# Red Hat JBoss Enterprise Application Platform 8.0

## Red Hat JBoss Enterprise Application Platform のパフォーマンスチューニング

Red Hat JBoss Enterprise Application Platform のパフォーマンスを評価し、更新をパフォーマンスを向上するように設定する手順

Last Updated: 2024-02-10

# Red Hat JBoss Enterprise Application Platform 8.0 Red Hat JBoss Enterprise Application Platform のパフォーマンスチューニング

Red Hat JBoss Enterprise Application Platform のパフォーマンスを評価し、更新をパフォーマンスを向上するように設定する手順

## 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

**JBOSS EAP ドキュメントへのフィードバック (英語のみ)** ..... 4

**多様性を受け入れるオープンソースの強化** ..... 5

**第1章 パフォーマンスチューニングの概要** ..... 6
1.1. 本書における EAP_HOME の使用 6

**第2章 パフォーマンスの監視** ..... 7
2.1. リモート監視接続のための JBOSS EAP の設定 7
2.2. JCONSOLE 9
2.3. JAVA MISSION CONTROL 11

**第3章 パフォーマンスの問題の診断** ..... 12
3.1. ガベージコレクションのロギングの有効化 12
3.2. JAVA ヒープダンプ 12
3.3. JAVA FLIGHT RECORDER 14
3.4. JAVA スレッドによる CPU 高使用率の特定 23
3.5. マネージドエグゼキュータサービスおよびマネージドスケジュールエグゼキュータサービスのランタイム統計 24

**第4章 JVM の調整** ..... 26
4.1. 固定ヒープサイズの設定 26
4.2. ガベージコレクターの設定 26
4.3. ラージページの有効化 26
4.4. ULIMITS の設定 27
4.5. ホストコントローラーとプロセスコントローラーの JVM の調整 28

**第5章 JAKARTA ENTERPRISE BEANS サブシステムの調整** ..... 30
5.1. BEAN インスタンスプール 30
5.2. BEAN スレッドプール 31
5.3. BEAN のランタイムデプロイメント情報 32
5.4. エンタープライズ BEAN サブシステムの調整の必要性を示す例外 35

**第6章 データソースおよびリソースアダプターの調整** ..... 38
6.1. プール統計の監視 38
6.2. プールの属性 41
6.3. プール属性の設定 42

**第7章 MESSAGING サブシステムの調整** ..... 44
7.1. メッセージング統計の有効化 44
7.2. メッセージング統計の表示 45
7.3. メッセージカウンターの設定 46
7.4. キューのメッセージカウンターと履歴の表示 46
7.5. キューのメッセージカウンターのリセット 47
7.6. 管理コンソールを使用したランタイム操作 47
7.7. JAKARTA MESSAGING のチューニング 50
7.8. 永続性の調整 51
7.9. その他の調整オプション 52
7.10. アンチパターンの回避 53

**第8章 LOGGING サブシステムの調整** ..... 54
8.1. コンソールへのロギングの無効化 54
8.2. ロギングレベルの設定 54
8.3. ログファイルの場所の設定 54

**第9章 UNDERTOW サブシステムの調整** ..... **55**
9.1. バッファーキャッシュ設定 55
9.2. バイトバッファープールの設定 55
9.3. JAKARTA SERVER PAGES 設定オプション 56
9.4. リスナー設定オプション 57

**第10章 IO サブシステムの調整** ..... **59**
10.1. ワーカーの設定 59
10.2. バッファープールの設定 59

**第11章 JGROUPS サブシステムの調整** ..... **60**
11.1. JGROUPS 統計の監視 60
11.2. ネットワーキングおよびジャンボフレーム 61
11.3. メッセージバンドル 61
11.4. JGROUPS スレッドプール 62
11.5. JGROUPS の送受信バッファー 62

**第12章 TRANSACTIONS サブシステムの調整** ..... **63**
12.1. 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 63
12.2. 管理 CLI を使用したジャーナルログストアの有効化 63

**付録A 参考資料** ..... **64**
A.1. データソースの統計 64
A.2. リソースアダプターの統計 67
A.3. IO サブシステムの属性 67

# JBOSS EAP ドキュメントへのフィードバック (英語のみ)

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

**手順**

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

# 多様性を受け入れるオープンソースの強化

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

# 第1章 パフォーマンスチューニングの概要

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。

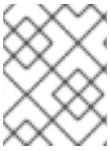
**重要**

パフォーマンス設定の変更は、必ずステージング環境またはテスト環境にて予想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。

- JBoss EAP 圧縮ファイル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圧縮アーカイブを展開した **jboss-eap-8.0** ディレクトリーです。
- 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 **${user.home}/EAP-8.0.0** です。
  - Red Hat Enterprise Linux および Solaris の場合: **/home/USER_NAME/EAP-8.0.0/**
  - Microsoft Windows の場合: **C:\Users\USER_NAME\EAP-8.0.0**
- JBoss Tools インストーラーを使用して JBoss EAP サーバーをインストールおよび設定した場合、**EAP_HOME** のデフォルトのパスは **${user.home}/devstudio/runtimes/jboss-eap** です。
  - 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**

JBoss Tools で **Target runtime** を 8.0 以降のランタイムバージョンに設定した場合、プロジェクトは Jakarta EE 8 仕様と互換性があります。

**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、事前設定されたラッパースクリプトが JBoss EAP に含まれている JConsole か、Java Mission Control (JMC) を使用することを推奨します。どちらのツールも、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 メトリクスを含む、JVM プロセスの基本的な監視機能を提供します。

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、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

### 2.1.1. スタンドアロンサーバーのリモート監視接続の設定

**手順**

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。

2. JBoss EAP を開始するとき、管理インターフェイ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 管理インターフェイ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イ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
```

### 2.1.2. マネージドドメインホストの JBoss EAP リモート監視接続の設定

管理インターフェイスをバインドする上記の手順をマネージド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、マネージド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。

**手順**

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。
2. Elytron を使用するように **remoting** サブシステムを設定するには、次のコマンドを実行します。

```
/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,sasl-authentication-factory=application-sasl-authentication)
```

3. JBoss EAP マネージドドメインホストを起動するとき、以下のインターフェイスの 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。
   - マネージド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イ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、管理インターフェイ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 管理インターフェイ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イ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。
   - マネージド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+http://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+http://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。

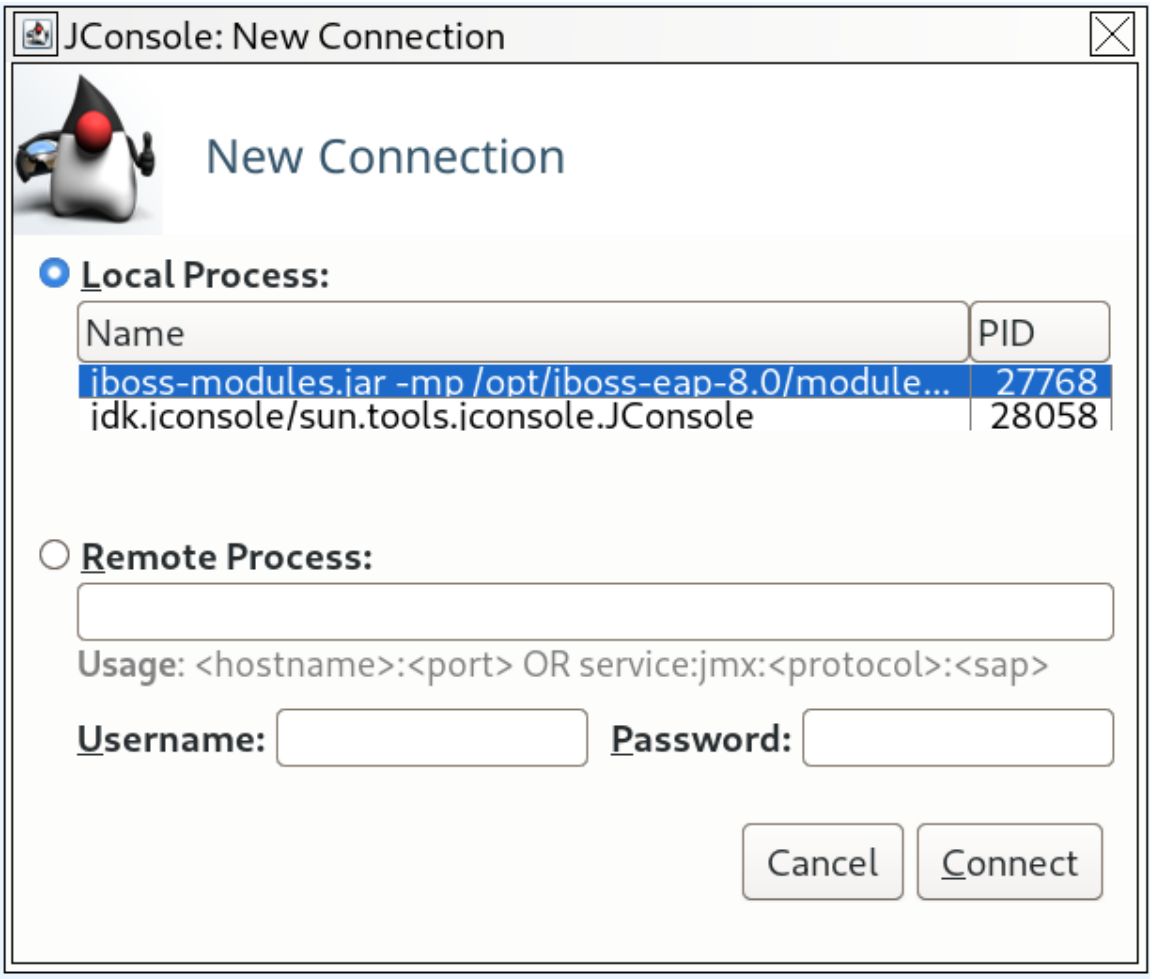
### 2.2.1. JConsole を使用したローカル JBoss EAP JVM への接続

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。

**手順**

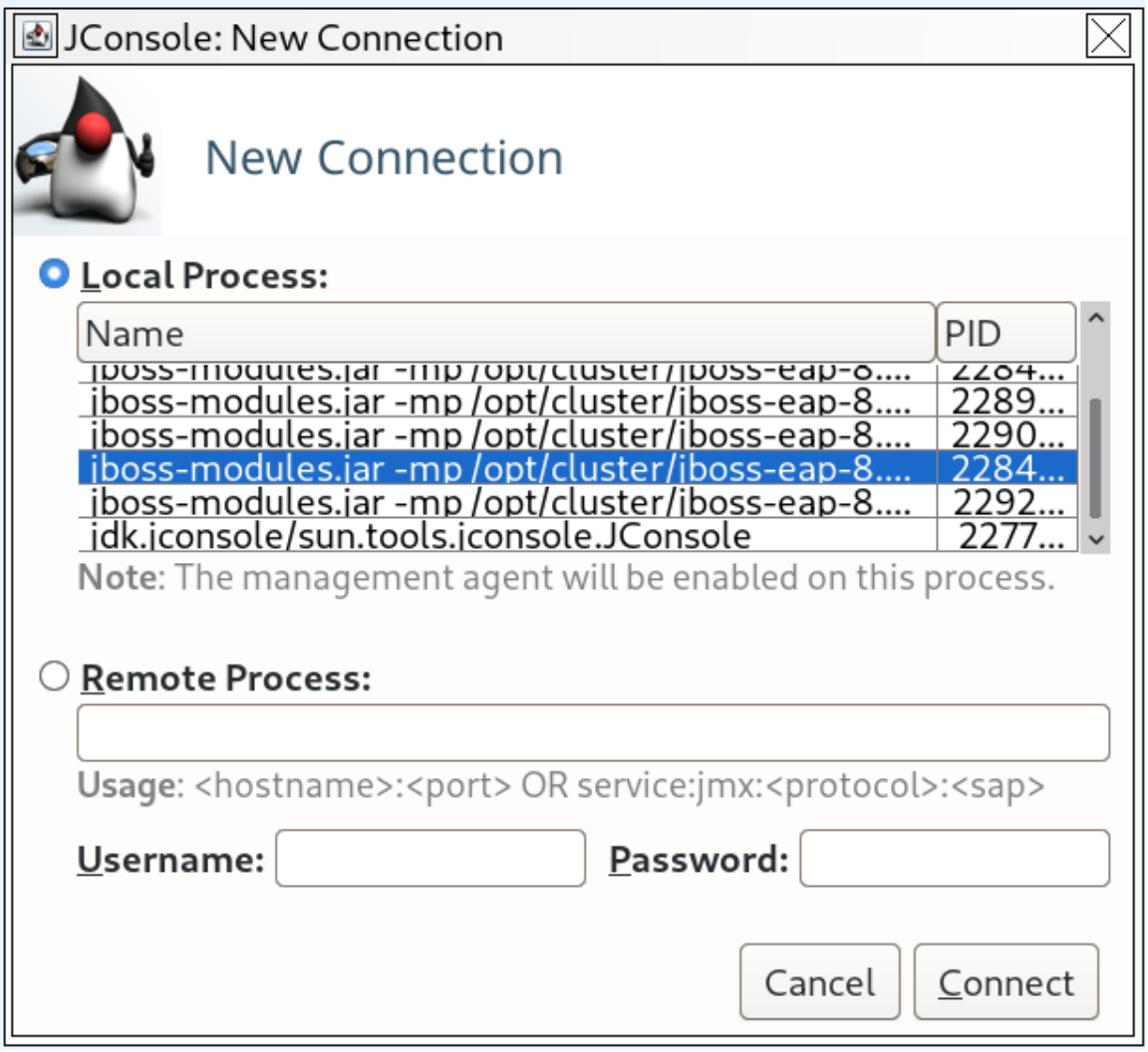
1. **EAP_HOME/bin** で **jconsole** スクリプトを実行します。
2. **Local Process** で、監視する JBoss EAP JVM プロセスを選択します。
   - スタンドアロン JBoss EAP サーバーの場合は、JBoss EAP の JVM プロセスは 1 つになります。

     **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 のマネージド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 のマネージドドメイン 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

JConsole を使用すると、**jconsole** スクリプトを使用してリモート JBoss EAP JVM に接続できます。

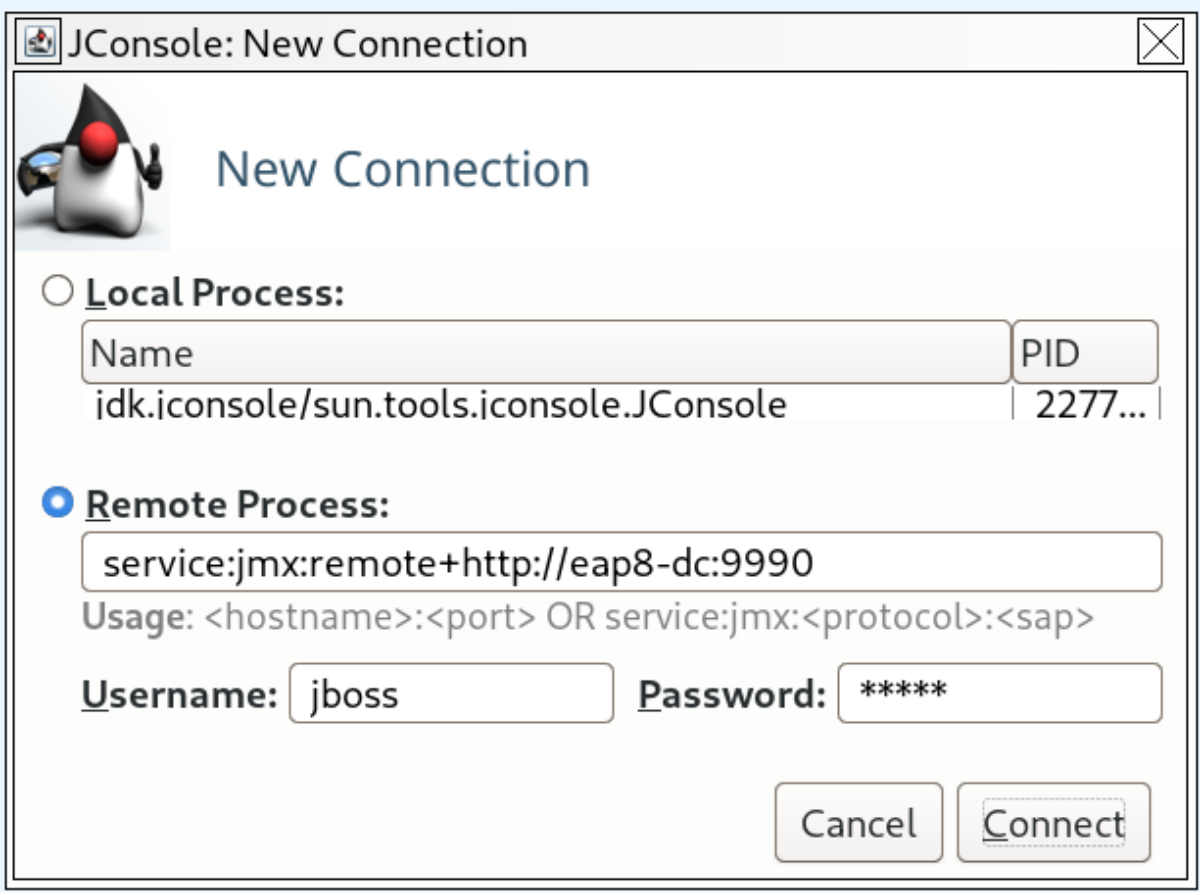
**前提条件**

- リモート監視接続用に JBoss EAP を設定している。
- 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て展開している。

**手順**

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。

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 MISSION CONTROL

開発者は Java Mission Control (JMC) を使用してパフォーマンスの問題を特定できます。JMC は次のコンポーネントで構成されるプロファイリングおよび診断ツールです。

- 実行中の Java アプリケーションとそれに関連する JVM を表示する **Java 仮想マシン (JVM) ブラウザー**
- JVM を監視するための **Java Management (JMX) コンソール**
- 実行中の Java アプリケーションから診断データを収集する **Java Flight Recorder (JFR)**
- ヒープダンプ分析用の **プラグイン**

**関連情報**

- JMC をローカルまたはリモートの JBoss EAP JVM に接続する方法については、Red Hat カスタマーポータルの How to connect Java Mission Control with EAP remotely? を参照してください。
- JMC のダウンロードとインストールの詳細は、Oracle の「JDK Mission Control User Guide」の Install JDK Mission Control and Supported Plug-insセクションを参照してください。

# 第3章 パフォーマンスの問題の診断

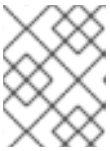
## 3.1. ガベージコレクションのロギングの有効化

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、ガベー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。

ガベー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。

ガベー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 マネージドドメインの場合、ガベー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 でガベー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 11 または Oracle JDK 11 の場合:

```
-verbose:gc -Xloggc:<path_to_directory>/gc.log -XX:+PrintGCDetails -
XX:+PrintGCDateStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -
XX:GCLogFileSize=3M -XX:-TraceClassUnloading
```

- OpenJDK、Oracle JDK、または JEP 271 をサポートする JDK のバージョン 9 以降の場合:

```
-Xlog:gc*:file=<path_to_directory>/gc.log:time,uptimemillis:filecount=5,filesize=3M
```

**関連情報**

- JEP 271 の詳細は、Open JDK Web ページの JEP 271: Unified GC Logging 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。このセクションでは、Oracle JDK と OpenJDK に共通する方法を説明します。

### 3.2.1. OpenJDK と Oracle JDK を使用したヒープダンプの作成

#### 3.2.1.1. 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。

**手順**

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。

#### 3.2.1.2. OutOfMemoryError 時のヒープダンプの自動作成

**-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。

### 3.2.2. ヒープダンプの分析

#### 3.2.2.1. 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。

**関連情報**

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。

#### 3.2.2.2. ヒープダンプ分析のヒント

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。

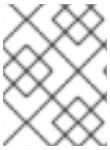
メモリー問題の原因を特定したら、ガベージコレクションのルートからパスを確認し、オブジェクトが何によって維持されているかを確認します。

## 3.3. JAVA FLIGHT RECORDER

### 3.3.1. Java Flight Recorder について

Oracle の **JDK Mission Control User Guide**では、Java Flight Recorder (JFR) を "プロファイリングおよびイベント収集フレームワーク" と呼んでいます。開発者は JFR と JDK Mission Control (JMC) を使用して、Java 仮想マシン (JVM) およびその他の Java アプリケーションに関するデータを収集できます。開発者はこのデータを使用してパフォーマンスの問題を特定し、修正できます。

JFR は、必要なオーバーヘッド (リソースの消費量) を低く抑えるように、慎重に設計されています。そのため、JFR のプロファイリングは、影響を最小限に抑えながら、特定の実稼働環境で継続的に実行できます。開発者は JFR と JMC を使用して、インシデント発生後にランタイム情報を迅速に分析できます。



**注記**

JFR は、Java OpenJDK **8u262** 以降から Java Diagnostic Command Tool の一部として利用できます。

**関連情報**

- JFR の詳細は、Oracle の **JDK Mission Control User Guide**の Flight Recorder のセクションを参照してください。

### 3.3.2. Java Flight Recorder のプロファイリング設定

開発者はプロファイリング設定を変更して、Java Flight Recorder (JFR) のインスタンスをカスタマイズできます。JFR では 2 つの異なるプロファイリング設定を使用できます。

- **default**: まばらな情報のサンプリングを提供します (低レベルのプロファイリングの詳細)。
- **profile**: より包括的な情報のサンプリングを提供します (中レベルのプロファイリングの詳細)。

開発者はどちらかの設定ファイルを変更して、追加のイベントメトリクスのサンプリングを有効にすることができます。

### 3.3.3. Java Flight Recorder のプロファイルキャプチャーの有効化

開発者は Java Flight Recorder (JFR) を使用して、ベアメタル上または Red Hat OpenShift Container Platform 上の JBoss EAP インストールのプロファイリングを行うことができます。

OpenShift で JFR を使用する方法の詳細は、Introduction to Cryostat: JDK Flight Recorder for containers を参照してください。

#### 3.3.3.1. ベアメタル上での Java Flight Recorder プロファイリングの有効化

開発者は、コマンドラインを使用するか、Java Mission Control (JMC) デスクトップアプリケーションで Java Management Extensions (JMX) を使用して、Java Flight Recorder (JFR) プロファイリングを開始できます。

#### 3.3.3.2. Java 仮想マシンの設定フラグを使用した JBoss EAP 向けの Java Flight Recorder プロファイリングの設定

設定フラグを使用すると、Java 仮想マシン (JVM) で JBoss EAP の Java Flight Recorder (JFR) プロファイリングを設定できます。

**JVM 設定の例**

```
-XX:StartFlightRecording=delay=15s,duration=60s,name=jboss-eap-profile,
filename=C:\TEMP\jboss-eap-profile.jfr,settings=default
```

**StartFlightRecording=delay** 設定フラグを使用すると、JVM の起動後、プロファイリングセッションを開始するまでに JFR が待機する時間を設定できます。前述の例では、**StartFlightRecording=delay** が 15 秒に設定されています。これは、プロファイリングを 15 秒遅らせて開始することを意味します。

**duration** 設定フラグを使用すると、各プロファイリングセッションの時間の長さを設定できます。前述の例では、**duration** は 60 秒に設定されています。

**name** 設定フラグを使用すると、メモリー内のプロファイル名を設定できます。前述の例では、メモリー内のプロファイル名は **jboss-eap-profile** に設定されています。

**filename** 設定フラグを使用すると、ファイルの保存先となるファイル名とパスを設定できます。この例では、**filename** は **C:\TEMP\jboss-eap-profile.jfr** に設定されています。

**settings** 設定フラグを使用すると、プロファイリング設定を選択できます。この例では、**settings** は **default** に設定されています。プロファイリング設定のファイル拡張子は除外されていることに注意してください。

プロファイリングセッションが完了すると、**filename** オプションで定義されたファイルパスにファイルが作成されます。

#### 3.3.3.3. 実行中の JBoss EAP Java 仮想マシンの Java コマンドツールを使用したプロファイリング

Java Flight Recorder (JFR) の **JFR.start** コマンドを使用すると、実行中の JBoss EAP Java 仮想マシン (JVM) を設定し、Java コマンドツール **jcmd** を使用したプロファイリングを行うことができます。

**手順**

- 以下のコマンドの 1 つを使用します。
  - Linux オペレーティングシステムの場合:

    ```
    $ jcmd <PID> JFR.start duration=TIME filename=path/to/YOUR_PROFILE_NAME.jfr
    ```

    以下に例を示します。

    **Linux の JFR.start コマンドの例**

    ```
    $ jcmd <PID> JFR.start duration=60s filename=/tmp/jboss-eap-profile.jfr
    ```

JFR プロファイリングセッションが開始すると、次の確認メッセージが表示されます。

```
$ jcmd <PID> JFR.start duration=60s filename=/tmp/jboss-eap-profile.jfr
<PID>:
Started recording 1. The result will be written to:

/tmp/jboss-eap-profile.jfr
```

- Windows オペレーティングシステムの場合:

```
> jcmd.exe <PID> JFR.start duration=TIME
filename=path/to/YOUR_PROFILE_NAME.jfr
```

以下に例を示します。

**Windows の JFR.start コマンドの例**

```
> jcmd.exe <PID> JFR.start duration=60s filename=C:\TEMP\jboss-eap-profile.jfr
```

JFR プロファイリングセッションが開始すると、次の確認メッセージが表示されます。

```
> jcmd.exe <PID> JFR.start duration=60s filename=C:\TEMP\jboss-eap-profile.jfr
<PID>:
Started recording 1. The result will be written to:

C:\TEMP\jboss-eap-profile.jfr
```

**duration** オプションを使用すると、各プロファイリングセッションの時間の長さを設定できます。前述のコマンド例では、**duration** は 60 秒に設定されています。

**filename** オプションを使用すると、ファイルの保存先となるファイル名とパスを設定できます。前述のコマンド例では、Linux の例では **filename** は **/tmp/jboss-eap-profile.jfr** に設定され、Windows の例では **C:\TEMP\jboss-eap-profile.jfr** に設定されています。

#### 3.3.3.4. Java Mission Control を使用したローカル Java 仮想マシンへの接続

Java Mission Control (JMC) を使用して、JMC のインスタンスと同じサーバー上で実行されているローカル Java 仮想マシン (JVM) に接続できます。

**前提条件**

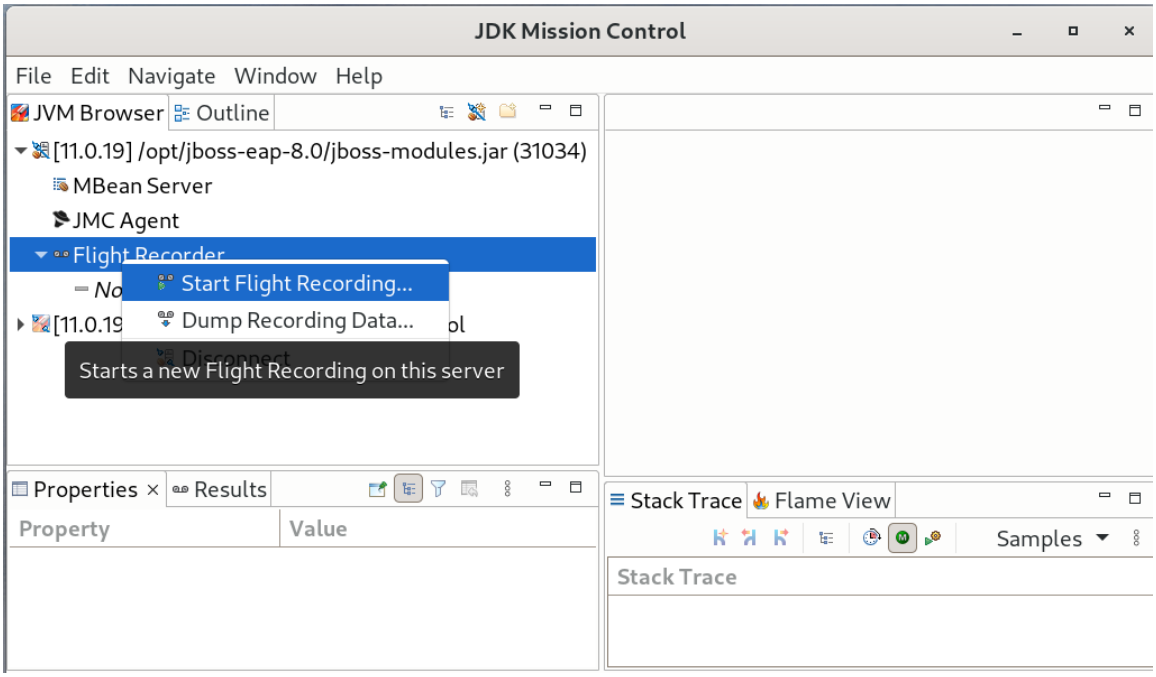
- JBoss EAP ライブラリーを使用して Java Mission Control を設定している。手順については、How to connect Java Mission Control with EAP remotely? を参照してください。
- JBoss EAP をリモート監視接続用に設定しており、監視のために **ApplicationRealm** にユーザーを作成している。

**手順**

1. Java Mission Control を開きます。
2. **JVM Browser** ペインで、プロファイリングする JVM を選択します。
3. JVM のドロップダウンメニューを展開して、**Flight Recorder** 項目を表示します。

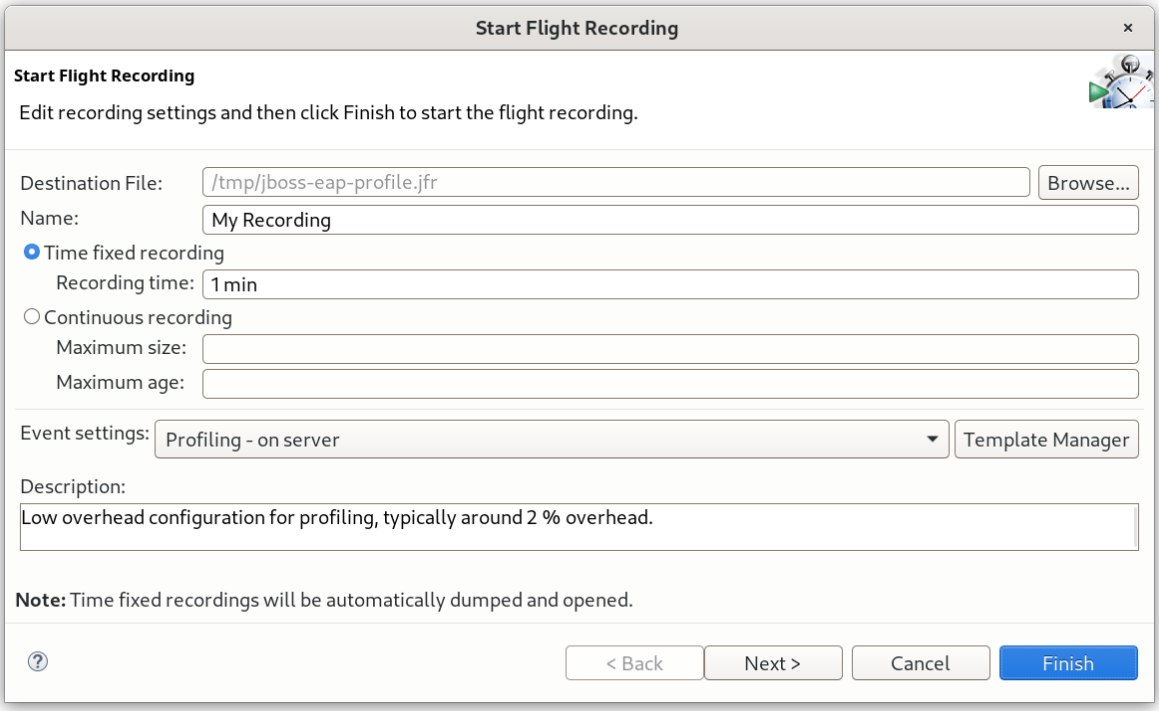
a. **Flight Recorder** を右クリックしてサブメニューを開き、**Start Flight Recording...** を選択します。

**図3.1 JMC の JVM Browser**



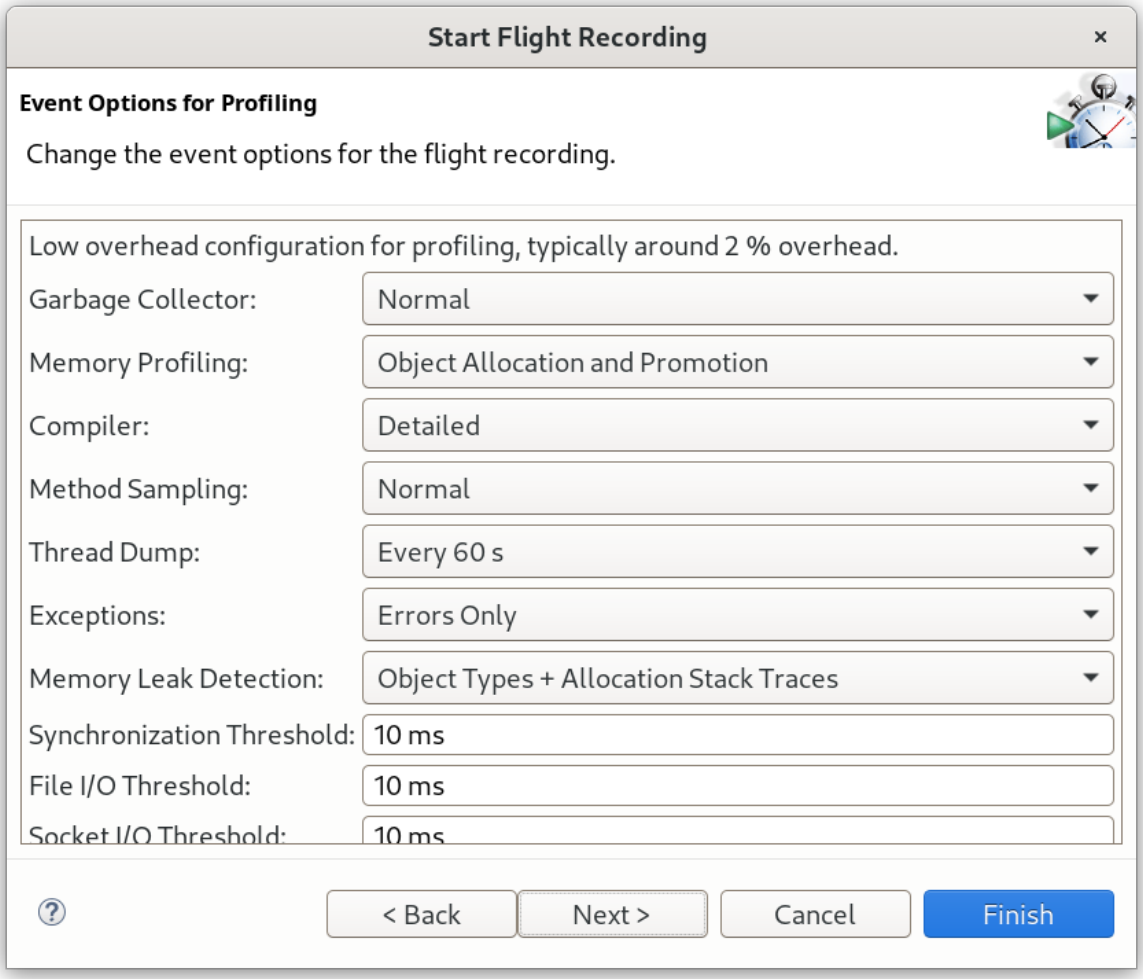

4. **Start Flight Recording** ウィンドウで、プロファイリングのオプションを設定します。

**図3.2 JVM プロファイリング設定**




5. **Next** をクリックし、低レベルの詳細な設定を行います。

図3.3 JVM プロファイリングの詳細設定




6. **Finish** をクリックしてプロファイリングを開始します。

### 3.3.3.5. Java Mission Control を使用したリモート Java 仮想マシンへの接続

Java Mission Control (JMC) を使用して、リモート Java 仮想マシン (JVM) のプロファイルに接続できます。

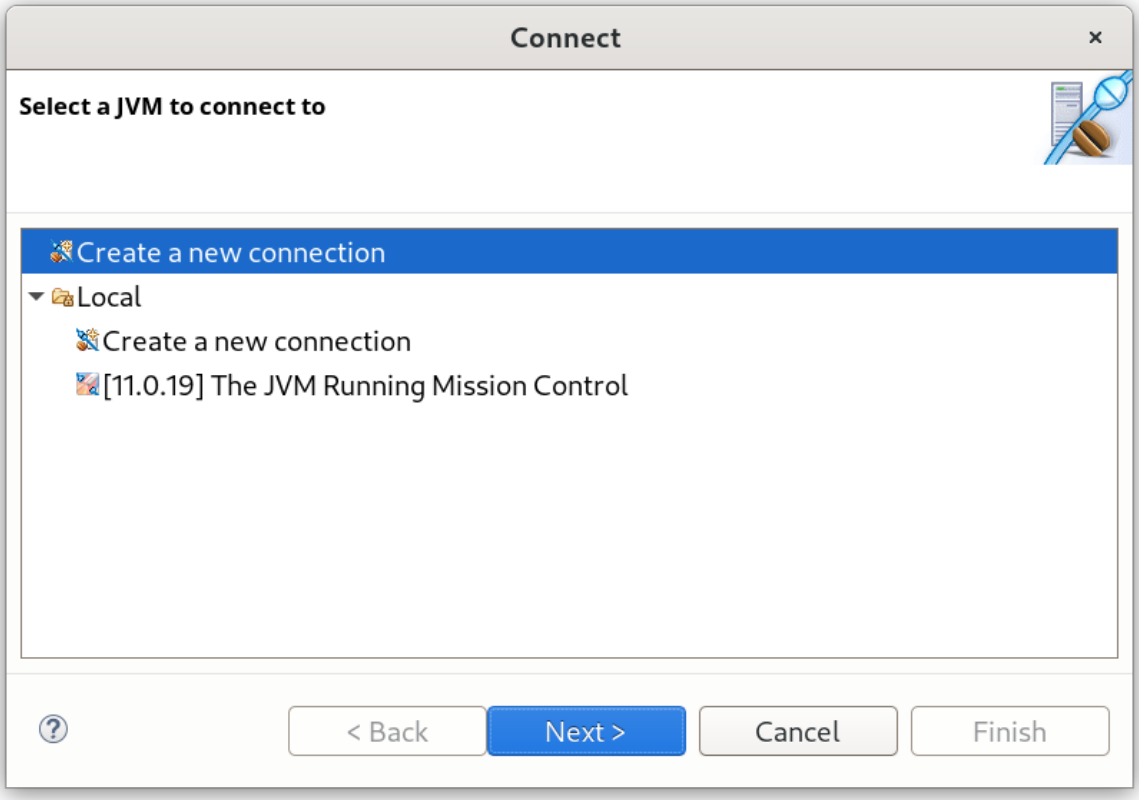
**前提条件**

- JBoss EAP ライブラリーを使用して Java Mission Control を設定している。手順については、How to connect Java Mission Control with EAP remotely? を参照してください。
- JBoss EAP をリモート監視接続用に設定しており、監視のために **ApplicationRealm** にユーザーを作成している。

**手順**

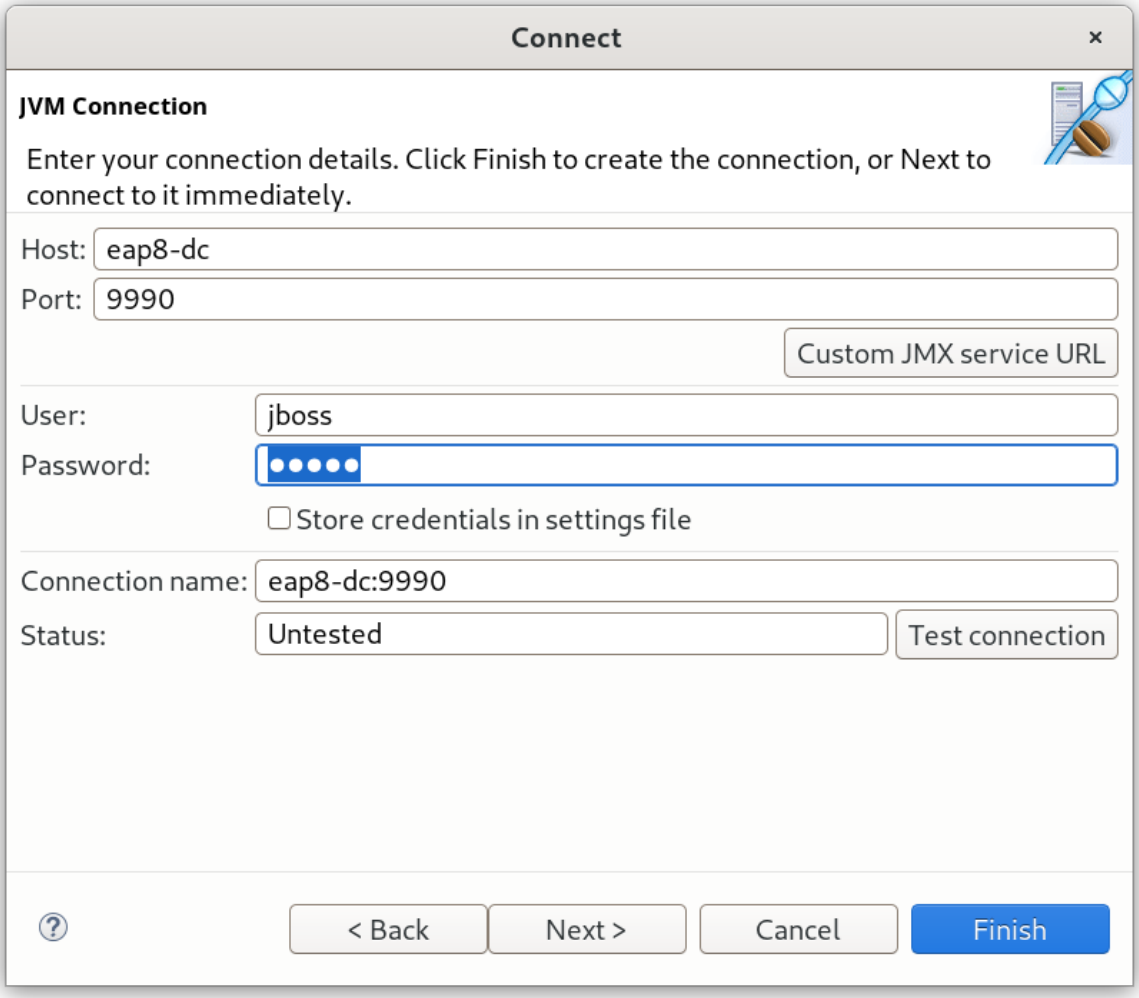
1. Java Mission Control を開きます。
2. **File** メニューで、**Connect** を選択します。
3. **Connect** ウィンドウで、**Create a new connection** を選択し、**Next** をクリックします。

図3.4 JMC の Connect ウィンドウ




4. **JVM Connection** ウィンドウで、プロファイリングするリモート JBoss EAP JVM の詳細を入力します。

図3.5 JMC の JVM 接続の詳細




a. **Host** フィールドに、ホスト名または IP アドレスを追加します。

b. **Port** フィールドにポート番号を追加します。

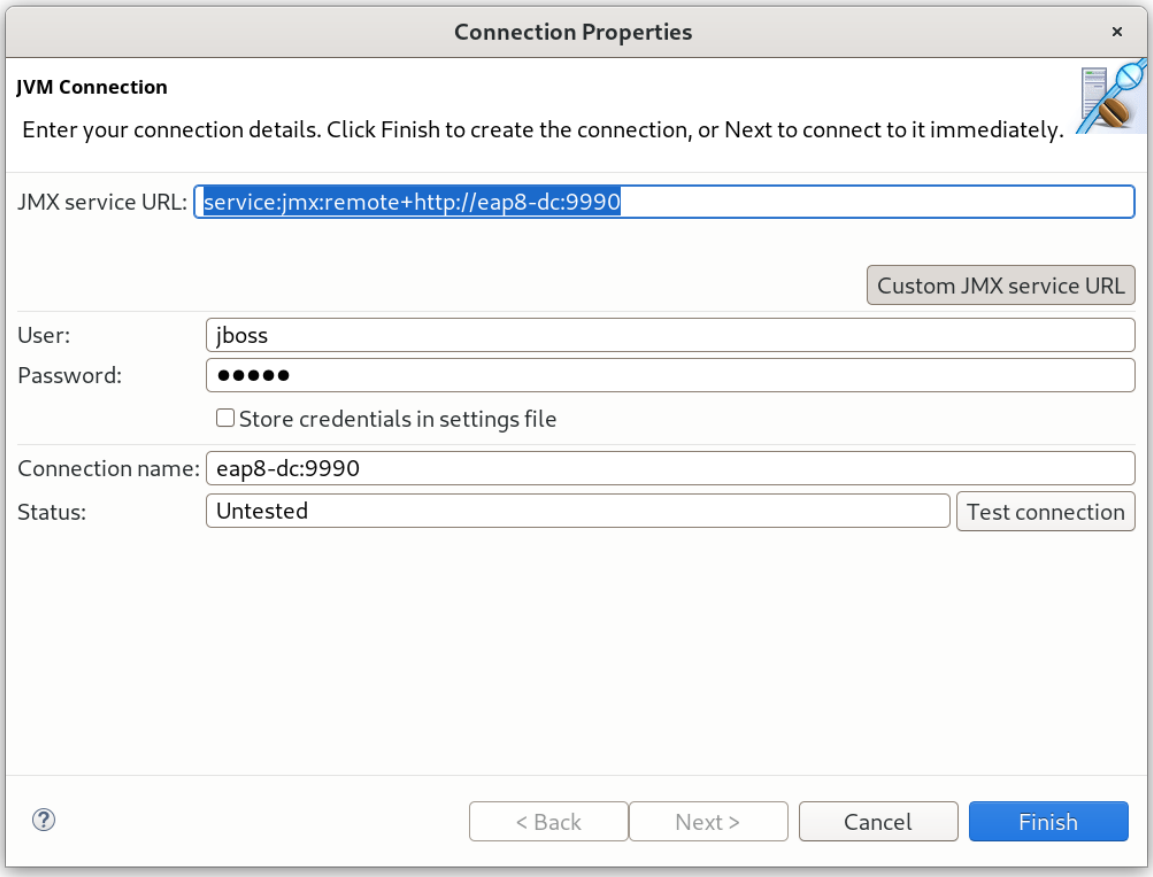
c. **User** フィールドに、**ApplicationRealm** に作成したユーザーを追加します。

d. **Password** フィールドに、**ApplicationRealm** に作成したパスワードを追加します。

e. **オプション**: 認証情報を設定ファイルに保存するには、**Store credentials in settings file**の横のチェックボックスをクリックします。

5. **Custom JMX Service URL**をクリックして、デフォルト設定をオーバーライドします。

図3.6 JVM 接続の JMX Service URL

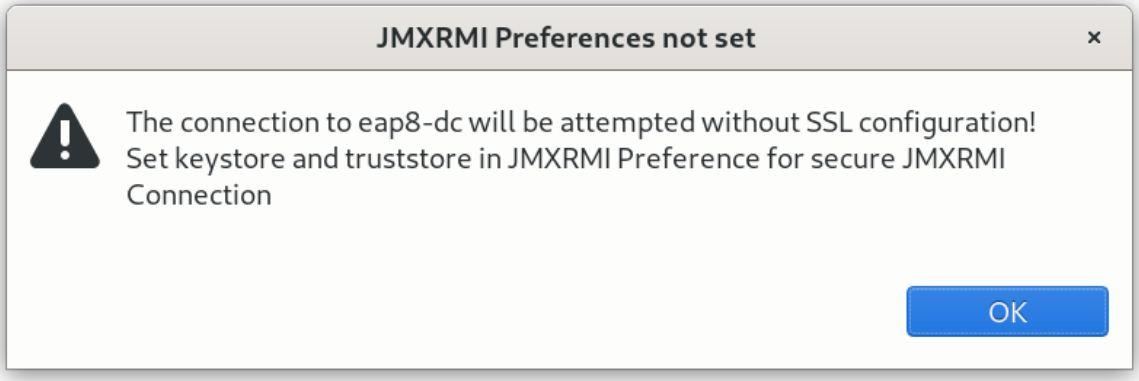



6. JMX サービス URL を変更して、JBoss リモートプロトコルを定義します。

```
service:jmx:remote+http://<host>:9990
```

7. **Test connection** をクリックして設定を確認します。
8. **Finish** をクリックして設定を保存します。
9. **JMXRMI Preferences not set** という警告メッセージが表示されます。

図3.7 JMXRMI 設定の警告メッセージ



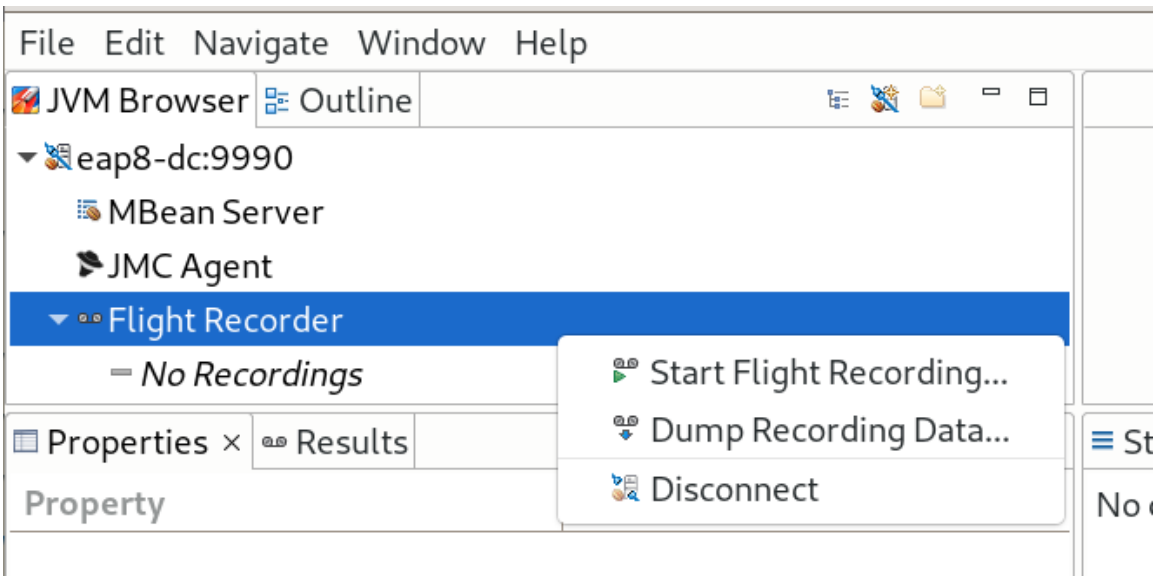

a. **OK** をクリックして接続の試行を許可します。

10. **JVM Browser** ペインで、プロファイリングする JVM を選択します。

a. JVM のドロップダウンメニューを展開して、**Flight Recorder** 項目を表示します。

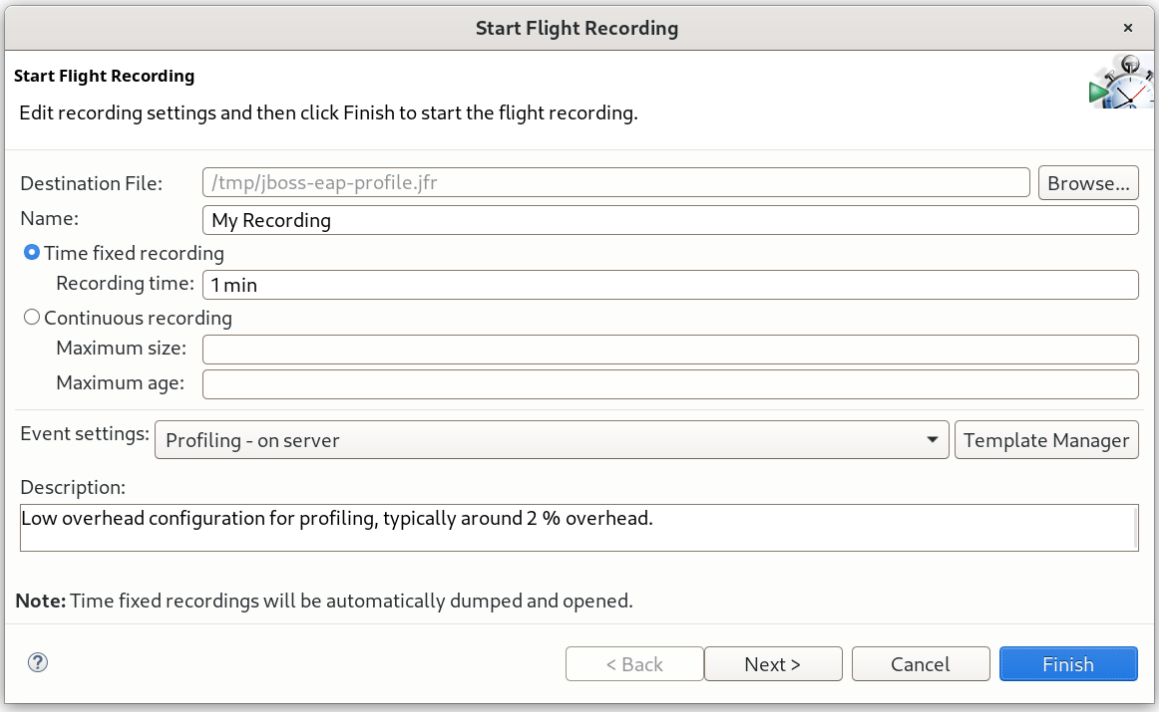
b. **Flight Recorder** を右クリックしてサブメニューを開き、**Start Flight Recording...** を選択します。

**図3.8 JMC プロファイルメニューを使用したリモート JVM への接続**



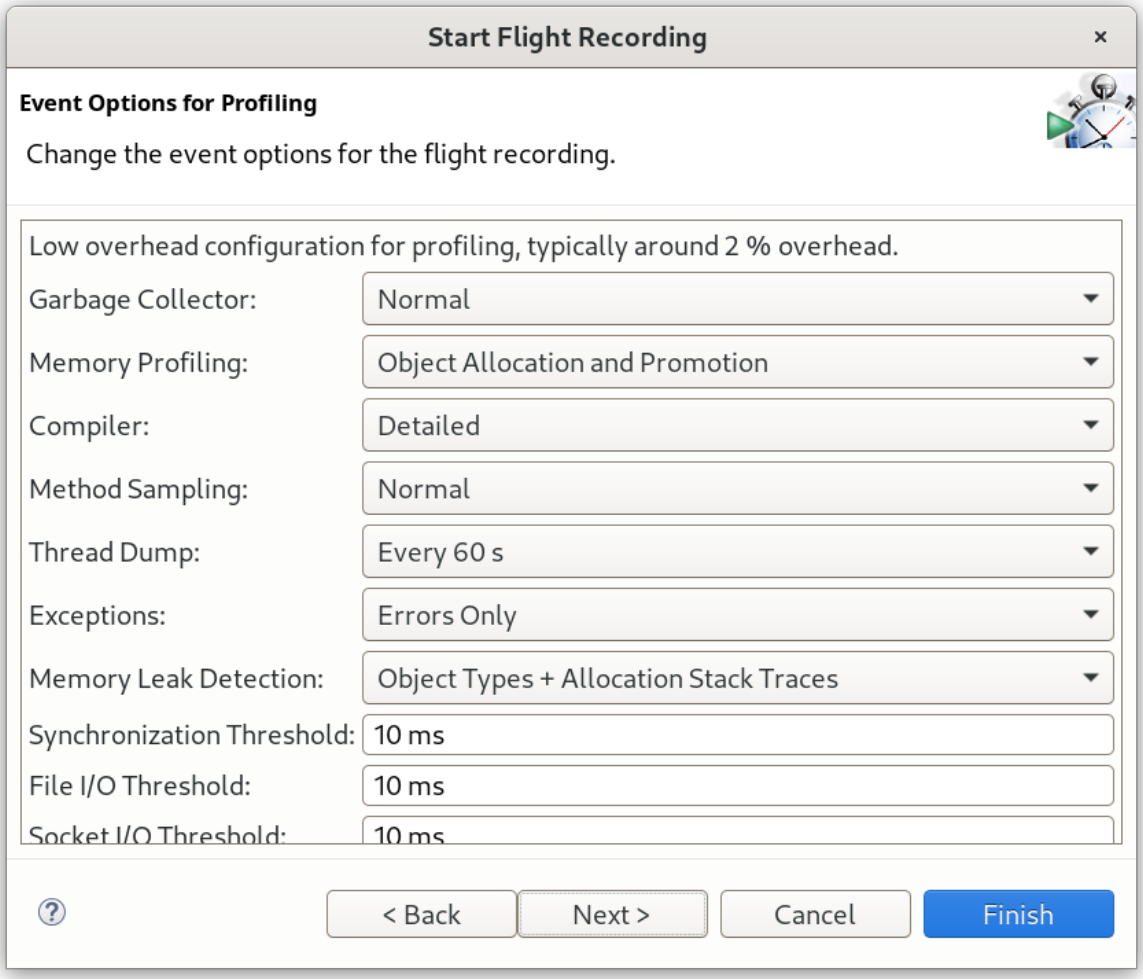

11. **Start Flight Recording** ウィンドウで、プロファイリングのオプションを設定します。

**図3.9 JVM プロファイリング設定**



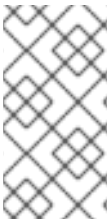

12. **Next** をクリックし、低レベルの詳細な設定を行います。

図3.10 JVM プロファイリングの詳細設定




13. **Finish** をクリックしてプロファイリングを開始します。

## 3.4. 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。

## 3.5. マネージドエグゼキュータサービスおよびマネージドスケジュールエグゼキュータサービスのランタイム統計

管理 CLI 属性で生成された実行時統計を表示することにより、マネージドエグゼキューターサービスおよびマネージドスケジュール済みエグゼキューターサービスのパフォーマンスを監視できます。スタンドアロンサーバーまたはホストにマップされた個々のサーバーの実行時統計を表示できます。

**重要**

**domain.xml** 設定には、実行時統計管理 CLI 属性のリソースが含まれていないため、管理 CLI 属性を使用してマネージドドメインの実行時統計を表示することはできません。

**表3.1 管理されたエグゼキュータサービスおよび管理されたスケジュールされたエグゼキュータサービスのパフォーマンスを監視するための管理 CLI 属性を表示します。**

| 属性 | 説明 |
|---|---|
| active-thread-count | タスクをアクティブに実行しているスレッドの概算数。 |
| completed-task-count | 実行を完了したタスクのおおよその総数。 |
| hung-thread-count | ハングしているエグゼキュータスレッドの数。 |
| max-thread-count | エグゼキュータスレッドの最大数。 |
| current-queue-size | エグゼキュータのタスクキューの現在のサイズ。 |
| task-count | 実行用に送信されたタスクの総数 (概算) |
| thread-count | エグゼキュータスレッドの現在の数。 |

**スタンドアロンサーバーで実行しているマネージドエグゼキュータサービスのランタイム統計を表示する例。**

```
[standalone@localhost:9990 /] /subsystem=ee/managed-executor-service=default:read-resource(include-runtime=true,recursive=true)
```

**スタンドアロンサーバーで実行しているマネージドスケジュールエグゼキュータサービスの実行時統計の例。**

```
[standalone@localhost:9990 /] /subsystem=ee/managed-scheduled-executor-service=default:read-resource(include-runtime=true,recursive=true)
```

**ホストにマップされたサーバーで実行しているマネージドエグゼキューターサービスのランタイム統計を表示する例。**

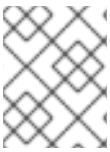
```
[domain@localhost:9990 /] /host=<host_name>/server=<server_name>/subsystem=ee/managed-executor-service=default:read-resource(include-runtime=true,recursive=true)
```

**ホストにマップされたサーバーで実行しているマネージドスケジュールエグゼキューターサービスのランタイム統計の例。**

```
[domain@localhost:9990 /] /host=<host_name>/server=<server_name>/subsystem=ee/managed-scheduled-executor-service=default:read-resource(include-runtime=true,recursive=true)
```

# 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。

## 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. ガベージコレクターの設定

Java 8 のサーバークラスマシンのデフォルトのガベージコレクター は、並列ガベージコレクター (別称スループットガベージコレクター) です。ただし、Red Hat は Java 9 以降からデフォルトとなる予定の G1 ガベージコレクターの使用を推奨します。通常、G1 ガベージコレクターは、ほとんどの状況で CMS および並列ガベージコレクターよりも優れたパフォーマンスを発揮します。

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
```

### 4.2.1. ガベージコレクションのロギングオプション

ガベー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。

## 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。
    - 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ーポリシー** 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. **メモリー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. 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。

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。

**関連情報**

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。

## 4.5. ホストコントローラーとプロセスコントローラーの JVM の調整

JBoss EAP マネージド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 JVM 設定を調整できますが、大きなマネージドドメイン環境でも、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 のマネージドドメインサイズでテストされています。

大規模なマネージドドメインで問題が発生した場合は、環境内のホストコントローラーまたはプロセスコントローラーの JVM を監視して、ヒープサイズなどの JVM オプションの適切な値を決定する必要がある場合があります。

# 第5章 JAKARTA ENTERPRISE BEANS 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した開発環境で、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を推奨します。

## 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特定のエンタープライズ bean のインスタンス数が制限されます。特定のエンタープライズ bean 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。

エンタープライズ Bean が特定のインスタンスプールを使用するように設定できます。これにより、各エンタープライズ bean タイプで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 **Jakarta Enterprise Beans** 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

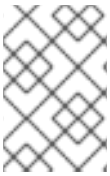
特定の bean が使用する特定のインスタンスプールを設定するには、**@org.jboss.ejb3.annotation.Pool** アノテーションを使用するか、bean の **jboss-ejb3.xml** デプロイメント記述子を編集します。

### 5.1.3. デフォルトの 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、エンタープライズ bean 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、スレッドが新しいエンタープライズ bean でメソッドを呼び出す必要があるときに新しいエンタープライズ bean インスタンスが作成されます。これは、作成されたエンタープライズ bean 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**

If a bean is configured to use a particular bean instance pool, disabling the default instance pool does not affect the pool that the bean uses.

## 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エンタープライズ bean とエンタープライズ bean 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、リモートエンタープライズ bean 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、各エンタープライズ bean 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 **Jakarta Enterprise Beans** 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にエンタープライズ Bean サービスを設定する

特定の bean スレッドプールを使用するよう、エンタープライズ bean 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 エンタープライズ bean 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 **Enterprise Bean** 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エンタープライズ bean 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エンタープライズ bean サービスに置き換えてください。

- エンタープライズ bean 非同期呼び出しサービスの場合は **async**
- エンタープライズ bean 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エンタープライズ bean 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. BEAN のランタイムデプロイメント情報

パフォーマンス監視のために、Bean にランタイムデプロイメント情報を追加できます。

利用可能なランタイムデータの詳細は、JBoss EAP 管理モデルの **ejb3** サブシステムを参照してください。アプリケーションには、Bean コードまたはデプロイメント記述子にアノテーションとしてランタイムデータが含まれています。アプリケーションは両方のオプションを使用できます。

**関連情報**

- 利用可能なランタイムデータの詳細は、JBoss EAP 管理モデルの **ejb3** サブシステム を参照してください。

### 5.3.1. Jakarta Enterprise Beans からランタイムデータを取得するためのコマンドラインオプション

Jakarta Enterprise Beans のランタイムデータは管理 CLI から利用できるため、Jakarta Enterprise Beans のパフォーマンスを評価できます。

すべてのタイプの Bean のランタイムデータを取得するコマンドは、次のパターンを使用します。

```
/deployment=<deployment_name>/subsystem=ejb3/<bean_type>=<bean_name>:read-resource(include-runtime)
```

**<deployment_name>** を、ランタイムデータを取得するデプロイメント **.jar** ファイルの名前に置き換えます。**<bean_type>** を、ランタイムデータを取得する Bean のタイプに置き換えます。このプレースホルダーには、次のオプションが有効です。

- **stateless-session-bean**
- **stateful-session-bean**
- **singleton-bean**
- **message-driven-bean**

**<bean_name>** を、ランタイムデータを取得する Bean の名前に置き換えます。

システムは、Java Script Object Notation (JSON) データとしてフォーマットされた **stdout** に結果を配信します。

**ejb-management.jar という名前のファイルにデプロイされた ManagedSingletonBean という名前のシングルトン Bean のランタイムデータを取得するコマンドの例**

```
/deployment=ejb-management.jar/subsystem=ejb3/singleton-bean=ManagedSingletonBean:read-resource(include-runtime)
```

**シングルトン Bean の出力ランタイムデータの例**

```
{
    "outcome" => "success",
    "result" => {
        "async-methods" => ["void async(int, int)"],
        "business-local" => ["sample.ManagedSingletonBean"],
        "business-remote" => ["sample.BusinessInterface"],
        "component-class-name" => "sample.ManagedSingletonBean",
```

```
        "concurrency-management-type" => undefined,
        "declared-roles" => [
            "Role3",
            "Role2",
            "Role1"
        ],
        "depends-on" => undefined,
        "execution-time" => 156L,
        "init-on-startup" => false,
        "invocations" => 3L,
        "jndi-names" => [
            "java:module/ManagedSingletonBean!sample.ManagedSingletonBean",
            "java:global/ejb-management/ManagedSingletonBean!sample.ManagedSingletonBean",
            "java:app/ejb-management/ManagedSingletonBean!sample.ManagedSingletonBean",
            "java:app/ejb-management/ManagedSingletonBean!sample.BusinessInterface",
            "java:global/ejb-management/ManagedSingletonBean!sample.BusinessInterface",
            "java:module/ManagedSingletonBean!sample.BusinessInterface"
        ],
        "methods" => {"doIt" => {
            "execution-time" => 156L,
            "invocations" => 3L,
            "wait-time" => 0L
        }},
        "peak-concurrent-invocations" => 1L,
        "run-as-role" => "Role3",
        "security-domain" => "other",
        "timeout-method" => "public void sample.ManagedSingletonBean.timeout(javax.ejb.Timer)",
        "timers" => [{
            "time-remaining" => 4304279L,
            "next-timeout" => 1577768415000L,
            "calendar-timer" => true,
            "persistent" => false,
            "info" => "timer1",
            "schedule" => {
                "year" => "*",
                "month" => "*",
                "day-of-month" => "*",
                "day-of-week" => "*",
                "hour" => "0",
                "minute" => "0",
                "second" => "15",
                "timezone" => undefined,
                "start" => undefined,
                "end" => undefined
            }
        }],
        "transaction-type" => "CONTAINER",
        "wait-time" => 0L,
        "service" => {"timer-service" => undefined}
    }
}
```

**ejb-management.jar という名前のファイルにデプロイされた NoTimerMDB という名前のメッセージ駆動型 Bean のランタイムデータを取得するコマンドの例**

```
/deployment=ejb-management.jar/subsystem=ejb3/message-driven-bean=NoTimerMDB:read-
resource(include-runtime)
```

**メッセージ駆動型 Bean の出力例**

```
{
    "outcome" => "success",
    "result" => {
        "activation-config" => [
            ("destination" => "java:/queue/NoTimerMDB-queue"),
            ("destinationType" => "javax.jms.Queue"),
            ("acknowledgeMode" => "Auto-acknowledge")
        ],
        "component-class-name" => "sample.NoTimerMDB",
        "declared-roles" => [
            "Role3",
            "Role2",
            "Role1"
        ],
        "delivery-active" => true,
        "execution-time" => 0L,
        "invocations" => 0L,
        "message-destination-link" => "queue/NoTimerMDB-queue",
        "message-destination-type" => "javax.jms.Queue",
        "messaging-type" => "javax.jms.MessageListener",
        "methods" => {},
        "peak-concurrent-invocations" => 0L,
        "pool-available-count" => 16,
        "pool-create-count" => 0,
        "pool-current-size" => 0,
        "pool-max-size" => 16,
        "pool-name" => "mdb-strict-max-pool",
        "pool-remove-count" => 0,
        "run-as-role" => "Role3",
        "security-domain" => "other",
        "timeout-method" => undefined,
        "timers" => [],
        "transaction-type" => "CONTAINER",
        "wait-time" => 0L,
        "service" => undefined
    }
}
```

## 5.4. エンタープライズ BEAN サブシステムの調整の必要性を示す例外

- **ステートレスエンタープライズ bean 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)
```

- **エンタープライズ bean スレッドプールが小さすぎる、またはエンタープライズ bean 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
```

### 5.4.1. ステートフルセッション Bean のデフォルトのグローバルタイムアウト値

**ejb3** サブシステムでは、**default-stateful-bean-session-timeout** 属性を使用して、サーバーインスタンスにデプロイされたすべてのステートフルセッション bean (SFSB) にデフォルトのグローバルタイムアウト値を設定できます。

**default-stateful-bean-session-timeout** 属性を使用すると、**ejb3** サブシステムで次の管理 CLI 操作を使用できます。

- 属性の現在のグローバルタイムアウト値を表示するための管理 CLI での **read-attribute** 操作。
- 管理 CLI を使用して属性を設定するための **write-attribute** 操作。

属性の動作はサーバーモードによって異なります。以下に例を示します。

- スタンドアロンサーバーで実行している場合、設定された値はアプリケーションサーバーにデプロイされたすべての SFSB に適用されます。
- マネージドドメインでサーバーを実行している場合、サーバーグループ内のサーバーインスタンスにデプロイされたすべての SFSB は同時タイムアウト値を受け取ります。

**注記**

属性のグローバルタイムアウト値を変更する場合、更新された設定は新規デプロイメントにのみ適用されます。サーバーをリロードして、新しい設定を現在のデプロイメントに適用します。

デフォルトでは、属性値は **-1** で設定されています。つまり、デプロイされた SFSB はタイムアウトしないように設定されています。ただし、属性には次の 2 つのタイプの有効な値を設定できます。

- 属性値を **0** に設定すると、属性は、**ejb** コンテナーによる削除の対象となる SFSB をすぐにマークします。
- **0** より大きい属性値を設定すると、 SFSB は、**ejb** コンテナーが適格な SFSB を削除する前に、ミリ秒単位で指定された時間アイドル状態のままになります。

**注記**

**ejb-jar.xml** デプロイメント記述子にある既存の **@StatefulTimeout** アノテーションまたは **stateful-timeout** 要素を使用して、SFSB のタイムアウト値を設定できます。ただし、このような設定を設定すると、デフォルトのグローバルタイムアウト値が SFSB に上書きされます。

属性に設定した新しい値を確認するには、次の 2 つの方法があります。

- 管理 CLI で **read-attribute** 操作を使用します。
- サーバーの設定ファイルの **ejb3** サブシステムセクションを調べます。

**関連情報**

- 属性の現在のグローバルタイムアウト値を表示する方法の詳細は、**管理 CLI ガイド** の 属性値の表示 を参照してください。
- 属性の現在のグローバルタイムアウト値を更新する方法の詳細は、**管理 CLI ガイド** の 属性の更新 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

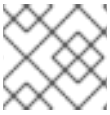
データソースの統計収集が有効になっている場合、データソースのランタイム統計を表示できます。

#### 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。

##### 6.1.1.1.1. 管理 CLI を使用したデータソース統計の有効化

以下の管理 CLI コマンドは、**ExampleDS** データソースの統計の収集を有効にします。



**注記**

マネージドドメインでは、このコマンドの前に **/profile=PROFILE_NAME** を付けます。

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

変更を反映するためにサーバーをリロードします。

##### 6.1.1.1.2. 管理コンソールを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。

**手順**

1. **Configuration → Subsystems → Datasources & Drivers → Datasources** と選択します。
2. データソースを選択し、**View** をクリックします。
3. **Attributes** タブ下の **Edit** をクリックします。
4. **Statistics Enabled** フィールドを **ON** に設定し、**Save** をクリックします。変更の反映にはリロードが必要であることを伝えるポップアップが表示されます。
5. サーバーをリロードします。

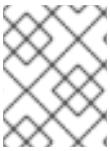
- スタンドアロンサーバーの場合は、ポップアップの **Reload** ボタンをクリックしてサーバーをリロードします。
- マネージドドメインの場合は、ポップアップの **Topology** リンクをクリックします。**Topology** タブで該当するサーバーを選択し、**Reload** ドロップダウンオプションを選択してサーバーをリロードします。

#### 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。

##### 6.1.1.2.1. 管理 CLI を使用したデータソース統計の表示

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



**注記**

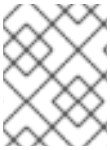
マネージドドメインでは、コマンドの前に **/host=HOST_NAME/server=SERVER_NAME** を追加します。

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

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

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

```
        "statistics-enabled" => true
    }
}
```



**注記**

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

#### 6.1.1.2.2. 管理コンソールを使用したデータソース統計の表示

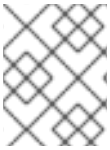
管理コンソールからデータソースの統計を表示するには、**Runtime** タブで **Datasources** サブシステムを選択し、データソースを選択してから **表示** をクリックします。

### 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。利用可能なすべての統計の詳細なリストについては、付録の **リソースアダプターの統計** を参照してください。

#### 6.1.2.1. 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。



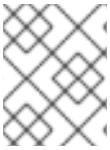
**注記**

マネージドドメインでは、このコマンドの前に **/host=HOST_NAME/server=SERVER_NAME/** を追加する必要があります。

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

#### 6.1.2.2. 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。

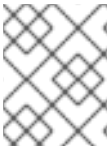


**注記**

マネージドドメインでは、このコマンドの前に **/host=HOST_NAME/server=SERVER_NAME/** を追加する必要があります。

```
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。

**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

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。

**前提条件**

- JDBC ドライバーをインストールします。
- データソースを作成します。

**手順**

- 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の設定

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。

**前提条件**

- 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。

**手順**

- 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** サブシステムのメッセージングサーバーの統計収集が有効になっている場合、メッセージングサーバーのリソースに関するランタイム統計を表示できます。

## 7.1. メッセージング統計の有効化

パフォーマンスに悪影響を及ぼす可能性があるため、**messaging-activemq** サブシステムの統計収集はデフォルトで有効になって**いません**。キューのメッセージ数やキューに追加されたメッセージ数など、基本情報を取得するためにキュー統計を有効にする必要はありません。これらの統計は、**statistics-enabled** を **true** に設定しなくてもキュー属性を使用して利用できます。

追加の統計収集は、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。

### 7.1.1. 管理 CLI を使用したメッセージング統計の有効化

以下の管理 CLI コマンドは、**default** メッセージングサーバーの統計の収集を有効にします。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=statistics-enabled,value=true)
```

プールされた接続ファクトリーの統計は、他のメッセージングサーバー統計とは別に有効になります。以下のコマンドを使用して、プールされた接続ファクトリーの統計を有効にします。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=activemq-ra: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。

### 7.1.2. 管理コンソールを使用したメッセージング統計の有効化

管理コンソールを使用してメッセージングサーバーの統計収集を有効にするには、以下の手順に従います。

**手順**

1. **設定 → サブシステム → Messaging (ActiveMQ) → サーバー**に移動します。
2. サーバーを選択し、**表示**をクリックします。
3. **統計**タブの下の**編集**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。

プールされた接続ファクトリーの統計は、他のメッセージングサーバー統計とは別に有効になります。プールされた接続ファクトリーの統計収集を有効にするには、以下の手順に従います。

1. **設定 → サブシステム → Messaging (ActiveMQ) → サーバー**に移動します。
2. サーバーを選択し、**接続**を選択し、**表示**をクリックします。
3. **Pooled Connection Factory** タブを選択します。
4. プールされた接続ファクトリーを選択し、**属性**タブで **編集**をクリックします。

5. **Statistics Enabled** フィールドを **ON** に設定し、**Save** をクリックします。

6. 変更を反映するためにサーバーをリロードします。

## 7.2. メッセージング統計の表示

管理 CLI または管理コンソールを使用して、メッセージングサーバーのランタイム統計を表示できます。

### 7.2.1. 管理 CLI を使用したメッセージング統計の表示

以下の管理 CLI コマンドを使用するとメッセージング統計を表示できます。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。

- キューの統計を表示します。

```
/subsystem=messaging-activemq/server=default/jms-queue=DLQ:read-resource(include-
runtime=true)
{
    "outcome" => "success",
    "result" => {
        "consumer-count" => 0,
        "dead-letter-address" => "jms.queue.DLQ",
        "delivering-count" => 0,
        "durable" => true,
        ...
    }
}
```

- トピックの統計を表示します。

```
/subsystem=messaging-activemq/server=default/jms-topic=testTopic:read-resource(include-
runtime=true)
{
    "outcome" => "success",
    "result" => {
        "delivering-count" => 0,
        "durable-message-count" => 0,
        "durable-subscription-count" => 0,
        ...
    }
}
```

- プールされた接続ファクトリーの統計を表示します。

```
/subsystem=messaging-activemq/server=default/pooled-connection-factory=activemq-
ra/statistics=pool:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        "ActiveCount" => 1,
        "AvailableCount" => 20,
        "AverageBlockingTime" => 0L,
        "AverageCreationTime" => 13L,
```

```
        "AverageGetTime" => 14L,
        ...
    }
}
```

**注記**

プールされた接続ファクトリーの統計は、他のメッセージングサーバー統計とは別に有効になります。

### 7.2.2. 管理コンソールを使用したメッセージング統計の表示

管理コンソールからメッセージング統計を表示するには、**ランタイム**タブで Messaging (ActiveMQ) サブシステムに移動し、サーバーを選択します。統計を表示する宛先を選択します。

**注記**

**準備済みトランザクション**ページでは、準備済みトランザクションを表示、コミット、およびロールバックできます。

## 7.3. メッセージカウンターの設定

メッセージングサーバーの以下のメッセージカウンター属性を設定できます。

- **message-counter-max-day-history**: メッセージカウンター履歴が保持される日数。
- **message-counter-sample-period**: キューのサンプルとなる頻度 (ミリ秒単位)。これらのオプションを設定する管理 CLI コマンドは以下の構文を使用します。`STATISTICS_NAME` と `STATISTICS_VALUE` は、設定する統計の名前と値に置き換えてください。

```
/subsystem=messaging-activemq/server=default::write-
attribute(name=STATISTICS_NAME,value=STATISTICS_VALUE)
```

たとえば、以下のコマンドを使用して **message-counter-max-day-history** を 5 日に設定し、**message-counter-sample-period** を 2 秒に設定します。

```
/subsystem=messaging-activemq/server=default:write-attribute(name=message-counter-max-day-
history,value=5)
/subsystem=messaging-activemq/server=default:write-attribute(name=message-counter-sample-
period,value=2000)
```

## 7.4. キューのメッセージカウンターと履歴の表示

以下の管理 CLI 操作を使用して、キューのメッセージカウンターとメッセージカウンター履歴を表示できます。

- **list-message-counter-as-json**
- **list-message-counter-as-html**
- **list-message-counter-history-as-json**
- **list-message-counter-history-as-html**

これらの値の表示に使用する管理 CLI コマンドは、以下の構文を使用します。`QUEUE_NAME` と `OPERATION_NAME` は、使用するキュー名と操作に置き換えてください。

```
/subsystem=messaging-activemq/server=default/jms-queue=QUEUE_NAME:OPERATION_NAME
```

たとえば、以下のコマンドを使用して、JSON 形式で **TestQueue** キューのメッセージカウンターを表示します。

```
/subsystem=messaging-activemq/server=default/jms-queue=TestQueue:list-message-counter-as-
json
{
    "outcome" => "success",
    "result" => "
{\"destinationName\":\"TestQueue\",\"destinationSubscription\":null,\"destinationDurable\":true,\"count\":
0,\"countDelta\":0,\"messageCount\":0,\"messageCountDelta\":0,\"lastAddTimestamp\":\"12/31/69
7:00:00 PM\",\"updateTimestamp\":\"2/20/18 2:24:05 PM\"}"
}
```

## 7.5. キューのメッセージカウンターのリセット

**reset-message-counter** 管理 CLI 操作を使用して、キューのメッセージカウンターをリセットできます。

```
/subsystem=messaging-activemq/server=default/jms-queue=TestQueue:reset-message-counter
{
    "outcome" => "success",
    "result" => undefined
}
```

## 7.6. 管理コンソールを使用したランタイム操作

管理コンソールを使用して、以下を操作できます。

- 別のメッセージングサーバーへの強制フェイルオーバーを実行する
- メッセージングサーバーのすべてのメッセージカウンターをリセットする
- メッセージングサーバーのすべてのメッセージカウンター履歴をリセットする
- メッセージングサーバーに関連した情報を表示する
- メッセージングサーバーの接続を閉じる
- トランザクションをロールバックする
- トランザクションをコミットする

### 7.6.1. 別のメッセージングサーバーへの強制フェイルオーバーの実行

管理コンソールを使用して、別のメッセージングサーバーへの強制フェイルオーバーを実行できます。

**手順**

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックします。
3. **表示**の横にある矢印ボタンをクリックして、Force Failover をクリックします。
4. Force Failover ウィンドウで、Yes をクリックします。

### 7.6.2. メッセージングサーバーのすべてのメッセージカウンターのリセット

管理コンソールを使用して、メッセージングサーバーのすべてのメッセージカウンターをリセットできます。

**手順**

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックします。
3. **表示**の横にある矢印ボタンをクリックして、**リセット**をクリックします。
4. **リセット**ウィンドウで、**すべてのメッセージカウンターをリセットする**の横にあるトグルボタンをクリックして機能を有効にします。
   ボタンが青色の背景で **ON** になりました。
5. **リセット**をクリックします。

### 7.6.3. メッセージングサーバーのメッセージカウンター履歴のリセット

管理コンソールを使用して、メッセージングサーバーのメッセージカウンター履歴をリセットできます。

**手順**

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックします。
3. **表示**の横にある矢印ボタンをクリックして、**リセット**をクリックします。
4. **リセット**ウィンドウで、**すべてのメッセージカウンター履歴をリセットする**の横にあるトグルボタンをクリックして機能を有効にします。
   ボタンが青色の背景で **ON** になりました。

5. **リセット**をクリックします。

### 7.6.4. メッセージングサーバーに関連した情報を表示する

管理コンソールを使用して、メッセージングサーバーに関連する以下の情報のリストを表示できます。

- 接続
- コンシューマー
- プロデューサー
- コネクター
- ロール
- トランザクション

メッセージングサーバーに関連する情報を表示するには、以下を行います。

**手順**

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックし、**表示**をクリックします。
3. ナビゲーションペインの該当する項目をクリックして、右側のペインの項目のリストを表示します。

### 7.6.5. メッセージングサーバーの接続を閉じる

IP アドレス、ActiveMQ アドレス一致、またはユーザー名を指定して接続を閉じることができます。

メッセージングサーバーの接続を閉じるには、以下を行います。

**手順**

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックし、**表示**をクリックします。
3. ナビゲーションペインで、**接続**をクリックします。
4. **閉じる**ウィンドウで、閉じる接続に基づいて該当するタブをクリックします。
5. 選択内容に基づいて、IP アドレス、ActiveMQ アドレス一致、またはユーザー名を入力し、**閉じる**をクリックします。

### 7.6.6. メッセージングサーバーのトランザクションのロールバック

管理コンソールを使用して、メッセージングサーバーのトランザクションをロールバックできます。

**手順**

1. 管理コンソールにアクセスし、以下のいずれかを使用して、**サーバー**に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックし、**表示**をクリックします。
3. ナビゲーションペインで、**Transactions** をクリックします。
4. ロールバックするトランザクションを選択し、**ロールバック**をクリックします。

### 7.6.7. メッセージングサーバーのトランザクションのコミット

管理コンソールを使用して、メッセージングサーバーのトランザクションをコミットできます。

**手順**

1. 管理コンソールにアクセスし、以下のいずれかを使用して Server に移動します。
   - **ランタイム → 参照 → ホスト → ホスト → サーバー**
   - **ランタイム → 参照 → サーバーグループ → サーバーグループ → サーバー**
2. **メッセージング ActiveMQ → サーバー**をクリックし、**表示**をクリックします。
3. ナビゲーションペインで、**Transactions** をクリックします。
4. コミットするトランザクションを選択し、**コミット**をクリックします。

## 7.7. JAKARTA MESSAGING のチューニング

Jakarta Messaging API を使用する場合は、パフォーマンスを改善するためのヒントについて、以下の情報を確認してください。

- メッセージ ID を無効にします。
  メッセージ ID が必要ない場合は、**MessageProducer** クラスで **setDisableMessageID()** メソッドを使用して無効にします。値を true に設定すると、一意の ID 作成のオーバーヘッドがなくなり、メッセージのサイズが小さくなります。

- メッセージのタイムスタンプを無効にします。
  メッセージのタイムスタンプが必要ない場合は、**MessageProducer** クラスで **setDisableMessageTimeStamp()** メソッドを使用して無効にします。値を true に設定すると、タイムスタンプ作成のオーバーヘッドがなくなり、メッセージのサイズが小さくなります。

- **ObjectMessage** を使用しません。
  **ObjectMessage** は、シリアライズされたオブジェクトを含むメッセージを送信するために使用されます。つまり、メッセージの本文 (ペイロード) が、バイトストリームとしてネットワーク経由で送信されます。オブジェクトが小さくても Java のシリアライズ形式は非常に大きく、

ネットワーク上の多くのスペースを占有します。また、カスタムマーシャリング手法と比較すると低速です。**ObjectMessage** は、他のメッセージタイプの 1 つを使用できない場合にのみ使用します。たとえば、ランタイムまでペイロードタイプがわからない場合に使用します。

- **AUTO_ACKNOWLEDGE** を使用しません。
  コンシューマーで確認応答モードを選択すると、ネットワーク経由で送信される確認応答メッセージの送信により発生する追加のオーバーヘッドとトラフィックによってパフォーマンスに影響します。**AUTO_ACKNOWLEDGE** でこのオーバーヘッドが発生するのは、クライアント上で受信される各メッセージについてサーバーから確認応答の送信が必要であるためです。可能であれば **DUPS_OK_ACKNOWLEDGE** を使用して、レイジー方法でメッセージを確認応答します。**CLIENT_ACKNOWLEDGE** は、クライアントコードがメソッドを呼び出してメッセージを確認応答するか、トランザクションセッションの 1 つの確認応答またはコミットとして数多くの確認応答をまとめます。

- 永続メッセージを使用しません。
  デフォルトでは、Jakarta Messaging メッセージは永続化されます。永続メッセージが必要ない場合は、それらを **non-durable** に設定します。永続メッセージは、ストレージに永続化されるために多くのオーバーヘッドが発生します。

- **TRANSACTED_SESSION** モードを使用して、単一のトランザクションでメッセージを送受信します。
  単一のトランザクションでメッセージをバッチ処理することで、JBoss EAP に統合された ActiveMQ Artemis サーバーで必要なネットワークラウンドトリップは、送信または受信ごとではなく、コミット時に 1 つのみになります。

## 7.8. 永続性の調整

- メッセージジャーナルを独自の物理ボリュームに配置します。
  追加のみのジャーナルの利点の 1 つは、ディスクヘッド移動が最小限に抑えられることです。ディスクが共有されている場合は、この利点は失われます。トランザクションコーディネーター、データベース、その他のジャーナルなど、複数のプロセスが同じディスクから読み書きされる場合、ディスクヘッドが異なるファイル間でスキップする必要があるため、パフォーマンスに影響を及ぼします。ページングまたは大きいメッセージを使用している場合は、それらが別のボリュームに配置されていることを確認します。

- **journal-min-files** 値を調整します。
  **journal-min-files** パラメーターを、平均的に持続可能な割合に一致するファイル数に設定します。ジャーナルデータディレクトリーに新しいファイルが頻繁に作成される場合は、大量のデータが保持されるため、ファイルの最小数を増やす必要があります。これにより、新規データファイルを作成せず、ジャーナルを再利用できます。

- ジャーナルファイルサイズを最適化します。
  ジャーナルファイルのサイズは、ディスク上のシリンダーの容量に合わせて調整する必要があります。多くのシステムでは、デフォルト値の **10 MB** で十分です。

- **AIO** ジャーナルタイプを使用します。
  Linux オペレーティングシステムの場合、ジャーナルタイプは **AIO** のままにします。**AIO** の拡張性は、Java **NIO** よりも優れています。

- **journal-buffer-timeout** 値を調整します。
  **journal-buffer-timeout** 値を増やすと、スループットは向上しますが、待ち時間が犠牲になります。

- **journal-max-io** 値を調整します。

**AIO** を使用している場合は、**journal-max-io** パラメーター値を増やして、パフォーマンスを向上させることができる可能性があります。**NIO** を使用している場合は、この値を変更しないでください。

## 7.9. その他の調整オプション

このセクションでは、JBoss EAP メッセージングで、調整できる他の箇所を説明します。

- 非同期送信の確認応答を使用します。
  トランザクション以外の永続メッセージを送信する必要があり、**send()** の呼び出しが返るまでにメッセージがサーバーに到達したという保証が必要ない場合は、ブロック送信するように設定しないでください。代わりに、非同期送信の確認応答を使用して、別のストリームで返される送信の確認応答を取得します。ただし、サーバーがクラッシュした場合に、一部のメッセージが失わる可能性があります。

- **pre-acknowledge** モードを使用します。
  **pre-acknowledge** モードでは、メッセージはクライアントに送信される前に確認応答されます。これにより、ネットワーク上の確認応答トラフィックの量が減少します。ただし、クライアントがクラッシュすると、クライアントが再接続した場合にメッセージは再配信されません。

- セキュリティーを無効にします。
  **security-enabled** 属性を false に設定してセキュリティーを無効にすると、パフォーマンスが若干改善されます。

- 永続性を無効にします。
  **persistence-enabled** を **false** に設定すると、メッセージ永続性を完全にオフにできます。

- トランザクションを遅れて同期します。
  **journal-sync-transactional** を **false** に設定すると、トランザクションの永続的なパフォーマンスが向上しますが、障害時にトランザクションが失われる可能性がいくらかあります。

- 非トランザクションを遅れて同期します。
  **journal-sync-non-transactional** を **false** に設定すると、非トランザクションの永続的なパフォーマンスが向上しますが、障害時に永続メッセージが失われる可能性がいくらかあります。

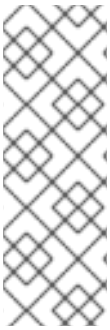
- メッセージを非ブロックで送信します。
  送信されるすべてのメッセージのネットワークラウンドトリップの待機を回避するには、Jakarta Messaging および JNDI を使用している場合は、**block-on-durable-send** と **block-on-non-durable-send** を **false** に設定するか、**setBlockOnDurableSend()** メソッドと **setBlockOnNonDurableSend()** メソッドを呼び出して **ServerLocator** に直接設定します。

- **consumer-window-size** を最適化します。
  高速なコンシューマーがある場合には、**consumer-window-size** を増やしてコンシューマーフロー制御を効果的に無効にできます。

- Jakarta Messaging API の代わりにコア API を使用します。
  Jakarta Messaging 操作は、サーバーが処理する前にコア操作に変換する必要があるため、コア API を使用する場合よりもパフォーマンスが低下します。コア API を使用する場合は、可能な限り **SimpleString** を取得するメソッドを使用するようにします。**SimpleString** は、**java.lang.String** とは異なり、ネットワークに書き込まれる前にコピーする必要がないため、呼び出し間で **SimpleString** インスタンスを再利用する場合に、不要なコピーを避けることができます。コア API は他のブローカーに移植できないことに注意してください。

## 7.10. アンチパターンの回避

- 可能な場合は、接続、セッション、コンシューマー、プロデューサーを再利用します。
  メッセージングの最も一般的なアンチパターンは、送信または消費されるメッセージごとに新しい接続、セッション、プロデューサーの作成です。これらのオブジェクトは作成に時間がかかり、複数のネットワークラウンドトリップを伴う可能性があるため、リソースの使用率が低くなります。常に再利用します。



**注記**

Spring Messaging Template テンプレートなどの一般的なライブラリーは、これらのアンチパターンを使用します。Spring Messaging Template を使用している場合は、パフォーマンスが低下する可能性があります。Spring Messaging Template は、たとえば Jakarta Connectors を使用して Jakarta Messaging セッションをキャッシュするアプリケーションサーバーでのみ安全に使用でき、その後メッセージを送信するためにのみ使用できます。アプリケーションサーバーであっても、同期的に消費するメッセージに安全に使用することはできません。

- サイズの大きいメッセージを使用しません。
  XML のような詳細形式は、ネットワーク上の多くのスペースを占有し、結果としてパフォーマンスが低下します。可能な場合は、メッセージ本文では XML を使用しません。

- 各リクエストに一時キューを作成しません。
  この一般的なアンチパターンには、一時キューの要求/応答パターンが含まれます。一時キュー要求/応答パターンにより、メッセージはターゲットに送信され、返信先ヘッダーはローカルの一時キューのアドレスで設定されます。受信側がメッセージを受信すると、メッセージを処理し、返信先ヘッダーに指定されたアドレスに応答を返します。このパターンでよくある間違いは、送信されるメッセージごとに新しい一時キューを作成することです。これにより、パフォーマンスが大幅に低下します。代わりに、一時キューを多くのリクエストに再利用する必要があります。

- 必要でない限り、メッセージ駆動 Bean は使用しないでください。
  MDB を使用したメッセージの消費は、単純な Jakarta Messaging メッセージコンシューマーを使用するメッセージの消費よりも遅くなります。

# 第8章 LOGGING サブシステムの調整

コンソールへのロギングを無効にして、適切なロギングレベルを設定し、最適なログファイル保存場所を指定すると、実稼働環境での JBoss EAP ロギングサブシステムのパフォーマンスをさらに向上できます。

## 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。デフォルトのマネージド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. ロギングレベルの設定

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。

## 8.3. ログファイルの場所の設定

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。

# 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 (NIO) **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。ご使用の環境に合わせて **undertow** サブシステムを調整する機会は次のとおりです。

- バッファーキャッシュ設定
- バイトバッファープールの設定
- Jakarta Server Pages 設定オプション
- リスナー設定オプション
- セッション属性のマーシャリング

## 9.1. バッファーキャッシュ設定

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格納します。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。最大の静的ファイルを完全に保存するようにバッファーサイズを設定します。

**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。

## 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。

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。

## 9.3. JAKARTA SERVER PAGES 設定オプション

次の Jakarta Server Pages 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 Jakarta Server Pages 設定オプションがあります。

**generate-strings-as-char-arrays**

Jakarta Server Pages に多くの String 定数が含まれている場合、このオプションを有効にすると、String 定数が char 配列に変換され、スクリプトレットが最適化されます。

**optimize-scriptlets**

Jakarta Server Pages に多くの String 連結が含まれている場合、このオプションを有効にすると、各 Jakarta Server Pages リクエストの String 連結が削除され、スクリプトレットが最適化されます。

**trim-spaces**

Jakarta Server Pages 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。

### 9.3.1. 管理コンソールを使用して Jakarta Server Pages オプションを有効にする

管理コンソールを使用して Undertow Jakarta Server Pages 設定オプションを有効にするには、次の手順を実行します。

**手順**

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. **Jakarta Server Pages** 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。

### 9.3.2. 管理 CLI を使用した Jakarta Server Pages オプションの有効化

管理 CLI を使用して Undertow Jakarta Server Pages 設定オプションを有効にするには、次の手順を実行します。

**手順**

- 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. リスナー設定オプション

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。

### 9.4.1. 管理コンソールを使用したリスナーオプションの設定

管理コンソールを使用してリスナーオプションを設定するには、次の手順を実行します。

**手順**

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。

### 9.4.2. 管理 CLI を使用したリスナーオプションの設定

管理 CLI を使用してリスナーオプションを設定するには、次の手順を実行します。

**手順**

- 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、Jakarta Enterprise Beans 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。

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。

### 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。

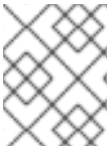
以下のコマンドは、**default** ワーカーのランタイム統計を表示します。

```
/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**

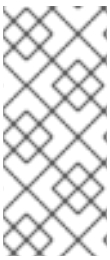
IO バッファープールは非推奨となっていますが、現行リリースではデフォルトとして設定されています。

**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 タスクの処理専用にできる個別のバッファープールを作成できます。

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。

# 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。



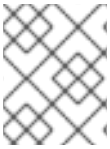
**注記**

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。

## 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。

**手順**

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**

マネージドドメインでは、これらのコマンドの前に **/profile=PROFILE_NAME** を追加してください。

```
/subsystem=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**

マネージドドメインでは、これらのコマンドの前に **/host=HOST_NAME/server=SERVER_NAME** を追加してください。

以下に例を示します。

- **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」の章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イ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。

**関連情報**

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。

### 11.3.1. 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。

**手順**

- 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** スタックの **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

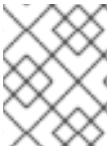
**jgroups** サブシステムは独自のスレッドプールをクラスター通信の処理に使用します。JGroups には個別に設定できる **default**、**internal**、**oob**、および **timer** 関数のスレッドプールが含まれます。各 JGroups スレッドプールには、**keepalive-time**、**max-threads**、**min-threads**、および **queue-length** の設定可能な属性が含まれます。

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。

## 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**

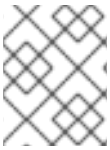
オペレーティングシステムが利用可能なバッファーサイズを制限し、JBoss EAP が設定済みのバッファーサイズを使用できないことがあります。

# 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 トランザクションの場合、ジャーナルストアのほうがはるかに効率的です。ジャーナルストアは、すべてのトランザクションに、1 つのファイルで構成されるジャーナルを使用するためです。

XA トランザクションのパフォーマンスを向上させるために、ジャーナルログストアを使用することを推奨します。Red Hat Enterprise Linux システムでは、ジャーナルストアの非同期 I/O (AIO) を追加で有効にすると、パフォーマンスをさらに向上できます。



**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、**libaio** パッケージがインストールされていることを確認してください。

## 12.1. 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

管理コンソールを使用して、ジャーナルログストアを有効にすることができます。

**手順**

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. **オプション**: 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。

## 12.2. 管理 CLI を使用したジャーナルログストアの有効化

管理 CLI を使用して、ジャーナルログストアを有効にすることができます。

**手順**

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. **オプション**: Red Hat Enterprise Linux システムの場合、次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 参考資料

## 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**

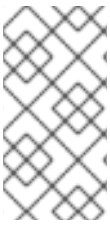
| 名前 | 説明 |
| --- | --- |
| PreparedStatementCacheAccessCount | ステートメントキャッシュがアクセスされた回数。 |
| PreparedStatementCacheAddCount | ステートメントキャッシュに追加されたステートメントの数。 |
| PreparedStatementCacheCurrentSize | ステートメントキャッシュに現在キャッシュされた準備済みおよび呼び出し可能ステートメントの数。 |
| PreparedStatementCacheDeleteCount | キャッシュから破棄されたステートメントの数。 |
| 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

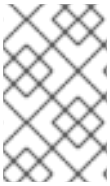
これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP_HOME/docs/schema/wildfly-io_2_0.xsd** のスキーマ定義ファイルで確認してください。

**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 |
| --- | --- | --- |
| 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。<br>● RAM が 64 MB 未満の場合は 512 バイト<br>● RAM が 64 - 128 MB の場合は 1024 バイト (1 KB)<br>● RAM が 128 MB を超える場合は 16384 バイト (16 KB) |
| 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。 |



**注記**

IO バッファープールは JBoss EAP 7.2 で非推奨になりました。現在のリリースでもデフォルトとして設定されていますが、今後のリリースでは Undertow バイトバッファープールに置き換えられる予定です。

**改訂日時: 2024-02-08**