Red Hat JBoss Enterprise Application Platform 7.4-Beta

パフォーマンスチューニングガイド

Red Hat JBoss Enterprise Application Platform のパフォーマンスを評価する手順と、パフォーマンスを向上させるために更新を設定する手順。

Last Updated: 2021-05-17

Red Hat JBoss Enterprise Application Platform 7.4-Beta パフォーマンスチューニングガイド

Red Hat JBoss Enterprise Application Platform のパフォーマンスを評価する手順と、パフォーマンスを向上させるために更新を設定する手順。

法律上の通知

概要

本書は、Red Hat JBoss Enterprise Application Platform のパフォーマンスを調整するためのガイドです。

目次

第1章 はじめに

JBoss EAP インストールは、デフォルトで最適化されています。しかし、環境、アプリケーション、および JBoss EAP サブシステムの使用の設定はパフォーマンスに影響するため、追加の設定が必要になることがあります。

本書は、一般的な JBoss EAP のユースケースを最適化する推奨設定を取り上げ、パフォーマンスの監視やパフォーマンス問題の分析に関する手順を提供します。

重要

パフォーマンス設定の変更は、開発またはテスト環境にて予期される条件下でストレステストを行い、検証してから、実稼働環境にデプロイしてください。

1.1. 本書における EAP_HOME の使用

本書では、変数 **EAP_HOME** を使用して JBoss EAP へのパスを示しています。この変数は JBoss EAP インストールへの実際のパスに置き換えてください。

- ZIP インストール方法で JBoss EAP をインストールした場合、インストールディレクトリーは、ZIP アーカイブを抽出した **jboss-eap-7.4** ディレクトリーとなります。
- RPM インストール方法で JBoss EAP をインストールした場合、インストールディレクトリーは **/opt/rh/eap7/root/usr/share/wildfly/** になります。
- インストーラーを使用して JBoss EAP をインストールした場合、**EAP_HOME** のデフォルトのパスは **${user.home}/EAP-7.4.0** になります。
  - For Red Hat Enterprise Linux and Solaris: **/home/USER_NAME/EAP-7.4.0/**
  - For Microsoft Windows: **C:\Users\USER_NAME\EAP-7.4.0**
- Red Hat CodeReady Studio インストーラーを使用して JBoss EAP サーバーをインストールおよび設定した場合、**EAP_HOME** のデフォルトのパスは **${user.home}/devstudio/runtimes/jboss-eap** になります。
  - Red Hat Enterprise Linux の場合、**/home/USER_NAME/devstudio/runtimes/jboss-eap/** になります。
  - Microsoft Windows の場合、**C:\Users\USER_NAME\devstudio\runtimes\jboss-eap** または **C:\Documents and Settings\USER_NAME\devstudio\runtimes\jboss-eap** になります。

注記

EAP_HOME は環境変数ではありません。**JBOSS_HOME** がスクリプトで使用される環境変数です。

第2章 パフォーマンスの監視

JBoss EAP のパフォーマンスは、マシン上で実行される JVM を分析できるツールを使用して監視できます。Red Hat は、JBoss EAP に事前設定されたラッパースクリプトが含まれる JConsole か、Java VisualVM の使用を推奨します。これらのツールは、メモリー使用量、スレッド使用状態、ロードされたクラス、その他の JVM メトリックスなどの JVM の基本的な監視を行います。

これらのツールの 1 つを JBoss EAP が稼働している同じマシン上で実行する場合、設定は必要ありません。しかし、これらのツールの 1 つを実行して、リモートマシン上で稼働している JBoss EAP を監視する場合、JBoss EAP がリモート JMX 接続を許可する必要があるため、一部の設定が必要になります。

2.1. リモート監視接続のための JBOSS EAP の設定

スタンドアロンサーバーの場合

1. 管理ユーザーが作成済みであることを確認してください。JBoss EAP サーバーの監視に個別の管理ユーザーを作成することがあります。詳細は、JBoss EAP **『設定ガイド』** を参照してください。

2. JBoss EAP を開始するとき、管理インターフェースをサーバーのリモート監視に使用する IP アドレスにバインドしてください。

```
$ EAP_HOME/bin/standalone.sh -bmanagement=IP_ADDRESS
```

> **警告**
>
> これにより、管理コンソールと管理 CLI を含むすべての JBoss EAP 管理インターフェースが指定のネットワークに公開されます。必ず管理インターフェースのみをプライベートネットワークにバインドするようにしてください。

3. JVM 監視ツールの管理ユーザー名とパスワードに以下の URI を使用して JBoss EAP サーバーに接続します。以下の URI はデフォルトの管理ポート (**9990**) を使用します。

```
service:jmx:remote+http://IP_ADDRESS:9990
```

管理対象ドメインホストの場合

管理インターフェースをバインドする上記の手順を管理対象ドメインホストで使用すると、リモート監視するホストコントローラー JVM のみが公開され、そのホスト上で稼働している個別の JBoss EAP サーバーは公開されません。

JBoss EAP を設定して、管理対象ドメインホストで各サーバーをリモート監視するには、以下の手順に従います。

1. リモート監視する JBoss EAP サーバーに接続するために使用する新規ユーザーを **ApplicationRealm** で作成します。詳細は、JBoss EAP **『設定ガイド』** を参照してください。

2. 管理 CLI で以下のコマンドを実行し、リモーティングポートをソケットバインディングツールに追加し、リモーティングを **ApplicationRealm** に追加します。必要な場合は、以下のコマンドのプロファイル名とソケットバインディンググループを使用中のものに置き換えます。

```
/profile=full/subsystem=jmx/remoting-connector=jmx:add(use-management-endpoint=false)
/socket-binding-group=full-sockets/socket-binding=remoting:add(port=4447)
/profile=full/subsystem=remoting/connector=remoting-connector:add(socket-
binding=remoting,security-realm=ApplicationRealm)
```

3. JBoss EAP 管理対象ドメインホストを起動するとき、以下のインターフェースの 1 つまたは両方を、監視に使用する IP アドレスにバインドします。

- 管理対象ドメインホスト上で実行されている個別の JBoss EAP サーバーの JVM に接続する場合は、パブリックインターフェースをバインドします。

```
$ EAP_HOME/bin/domain.sh -b=IP_ADDRESS
```

- JBoss EAP ホストコントローラーの JVM に接続する場合は、管理インターフェースもバインドします。

```
$ EAP_HOME/bin/domain.sh -bmanagement=IP_ADDRESS
```

警告

これにより、管理コンソールと管理 CLI を含むすべての JBoss EAP 管理インターフェースが指定のネットワークに公開されます。必ず管理インターフェースのみをプライベートネットワークにバインドするようにしてください。

4. JVM 監視ツールで以下の詳細を使用します。

- 管理対象ドメインホスト上で実行されている個別の JBoss EAP サーバーの JVM に接続するには、前の手順で作成した **ApplicationRealm** のユーザー名およびパスワードで以下の URI を使用します。

```
service:jmx:remote://IP_ADDRESS:4447
```

単一ホスト上の別の JBoss EAP サーバーに接続するには、対象となるサーバーのポートオフセット値を上記のポート番号に追加します。

- JBoss EAP ホストコントローラーの JVM に接続するには、管理ユーザー名とパスワードで以下の URI を使用します。

```
service:jmx:remote://IP_ADDRESS:9990
```

2.2. JCONSOLE

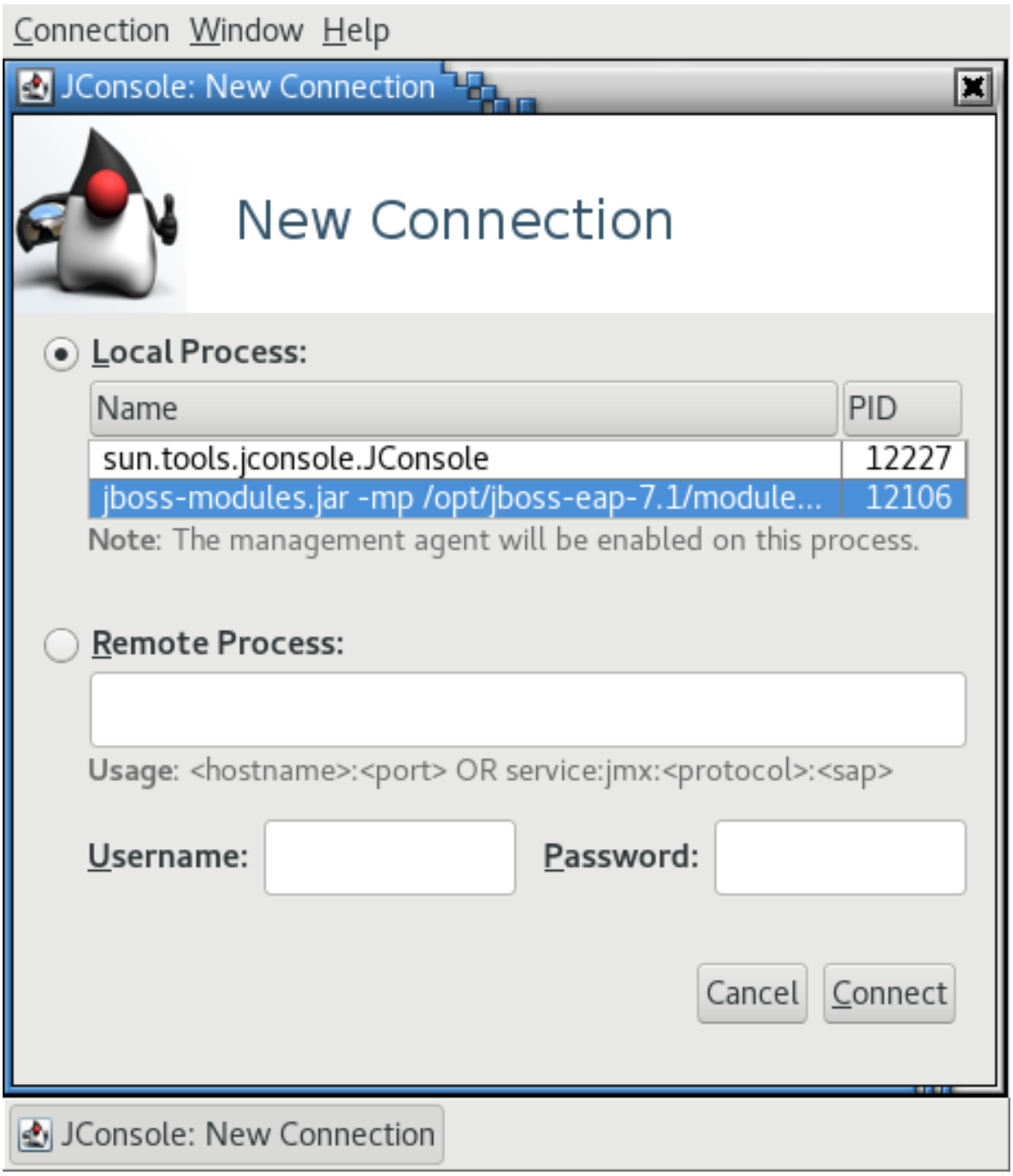
JBoss EAP には事前設定された JConsole ラッパースクリプトがバンドルされています。このラッパースクリプトを使用すると、必要なライブラリーすべてがクラスパスに追加され、さらに JConsole 内から JBoss EAP 管理 CLI へアクセスできるようになります。

2.2.1. JConsole を使用したローカル JBoss EAP JVM への接続

JConsole と同じマシン上で実行している JBoss EAP JVM に接続するには、以下を行います。

1. **EAP_HOME/bin** で **jconsole** スクリプトを実行します。
2. **Local Process** で、監視する JBoss EAP JVM プロセスを選択します。
   - スタンドアロン JBoss EAP サーバーの場合は、JBoss EAP の JVM プロセスは 1 つになります。

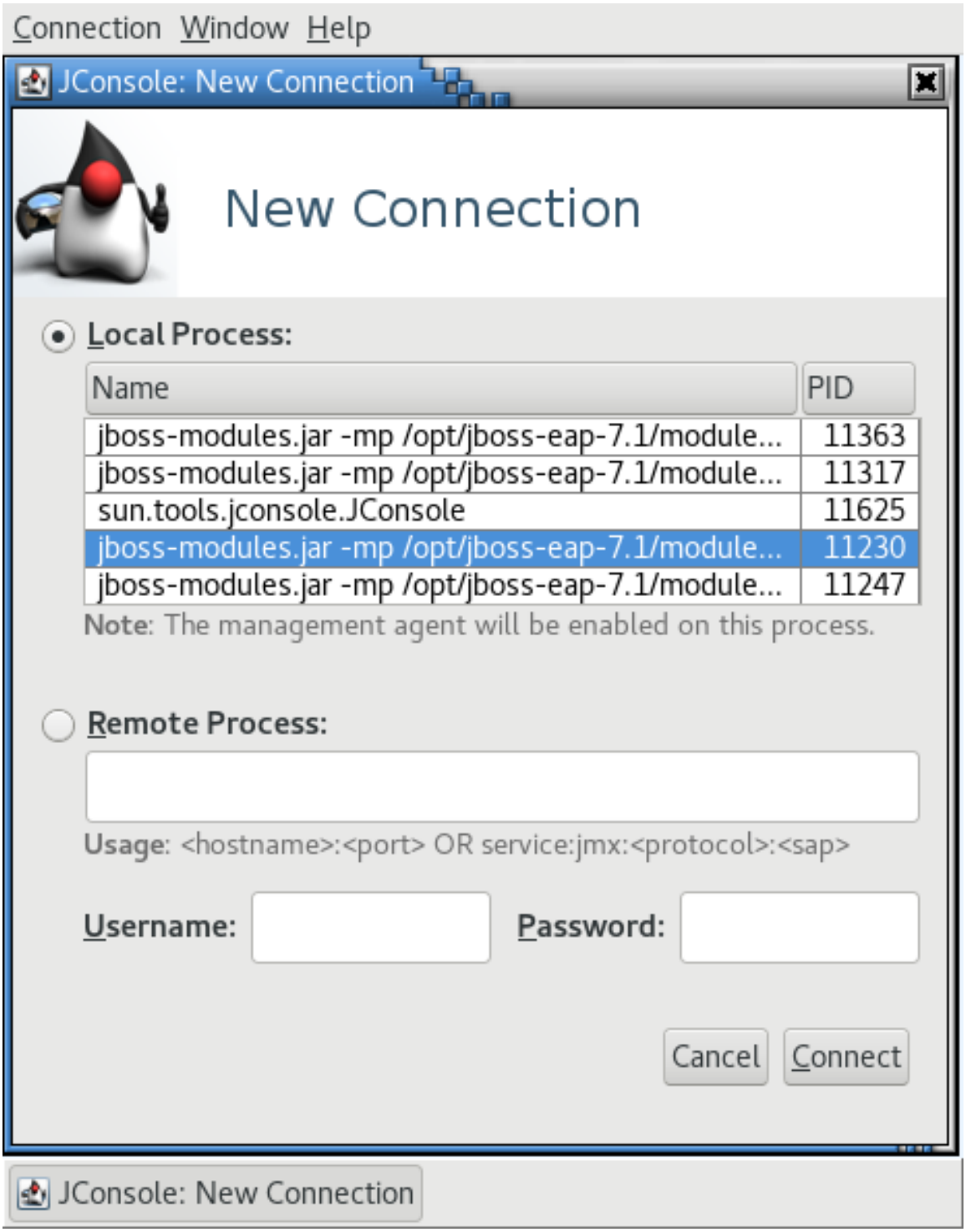
     図2.1 JConsole のローカルスタンドアロン JBoss EAP サーバーの JVM



   - JBoss EAP の管理対象ドメインホストには、ホストコントローラー、JVM プロセス、プロセスコントローラーの JVM プロセス、およびホスト上の各 JBoss EAP サーバーの JVM プ

ロセスなど、接続できる複数の JVM プロセスがあります。JVM 引数を確認すると、接続した JVM を判断できます。

図2.2 JConsole の管理対象ドメイン JBoss EAP の JVM



3. **Connect** をクリックします。

2.2.2. JConsole を使用したリモート JBoss EAP JVM への接続

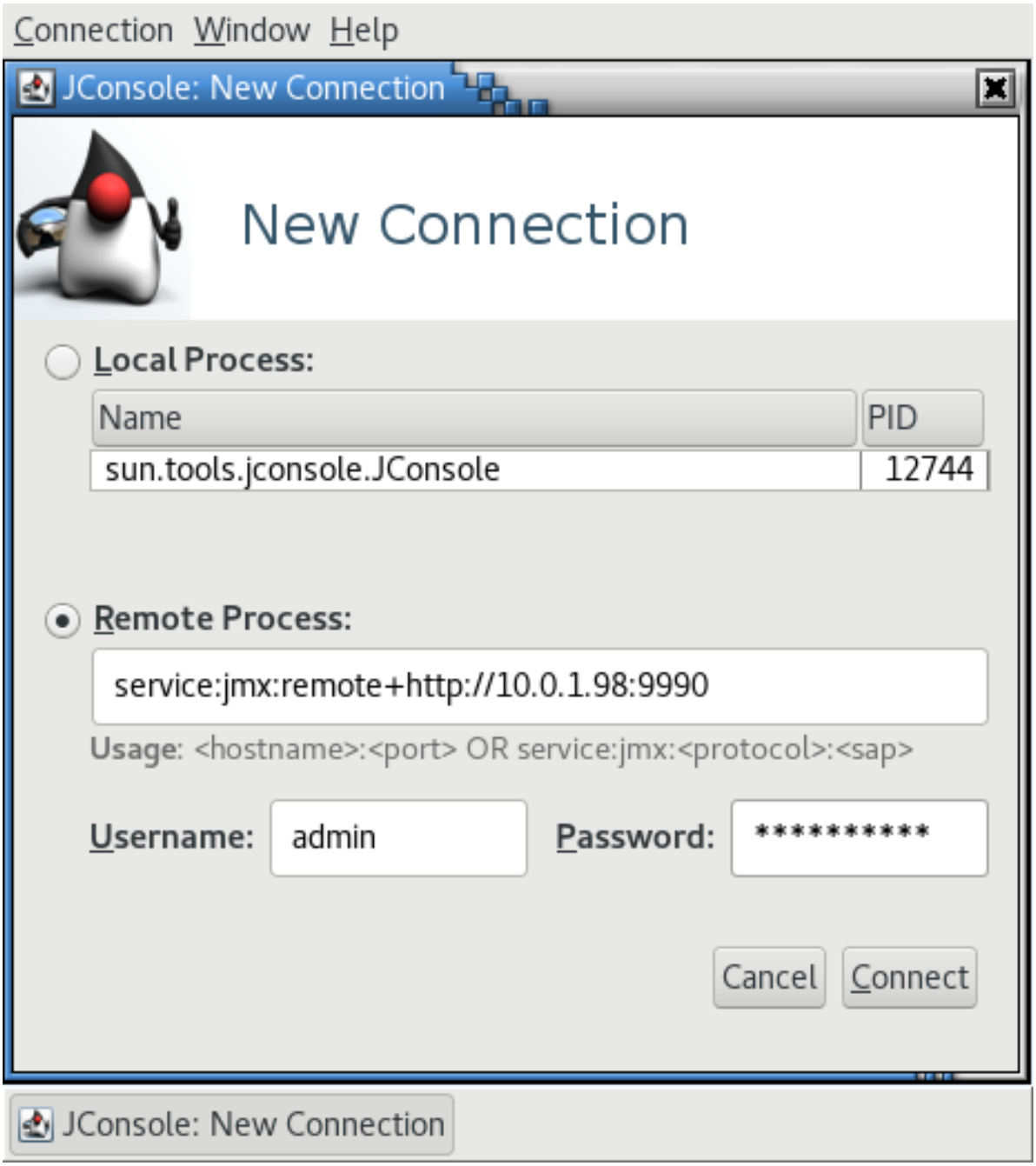
要件

- リモート監視接続のために JBoss EAP を設定します。

- JBoss EAP の ZIP インストールをローカルマシンにダウンロードし、展開します。詳細は、JBoss EAP **『インストールガイド』** を参照してください。

1. **EAP_HOME/bin** で **jconsole** スクリプトを実行します。
2. **Remote Process** で、監視するリモート JBoss EAP JVM プロセスの URI を挿入します。使用する URI については「リモート監視接続のための JBoss EAP の設定」の手順を参照してください。

図2.3 JConsole のリモート JBoss EAP JVM



3. 必ず、監視接続のユーザー名およびパスワードを提供してください。
4. **Connect** をクリックします。

2.3. JAVA VISUALVM

Java VisualVM は Oracle JDK に含まれ、**JAVA_HOME/bin/jvisualvm** にあります。Oracle JDK を使用していない場合、VisualVM は VisualVM の Web サイト からダウンロードすることもできます。VisualVM は IBM JDK とは動作しないため注意してください。

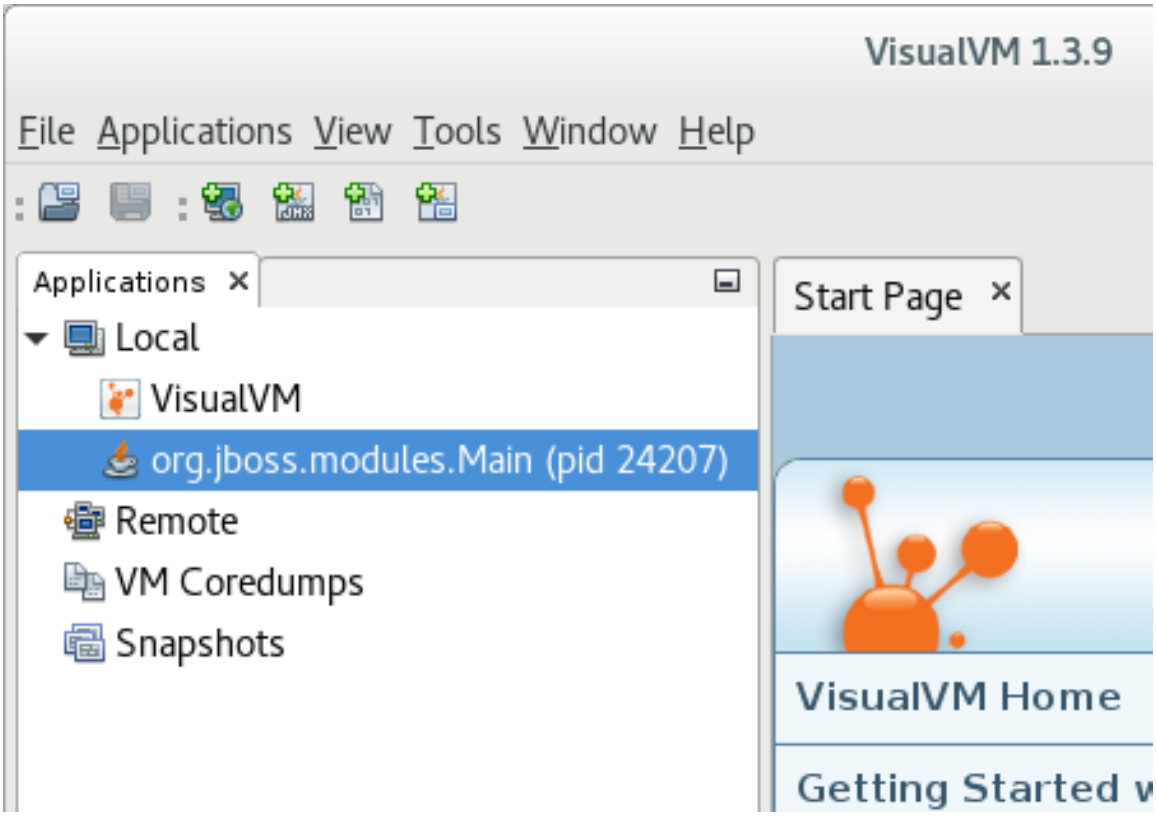
以下のセクションでは、VisualVM を使用してローカルまたはリモート JBoss EAP JVM に接続する手順を取り上げます。VisualVM の使用に関するその他の情報は、VisualVM のドキュメントを参照してください。

2.3.1. VisualVM を使用したローカル JBoss EAP JVM への接続

VisualVM と同じマシン上で実行している JBoss EAP JVM に接続するには、以下を行います。

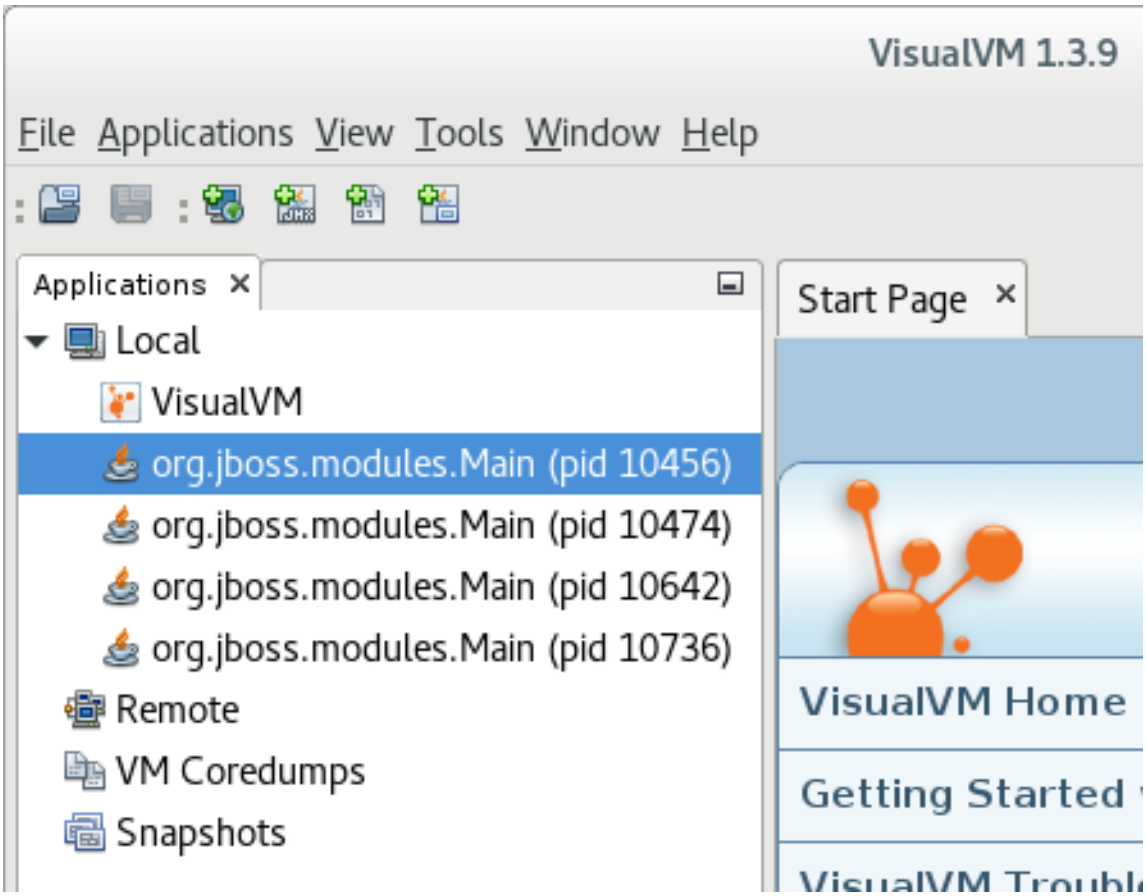
1. VisualVM を開き、VisualVM ウインドウの左側にある **Applications** ペインを見つけます。
2. **Local** で、監視する JBoss EAP JVM プロセスをダブルクリックします。
   - スタンドアロン JBoss EAP サーバーの場合は、JBoss EAP の JVM プロセスは 1 つになります。

     図2.4 VisualVM のローカルスタンドアロン JBoss EAP サーバーの JVM

   - JBoss EAP の管理対象ドメインホストには、ホストコントローラー、JVM プロセス、プロセスコントローラーの JVM プロセス、およびホスト上の各 JBoss EAP サーバーの JVM プロセスなど、接続できる複数の JVM プロセスがあります。JVM 引数を確認すると、接続した JVM を判断できます。

図2.5 VisualVM のローカル管理対象ドメイン JBoss EAP の JVM



2.3.2. VisualVM を使用したリモート JBoss EAP JVM への接続

要件

- リモート監視接続のために JBoss EAP を設定します。
- JBoss EAP の ZIP インストールをローカルマシンにダウンロードし、展開します。詳細は、JBoss EAP **『インストールガイド』** を参照してください。

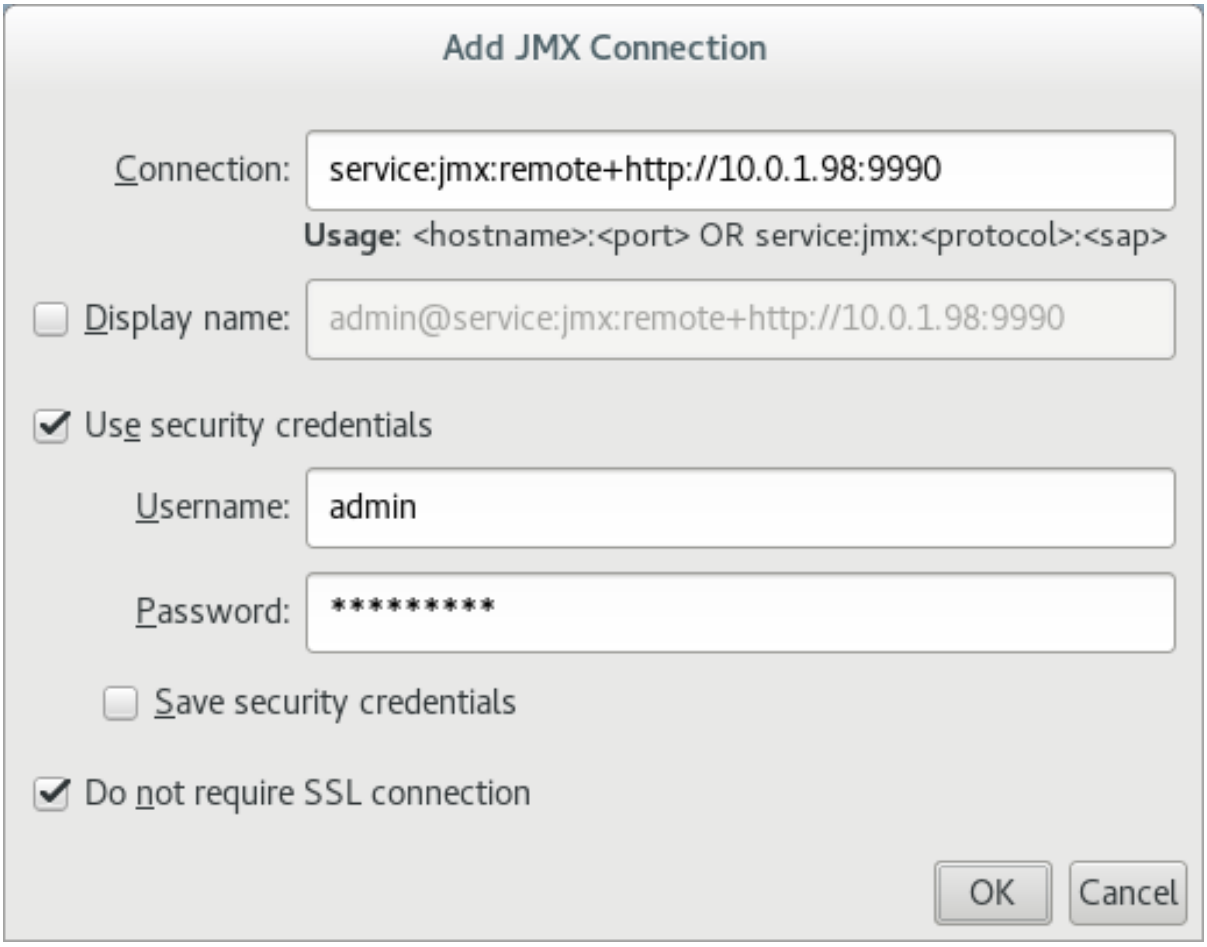
1. JBoss EAP JVM をリモートで監視するには、必要な JBoss EAP ライブラリーをクラスパスに追加する必要があります。ローカルマシンで必要なライブラリーの引数を用いて VisualVM を起動します。例を以下に示します。

```
$ visualvm -cp:a EAP_HOME/bin/client/jboss-cli-client.jar  -J-Dmodule.path=EAP_HOME/modules
```

2. **File** メニューで **Add JMX Connection** を選択します。
3. リモート JBoss EAP JVM の詳細を入力します。
   - 監視するリモート JBoss EAP JVM プロセスの URI を**Connection** フィールドに挿入します。使用する URI については「リモート監視接続のための JBoss EAP の設定」の手順を参照してください。
   - **Use security credentials** チェックボックスを選択し、監視接続のユーザー名およびパスワードを入力します。

- SSL 接続を使用していない場合は、**Do not require SSL connection** チェックボックスを選択します。

図2.6 VisualVM のリモート JBoss EAP JVM



4. **OK** をクリックします。
5. VisualVM ウインドウの左側にある **Applications** ペインで、リモートホストの下にある JMX 項目をダブルクリックし、監視接続を開きます。

第3章 パフォーマンス問題の分析

3.1. ガベッジコレクションロギングの有効化

Java のパフォーマンス問題、特にメモリー使用量に関連する問題をトラブルシューティングする場合、ガベッジコレクションのログを分析すると役立つことがあります。

ガベッジコレクションのロギングを有効にしても、ログファイルへの書き込みによって追加のディスク I/O アクティビティーが発生する以外に、サーバーのパフォーマンスに著しく影響することはありません。

ガベッジコレクションのロギングは、OpenJDK または Oracle JDK で実行しているスタンドアロン JBoss EAP サーバーではすでにデフォルトで有効になっています。JBoss EAP 管理対象ドメインの場合、ガベッジコレクションのロギングはホストコントローラー、プロセスコントローラー、または個別の JBoss EAP サーバーに対して有効にできます。

1. ご使用の JDK でガベッジコレクションのロギングを有効にするために正しい JVM オプションを使用してください。以下のオプションのパスはログを作成する場所に置き換えてください。

注記

Red Hat カスタマーポータルでは、最適な JVM 設定の生成をお手伝いする JVM Options Configuration Tool を使用できます。

- OpenJDK または Oracle JDK の場合

```
-verbose:gc -Xloggc:/path/to/gc.log -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -
XX:+PrintGCApplicationStoppedTime
```

- IBM JDK の場合

```
-verbose:gc -Xverbosegclog:/path/to/gc.log
```

2. ガベッジコレクションの JVM オプションを JBoss EAP サーバーに適用します。
JVM オプションを適用する方法は、JBoss EAP **『設定ガイド』** を参照して ください。

3.2. JAVA ヒープダンプ

Java ヒープダンプは、特定時に作成された JVM ヒープのスナップショットです。ヒープダンプの作成および分析は、Java アプリケーションの問題の分析やトラブルシューティングに役立つことがあります。

JBoss EAP プロセスの Java ヒープダンプの作成および分析方法は、使用している JDK に応じて異なります。ここでは、Oracle JDK、OpenJDK、および IBM JDK での一般的な方法を取り上げます。

3.2.1. ヒープダンプの作成

3.2.1.1. OpenJDK および Oracle JDK

オンデマンドヒープダンプの作成

jcmd コマンドを使用すると、OpenJDK または Oracle JDK で実行している JBoss EAP のオンデマンドヒープダンプを作成できます。

1. ヒープダンプを作成する JVM のプロセス ID を判断します。
2. 以下のコマンドでヒープダンプを作成します。

```
$ jcmd JAVA_PID GC.heap_dump -all=true FILENAME.hprof
```

これにより、ヒープダンプファイルが HPROF 形式で作成され、通常 **EAP_HOME** または **EAP_HOME/bin** に格納されます。代わりに、別のディレクトリーへのファイルパスを指定することもできます。

OutOfMemoryError での自動的なヒープダンプの作成

-XX:+HeapDumpOnOutOfMemoryError JVM オプションを使用すると、**OutOfMemoryError** 例外の発生時に自動的にヒープダンプを作成することができます。

これにより、ヒープダンプファイルが HPROF 形式で作成され、通常 **EAP_HOME** または **EAP_HOME/bin** に格納されます。代わりに、**-XX:HeapDumpPath=/path/** を使用してヒープダンプのカスタムパスを設定することもできます。**-XX:HeapDumpPath=/path/filename.hprof** のように **-XX:HeapDumpPath** を使用してファイル名を指定すると、ヒープダンプはお互いに上書きされます。

JVM オプションを適用する方法は、JBoss EAP **『設定ガイド』** を参照して ください。

3.2.1.2. IBM JDK

IBM JDK を使用している場合、ヒープダンプは **OutOfMemoryError** の発生時に自動的に生成されます。

IBM JDK のヒープダンプは、portable heap dump (PHD) 形式ファイルとして **/tmp/** ディレクトリーに保存されます。

3.2.2. ヒープダンプの分析

ヒープダンプ分析ツール

ヒープダンプを解析し、問題の特定を手助けするツールは多く存在します。Red Hat は、HPROF または PHD 形式でフォーマットされたヒープダンプを解析できる Eclipse Memory Analyzer ツール (MAT) の使用を推奨します。

Eclipse MAT の使用に関する詳細は、Eclipse MAT のドキュメント を参照してください。

ヒープダンプ解析のヒント

ヒープパフォーマンスの問題の原因が明白であることもありますが、アプリケーションのコードや、**OutOfMemoryError** のような問題を引き起こす状況を理解する必要があることもあります。これにより、メモリーリークの問題であるかまたはヒープのサイズが小さすぎるのかを特定することができます。

メモリー使用率の問題特定に推奨される方法には以下が含まれます。

- メモリーを大量に消費している単一のオブジェクトが見つからない場合、クラスでグループ化して、多くの小さなオブジェクトが大量のメモリーを消費していないか確認します。

- 最もメモリーを使用しているのが 1 つのスレッドであるかを確認します。これには、**OutOfMemoryError** によって引き起こされたヒープダンプが指定の **Xmx** 最大ヒープサイズよりも大幅に小さいかどうかを確認するとよいでしょう。

- 通常の最大ヒープサイズを一時的に 2 倍にすると、メモリーリークをより検出しやすくなります。**OutOfMemoryError** の発生時、メモリーリークに関連するオブジェクトのサイズはヒープのサイズの約半分になります。

メモリー問題の原因を特定したら、ガベッジコレクションのルートからパスを確認し、オブジェクトが何によって維持されているかを確認します。

3.3. JAVA スレッドによる CPU 高使用率の特定

注記

Red Hat Enterprise Linux または Solaris 上で JBoss EAP を使用している場合は、Red Hat カスタマーポータル上で JVMPeg ラボツールを利用すると、CPU の高使用率を特定するための Java スレッド情報の収集および分析が容易になります。以下の手順を使用する代わりに JVMPeg ラボツールの使用手順 に従います。

OpenJDK および Oracle JDK 環境では、**jstack** ユーティリティーを使用して Java スレッドの分析情報を取得できます。

1. CPU の使用率が高い Java プロセスのプロセス ID を特定します。
   使用率が高いプロセスのスレッドごとの CPU データを取得すると便利なこともあります。このデータを取得するには、Red Hat Enterprise Linux システム上で **top -H** コマンドを使用します。

2. **jstack** ユーティリティーを使用して、Java プロセスのスタックダンプを作成します。Linux および Solaris の例を以下に示します。

   ```
   jstack -l JAVA_PROCESS_ID > high-cpu-tdump.out
   ```

   複数のダンプを周期的に作成し、一定期間での変更および傾向を確認する必要があることがあります。

3. スタックダンプを分析します。Thread Dump Analyzer (TDA) などのツールを使用できます。

第4章 JVM の調整

アプリケーションおよび JBoss EAP 環境に対して最良の JVM オプションを設定することは、パフォーマンスを調整する上で最も基本的なことの 1 つです。本章では、一般的な JVM オプションの設定について説明します。

注記

本章にリストされている JVM オプションの多くは、Red Hat カスタマーポータルの JVM Options Configuration Tool を使用して簡単に生成できます。

JVM オプションを適用する方法は、JBoss EAP **『設定ガイド』** を参照して ください。

4.1. 固定ヒープサイズの設定

メモリー不足エラーが発生しないようにするため、適切なヒープサイズを設定する必要があります。

-Xms オプションは初期ヒープサイズを設定し、 **-Xmx** は最大ヒープサイズを設定します。実稼働環境では、初期および最大ヒープサイズを同じサイズに設定し、ヒープサイズを固定および事前割り当てすることが推奨されます。

たとえば、以下のオプションは 2048 MB のヒープサイズを設定します。

```
-Xms2048M -Xmx2048M
```

開発環境の負荷下でアプリケーションをテストし、最大メモリー使用率を判断することが推奨されます。実稼働でのヒープサイズは、オーバーヘッドを考慮して、テストした最大ヒープサイズよりも 25% 以上高くする必要があります。

4.2. ガベッジコレクターの設定

並行ガベッジコレクターはスループットガベッジコレクターとも呼ばれ、サーバークラスマシンの Java 8 でのデフォルトガベッジコレクター ですが、Red Hat は Java 9 からデフォルトになる予定の G1 ガベッジコレクターの使用を推奨します。一般的に、G1 ガベッジコレクターのパフォーマンスはほとんどの場合で CMS および平行ガベッジコレクターを上回ります。

G1 コレクターを有効にするには、以下の JVM オプションを使用します。

```
-XX:+UseG1GC
```

ガベッジコレクションのロギングオプション
ガベッジコレクションのロギングは、スタンドアロン JBoss EAP サーバーではデフォルトで有効になっています。JBoss EAP 管理対象ドメインでガベッジコレクションのロギングを有効にする場合は、「 「ガベッジコレクションロギングの有効化」 」を参照してください。

4.3. ラージページの有効化

JBoss EAP JVM のラージページを有効にすると、ページがメモリーでロックされ、通常のメモリーのようにディスクへのスワップ処理を行うことができません。

特にメモリー集中型のアプリケーションでは、ヒープをディスクへページまたはスワップできず、常にラージページを利用できることがラージページを使用する場合の利点となります。

ラージページを使用する場合の難点の 1 つは、システムで実行されている別のプロセスがメモリーに即時アクセスできない可能性があり、これらのプロセスに対して過剰なページングが行われる可能性があることです。

他のパフォーマンス設定の変更と同様に、テスト環境で変更の影響をテストすることが推奨されます。

1. プロセスがラージページを使用できるようにオペレーティングシステムが設定されている必要があります。
   - Red Hat Enterprise Linux システムでは、**HugeTLB** ページを明示的に設定して、確実に JBoss EAP のプロセスがラージページにアクセスできるようにする必要があります。Red Hat Enterprise Linux のメモリーオプションの設定に関する詳細は、Red Hat Enterprise Linux『パフォーマンスチューニングガイド』の「**メモリー**」の章を参照してください。
   - Windows サーバーシステムでは、JBoss EAP を実行しているユーザーにラージページの特権が割り当てられている必要があります。
     1. **コントロールパネル → 管理ツール → ローカルセキュリティポリシー** と選択します。
     2. **ローカルポリシー → ユーザー権利の割り当て** と選択します。
     3. **メモリ内のページのロック** をダブルクリックします。
     4. ラージページを使用する Windows Server ユーザーおよびユーザーグループを追加します。
     5. マシンを再起動します。
2. ラージページのサポートを有効または無効にします。
   - 明示的に JBoss EAP JVM のラージページのサポートを有効にするには、以下の JVM オプションを使用します。

     ```
     -XX:+UseLargePages
     ```

   - 明示的に JBoss EAP JVM のラージページのサポートを無効にするには、以下の JVM オプションを使用します。

     ```
     -XX:-UseLargePages
     ```

3. JBoss EAP の起動時に、メモリーの確保に関する警告がないことを確認してください。
   - Red Hat Enterprise Linux では、以下のようなエラーが表示されます。

     ```
     OpenJDK 64-Bit Server VM warning: Failed to reserve shared memory. (error = 1)
     ```

   - Windows Server では、以下のようなエラーが表示されます。

     ```
     Java HotSpot(TM) 64-Bit Server VM warning: JVM cannot use large page memory
     because it does not have enough privilege to lock pages in memory.
     ```

   警告が表示された場合、オペレーティングシステムと JVM オプションが正しく設定されていることを確認してください。

詳細は、ラージページの Java サポートに関する Oracle のドキュメント を参照してください。

4.4. アグレッシブな最適化の有効化

アグレッシブな最適化 (AggressiveOpts) の JVM オプションを使用すると、ご使用の環境でパフォーマンスを改善できます。このオプションは、今後の Java リリースでデフォルトとなる予定の Java 最適化機能を有効にします。

AggressiveOpts を有効にするには、以下の JVM オプションを使用します。

```
-XX:+AggressiveOpts
```

4.5. ULIMITS の設定

Red Hat Enterprise Linux および Solaris プラットフォームでは、JBoss EAP JVM プロセスに適切な **ulimit** 値を設定する必要があります。「ソフトな」**ulimit** の場合は一時的にその値を超えることが許されますが、「ハードな」**ulimit** はリソース使用率の厳格な限度になります。適切な **ulimit** の値は、環境とアプリケーションによって異なります。

重要

IBM JDK を使用している場合、IBM JDK は JVM プロセスによって使用されるオープンファイルの最大数をソフトな制限として使用することに注意してください。Red Hat Enterprise Linux では、ソフトな制限のデフォルト値 (**1024**) は、IBM JDK を使用する JBoss EAP プロセスでは低すぎると見なされます。

JBoss EAP プロセスに適用される制限が低すぎると、JBoss EAP の起動時に以下のような警告が表示されます。

```
WARN  [org.jboss.as.warn.fd-limit] (main) WFLYSRV0071: The operating system has limited the number of open files to 1024 for this process; a value of at least 4096 is recommended.
```

現在の **ulimit** 値を確認するには、以下のコマンドを使用します。

- ソフトな **ulimit** 値の場合:

```
ulimit -Sa
```

- ハードな **ulimit** 値の場合:

```
ulimit -Ha
```

ulimit をオープンファイルの最大数に設定するには、適用する数を指定して以下のコマンドを使用します。

- オープンファイルの最大数にソフト **ulimit** を設定する場合:

```
ulimit -Sn 4096
```

- オープンファイルの最大数にハード **ulimit** を設定する場合:

```
ulimit -Hn 4096
```

注記

ulimit の設定が効果的であるようにするため、実稼働システムではソフトな制限とハードな制限に同じ値を設定することが推奨されます。

設定ファイルを使用した **ulimit** 値の設定に関する詳細は、カスタマーポータルの「ulimit 値を設定する」を参照してください。

4.6. ホストコントローラーおよびプロセスコントローラー JVM の調整

JBoss EAP 管理対象ドメインホストのホストコントローラーとプロセスコントローラーには個別の JVM があります。ホストコントローラー と プロセスコントローラーのロールに関する詳細は、JBoss EAP **『設定ガイド』** を参照してください。

ホストコントローラーとプロセスコントローラーの JVM 設定を調整できますが、大きな管理対象ドメイン環境でも、ホストコントローラーおよびプロセスコントローラーのデフォルトの JVM 設定で十分です。

ホストコントローラーおよびプロセスコントローラーのデフォルトの JVM 設定は、最大 20 個の JBoss EAP ホストが各自 10 個の JBoss EAP サーバーを実行する、JBoss EAP サーバーの合計ドメインサイズが 200 の管理対象ドメインサイズでテストされています。

大型の管理対象ドメインで問題が発生した場合、ご使用の環境で ホストコントローラーまたはプロセスコントローラー JVM を監視 し、ヒープサイズなどの JVM オプションの適切な値を判断する必要がある場合があります。

第5章 EJB サブシステムの調整

JBoss EAP は Jakarta Enterprise Bean をキャッシュして初期化時間を節約できます。これは、bean プールを使用して行います。

JBoss EAP では、bean インスタンスプール と bean スレッドプール の 2 種類の bean プールを調整できます。

適切な bean プールサイズは、ご使用の環境とアプリケーションによって異なります。予想される実際の条件を模倣する開発環境で、さまざまな bean プールサイズを試し、ストレステストを実行することが推奨されます。

5.1. BEAN インスタンスプール

bean インスタンスプールは、ステートレスセッション bean (SLSB) およびメッセージ駆動型 bean (MDB) に使用されます。デフォルトでは、SLSB はインスタンスプールの **default-slsb-instance-pool** を使用し、MDB はインスタンスプールの **default-mdb-instance-pool** を使用します。

bean インスタンスプールのサイズによって、一度に作成可能な特定の EJB のインスタンス数が制限されます。特定の EJB の空きがない場合、クライアントはインスタンスが利用可能になるまでブロックおよび待機します。プールの **timeout** 属性に設定された時間内にクライアントがインスタンスを取得できないと、例外が発生します。

bean インスタンスプールのサイズは、**derive-size** または **max-pool-size** のいずれかを使用して設定されます。**derive-size** 属性を使用する場合、以下の値の 1 つを使用してプールサイズを設定できます。

- **from-worker-pools**: 最大プールサイズはシステム上で設定されたワーカープールすべての合計スレッドのサイズから派生することを意味します。
- **from-cpu-count**: 最大プールサイズはシステム上で利用可能なプロセッサーの合計数から派生することを意味します。必ずしも 1 対 1 のマッピングではなく、他の要因によって拡張される可能性があることに注意してください。

derive-size が未定義の場合、**max-pool-size** の値が bean インスタンスプールのサイズに使用されます。

注記

derive-size 属性は、**max-pool-size** に指定された値をすべて上書きします。**max-pool-size** 値を有効にするには、**derive-size** を未定義にする必要があります。

特定のインスタンスプールを使用するよう EJB を設定することができます。これにより、各 EJB タイプが使用できるインスタンスをより細かく制御できます。

5.1.1. Bean インスタンスプールの作成

ここでは、管理 CLI を使用して新たに bean インスタンスプールを作成する方法を説明します。また、管理コンソールを使用して bean インスタンスプールを設定することもできます。 この場合、**Configuration** タブで **EJB** サブシステムを選択し、**Bean Pool** タブを選択します。

新しいインスタンスプールを作成するには、以下のコマンドの 1 つを使用します。

- 派生した最大プールサイズで bean インスタンスプールを作成する場合:

```
/subsystem=ejb3/strict-max-bean-instance-pool=POOL_NAME:add(derive-size=DERIVE_OPTION,timeout-unit=TIMEOUT_UNIT,timeout=TIMEOUT_VALUE)
```

以下の例は、CPU 数から派生した最大サイズと、2 分のタイムアウトを指定して、**my_derived_pool** という名前の bean インスタンスプールを作成します。

```
/subsystem=ejb3/strict-max-bean-instance-pool=my_derived_pool:add(derive-size=from-cpu-count,timeout-unit=MINUTES,timeout=2)
```

- 明示的な最大プールサイズで bean インスタンスプールを作成する場合:

```
/subsystem=ejb3/strict-max-bean-instance-pool=POOL_NAME:add(max-pool-size=POOL_SIZE,timeout-unit=TIMEOUT_UNIT,timeout=TIMEOUT_VALUE)
```

以下の例は、30 個の最大インスタンスと、30 秒のタイムアウトを指定して、**my_pool** という名前の bean インスタンスプールを作成します。

```
/subsystem=ejb3/strict-max-bean-instance-pool=my_pool:add(max-pool-size=30,timeout-unit=SECONDS,timeout=30)
```

5.1.2. Bean が使用するインスタンスプールの指定

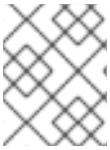
特定の bean が使用する特定のインスタンスプールを設定するには、**@org.jboss.ejb3.annotation.Pool** アノテーションを使用するか、bean の **jboss-ejb3.xml** デプロイメント記述子を編集します。詳細は、『**Developing EJB Applications**』の「**jboss-ejb3.xml** Deployment Descriptor Reference」を参照してください。

5.1.3. デフォルト Bean インスタンスプールの無効化

デフォルトの bean インスタンスプールは無効にすることができます。無効にすると、EJB はデフォルトでインスタンスプールを何も使用しません。代わりに、スレッドが EJB でメソッドを呼び出す必要があるときに新しい EJB インスタンスが作成されます。これは、作成された EJB インスタンスの数を制限したい場合に便利です。

デフォルトの beanインスタンスプールを無効にするには、以下の管理 CLI コマンドを使用します。

```
/subsystem=ejb3:undefine-attribute(name=default-slsb-instance-pool)
```



注記

bean が 特定の bean インスタンスプールを使用するよう設定されている場合、デフォルトのインスタンスプールを無効にしても、bean が使用するプールは影響を受けません。

5.2. BEAN スレッドプール

デフォルトでは、**default** という名前の bean スレッドプールは非同期 EJB 呼び出しと EJB タイマーに使用されます。

注記

JBoss EAP 7 以降のリリースでは、リモート EJB リクエストはデフォルトで **io** サブシステムに定義されたワーカーで処理されます。

必要な場合は、各 EJB サービスが異なる bean スレッドプールを使用するよう設定することができます。これは、各サービスによる bean スレッドプールへのアクセスをより細かく制御する場合に便利です。

適切なスレッドプールサイズを判断するとき、想定される同時リクエストが 1 度に処理される数を考慮してください。

5.2.1. Bean スレッドプールの作成

ここでは、管理 CLI を使用して新たに bean スレッドプールを作成する方法を説明します。また、管理コンソールを使用して bean スレッドプールを設定することもできます。 この場合、**Configuration** タブで **EJB** サブシステムを選択し、左側のメニューで **Container → Thread Pool** とを選択します。

新しいスレッドプールを作成するには、以下のコマンドを使用します。

```
/subsystem=ejb3/thread-pool=POOL_NAME:add(max-threads=MAX_THREADS)
```

以下の例は、最大 30 個のスレッドを持つ **my_thread_pool** という名前の bean スレッドプールを作成します。

```
/subsystem=ejb3/thread-pool=my_thread_pool:add(max-threads=30)
```

5.2.2. 特定の Bean スレッドプールを使用するよう EJB サービスを設定

特定の bean スレッドプールを使用するよう、EJB3 非同期呼び出しサービスおよびタイマーサービスをそれぞれ設定することができます。デフォルトでは、これらのサービスは **default** bean スレッドプールを使用します。

ここでは、管理 CLI を使用して、特定の beanスレッドプールを使用するよう上記の EJB サービスを設定する方法を説明します。また、管理コンソールを使用してこれらのサービスを設定することもできます。 この場合、**Configuration** タブで **EJB** サブシステムを指定し、**Services** タブを選択して該当するサービスを選択します。

特定の bean スレッドプールを使用するよう EJB サービスを設定するには、以下のコマンドを使用します。

```
/subsystem=ejb3/service=SERVICE_NAME:write-attribute(name=thread-pool-
name,value=THREAD_POOL_NAME)
```

SERVICE_NAME は、以下のように設定する EJB サービスに置き換えてください。

- EJB3 非同期呼び出しサービスの場合は **async**
- EJB3 タイマーサービスの場合は **timer-service**

以下の例は、**my_thread_pool** という名前の bean スレッドプールを使用するよう EJB3 非同期サービスを設定します。

```
/subsystem=ejb3/service=async:write-attribute(name=thread-pool-name,value=my_thread_pool)
```

5.3. EJB サブシステムの調整の必要性を示す例外

- **ステートレス EJB インスタンスプールが小さすぎる、またはタイムアウトが短すぎる。**

```
javax.ejb.EJBException: JBAS014516: Failed to acquire a permit within 20 SECONDS
    at org.jboss.as.ejb3.pool.strictmax.StrictMaxPool.get(StrictMaxPool.java:109)
```

Bean インスタンスプール を参照してください。

- **EJB スレッドプールが小さすぎる、または EJB の処理時間が呼び出しタイムアウトよりも長い。**

```
java.util.concurrent.TimeoutException: No invocation response received in 300000
milliseconds
```

Bean スレッドプール を参照してください。

第6章 データソースおよびリソースアダプターの調整

接続プールは、リレーショナルデータベースやリソースアダプターなどのデータソースを使用する環境のパフォーマンスを最適化するために JBoss EAP が使用するツールです。

データソースおよびリソースアダプター接続のリソースを割り当てまたは割り当て解除することは、時間やシステムリソースのコストが大変高くなります。接続プールは、アプリケーションが使用できる接続の「プール」を作成して、接続のコストを削減します。

パフォーマンスを最適化するために接続プールを設定する前に、負荷がかかった状態で データソースプール統計 または リソースアダプターの統計 を監視し、ご使用の環境に適した設定を判断する必要があります。

6.1. プール統計の監視

6.1.1. データソースの統計

データソースの統計収集が 有効化 されている場合、データソースの ランタイム統計を表示 できます。

6.1.1.1. データソース統計の有効化

データソース統計は、デフォルトでは有効に**なっていません**。データソース統計の収集は、 管理 CLI または 管理コンソール を使用して有効にできます。

管理 CLI を使用したデータソース統計の有効化
以下の管理 CLI コマンドは、**ExampleDS** データソースの統計の収集を有効にします。

注記

管理対象ドメインでは、このコマンドの前に **/profile=PROFILE_NAME** を付けます。

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

変更を反映するためにサーバーをリロードします。

管理コンソール使用したデータソース統計の有効化
以下の手順にしたがって管理コンソールを使用し、統計の収集を有効にします。

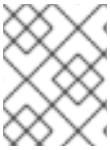
1. **Configuration → Subsystems → Datasources & Drivers → Datasources** と選択します。
2. データソースを選択し、**表示** をクリックします。
3. **Attributes** タブ下の **Edit** をクリックします。
4. **Statistics enabled?** フィールドを **ON** に設定し、**Save** をクリックします。変更の反映にはリロードが必要であることを伝えるポップアップが表示されます。
5. サーバーをリロードします。
   - スタンドアロンサーバーの場合は、ポップアップの **Reload** ボタンをクリックしてサーバーをリロードします。

- 管理対象ドメインの場合は、ボップアップの **Topology** リンクをクリックします。**Topology** タブで該当するサーバーを選択し、**Reload** ドロップダウンオプションを選択してサーバーをリロードします。

6.1.1.2. データソース統計の表示

管理 CLI または 管理コンソール を使用してデータソースのランタイム統計を表示できます。

管理 CLI を使用したデータソース統計の表示
以下の管理 CLI コマンドは、**ExampleDS** データソースのコア**プール**の統計を取得します。



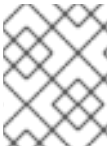
注記

管理対象ドメインでは、コマンドの前に **/host=HOST_NAME/server=SERVER_NAME** を追加します。

```
/subsystem=datasources/data-source=ExampleDS/statistics=pool:read-resource(include-
runtime=true)
{
    "outcome" => "success",
    "result" => {
        "ActiveCount" => 1,
        "AvailableCount" => 20,
        "AverageBlockingTime" => 0L,
        "AverageCreationTime" => 122L,
        "AverageGetTime" => 128L,
        "AveragePoolTime" => 0L,
        "AverageUsageTime" => 0L,
        "BlockingFailureCount" => 0,
        "CreatedCount" => 1,
        "DestroyedCount" => 0,
        "IdleCount" => 1,
        ...
}
```

以下の管理 CLI コマンドは、**ExampleDS** データソースの **JDBC** の統計を取得します。

```
/subsystem=datasources/data-source=ExampleDS/statistics=jdbc:read-resource(include-
runtime=true)
{
    "outcome" => "success",
    "result" => {
        "PreparedStatementCacheAccessCount" => 0L,
        "PreparedStatementCacheAddCount" => 0L,
        "PreparedStatementCacheCurrentSize" => 0,
        "PreparedStatementCacheDeleteCount" => 0L,
        "PreparedStatementCacheHitCount" => 0L,
        "PreparedStatementCacheMissCount" => 0L,
        "statistics-enabled" => true
    }
}
```



注記

統計はランタイム情報であるため、必ず **include-runtime=true** 引数を指定してください。

利用可能な統計の詳細リストは、「データソースの統計」を参照してください。

管理コンソールを使用したデータソース統計の表示
管理コンソールからデータソースの統計を表示するには、**Runtime** タブで **Datasources** サブシステムを選択し、データソースを選択してから **表示** をクリックします。

利用可能な統計の詳細リストは、「データソースの統計」を参照してください。

6.1.2. リソースアダプターの統計

デプロイされたリソースアダプターのコア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、リソースアダプターの統計の付録 を参照してください。

リソースアダプター統計の有効化
リソースアダプター統計は、デフォルトでは有効になって**いません**。以下の管理 CLI コマンドは、接続ファクトリーが **java:/eis/AcmeConnectionFactory** として JNDI にバインドされた簡単なリソースアダプター **myRA.rar** の統計収集を有効にします。

注記

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

```
/deployment=myRA.rar/subsystem=resource-adapters/statistics=statistics/connection-definitions=java\:\/eis\/AcmeConnectionFactory:write-attribute(name=statistics-enabled,value=true)
```

リソースアダプター統計の表示
リソースアダプター統計は管理 CLI から取得できます。以下の管理 CLI コマンドは、接続ファクトリーが JNDI で **java:/eis/AcmeConnectionFactory** としてバインドされた、リソースアダプター **myRA.rar** の統計を返します。

注記

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

```
deployment=myRA.rar/subsystem=resource-adapters/statistics=statistics/connection-definitions=java\:\/eis\/AcmeConnectionFactory:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "ActiveCount" => "1",
        "AvailableCount" => "20",
        "AverageBlockingTime" => "0",
        "AverageCreationTime" => "0",
        "CreatedCount" => "1",
        "DestroyedCount" => "0",
        "InUseCount" => "0",
```

```
        "MaxCreationTime" => "0",
        "MaxUsedCount" => "1",
        "MaxWaitCount" => "0",
        "MaxWaitTime" => "0",
        "TimedOut" => "0",
        "TotalBlockingTime" => "0",
        "TotalCreationTime" => "0"
    }
}
```

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。

6.2. プールの属性

ここでは、データソースまたはリソースアダプターのパフォーマンスを最適化するために設定できる一部のプール属性の推奨設定について説明します。各属性の設定方法は以下を参照してください。

- データソースプール属性の設定
- リソースアダプタープール属性

最小プールサイズ

min-pool-size 属性は、接続プールの最小サイズを定義します。デフォルトではゼロ個の接続が最小です。**min-pool-size** がゼロの場合、最初のトランザクションの発生時に接続が作成され、プールに置かれます。
min-pool-size が小さすぎると、新しい接続の確立が必要となる可能性があるため、最初のデータベースコマンドの実行中に待ち時間が長くなります。**min-pool-size** が大きすぎると、データソースまたはリソースアダプターへ無駄な接続が発生します。

アクティビティーのない期間が続くと、接続プールは縮小され、**min-pool-size** の値まで縮小される可能性があります。

Red Hat は、**min-pool-size** をアプリケーションに適したオンデマンドスループットを可能にする接続数に設定することを推奨します。

最大プールサイズ

max-pool-size 属性は、接続プールの最大サイズを定義します。これは、アクティブな接続の数を制限し、同時アクティビティーの量も制限するため、重要なパフォーマンスパラメーターとなります。
max-pool-size が小さすぎると、リクエストが不必要にブロックされます。**max-pool-size** が大きすぎると、JBoss EAP の環境、データソース、またはリソースアダプターによって、処理能力を超えた量のリソースが使用されます。

Red Hat は、負荷のかかった状態で **パフォーマンスを監視** した後に測定された許容範囲の **MaxUsedCount** よりも 15% 以上高い max-pool-size を設定することを推奨します。これにより、想定以上の条件でも多少のバッファーを確保できるようにします。

プレフィル

pool-prefill 属性は、JBoss EAP の起動時に JBoss EAP が 最低接続数で接続プールをプレフィルするかどうかを指定します。デフォルト値は **false** です。

pool-prefill を **true** に設定すると、JBoss EAP は起動時により多くのリソースを使用しますが、初期トランザクションの待ち時間が短縮されます。

min-pool-size を最適化した場合、Red Hat は **pool-prefill** を **true** に設定することを推奨します。

厳密な最小値

pool-use-strict-min 属性は、プールの接続数が指定の最小値を下回ることが可能であるかどうかを指定します。
pool-use-strict-min を **true** に設定すると、接続数は一時的に指定の最小値を下回ることができません。デフォルト値は **false** です。

プール接続の最小数が指定されていても、JBoss EAP が接続を閉じるときなどに接続がアイドル状態になり、タイムアウトすると、新しい接続が作成されプールに追加される前に、合計接続数が一時的に最小数を下回ることがあります。

タイムアウト

接続プールに設定可能なタイムアプトオプションは複数ありますが、パフォーマンスのチューニングには **idle-timeout-minutes** が重要です。
idle-timeout-minutes 属性は、接続が閉じられるまでにアイドル状態でいられる最大期間を分単位で指定します。アイドル接続が閉じられると、プールの接続数が指定の最小値まで減少します。

タイムアウトが長いほどより多くのリソースが使用されますが、リクエストはより迅速に対応されます。タイムアウトが短いほどより少ないリソースが使用されますが、リクエストは新しい接続が作成されるまで待機する必要があることがあります。

6.3. プール属性の設定

6.3.1. データソースプール属性の設定

要件

- JDBC ドライバーをインストールします。JBoss EAP **『設定ガイド』** の「 JDBC ドライバー 」を参照してください。
- データソースを作成します。JBoss EAP **『設定ガイド』** の「 データソースの作成 」を参照してください。

管理 CLI または管理コンソールのいずれかを使用してデータソースプール属性を設定できます。

- 管理コンソールを使用する場合は、**Configuration → Subsystems → Datasources & Drivers → Datasources** と選択し、データソースを選択したら **表示** をクリックします。プールのオプションはデータソースの **Pool** タブで設定可能です。タイムアウトのオプションは **Timeouts** タブで設定可能です。
- 管理 CLI を使用する場合は、以下のコマンドを実行します。

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

たとえば、**ExampleDS** データソースの **min-pool-size** 属性の値を5 (5 つの接続) に設定するには、以下のコマンドを実行します。

```
/subsystem=datasources/data-source=ExampleDS/:write-attribute(name=min-pool-size,value=5)
```

6.3.2. リソースアダプタープール属性

要件

- リソースアダプターをデプロイし、接続定義を追加します。JBoss EAP **『設定ガイド』** の「**リソースアダプター の設定** 」を参照してください。

管理 CLI または管理コンソールのいずれかを使用してリソースアダプタープール属性を設定できます。

- 管理コンソールを使用する場合は、**Configuration → Subsystems → Resource Adapters** と選択し、データソースを選択したら **表示** をクリックして、左側のメニューで **Connection Definitions** を選択します。プールのオプションは **Pool** タブで設定可能です。タイムアウトのオプションは **Attributes** タブで設定可能です。

- 管理 CLI を使用する場合は、以下のコマンドを実行します。

```
/subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER_NAME/connection-definitions=CONNECTION_DEFINITION_NAME:write-attribute(name=ATTRIBUTE_NAME,value=ATTRIBUTE_VALUE)
```

たとえば、**my_RA** リソースアダプター **my_CD** 接続定義の **min-pool-size** 属性の値を 5 (5 つの接続) に設定するには、以下のコマンドを使用します。

```
/subsystem=resource-adapters/resource-adapter=my_RA/connection-definitions=my_CD:write-attribute(name=min-pool-size,value=5)
```

第7章 MESSAGING サブシステムの調整

messaging-activemq サブシステムのパフォーマンスチューニングは、JBoss EAP『 **Configuring Messaging** 』の 「 Performance Tuning 」を参照してください。

第8章 LOGGING サブシステムの調整

実稼働環境で、コンソールへのロギングを無効にし, 適切なログレベルを設定し、さらに ログファイルの保存に最も適した場所を指定 すると、JBoss EAP の logging サブシステムのパフォーマンスをさらに向上できます。

logging サブシステムの設定に関する詳細は、JBoss EAP **『設定ガイド』** の ロギングの章 を参照してください。

8.1. コンソールへのロギングの無効化

コンソールのロギングを無効にすると JBoss EAP のパフォーマンスを向上できます。ログをコンソールへ出力することは開発環境およびテスト環境では便利ですが、実稼働環境ではほとんどの場合で必要ありません。JBoss EAP のルートロガーには、**standalone-full-ha** 以外のデフォルトのスタンドアロンサーバープロファイルすべてのコンソールログハンドラーが含まれています。デフォルトの管理対象ドメインプロファイルにはコンソールハンドラーが含まれていません。

デフォルトのコンソールハンドラーをルートロガーから削除するには、以下の管理 CLI コマンドを使用します。

```
/subsystem=logging/root-logger=ROOT:remove-handler(name=CONSOLE)
```

8.2. ログレベルの設定

理想のパフォーマンスを実現するには、実稼働環境のログレベルを適切に設定する必要があります。たとえば、**INFO** または **DEBUG** レベルが開発環境またはテスト環境で適切である場合、実稼働環境ではほとんどの場合でログレベルを **WARN** や **ERROR** などのより高いレベルに設定します。

異なるログハンドラーのログレベルの設定については、JBoss EAP『設定 **ガイド』の「ログハンドラーの 設定** 」を参照してください。

8.3. ログファイルの場所設定

ログファイルの保存場所によってはパフォーマンスの問題を引き起こす可能性があることに注意してください。I/O スループットが不十分なファイルシステムまたはディスク設定にログを保存する場合、プラットフォーム全体のパフォーマンスに影響する可能性があります。

ロギングが JBoss EAP パフォーマンスに影響しないようにするには、ログの場所を十分な領域がある高パフォーマンスな専用ディスクに設定することが推奨されます。

異なるログハンドラーのログファイルの場所設定については、JBoss EAP **『設定ガイド』の「ログハンドラーの 設定** 」を参照してください。

第9章 UNDERTOW サブシステムの調整

JBoss EAP 7 で導入された非ブロッキング I/O **undertow** サブシステムは、JBoss EAP 6 の **web** サブシステムよりも大幅にパフォーマンスが改善されました。ご使用の環境で **undertow** サブシステムを調整するには、バッファーキャッシュ の設定、JSP 設定、および リスナー の設定などを行います。

9.1. バッファーキャッシュ

バッファーキャッシュは、**undertow** サブシステムによって処理される静的ファイルをキャッシュするために使用されます。これには、イメージ、静的 HTML、CSS、および JavaScript ファイルが含まれます。各 Undertow サーブレットコンテナーにデフォルトのバッファーキャッシュを指定できます。サーブレットコンテナーのバッファーキャッシュを最適化すると、静的ファイルに対する Undertow のパフォーマンスを向上できます。

バッファーキャッシュのバッファーは固定のサイズで、リージョンに割り当てられます。各バッファーキャッシュには設定可能な属性が 3 つあります。

buffer-size

各バッファーのバイト単位のサイズ。デフォルトは 1024 バイトです。Red Hat は、最も大きな静的ファイル全体を保存できるバッファーサイズに設定することを推奨します。

buffers-per-region

リージョンごとのバッファー数。デフォルトは 1024 です。

max-regions

バッファーキャッシュに割り当てられるメモリーの最大容量を設定する、リージョンの最大数。デフォルトは 10 リージョンです。

バッファーキャッシュによって使用されるメモリーの最大容量を算出するには、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ます。たとえばすべてがデフォルト値である場合、1024 (バイト単位のバッファーキャッシュ) * 1024 (リージョンごとのバッファー数) * 10 (リージョン数) = 10 MB になります。

バッファーキャッシュは、静的ファイルのサイズと、開発環境での想定負荷のテスト結果を基にして設定します。パフォーマンスの影響を判断するとき、バッファーキャッシュのパフォーマンスと、使用されるメモリーのバランスを考慮してください。

管理 CLI を使用したバッファーキャッシュの設定手順は、JBoss EAP **『設定ガイド』の「バッファーキャッシュの 設定** 」を参照してください。

9.2. バイトバッファープールの設定

Undertow バイトバッファープールは、プールされた NIO **ByteBuffer** インスタンスの割り当てに使用されます。すべてのリスナーにバイトバッファープールがあり、各リスナーに異なるバッファープールおよびワーカーを使用できます。バイトバッファープールは異なるサーバーインスタンス間で共有できます。

バイトバッファープールの設定に使用できる属性の完全リストは、JBoss EAP『設定 **ガイド』の「バイトバッファープールの属性** 」 を参照してください。

パフォーマンスに大きく影響する主なバイトバッファープール属性は **buffer-size** です。デフォルトはシステムの RAM リソースを基に算出され、デフォルトの値はほとんどの場合で適切です。この属性を手作業で設定する場合、ほとんどのサーバーに適切なサイズは 16 KB です。

バイトバッファープールの作成および **設定方法の手順は、JBoss EAP『設定ガイド』** を参照してください。

9.3. JSP 設定

JSP ページが Java バイトコードにコンパイルされる方法を最適化する、Undertow サーブレットコンテナーの JSP 設定オプションがあります。

generate-strings-as-char-arrays

JSP に多くの **String** 定数が含まれる場合、このオプションを有効にすると **String** 定数を **char** アレイに変換してスクリプトレットを最適化します。

optimize-scriptlets

JSP に多くの **String** 連結が含まれる場合、このオプションを有効にすると各 JSP リクエストの **String** 連結を削除してスクリプトレットを最適化します。

trim-spaces

JSP に多くの空白が含まれる場合、このオプションを有効にすると HTTP リクエストの空白を減らして、HTTP リクエストのペイロードを削減します。

JSP オプションの設定

これらの Undertow JSP 設定オプションは、管理コンソールまたは管理 CLI を使用して有効にできます。

- 管理コンソールを使用して有効にする場合:
  1. **Configuration → Subsystems → Web (Undertow) → Servlet Container** と選択します。
  2. 設定するサーブレットコンテナーを選択し、**表示** をクリックします。
  3. **JSP** を選択し、**Edit** をクリックします。
  4. 有効にする各オプションに対して、フィールドを **ON** に設定し、**Save** をクリックします。
- 管理 CLI を使用して有効にする場合は、以下のコマンドを使用します。

```
/subsystem=undertow/servlet-container=SERVLET_CONTAINER/setting=jsp/:write-attribute(name=OPTION_NAME,value=true)
```

たとえば、**default** サーブレットコンテナーの **generate-strings-as-char-arrays** を有効にするには、以下のコマンドを使用します。

```
/subsystem=undertow/servlet-container=default/setting=jsp/:write-attribute(name=generate-strings-as-char-arrays,value=true)
```

9.4. リスナー

アプリケーションと環境に応じて、特定ポートのトラフィックなどの一部のタイプのトラフィックに固有する複数のリスナーを設定し、各リスナーにオプションを設定することができます。

以下は、HTTP、HTTPS、および AJP リスナー上に設定できるパフォーマンス関連のオプションになります。

max-connections

リスナーが処理可能な最大同時接続数。デフォルトではこの属性は未定義で、接続数の制限はありません。
このオプションを使用すると、リスナーが処理できる接続数の上限を設定できます。これは、リソースの使用度を制限するのに役立つことがあります。この値を設定するには、ワークロードとトラフィックタイプを考慮する必要があります。これは、リソースの使用度を制限するのに役立つことがあります。以下の **no-request-timeout** も参照してください。

no-request-timeout

接続が閉じられるまでに接続がアイドル状態でいられる期間 (ミリ秒単位)。デフォルトの値は 60000 ミリ秒 (1 分) です。
接続の効率を最適化するためにご使用の環境でこのオプションを調整すると、ネットワークパフォーマンスの向上に役立ちます。アイドル接続が途中で閉じられた場合、接続を再確立するオーバーヘッドが発生します。アイドル接続の開かれている期間が長すぎると、リソースが不必要に使用されます。

max-header-size

バイト単位の HTTP リクエストヘッダーの最大サイズ。デフォルトは 1048576 (1024KB) です。
ヘッダーサイズを制限すると、一部の DOS 攻撃の防止に役立ちます。

buffer-pool

リスナーに使用する **io** サブシステムのバッファープールを指定します。デフォルトでは、すべてのリスナーが **default** バッファーリスナーを使用します。
このオプションを使用して各リスナーが一意のバッファープールを使用するよう設定したり、複数のリスナーが同じバッファープールを使用するよう設定できます。

worker

undertow サブシステムは **io** サブシステムに依存して XNIO ワーカーを提供します。このオプションはリスナーが使用する XNIO ワーカーを指定します。デフォルトでは、リスナーは **default** ワーカーを **io** サブシステムで使用します。
異なるワーカーリソースを特定のネットワークトラフィックに割り当てできるようにするため、特定のワーカーを使用するよう各リスナーを設定すると便利であることがあります。

リスナーオプションの設定

管理コンソールまたは管理 CLI を使用してリスナーオプションを設定できます。

- 管理コンソールを使用して設定する場合:
  1. **Configuration → Subsystems → Web (Undertow) → Server** と選択します。
  2. 設定するサーバーを選択し、**表示** をクリックします。
  3. 左側のメニューで、**Listener** を選択し (**HTTP Listener** など)、表でリスナーを選択します。
  4. **Edit** をクリックして設定するオプションを変更し、**Save** をクリックします。
- 管理 CLI を使用して設定する場合は、以下のコマンドを使用します。

```
/subsystem=undertow/server=SERVER_NAME/LISTENER_TYPE=LISTENER_NAME:write-attribute(name=OPTION_NAME,value=OPTION_VALUE)
```

default-server Undertow サーバーの **default** HTTP リスナーに対し、 **max-connections** を **100000** に設定するには、以下のコマンドを使用します。

```
/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=max-connections,value=100000)
```

第10章 IO サブシステムの調整

io サブシステムは、Undertow や Remoting などの別の JBoss EAP サブシステムによって使用される XNIO ワーカーやバッファープールを定義します。

10.1. ワーカーの設定

それぞれが独自のパフォーマンス設定を持ち、異なる I/O タスクを処理するワーカーを複数作成することができます。たとえば、HTTP I/O を処理する 1 つのワーカーを作成して、EJB I/O を処理する別のワーカーを作成し、特定の負荷要件に対応する各ワーカーの属性を個別に設定できます。

設定可能なワーカー属性のリストは、IO サブシステムの属性 を参照してください。

パフォーマンスに大きく影響するワーカー属性には、ワーカーが使用できる I/O スレッドの合計数を設定する **io-threads** や、特定のタスクに使用できるスレッドの最大数を設定する **task-max-threads** などがあります。これら 2 つの属性のデフォルト値は、サーバー の CPU 値を基に算出されます。

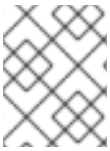
ワーカーの作成および **設定方法の手順は、JBoss EAP『設定ガイド』** を参照してください。

10.1.1. ワーカー統計の監視

管理 CLI を使用してワーカーのランタイム統計を確認できます。これは、接続数、スレッド数、キューのサイズなどのワーカー統計を表示します。

以下のコマンドは、**default** ワーカーのランタイム統計を表示します。

```
/subsystem=io/worker=default:read-resource(include-runtime=true,recursive=true)
```



注記

core-pool-size の統計によって追跡されるコアスレッドの数は、現在常に **max-pool-size** の統計によって追跡されるスレッドの最大数と同じ値に設定されます。

10.2. バッファープールの設定

注記

IO バッファープールは非推奨となっていますが、現行リリースではデフォルトとして設定されています。Undertow バイトバッファープールの設定に関する詳細は、JBoss EAP **『設定ガイド』**の「 Undertow バイトバッファープール 」を参照してください。

io サブシステムのバッファープールは、I/O 操作 のために使用されるプールされた NIO バッファーインスタンスです。ワーカー と同様に、特定の I/O タスクの処理に専念する個別のバッファープールを作成できます。

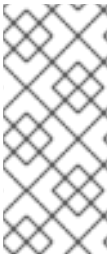
設定可能なバッファープール属性のリストは、IO サブシステムの属性 を参照してください。

パフォーマンスに大きく影響する主なバッファープール属性は **buffer-size** です。デフォルトはシステムの RAM リソースを基に算出され、デフォルトの値はほとんどの場合で適切です。この属性を手作業で設定する場合、ほとんどのサーバーに適切なサイズは 16 KB です。

バッファープールの作成および **設定方法の手順は、JBoss EAP『設定ガイド』** を参照してください。

第11章 JGROUPS サブシステムの調整

ネットワークのパフォーマンスを最適化するため、UDP マルチキャストをサポートする環境では JGroups に UDP マルチキャストを使用することが推奨されます。



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。

本章では、JGroups スタックトランスポートプロトコル (UDP または TCP) と JGroups クラスターの通信で使用される通信プロトコルが選択済みであることを前提としています。JGroups でのクラスター通信 に関する詳細は、JBoss EAP **『設定ガイド』** を参照してください。

11.1. JGROUPS 統計の監視

jgroups サブシステムの統計を有効にして、JBoss EAP のクラスタリングを監視するには、管理 CLI を使用するか、 JMX を経由します。

注記

統計を有効にすると、パフォーマンスに悪影響が及びます。必要時のみ統計を有効にします。

1. 以下のコマンドを使用して JGroups チャネルの統計を有効にします。

注記

管理対象ドメインの場合はコマンドの前に **/profile=PROFILE_NAME** を追加してください。

```
/subsystem=jgroups/channel=CHANNEL_NAME:write-attribute(name=statistics-enabled,value=true)
```

たとえば、以下のコマンドを使用して、デフォルトの **ee** チャネルの統計を有効にします。

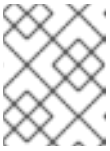
```
/subsystem=jgroups/channel=ee:write-attribute(name=statistics-enabled,value=true)
```

2. JBoss EAP サーバーをリロードします。

```
reload
```

3. JVM 監視ツールで管理 CLI を使用または JMX を経由すると、JGroups の統計を表示できるようになります。

- 管理 CLI を使用する場合は、統計を表示する JGroups チャネルまたはプロトコルで **:read-resource(include-runtime=true)** コマンドを使用します。



注記

管理対象ドメインでは、コマンドの前に **/host=HOST_NAME/server=SERVER_NAME** を追加します。

例を以下に示します。

- **ee** チャネルの統計を表示するには、以下のコマンドを使用します。

```
/subsystem=jgroups/channel=ee:read-resource(include-runtime=true)
```

- **ee** チャネルの **FD_ALL** プロトコルの統計を表示するには、以下のコマンドを使用します。

```
/subsystem=jgroups/channel=ee/protocol=FD_ALL:read-resource(include-runtime=true)
```

- JVM 監視ツールを使用して JBoss EAP に接続する場合は、「パフォーマンスの監視」を参照してください。JMX 接続を介して JGroups MBean の統計を表示できます。

11.2. ネットワーキングおよびジャンボフレーム

可能な限り、JGroups トラフィックのネットワークインターフェースが専用の仮想ローカルエリアネットワーク (VLAN) の一部であるようにしてください。これにより、クラスターの通信を他の JBoss EAP ネットワークトラフィックから分離でき、ネットワークのパフォーマンス、スループット、およびセキュリティーの制御をより簡単にすることができます。

クラスターパフォーマンスを向上するために考慮する他のネットワーク設定は、ジャンボフレームの有効化です。ネットワーク環境がサポートする場合は、MTU (Maximum Transmission Unit) を増加してジャンボフレームを有効にすると、特にスループットが高い環境ではネットワークパフォーマンスの向上に貢献できます。

ジャンボフレームを使用するには、ネットワークのすべての NIC およびスイッチがジャンボフレームをサポートする必要があります。Red Hat カスタマーポータルの Red Hat Enterprise Linux でジャンボフレームを有効にする手順を参照してください。

11.3. メッセージのバンドル

JGroups のメッセージバンドルは、複数の小さなメッセージを大きなバンドルにアセンブルし、ネットワークのパフォーマンスを向上します。ネットワーク上で多くの小さなメッセージをクラスターノードに送信する代わりに、メッセージは最大バンドルサイズに達するか、送信する他のメッセージがなくなるまでキューに置かれます。キューに置かれたメッセージは大きなメッセージバンドルにアセンブルされた後、送信されます。

このバンドル化により、特にネットワーク通信のオーバーヘッドが高い TCP 環境では通信のオーバーヘッドが削減されます。

メッセージバンドルの設定

JGroups のメッセージバンドルは **max_bundle_size** プロパティーを使用して設定されます。デフォルトの **max_bundle_size** は 64 KB です。

バンドルサイズの調整によるパフォーマンスの向上は環境によって異なり、バンドルのアセンブル中に効率的なネットワークトラフィックと通信遅延の可能性の釣り合いがとれたかどうかによっても異なります。

以下の管理 CLI コマンドを使用して **max_bundle_size** を設定します。

```
/subsystem=jgroups/stack=STACK_NAME/transport=TRANSPORT_TYPE/property=max_bundle_size:add(value=BUNDLE_SIZE)
```

たとえば、デフォルトの **udp** スタックに対して **max_bundle_size** を **60K** に設定するには、以下を実行します。

```
/subsystem=jgroups/stack=udp/transport=UDP/property=max_bundle_size:add(value=60K)
```

11.4. JGROUPS スレッドプール

jgroups サブシステムは独自のスレッドプールをクラスター通信の処理に使用します。JGroups には個別に設定できる **default**、**internal**、**oob**、および **timer** 関数のスレッドプールが含まれます。各 JGroups スレッドプールには、**keepalive-time**、**max-threads**、**min-threads**、および **queue-length** の設定可能な属性が含まれます。

各スレッドプール属性の適切な値は、環境によって異なりますが、ほとんどの場合でデフォルト値で対応できます。

JGroups スレッドプールの **設定方法の手順は、JBoss EAP『設定ガイド』** を 参照してください。

11.5. JGROUPS の送受信バッファー

jgroups サブシステムには、 UDP および TCP スタック向けの設定可能な送受信バッファーがあります。

JGroups バッファーの適切な値は環境によって異なりますが、ほとんどの場合でデフォルト値で対応できます。開発環境にて負荷がかかった状態でクラスターをテストし、バッファーサイズの適切な値を調整することが推奨されます。

注記

オペレーティングシステムが利用可能なバッファーサイズを制限し、JBoss EAP が設定済みのバッファーサイズを使用できないことがあります。JBoss EAP **『設定ガイド』** の「 バッファーサイズ警告の解決 」を参照してください。

JGroups の送受信バッファーの **設定方法は、JBoss EAP『設定ガイド』** を参照し てください。

第12章 TRANSACTIONS サブシステムの調整

お使いの環境が XA 分散トランザクションを使用する場合、トランザクションマネージャーのログストアを調整してパフォーマンスを向上できます。

デフォルトのトランザクションログストアは、簡単なファイルストアを使用します。トランザクションログごとに 1 つのシステムファイルが作成されるため、XA トランザクションではこのようなログストアは効率的ではありません。特に XA トランザクションでは、すべてのトランザクションに対して 1 つのファイルで構成されるジャーナルを使用する、ジャーナルストアの方が大変効率的です。

XA トランザクションのパフォーマンスを向上するため、ジャーナルログストアの使用が推奨されます。Red Hat Enterprise Linux システムでは、ジャーナルストアに追加で非同期 I/O (AIO) を追加し、さらにパフォーマンスを向上できます。

注記

Red Hat Enterprise Linux システムで、ジャーナルストアの非同期 I/O (AIO) を有効にする場合は、必ず **libaio** パッケージがインストールされているようにしてください。

管理コンソールを使用したジャーナルログストアの有効化

1. **Configuration → Subsystems → Transaction** と選択し、**表示** をクリックします。
2. **Configuration** タブで **Edit** をクリックします。
3. **Use Journal Store** フィールドを **ON** に設定します。
4. **任意設定**: Red Hat Enterprise Linux システムの場合は **Journal Store Enable Async IO** フィールドを **ON** に設定します。
5. **Save** をクリックします。

管理 CLI を使用したジャーナルログストアの有効化

1. 管理 CLI を使用してジャーナルログストアを有効にする場合は、以下のコマンドを使用します。

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

2. **任意設定**: Red Hat Enterprise Linux システムの場合、以下のコマンドを使用して、ジャーナルログストアの非同期 I/O を有効にします。

```
/subsystem=transactions:write-attribute(name=journal-store-enable-async-io, value=true)
```

付録A リファレンス資料

A.1. データソースの統計

表A.1 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。
AveragePoolTime	接続がプールで費やす時間の平均。
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。
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。

名前	説明
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。
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。
XAPrepareCount	XAResource prepare 呼び出しの数。

名前	説明
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.2 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.2. リソースアダプターの統計

表A.3 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。
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.3. IO サブシステムの属性

注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-io_2_0.xsd** のスキーマ定義ファイルで確認してください。

表A.4 worker の属性

属性	デフォルト	説明
io-threads		ワーカーに作成する I/O スレッドの数。指定のない場合は、スレッドの数が CPU の数の 2 倍に設定されます。
stack-size	0	ワーカースレッドへの使用を試みるスタックサイズ (バイト単位)。
task-keepalive	60000	コアでないタスクスレッドを生存状態にするミリ秒数。
task-core-threads	2	コアタスクスレッドプールのスレッド数。
task-max-threads		ワーカータスクスレッドプールの最大スレッド数。指定のない場合は、**MaxFileDescriptorCount** JMX プロパティーを考慮して (設定されている場合)、最大スレッド数が CPU の数の 16 倍に設定されます。

表A.5 buffer-pool の属性

属性	デフォルト	説明
 **注記** IO バッファープールは非推奨となっていますが、現行リリースではデフォルトとして設定されています。Undertow バイトバッファープールの設定に関する詳細は、JBoss EAP **『設定ガイド』**の「Undertow バイトバッファープール」を参照してください。また、バイトバッファープールの属性 リストは、JBoss EAP **『設定ガイド』** の「バイトバッファープールの属性」を参照してください。		
buffer-size		各バッファースライスのサイズ (バイト単位)。指定のない場合は、以下のようにシステムで利用できる RAM を基にサイズが設定されます。 ● RAM が 64 MB 未満の場合は 512 バイト ● RAM が 64 - 128 MB の場合は 1024 バイト (1 KB) ● RAM が 128 MB を超える場合は 16384 バイト (16 KB) この属性に対するパフォーマンスチューニングのアドバイスは、「バッファープールの設定」を参照してください。

属性	デフォルト	説明
buffers-per-slice		大型のバッファーをいくつのスライス (セクション) に分割するか。これは、多数の個別のバッファーに割り当てするよりもメモリーの効率がよくなります。指定のない場合、システムで利用可能な RAM を基にしてスライスの数が設定されます。 ● RAM が 128 MB 未満の場合は 10 ● RAM が 128 MB を超える場合は 20
direct-buffers		バッファープールが直接バッファーを使用するかどうか。直接バッファーをサポートしないプラットフォームがあることに注意してください。

Revised on 2021-05-17 08:58:00 CEST