories** タブを選択します。

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、管理対象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
```

12.5. ログハンドラーの設定

ログハンドラーはキャプチャーされたログメッセージが記録される方法を定義します。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
- カスタムログハンドラー
- Async ログハンドラー

12.5.1. Console ログハンドラーの設定

ここでは、管理 CLI を使用した Console ログハンドラーの設定方法を説明します。管理コンソールを使用して Console ログハンドラーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択します。**Handler** タブを選択して、左側のメニューから **Console** を選択します。

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、管理対象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。

  ハンドラーのフォーマッター文字列を設定します。たとえば、デフォルトのフォーマット文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。必ず **FORMAT** の値を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。デフォルトの値は **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
```

12.5.2. File ログハンドラーの設定

ここでは、管理 CLI を使用した File ログハンドラーの設定方法を説明します。管理コンソールを使用して File ログハンドラーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択します。**Handler** タブを選択して、左側のメニューから **File** を選択します。

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、管理対象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** 属性で構成される **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。

  ハンドラーのフォーマッター文字列を設定します。たとえば、デフォルトのフォーマット文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。必ず **FORMAT** の値を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。デフォルトの値は **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
```

12.5.3. Periodic Rotating ログハンドラーの設定

ここでは、管理 CLI を使用した Periodic Rotating ログハンドラーの設定方法を説明します。管理コンソールを使用して Periodic ログハンドラーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択します。**Handler** タブを選択して、左側のメニューから **Periodic** を選択します。

周期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、管理対象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** 属性で構成される **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。

  ハンドラーのフォーマッター文字列を設定します。たとえば、デフォルトのフォーマット文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。必ず **FORMAT** の値を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。デフォルトの値は **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
```

12.5.4. Size Rotating ログハンドラーの設定

ここでは、管理 CLI を使用した Size Rotating ログハンドラーの設定方法を説明します。管理コンソールを使用して Size ログハンドラーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択します。**Handler** タブを選択して、左側のメニューから **Size** を選択します。

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、管理対象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** 属性で構成される **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。

  ハンドラーのフォーマッター文字列を設定します。たとえば、デフォルトのフォーマット文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。必ず **FORMAT** の値を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。デフォルトの値は **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
```

12.5.5. Periodic Size Rotating ログハンドラーの設定

ここでは、管理 CLI を使用した Periodic Size ログハンドラーの設定方法を説明します。管理コンソールを使用して Periodic Size ログハンドラーを設定することもできます。管理コンソールを使用する場合は **Logging** サブシステムに移動し、**Handler** タブを選択して、左側のメニューから **Periodic Size** を選択します。

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、管理対象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** 属性で構成される **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。

ハンドラーのフォーマッター文字列を設定します。たとえば、デフォルトのフォーマット文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。必ず **FORMAT** の値を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。デフォルトの値は **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
```

12.5.6. Syslog ハンドラーの設定

ここでは、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へのメッセージの送信に使用される、管理 CLI を使用した Syslog ハンドラーの設定方法を説明します。管理コンソールを使用して Syslog ハンドラーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択します。**Handler** タブを選択して、左側のメニューから **Syslog** を選択します。

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、管理対象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 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
```

12.5.7. カスタムログハンドラーの設定

ここでは、管理 CLI を使用したカスタムログハンドラーの設定方法を説明します。管理コンソールを使用してカスタムログハンドラーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択します。**Handler** タブを選択して、左側のメニューから **Custom** を選択します。

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、管理対象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。

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)
```

カスタム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、カスタムログハンドラー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。

  ハンドラーのフォーマッター文字列を設定します。たとえば、デフォルトのフォーマット文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。必ず **FORMAT** の値を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。

カスタム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カスタムログハンドラーを割り当てます。

```
/subsystem=logging/logger=CATEGORY:add-
handler(name=CUSTOM_HANDLER_NAME)
```

カスタム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
```

12.5.8. Async ログハンドラーの設定

ここでは、管理 CLI を使用した Async ログハンドラーの設定方法を説明します。管理コンソールを使用して Async ログハンドラーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択します。**Handler** タブを選択して、左側のメニューから **Async** を選択します。

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、管理対象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
```

12.6. 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 を使用したルートロガーの設定方法を説明します。管理コンソールを使用してルートロガーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択し、 **Root Logger** タブを選択します。

ルートロガーの設定

重要

ロギングプロファイルに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、管理対象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、 ルートロガー属性 を参照してください。

12.7. ログフォーマッターの設定

ログフォーマッターはそのハンドラーからログメッセージの表示を定義します。 名前付きパターンフォーマッターまたはカスタムログフォーマッターを設定できます。

12.7.1. 名前付き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。

ここでは、管理 CLI を使用したログフォーマッターの設定方法を説明します。管理コンソールを使用してログフォーマッターを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択します。**Formatter** タブを選択して、左側のメニューから **Pattern** を選択します。

重要

ロギングプロファイルに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、管理対象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。

名前付きフォーマッターの作成

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、ログフォーマッターを参照してください。

```
/subsystem=logging/pattern-
formatter=PATTERN_FORMATTER_NAME:add(pattern=PATTERN_STRING)
```

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** のカ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")
```

名前付きフォーマッターのログハンドラーへの割り当て

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パターン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=PATTERN_FORMATTER_NAME)
```

12.7.2. カスタム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 を使用したカスタムログフォーマッターの設定方法を説明します。管理コンソールを使用してログフォーマッターを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Logging** サブシステムを選択します。**Formatter** タブを選択して、左側のメニューから **Custom** を選択します。

カスタム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、管理対象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>
```

12.8. 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 **開発ガイド**のロギングの章を参照してください。

12.8.1. 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 **開発ガイド**のデプロイメントごとのロギングをアプリケーションに追加を参照してください。

12.8.1.1. 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 **開発ガイド**のサブシステムをデプロイメントから除外を参照してください。

12.8.2. 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な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、影響を受けるアプリケーションを再デプロイせずに、ロギング設定を変更できます。ただし、ロギングプロファイルは管理コンソールを使用して設定できません。

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 (必須)
- 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。

12.8.2.1. 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はメインの **logging** サブシステムとほぼ同じ構文を使用します。メインの **logging** サブシステムとロギングプロファイルの設定には 2 つの違いがあります。

- ロート設定パスは **/subsystem=logging/logging-profile=NAME** です。
- 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。

ロギングプロファイルの作成および設定

以下の手順は、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。

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")
```

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)
```

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 **開発ガイド**のアプリケーションでのロギングプロファイルの指定を参照してください。

12.8.2.2. 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
```

12.8.3. デプロイメントロギング設定の表示

以下の管理 CLI コマンドを使用して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します。

```
/deployment=DEPLOYMENT_NAME/subsystem=logging/configuration=CONFIG:read
-resource
```

デプロイメントのロギング **configuration** の値には、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** 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)
```

第13章 データソース管理

13.1. 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 4.0 specification 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 でサポートされる構成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 管理インターフェー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。

13.2. 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。

13.2.1. コアモジュールとしての JDBC ドライバーのインストール

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展開してください。

2. JBoss EAP サーバーを起動します。

3. 管理 CLI を起動しますが、稼働しているインスタンスへの接続に **--connect** または **-c** 引数を使用しないでください。

   ```
   $ 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。

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

   たとえば、以下のコマンドは MySQL JDBC ドライバーモジュールを追加します。

   ```
   module add --name=com.mysql --resources=/path/to/mysql-connector-
   java-5.1.36-bin.jar --
   dependencies=javax.api,javax.transaction.api
   ```

   注記

   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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

   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。

5. **connect** 管理 CLI コマンドを使用して稼働中のインスタンスに接続します。

   ```
   connect
   ```

6. JDBC ドライバーを登録します。管理対象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.jdbc.Driver
- com.mysql.fabric.jdbc.FabricMySQLDriver

この場合、 **driver-class-name=com.mysql.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.jdbc.jdbc2.optional.MysqlXADataSource,
driver-class-name=com.mysql.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。

13.2.2. 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展開してください。

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

管理対象ドメインでは、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-5.1.36-bin.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-5.1.36-bin.jar_com.mysql.jdbc.Driver_5_1
```

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.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
```

13.2.3. 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13.1 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-306.ibm.com/software/data/db2/java/
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/

13.2.4. 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、高度な手順であり、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、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;
import org.jboss.jca.adapters.jdbc.WrappedConnection;

...

Connection c = ds.getConnection();
WrappedConnection wc = (WrappedConnection)c;
com.mysql.jdbc.Connection mc = wc.getUnderlyingConnection();
```

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。

13.3. 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。

13.3.1. 非 XA データソースの作成

非 XA データソースは **data-source add** 管理 CLI コマンドを使用して定義できます。管理コンソールを使用して非 XA データソースを定義することもできます。管理コンソールを使用する場合は **Configuration** → **Subsystems** → **Datasources** → **Non-XA** と選択し、**追加** をクリックして **Datasource の作成**ウィザードを開きます。

以下の手順で、管理 CLI を使用した非 XA データソースの定義方法を説明します。

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、**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
```

注記

管理対象ドメインでは、--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:jboss/datasources/ExampleDS** のように **java:/** または **java:jboss/** で始まる必要があります。

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.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。

利用可能なすべてのデータソースパラメーターの完全リストは、 データソースパラメーターの項を参照してください。

13.3.2. XA データソースの作成

XA データソースは **xa-data-source add** 管理 CLI コマンドを使用して定義できます。管理コンソールを使用して XA データソースを定義することもできます。管理コンソールを使用する場合は **Configuration** → **Subsystems** → **Datasources** → **XA** と選択し、**追加** をクリックして **XA Datasource の作成**ウィザードを開きます。

以下の手順で、管理 CLI を使用した XA データソースの定義方法を説明します。

注記

管理対象ドメインでは、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"=>"HOSTNAME","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=HOSTNAME)
   ```

   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:jboss/datasources/ExampleDS`** のように **`java:/`** または **`java:jboss/`** で始まる必要があります。

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.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。

利用可能なすべてのデータソースパラメーターの完全リストは、 データソースパラメーターの項を参照してください。

13.4. 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。

13.4.1. 非 XA データソースの編集

非 XA データソースは **data-source** 管理 CLI コマンドを使用して更新できます。また、管理コンソールの **datasources** サブシステムからデータソース属性を更新することもできます。

注記

非 XA データソースは JTA トランザクションと統合できます。データソースを JTA 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

管理対象ドメインでは、**--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。

13.4.2. XA データソースの編集

XA データソースは **xa-data-source** 管理 CLI コマンドを使用して更新できます。また、管理コンソールの **datasources** サブシステムからデータソース属性を更新することもできます。

- 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

管理対象ドメインでは、**--profile=PROFILE_NAME** 引数を指定する必要があります。

- 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

管理対象ドメインでは、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。

13.5. 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。

13.5.1. 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。管理コンソールの **datasources** サブシステムからデータソースを削除することもできます。

```
data-source remove --name=DATASOURCE_NAME
```

注記

管理対象ドメインでは、**--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。

13.5.2. 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。管理コンソールの **datasources** サブシステムから XA データソースを削除することもできます。

```
xa-data-source remove --name=XA_DATASOURCE_NAME
```

注記

管理対象ドメインでは、**--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。

13.6. データソース接続のテスト

データソースが JBoss EAP に追加されたら、接続をテストして正しく設定されたことを検証できます。データソースの接続は管理 CLI コマンドを使用してテストするか、**datasources** サブシステムの **Test Connection** ボタンを使用して管理コンソールからテストします。

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

管理対象ドメインでは、このコマンドの前に **/host=HOSTNAME/server=SERVER_NAME** を追加する必要があります。

13.7. 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。

13.7.1. 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、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 データソースの設定については、 XA データソースの編集を参照してください。

表13.2 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	リカバリーでリソースに接続するため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。
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)
```

13.7.2. 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。

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 データのアップデートが、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 でサポートされる構成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 bug 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

一部のユーザークレデンシャル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クレデンシャル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。

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。

13.8. 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。

データベース接続検証を設定するには、検証タイミングメソッド (検証が実行されるタイミング)、検証メカニズム (検証の実行方法)、および例外ソーター (例外の対処方法) 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 オプションが **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。

この設定により、最も早いリカバリーが実現されますが、データベースへの負荷が最も大きくなります。ただし、これは、パフォーマンスを気にする必要がない場合は最も安全な方法です。

background-validation

background-validation オプションを **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

これらのオプションは相互排他的です。**validate-on-match** が **true** に設定された場合は、**background-validation** を **false**に設定する必要があります。 **background-validation** が true に設定された場合は **validate-on-match**を **false** に設定する必要があります。

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**

13.9. 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のソリューションには、セキュリティードメインまたはパスワード vault のいずれかを使用することが推奨されます。以下にそれぞれの例を示します。

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

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。

```
 <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>
```

注記

セキュリティードメインが複数のデータソースと使用される場合は、セキュリティードメインでキャッシュを無効にする必要があります。キャッシュを無効にするには、**cache-type** 属性の値を **none** に設定するか、この属性を削除します。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する必要があります。

DsRealm セキュリティードメインはデータソース設定によって参照されます。

```
<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、**How to Configure Identity Management** 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

```
<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 **How To Configure Server Security** を参照してください。

13.10. データソースの統計

定義したデータソースのコア**プール**および **JDBC** ランタイム統計を表示できます。利用可能な統計の詳細リストは、データソース統計を参照してください。

データソース統計の有効化

デフォルトでは、データソース統計は有効になって**いません**。以下の管理 CLI コマンドは、**ExampleDS** データソースの統計の取得を有効にします。

注記

管理対象ドメインでは、これらのコマンドの前に **/profile=PROFILE_NAME** を追加してください。

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

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

データソース統計の表示

データソース統計はすべて管理 CLI から取得できます。これらの統計のサブセットは管理コンソールの **Runtime** タブで確認できます。

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

管理対象ドメインでは、これらのコマンドの前に **/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。

13.11. キャパシティーポリシー

JBoss EAP は、データソースを含む JCA 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、プールの物理接続の作成方法 (キャパシティーのインクリメント) および破棄方法 (キャパシティーのデクリメント) 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やキャパシティーデクリメンタークラスを指定する必要があります。

コマンド例

```
/subsystem=datasources/data-source=ExampleDS:write-
attribute(name=capacity-incrementer-class,
value="org.jboss.jca.core.connectionmanager.pool.capacity.SizeIncrement
er")

/subsystem=datasources/data-source=ExampleDS:write-
attribute(name=capacity-decrementer-class,
value="org.jboss.jca.core.connectionmanager.pool.capacity.SizeDecrement
er")
```

指定の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。

コマンド例

```
/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13.3 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、 **size** の値が 1 のデフォルトインクリメントポリシーです。

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13.4 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デクリメントします。

表13.5 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13.6 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

13.12. 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、エンリストメントトレースが記録されます。パフォーマンスオーバーヘッドが生じるため、状況によってはエンリストメントトレースを無効にした方がよい場合があります。

データソースのエンリストメントトレースの記録を無効にするには、管理 CLI を使用して **`enlistment-trace`** 属性を **`false`** 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無効にします。

```
data-source --name=DATASOURCE_NAME --enlistment-trace=false
```

XA データソースのエンリストメントトレースを無効にします。

```
xa-data-source --name=XA_DATASOURCE_NAME --enlistment-trace=false
```

警告

エンリストメントトレースを無効にすると、トランザクションエンリストメント中にエラーを追跡するのが難しくなります。

13.13. データソース設定例

13.13.1. 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。

MySQL データソースの設定例

```
<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.MySQLValidConnectionCh
ecker"/>
      <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.jdbc.Driver</driver-class>
      <xa-datasource-
class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-
class>
    </driver>
  </drivers>
</datasources>
```

MySQL JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.36-bin.jar"/>
  </resources>
  <dependencies>
    <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-5.1.36-bin.jar --
dependencies=javax.api,javax.transaction.api
```

注記

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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.jdbc.jdbc2.optional.MysqlXADataSource,
driver-class-name=com.mysql.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.MySQLValidConne
ctionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionS
orter
```

13.13.2. 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。

MySQL XA データソースの設定例

```
<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.MySQLValidConnectionCh
ecker"/>
      <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.jdbc.Driver</driver-class>
      <xa-datasource-
class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-
class>
    </driver>
  </drivers>
</datasources>
```

MySQL JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.36-bin.jar"/>
  </resources>
  <dependencies>
    <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-5.1.36-bin.jar --
dependencies=javax.api,javax.transaction.api
```

注記

`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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.jdbc.jdbc2.optional.MysqlXADataSource,
driver-class-name=com.mysql.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"}
```

13.13.3. 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。

PostgreSQL データソースの設定例

```
<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>
```

PostgreSQL JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.postgresql">
  <resources>
    <resource-root path="postgresql-9.3-1102.jdbc4.jar"/>
  </resources>
  <dependencies>
```

```
    <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. 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-
9.3-1102.jdbc4.jar --dependencies=javax.api,javax.transaction.api
```

注記

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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.PostgreSQLVa
lidConnectionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLEx
ceptionSorter
```

13.13.4. 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。

PostgreSQL XA データソースの設定例

```
<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.PostgreSQLValidConn
ectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLException
Sorter"/>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="postgresql" module="com.postgresql">
      <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-
datasource-class>
    </driver>
  </drivers>
</datasources>
```

PostgreSQL JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.postgresql">
  <resources>
    <resource-root path="postgresql-9.3-1102.jdbc4.jar"/>
  </resources>
  <dependencies>
    <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. 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-
9.3-1102.jdbc4.jar --dependencies=javax.api,javax.transaction.api
```

注記

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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.PostgreSQLVa
lidConnectionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLEx
ceptionSorter --xa-datasource-properties=
{"ServerName"=>"localhost","PortNumber"=>"5432","DatabaseName"=>"
postgresdb"}
```

13.13.5. 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。

Oracle データソースの設定例

```
<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.OracleValidConnection
Checker"/>
        <validate-on-match>true</validate-on-match>
        <background-validation>false</background-validation>
        <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnection
Checker"/>
        <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>
```

Oracle JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
    <resources>
        <resource-root path="ojdbc7.jar"/>
    </resources>
    <dependencies>
        <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/misc/jdbc_drivers/oracle/ojdbc7.jar --
dependencies=javax.api,javax.transaction.api
```

注記

`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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.OracleValidCon
```

```
nectionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptio
nSorter --stale-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleCon
nectionChecker
```

13.13.6. 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。

Oracle XA データソースの設定例

```
<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.OracleValidConnection
Checker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnection
Checker"/>
      <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>
```

Oracle JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
    <resources>
        <resource-root path="ojdbc7.jar"/>
    </resources>
    <dependencies>
        <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/misc/jdbc_drivers/oracle/ojdbc7.jar --
dependencies=javax.api,javax.transaction.api
```

注記

`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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.OracleValidCon
```

```
nectionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptio
nSorter --stale-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleCon
nectionChecker --same-rm-override=false --xa-datasource-
properties={"URL"=>"jdbc:oracle:thin:@oracleHostName:1521:orcl"}
```

13.13.7. 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。

Microsoft SQL Server データソースの設定例

```
<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.MSSQLValidConnectionCh
ecker"/>
      <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>
```

Microsoft SQL Server の JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.microsoft">
  <resources>
    <resource-root path="sqljdbc41.jar"/>
  </resources>
  <dependencies>
```

```
    <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. 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/sqljdbc41.jar --
dependencies=javax.api,javax.transaction.api
```

注記

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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.MSSQLValidConne
ctionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionS
orter
```

13.13.8. 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。

Microsoft SQL Server XA データソースの設定例

```
<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.MSSQLValidConnectionCh
ecker"/>
      <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>
```

Microsoft SQL Server の JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.microsoft">
  <resources>
    <resource-root path="sqljdbc41.jar"/>
  </resources>
  <dependencies>
    <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. 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/sqljdbc41.jar --
dependencies=javax.api,javax.transaction.api
```

注記

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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.MSSQLValidConne
ctionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionS
orter --same-rm-override=false --xa-datasource-properties=
{"ServerName"=>"localhost","DatabaseName"=>"mssqldb","SelectMetho
d"=>"cursor"}
```

13.13.9. 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。

IBM DB2 データソースの設定例

```
<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.DB2ValidConnectionChecke
```

```
r"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectionChecke
r"/>
      <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>
```

IBM DB2 の JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm">
  <resources>
    <resource-root path="db2jcc4.jar"/>
  </resources>
  <dependencies>
    <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=javax.api,javax.transaction.api
```

注記

`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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.DB2ValidConnectio
nChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorte
r --stale-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectio
nChecker --min-pool-size=0 --max-pool-size=50
```

13.13.10. 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。

IBM DB2 XA データソースの設定例

```
<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.DB2ValidConnectionChecke
r"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectionChecke
r"/>
      <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>
```

IBM DB2 の JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm">
  <resources>
    <resource-root path="db2jcc4.jar"/>
  </resources>
  <dependencies>
    <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=javax.api,javax.transaction.api
```

注記

`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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.DB2ValidConnectio
nChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorte
r --stale-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectio
nChecker --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","DriverType"=>"4"}
```

13.13.11. 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。

Sybase データソースの設定例

```
<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.SybaseValidConnection
Checker"/>
      <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>
```

Sybase JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.sybase">
    <resources>
      <resource-root path="jconn4.jar"/>
    </resources>
    <dependencies>
      <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=javax.api,javax.transaction.api
```

注記

`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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.SybaseValidCon
nectionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptio
nSorter
```

13.13.12. Sybase XA データソースの例

13.13.12. 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。

Sybase XA データソースの設定例

```
<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.SybaseValidConnection
Checker"/>
      <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>
```

Sybase JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.sybase">
    <resources>
      <resource-root path="jconn4.jar"/>
    </resources>
```

```
    <dependencies>
      <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=javax.api,javax.transaction.api
```

注記

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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.SybaseValidCon
nectionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptio
nSorter --same-rm-override=false --xa-datasource-properties=
{"ServerName"=>"localhost","DatabaseName"=>"mydatabase","PortNumb
er"=>"4100","NetworkProtocol"=>"Tds"}
```

13.13.13. 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。

MariaDB データソースの設定例

```
<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.MySQLValidConnectionCh
ecker"/>
      <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>
```

MariaDB JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="org.mariadb">
  <resources>
    <resource-root path="mariadb-java-client-1.2.3.jar"/>
  </resources>
  <dependencies>
    <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. 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-1.2.3.jar --dependencies=javax.api,javax.transaction.api
```

注記

`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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.MySQLValidConne
ctionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionS
orter
```

13.13.14. 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。

MariaDB XA データソースの設定例

```
<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.MySQLValidConnectionCh
ecker"/>
      <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>
```

MariaDB JDBC ドライバー `module.xml` ファイルの例

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="org.mariadb">
  <resources>
    <resource-root path="mariadb-java-client-1.2.3.jar"/>
  </resources>
  <dependencies>
    <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. 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-1.2.3.jar --dependencies=javax.api,javax.transaction.api
```

注記

`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、技術プレビューとしてのみ提供されています。本番環境では、モジュールを手動 で追加および削除する必要があります。

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.MySQLValidConne
ctionChecker --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionS
orter --xa-datasource-properties=
{"ServerName"=>"localhost","DatabaseName"=>"mariadbdb"}
```

第14章 トランザクションの設定

14.1. TRANSACTIONS サブシステムの設定

14.1.1. トランザクションマネージャーの設定

トランザクションマネージャーは、Web ベースの管理コンソールまたはコマンドライン管理 CLI を使用して設定できます。

管理コンソールを使用したトランザクションマネージャーの設定

以下の手順は、Web ベースの管理コンソールを使用してトランザクションマネージャーを設定する方法を示しています。

1. 画面上部の **Configuration** タブを選択します。
2. JBoss EAP を管理対象ドメインとして実行している場合は、変更する任意のプロファイルを選択します。
3. **Subsystem** リストから、**Transactions** を選択し、**View** をクリックします。
4. 編集する設定に応じたタブ (リカバリーオプションの場合の **Recovery** など) で **Edit** をクリックします。
5. 必要な変更を行い、**Save** をクリックして変更を保存します。
6. **ヘルプ** をクリックしてヘルプテキストを表示します。

管理 CLI を使用したトランザクションマネージャーの設定

管理 CLI で一連のコマンドを使用してトランザクションマネージャーを設定できます。これらのコマンドはすべて **`/subsystem=transactions`** (スタンドアロンサーバー向け) または **`/profile=default/subsystem=transactions/`** (管理対象ドメインの **`default`** プロファイル向け) で始まります。

トランザクションマネージャーのすべての設定オプションの詳細なリストについては、トランザクションマネージャーの設定オプションを参照してください。

14.1.2. JTA を使用するようデータソースを設定

ここでは、データソースで Java Transaction API (JTA) を有効にする方法を説明します。

前提条件

- データベースが Java Transaction API をサポートする必要があります。不明な場合は、データベースのドキュメントを参照してください。
- 非 XA データソース を作成します。

注記

XA データソース はすでにデフォルトで JTA が有効になっています。

Java Transaction API を使用するようデータソースを設定

1. 以下の管理 CLI コマンドを使用して **jta** 属性を **true** に設定します。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:write-
attribute(name=jta,value=true)
```

注記

管理対象ドメインでは、このコマンドの前に **/profile=PROFILE_NAME** を追加します。

2. 変更を反映するためにサーバーをリロードします。

```
reload
```

データソースが JTA を使用するように設定されます。

14.1.3. トランザクションログメッセージ

トランザクションロガーに **DEBUG** ログレベルを使用することにより、ログファイルを読み取り可能な状態に保ちつつトランザクションを追跡できます。詳細なデバッグの場合は、**TRACE** ログレベルを使用します。トランザクションロガーの設定に関する詳細については、Transactions サブシステムのロギング設定を参照してください。

TRACE ログレベルでログを記録するよう設定すると、トランザクションマネージャー (TM) は多くのロギング情報を生成できます。最も一般的なメッセージの一部は次のとおりです。このリストは包括的ではなく、他のメッセージが表示されることもあります。

表14.1 トランザクション状態の変更

トランザクションの開始	トランザクションが開始されたら、クラス **com.arjuna.ats.arjuna.coordinator.BasicAction** のメソッド **Begin** が実行され、メッセージ **BasicAction::Begin() for action-id <transaction uid>** でログに示されます。
トランザクションのコミット	トランザクションがコミットされたら、クラス **com.arjuna.ats.arjuna.coordinator.BasicAction** のメソッド **Commit** が実行され、メッセージ **BasicAction::Commit() for action-id <transaction uid>** でログに示されます。

トランザクションのロールバック	トランザクションがロールバックされたら、クラス **com.arjuna.ats.arjuna.coordinator.BasicAction** のメソッド **Rollback** が実行され、メッセージ **BasicAction::Rollback() for action-id <transaction uid>** でログに示されます。
トランザクションのタイムアウト	トランザクションがタイムアウトすると、**com.arjuna.ats.arjuna.coordinator.TransactionReaper** のメソッド **doCancellations** が実行され、**Reaper Worker <thread id> attempting to cancel <transaction uid>** とログに示されます。この結果、上記のように同じスレッドがトランザクションをロールバックすることが示されます。

14.1.4. Transactions サブシステムのロギング設定

JBoss EAP の他のログ設定に依存せずにログに記録されたトランザクションに関する情報の量を制御できます。ログ設定は、管理コンソールまたは管理 CLI を使用して設定できます。

管理コンソールを使用したトランザクションロガーの設定

1. **Logging** サブシステム設定に移動します。
    a. 管理コンソールで、**Configuration** タブをクリックします。管理対象ドメインを使用する場合は、最初に適切なサーバープロファイルを選択する必要があります。
    b. **Logging** サブシステムを選択し、**View** をクリックします。
2. **com.arjuna** 属性を編集します。

    Log Categories タブを選択します。**com.arjuna** エントリーがすでに存在します。**com.arjuna** を選択し、**属性**セクションの **Edit** をクリックします。ログレベルを変更し、親ハンドラーを使用するかどうかを選択できます。

    - ログレベル:

      トランザクションにより大量のロギング出力が生成されることがあるため、サーバーのログがトランザクション出力で満たされないようデフォルトのロギングレベルは **WARN** に設定されます。トランザクション処理の詳細を確認する必要がある場合は、トランザクション ID が表示されるよう **TRACE** ログレベルを使用します。

    - 親ハンドラーの使用:

      親ハンドラーはロガーが出力を親ロガーに送信するかどうかを指定します。デフォルトの動作は **true** です。
3. **Save** をクリックして変更を保存します。

管理 CLI を使用したトランザクションロガーの設定

以下のコマンドを使用して管理 CLI からログレベルを設定します。スタンドアロンサーバーの場合は、コマンドから **/profile=default** を削除します。

```
/profile=default/subsystem=logging/logger=com.arjuna:write-
attribute(name=level,value=VALUE)
```

14.2. トランザクション管理

14.2.1. トランザクションの参照と管理

管理 CLI では、トランザクションレコードを参照および操作する機能がサポートされます。この機能は、TM と JBoss EAP の管理 API 間の対話によって提供されます。

TM は、待機中の各トランザクションとトランザクションに関連する参加者に関する情報を、オブジェクトストアと呼ばれる永続ストレージに格納します。管理 API は、オブジェクトストアを **log-store** と呼ばれるリソースとして公開します。**probe** と呼ばれる API 操作はトランザクションログを読み取り、各レコードに対してノードパスを作成します。**probe** コマンドは、**log-store** を更新する必要があるときに、いつでも手動で呼び出すことができます。トランザクションログが即座に表示され、非表示になるのは、正常な挙動です。

ログストアの更新

このコマンドは、管理対象ドメインでプロファイル **default** を使用するサーバーグループに対してログストアを更新します。スタンドアローンサーバーの場合は、コマンドから **profile=default** を削除します。

```
/profile=default/subsystem=transactions/log-store=log-store/:probe
```

準備済みトランザクションすべての表示

準備されたすべてのトランザクションを表示するには、最初にログストアを更新し (ログストアの更新 を参照)、ファイルシステムの **ls** コマンドに類似した機能を持つ次のコマンドを実行します。

```
ls /profile=default/subsystem=transactions/log-store=log-
store/transactions
```

または

```
/host=master/server=server-one/subsystem=transactions/log-store=log-
store:read-children-names(child-type=transactions)
```

各トランザクションが一意の ID とともに表示されます。個々の操作は、各トランザクションに対して実行できます (トランザクションの管理 を参照)。

14.2.1.1. トランザクションの管理

トランザクションの属性の表示

JNDI 名、EIS 製品名およびバージョン、ステータスなどのトランザクションに関する情報を表示するには、**:read-resource** 管理 CLIコマンドを使用します。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-b66efc2\:4f9e6f8f\:9:read-resource
```

トランザクションの参加者の表示

各トランザクションログには、**participants** (参加者) と呼ばれる子要素が含まれます。トランザクションの参加者を確認するには、この要素に対して **read-resource** 管理 CLI コマンドを使用します。参加者は JNDI 名によって識別されます。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-
b66efc2\:4f9e6f8f\:9/participants=java\:\/JmsXA:read-resource
```

結果は以下のようになります。

```
{
   "outcome" => "success",
   "result" => {
       "eis-product-name" => "ActiveMQ",
       "eis-product-version" => "2.0",
       "jndi-name" => "java:/JmsXA",
       "status" => "HEURISTIC",
       "type" => "/StateManager/AbstractRecord/XAResourceRecord"
   }
}
```

ここで示された結果ステータスは **HEURISTIC** であり、リカバリーが可能です。詳細については、トランザクションのリカバリーを参照してください。

特別な場合では、ログに対応するトランザクションレコードがないオーファンレコード (XAResourceRecords) をオブジェクトストアに作成できます。たとえば、準備済みの XA リソースが TM が記録する前にクラッシュした場合や、ドメイン管理 API では XA リソースにアクセスできない場合などがこれに該当します。このようなレコードにアクセスするには、管理オプション **expose-all-logs** を **true** に設定する必要があります。このオプションは管理モデルには保存されず、サーバーが再起動されると **false** に戻ります。

```
/profile=default/subsystem=transactions/log-store=log-store:write-
attribute(name=expose-all-logs, value=true)
```

代わりに以下のコマンドを実行すると、トラザクション参加者 ID が集約され表示されます。

```
/host=master/server=server-one/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-b66efc2\:4f9e6f8f\:9:read-
children-names(child-type=participants)
```

トランザクションの削除

各トランザクションログは、トランザクションを表すトランザクションログを削除する **:delete** 操作をサポートします。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-b66efc2\:4f9e6f8f\:9:delete
```

トランザクションのリカバリー

トランザクションの各参加者は、**:recover** 管理 CLI コマンドを使用したリカバリーをサポートします。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-
b66efc2\:4f9e6f8f\:9/participants=2:recover
```

- トランザクションの状態が **HEURISTIC** である場合は、リカバリー操作によって状態が **PREPARE** に変わり、リカバリーがトリガーされます。
- トランザクションの参加者の 1 つがヒューリスティックな場合、リカバリー操作は **commit** 操作を再実行しようとします。成功した場合、トランザクションログから参加者が削除されます。これを確認するには、**log-store** 上で **:probe** 操作を再実行し、参加者がリストされていないことを確認します。これが最後の参加者の場合は、トランザクションも削除されます。

リカバリーが必要なトランザクション状態の更新

トランザクションをリカバリーする必要がある場合は、リカバリーを試行する前に **:refresh** 管理 CLI コマンドを使用して、トランザクションのリカバリーが必要であるかを確認できます。

```
/profile=default/subsystem=transactions/log-store=log-
store/transactions=0\:ffff7f000001\:-
b66efc2\:4f9e6f8f\:9/participants=2:refresh
```

14.2.2. トランザクション統計情報の表示

トランザクションマネージャーの統計が有効になっていると、トランザクションマネージャーによる処理されたトラザクションの統計を表示できます。トランザクションマネージャーの統計を有効にする方法については、トランザクションマネージャーの設定を参照してください。

管理コンソールまたは管理 CLI を使用して統計を表示できます。管理コンソールでは、**Runtime** タブから **Transactions** サブシステムを選択するとトラザクションの統計を表示できます。管理コンソールではすべての統計を利用できません。

以下の表は、利用できる統計とその説明を示しています。

表14.2 Transactions サブシステムの統計

統計	説明
number-of-transactions	このサーバー上でトランザクションマネージャーにより処理されるトランザクションの合計数。
number-of-committed-transactions	このサーバー上でトランザクションマネージャーにより処理されるコミット済みトランザクションの数。
number-of-aborted-transactions	このサーバー上でトランザクションマネージャーにより処理されるアボートされたトランザクションの数。

統計	説明
number-of-timed-out-transactions	このサーバー上でトランザクションマネージャーにより処理されるタイムアウトのトランザクションの数。タイムアウトのトランザクションの数はアボートされたトランザクションの数まで計算されます。
number-of-heuristics	ヒューリスティック状態のトランザクションの数。
number-of-inflight-transactions	開始した未終了のトランザクションの数。
number-of-application-rollbacks	障害の原因がアプリケーションであった失敗トランザクションの数。
number-of-resource-rollbacks	障害の原因がリソースであった失敗トランザクションの数。

14.2.3. トランザクションオブジェクトストア

トランザクションにはオブジェクトを保存する場所が必要です。オブジェクトストレージのオプションの 1 つが JDBC データソースです。ファイルストアと比べると、JDBC インターフェースは同じネットワークのすべてのシステムからデータベースにアクセスできます。ファイルストアを使用する場合は、ディスクを共有または移行してからでないと、オブジェクトストレージとしてデータベースにアクセスすることはできません。特にパフォーマンスが気になる場合、JDBC オフジェクトストアはファイルシステムまたは ActiveMQ ジャーナルオブジェクトストアよりも速度が遅くなる場合があります。

重要

トラザクションログのストレージタイプとして Apache ActiveMQ Artemis ジャーナルを使用するよう **`transactions`** サブシステムが設定されている場合、JBoss EAP の 2 つのインスタンスは同じディレクトリーを使用してジャーナルを保存することはできません。アプリケーションサーバーインスタンスは同じ場所を共有することはできず、アプリケーションサーバーインスタンスごとに一意な場所を設定する必要があります。

JDBC データソースをトランザクションオブジェクトストアとして使用

JDBC データソースをトランザクションオブジェクトストアとして使用するには、以下の手順に従います。

1. データソース (例: **`TransDS`**) を作成します。手順は 非 XA データソースの作成を参照してください。データソースの JDBC ドライバーは、JAR デプロイメントとしてではなくコアモジュールとしてインストールしないとオブジェクトストアが適切に動作しないことに注意してください。

2. データソースの **jta** 属性を **false** に設定します。

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

3. **jdbc-store-datasource** 属性を、使用するデータソースの JNDI 名に設定します (例: **java:jboss/datasources/TransDS**)。

```
/subsystem=transactions:write-attribute(name=jdbc-store-datasource,
value=java:jboss/datasources/TransDS)
```

4. **use-jdbc-store** 測定を **true** に設定します。

```
/subsystem=transactions:write-attribute(name=use-jdbc-store,
value=true)
```

5. JBoss EAP サーバーをリロードし、変更を反映します。

```
reload
```

以下の表は、JDBC オブジェクトストレージに関係する利用可能な属性をすべて表しています。

表14.3 トランザクション JDBC ストア属性

プロパティー	説明
use-jdbc-store	トランザクションに対して JDBC ストアを有効にするには、**true** に設定します。
jdbc-store-datasource	ストレージに使用される JDBC データソースの JNDI 名。
jdbc-action-store-drop-table	起動時にアクションストアテーブルをドロップおよび再作成するかどうか。デフォルトは **false** です。
jdbc-action-store-table-prefix	アクションストアテーブル名の接頭辞。
jdbc-communication-store-drop-table	起動時にコミュニケーションストアテーブルをドロップおよび再作成するかどうか。デフォルトは **false** です。
jdbc-communication-store-table-prefix	コミュニケーションストアテーブル名の接頭辞。

プロパティー	説明
jdbc-state-store-drop-table	起動時にステートストアテーブルをドロップおよび再作成するかどうか。デフォルトは **`false`** です。
jdbc-state-store-table-prefix	ステートストアテーブル名の接頭辞。

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 Java Transaction Service (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。

15.2. JTS トランザクション用 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 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、管理対象ドメインで **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。

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。

以下の管理 CLI コマンドは、**`full`** プロファイルを使用する管理対象ドメインを対象としています。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。
2. **Subsystems** を選択します。管理対象ドメインの場合は、最初に適切なプロファイルを選択する必要があります。
3. **IIOP** サブシステムを選択し、**View** をクリックします。
4. **Edit** ボタンをクリックし、必要な属性を編集します。各フィールドの詳細を表示するには、**ヘルプ** リンクをクリックします。
5. **Save** をクリックして変更を保存します。

第16章 JAVA CONNECTOR ARCHITECTURE (JCA) の管理

16.1. JAVA CONNECTOR ARCHITECTURE (JCA)

Java EE Connector Architecture (JCA) は外部にある異種のエンタープライズ情報システム (EIS) に対して Java 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 Java EE Connector API アーキテクチャーを実装するコンポーネントです。

データソースオブジェクトと似ていますが、JCA 1.7 は以下を管理する機能を提供します。

- connections
- transactions
- security
- life-cycle
- work instances
- transaction inflow
- message inflow

JCA 1.7 は Java Community Process で JSR-322 として開発されました。

16.2. リソースアダプター

リソースアダプターは、 Java Connector Architecture (JCA) 仕様を使用して Java 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 Java EE コンポーネントです。EIS ベンダーの製品と Java 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。

16.3. JCA サブシステムの設定

`jca` サブシステムは、JCA 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 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

- 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、適切なタブを選択します。

- **Common Config** タブ - 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** ボタンをクリックします。
- **Bootstrap Contexts** タブ - 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。
- **Work Manager**タブ - 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 **View** をクリックします。

管理 CLI での JCA 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。管理対象ドメインでは、コマンドの前に **/profile=PROFILE_NAME** を追加する必要があります。

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
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 JCA 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。例は次のとおりです。

```
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 バリデーション

この設定は、Bean バリデーション(JSR-303) がデプロイメントユニットで実行されるかどうかを決定します。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う。

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。
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 JCA 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16.4 スレッドプールの属性

属性	デフォルト値	説明
name	default	スレッドプールの名前を指定します。
keepalive-time	10 秒	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。
allow-core-timeout	false	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。
thread-factory		スレッドファクトリーへの参照。
max-thread	50	スレッドプールの最大サイズ。
core-threads	50	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。
queue-length	50	キューの最大長。

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16.5 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16.6 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
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。

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管理対象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** をクリックします。管理対象ドメインの場合は、最初に **Content Repository** を選択する必要があります。
3. **新規デプロイメントのアップロード**オプションを選択し、**次へ** をクリックします。
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. 管理対象ドメインでは、デプロイメントを適切なサーバーグループに割り当て、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

このオプションは管理対象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

管理インターフェー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無効にするには、**enlistment-trace** 属性を **false** に設定します。

  ```
  /subsystem=resource-adapters/resource-adapter=eis.rar/connection-
  definitions=cfName:write-attribute(name=enlistment-
  trace,value=false)
  ```

  警告

  エンリストメントトレースを無効にすると、トランザクションエンリストメント中にエラーを追跡するのが難しくなります。

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. WebSphere MQ リソースアダプターのデプロイおよび設定

Websphere MQ リソースアダプターのデプロイ手順については、JBoss EAP **Configuring Messaging** の Deploying a Websphere MQ Resource Adapter を参照してください。

16.4.4. 汎用 JMS リソースアダプターのデプロイおよび設定

汎用 JMS リソースアダプターの設定方法については、JBoss EAP **Configuring Messaging** の Configuring a Generic JMS Resource Adapter を参照してください。

16.5. 管理接続プールの設定

JBoss EAP は **`ManagedConnectionPool`** インターフェースの 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。

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 の過去のバージョンで使用された **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。Java Servlet 3.1 仕様や WebSocket を実装し、HTTP Upgrade をサポートします。また、サーブレットデプロイメントでパフォーマンスの高い非ブロッキングハンドラーの使用をサポートします。**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
- サーバー
- サーブレットコンテナー
- ハンドラー
- 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:3.1">
  <buffer-cache name="default"/>
  <server name="default-server">
      <http-listener name="default" socket-binding="http" redirect-
socket="https"/>
      <host name="default-host" alias="localhost">
          <location name="/" handler="welcome-content"/>
          <filter-ref name="server-header"/>
          <filter-ref name="x-powered-by-header"/>
      </host>
  </server>
  <servlet-container name="default">
      <jsp-config/>
      <websockets/>
  </servlet-container>
  <handlers>
      <file name="welcome-content" path="${jboss.home.dir}/welcome-
```

```
content"/>
  </handlers>
  <filters>
      <response-header name="server-header" header-name="Server" header-
value="JBoss-EAP/7"/>
      <response-header name="x-powered-by-header" header-name="X-Powered-
By" header-value="Undertow/1"/>
  </filters>
</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 の過去のリリースに含まれていた **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。

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:3.1">
  <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. サーバーの設定

サーバーは Undertow のインスタンスを表し、複数の要素で構成されます。

- 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。

注記

複数のサーバーを設定でき、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:3.1">
  <buffer-cache name="default"/>
  <server name="default-server">
      <http-listener name="default" socket-binding="http" redirect-
socket="https"/>
      <host name="default-host" alias="localhost">
          <location name="/" handler="welcome-content"/>
          <filter-ref name="server-header"/>
          <filter-ref name="x-powered-by-header"/>
      </host>
  </server>
  ...
</subsystem>
```

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. 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、 JSP、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:3.1">
  <buffer-cache name="default"/>
  <server name="default-server">
    ...
```

```
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
...
</subsystem>
```

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。

17.5. 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。AJP、HTTP、および HTTP.2 バックエンドを処理でき、mod_cluster をサポートします。

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:3.1">
  <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
```

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。

17.6. 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、過去のバージョンの JBoss EAP 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

JBoss EAP はデフォルトで 2 つのフィルターを提供します。

デフォルトの **Undertow** サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
  <buffer-cache name="default"/>
  <server name="default-server">
    ...
  </server>
  <servlet-container name="default">
    ...
  </servlet-container>
  <handlers>
      ...
  </handlers>
  <filters>
      <response-header name="server-header" header-name="Server" header-
value="JBoss-EAP/7"/>
      <response-header name="x-powered-by-header" header-name="X-Powered-
By" header-value="Undertow/1"/>
  </filters>
</subsystem>
```

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=server-
header: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。

17.7. デフォルトの WELCOME WEB アプリケーションの設定

JBoss EAP には、デフォルトでポート 8080 のルートコンテキストで表示される **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:3.1">
  ...
  <server name="default-server">
      <http-listener name="default" socket-binding="http" redirect-
socket="https"/>
      <host name="default-host" alias="localhost">
          <location name="/" handler="welcome-content"/>
          <filter-ref name="server-header"/>
          <filter-ref name="x-powered-by-header"/>
      </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を変更する
- **default-web-module** を変更する

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

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)
```

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

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)
```

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

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
```

変更を反映するためにサーバーをリロードします。

```
reload
```

17.8. HTTPS の設定

両方の Web アプリケーションおよび管理インターフェースと使用するために HTTPS を設定する方法の詳細は、How to Configure Server Security Guide を参照してください。

17.9. 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
```

重要

HTTP セッションタイムアウトを変更する場合は、影響を受けるすべての JBoss EAP インスタンスを再起動する必要があります。この操作が行われるまで元の HTTP セッションタイムアウト値が適用されます。

17.10. 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、HTTP API と非 HTTP API (JavaScript など) 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

http-only 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
```

17.11. 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。新しい仕様に更新していないクライアントをサポートするため、Undertow は HTTP/2 以前の SPDY とも互換性を維持します。

重要

JBoss EAP 7.0 では HTTP/2 は技術プレビューとしてのみサポートされ、HTTP/2 標準をサポートするブラウザーでのみ動作します。

重要

HTTP/2 を使用するには、Java 8 を使用し、クラスパス上に ALPN を設定する必要があります。これは、HTTP/2 には Java 8 のデフォルトインストールに含まれていない ALPN をサポートする TLS スタックが必要であるためです。

17.11.1. Undertow が HTTP/2 を使用するよう設定

Undertow が HTTP/2 を使用するよう設定するには、以下を行う必要があります。

Undertow が HTTPS を使用するよう設定する

Web アプリケーションに HTTPS を使用するよう Undertow を設定するには、How to Configure Server Security Guide を参照してください。

注記

HTTP アップグレードを使用するプレーン HTTP の場合は HTTPS を使用せずに HTTP/2 を使用することは可能です。この場合、ALPN をインストールする必要はなく、Undertow 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)
```

ALPN JAR のダウンロード

最初に、Java のバージョンを確認します。ターミナルで以下のコマンドを実行し、インストールされている Java のバージョンを出力します。

```
java -version
```

このバージョンを基に このページ を参照し、このリスト からダウンロードする ALPN JAR の正しいバージョンを判断します。たとえば、実行している Java のバージョンが **`1.8.0_51`** である場合、使用する ALPN バージョンは **`8.1.4.v20150727`** で、**`alpn-boot-8.1.4.v20150727.jar`** をダウンロードします。

ALPN JAR の ブートクラスパスへの追加

正しいバージョンの ALPN JAR をダウンロードしたら、**EAP_HOME/bin** にコピーします。また、以下を **bin/standalone.conf** (管理対象ドメインで実行している場合は **bin/domain.conf**) に追加する必要もあります。**$JBOSS_HOME** と **$ALPN_VERSION** は適切な値に置き換えてください。

```
JAVA_OPTS="$JAVA_OPTS -Xbootclasspath/p:$JBOSS_HOME/bin/alpn-boot-
$ALPN_VERSION.jar"
```

重要

クラスパスの変更を反映するには、JBoss EAP を再起動する必要があります。

HTTPS リスナーでの HTTP/2 の有効化

HTTP/2 を使用するよう Undertow の HTTPS リスナーを有効にするには、**enable-http2** 属性を **true** に設定します。

```
/subsystem=undertow/server=default-server/https-listener=https:write-
attribute(name=enable-http2,value=true)
```

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 HTTP/2 擬似ヘッダーを表示します (**:path**, **:authority**、**:method**、および **:scheme**)。 Firefox や Safari などの他のブラウザーは、**HTTP/2.0** のようにヘッダーの状態またはバージョンを表示します。

17.12. REQUESTDUMPING ハンドラーの設定

RequestDumping ハンドラー (**io.undertow.server.handlers.RequestDumpingHandler**)、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 の過去のバージョンで使用された **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で設定できます。

17.12.1. 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**、または **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。

17.12.2. 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。

`WEB-INF/undertow-handlers.conf` の例

```
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。

`WEB-INF/undertow-handlers.conf` の例

```
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。

第18章 リモーティングの設定

18.1. REMOTING サブシステム

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 には、設定可能な要素としてエンドポイント、コネクター、複数のローカルおよびリモート接続 URI が含まれます。独自のアプリケーションにカスタムコネクターを使用する場合を除き、**remoting** のサブシステムの設定は必要でないことがほとんどです。EJB 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:3.0">
    <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。

リモーティングエンドポイント

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。

コネクター

コネクターは主なリモーティング設定要素です。複数のコネクターを設定できます。各コネクターは、複数のサブ要素を持つ **<connector>** 要素とその他の属性で構成されます。デフォルトのコネクターは複数の JBoss EAP サブシステムによって使用されます。カスタムコネクターの要素や属性の設定はアプリケーションによって異なるため、詳細は Red Hat グローバルサポートサービスまでご連絡ください。

コネクターの設定方法の詳細は、コネクターの設定を参照してください。

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

追加設定

リモーティングは、ネットワークインターフェースや 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:3.0">
  <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:3.0">
    ...
    <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ース

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ースは **public** インターフェースです。このインターフェー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>
```

管理対象ドメインでは、**public** インターフェースはホストごとに **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。

EJB の リモーティングコネクター参照

ejb3 サブシステムにはリモートメソッド呼び出しに対するリモーティングコネクターへの参照が含まれています。デフォルト設定は次のとおりです。

```
<remote connector-ref="remoting-connector" thread-pool-
name="default"/>
```

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。安全な接続と安全でない接続の両方で同じソケットバインディング (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安全なトランスポートまたは安全でないトランスポートを要求します。EJB、ORB、JMS 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安全なインターフェースを使用します。

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:1.1">
    <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

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。

重要

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 16 K が理想のサイズになります。

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。

第20章 バッチアプリケーションの設定

JBoss EAP 7 には JSR-352 に定義されている Java バッチアプリケーションのサポートが導入されました。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 **開発ガイド**の Java バッチアプリケーションの開発を参照してください。

20.1. 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:1.0">
    <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、サーバーの中断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。

20.1.1. 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。管理コンソールを使用してジョブリポジトリーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Batch** サブシステムを選択し、左側のメニューから **In Memory** または **JDBC** を選択します。

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。**data-source** 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=java:jboss/datasources/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
```

20.1.2. バッチスレッドプールの設定

本項では、管理 CLI を使用を使用してバッチジョブに使用されるスレッドプールおよびスレッドファクトリーを設定する方法を説明します。管理コンソールを使用してスレッドプールおよびスレッドファクトリーを設定することもできます。管理コンソールを使用する場合は **Configuration** タブで **Batch** サブシステムを選択し、左側のメニューから **Thread Pools** または **Thread Factories** を選択します。

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。

20.2. 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、バッチジョブを開始、停止、および再開できます。ジョブ実行の詳細を表示することもできます。

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 **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"
            }
        }
    }}}
}
```

第21章 NAMING サブシステムの設定

21.1. 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>
```

21.2. 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(b
inding-type=external-context, cache=true,
class=javax.naming.directory.InitialDirContext, 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"
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:remov
e
```

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
```

21.3. リモート JNDI インターフェースの設定

リモート JNDI インターフェースは、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、このインターフェー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 インターフェースは **<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 インターフェー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 インターフェー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22章 高可用性の設定

22.1. 高可用性

JBoss EAP はデプロイされた Java 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。ワークロードを共有し (ロードバランシング)、他のクラスターメンバーに障害が発生したときにクライアントの処理を引き継ぐよう (フェイルオーバー)、クラスターのメンバーを設定することができます。

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**管理対象ドメイン**) 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 Enterprise JavaBean (EJB)
- シングルサインオン (SSO) メカニズム
- HTTP セッション
- JMS 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 **開発ガイド** を参照してください。

22.2. JGROUPS を用いたクラスター通信

22.2.1. 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 (管理対象ドメインでは **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。

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独自に定義することもできます。

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。

22.2.2. 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 UDP プロトコルを使用して通信します。デフォルトの **ee** JGroups は事前定義された **udp** プロトコルスタックを使用します。

```
<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>
```

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。他のメンバーシップ検出プロトコルに対してスタックを設定する場合は以下の項を参照してください。

- TCPPING プロトコルを使用して静的クラスターメンバーシップのリストを定義する。
- TCPGOSSIP プロトコルを使用し、外部ゴシップルーターを使ってクラスターのメンバーを検出する。

22.2.3. 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。

```
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
# Set the properties for the TCPPING protocol
/subsystem=jgroups/stack=tcpping/protocol=TCPPING:write-
attribute(name=properties,value=
{initial_hosts="HOST_A[7600],HOST_B[7600]",port_range=0,timeout=3
000})
/subsystem=jgroups/stack=tcpping/protocol=MERGE3:add
/subsystem=jgroups/stack=tcpping/protocol=FD_SOCK:add(socket-
binding=jgroups-tcp-fd)
/subsystem=jgroups/stack=tcpping/protocol=FD: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=FRAG2:add
# Set tcpping as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-
attribute(name=stack,value=tcpping)
run-batch
```

```
reload
```

定義するプロトコルの順序が重要になることに注意してください。

2. 環境に合わせてスクリプトを変更します。
   - 管理対象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。
   - 任意で、環境に合わせて TCPPING プロパティーを調整します。
     - **initial_hosts**: 既知と見なされたホストのカンマ区切りのリストであり、初期メンバーシップを検索するために使用できます。
     - **port_range**: 必要な場合はポートの範囲を割り当てできます。ポート範囲として **2** を割り当て、初期ポートが **7600** である場合、TCPPING はポート **7600**-**7601** 上の各ホストと通信しようとします。
     - **timeout**: クラスターメンバーのタイムアウト値 (ミリ秒単位)。
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。

22.2.4. 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。

```
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
# Set the properties for the TCPGOSSIP protocol
/subsystem=jgroups/stack=tcpgossip/protocol=TCPGOSSIP:write-
attribute(name=properties,value={initial_hosts="HOST_A[13001]"})
/subsystem=jgroups/stack=tcpgossip/protocol=MERGE3:add
/subsystem=jgroups/stack=tcpgossip/protocol=FD_SOCK:add(socket-
binding=jgroups-tcp-fd)
/subsystem=jgroups/stack=tcpgossip/protocol=FD: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=FRAG2:add
# Set tcpgossip as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-
attribute(name=stack,value=tcpgossip)
run-batch
reload
```

定義するプロトコルの順序が重要になることに注意してください。

2. 環境に合わせてスクリプトを変更します。
   - 管理対象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。
   - 任意で、環境に合わせて TCPGOSSIP プロパティーを調整します。
     - **initial_hosts**: 既知と見なされたホストのカンマ区切りのリストであり、初期メンバーシップを検索するために使用できます。
     - **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。

22.2.5. JGroups のネットワークインターフェー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ースのみへバインドし、**private** ネットワークインターフェースは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ー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ースにはバインドしません。

ネットワークインターフェー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ー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ースは専用の VLAN (Virtual Local Area Network) の一部にすることが推奨されます。

22.2.6. 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。

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。

```
<subsystem xmlns="urn:jboss:domain:jgroups:4.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <protocol type="pbcast.NAKACK2"/>
      <protocol type="UNICAST3"/>
      <protocol type="pbcast.STABLE"/>
      <protocol type="AUTH">
        <property name="auth_class">org.jgroups.auth.MD5Token</property>
        <property name="auth_value">mytoken</property> <!-- Change this
password -->
        <property name="token_hash">MD5</property>
      </protocol>
      <protocol type="pbcast.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG2"/>
    </stack>
  </stacks>
</subsystem>
```

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。

   以下のコマンドの **VERSION** は適切な JGroups JAR バージョンに置き換え、**PASSWORD** は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. **jgroups** サブシステムで **SYM_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 **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。**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:4.0">
     <stacks>
       <stack name="udp">
         <transport type="UDP" socket-binding="jgroups-udp"/>
         <protocol type="PING"/>
         <protocol type="MERGE3"/>
         <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
         <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="FRAG2"/>
    </stack>
  </stacks>
</subsystem>
```

注記

SYM_ENCRYPT を使用する場合、**AUTH** の設定は任意です。

非対称暗号化の使用

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。**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:4.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
      <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="FRAG2"/>
    </stack>
  </stacks>
</subsystem>
```

2. **jgroups** サブシステムで **AUTH** プロトコルを設定します。

   ASYM_ENCRYPT には **AUTH** が必要です。手順は 認証の設定の項を参照してください。

22.2.7. 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 スタックに設定できます。

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	**keepalive-time**	**max-threads**	**min-threads**	**queue-length**
default	60000L	300	20	100
internal	60000L	4	2	100
oob	60000L	300	20	0
timer	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")
```

22.2.8. 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。例を以下に示します。

```
# 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=PROPER
TY_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て関連するスタックを確認します。**Transport** を選択し、**Properties** タブを選択します。

22.2.9. JGroups トラブルシューティング

22.2.9.1. 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ースカード (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ースがある場合は、適切なインターフェー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ー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。

22.2.9.2. 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。

22.3. INFINISPAN

22.3.1. 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 JSR-107 準拠のキャッシュインターフェースを提供します。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 (管理対象ドメインでは **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 のプライベートモジュールとして含まれ、JBoss EAP のキャッシュ機能を提供します。アプリケーションによる Infinispan の直接使用はサポートされません。

22.3.2. 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:4.0">
  <cache-container name="server" aliases="singleton cluster" default-
cache="default" module="org.wildfly.clustering.server">
    <transport lock-timeout="60000"/>
    <replicated-cache name="default" mode="SYNC">
      <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" mode="ASYNC" l1-lifespan="0"
owners="2">
```

```
      <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" mode="ASYNC" l1-lifespan="0"
owners="2">
      <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">
      <eviction strategy="LRU" max-entries="10000"/>
      <expiration max-idle="100000"/>
    </local-cache>
    <invalidation-cache name="entity" mode="SYNC">
      <transaction mode="NON_XA"/>
      <eviction strategy="LRU" max-entries="10000"/>
      <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。

重要

HTTP セッション、ステートフルセッション Bean、シングルトンサービスやデプロイメントなどのキャッシュやキャッシュコンテナーを追加できます。ユーザーアプリケーションによるこれらのキャッシュの直接使用はサポートされません。

22.3.2.1. 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。

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。管理対象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、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。

  適切なキャッシュコンテナーを選択し、ドロップダウンメニューで **コンテナーの設定**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。

  適切なキャッシュコンテナーを選択し、ドロップダウンメニューで **トランスポートの設定**を選択します。必要に応じてキャッシュコンテナートランスポート設定を行います。

- キャッシュを設定します。

  適切な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 Caches** など) でキャッシュを追加、更新、および削除できます。

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。管理対象ドメインでは 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)
```

22.3.3. 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

Replicated

Replicated モードは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、通常レプリケート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。

Distributed

Distributed モードでは、Infinispan はクラスターを線形にスケールできます。Distributed 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 (所有者) 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分散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。Distributed 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。

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。

同期のレプリケーション

同期レプリケーションでは、ユーザーの要求を処理するスレッドはレプリケーションが正常に終了するまでブロックされます。レプリケーションに成功すると応答がクライアントに送信され、その場合のみスレッドが開放されます。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。

22.3.3.1. 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 Distributed 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。

Replicated 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。管理対象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. レプリケートされたキャッシュ **repl** を追加します。

```
/subsystem=infinispan/cache-container=web/replicated-
cache=repl:add(mode=ASYNC)
/subsystem=infinispan/cache-container=web/replicated-
cache=repl/component=transaction:write-
attribute(name=mode,value=BATCH)
/subsystem=infinispan/cache-container=web/replicated-
cache=repl/component=locking:write-attribute(name=isolation,
value=REPEATABLE_READ)
/subsystem=infinispan/cache-container=web/replicated-
cache=repl/store=file:add
```

2. デフォルトのキャッシュをレプリケートされたキャッシュ **repl** に変更します。

```
/subsystem=infinispan/cache-container=web:write-
attribute(name=default-cache,value=repl)
```

3. サーバーをリロードします。

```
reload
```

Distributed 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 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。管理対象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
```

22.3.3.2. 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。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となり、応答時間が早くなります。

22.3.4. 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
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")
```

22.3.5. 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。

22.3.5.1. 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)
```

22.3.6. 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。

22.3.6.1. 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)。

22.3.6.2. 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)
```

22.3.7. HTTP セッションの JBoss Data Grid への外部化

注記

この機能を使用するには JBoss Data Grid のサブスクリプションが必要になります。

Red Hat JBoss 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 JBoss 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 JBoss 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管理対象ドメインの両方に適用されますが、管理対象ドメインでは各サーバーグループに一意のリモートキャッシュが設定されている必要があります。複数のサーバーグループは同じ Red Hat JBoss Data Grid クラスターを使用できますが、各リモートキャッシュは JBoss EAP サーバーグループに固有となります。

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 JBoss Data Grid サーバーの場所を定義します。

   リモートソケットバインディングの追加例

   ```
   /socket-binding-group=standard-sockets/remote-destination-
   ```

```
outbound-socket-binding=remote-jdg-server1:add(host=JDGHostName1,
port=11222)

/socket-binding-group=standard-sockets/remote-destination-
outbound-socket-binding=remote-jdg-server2:add(host=JDGHostName2,
port=11222)
```

結果の **XML**

```
<socket-binding-group name="standard-sockets" ... >
  ...
  <outbound-socket-binding name="remote-jdg-server1">
    <remote-destination host="JDGHostName1" port="11222"/>
  </outbound-socket-binding>
  <outbound-socket-binding name="remote-jdg-server2">
    <remote-destination host="JDGHostName2" port="11222"/>
  </outbound-socket-binding>
</socket-binding-group>
```

注記

各 Red Hat JBoss 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 JBoss Data Grid サーバーのキャッシュ名が定義されます。

   管理対象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。

   リモートキャッシュコンテナーの追加例

```
/subsystem=infinispan/cache-container=web/invalidation-
cache=jdg:add(mode=SYNC)

/subsystem=infinispan/cache-container=web/invalidation-
cache=jdg/component=locking:write-
attribute(name=isolation,value=REPEATABLE_READ)

/subsystem=infinispan/cache-container=web/invalidation-
cache=jdg/component=transaction:write-
attribute(name=mode,value=BATCH)

/subsystem=infinispan/cache-container=web/invalidation-
cache=jdg/store=remote:add(remote-servers=["remote-jdg-
server1","remote-jdg-server2"], cache=default, socket-
timeout=60000, passivation=false, purge=false, shared=true)
```

   結果の **XML**

```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
  ...
  <cache-container name="web" default-cache="dist"
module="org.wildfly.clustering.web.infinispan" statistics-
enabled="true">
    <transport lock-timeout="60000"/>
    <invalidation-cache name="jdg" mode="SYNC">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <remote-store cache="default" socket-timeout="60000" remote-
servers="remote-jdg-server1 remote-jdg-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** 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、**jdg** 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。

jboss-web.xml ファイルの例

```
<?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.jdg</cache-name>
    </replication-config>
</jboss-web>
```

22.4. 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**)

注記

HTTP/2 は 技術プレビューとしてのみ提供されます。

- 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**)

注記

HTTP/2 は 技術プレビューとしてのみ提供されます。

- 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。

22.4.1. mod_clusger 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、管理対象ドメインを実行し、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、**standard-sockets** ソケットバインディンググループにバインドされる **default** プロファイルを使用します。
- 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、管理対象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/mod-cluster-
   config=configuration:write-attribute(name=advertise-security-key,
   value=mypassword)
   ```

2. mod_cluster のマルチキャストアドレスとポートを持つソケットバインディングを作成します。

   サーバーの検出と負荷分散するサーバーとの通信に使用する mod_cluster のソケット設定を作成する必要があります。

   以下の管理 CLI コマンドを使用して、適切なマルチキャストアドレスとポートが設定された **modcluster** ソケットバインディングを追加します。

```
/socket-binding-group=standard-sockets/socket-
binding=modcluster:add(multicast-port=23364, multicast-
address=224.0.1.105)
```

3. mod_cluster ロードバランサーを含めます。

   アドバタイズセキュリティーとソケットバインディングが設定されたら、JBoss EAP のロードバランサーインスタンス向けに mod_cluster フィルターを Undertow に追加する必要があります。

   以下の管理 CLI コマンドを使用して mod_cluster フィルターを追加します。

```
/profile=default/subsystem=undertow/configuration=filter/mod-
cluster=modcluster:add(management-socket-binding=http, advertise-
socket-binding=modcluster, security-key=mypassword)
```

   以下の管理 CLI コマンドを使用して mod_cluster フィルターをデフォルトのホストにバインドします。

```
/profile=default/subsystem=undertow/server=default-
server/host=default-host/filter-ref=modcluster:add
```

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。

22.4.2. 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=myroute, path=/test)

/subsystem=undertow/configuration=handler/reverse-proxy=my-
handler/host=host2:add(outbound-socket-binding=remote-host2,
scheme=ajp, instance-id=myroute, 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=\/app: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。

22.5. 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構成に関する最新情報は、 JBoss EAP でサポートされる構成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。

22.5.1. 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、サポートされる構成と、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_clusger 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構成に関する最新情報は、 JBoss EAP でサポートされる構成を参照してください。

表22.1 HTTP コネクターでサポートされる構成

コネクター	**Web** サーバー	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	Red Hat Enterprise Linux、Microsoft Windows Server、Oracle Solaris	HTTP、HTTPS、AJP
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

コネクター	Web サーバー	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
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22.2 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	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	可	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	可	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	可	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	可	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。

22.5.2. 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。

22.5.2.1. 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。

22.5.3. 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)
```

22.6. 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。

22.6.1. 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。

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 Server **HTTP 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。

22.6.2. 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、管理対象ドメインで **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/mod-cluster-
   config=configuration/: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/mod-cluster-
config=configuration:list-add(name=proxies,value=proxy1)
/profile=full-ha/subsystem=modcluster/mod-cluster-
config=configuration: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。

22.6.3. mod_cluster ワーカーノードの設定

mod_cluster ワーカーノードは、JBoss EAP サーバーで構成されます。このサーバーは、スタンドアロンサーバーまたは管理対象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。

管理対象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されているワーカーノードは個別に設定されますが、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。
- 管理対象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** プロファイルの管理対象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ースの設定

   デフォルトでは、ネットワークインターフェースがすべて **127.0.0.1** に設定されます。1 つのスタンドアロンサーバーまたはサーバーグループの 1 つ以上のサーバーをホストする各物理ホストが他のサーバーが認識できるパブリック IP アドレスを使用するよう、インターフェースを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** インターフェースの外部 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="${jboss.bind.address.management:EXTERNAL_IP_ADDRES
   S}")
   /interface=public:write-attribute(name=inet-
   address,value="${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。

   管理対象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管理対象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.d
omain.master.port:9999},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:9999}" security-
realm="ManagementRealm"/>
</domain-controller>
```

詳細は、ドメインコントローラーへの接続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。

   add-user スクリプト出力の例 **(**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 EJB calls.
   yes/no? yes
   To represent the user add the following to the server-identities
   definition <secret value="SECRET_VALUE" />
   ```

   この出力で提供される 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、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、vault よりパスワードを取得します。

    - **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。

    - 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 の項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。

    - 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
```

- 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。

  - 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。

  - 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管理対象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。

22.6.4. 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。

注記

Hewlett-Packard の HP-UX v11.3 hpws httpd B.2.2.15.15 はこの機能をサポートしないため、**fail_on_status** パラメーターを使用できません。

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カ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。

fail_on_status の設定例

```
ProxyPass / balancer://MyBalancer stickysession=JSESSIONID|jsessionid
nofailover=on failonstatus=203,204
ProxyPassReverse / balancer://MyBalancer
ProxyPreserveHost on
```

22.6.5. 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** とします)。
- 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** とします)。

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 **true** に設定してください (このオプションはデフォルで **true** に設定されています)。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/mod-cluster-
config=configuration/: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/mod-cluster-
config=configuration/:write-attribute(name=load-balancing-
group,value=ClusterOLD)
```

```
<subsystem xmlns="urn:jboss:domain:modcluster:2.0">
  <mod-cluster-config load-balancing-group="ClusterOLD" advertise-
socket="modcluster" connector="ajp">
    <dynamic-load-provider>
      <load-metric type="cpu"/>
    </dynamic-load-provider>
  </mod-cluster-config>
</subsystem>
```

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

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 (**mod_cluster-manager** コンソールの LBGroup を確認)、**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、管理 CLI を使用して特定のコンテキストを無効にできます。下記の操作は stop-context と呼ばれますが、これによりクラスターノードは **DISABLE-APP** コマンドをロードバランサーに送信するため、**mod_cluster-manager** コンソールの特定コンテキストの横にある **Disable** リンクをクリックするのと同じです (default-host などの仮想ホストエイリアスは前述の **mod_cluster-manager** コンソールの出力例から削除されていることに注意してください)。

mod_cluster-manager Web コンソールを使用する他に、管理 CLI を使用して特定のコンテキストを無効にできます。使用される操作は **`stop-context`** と呼ばれますが、これによりクラスターノードは **DISABLE-APP** コマンドをロードバランサーに送信するため、**mod_cluster-manager** コンソールの特定コンテキストの横にある **Disable** リンクをクリックするのと同じです (default-host などの仮想ホストエイリアスは前述の **mod_cluster-manager** コンソールの出力例から削除されていることに注意してください)。

```
/profile=full-ha/subsystem=modcluster/:stop-context(context=/my-
deployed-application-context, virtualhost=default-host, waittime=50)
```

特定のコンテキストを停止するため、クラスターノードまたはロードバランシンググループ全体はバランサーによるリクエストのルーティングを即座に停止し、利用可能な他のコンテキストへのフェイルオーバーを強制します。特定のコンテキストを無効にするため、クラスターノードまたはロードバランシンググループ全体は特定のコンテキスト、ノード、またはロードバランシンググループに新しいセッションを作成しないようバランサーに伝えます。

22.7. 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 プロキシサーバーからのワークロードを許可します。プロキシサーバーは 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。

22.7.1. Apache HTTP Server での mod_jk の設定

JBoss Core Services Apache HTTP Server のインストール時や JBoss Web Server の使用時、mod_jk モジュール (**`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。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 Server **HTTP Connectors and Load Balancing Guide** の Configuring Apache HTTP Server to Load mod_jk を参照してください。

22.7.2. 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。

22.8. 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。

22.8.1. 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。

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 アドレスは架空であるため、環境に適した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。

22.8.2. 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の 1 つ (**ha** または **full-ha**) 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。

22.9. MICROSOFT ISAPI コネクター

Internet Server API (ISAPI) は、Microsoft の IIS (インターネットインフォーメーションサービス) などの Web サーバー向けの OLE サーバー拡張機能とフィルターを記述するために使用される 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、IIS を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 のサポートされる構成については、JBoss Enterprise Application Platform (EAP) 7 でサポートされる構成を参照してください。

22.9.1. 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** を展開します。
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、**Add ISAPI or CGI Restriction** ウインドウを開きます。
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** と展開します。
   - **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。

22.9.2. 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 **instance-id** ディレクトリーに格納します。

```
# An entry that lists all the workers defined
worker.list=worker01, worker02
```

```
# Entries that define the host and port associated with these
workers

# 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。

22.9.3. 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、それらへのリクエスト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** で構成されます。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 **instance-id** ディレクトリーに格納します。

```
# 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。

22.10. 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 のサポートされる構成については、JBoss Enterprise Application Platform (EAP) 7 でサポートされる構成を参照してください。

22.10.1. 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/** ディレクトリーに展開します。

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 JSP 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

22.10.2. 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。

22.10.3. 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。以下の内容をファイルに貼り付け、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 リファレンス資料

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ースが開かれ、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。
--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ースが開かれ、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ー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。他のインターフェー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 (例: **-bmanagement=IP_ADDRESS**)。

引数またはスイッチ	操作モード	説明
--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。
-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。
-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。
--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。

引数またはスイッチ	操作モード	説明
--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の設定では、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の設定では、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。

引数またはスイッチ	操作モード	説明
--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。
--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。
-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 では異なります。

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 で管理対象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 の RPM 設定ファイル

File	説明
/etc/opt/rh/eap7/wildfly/eap7-standalone.conf	Red Hat Enterprise Linux 7 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 で管理対象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`**

プロパティー	説明
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`**
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`**、管理対象ドメインの場合は **`/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`**、管理対象ドメインの場合は **`/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	管理対象ドメインでは、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`**

プロパティー	説明
WILDFLY_BIND	パブリックインターフェー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。

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。
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、Java 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 を含むエンタープライズ JavaBean (EJB) を設定します。 **`ejb3`** サブシステムの詳細は、JBoss EAP の **Developing EJB Applications** を参照してください。

JBoss EAP サブシステム	説明
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	JAX-RS 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	JCA (Java EE Connector Architecture) 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。
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 JMX (Java Management Extensions) のアクセスを設定します。
jpa	JPA (Java Persistence API) 2.1 のコンテナ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 開発ガイドを参照してください。
jsf	JSF (JavaServer Faces) 実装を管理します。
jsr77	JSR-77 仕様によって定義された Java EE 管理機能を提供します。

JBoss EAP サブシステム	説明
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 JMS 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。
modcluster	サーバー側の mod_cluster ワーカーノードを設定します。
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 インターフェースを設定します。
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。
request-controller	サーバーを正常に停止およびシャットダウンするよう設定します。

JBoss EAP サブシステム	説明
resource-adapters	JCA (Java Connector Architecture) 仕様を使用して Java 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します。 **`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 開発ガイドを参照してください。
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。
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。

JBoss EAP サブシステム	説明
weld	JBoss EAP の CDI (Contexts and Dependency Injection) 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既定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。
-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。

コマンドライン引数	説明
-u、--user <value>	ユーザーの名前。英数字と **,./=@** のみが有効です。
-r、--realm <value>	管理インターフェー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。

A.6. 管理監査ロギング属性

表A.7 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 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** を上書きします。
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 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。

属性	説明
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。
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。

表A.10 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。

属性	説明
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. インターフェース属性

表A.11 インターフェース属性と値

インターフェース要素	説明
any	インターフェー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	このインターフェー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。

インターフェース要素	説明
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	インターフェー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	インターフェースの選択基準の一部として、ループバックインターフェー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ー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。
multicast	インターフェー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。
nic	ネットワークインターフェースの名前 (eth0、eth1、lo など)。
nic-match	使用できるインターフェー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ースの名前を検索する正規表現。
not	インターフェー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	インターフェー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ースであるかどうかを示す空の要素。
public-address	インターフェー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	インターフェー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。

インターフェース要素	説明
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 (たとえば、192.168.0.0/16)。
up	インターフェー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	インターフェースの選択基準の一部として、仮想インターフェー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

表A.12 ソケットバインディング属性

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ー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ー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ース。宣言されたインターフェースの 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。

属性	説明
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に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。

A.9. デフォルトのソケットバインディング

表A.13 デフォルトのソケットバインディング

名前	ポート	マルチキャストポート	説明	ソケットバインディンググループ
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されます。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
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。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
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との間の SSL で暗号化された接続。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
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。	full-sockets、full-ha-sockets

名前	ポート	マルチキャストポート	説明	ソケットバインディンググループ
iiop-ssl	3529		SSL 暗号化 CORBA サービス。	full-sockets、full-ha-sockets
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。	ha-sockets、full-ha-sockets
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。	ha-sockets、full-ha-sockets
jgroups-tcp-fd	57600		TCP を介した HA 障害検出に使用されます。	ha-sockets、full-ha-sockets
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。	ha-sockets、full-ha-sockets
jgroups-udp-fd	54200		UDP を介した HA 障害検出に使用されます。	ha-sockets、full-ha-sockets
management-http	9990		管理レイヤーを用いた HTTP 通信に使用されます。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
management-https	9993		管理レイヤーを用いた HTTPS 通信に使用されます。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
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。	ha-sockets、full-ha-sockets

名前	ポート	マルチキャストポート	説明	ソケットバインディンググループ
txn-recovery-environment	4712		JTA トランザクションリカバリーマネージャー。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets
txn-status-manager	4713		JTA / JTS トランザクションマネージャー。	standard-sockets、ha-sockets、full-sockets、full-ha-sockets

A.10. 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.14 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。
.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。

ファイル名接尾辞	生成	説明
.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。アンデプロイの完了後、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。展開された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.11. 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。

表A.15 デプロイメントスキャナーの属性

名前	デフォルト	説明

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、展開された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、展開された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.12. ルートロガーの属性

表A.16 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`** が導入されたため廃止されました。
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一覧。
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.13. ログカテゴリーの属性

表A.17 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`** が導入されたため廃止されました。
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。

属性	説明
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.14. ログハンドラーの属性

表A.18 コンソール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** が導入されたため廃止されました。
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廃止されました。
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。

属性	説明
target	ログハンドラーの出力先となるシステム出力ストリーム。これは システムエラーストリームの場合は System.err、標準出力ストリームの場合は System.out とすることができます。

表A.19 ファイル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`** が導入されたため廃止されました。
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廃止されました。
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.20 周期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`** が導入されたため廃止されました。
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廃止されました。
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.21 サイズ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`** が導入されたため廃止されました。

属性	説明
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廃止されました。
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。デフォルト値は **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.22 周期サイズ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廃止されました。

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。デフォルト値は **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.23 syslog ハンドラー属性

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。

属性	説明
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。

表A.24 カスタム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`** が導入されたため廃止されました。
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。
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廃止されました。
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。

属性	説明
properties	ロギングハンドラーに使用されるプロパティー。

表A.25 非同期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`** が導入されたため廃止されました。
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廃止されました。
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	この非同期ハンドラーがログメッセージを渡すログハンドラーの一覧。

A.15. データソース接続 URL

表A.26 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
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.16. データソースのパラメーター

表A.27 データソースのパラメーター

パラメーター	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 **`5000`** 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。

パラメーター	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、または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	接続待機中にブロックする最大時間数 (ミリ秒単位) を指定します。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。

パラメーター	データソースタイプ	説明
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。
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。指定されたクラスは、2 つのリソース jar (JDBC ドライバーをコアモジュールとしてインストール) を使用して 1 つのモジュールで JDBC ドライバーとバンドルするか、個別のグローバルモジュール (グローバルモジュールの定義) 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。
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。

パラメーター	データソースタイプ	説明
enabled	非 XA、XA	データソースを有効にするかどうか。
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。
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。
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。有効な値は次のとおりです。 - FailingConnectionOnly (デフォルト) - IdleConnections - EntirePool
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。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。
initial-pool-size	非 XA、XA	プールが保持する最初の接続数。
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	JTA の統合を有効にします。
max-pool-size	非 XA、XA	プールが保持可能な最大接続数。

パラメーター	データソースタイプ	説明
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、JTA 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。
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、JCA 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。

パラメーター	データソースタイプ	説明
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。
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-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	リカバリーのリソースへの接続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。

パラメーター	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	閉じずに同じステートメントを 2 回要求した場合に、同じ基盤の準備済みステートメントを使用するかどうかを指定します。
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。

パラメーター	データソースタイプ	説明
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)。
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。

パラメーター	データソースタイプ	説明
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`** パラメーターが上書きされます。
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。

パラメーター	データソースタイプ	説明
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。

A.17. データソースの統計

表A.28 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。
AverageUsageTime	接続の使用に費やされた平均時間。
BlockingFailureCount	接続の取得に失敗した回数。
CreatedCount	作成された接続の数。

名前	説明
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。
MaxPoolTime	プールの接続の最大時間。
MaxUsageTime	接続使用の最大時間。
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。
TotalGetTime	接続の取得に費やされた合計時間。
TotalPoolTime	プールの接続によって費やされた合計時間。

名前	説明
TotalUsageTime	接続の使用に費やされた合計時間。
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。
XACommitCount	XAResource commit 呼び出しの数。
XACommitMaxTime	XAResource commit 呼び出しの最大時間。
XACommitTotalTime	すべての XAResource commit 呼び出しの合計時間。
XAEndAverageTime	XAResource end 呼び出しの平均時間。
XAEndCount	XAResource end 呼び出しの数。
XAEndMaxTime	XAResource end 呼び出しの最大時間。
XAEndTotalTime	すべての XAResource end 呼び出しの合計時間。
XAForgetAverageTime	XAResource forget 呼び出しの平均時間。
XAForgetCount	XAResource forget 呼び出しの数。
XAForgetMaxTime	XAResource forget 呼び出しの最大時間。
XAForgetTotalTime	すべての XAResource forget 呼び出しの合計時間。
XAPrepareAverageTime	XAResource prepare 呼び出しの平均時間。

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。
XAPrepareTotalTime	すべての XAResource prepare 呼び出しの合計時間。
XARecoverAverageTime	XAResource recover 呼び出しの平均時間。
XARecoverCount	XAResource recover 呼び出しの数。
XARecoverMaxTime	XAResource recover 呼び出しの最大時間。
XARecoverTotalTime	すべての XAResource recover 呼び出しの合計時間。
XARollbackAverageTime	XAResource rollback 呼び出しの平均時間。
XARollbackCount	XAResource rollback 呼び出しの数。
XARollbackMaxTime	XAResource rollback 呼び出しの最大時間。
XARollbackTotalTime	すべての XAResource rollback 呼び出しの合計時間。
XAStartAverageTime	XAResource start 呼び出しの平均時間。
XAStartCount	XAResource start 呼び出しの数。
XAStartMaxTime	XAResource start 呼び出しの最大時間。
XAStartTotalTime	すべての XAResource start 呼び出しの合計時間。

表A.29 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。
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.18. トランザクションマネージャーの設定オプション

表A.30 トランザクションマネージャーの設定オプション

オプション	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** が導入されたため廃止されました。
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。

オプション	説明
hornetq-store-enable-async-io	**`journal-store-enable-async-io`** が導入されたため廃止されました。
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、JTA トランザクションのみを使用します。

オプション	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 JTA 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、管理対象ドメインの場合は **`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。

オプション	説明
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。
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。デフォルト値は **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	統計を有効にするかどうか。
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** が導入されたため廃止されました。
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。

オプション	説明
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。

A.19. 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。

表A.31 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。
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。

属性	説明
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	セキュリティークレデンシャル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。

表A.32 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.33 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)。
background-validation	接続をバックグラウンドで検証するか、または使用前に検証するかを指定します。値を変更した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、サーバーを再起動する必要があります。
blocking-timeout-wait-millis	接続待機中にブロックする最大時間数 (ミリ秒単位) を指定します。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。
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。
flush-strategy	エラー発生時にプールがどのようにフラッシュされるか。有効な値は **`FailingConnectionOnly`** (デフォルト)、**`IdleConnections`**、および **`EntirePool`** です。
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。この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。
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。

属性	説明
no-tx-separate-pool	Oracle では、XA 接続を JTA 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-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。
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。

属性	説明
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`** (セキュリティードメインから) など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指定します。
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	無効な場合に最初の接続で接続割り当てを失敗させるか (**`true`**)、プールですべての接続が使用されるまで試行を続けるか (**`false`**)。
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。

リソースアダプタースキーマは **`EAP_HOME/docs/schema/wildfly-resource-adapters_4_0.xsd`** にあります。

A.20 リソースアダプターの統計

A.20. リソースアダプターの統計

表A.34 リソースアダプターの統計

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。
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。

名前	説明
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.21. UNDERTOW サブシステムの属性

表A.35 undertow の属性

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。
statistics-enabled	false	統計を有効にするかどうか。

バッファーキャッシュの属性

表A.36 buffer-cache の属性

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。

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
  - welcome-file
  - crawler-session-management (設定の一部)
  - jsp (設定の一部)
  - persistent-sessions (設定の一部)
  - session-cookie (設定の一部)
  - websockets (設定の一部)

servlet-container の属性

表A.37 servlet-container の属性

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-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。

属性	デフォルト	説明
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。
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。
proactive-authentication	false	プロアクティブ認証を使用すべきかどうか。**`true`** の場合、クレデンシャルのあるユーザーは常に認証されます。
session-id-length	30	生成されたセッション ID の長さ。セッション ID が長いほどセキュアになります。
stack-trace-on-error	local-only	エラーの発生時にスタックトレースがあるエラーページを生成するかどうか。値は all、none、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.38 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。

welcome-file の属性

ウェルカムファイルを定義し、オプションはありません。

crawler-session-management の属性

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.39 crawler-session-management の属性

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 の属性

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.40 jsp の属性

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 JSP 更新の間隔をチェックします。
development	false	JSP 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	JSP 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 JSP 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。
mapped-file	true	JSP ソースへマップします。
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 JSP スクリプトレットを最適化するかどうか。

属性	デフォルト	説明
recompile-on-fail	false	各リクエストで失敗した JSP 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 JSP エンジンのアドバタイズを有効にします。

persistent-sessions の属性

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.41 persistent-sessions の属性

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 の属性

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.42 session-cookie の属性

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 の属性

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.43 websockets の属性

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。
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。
worker	default	websocket デプロイメントに使用するワーカー。

フィルターの属性

custom-filter フィルター

表A.44 custom-filter の属性

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.45 error-page の属性

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.46 expression-filter の属性

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.47 mod-cluster の属性

属性	デフォルト	説明

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	永遠にキープアライブを使用する接続の数。
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サーバーの HTTP2 へのアップグレードを使用するかどうか。HTTP2 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。
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。

属性	デフォルト	説明
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-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。
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。
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.48 balancer の属性

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。
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 の属性

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.49 node の属性

属性	デフォルト	説明
aliases		ノードのエイリアス。
cache-connections		永遠にキープアライブを使用する接続の数。
elected		選択 (elected) 数。

属性	デフォルト	説明
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。
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。

属性	デフォルト	説明
written		ノードに転送されたバイト数。

表A.50 context の属性

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.51 request-limit の属性

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.52 response-header の属性

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.53 rewrite の属性

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。

Handler の属性

file の属性

表A.54 file の属性

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	ディレクトリーの一覧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 の属性

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.55 reverse-proxy の属性

属性	デフォルト	説明
cached-connections-per-thread	5	永遠にキープアライブを使用する接続の数。
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。
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。
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。

属性	デフォルト	説明
session-cookie-names	JSESSIONID	セッションクッキー名のカンマ区切りリスト。通常は JSESSIONID。

表A.56 host の属性

属性	デフォルト	説明
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。
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。

サーバー属性

server コンポーネントの構造は次のとおりです。

- server
  - http-listener
  - https-listener
  - ajp-listener
  - host
    - access-log (設定の一部)
    - single-sign-on (設定の一部)
    - filter-ref
    - location
      - filter-ref

server の属性

表A.57 server の属性

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。
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。

http-listener の属性

表A.58 http-listener の属性

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、残りのクッキー値は破棄されます。
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	true	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。

属性	デフォルト	説明
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	パーサーが選択された文字エンコーディング (デフォルトでは UTF-8) を使用して 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 メソッドのカンマ区切りリスト。
enable-http2	false	このリスナーの HTTP2 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。
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		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)。圧縮の接続ごとにメモリー量が割り当てされます。大きな値ほどメモリーを使用しますが、圧縮が向上する可能性があります。
http2-initial-window-size		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。
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。

属性	デフォルト	説明
http2-max-frame-size		HTTP/2 の最大フレームサイズ。
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。
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		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。

属性	デフォルト	説明
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。
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 {@link 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)。
resolve-peer-address	false	ホストの DNS ルックアップを有効にします。
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 {@link ReadTimeoutException} が発生します。

以下の属性は読み取り専用で、**undertow** サブシステムに対して統計が有効になっている場合のみ使用できます。

表A.59 http-listener メトリック属性

属性	デフォルト	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。

https-listener の属性

表A.60 https-listener の属性

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、残りのクッキー値は破棄されます。
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	true	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。
decode-url	true	パーサーが選択された文字エンコーディング (デフォルトでは UTF-8) を使用して 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 メソッドのカンマ区切りリスト。
enable-http2	false	このリスナーの HTTP2 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。
enabled	true	リスナーが有効であるかどうか。
enabled-cipher-suites		有効な SSL 暗号を設定します。

属性	デフォルト	説明
enabled-protocols		SSL プロトコルを設定します。
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		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)。圧縮の接続ごとにメモリー量が割り当てされます。大きな値ほどメモリーを使用しますが、圧縮が向上する可能性があります。
http2-initial-window-size		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。
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		HTTP/2 の最大フレームサイズ。
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。
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)。

属性	デフォルト	説明
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		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 {@link 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)。
resolve-peer-address	false	ホストの DNS ルックアップを有効にします。
security-realm		リスナーのセキュリティーレルム。
send-buffer		送信バッファーサイズ。

属性	デフォルト	説明
socket-binding		リスナーのソケットバインディング。
ssl-session-cache-size		アクティブな SSL セッションの最大数。
ssl-session-timeout		SSL セッションのタイムアウト (秒単位)。
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。
verify-client	NOT_REQUESTED	SSL チャネルに対する希望の SSL クライアント認証モード。
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 {@link ReadTimeoutException} が発生します。

以下の属性は読み取り専用で、**undertow** サブシステムに対して統計が有効になっている場合のみ使用できます。

表A.61 https-listener メトリック属性

属性	デフォルト	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。

属性	デフォルト	説明
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。

ajp-listener 属性

表A.62 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、残りのクッキー値は破棄されます。
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	true	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。

属性	デフォルト	説明
decode-url	true	true の場合、パーサーは選択された文字エンコーディング (デフォルトでは UTF-8) を仕様して 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 メソッドのカンマ区切りリスト。
enabled	true	リスナーが有効であるかどうか。
max-ajp-packet-size		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。
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。

属性	デフォルト	説明
max-parameters	100	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		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 {@link 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)。
resolve-peer-address	false	ホストの DNS ルックアップを有効にします。

属性	デフォルト	説明
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。
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 {@link ReadTimeoutException} が発生します。

以下の属性は読み取り専用で、**`undertow`** サブシステムに対して統計が有効になっている場合のみ使用できます。

表A.63 ajp-listener メトリック属性

属性	デフォルト	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。

属性	デフォルト	説明
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。

host の属性

表A.64 host の属性

属性	デフォルト	説明
alias		ホストのエイリアスのカンマ区切りリスト。
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 に設定した場合、このホストに対して /console リダイレクトが有効になりません。

filter-ref の属性

表A.65 filter-ref の属性

属性	デフォルト	説明

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 の属性

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.66 access-log の属性

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。
pattern	common	アクセスログパターン。
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。

属性	デフォルト	説明
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。
worker	default	ロギングに使用するワーカーの名前。

single-sign-on の属性

注記

管理 CLI を使用して **single-sign-on** 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=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.67 single-sign-on の属性

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。

location の属性

表A.68 location の属性

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。

A.22. HTTP メソッドのデフォルトの動作。

JBoss EAP 7.0 の **undertow** subsystem は、これまでの JBoss EAP リリースの **web** サブシステムと比較して 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 7.0 のデフォルト動作を表しています。

表A.69 HTTP メソッドのデフォルト動作

HTTP メソッド	**JSP**	**Servlet**	静的 **HTML**
GET	OK	実装によって異なる	OK
POST	OK	実装によって異なる	NOT_ALLOWED

HTTP メソッド	JSP	Servlet	静的 HTML
HEAD	OK	実装によって異なる	OK
PUT	NOT_ALLOWED	実装によって異なる	NOT_ALLOWED
TRACE	NOT_ALLOWED	NOT_ALLOWED	NOT_ALLOWED
DELETE	NOT_ALLOWED	実装によって異なる	NOT_ALLOWED
OPTIONS	NOT_ALLOWED	実装によって異なる	OK

A.23. IO サブシステムの属性

表A.70 worker の属性

属性	デフォルト	説明
io-threads		使用する IO スレッドの数。
stack-size	0	スタックのサイズ。
task-keepalive	60	タスクのキープアライブ時間。
task-max-threads		タスクの最大スレッド数。

表A.71 buffer-pool の属性

属性	デフォルト	説明
buffer-size		バッファーのサイズ
buffers-per-slice		スライスごとのバッファー数。

属性	デフォルト	説明
direct-buffers		直接バッファーをバッファープールが使用するかどうか。

A.24. REMOTING サブシステムの属性

表A.72 remoting の属性

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.73 endpoint の属性

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。

属性	デフォルト	説明
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	**remoting**	SaslServer または SaslClient が作成され、デフォルトで **remoting** プロトコルが指定された場合、これを使用して上書きできます。
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	接続チャネルの受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、**remoting** 要素の **configuration** (例: **`/subsystem=remoting/configuration=endpoint/`**) で使用できます。

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.74 connector の属性

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-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。
socket-binding		アタッチするソケットバインディングの名前。

表A.75 property の属性

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。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.76 sasl の属性

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リストが含まれます。
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」要素には暗号強度の値が希望順に並ぶリストが含まれます。

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.77 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。

属性	デフォルト	説明
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。
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。
pass-credentials	true	オプションネストされた **pass-credentials** 要素には、クライアントのクレデンシャル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.78 http-connector の属性

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。
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.79 outbound-connection の属性

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.80 property の属性

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.81 remote-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。
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。
security-realm		パスワードおよび 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。
username		リモートサーバーの認証時に使用するユーザー名。

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.82 local-outbound-connection の属性

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.25. 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.83 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**)**。
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

ディレクティブ	説明	値
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。たとえば、**`ProxyPassMatch ^(/.*\.gif)$ !`** を指定すると、Apache HTTP Server は直接 .gif ファイルに対応できます。	

mod_cluster ロジックのホットスタンバイノードは、他のすべてのノードがダウンした場合にすべての要求がルーティングされる最後のノードです。これは、mod_proxy のホットスタンバイロジックと似ています。

ホットスタンバイノードを設定するには、**`modcluster`** サブシステムの dynamic-load-provider を、ファクターが **`0`** に設定されている **`simple-load-provider`** に置き換えます。

```
<subsystem xmlns="urn:jboss:domain:modcluster:2.0">
  <mod-cluster-config advertise-socket="modcluster" connector="ajp">
    <dynamic-load-provider>
      <load-metric type="cpu"/>
    </dynamic-load-provider>
    <simple-load-provider factor="0"/>
  </mod-cluster-config>
</subsystem>
```

mod_cluster-manager コンソールで、状態が **OK** で **Load: 0** のノードが表示されます。

たとえば、以下の 3 つのノードがあるとします。

- ノード A、Load: 10
- ノード B、Load: 10
- ノード C、Load: 0

この場合、負荷はノード A と B の間で分散されます。両方のノードが使用できない場合は、ノード C に負荷が集中します。

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。**server config** コンテキストは、ディレクティブが VirtualHost 設定の外部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.84 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。	
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。これらのファイルは NFS 共有ではなくローカルドライブに格納することが推奨されます。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`**

ディレクティブ	説明	値
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
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

ディレクティブ	説明	値
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.26. 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。

- mod-cluster-config
  - dynamic-load-provider
    - custom-load-metric
    - load-metric
  - ssl

表A.85 mod-cluster-config 設定オプション

属性	デフォルト	説明
advertise	true	Advertise ロジックを使用するかどうか。
advertise-security-key		Advertise ロジックのセキュリティーキーが含まれる文字列。
advertise-socket		Advertise ソケットに使用するソケットバインディングの名前。
auto-enable-contexts	true	mod_cluster_manager (httpd) で無効になっている場合でもコンテキストを有効にします。
balancer		バランサーの名前。
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		mod_cluster が無視するコンテキストのリスト。形式は文字列のカンマ区切り。
flush-packets	false	httpd でのパケットのフラッシュを有効または無効にします。
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**。
load-balancing-group		loadBalancingGroup の名前。
max-attempts	1	べき等のリクエストを処理する最大試行回数。**-1** から **2,147,483,647** までの値が許可されます。
node-timeout	-1	ノードへのプロキシー接続のタイムアウト値 (秒単位)。**-1** から **2,147,483,647** までの値が許可されます。

属性	デフォルト	説明
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 のプロキシーリスト。
proxy-list		プロキシーのリスト。形式 (hostname:port) はカンマ区切り。 注記 **`proxy-list`** 属性は非推奨となり、**`proxies`** 属性 (outbound-socket-binding への参照のリスト) の使用が推奨されます。
proxy-url	/	MCMP リクエストのベース URL。
session-draining-strategy	DEFAULT	Web アプリケーションのアンデプロイ中に使用されるセッションドレインストラテジー。
simple-load-provider		簡単なロードプロバイダー。**`-1`** から **`2,147,483,647`** までの値が許可されます。
smax	-1	httpd でのソフトリミットの最大アイドル接続数。**`-1`** から **`2,147,483,647`** までの値が許可されます。
socket-timeout	20	httpd による MCMP メッセージへの返答待ちのタイムアウト値。**`-1`** から **`2,147,483,647`** までの値が許可されます。
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。

属性	デフォルト	説明
sticky-session	true	リクエストにスティッキーセッションを使用します。
sticky-session-force	false	セッション情報を持つリクエストをフェイルオーバーしません。
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。
stop-context-timeout	10	コンテキストによる保留中リクエストの処理を待つ最大時間。**-1** から **2,147,483,647** までの値が許可されます。
ttl	-1	smax を超えるアイドル接続の TTL (Time To Live)。**-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.86 dynamic-load-provider 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。

表A.87 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。

属性	デフォルト	説明
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	メトリックの重み。

表A.88 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。
weight	1	メトリックの重み。

表A.89 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。

属性	デフォルト	説明
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.27. 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 の **undertow** サブシステムに設定された **instance-id** と一致する必要がある一意な名前です。**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.90 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カンマ区切りリスト。

表A.91 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.92 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。カンマ区切りのワーカーノード名のリストで構成されます。
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.93 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。
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。

プロパティー	説明
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 2 つのワーカーノード (**node1** および **node2**) 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。

workers.properties ファイルの例

```
# 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.28. 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。

表A.94 デフォルトの設定オプション

属性	デフォルト	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。

Revised on 2017-02-16 23:39:36 EST