



# Red Hat JBoss Enterprise Application Platform 7.2

## 設定ガイド

Red Hat JBoss Enterprise Application Platform 7.2 向け



# Red Hat JBoss Enterprise Application Platform 7.2 設定ガイド

---

Red Hat JBoss Enterprise Application Platform 7.2 向け

## 法律上の通知

Copyright © 2019 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## 概要

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

## 目次

<b>第1章 はじめに</b> .....	<b>7</b>
<b>第2章 JBOSS EAP の開始および停止</b> .....	<b>8</b>
2.1. JBOSS EAP の開始	8
2.2. JBOSS EAP の停止	9
2.3. JBOSS EAP の管理専用モードでの実行	9
2.4. JBOSS EAP の正常な一時停止およびシャットダウン	11
2.5. JBOSS EAP の起動および停止 (RPM インストール)	14
2.6. POWERSHELL スクリプト (WINDOWS SERVER)	17
<b>第3章 JBOSS EAP の管理</b> .....	<b>18</b>
3.1. サブシステム、エクステンション、およびプロファイル	18
3.2. 管理ユーザー	18
3.3. JBOSS EAP サーバー設定の最適化	21
3.4. 管理インターフェース	22
3.5. 管理 API	25
3.6. 設定データ	27
3.7. ファイルシステムパス	38
3.8. ディレクトリーのグループ化	41
3.9. システムプロパティー	43
3.10. 管理監査ロギング	45
3.11. サーバーライフサイクルイベントの通知	50
<b>第4章 ネットワークおよびポート設定</b> .....	<b>56</b>
4.1. インターフェース	56
4.2. ソケットバインディング	58
4.3. IPV6 アドレス	62
<b>第5章 JBOSS EAP のセキュリティー</b> .....	<b>64</b>
<b>第6章 JBOSS EAP クラスローディング</b> .....	<b>65</b>
6.1. モジュール	65
6.2. モジュールの依存性	66
6.3. カスタムモジュールの作成	67
6.4. カスタムモジュールの削除	70
6.5. グローバルモジュールの定義	71
6.6. サブデプロイメント分離の設定	72
6.7. 外部 JBOSS EAP モジュールディレクトリーの定義	72
6.8. 動的モジュールの命名規則	73
<b>第7章 アプリケーションのデプロイ</b> .....	<b>74</b>
7.1. 管理 CLI を使用したアプリケーションのデプロイ	74
7.2. 管理コンソールを使用したアプリケーションのデプロイ	78
7.3. デプロイメントスキャナーを使用したアプリケーションのデプロイ	80
7.4. MAVEN を使用したアプリケーションのデプロイ	82
7.5. HTTP API を使用したアプリケーションのデプロイ	85
7.6. デプロイメントの動作のカスタマイズ	86
7.7. 展開形式のデプロイメントの管理	92
7.8. デプロイメントのコンテンツの表示	93
<b>第8章 ドメイン管理</b> .....	<b>96</b>
8.1. 管理対象ドメイン	96
8.2. ドメイン設定のナビゲート	98

8.3. 管理対象ドメインの起動	100
8.4. サーバーの管理	103
8.5. ドメインコントローラーの検出およびフェイルオーバー	106
8.6. 管理対象ドメインの設定	109
8.7. 複数の JBOSS EAP バージョンの管理	111
8.8. JBOSS EAP プロファイルの管理	118
<b>第9章 JVM の設定</b>	<b>120</b>
9.1. スタンドアロンサーバーの JVM 設定	120
9.2. 管理対象ドメインの JVM 設定	120
9.3. JVM 状態の表示	122
9.4. JVM の調整	122
<b>第10章 MAIL サブシステム</b>	<b>123</b>
10.1. MAIL サブシステムの設定	123
10.2. カスタムトランスポートの設定	123
10.3. パスワードに認証情報ストアを使用	125
<b>第11章 JBOSS EAP を用いたロギング</b>	<b>126</b>
11.1. サーバーロギング	126
11.2. ログファイルの表示	130
11.3. LOGGING サブシステム	131
11.4. ログカテゴリーの設定	137
11.5. ログハンドラーの設定	139
11.6. ルートロガーの設定	159
11.7. ログフォーマッターの設定	159
11.8. アプリケーションのロギング	165
11.9. LOGGING サブシステムの調整	169
<b>第12章 データソース管理</b>	<b>170</b>
12.1. JBOSS EAP データソース	170
12.2. JDBC ドライバー	170
12.3. データソースの作成	175
12.4. データソースの編集	178
12.5. データソースの削除	180
12.6. データソース接続のテスト	180
12.7. データソース接続のフラッシュ	181
12.8. XA データソースのリカバリー	181
12.9. データベース接続の検証	185
12.10. データソースセキュリティー	187
12.11. データソースの統計	192
12.12. データソースの調整	194
12.13. キャパシティーポリシー	194
12.14. エンリストメントトレース	196
12.15. データソース設定例	197
<b>第13章 AGROAL でのデータソース管理</b>	<b>224</b>
13.1. AGROAL データソースサブシステム	224
13.2. AGROAL データソースサブシステムの有効化	224
13.3. AGROAL データソースに対するコアモジュールとしての JDBC ドライバーのインストール	225
13.4. AGROAL 非 XA データソースの設定	226
13.5. AGROAL XA データソースの設定	227
13.6. AGROAL データソースの設定例	227
<b>第14章 TRANSACTIONS サブシステムの設定</b>	<b>231</b>

<b>第15章 ORB 設定</b> .....	<b>232</b>
15.1. COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)	232
15.2. JTS トランザクション用 ORB の設定	232
15.3. ELYTRON サブシステムで SSL/TLS を使用するよう IIOP を設定	233
<b>第16章 JAVA CONNECTOR ARCHITECTURE (JCA) の管理</b> .....	<b>235</b>
16.1. JAVA CONNECTOR ARCHITECTURE (JCA)	235
16.2. リソースアダプター	235
16.3. JCA サブシステムの設定	235
16.4. リソースアダプターの設定	241
16.5. 管理接続プールの設定	247
16.6. 接続統計の表示	248
16.7. リソースアダプター接続のフラッシュ	248
16.8. RESOURCE ADAPTERS サブシステムの調整	249
<b>第17章 WEB サーバーの設定 (UNDERTOW)</b> .....	<b>250</b>
17.1. UNDERTOW サブシステムの概要	250
17.2. バッファークャッシュの設定	253
17.3. バイトバッファープールの設定	253
17.4. サーバーの設定	254
17.5. サブレットコンテナの設定	255
17.6. サブレット拡張の設定	257
17.7. ハンドラーの設定	257
17.8. フィルターの設定	258
17.9. デフォルトの WELCOME WEB アプリケーションの設定	260
17.10. HTTPS の設定	262
17.11. HTTP セッションタイムアウトの設定	262
17.12. HTTP のみのセッション管理クッキーの設定	262
17.13. HTTP/2 の設定	263
17.14. REQUESTDUMPING ハンドラーの設定	265
17.15. クッキーセキュリティーの設定	266
17.16. UNDERTOW サブシステムの調整	266
<b>第18章 リモータイングの設定</b> .....	<b>268</b>
18.1. REMOTING サブシステム	268
18.2. エンドポイントの設定	269
18.3. コネクターの設定	269
18.4. HTTP コネクターの設定	270
18.5. アウトバウンド接続の設定	270
18.6. リモートアウトバウンド接続の設定	271
18.7. ローカルアウトバウンド接続の設定	271
18.8. リモータイングの追加設定	272
<b>第19章 IO サブシステムの設定</b> .....	<b>274</b>
19.1. IO サブシステムの概要	274
19.2. ワーカーの設定	274
19.3. バッファープールの設定	274
19.4. IO サブシステムの調整	275
<b>第20章 WEB サービスの設定</b> .....	<b>276</b>
<b>第21章 JAVASERVER FACES (JSF) の設定</b> .....	<b>277</b>
21.1. JAVASERVER FACES の複数の JSF 実装	277
21.2. DOCTYPE 宣言の拒否	279

<b>第22章 バッチアプリケーションの設定</b> .....	<b>281</b>
22.1. BATCH ジョブの設定	281
22.2. バッチジョブの管理	283
22.3. バッチジョブのセキュリティー設定	285
<b>第23章 NAMING サブシステムの設定</b> .....	<b>286</b>
23.1. NAMING サブシステム	286
23.2. グローバルバインディングの設定	286
23.3. JNDI バンディングの動的な変更	289
23.4. リモート JNDI インターフェースの設定	290
<b>第24章 高可用性の設定</b> .....	<b>291</b>
24.1. 高可用性	291
24.2. JGROUPS を用いたクラスター通信	292
24.3. INFINISPAN	305
24.4. JBOSS EAP をフロントエンドロードバランサーとして設定	321
24.5. 外部 WEB サーバーのプロキシサーバーとしての使用	324
24.6. MOD_CLUSTER HTTP コネクター	327
24.7. APACHE MOD_JK HTTP コネクター	338
24.8. APACHE MOD_PROXY HTTP コネクター	341
24.9. MICROSOFT ISAPI コネクター	344
24.10. ORACLE NSAPI コネクター	350
<b>第25章 ECLIPSE MICROPROFILE</b> .....	<b>356</b>
25.1. ECLIPSE MICROPROFILE CONFIG を使用した設定の管理	356
25.2. MICROPROFILE OPENTRACING SMALLRYE サブシステムでのリクエストのトレース	359
25.3. ECLIPSE MICROPROFILE HEALTH を使用したサーバー状態の監視	361
<b>付録A リファレンス資料</b> .....	<b>363</b>
A.1. サーバーランタイム引数	363
A.2. RPM サービス設定ファイル	366
A.3. RPM サービス設定プロパティ	367
A.4. JBOSS EAP サブシステムの概要	368
A.5. ADD-USER ユーティリティー引数	373
A.6. 管理監査ロギング属性	375
A.7. インターフェース属性	377
A.8. ソケットバインディング属性	378
A.9. デフォルトのソケットバインディング	380
A.10. モジュールコマンド引数	384
A.11. デプロイメントスキャナーマーカーファイル	385
A.12. デプロイメントスキャナーの属性	386
A.13. 管理対象ドメインの JVM 設定属性	387
A.14. MAIL サブシステムの属性	389
A.15. ルートロガーの属性	391
A.16. ログカテゴリーの属性	391
A.17. ログハンドラーの属性	392
A.18. ログフォーマッター属性	400
A.19. データソース接続 URL	402
A.20. データソースの属性	403
A.21. データソースの統計	413
A.22. AGROAL データソースの属性	415
A.23. トランザクションマネージャーの設定オプション	417
A.24. IIOP サブシステムの属性	421
A.25. リソースアダプターの属性	424



---

A.26. リソースアダプターの統計	429
A.27. UNDERTOW サブシステムの属性	430
A.28. UNDERTOW サブシステムの統計	462
A.29. HTTP メソッドのデフォルトの動作	463
A.30. REMOTING サブシステムの属性	464
A.31. IO サブシステムの属性	471
A.32. JSF モジュールテンプレート	473
A.33. JGROUPS サブシステムの属性	478
A.34. JGROUPS プロトコル	482
A.35. MICROPROFILE CONFIG SMALLRYE サブシステムの属性	490
A.36. APACHE HTTP SERVER の MOD_CLUSTER ディレクティブ	491
A.37. MODCLUSTER サブシステムの属性	495
A.38. MOD_JK ワーカープロパティ	499
A.39. SECURITY MANAGER サブシステム	502
A.40. JBOSS CORE SERVICES からの OPENSLL のインストール	502
A.41. OPENSLL を使用するよう JBOSS EAP を設定	504
A.42. JAVA 8 向けに提供されるプラットフォームモジュール	505



## 第1章 はじめに

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



### 重要

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

## 第2章 JBOSS EAP の開始および停止

### 2.1. JBOSS EAP の開始

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

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

#### JBoss EAP をスタンドアロンサーバーとして起動

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



#### 注記

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

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

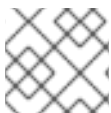
JBoss EAP はデフォルトで **standalone.xml** 設定ファイルを使用しますが、別の設定ファイルを使用して起動することもできます。利用できるスタンドアロン設定ファイルとそれらの使用方法については、「[スタンドアロンサーバー設定ファイル](#)」の項を参照してください。

使用できる起動スクリプトの引数の完全リストとそれら引数の目的については、**--help** 引数を使用するか、「[サーバーランタイム引数](#)」を参照してください。

#### 管理対象ドメインでの JBoss EAP の起動

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

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



#### 注記

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

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

JBoss EAP はデフォルトで **host.xml** ホスト設定ファイルを使用しますが、別の設定ファイルを使用して起動することもできます。利用できる管理対象ドメイン設定ファイルとそれらの使用方法については、「[管理対象ドメイン設定ファイル](#)」の項を参照してください。

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

## 2.2. JBOSS EAP の停止

JBoss EAP の停止方法は、開始した方法によって異なります。

### JBoss EAP の対話的なインスタンスの停止

JBoss EAP を起動したターミナルで **Ctrl+C** を押します。

### JBoss EAP のバックグラウンドインスタンスの停止

管理 CLI を使用して、稼働中のインスタンスへ接続し、サーバーをシャットダウンします。

1. 管理 CLI を起動します。

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

2. **shutdown** コマンドを実行します。

```
shutdown
```



#### 注記

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

## 2.3. JBOSS EAP の管理専用モードでの実行

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

### 管理専用モードでのスタンドアロンサーバーの実行

管理専用モードでのサーバーの起動

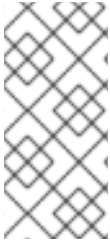
JBoss EAP インスタンスを管理専用モードで起動するには、JBoss EAP インスタンスの起動時に **--start-mode=admin-only** ランタイム引数を使用します。

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

サーバーが管理専用モードで実行されていることを確認

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

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



## 注記

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

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

管理 CLI から別のモードでリロード

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

サーバーを管理専用モードでリロードするには、以下を実行します。

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

サーバーを通常モードでリロードするには、以下を実行します。

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

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

## 管理専用モードでの管理対象ドメインの実行

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

管理専用モードでのホストコントローラーの起動

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

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

ホストコントローラーが管理専用モードで実行されていることを確認

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

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

管理 CLI から別のモードでリロード

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

ホストコントローラーを管理専用モードでリロードするには、以下を実行します。

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

ホストコントローラーを通常モードでリロードするには、以下を実行します。

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

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

## 2.4. JBOSS EAP の正常な一時停止およびシャットダウン

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

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

### Undertow

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

### mod\_cluster

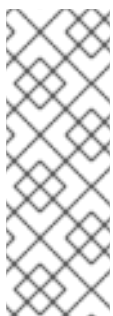
**modcluster** サブシステムは **PRE\_SUSPEND** フェーズでサーバーが一時停止することをロードバランサーに通知します。

### EJB

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

### Transactions

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



### 注記

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

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

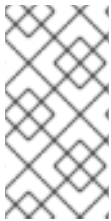
### EE Concurrency

サーバーはすべてのアクティブなジョブが終了するまで待機します。キューに置かれたジョブはすべてスキップされます。現在、EE Concurrency には永続性がないため、キューに置かれスキップされたジョブは失われます。

サーバーが一時停止状態である間、スケジュールされたタスクはスケジュールどおりの時間に実行されますが、**java.lang.IllegalStateException** が発生します。サーバーが再開されると、スケジュールされたタスクは通常どおり実行され、ほとんどの場合でタスクのスケジュールを変更する必要はありません。

### Batch

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



## 注記

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

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

### 2.4.1. サーバーの一時停止

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



## 注記

管理インターフェースのサーバーの一時停止による影響はありません。

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

#### サーバーの一時停止状態のチェック

以下の管理 CLI コマンドを使用するとサーバーの中断状態を表示できます。結果の値は、**RUNNING**、**PRE\_SUSPEND**、**SUSPENDING**、または **SUSPENDED** のいずれかになります。

- スタンドアロンサーバーの中断状態をチェックします。

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

- 管理対象ドメインのサーバーの中断状態をチェックします。

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

#### 中断

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

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

- スタンドアロンサーバーを一時停止します。

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

- 管理対象ドメインのすべてのサーバーを一時停止します。

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

- 管理対象ドメインの単一のサーバーを一時停止します。



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

- サーバークラスのすべてのサーバーを一時停止します。

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

#### 再開

**resume** コマンドを適切なレベル (サーバー、サーバークラス、ドメイン全体) で使用すると、サーバーが正常な実行状態に戻り、新しいリクエストを許可できるようになります。例を以下に示します。

```
:resume
```

#### サーバーを一時停止状態で起動

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

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

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

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

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



#### 注記

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

- 管理対象ドメインサーバークラスのすべてのサーバーを一時停止モードで起動するには、管理 CLI コマンドで **start-mode=suspend** 引数を **start-servers** 操作に渡します。

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



#### 注記

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

## 2.4.2. 管理 CLI を使用したサーバーの正常なシャットダウン

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

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

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

- スタンドアロンサーバーを正常にシャットダウンします。

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

- 管理対象ドメインのすべてのサーバーを停止します。

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

- 管理対象ドメインの単一のサーバーを停止します。

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

- サーバークラスのすべてのサーバーを正常に停止します。

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

### 2.4.3. OS シグナルを使用したサーバーの正常なシャットダウン

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



#### 重要

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

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

## 2.5. JBOSS EAP の起動および停止 (RPM インストール)

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

### 2.5.1. JBoss EAP の起動 (RPM インストール)

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

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

- Red Hat Enterprise Linux 6 の場合

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

- Red Hat Enterprise Linux 7 以降の場合:

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

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

#### 管理対象ドメインでの JBoss EAP の起動 (RPM インストール)

- Red Hat Enterprise Linux 6 の場合

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

- Red Hat Enterprise Linux 7 以降の場合:

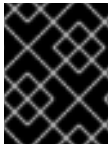
```
$ systemctl start eap7-domain.service
```

これにより、**host.xml** 設定ファイルをデフォルトで使用して JBoss EAP が起動されます。JBoss EAP を別の [管理対象ドメイン設定ファイル](#) で起動するには、[RPM サービス設定ファイル](#) にプロパティを設定します。詳細は「[RPM サービスプロパティの設定](#)」の項を参照してください。

#### RPM サービスプロパティの設定

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

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



#### 重要

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

- サーバー設定ファイルを指定します。  
スタンドアロンサーバーを起動する場合、デフォルトで **standalone.xml** ファイルが使用されます。管理対象ドメインで実行する場合、デフォルトで **host.xml** ファイルが使用されます。他の設定ファイルを使用して JBoss EAP を起動するには、適切な **RPM** 設定ファイル (例: [eap7-standalone.conf](#)) に **WILDFLY\_SERVER\_CONFIG** プロパティを設定します。

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

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

```
WILDFLY_BIND=192.168.0.1
```



## 注記

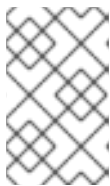
管理インターフェースを特定の IP アドレスにバインドする場合は、次の例のように JBoss EAP 起動設定ファイルに設定を追加します。

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

```

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

```



## 注記

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



## 注記

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

## 2.5.2. JBoss EAP の停止 (RPM インストール)

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

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

- Red Hat Enterprise Linux 6 の場合

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

- Red Hat Enterprise Linux 7 以降の場合:

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

### 管理対象ドメインでの JBoss EAP の停止 (RPM インストール)

- Red Hat Enterprise Linux 6 の場合

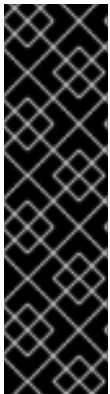
```
$ service eap7-domain stop
```

- Red Hat Enterprise Linux 7 以降の場合:

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

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

## 2.6. POWERSHELL スクリプト (WINDOWS SERVER)



### 重要

複数の PowerShell スクリプトはテクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

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

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

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

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

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



### 注記

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

## 第3章 JBOSS EAP の管理

JBoss EAP は簡単な設定を使用し、スタンドアロンサーバーまたは管理対象ドメインごとに1つの設定ファイルを使用します。スタンドアロンサーバーのデフォルト設定は

**EAP\_HOME/standalone/configuration/standalone.xml** ファイルに保存され、管理対象ドメインのデフォルト設定は **EAP\_HOME/domain/configuration/domain.xml** ファイルに保存されます。また、ホストコントローラーのデフォルト設定は **EAP\_HOME/domain/configuration/host.xml** ファイルに保存されます。

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

### 3.1. サブシステム、エクステンション、およびプロファイル

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

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

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

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

#### 管理コンソールまたは管理 CLI の使用

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

### 3.2. 管理ユーザー

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

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

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

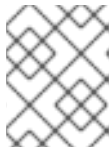
本ガイドでは、**add-user** スクリプトを使用した JBoss EAP の簡単なユーザー管理を取り上げます。このスクリプトは既定の認証のプロパティファイルに新しいユーザーを追加するためのユーティリティです。

LDAP やロールベースアクセス制御 (RBAC) などの高度な認証および承認のオプションについては、JBoss EAP 『[セキュリティアーキテクチャー](#)』の「[コア管理認証](#)」を参照してください。

### 3.2.1. 管理ユーザーの追加

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

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



#### 注記

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

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



#### 注記

ユーザー名には、以下の文字のみを使用できます。文字の数と順番は自由です。

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

デフォルトでは、脆弱なパスワードは許可されますが、警告が表示されます。

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

4. ユーザーが属するグループのコンマ区切りリストを入力します。ユーザーがグループに属さないようにする場合は **ENTER** を押して空白のままにします。
5. 情報を確認し、正しければ **yes** を入力します。
6. このユーザーがリモート JBoss EAP サーバーインスタンスを表すかどうかを決定します。基本的な管理ユーザーの場合は **no** を入力します。  
**ManagementRealm** への追加が必要になることがあるユーザータイプの1つが、JBoss EAP の別のインスタンスを表すユーザーで、メンバーとしてクラスターに参加することを承認する必要があります。この場合は、プロンプトで **yes** を選択すると、異なる設定ファイルに追加する必要がある、ユーザーのパスワードを表すハッシュ化された秘密の値が提供されます。

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

### 3.2.2. Add-User ユーティリティを非対話的に実行

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



### 警告

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

複数のグループに属するユーザーの作成

以下のコマンドは、**guest** および **mgmtgroup** グループの管理ユーザー **mgmtuser1** を追加します。

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

代替プロパティファイルの指定

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

ユーザー情報は以下のプロパティファイルに保存されます。

- **EAP\_HOME/standalone/configuration/mgmt-users.properties**
- **EAP\_HOME/domain/configuration/mgmt-users.properties**

グループ情報は以下のプロパティファイルに保存されます。

- **EAP\_HOME/standalone/configuration/mgmt-groups.properties**
- **EAP\_HOME/domain/configuration/mgmt-groups.properties**

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

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

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

使用できる **add-user** の引数の完全リストとそれら引数の目的については、**--help** 引数を使用するか、「[Add-User ユーティリティー引数](#)」の項を参照してください。

### 3.2.3. Add-User ユーティリティーのパスワード制限の設定

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

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



**EAP\_HOME/bin/add-user.properties** ファイルで設定できる追加のパスワード要件は次のとおりです。

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

### 3.2.4. 管理ユーザーの更新

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

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

```
What type of user do you wish to add?
```

- ```
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
```

```
(a): a
```

```
Enter the details of the new user to add.
```

```
Using realm 'ManagementRealm' as discovered from the existing property files.
```

```
Username : test-user
```

```
User 'test-user' already exists and is enabled, would you like to...
```

- ```
a) Update the existing user password and roles
b) Disable the existing user
c) Type a new username
```

```
(a):
```

すでに存在するユーザー名を入力すると、複数のオプションが出力されます。

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



#### 警告

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

## 3.3. JBOSS EAP サーバー設定の最適化

JBoss EAP サーバーをインストールし、管理ユーザーを作成したら、サーバー設定を最適化することが推奨されます。

『パフォーマンスチューニングガイド』で、本番環境にアプリケーションをデプロイするときに一般的な問題が発生しないようサーバー設定を最適化する方法を必ず確認してください。通常の最適化には、[ulimit の設定](#)、[ガベージコレクションの有効化](#)、[Java ヒープダンプの作成](#)、[スレッドプールサイズの調整](#)などが含まれます。

また、製品のリリースに既存のパッチを適用するとよいでしょう。EAP の各パッチには、多くのバグ修正が含まれています。詳細は、JBoss EAP 『[パッチおよびアップグレードガイド](#)』の「JBoss EAP のパッチ適用」を参照してください。

## 3.4. 管理インターフェース

### 3.4.1. 管理 CLI

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

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

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

コマンドと操作、構文、およびバッチモードでの実行を含む、管理 CLI の使用に関する詳細は、JBoss EAP 『[管理 CLI ガイド](#)』を参照してください。

管理 CLI の起動

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



#### 注記

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

稼働中のサーバーへの接続

```
connect
```

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

ヘルプの表示

以下のコマンドを実行してヘルプを表示します。

```
help
```

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

```
deploy --help
```

管理 CLI の終了

```
quit
```

システム設定の表示

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

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

管理対象ドメインで実行している場合、コマンドの前に **/profile=PROFILE\_NAME** を付けて更新するプロファイルを指定する必要があります。

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

システム設定の更新

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

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

サーバーの起動

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

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

### 3.4.2. 管理コンソール

管理コンソールは、JBoss EAP の web ベースの管理ツールです。

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

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

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

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

#### Home (ホーム)

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

#### Deployments (デプロイメント)

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

### Configuration (設定)

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

### Runtime (ランタイム)

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

### Patching (パッチ)

JBoss EAP インスタンスにパッチを適用します。

### Access Control (アクセス制御)

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

## 3.4.2.1. 管理コンソールの属性の更新

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

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



### 注記

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

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

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

## 3.4.2.2. 管理コンソールの有効化/無効化

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

たとえば、有効にする場合は以下を設定します。

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

無効にする場合は以下を設定します。

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

### 3.4.2.3. 管理コンソールの言語の変更

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

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

管理コンソールの言語の変更方法

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

## 3.5. 管理 API

### 3.5.1. HTTP API

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

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

リソースの読み取り

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

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

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

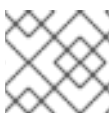
Value	説明
attribute	<b>read-attribute</b> 操作を実行します。
operation-description	<b>read-operation-description</b> 操作を実行します。
operation-names	<b>read-operation-names</b> 操作を実行します。
resource	<b>read-resource</b> 操作を実行します。
resource-description	<b>read-resource-description</b> 操作を実行します。
snapshots	<b>list-snapshots</b> 操作を実行します。

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

#### 例: リソースに対するすべての属性および値の読み取り

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

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



#### 注記

デフォルトの操作は **read-resource** です。

#### 例: リソースに対する属性の値の読み取り

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

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

#### リソースの更新

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

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

#### 例: リソースに対する属性の値の更新

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

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

### 例: サーバーに対する操作の実行

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

これは、サーバーをリロードします。

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

### 3.5.2. ネイティブ API

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

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



#### 注記

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

### 例: ネイティブ API を使用したリソースの読み取り

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

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

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

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

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

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

## 3.6. 設定データ

### 3.6.1. スタンドアロンサーバー設定ファイル

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

表3.1 スタンドアロン設定ファイル

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

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

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

### 3.6.2. 管理対象ドメイン設定ファイル

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

表3.2 管理対象ドメイン設定ファイル

設定ファイル	目的
<b>domain.xml</b>	これは、管理対象ドメインの主要設定ファイルです。ドメインマスターのみがこのファイルを読み取ります。このファイルには、すべてのプロファイル (default、ha、full、full-ha、および load-balancer) の設定が含まれています。



設定ファイル	目的
<b>host.xml</b>	このファイルには、管理対象ドメインの物理ホスト固有の設定情報が含まれています (ネットワークインターフェース、ソケットバインディング、ホスト名、その他のホスト固有の詳細など)。 <b>host.xml</b> ファイルには、 <b>host-master.xml</b> および <b>host-slave.xml</b> (詳細は下記参照) の両方の機能がすべて含まれています。
<b>host-master.xml</b>	このファイルには、サーバーをマスタードメインコントローラーとして実行するために必要な設定情報のみが含まれています。
<b>host-slave.xml</b>	このファイルには、サーバーを管理対象ドメインのホストコントローラーとして実行するために必要な設定情報のみが含まれています。

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

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

### 3.6.3. 設定データのバックアップ

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

- **EAP\_HOME/standalone/configuration/**
  - ディレクトリー全体をバックアップして、スタンドアロンサーバーのユーザーデータ、サーバー設定、およびロギング設定を保存します。
- **EAP\_HOME/domain/configuration/**
  - ディレクトリー全体をバックアップして、管理対象ドメインのユーザーおよびプロフィールデータ、ドメインおよびホスト設定、およびロギング設定を保存します。
- **EAP\_HOME/modules/**
  - カスタムモジュールをバックアップします。
- **EAP\_HOME/welcome-content/**
  - カスタムのウェルカムコンテンツをバックアップします。
- **EAP\_HOME/bin/**
  - カスタムスクリプトまたは起動設定ファイルをバックアップします。

### 3.6.4. 設定ファイルのスナップショット

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

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

#### スナップショットの作成

管理 CLI を使用して、現在の設定のスナップショットを作成します。

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

#### スナップショットのリスト

管理 CLI を使用して、作成したすべてのスナップショットをリストします。

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

#### スナップショットの削除

管理 CLI を使用して、スナップショットを削除します。

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

#### スナップショットを用いたサーバーの起動

スナップショットまたは自動保存された設定を使用してサーバーを起動できます。

1. **EAP\_HOME/standalone/configuration/standalone\_xml\_history** ディレクトリーへ移動し、ロードするスナップショットまたは保存された設定ファイルを確認します。
2. サーバーを起動し、選択した設定ファイルを示します。設定ディレクトリー **EAP\_HOME/standalone/configuration/** からの相対パスを渡します。

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



#### 注記

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

### 3.6.5. 設定変更の確認

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



## 重要

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

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

管理 CLI からの設定変更の追跡および表示

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

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



## 注記

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

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

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

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



## 注記

管理対象ドメインでは、以下のコマンドを使用してホストの設定変更をリストできます。

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

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

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

このコマンドは、各設定変更とその変更日、変更元、結果、および操作の詳細を一覧で表示します。たとえば、以下の **list-changes** コマンドの出力は、変更日が新しい順に設定変更を表示しています。

```
{
  "outcome" => "success",
  "result" => [
    {
      "operation-date" => "2016-02-12T18:37:00.354Z",
      "access-mechanism" => "NATIVE",
      "remote-address" => "127.0.0.1/127.0.0.1",
    }
  ]
}
```

```

"outcome" => "success",
"operations" => [{
  "address" => [],
  "operation" => "reload",
  "operation-headers" => {
    "caller-type" => "user",
    "access-mechanism" => "NATIVE"
  }
}]
},
{
  "operation-date" => "2016-02-12T18:34:16.859Z",
  "access-mechanism" => "NATIVE",
  "remote-address" => "127.0.0.1/127.0.0.1",
  "outcome" => "success",
  "operations" => [{
    "address" => [
      ("subsystem" => "datasources"),
      ("data-source" => "ExampleDS")
    ],
    "operation" => "write-attribute",
    "name" => "enabled",
    "value" => false,
    "operation-headers" => {
      "caller-type" => "user",
      "access-mechanism" => "NATIVE"
    }
  ]
}],
{
  "operation-date" => "2016-02-12T18:24:11.670Z",
  "access-mechanism" => "HTTP",
  "remote-address" => "127.0.0.1/127.0.0.1",
  "outcome" => "success",
  "operations" => [{
    "operation" => "remove",
    "address" => [
      ("subsystem" => "messaging-activemq"),
      ("server" => "default"),
      ("jms-queue" => "ExpiryQueue")
    ],
    "operation-headers" => {"access-mechanism" => "HTTP"}
  ]
}
]
}
}

```

この例は、設定に影響した以下3つの操作の詳細を表しています。

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

管理コンソールからの設定変更の追跡および表示

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

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

### 3.6.6. プロパティーの置き換え

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

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

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

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

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

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

ネストされた式

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

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

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

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

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

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

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

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

#### 記述子ベースのプロパティ置換

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

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

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

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

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

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

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

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

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

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

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

### 3.6.7. Git を使用した設定データの管理

JBoss EAP 7.2 より、Git を使用してサーバー設定データ、プロパティファイル、およびデプロイメ

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

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

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

表3.3 Git 設定管理のサーバー起動引数

引数	説明
<code>--git-repo</code>	サーバー設定データの管理および永続化に使用される Git リポジトリの場所。これは、ローカルで保存する場合は <b>local</b> を指定し、リモートの場合はリモートリポジトリへの URL を指定します。
<code>--git-branch</code>	使用する Git リポジトリのブランチまたはタグ名。ブランチまたはタグ名が存在しないと作成されないため、この引数には既存のブランチまたはタグ名を指定する必要があります。タグ名を使用する場合、リポジトリを detached HEAD 状態にし、今後のコミットがブランチにアタッチされないようにします。タグ名は読み取り専用で、通常複数のノード全体で設定をレプリケートする必要があるときに使用されます。
<code>--git-auth</code>	Elytron 設定ファイルへの URL には、リモート Git リポジトリへの接続時に使用される認証情報が含まれています。この引数は、Git リポジトリに認証が必要な場合に必要となります。Git は SSH 認証をサポートしますが、Elytron はサポートしないため、プライベートキーを使用してパスワードのないデフォルトの SSH 認証のみがサポートされます。この引数は <b>local</b> リポジトリとは使用されません。

#### ローカル Git リポジトリの使用

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

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

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

**local** Git リポジトリを使用する引数を使用してサーバーを起動した場合、JBoss EAP は `jboss.server.base.dir` ディレクトリがすでに Git に対して設定されているかどうかを確認します。設定されていない場合、JBoss EAP は既存の設定コンテンツを使用して `jboss.server.base.dir` ディレクトリに Git リポジトリを作成し、初期化します。JBoss EAP は `--git-branch` 引数によって渡された

ブランチ名をチェックアウトします。引数が渡されていない場合は **master** ブランチをチェックアウトします。初期化後、スタンドアロンサーバーコンテンツのベースディレクトリーに **.git/** ディレクトリーと **.gitignore** ファイルがあることを確認できるはずです。

#### リモート Git リポジトリーの使用

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <authentication-rules>
      <rule use-configuration="test-login">
        </rule>
    </authentication-rules>
    <authentication-configurations>
      <configuration name="test-login">
        <sasl-mechanism-selector selector="BASIC" />
        <set-user-name name="eap-user" />
        <credentials>
          <clear-password password="my_api_key" />
        </credentials>
        <set-mechanism-realm name="testRealm" />
      </configuration>
    </authentication-configurations>
  </authentication-client>
</configuration>
```



#### 注記

Git は SSH 認証をサポートしますが、Elytron はサポートしないため、プライベートキーを使用してパスワードのないデフォルトの SSH 認証のみがサポートされます。

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

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

リモート Git リポジトリーを使用する引数を使用してサーバーを起動した場合、JBoss EAP は **jboss.server.base.dir** ディレクトリーがすでに Git に対して設定されているかどうかを確認します。設定されていない場合、JBoss EAP は **jboss.server.base.dir** ディレクトリーにある既存の設定ファイル



を削除し、代わりにリモート Git 設定データを置きます。JBoss EAP は **--git-branch** 引数によって渡されたブランチ名をチェックアウトします。引数が渡されていない場合は **master** ブランチをチェックアウトします。この処理の完了後、スタンドアロンサーバーコンテンツのベースディレクトリーに **.git/** ディレクトリーと **.gitignore** ファイルがあることを確認できるはずですが、



### 警告

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

### Git 使用時のリモート設定データの公開

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

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

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

### Git でのスナップショットの使用

Git のコミット履歴を使用して設定の変更を追跡する他に、スナップショットを作成して特定時の設定を保持することもできます。スナップショットを一覧表示し、削除することができます。

### Git 使用時におけるスナップショットの作成

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

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

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

### Git 使用時におけるスナップショットの一覧表示

**list-snapshots** 操作を使用すると、スナップショットタグをすべて一覧表示できます。

以下の管理 CLI 操作は、スナップショットタグを一覧表示します。

```
:list-snapshots
{
  "outcome" => "success",
  "result" => {
```

```

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

```

Git 使用時におけるスナップショットの削除

**delete-snapshot** 操作にタグ名を渡すと特定のスナップショットを削除できます。

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

```

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

```

### 3.7. ファイルシステムパス

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

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

例: サーバーログディレクトリーの相対パスの例

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

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

表3.4 標準的なパス

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

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

標準パスの上書き または [カスタムパスの追加](#) を行うことができます。

### 3.7.1. ファイルシステムパスの表示

以下の管理 CLI コマンドを使用して、ファイルシステムパスの一覧を表示します。

```
ls /path
```



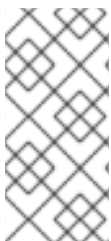
#### 注記

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

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

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

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



### 注記

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

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

## 3.7.2. 標準パスの上書き

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

- サーバーの起動時にコマンドライン引数を渡します。例を以下に示します。

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

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

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

管理対象ドメインの標準パスの上書き

この例の目的は、**/opt/jboss\_eap/domain\_data** ディレクトリーにドメインファイルを格納し、各トップレベルディレクトリーにカスタム名を付けることです。デフォルトのディレクトリーグルーピングである **by-server** が使用されます。

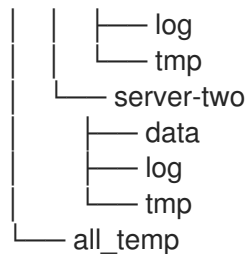
- ログファイルは **all\_logs** サブディレクトリーに格納します。
- データファイルは **all\_data** サブディレクトリーに格納します。
- 一時ファイルは **all\_temp** サブディレクトリーに格納します。
- サーバーのファイルは **all\_servers** サブディレクトリーに格納します。

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

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

この結果、パス構造は次のようになります。

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



### 3.7.3. カスタムパスの追加

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

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

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

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

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

```

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

```

## 3.8. ディレクトリーのグループ化

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

### サーバーを基にしたディレクトリーのグループ化

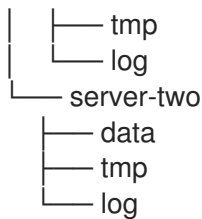
デフォルトでは、ディレクトリーはサーバーを基にしてグループ化されます。管理作業がサーバー中心である場合はこの設定が推奨されます。たとえば、サーバーインスタンスごとにバックアップやログファイルの処理を設定することができます。

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

```

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

```



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

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

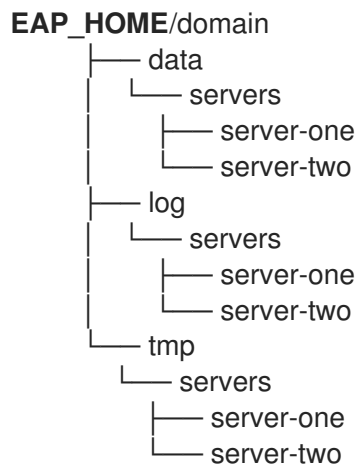
このコマンドにより、ホストコントローラーの **host.xml** 設定ファイルが更新されます。

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

### タイプを基にしたディレクトリーのグループ化

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

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



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

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

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

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

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

### 3.9. システムプロパティ

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

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

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

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

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

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

#### 起動スクリプトにシステムプロパティを渡す

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

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

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

#### 管理 CLI を使用したシステムプロパティの設定

管理 CLI で以下の構文を使用すると、システムプロパティを設定できます。

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

例を以下に示します。

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

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

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

## 管理コンソールを使用したシステムプロパティの設定

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

<ph x="1"/>

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

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



### 注記

RPM インストールの場合、[RPM サービス設定ファイル](#) で **JAVA\_OPTS** を編集してシステムプロパティを設定することが推奨されます。「[RPM サービスプロパティの設定](#)」を参照してください。

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

- **standalone.conf** では、**JAVA\_OPTS** システムプロパティ定義をファイルの最後に追加します。例を以下に示します。

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

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

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

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



### 3.10. 管理監査ロギング

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

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



#### 注記

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

#### スタンドアロンサーバーの監査ロギング

監査ロギングはデフォルトでは無効になっていますが、デフォルトの監査ロギング設定はファイルに書き込みします。

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

この設定は、以下の管理 CLI コマンドを使用して読み取ることができます。

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

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

#### 管理対象ドメインの監査ロギング

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

```
<audit-log>
  <formatters>
    <json-formatter name="json-formatter"/>
  </formatters>
  <handlers>
    <file-handler name="host-file" formatter="json-formatter" relative-to="jboss.domain.data.dir" path="audit-log.log"/>
    <file-handler name="server-file" formatter="json-formatter" relative-to="jboss.server.data.dir" path="audit-log.log"/>
  </handlers>
</audit-log>
```

```

</handlers>
<logger log-boot="true" log-read-only="false" enabled="false">
  <handlers>
    <handler name="host-file"/>
  </handlers>
</logger>
<server-logger log-boot="true" log-read-only="false" enabled="false">
  <handlers>
    <handler name="server-file"/>
  </handlers>
</server-logger>
</audit-log>

```

この設定は、以下の管理 CLI コマンドを使用して読み取ることができます。

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

管理対象ドメインの監査ロギングを有効にする場合は、「[監査ロギングの有効化](#)」を参照してください。

### 3.10.1. 管理監査ロギングの有効化

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

次の手順では、**NATIVE** および **HTTP** 監査ロギングを有効にします。JMX 監査ロギングを設定する場合は「[JMX 管理監査ロギングの有効化](#)」を参照してください。

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

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

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

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

管理対象ドメイン監査ロギングの有効化

管理対象ドメインのデフォルトの監査ログ設定は、各ホストおよび各サーバーに対して監査ログを書き込むよう事前設定されています。

各ホストの監査ロギングは以下のコマンドを使用して有効にできます。

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

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

各サーバーの監査ロギングは以下のコマンドを使用して有効にできます。

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

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

### 3.10.2. JMX 管理監査ロギングの有効化

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

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

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

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

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

管理対象ドメインの **JMX** 監査ロギングの有効化  
管理対象ドメインの各ホストおよびプロファイルに **JMX** 監査ロギングを有効にすることができます。

ホストの **JMX** 監査ロギングの有効化

1. ホストの **jmx** サブシステムで監査ロギングを有効にします。

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

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

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

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

プロファイルの **JMX** 監査ロギングの有効化

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

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

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

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

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

### 3.10.3. syslog サーバーへの管理監査ロギングの送信

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

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

1. syslog ハンドラーを追加します。

syslog サーバーのホストとポートを指定して syslog ハンドラーを作成します。管理対象ドメインでは、**/core-service** コマンドの前に **/host=HOST\_NAME** を追加する必要があります。

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

#### 注記

渡すパラメーターは指定されたプロトコルによって異なります。

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

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

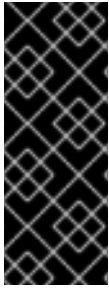
2. syslog ハンドラーへの参照を追加します。

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

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

3. 監査ロギングを有効にします。

「[管理監査ロギングの有効化](#)」を参照して監査ロギングを有効にします。



## 重要

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

Red Hat Enterprise Linux の **rsyslog** 設定の詳細

は、<https://access.redhat.com/documentation/ja/red-hat-enterprise-linux/> にて『システム管理者のガイド』の「[Rsyslog の基本設定](#)」の項を参照してください。

### 3.10.4. 監査ログエントリーの読み取り

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



## 注記

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

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

表3.5 管理監査ログフィールド

フィールド名	説明
access	以下のいずれかの値を使用できます。 <ul style="list-style-type: none"> <li>● <b>NATIVE</b> - 操作がネイティブ管理インターフェースから実行されます。</li> <li>● <b>HTTP</b> - 操作がドメイン HTTP インターフェースから実行されます。</li> <li>● <b>JMX</b> - 操作が <b>jmx</b> サブシステムから実行されます。</li> </ul>
booting	起動プロセス中に操作が実行された場合は、値が <b>true</b> になります。サーバーの起動後に操作が実行された場合は <b>false</b> になります。
domainUUID	ドメインコントローラーからサーバー、スレーブホストコントローラー、およびスレーブホストコントローラーサーバーへ伝播されるすべての操作をリンクする ID。
ops	実行される操作。JSON ヘシリアライズ化された操作のリストになります。起動時は、XML の解析で生じたすべての操作がリストされます。起動後は、通常1つのエントリーがリストされます。
r/o	操作によって管理モデルが変更されない場合は、値が <b>true</b> になります。変更される場合は <b>false</b> になります。
remote-address	この操作を実行するクライアントのアドレス。

フィールド名	説明
success	操作に成功した場合は値が <b>true</b> になります。ロールバックされた場合は <b>false</b> になります。
type	管理操作であることを示す <b>core</b> 、または <b>jmx</b> サブシステムからであることを示す <b>jmx</b> を値として指定できます。
user	認証されたユーザーのユーザー名。稼働しているサーバーと同じマシンの管理 CLI を使用して操作を行った場合は、特別な <b>\$local</b> ユーザーが使用されます。
version	JBoss EAP インスタンスのバージョン番号。

### 3.11. サーバライフサイクルイベントの通知

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

JBoss EAP のサーバランタイム設定の状態には、**STARTING**、**RUNNING**、**RELOAD\_REQUIRED**、**RESTART\_REQUIRED**、**STOPPING**、および **STOPPED** があります。

JBoss EAP のサーバ稼働の状態には、**STARTING**、**NORMAL**、**ADMIN\_ONLY**、**PRE\_SUSPEND**、**SUSPENDING**、**SUSPENDED**、**STOPPING**、および **STOPPED** があります。

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

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

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

##### 例: リスナークラス

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

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

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

public class SimpleListener implements ProcessStateListener {
```

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

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

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

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

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



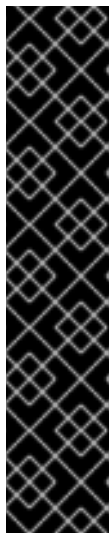
## 注記

リスナーの実装時には以下に気をつけてください。

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

2. クラスおよびパッケージを JAR にコンパイルします。  
コンパイルするには、**org.wildfly.core:wildfly-core-management-client** Maven モジュールに依存する必要があります。
3. JAR を JBoss EAP モジュールとして追加します。  
以下の管理 CLI コマンドを使用し、モジュール名と JAR へのパスを提供します。

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



## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

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

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



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

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

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

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

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

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

#### 例: リスナークラス

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

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

import org.jboss.logging.Logger;

public class StateNotificationListener implements NotificationListener {

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

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

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

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

    protected final void init(Path targetFile) {
```

```

try {
    Files.createDirectories(targetFile);

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

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

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

private void writeNotification(AttributeChangeNotification notification, Path path) {
    try (BufferedWriter in = Files.newBufferedWriter(path, StandardCharsets.UTF_8,
StandardOpenOption.APPEND)) {
        in.write(String.format("%s %s %s %s", notification.getType(),
notification.getSequenceNumber(), notification.getSource().toString(),
notification.getMessage()));
        in.newLine();
        in.flush();
    } catch (IOException ex) {
        Logger.getLogger(StateNotificationListener.class).error("Problem handling JMX
Notification", ex);
    }
}
}

```

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

#### 例: 通知リスナーの追加

```

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

```

3. JBoss EAP にパッケージ化およびデプロイします。

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

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

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

## 第4章 ネットワークおよびポート設定

### 4.1. インターフェース

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

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

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

#### 4.1.1. デフォルトインターフェース設定

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

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

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

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



#### 重要

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

#### 4.1.2. インターフェースの設定

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

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

#### NIC 値があるインターフェースの追加

NIC 値が **eth0** であるインターフェースを新たに追加します。

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

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

#### 複数の条件値があるインターフェースの追加

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

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

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

#### インターフェース属性の更新

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

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

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

#### 管理対象ドメインでインターフェースをサーバーに追加

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

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

```

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

```

## 4.2. ソケットバインディング

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

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

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

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

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

### インバウンドソケットバインディング

**socket-binding** 要素は、JBoss EAP サーバーのインバウンドソケットバインディングを設定するために使用されます。デフォルトの JBoss EAP 設定には、HTTP や HTTPS トラフィック用などの、事前設定された **socket-binding** 要素が複数提供されます。JBoss EAP 『[Configuring Messaging](#)』の「**Broadcast Groups**」には他の例も記載されています。

この要素の属性については、[インバウンドソケットバインディングの属性](#)の表を参照してください。

### リモートアウトバウンドソケットバインディング

**remote-destination-outbound-socket-binding** 要素は、JBoss EAP サーバーのリモートとなる宛先のアウトバウンドソケットバインディングを設定するために使用されます。デフォルトの JBoss EAP 設定には、メールサーバーに使用できるリモート宛先のソケットバインディングの例が含まれています。JBoss EAP 『[Configuring Messaging](#)』の「**Using the Integrated Artemis Resource Adapter for Remote Connections**」には、他の例も記載されています。

この要素の属性については、[リモートアウトバウンドソケットバインディングの属性](#)の表を参照してください。

### ローカルアウトバウンドソケットバインディング

**local-destination-outbound-socket-binding** 要素は、JBoss EAP サーバーのローカルとなる宛先のアウトバウンドソケットバインディングを設定するために使用されます。通常、このソケットバインディングはあまり使用されません。

この要素の属性については、[ローカルアウトバウンドソケットバインディングの属性](#)の表を参照してください。

#### 4.2.1. 管理ポート

JBoss EAP 7では、管理ポートが集約されました。JBoss EAP 7は、管理 CLI によって使用されるネイティブ管理と、Web ベース管理コンソールによって使用される HTTP 管理の両方に **9990** ポートを使用します。JBoss EAP 6 でネイティブ管理ポートとして使用されていた **9999** ポートは使用されなくなりましたが、必要な場合は有効にできます。

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

## 4.2.2. デフォルトのソケットバインディング

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

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



### 重要

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

### スタンドアロンサーバー

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

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

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

### 管理対象ドメイン

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

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

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

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

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





## 注記

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

### 4.2.3. ソケットバインディングの設定

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

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

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

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

2. ソケットバインディングを追加します。

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

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

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

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

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

### 4.2.4. ポートオフセット

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

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

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

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

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

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

### 4.3. IPV6 アドレス

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

#### IPv6 アドレスの JVM スタックの設定

IPv6 アドレスを優先するように、起動設定を更新します。

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

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

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

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

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

```
# Specify options to pass to the Java VM.
#
if [ "x$JAVA_OPTS" = "x" ]; then
  JAVA_OPTS="-Xms1303m -Xmx1303m -Djava.net.preferIPv4Stack=false"
  JAVA_OPTS="$JAVA_OPTS -
  Djboss.modules.system.pkgs=$JBOSS_MODULES_SYSTEM_PKGS -Djava.awt.headless=true"
  JAVA_OPTS="$JAVA_OPTS -Djava.net.preferIPv6Addresses=true"
else
```

#### IPv6 アドレスのインターフェース宣言の更新

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

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

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

```
<interfaces>
```

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

## 第5章 JBOSS EAP のセキュリティー

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

- 一般的なセキュリティー概念と JBoss EAP 固有のセキュリティー概念については、『[セキュリティーアーキテクチャー](#)』を参照してください。
- JBoss EAP 自体のセキュア化に関する情報は、『[How to Configure Server Security](#)』を参照してください。
- JBoss EAP にデプロイされたアプリケーションのセキュリティーに関する詳細は、『[How to Configure Identity Management](#)』を参照してください。
- Kerberos を使用した JBoss EAP のシングルサインオンの設定に関する情報は、『[Kerberos による SSO のセットアップ方法](#)』を参照してください。
- SAML v2 を使用して JBoss EAP のシングルサインオンを設定するための情報は『[How To Set Up SSO with SAML v2](#)』を参照してください。

## 第6章 JBOSS EAP クラスローディング

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

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

### 6.1. モジュール

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

#### 静的モジュール

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

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

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

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

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

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

モジュールごとに1つのディレクトリーを使用して、カスタムモジュールが **EAP\_HOME/modules** ディレクトリーにインストールされるようにする必要があります。こうすると、同梱されたバージョン

ではなく、**system** ディレクトリーに存在するカスタムバージョンのモジュールがロードされるようになります。これにより、ユーザー提供のモジュールがシステムモジュールよりも優先されます。

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

### 注記

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

### 例: module.xml ファイルの絶対パス

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

### 動的モジュール

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

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

### 事前定義されたモジュール

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

Java 9 以上で利用できるプラットフォームモジュールの一覧は、該当する JDK のドキュメントを参照してください。

Java 8 向けに提供されるプラットフォームモジュールの一覧は、「[Java 8 向けに提供されるプラットフォームモジュール](#)」を参照してください。

## 6.2. モジュールの依存性

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



## 注記

モジュールとモジュール形式のクラスロードシステムに関する完全な詳細については、「[モジュール](#)」を参照してください。

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

依存関係には明示的と暗黙的の 2 つのタイプがあります。

### 明示的な依存関係

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

### 暗黙的な依存関係

暗黙的な依存関係は、デプロイメントで特定の状態やメタデータが見つかったときに自動的に追加されます。JBoss EAP に同梱される Java EE API は、デプロイメントで暗黙的な依存関係が検出されたときに追加されるモジュールの例になります。

**jboss-deployment-structure.xml** デプロイメント記述子ファイルを使用して、特定の暗黙的な依存関係を除外するようデプロイメントを設定することも可能です。これは、JBoss EAP が暗黙的な依存関係として追加しようとする特定バージョンのライブラリーをアプリケーションがバンドルする場合に役に立つことがあります。

### オプションの依存関係

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

### 依存関係のエクスポート

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

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

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

### グローバルモジュール

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

詳細は「[グローバルモジュールの定義](#)」の項を参照してください。

## 6.3. カスタムモジュールの作成

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

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

### カスタムモジュールの手動作成

カスタムモジュールを手動で作成するには、以下の手順に従います。

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

#### 例: MySQL JDBC ドライバーディレクトリー構造の作成

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

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

#### 例: MySQL JDBC ドライバー JAR のコピー

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

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

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

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

### 管理 CLI を使用したカスタムモジュールの作成

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



## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

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

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

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

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

### 例: MySQL モジュールの作成

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

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

### モジュールを依存関係として追加

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

- デプロイメント記述子を使用してアプリケーション固有の依存関係を追加するには、『[開発ガイド](#)』の「デプロイメントへの明示的なモジュール依存関係の追加」を参照してください。
- モジュールを依存関係としてすべてのアプリケーションに追加する手順については「[グローバルモジュールの定義](#)」の項を参照してください。

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

1. JAR ファイルをコアモジュールとして追加します。

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

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

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

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

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

## 6.4. カスタムモジュールの削除

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

### 手作業によるカスタムモジュールの削除

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

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

### 管理 CLI を使用したカスタムモジュールの削除

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

#### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

- JBoss EAP サーバーを起動します。
- 管理 CLI を起動します。

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

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

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

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

## 例: MySQL モジュールの削除

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

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

## 6.5. グローバルモジュールの定義

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



### 注記

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

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

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

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

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

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

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

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

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

または、複数のモジュールを追加する場合は以下を使用します。

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

新しいモジュールを既存のリストに追加するには、以下を行います。

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

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

## 6.6. サブデプロイメント分離の設定

エンタープライズアーカイブ (EAR) の各サブデプロイメントは独自のクラスローダーを持つ動的モジュールです。サブデプロイメントは、**EAR/lib** のクラスへのアクセスを提供する親モジュールの暗黙的な依存関係を常に持ちます。デフォルトでは、サブデプロイメントはその EAR 内にある他のサブデプロイメントのリソースにアクセスできます。

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

### すべてのデプロイメントを対象とするサブデプロイメントモジュール分離の有効化

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

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

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

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

## 6.7. 外部 JBOSS EAP モジュールディレクトリーの定義

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



### 注記

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

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

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

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



## 注記

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

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

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

## 6.8. 動的モジュールの命名規則

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

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

```
deployment.DEPLOYMENT_NAME
```

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

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

```
deployment.EAR_NAME.SUBDEPLOYMENT_NAME
```

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

## 第7章 アプリケーションのデプロイ

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

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

- サーバー起動時

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

- 以下の管理 CLI コマンドでサーバー設定に追加

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

### 7.1. 管理 CLI を使用したアプリケーションのデプロイ

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

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

アプリケーションのデプロイ

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

```
deployment deploy-file /path/to/test-application.war
```

正常にデプロイされると、管理 CLI には何も出力されませんが、サーバーログにデプロイメントメッセージが記録されます。

```
WFLYSRV0027: Starting deployment of "test-application.war" (runtime-name: "test-application.war")
WFLYUT0021: Registered web context: /test-application
WFLYSRV0010: Deployed "test-application.war" (runtime-name : "test-application.war")
```

同様に、以下の **deployment** コマンドを使用できます。

- **deployment deploy-cli-archive** を使用してコンテンツを **.cli** アーカイブファイルからデプロイします。CLI デプロイメントアーカイブは、**.cli** 拡張子を持つ **JAR** ファイルです。デプロイする必要があるアプリケーションアーカイブと、コマンドおよび操作が含まれる **deploy.scr** および **undeploy.scr** CLI スクリプトファイルが含まれます。**deploy.scr** スクリプトファイルには、アプリケーションアーカイブをデプロイし、環境を設定するコマンドと操作が含まれます。**undeploy.scr** スクリプトファイルには、アプリケーションアーカイブをアンデプロイし、環境をクリーンアップするコマンドが含まれます。

- **deployment deploy-url** を使用して、URL によって参照されるコンテンツをデプロイします。

**deployment enable** コマンドを使用して、無効になっているアプリケーションを有効にすることもできます。

アプリケーションのアンデプロイ

管理 CLI で **deployment undeploy** コマンドを使用し、デプロイメント名を指定します。これにより、リポジトリからデプロイメントコンテンツが削除されます。アンデプロイする際にデプロイメントコンテンツを維持する場合は、「[Disable an Application](#)」を参照してください。

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

正常にアンデプロイされると、管理 CLI には何も出力されませんが、サーバーログにアンデプロイメントメッセージが記録されます。

```
WFLYUT0022: Unregistered web context: /test-application
WFLYSRV0028: Stopped deployment test-application.war (runtime-name: test-application.war) in
62ms
WFLYSRV0009: Undeployed "test-application.war" (runtime-name: "test-application.war")
```

同様に、**deployment undeploy-cli-archive** を使用して **.cli** アーカイブファイルからコンテンツをアンデプロイできます。ワイルドカード (\*) を使用してすべてのデプロイメントをアンデプロイすることも可能です。

```
deployment undeploy *
```

アプリケーションの無効化

デプロイメントコンテンツをリポジトリから削除せずに、デプロイされたアプリケーションを無効にします。

```
deployment disable test-application.war
```

**deployment disable-all** コマンドを使用するとすべてのデプロイメントを無効化することができます。

```
deployment disable-all
```

アプリケーションの有効化

無効になっているデプロイされたアプリケーションを有効にします。

```
deployment enable test-application.war
```

**deployment enable-all** コマンドを使用するとすべてのデプロイメントを有効化することができます。

```
deployment enable-all
```

デプロイメントの一覧表示

管理 CLI で **deployment info** コマンドを使用して、デプロイメントの情報を表示します。

```
deployment info
```

出力には、ランタイム名、状態、有効であるかどうかなど、各デプロイメントの詳細が表示されます。

NAME	RUNTIME-NAME	PERSISTENT	ENABLED	STATUS
helloworld.war	helloworld.war	true	true	OK
test-application.war	test-application.war	true	true	OK

以下のコマンドは、名前を指定してデプロイメント情報を表示します。

```
deployment info helloworld.war
```

**deployment list** コマンドを使用して、デプロイメントをすべて表示することもできます。

## 7.1.2. 管理 CLI を使用した管理対象ドメインでのアプリケーションのデプロイ

アプリケーションのデプロイ

管理 CLI で **deployment deploy-file** コマンドを使用し、アプリケーションデプロイメントへのパスを指定します。また、アプリケーションをデプロイするサーバーグループを指定する必要もあります。

- すべてのサーバーグループにアプリケーションをデプロイする場合

```
deployment deploy-file /path/to/test-application.war --all-server-groups
```

- 特定のサーバーグループにアプリケーションをデプロイする場合

```
deployment deploy-file /path/to/test-application.war --server-groups=main-server-group,other-server-group
```

正常にデプロイされると、管理 CLI には何も出力されませんが、サーバーログに各サーバーのデプロイメントメッセージが記録されます。

```
[Server:server-one] WFLYSRV0027: Starting deployment of "test-application.war" (runtime-name: "test-application.war")
[Server:server-one] WFLYUT0021: Registered web context: /test-application
[Server:server-one] WFLYSRV0010: Deployed "test-application.war" (runtime-name : "test-application.war")
```

同様に、以下の **deployment** コマンドを使用できます。

- deployment deploy-cli-archive** コマンドを使用してコンテンツを **.cli** アーカイブファイルからデプロイします。CLI デプロイメントアーカイブは、**.cli** 拡張子を持つ **JAR** ファイルです。デプロイする必要があるアプリケーションアーカイブと、コマンドおよび操作が含まれる **deploy.scr** および **undeploy.scr** CLI スクリプトファイルが含まれます。**deploy.scr** スクリプトファイルには、アプリケーションアーカイブをデプロイし、環境を設定するコマンドと操作が含まれます。**undeploy.scr** スクリプトファイルには、アプリケーションアーカイブをアンデプロイし、環境をクリーンアップするコマンドが含まれます。
- deployment deploy-url** コマンドを使用して、URL によって参照されるコンテンツをデプロイします。

**deployment enable** コマンドを使用して、無効になっているアプリケーションを有効にすることもできます。

アプリケーションのアンデプロイ

管理 CLI で **deployment undeploy** コマンドを使用し、デプロイメント名を指定します。また、アプリケーションをアンデプロイするサーバーグループを指定する必要もあります。特定のサーバーグループからのアンデプロイについては、「[アプリケーションの無効化](#)」を参照してください。



すべてのサーバーグループからアプリケーションをアンデプロイします。

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

正常にアンデプロイされると、管理 CLI には何も出力されませんが、サーバーログに各サーバーのアンデプロイメントメッセージが記録されます。

```
[Server:server-one] WFLYUT0022: Unregistered web context: /test-application
[Server:server-one] WFLYSRV0028: Stopped deployment test-application.war (runtime-name: test-application.war) in 74ms
[Server:server-one] WFLYSRV0009: Undeployed "test-application.war" (runtime-name: "test-application.war")
```

同様に、**deployment undeploy-cli-archive** コマンドを使用して **.cli** アーカイブファイルからコンテンツをアンデプロイできます。ワイルドカード (\*) を使用してすべてのデプロイメントをアンデプロイすることも可能です。

```
deployment undeploy * --all-relevant-server-groups
```

アプリケーションの無効化

特定のサーバーグループからデプロイされたアプリケーションを無効にし、そのデプロイメントを持つ他のサーバーグループのリポジトリにそのコンテンツを保持します。

```
deployment disable test-application.war --server-groups=other-server-group
```

**deployment disable-all** コマンドを使用するとすべてのデプロイメントを無効化することができます。

```
deployment disable-all --server-groups=other-server-group
```

アプリケーションの有効化

無効になっているデプロイされたアプリケーションを有効にします。

```
deployment enable test-application.war
```

**deployment enable-all** コマンドを使用するとすべてのデプロイメントを有効化することができます。

```
deployment enable-all --server-groups=other-server-group
```

デプロイメントの一覧表示

管理 CLI で **deployment info** コマンドを使用して、デプロイメントの情報を表示します。デプロイメント名またはサーバーグループでデプロイメント情報を絞り込むことができます。

以下のコマンドは、名前を指定してデプロイメント情報を表示します。

```
deployment info helloworld.war
```

出力には、デプロイメントと各サーバーグループでの状態が表示されます。

```
NAME          RUNTIME-NAME
helloworld.war helloworld.war
```

```
SERVER-GROUP  STATE
main-server-group enabled
other-server-group added
```

以下のコマンドは、サーバーグループを指定してデプロイメント情報を表示します。

```
deployment info --server-group=other-server-group
```

出力には、デプロイメントと、指定のサーバーグループに対する状態が表示されます。

```
NAME          RUNTIME-NAME  STATE
helloworld.war helloworld.war added
test-application.war test-application.war enabled
```

**deployment list** コマンドを使用して、ドメインのデプロイメントをすべて表示することもできます。

## 7.2. 管理コンソールを使用したアプリケーションのデプロイ

管理コンソールを使用してアプリケーションをデプロイすると、使用が簡単なグラフィカルインターフェースを利用することができます。サーバーまたはサーバーグループにデプロイされたアプリケーションを一目で確認できるほか、必要に応じてアプリケーションを有効または無効にしたり、アプリケーションをコンテンツリポジトリから削除したりすることができます。

### 7.2.1. 管理コンソールを使用したアプリケーションのスタンドアロンサーバーへのデプロイ

JBoss EAP 管理コンソールの **Deployments** タブからデプロイメントを表示および管理できます。

#### アプリケーションのデプロイ

追加 (+) ボタンをクリックします。デプロイメントのアップロード、未管理デプロイメントの追加、または 空のデプロイメントの作成 を選択して、アプリケーションをデプロイできます。デプロイメントはデフォルトで有効になります。

- **新規デプロイメントのアップロード**  
サーバーのコンテンツリポジトリにコピーされ、JBoss EAP によって管理されるアプリケーションをアップロードします。
- **未管理デプロイメントの追加**  
デプロイメントの場所を指定します。このデプロイメントはサーバーのコンテンツリポジトリにはコピーされず、JBoss EAP によって管理されません。
- **空のデプロイメントの作成**  
空の展開形式 (exploded) のデプロイメントを作成します。このデプロイメントの作成後、ファイルをデプロイメントに追加できます。

#### アプリケーションのアンデプロイ

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

#### アプリケーションの無効化

デプロイメントを選択し、無効オプションを選択してアプリケーションを無効にします。これにより、デプロイメントがアンデプロイされますが、コンテンツリポジトリから削除されません。

#### アプリケーションの置換

デプロイメントを選択し、置換オプションを選択します。元のバージョンと同じ名前を持つ新しいバージョンのデプロイメントを選択し、**Finish** をクリックします。これにより、元のバージョンのデプロイメントがアンデプロイおよび削除され、新しいバージョンがデプロイされます。

## 7.2.2. 管理コンソールを使用した管理対象ドメインでのアプリケーションのデプロイ

JBoss EAP 管理コンソールの **Deployments** タブではデプロイメントを表示および管理できます。

- **Content Repository**  
管理されるデプロイメントと管理されないデプロイメントはすべて **Content Repository** セクションで表示されます。ここで、デプロイメントを追加したり、デプロイメントをサーバーグループにデプロイすることができます。
- **Server Groups**  
1つまたは複数のサーバーグループにデプロイされたデプロイメントは **Server Groups** セクションにリストされます。ここで、デプロイメントを直接サーバーグループに追加したり、有効にしたりすることができます。

### アプリケーションの追加

1. **Content Repository** で 追加ボタンをクリックします。
2. デプロイメントのアップロード または 未管理のデプロイメント作成 を選択し、アプリケーションを追加します。
3. プロンプトに従ってアプリケーションをデプロイします。  
デプロイメントを有効にするには、デプロイメントをサーバーグループにデプロイする必要があります。

### アプリケーションのサーバーグループへのデプロイ

1. **Content Repository** でデプロイメントを選択し、**Deploy** を選択します。
2. このデプロイメントをデプロイするサーバーグループを1つ以上選択します。
3. 選択したサーバーグループのデプロイメントを有効にするオプションを任意で選択することもできます。

### アプリケーションのサーバーグループからのアンデプロイ

1. **Server Groups** で適切なサーバーグループを選択します。
2. デプロイメントを選択し、削除 を選択します。

また、**Content Repository** でデプロイメントの **Undeploy** ボタンを選択して、複数のサーバーグループから1度にデプロイメントをアンデプロイすることもできます。

### アプリケーションの削除

1. デプロイメントがサーバーグループにデプロイされている場合は、必ずデプロイメントをアンデプロイします。
2. **Content Repository** でデプロイメントを選択し、削除 を選択します。

これにより、コンテンツリポジトリからデプロイメントが削除されます。

### アプリケーションの無効化

1. **Content Repository** で適切なデプロイメントを選択し、無効化 ボタンをクリックします。

1. **Server Groups** で適切なサーバーグループを選択します。
2. 無効にするデプロイメントを選択し、無効 を選択します。

これにより、デプロイメントがアンデプロイされますが、コンテンツリポジトリから削除されません。

アプリケーションの置換

1. **Content Repository** からデプロイメントを選択し、置換ボタンをクリックします。
2. 元のバージョンと同じ名前を持つ新しいバージョンのデプロイメントを選択し、置換 をクリックします。

これにより、元のバージョンのデプロイメントがアンデプロイおよび削除され、新しいバージョンがデプロイされます。

## 7.3. デプロイメントスキャナーを使用したアプリケーションのデプロイ

デプロイメントスキャナーは、デプロイするアプリケーションのデプロイメントディレクトリーを監視します。デフォルトでは、デプロイメントスキャナーは5秒ごとに

**EAP\_HOME/standalone/deployments/** をスキャンし、変更を確認します。デプロイメントの状態を示し、アンデプロイや再デプロイなどのデプロイメントに対するアクションをトリガーするため、マーカーファイルが使用されます。

本番環境では、アプリケーションのデプロイメントに管理コンソールまたは管理 CLI の使用が推奨されますが、デプロイメントスキャナーは開発者の便宜を図るために提供されます。これにより、ペースの早い開発サイクルに適した方法でアプリケーションを構築およびテストできます。デプロイメントスキャナーは、他のデプロイメント方法と併用しないでください。

デプロイメントスキャナーは JBoss EAP をスタンドアロンサーバーとして実行している場合のみ利用できます。

### 7.3.1. デプロイメントスキャナーを使用したアプリケーションのスタンドアロンサーバーへのデプロイ

デプロイメントスキャナーを設定して XML、zip 形式、および展開形式のコンテンツの自動デプロイメントを有効または無効にすることができます。自動デプロイメントが無効の場合、マーカーファイルを手作業で作成してデプロイメントのアクションをトリガーする必要があります。利用できるマーカーファイルタイプやそれらの目的に関する詳細は、「[デプロイメントスキャナーのマーカーファイル](#)」の項を参照してください。

デフォルトでは、XML および zip 形式のコンテンツの自動デプロイメントは有効になっています。各コンテンツタイプの自動デプロイメントの設定に関する詳細は「[デプロイメントスキャナーの設定](#)」を参照してください。



#### 警告

デプロイメントスキャナーを使用したデプロイメントは開発者の便宜を図るために提供され、本番環境での使用は推奨されません。デプロイメントスキャナーは他のデプロイメント方法と併用しないでください。

アプリケーションのデプロイ  
コンテンツをデプロイメントフォルダーにコピーします。

```
$ cp /path/to/test-application.war EAP_HOME/standalone/deployments/
```

自動デプロイメントが有効の場合、このファイルは自動的に選択され、デプロイされます。さらに、**.deployed** マーカーファイルが作成されます。自動デプロイメントが無効の場合、**.dodeploy** マーカーファイルを手作業で追加し、デプロイメントをトリガーする必要があります。

```
$ touch EAP_HOME/standalone/deployments/test-application.war.dodeploy
```

アプリケーションのアンデプロイ  
**.deployed** マーカーファイルを削除して、アンデプロイメントをトリガーします。

```
$ rm EAP_HOME/standalone/deployments/test-application.war.deployed
```

自動デプロイメントが有効な場合、アンデプロイメントをトリガーする **test-application.war** ファイルを削除することもできます。これは、展開形式のデプロイメントには適用されないことに注意してください。

アプリケーションの再デプロイ  
**.dodeploy** マーカーファイルを作成し、再デプロイを開始します。

```
$ touch EAP_HOME/standalone/deployments/test-application.war.dodeploy
```

### 7.3.2. デプロイメントスキャナーの設定

デプロイメントスキャナーは管理コンソールまたは管理 CLI を使用して設定できます。スキャンの間隔、デプロイメントフォルダーの場所、特定のアプリケーションファイルタイプの自動デプロイメントなど、デプロイメントスキャナーの動作を設定できます。また、デプロイメントスキャナーを完全に無効にすることもできます。

利用できるデプロイメントスキャナー属性の詳細は、「[デプロイメントスキャナーの属性](#)」の項を参照してください。

以下の管理 CLI コマンドを使用してデフォルトのデプロイメントスキャナーを設定します。

デプロイメントスキャナーの無効化

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=scan-enabled,value=false)
```

**default** デプロイメントスキャナーが無効になります。

スキャン間隔の変更

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=scan-interval,value=10000)
```

スキャンの間隔が **5000** ミリ秒 (5 秒) から **10000** ミリ秒 (10 秒) に変更されます。

デプロイメントフォルダーの変更

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=path,value=/path/to/deployments)
```

デプロイメントフォルダーの場所がデフォルトの **EAP\_HOME/standalone/deployments** から **/path/to/deployments** に変更されます。

**relative-to** 属性が指定されていない場合、**path** の値は絶対パスになります。relative-to 属性が指定されている場合は相対パスになります。

展開形式のコンテンツの自動デプロイメントの有効化

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=auto-deploy-exploded,value=true)
```

デフォルトで無効になっている展開形式のコンテンツの自動デプロイメントを有効にします。

zip 形式のコンテンツの自動デプロイメントの無効化

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=auto-deploy-zipped,value=false)
```

デフォルトで有効になっている zip 形式のコンテンツの自動デプロイメントを無効にします。

XML コンテンツの自動デプロイメントの無効化

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=auto-deploy-xml,value=false)
```

デフォルトで有効になっている XML コンテンツの自動デプロイメントを無効にします。

### 7.3.3. カスタムデプロイメントスキャナーの定義

新しいデプロイメントスキャナーを追加するには、管理 CLI を使用するか、管理コンソールの **Configuration** タブから **Deployment Scanners** サブシステムに移動します。デプロイメントを確認するためにスキャンする新しいディレクトリーを定義します。デフォルトのデプロイメントスキャナーは **EAP\_HOME/standalone/deployments** を監視します。既存のデプロイメントスキャナーの設定に関する詳細は「[デプロイメントスキャナーの設定](#)」を参照してください。

以下の管理 CLI コマンドは、**EAP\_HOME/standalone/new\_deployment\_dir** を 5 秒ごとにチェックしてデプロイメントを確認する新しいデプロイメントスキャナーを追加します。

```
/subsystem=deployment-scanner/scanner=new-scanner:add(path=new_deployment_dir,relative-to=jboss.server.base.dir,scan-interval=5000)
```



#### 注記

指定のディレクトリーがすでに存在しないと、このコマンドに失敗し、エラーが発生します。

新しいデプロイメントスキャナーが定義され、デプロイメントを確認するために指定のディレクトリーが監視されます。

## 7.4. MAVEN を使用したアプリケーションのデプロイ

Apache Maven を使用してアプリケーションをデプロイすると、JBoss EAP へのデプロイメントを簡単に既存の開発ワークフローに取り入れることができます。

アプリケーションをアプリケーションサーバーにデプロイおよびアンデプロイする簡単な操作を提供する [WildFly Maven Plugin](#) を使用すると、Maven を使用してアプリケーションを JBoss EAP にデプロイできます。

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

以下の手順では、Maven を使用して JBoss EAP の **helloworld** クイックスタートをスタンドアロンサーバーにデプロイおよびアンデプロイする方法を示します。

JBoss EAP クイックスタートの詳細は、JBoss EAP 『[スタートガイド](#)』の「クイックスタートサンプルの使用」を参照してください。

アプリケーションのデプロイ

Maven **pom.xml** ファイルで WildFly Maven Plugin を初期化します。これは、JBoss EAP クイックスタートの **pom.xml** ファイルで設定されているはずです。

```
<plugin>
  <groupId>org.wildfly.plugins</groupId>
  <artifactId>wildfly-maven-plugin</artifactId>
  <version>${version.wildfly.maven.plugin}</version>
</plugin>
```

**helloworld** クイックスタートディレクトリーで以下の Maven コマンドを実行します。

```
$ mvn clean install wildfly:deploy
```

デプロイする Maven コマンドの実行後、ターミナルウィンドウにはデプロイメントの成功を表す以下の出力が表示されます。

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.981 s
[INFO] Finished at: 2015-12-23T15:06:13-05:00
[INFO] Final Memory: 21M/231M
[INFO] -----
```

アクティブなサーバーインスタンスのサーバーログでデプロイメントの成功を確認することもできます。

```
WFLYSRV0027: Starting deployment of "helloworld.war" (runtime-name: "helloworld.war")
WFLYUT0021: Registered web context: /helloworld
WFLYSRV0010: Deployed "helloworld.war" (runtime-name : "helloworld.war")
```

アプリケーションのアンデプロイ

**helloworld** クイックスタートディレクトリーで以下の Maven コマンドを実行します。

```
$ mvn wildfly:undeploy
```

アンデプロイする Maven コマンドの実行後、ターミナルウィンドウにはアンデプロイメントの成功を表す以下の出力が表示されます。

```
[INFO] -----
[INFO] BUILD SUCCESS
```

```
[INFO] -----
[INFO] Total time: 1.237 s
[INFO] Finished at: 2015-12-23T15:09:10-05:00
[INFO] Final Memory: 10M/183M
[INFO] -----
```

アクティブなサーバーインスタンスのサーバーログでアンデプロイメントの成功を確認することもできます。

```
WFLYUT0022: Unregistered web context: /helloworld
WFLYSRV0028: Stopped deployment helloworld.war (runtime-name: helloworld.war) in 27ms
WFLYSRV0009: Undeployed "helloworld.war" (runtime-name: "helloworld.war")
```

#### 7.4.2. Maven を使用した管理対象ドメインでのアプリケーションのデプロイ

以下の手順では、Maven を使用して管理対象ドメインで JBoss EAP の **helloworld** クイックスタートをデプロイおよびアンデプロイする方法を示します。

JBoss EAP クイックスタートの詳細は、JBoss EAP『[スタートガイド](#)』の「クイックスタートサンプルの使用」を参照してください。

##### アプリケーションのデプロイ

管理対象ドメインでアプリケーションをデプロイする場合、アプリケーションをデプロイするサーバーグループを指定する必要があります。これは、Maven の **pom.xml** ファイルで設定されます。

**pom.xml** の以下の設定は WildFly Maven Plugin を初期化し、**main-server-group** をアプリケーションがデプロイされるサーバーグループとして指定します。

```
<plugin>
  <groupId>org.wildfly.plugins</groupId>
  <artifactId>wildfly-maven-plugin</artifactId>
  <version>${version.wildfly.maven.plugin}</version>
  <configuration>
    <domain>
      <server-groups>
        <server-group>main-server-group</server-group>
      </server-groups>
    </domain>
  </configuration>
</plugin>
```

**helloworld** クイックスタートディレクトリーで以下の Maven コマンドを実行します。

```
$ mvn clean install wildfly:deploy
```

デプロイする Maven コマンドの実行後、ターミナルウィンドウにはデプロイメントの成功を表す以下の出力が表示されます。

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.005 s
```



```
[INFO] Finished at: 2016-09-02T14:36:17-04:00
[INFO] Final Memory: 21M/226M
[INFO] -----
```

アクティブなサーバーインスタンスのサーバーログでデプロイメントの成功を確認することもできます。

```
WFLYSRV0027: Starting deployment of "helloworld.war" (runtime-name: "helloworld.war")
WFLYUT0021: Registered web context: /helloworld
WFLYSRV0010: Deployed "helloworld.war" (runtime-name : "helloworld.war")
```

アプリケーションのアンデプロイ

**helloworld** クイックスタートディレクトリーで以下の Maven コマンドを実行します。

```
$ mvn wildfly:undeploy
```

アンデプロイする Maven コマンドの実行後、ターミナルウィンドウにはアンデプロイメントの成功を表す以下の出力が表示されます。

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.750 s
[INFO] Finished at: 2016-09-02T14:45:10-04:00
[INFO] Final Memory: 10M/184M
[INFO] -----
```

アクティブなサーバーインスタンスのサーバーログでアンデプロイメントの成功を確認することもできます。

```
WFLYUT0022: Unregistered web context: /helloworld
WFLYSRV0028: Stopped deployment helloworld.war (runtime-name: helloworld.war) in 106ms
WFLYSRV0009: Undeployed "helloworld.war" (runtime-name: "helloworld.war")
```

## 7.5. HTTP API を使用したアプリケーションのデプロイ

HTTP API を使用してアプリケーションを JBoss EAP にデプロイするには、**curl** コマンドを使用します。HTTP API の使用に関する詳細は、「[HTTP API](#)」を参照してください。

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

デフォルトでは、HTTP API は **http://HOST:PORT/management** でアクセスできます (例: **http://localhost:9990/management**)。

アプリケーションのデプロイ

```
$ curl --digest -L -D - http://HOST:PORT/management --header "Content-Type: application/json" -u
USER:PASSWORD -d '{"operation" : "composite", "address" : [], "steps" : [{"operation" : "add",
"address" : {"deployment" : "test-application.war"}, "content" : [{"url" : "file:/path/to/test-
application.war"}]}, {"operation" : "deploy", "address" : {"deployment" : "test-
application.war"}]}], "json.pretty": 1}'
```

アプリケーションのアンデプロイ



スタンドアロンサーバーでのカスタムディレクトリーの定義  
 デフォルトでは、スタンドアロンサーバーのデプロイされたコンテンツは  
**EAP\_HOME/standalone/data/content** ディレクトリーに格納されます。この場所を変更するには、  
 サーバーの起動時に **-Djboss.server.deploy.dir** 引数を渡します。

```
$ EAP_HOME/bin/standalone.sh -Djboss.server.deploy.dir=/path/to/new_deployed_content
```

選択する場所は JBoss EAP インスタンスすべてで一意になる必要があります。



### 注記

**jboss.server.deploy.dir** プロパティーは、管理コンソールまたは管理 CLI を使用してデプロイされたコンテンツの格納に使用されるディレクトリーを指定します。デプロイメントスキャナーによって監視されるカスタムデプロイメントディレクトリーを定義する場合は、「[デプロイメントスキャナーの設定](#)」を参照してください。

管理対象ドメインでのカスタムディレクトリーの定義  
 デフォルトでは、管理対象ドメインのデプロイされたコンテンツは  
**EAP\_HOME/standalone/data/content** ディレクトリーに格納されます。この場所を変更するには、ドメインの起動時に **-Djboss.domain.deployment.dir** 引数を渡します。

```
$ EAP_HOME/bin/domain.sh -Djboss.domain.deployment.dir=/path/to/new_deployed_content
```

選択する場所は JBoss EAP インスタンスすべてで一意になる必要があります。

## 7.6.2. デプロイメント順序の制御

JBoss EAP では、サーバー起動時にアプリケーションがデプロイされる順序を細かく制御できます。複数の EAR ファイルに存在するアプリケーションのデプロイメント順序を徹底することができ、再起動後の順序を永続化することもできます。

**jboss-all.xml** デプロイメント記述子を使用してトップレベルのデプロイメント間で依存関係を宣言できます。

たとえば、最初にデプロイされた **framework.ear** に依存する **app.ear** がある場合、以下のように **app.ear/META-INF/jboss-all.xml** ファイルを作成できます。

```
<jboss xmlns="urn:jboss:1.0">
  <jboss-deployment-dependencies xmlns="urn:jboss:deployment-dependencies:1.0">
    <dependency name="framework.ear" />
  </jboss-deployment-dependencies>
</jboss>
```



### 注記

デプロイメントのランタイム名を **jboss-all.xml** ファイルの依存名として使用することができます。

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



## 重要

**jboss-all.xml** ファイルや他のデプロイメント記述子では、宣言しないとサーバーが検出しない依存関係を宣言できますが、これは厳格な順序付け機能ではありません。JBoss EAP は、デプロイメント記述子で指定されたすべての依存関係はデプロイ済みで利用可能であると仮定します。不足している依存関係がある場合、JBoss EAP はそれらの依存関係を自動的にデプロイせず、デプロイメントは失敗します。

### 7.6.3. デプロイメントコンテンツのオーバーライド

デプロイメントオーバーレイを使用すると、デプロイメントアーカイブのコンテンツを物理的に変更せずに既存デプロイメントにコンテンツをオーバーレイすることができます。これにより、アーカイブを再構築せずに実行時にデプロイメント記述子、ライブラリー JAR ファイル、クラス、JSP ページ、およびその他のファイルをオーバーライドできます。

これは、異なる設定が必要な異なる環境にデプロイメントを適応する必要がある場合に便利です。たとえば、アプリケーションのライフサイクルに従って開発からテスト、ステージ、および実稼働とデプロイメントを移動する場合、目的の環境に応じてデプロイメント記述子の交換、アプリケーションのブランディングを変更するための静的 Web リソースの変更、JAR ライブラリーの別バージョンへの置き換えなどを行う可能性があります。また、方針やセキュリティの制限によりアーカイブを変更できない、設定変更が必要なインストールにも便利な機能です。

デプロイメントオーバーレイを定義する場合、デプロイメントアーカイブのファイルを置き換える、ファイルシステム上のファイルを定義します。さらに、デプロイメントオーバーレイの影響を受けるデプロイメントを指定する必要もあります。変更を有効にするには、影響を受けるデプロイメントを再デプロイする必要があります。

**deployment-overlay add** 管理 CLI コマンドを使用してデプロイメントオーバーレイを追加します。

```
deployment-overlay add --name=new-deployment-overlay --content=WEB-INF/web.xml=/path/to/other/web.xml --deployments=test-application.war --redeploy-affected
```



## 注記

管理対象ドメインでは、**--server-groups** を使用して該当するサーバーグループを指定するか、**--all-server-groups** を使用してすべてのサーバーグループを指定します。

作成後、既存のオーバーレイへのコンテンツの追加、オーバーレイのデプロイメントへのリンク、またはオーバーレイの削除を行うことができます。使用方法の詳細を表示するには **deployment-overlay --help** を実行してください。

### パラメーター

#### name

デプロイメントオーバーレイの名前。

#### content

ファイルシステム上のファイルをアーカイブの置き換えるファイルにマップするカンマ区切りリスト。各エントリーの形式は **ARCHIVE\_PATH=FILESYSTEM\_PATH** です。

#### deployments

このオーバーレイがリンクされるデプロイメントのカンマ区切りリスト。

#### redeploy-affected

影響を受けるデプロイメントをすべて再デプロイします。

## 7.6.4. ロールアウト計画の使用

### ロールアウト計画

管理対象ドメインでは、ドメインまたはホストレベルのリソースで目的となる操作は複数のサーバーに影響する可能性があります。このような操作には、サーバーに適用される操作の順序を説明するロールアウト計画や、一部のサーバーで実行に失敗した場合に操作を元に戻すかどうかを指示するポリシーなどが含まれます。指定されたロールアウト計画がない場合、[デフォルトのロールアウト計画](#)が使用されます。

以下は5つのサーバーグループが関係するロールアウト計画の例になります。操作は順次 (**in-series**) または同時 (**concurrent-groups**) にサーバーグループへ適用することができます。構文の詳細は「[ロールアウト計画の構文](#)」を参照してください。

```
{ "my-rollout-plan" => { "rollout-plan" => {
  "in-series" => [
    { "concurrent-groups" => {
      "group-A" => {
        "max-failure-percentage" => "20",
        "rolling-to-servers" => "true"
      },
      "group-B" => undefined
    } },
    { "server-group" => { "group-C" => {
      "rolling-to-servers" => "false",
      "max-failed-servers" => "1"
    } },
    { "concurrent-groups" => {
      "group-D" => {
        "max-failure-percentage" => "20",
        "rolling-to-servers" => "true"
      },
      "group-E" => undefined
    } }
  ],
  "rollback-across-groups" => "true"
}}
```

上記の例を見ると、3つの段階を経てドメインのサーバーに操作が適用されることが分かります。あるサーバーグループのポリシーによってサーバーグループ全体で操作のロールバックが引き起こされると、他のサーバーグループもすべてロールバックされます。

1. サーバーグループ **group-A** と **group-B** には同時に操作が適用されます。**group-A** のサーバーには操作が順次適用され、**group-B** のすべてのサーバーは操作を同時に処理します。**group-A** で操作の適用に失敗したサーバーが 20% を超えると、グループ全体でロールバックが実行されます。**group-B** で操作を適用できなかったサーバーがあると、このグループ全体でロールバックが実行されます。
2. **group-A** と **group-B** のすべてのサーバーが完了すると、**group-C** のサーバーに操作が適用されます。これらのサーバーは操作を同時に処理します。**group-C** では操作を適用できなかったサーバーが 2 台以上あると、グループ全体でロールバックが実行されます。
3. **group-C** のすべてのサーバーが完了すると、操作が **group-D** と **group-E** に同時に適用されます。**group-D** のサーバーには操作が順次適用され、**group-E** のすべてのサーバーは操作を同時に処理します。**group-D** で操作の適用に失敗したサーバーが 20% を超えると、グループ全体でロールバックが実行されます。**group-E** で操作を適用できなかったサーバーがあると、このグループ全体でロールバックが実行されます。

## ロールアウト計画の構文

ロールアウト計画は以下のいずれかの方法で指定できます。

- **deploy** コマンド操作ヘッダーでロールアウト計画を定義します。詳細は「[ロールアウト計画を使用したデプロイ](#)」を参照してください。
- **rollout-plan** コマンドを使用してロールアウト計画を保存し、**deploy** コマンド操作ヘッダーでプラン名を参照します。詳細は「[保存したロールアウト計画を使用したデプロイ](#)」を参照してください。

上記の方法で最初に使用するコマンドは異なりますが、どちらも **rollout** 操作ヘッダーを使用してロールアウト計画を定義します。以下の構文を使用します。

```
rollout (id=PLAN_NAME | SERVER_GROUP_LIST) [rollback-across-groups]
```

- **PLAN\_NAME** は、**rollout-plan** コマンドを使用して保存されたロールアウト計画の名前です。
- **SERVER\_GROUP\_LIST** はサーバーグループのリストです。各サーバーグループで操作を順次実行する場合はコンマ (,) を使用して複数のサーバーグループを区切ります。各サーバーグループで同時に操作を実行する場合はキャレット (^) を区切り文字として使用します。
  - 各サーバーグループでは、以下のポリシーをカッコで囲んで設定します。複数のポリシーはコンマで区切ります。
    - **rolling-to-servers**: ブール値。 **true** に設定すると、操作はグループの各サーバーに順次適用されます。 **false** に設定した場合、または指定がない場合は、操作はグループのサーバーに同時に適用されます。
    - **max-failed-servers**: 整数値。グループにおける操作の適用に失敗したサーバー合計数の最大率 (パーセント) で、失敗したサーバーの率がこの値を超えるとグループのすべてのサーバーが元に戻されます。指定がない場合のデフォルト値は **0** で、操作の適用に失敗したサーバーが1台でもあるとグループ全体でロールバックが引き起こされます。
    - **max-failed-servers: 0** から **100** までの整数値。グループにおける操作の適用に失敗したサーバー合計数の最大率 (パーセント) で、失敗したサーバーの率がこの値を超えるとグループのすべてのサーバーが元に戻されます。指定がない場合のデフォルト値は **0** で、操作の適用に失敗したサーバーが1台でもあるとグループ全体でロールバックが引き起こされます。



### 注記

**max-failed-servers** と **max-failure-percentage** の両方がゼロ以外の値に設定された場合、**max-failure-percentage** が優先されます。

- **rollback-across-groups**: ブール値。1つのサーバーグループのサーバーすべてで操作をロールバックする必要があると、すべてのサーバーグループでロールバックの実行を引き起こすかどうかを指定します。デフォルトは **false** です。

## ロールアウト計画を使用したデプロイ

ロールアウト計画の完全詳細を直接 **deploy** コマンドに提供するには、**rollout** 設定を **headers** 引数に渡します。形式の詳細は「[ロールアウト計画の構文](#)」を参照してください。

以下の管理 CLI コマンドは、順次のデプロイメントに **rolling-to-servers=true** を指定するデプロイメント計画を使用して、アプリケーションを **main-server-group** サーバーグループにデプロイします。

```
deploy /path/to/test-application.war --server-groups=main-server-group --headers={rollout main-server-group(rolling-to-servers=true)}
```

保存したロールアウト計画を使用したデプロイ

ロールアウト計画は複雑になることがあるため、ロールアウト計画の詳細を保存するオプションがあります。これにより、毎回ロールアウト計画の完全詳細を必要とせずに、ロールアウト計画を使用するときにロールアウト計画の名前を参照することができます。

1. **rollout-plan** 管理 CLI コマンドを使用してロールアウト計画を保存します。形式の詳細は「[ロールアウト計画の構文](#)」を参照してください。

```
rollout-plan add --name=my-rollout-plan --content={rollout main-server-group(rolling-to-servers=false,max-failed-servers=1),other-server-group(rolling-to-servers=true,max-failure-percentage=20) rollback-across-groups=true}
```

これにより、以下のデプロイメント計画が作成されます。

```
"rollout-plan" => {
  "in-series" => [
    {"server-group" => {"main-server-group" => {
      "rolling-to-servers" => false,
      "max-failed-servers" => 1
    }},
    {"server-group" => {"other-server-group" => {
      "rolling-to-servers" => true,
      "max-failure-percentage" => 20
    }}
  ],
  "rollback-across-groups" => true
}
```

2. アプリケーションのデプロイ時に保存されたロールアウト計画名を指定します。以下の管理 CLI コマンドは、保存されたロールアウト計画 **my-rollout-plan** を使用してすべてのサーバーグループにアプリケーションをデプロイします。

```
deploy /path/to/test-application.war --all-server-groups --headers={rollout id=my-rollout-plan}
```

保存されたロールアウト計画の削除

削除するロールアウト計画の名前を指定して **rollout-plan** 管理 CLI コマンドを使用すると、保存されたロールアウト計画を削除できます。

```
rollout-plan remove --name=my-rollout-plan
```

デフォルトのロールアウト計画

複数のサーバーに影響する操作はすべてロールアウト計画を使用して実行されます。操作リクエストにロールアウト計画が指定されていない場合は、デフォルトのロールアウト計画が生成されます。計画には以下の特徴があります。

- ハイレベルなフェーズは1つのみです。操作に影響するすべてのサーバーグループには同時に操作が適用されます。
- 各サーバーグループ内では、操作がすべてのサーバーに同時に適用されます。

- サーバーグループのいずれかのサーバーに操作を適用できないと、グループ全体でロールバックが実行されます。
- あるサーバーグループに操作を適用できないと、他のすべてのサーバーグループでロールバックが実行されます。

## 7.7. 展開形式のデプロイメントの管理

JBoss EAP 7.1 以前では、ファイルシステム上のファイルを操作するのが展開形式のデプロイメントを管理する唯一の方法でした。JBoss EAP 7.1 より、管理インターフェースを使用して展開形式のデプロイメントを管理できるようになりました。これにより、新しいバージョンのアプリケーションをデプロイしなくても、展開形式のアプリケーションのコンテンツを変更できるようになりました。



### 注記

JavaScript や CSS ファイルなど、デプロイメントの静的ファイルが更新されると、即座に更新が反映されます。Java クラスなどの他のファイルが変更された場合は、変更の反映にアプリケーションの再デプロイが必要になることがあります。

最初に、[空のデプロイメント](#)を使用するか、[既存のアーカイブデプロイメントを展開](#)した後、[コンテンツ](#)を追加または削除します。

「[デプロイメントのコンテンツの表示](#)」を参照してデプロイメントのファイルを閲覧するか、ファイルのコンテンツを読み取ります。

### 空の展開形式のデプロイメントを作成

必要時にコンテンツを追加できる空の展開形式のデプロイメントを作成できます。以下の管理 CLI コマンドを使用して空の展開形式のデプロイメントを作成します。

```
/deployment=DEPLOYMENT_NAME.war:add(content={{empty=true}})
```

`empty=true` オプションは、空のデプロイメントの作成を確認するために必要です。

### 既存のアーカイブデプロイメントの展開

既存のアーカイブデプロイメントを展開してコンテンツを更新できます。展開する前に[デプロイメントを無効化](#)する必要があります。以下の管理 CLI コマンドを使用してデプロイメントを展開します。

```
/deployment=ARCHIVE_DEPLOYMENT_NAME.ear:explode
```

これで、このデプロイメントの[コンテンツを追加または削除](#)できるようになります。



### 注記

管理コンソールから既存のアーカイブデプロイメントを展開することもできます。[Deployments](#) タブからデプロイメントを選択し、ドロップダウンメニューで **Explode** を選択します。

### 展開形式のデプロイメントへのコンテンツの追加

デプロイメントにコンテンツを追加するには、`add-content` 管理 CLI 操作を使用します。コンテンツを追加するデプロイメントの場所へのパスを指定し、アップロードするコンテンツを提供します。アップロードするコンテンツは、ローカルファイルストリーム、URL、JBoss EAP コンテンツリポジトリに



すでに存在するコンテンツのハッシュ、またはコンテンツのバイトアレイとして提供できます。以下の管理 CLI コマンドは、**input-stream-index** オプションを使用してローカルファイルのコンテンツをデプロイメントにアップロードします。

```
/deployment=DEPLOYMENT_NAME.war:add-content(content={{target-path=/path/to/FILE_IN_DEPLOYMENT, input-stream-index=/path/to/LOCAL_FILE_TO_UPLOAD}})
```



### 注記

**add-content** 操作を使用してコンテンツをデプロイメントに追加するとき、デプロイメントのコンテンツはデフォルトで上書きされます。この挙動を変更するには、**overwrite** オプションを **false** に設定します。

## 展開形式のデプロイメントのコンテンツの削除

デプロイメントからコンテンツを削除するには、**remove-content** 管理 CLI 操作を使用し、デプロイメントの削除するコンテンツへのパスを指定します。

```
/deployment=DEPLOYMENT_NAME.war:remove-content(paths=/path/to/FILE_1, /path/to/FILE_2)
```

## 7.8. デプロイメントのコンテンツの表示

JBoss EAP の管理インターフェースを使用すると、管理されたデプロイメントのファイルに関する [情報](#) を閲覧し、ファイルの [内容を読み取る](#) ことができます。

### 7.8.1. デプロイメントのファイルの閲覧

管理されたデプロイメントのファイルやディレクトリーを閲覧するには、**browse-content** 操作を使用します。引数を指定しないと、デプロイメント構造全体が返されます。特定のディレクトリーへのパスを指定するには、**path** 引数を使用します。



### 注記

また、管理コンソールからデプロイメントの内容を閲覧することもできます。**Deployments** タブに移動してデプロイメントを選択し、ドロップダウンメニューで表示を選択します。

```
/deployment=helloworld.war:browse-content(path=META-INF/)
```

上記は、**helloworld.war** デプロイメントの **META-INF/** ディレクトリーにあるファイルとディレクトリーを表示します。

```
{
  "outcome" => "success",
  "result" => [
    {
      "path" => "MANIFEST.MF",
      "directory" => false,
      "file-size" => 827L
    },
    {
      "path" => "maven/org.jboss.eap.quickstarts/helloworld/pom.properties",
```

```

        "directory" => false,
        "file-size" => 106L
    },
    {
        "path" => "maven/org.jboss.eap.quickstarts/helloworld/pom.xml",
        "directory" => false,
        "file-size" => 2713L
    },
    {
        "path" => "maven/org.jboss.eap.quickstarts/helloworld/",
        "directory" => true
    },
    {
        "path" => "maven/org.jboss.eap.quickstarts/",
        "directory" => true
    },
    {
        "path" => "maven/",
        "directory" => true
    },
    {
        "path" => "INDEX.LIST",
        "directory" => false,
        "file-size" => 251L
    }
]
}

```

以下の引数を **browse-content** 操作に指定することもできます。

#### archive

アーカイブファイルのみを返すかどうか。

#### depth

返すファイルの深さを指定します。

### 7.8.2. デプロイメントコンテンツの読み取り

管理されたデプロイメントでファイルの内容を読み取るには、**read-content** 操作を使用します。引数を指定しないと、デプロイメント全体が返されます。または、**path** 引数を指定して、特定のファイルへのパスを提供します。例を以下に示します。

```
/deployment=helloworld.war:read-content(path=META-INF/MANIFEST.MF)
```

上記は、[管理 CLI に表示](#) または [ファイルシステムに保存](#) できるファイルストリームを返します。

```

{
  "outcome" => "success",
  "result" => {"uuid" => "24ba8e06-21bd-4505-b4d4-bdfb16451b95"},
  "response-headers" => {"attached-streams" => [{"uuid" => "24ba8e06-21bd-4505-b4d4-bdfb16451b95", "mime-type" => "text/plain"}]}
}

```

### 7.8.2.1. ファイルの内容の表示

**attachment display** コマンドを使用して **MANIFEST.MF** ファイルの内容を読み取ります。

```
attachment display --operation=/deployment=helloworld.war:read-content(path=META-INF/MANIFEST.MF)
```

上記は、**helloworld.war** デプロイメントからの **MANIFEST.MF** ファイルの内容を管理 CLI に表示します。

```
ATTACHMENT 8af87836-2abd-423a-8e44-e731cc57bd80:  
Manifest-Version: 1.0  
Implementation-Title: Quickstart: helloworld  
Implementation-Version: 7.2.0.GA  
Java-Version: 1.8.0_131  
Built-By: username  
Scm-Connection: scm:git:git@github.com:jboss/jboss-parent-pom.git/quic  
kstart-parent/helloworld  
Specification-Vendor: JBoss by Red Hat  
...
```

### 7.8.2.2. ファイルの内容の保存

**attachment save** コマンドを使用して、**MANIFEST.MF** ファイルの内容をファイルシステムに保存します。

```
attachment save --operation=/deployment=helloworld.war:read-content(path=META-INF/MANIFEST.MF) --file=/path/to/MANIFEST.MF
```

上記は、**helloworld.war** デプロイメントからの **MANIFEST.MF** ファイルを **path/to/MANIFEST.MF** のファイルシステムに保存します。 **--file** 引数を使用してファイルパスを指定しないと、一意な添付 ID を使用してファイル名が付けられ、管理 CLI の作業ディレクトリー (デフォルトは **EAP\_HOME/bin/**) に保存されます。

## 第8章 ドメイン管理

本項では、管理対象ドメインの操作モードに固有する概念や設定について説明します。

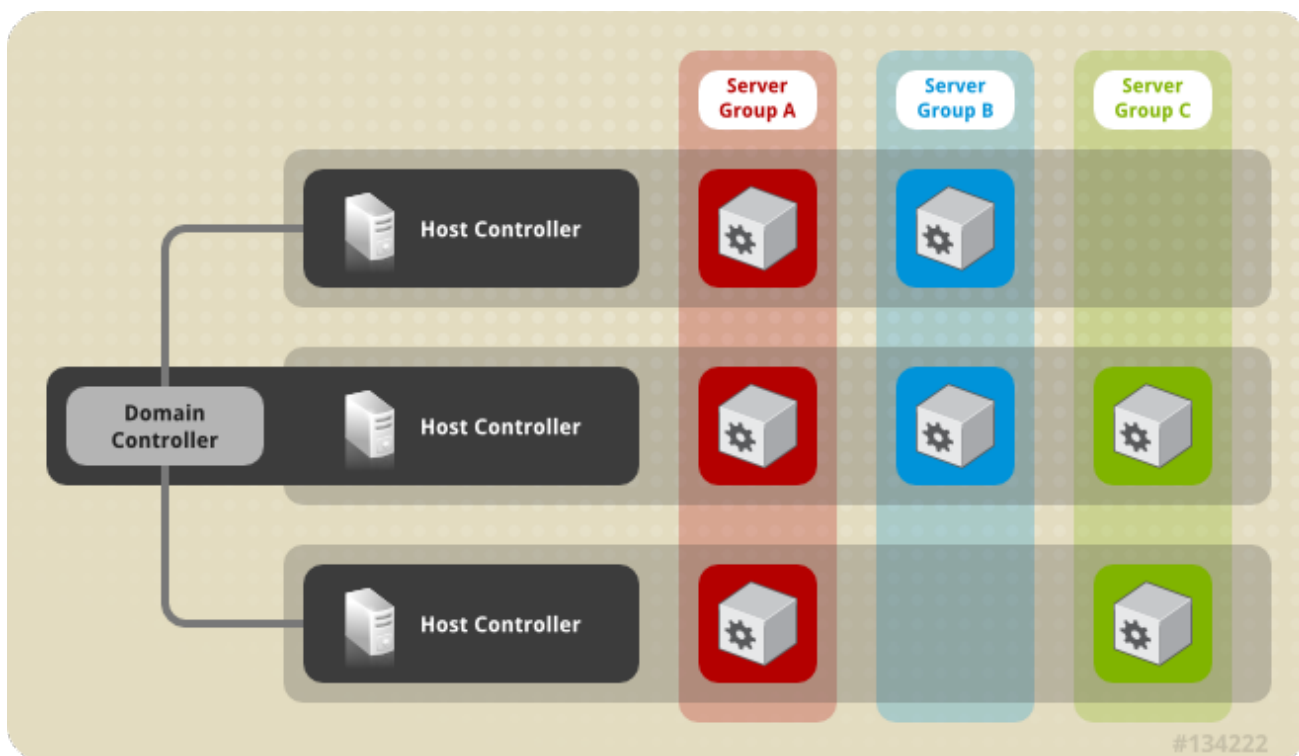
管理対象ドメインのセキュア化については、JBoss EAP『[How to Configure Server Security](#)』の「[Securing a Managed Domain](#)」の項を参照してください。

### 8.1. 管理対象ドメイン

管理対象ドメインの操作モードを使用すると、単一の制御ポイントから複数の JBoss EAP インスタンスを管理できます。

一元管理された複数の JBoss EAP サーバーは、ドメインのメンバーと呼ばれます。ドメインの JBoss EAP インスタンスは共通の管理ポリシーを共有します。

各ドメインは1つのドメインコントローラー、1つ以上のホストコントローラー、およびホスト毎に0個以上のサーバーグループによって構成されます。



**ドメインコントローラー**は、ドメインが制御される中心点であり、各サーバーはドメインの管理ポリシーに従って設定されます。ドメインの管理ポリシーに従って各サーバーが設定されるようにします。ドメインコントローラーはホストコントローラーでもあります。

**ホストコントローラー**はドメインコントローラーと対話して、ホスト上で実行されているアプリケーションサーバーインスタンスのライフサイクルを制御し、ドメインコントローラーがこれらのインスタンスを管理できるようにします。各ホストに複数のサーバーグループが含まれるようにすることが可能です。

**サーバーグループ**は、JBoss EAP がインストールされている **サーバー** インスタンスの集まりで、すべてが1つとして管理および設定されます。ドメインコントローラーはサーバーグループにデプロイされたアプリケーションとその設定を管理します。そのため、サーバーグループの各サーバーは同じ設定とデプロイメントを共有します。

ホストコントローラーは特定の物理または仮想ホストに割り当てられます。異なる設定を使用する場合は、同じハードウェア上で複数のホストコントローラーを実行でき、ポートとその他のリソースが競合

しないようにします。1つのドメインコントローラー、1つのホストコントローラー、および複数のサーバーを、同じ物理システムの同じ JBoss EAP インスタンス内で実行することができます。

### 8.1.1. ドメインコントローラー

ドメインコントローラーは、ドメインの集中管理点として動作する JBoss EAP サーバーインスタンスです。1つのホストコントローラーインスタンスがドメインコントローラーとして動作するように設定されます。

ドメインコントローラーの主な役割は次のとおりです。

- ドメインの集中管理ポリシーを維持する。
- すべてのホストコントローラーが現在のコンテンツを認識するようにする。
- 実行中のすべての JBoss EAP サーバーインスタンスがこのポリシーに従って設定されるよう、ホストコントローラーをサポートする。

デフォルトでは、集中管理ポリシーは **EAP\_HOME/domain/configuration/domain.xml** ファイルに格納されます。このファイルは、ドメインコントローラーとして実行するように設定されたホストコントローラーのこのディレクトリーに必要です。

**domain.xml** ファイルには、ドメインのサーバーに適用できるプロファイル設定が含まれています。[プロファイル](#)には、そのプロファイルで使用できるさまざまなサブシステムの詳細設定が含まれています。ドメイン設定には、ソケットグループの定義とサーバーグループの定義も含まれます。



#### 注記

JBoss EAP 7 ドメインコントローラーは、JBoss EAP 6.2 以上を実行している JBoss EAP 6 ホストおよびサーバーを管理できます。詳細は、「[JBoss EAP 6 インスタンスを管理するよう JBoss EAP 7.x ドメインコントローラーを設定](#)」を参照してください

詳細は「[管理対象ドメインの起動](#)」および「[ドメインコントローラーの設定](#)」の項を参照してください。

### 8.1.2. ホストコントローラー

ホストコントローラーの主な役割はサーバーを管理することです。ドメイン管理タスクを委譲し、ホスト上で実行される個別のアプリケーションサーバープロセスを開始および停止します。

ホストコントローラーはドメインコントローラーと対話し、サーバーとドメインコントローラー間の通信を管理できるようにします。ドメインの複数のホストコントローラーは1つのドメインコントローラーのみと対話できます。そのため、単一のドメインモード上で実行されるホストコントローラーおよびサーバーインスタンスはすべて単一のドメインコントローラーを持ち、同じドメインに属する必要があります。

デフォルトでは、各ホストコントローラーは、ホストのファイルシステムの未展開の JBoss EAP インストールファイルにある **EAP\_HOME/domain/configuration/host.xml** ファイルから設定を読み取ります。**host.xml** ファイルには、特定のホストに固有する以下の設定情報が含まれています。

- このインストールから実行されるサーバーインスタンスの名前。
- ローカルの物理インストールに固有する設定。たとえば、**domain.xml** に宣言された名前付きインターフェースの定義は **host.xml** にある実際のマシン固有の IP アドレスマッピングできません。さらに、**domain.xml** の抽象パス名を **host.xml** のファイルシステムパスマッピングできま

す。

- 次の設定のいずれか。
  - ホストコントローラーがドメインコントローラーへ通知して、ホストコントローラー自体を登録し、ドメイン設定へアクセスする方法。
  - リモートドメインコントローラーの検索および通知方法。
  - ホストコントローラーがドメインコントローラーとして動作するかどうか。

詳細は「[管理対象ドメインの起動](#)」および「[ホストコントローラーの設定](#)」の項を参照してください。

### 8.1.3. プロセスコントローラー

プロセスコントローラーは小型のライトウェイトなプロセスで、ホストコントローラープロセスを起動し、そのライフサイクルを監視します。ホストコントローラーがクラッシュすると、プロセスコントローラーによって再起動されます。さらに、ホストコントローラーの指示に従ってサーバープロセスを開始しますが、クラッシュしたサーバープロセスは自動的に再起動しません。

プロセスコントローラーのログは **EAP\_HOME/domain/log/process-controller.log** ファイルに記録されます。プロセスコントローラーの JVM オプションは、**PROCESS\_CONTROLLER\_JAVA\_OPTS** 変数を使用して **EAP\_HOME/bin/domain.conf** ファイルに設定します。

### 8.1.4. サーバーグループ

サーバーグループとは、1つのグループとして管理および設定される複数のサーバーインスタンスのことです。管理対象ドメインでは、各アプリケーションサーバーインスタンスは唯一のメンバーである場合でも1つのサーバーグループに属します。グループのサーバーインスタンスは同じプロファイル設定とデプロイされたコンテンツを共有します。

ドメインコントローラーとホストコントローラーは、ドメインにある各サーバーグループのすべてのインスタンスに標準設定を強制します。

ドメインを複数のサーバーグループで構成できます。異なるサーバーグループを異なるプロファイルやデプロイメントで設定できます。たとえば、ドメインは異なるサービスを提供する異なるサーバー層で設定できます。

異なるサーバーグループが同じプロファイルやデプロイメントを持つこともできます。これにより、最初のサーバーグループでアプリケーションがアップグレードされた後に2つ目のサーバーグループでアプリケーションがアップデートされるアプリケーションのローリングアップグレードが可能になり、サービスの完全停止を防ぎます。

詳細は、「[サーバーグループの設定](#)」の項を参照してください。

### 8.1.5. サーバー

サーバーはアプリケーションサーバーインスタンスを表します。管理対象ドメインでは、すべてのサーバーインスタンスがサーバーグループのメンバーになります。ホストコントローラーは各サーバーインスタンスを独自の JVM プロセスで起動します。

詳細は「[サーバーの設定](#)」の項を参照してください。

## 8.2. ドメイン設定のナビゲート

JBoss EAP は、規模の小さい管理対象ドメインと規模の大きい管理対象ドメインの両方をサポートするスケーラブルな管理インターフェースを提供します。

## 管理コンソール

JBoss EAP 管理コンソールは、ドメインのグラフィカルビューを提供し、ドメインのホスト、サーバー、デプロイメント、およびプロファイルの管理を容易にします。

### Configuration (設定)

**Configuration** タブからドメインで使用される各プロファイルのサブシステムを設定できます。必要な機能に応じて、メインの異なるサーバーグループが異なるプロファイルを使用することがあります。

希望のプロファイルを選択すると、そのプロファイルで利用できるサブシステムがすべて表示されます。プロファイルの設定に関する詳細は「[JBoss EAP プロファイルの管理](#)」を参照してください。

### Runtime (ランタイム)

**Runtime** タブから、サーバー、サーバーグループ、およびホストの設定を管理できます。ホストまたはサーバーグループごとにドメインを閲覧できます。

**Hosts** から、ホストプロパティと JVM を設定でき、そのホストのサーバーを追加および設定することもできます。

**Server Groups** から、新しいサーバーグループを追加でき、プロパティや JVM を設定できます。また、そのサーバーグループにサーバーを追加および設定することもできます。選択したサーバーグループの全サーバーの起動、停止、一時停止、およびリロードなど、操作を実行できます。

**Hosts** または **Server Groups** から、新しいサーバーを追加でき、サーバープロパティや JVM を設定できます。選択したサーバーの起動、停止、一時停止、およびリロードなどの操作を実行できます。また、JVM の使用率、サーバーログ、サブシステム固有の情報など、ランタイム情報を表示することもできます。

**Topology** から、ドメインのホスト、サーバーグループ、およびサーバーの概要や詳細情報を表示できます。再ロードや停止などの操作をそれぞれに実行できます。

### Deployments (デプロイメント)

**Deployments** タブから、デプロイメントをサーバーグループに追加およびデプロイできます。コンテンツリポジトリのすべてのデプロイメントや、特定のサーバーグループにデプロイされたデプロイメントを表示できます。

管理コンソールを使用したアプリケーションのデプロイに関する詳細は「[管理対象ドメインでのアプリケーションのデプロイ](#)」を参照してください。

## 管理 CLI

JBoss EAP 管理 CLI は、ドメインのホスト、サーバー、デプロイメントおよびプロファイルを管理するコマンドラインインターフェースを提供します。

適切なプロファイルを選択するとサブシステムの設定にアクセスできます。

```
/profile=PROFILE_NAME/subsystem=SUBSYSTEM_NAME:read-resource(recursive=true)
```



## 注記

本ガイドの手順や例では、スタンドアロンサーバーとして稼働している場合に適用されるサブシステム設定の管理 CLI コマンドが含まれている可能性があります。例を以下に示します。

```
/subsystem=datasources/data-source=ExampleDS:read-resource
```

管理対象ドメインで実行するために管理 CLI コマンドを調整するには、設定するプロファイルを指定する必要があります。例を以下に示します。

```
/profile=default/subsystem=datasources/data-source=ExampleDS:read-resource
```

適切なホストの指定後、ホストを設定し、そのホストのサーバーで操作を実行することができます。

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

適切なホストの指定後、そのホストのサーバーを設定できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:write-attribute(name=ATTRIBUTE_NAME,value=VALUE)
```

適切なサーバーグループの指定後、サーバーグループを設定し、選択したサーバーグループのすべてのサーバーで操作を実行することができます。

```
/server-group=SERVER_GROUP_NAME:read-resource
```

**deploy** 管理 CLI コマンドを使用し、適切なサーバーグループを指定すると、管理対象ドメインでアプリケーションをデプロイできます。手順は、「[管理対象ドメインでのアプリケーションのデプロイ](#)」を参照してください。

## 8.3. 管理対象ドメインの起動

### 8.3.1. 管理対象ドメインの起動

ドメインおよびホストコントローラーは、JBoss EAP に同梱される **domain.sh** または **domain.bat** スクリプトを使用して起動できます。使用できる起動スクリプトの引数の完全リストとそれら引数の目的については、**--help** 引数を使用するか、「[サーバーランタイム引数](#)」を参照してください。

ドメイン内のサーバーグループのスレーブサーバーを起動する前にドメインコントローラーを起動する必要があります。最初にドメインコントローラーを起動した後、ドメイン内の関連する他のホストコントローラーを起動します。

ドメインコントローラーの起動

専用のドメインコントローラー向けに事前設定された **host-master.xml** 設定ファイルを使用してドメインコントローラーを起動します。

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

ドメインの設定に応じて、ホストコントローラーの接続を可能にするために設定を追加する必要があります。また、以下のドメイン設定例を参照してください。



- 1台のマシンで管理対象ドメインを設定
- 2台のマシンで管理対象ドメインを設定

#### ホストコントローラーの起動

スレーブホストコントローラー向けに事前設定された **host-slave.xml** 設定ファイルを使用してホストコントローラーを起動します。

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

ドメインの設定に応じて、ドメインコントローラーへ接続し、ドメインコントローラーとの競合を回避するために設定を追加する必要があります。また、以下のドメイン設定例を参照してください。

- 1台のマシンで管理対象ドメインを設定
- 2台のマシンで管理対象ドメインを設定

### 8.3.2. ドメインコントローラーの設定

ドメイン内のホストをドメインコントローラーとして設定する必要があります。



#### 重要

RPM インストールで JBoss EAP をインストールした場合、複数のドメインまたはホストコントローラーを同じマシン上に設定することはサポートされません。

**<domain-controller>** 宣言に **<local/>** 要素を追加して、ホストをドメインコントローラーとして設定します。**<domain-controller>** には他のコンテンツを含めないでください。

```
<domain-controller>
  <local/>
</domain-controller>
```

ドメインコントローラーとして動作するホストは、ドメインの他のホストがアクセスできる管理インターフェースを公開する必要があります。HTTP インターフェースは標準の管理インターフェースです。

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

最小のドメインコントローラー設定ファイルの例 (**EAP\_HOME/domain/configuration/host-master.xml**) には、以下の設定が含まれます。

### 8.3.3. ホストコントローラーの設定

ホストコントローラー自体をドメインに登録するようにホストコントローラーを設定する必要があります。



## 重要

RPM インストールで JBoss EAP をインストールした場合、複数のドメインまたはホストコントローラーを同じマシン上に設定することはサポートされません。

設定の **<domain-controller>** 要素を使用して、ドメインコントローラーへの接続を設定します。

```
<domain-controller>
  <remote security-realm="ManagementRealm">
    <discovery-options>
      <static-discovery name="primary" protocol="{jboss.domain.master.protocol:remote}"
host="{jboss.domain.master.address}" port="{jboss.domain.master.port:9990}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

最小のホストコントローラー設定ファイルのサンプル **EAP\_HOME/domain/configuration/host-slave.xml** には、ドメインコントローラーに接続する設定が含まれています。この設定は、ホストコントローラーの起動時に **jboss.domain.master.address** プロパティを指定するものと想定しています。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -
Djboss.domain.master.address=IP_ADDRESS
```

ドメインコントローラーの検出に関する詳細は、「[ドメインコントローラーの検出およびフェイルオーバー](#)」の項を参照してください。

ドメインの設定によっては、認証を提供して、ホストコントローラーがドメインコントローラーによって認証されるようにする必要がある場合があります。管理ユーザーと秘密の値の生成や、その値を用いたホストコントローラー設定の更新に関する詳細は、「[2台のマシンで管理対象ドメインを設定](#)」を参照してください。

### 8.3.4. ホスト名の設定

管理対象ドメインで実行されている各ホストには一意な名前を付ける必要があります。管理を容易にし、複数のホストで同じホスト設定ファイルを使用できるようにするために、以下の優先順位を用いてホスト名が決定されます。

1. 設定されている場合、**host.xml** 設定ファイルのホスト要素名属性。
2. **jboss.host.name** システムプロパティの値。
3. **jboss.qualified.host.name** システムプロパティの最後のピリオド (.) の後に続く値。最後のピリオド (.) がない場合は全体の値。
4. POSIX ベースのオペレーティングシステムでは、**HOSTNAME** 環境変数のピリオド (.) の後に続く値。Microsoft Windows では **COMPUTERNAME** 環境変数のピリオド (.) の後に続く値。最後のピリオドがない場合は、全体の値。

関連する **host.xml** 設定ファイルの上部にある **host** 要素に設定されたホストコントローラーの名前。例は次のとおり。

```
<host xmlns="urn:jboss:domain:8.0" name="host1">
```

以下の手順に従って、管理 CLI を使用してホスト名を更新します。

1. JBoss EAP ホストコントローラーを起動します。

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

2. 管理 CLI を起動し、ドメインコントローラーに接続します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --
controller=DOMAIN_CONTROLLER_IP_ADDRESS
```

3. 以下のコマンドを実行して新しいホスト名を設定します。

```
/host=EXISTING_HOST_NAME:write-attribute(name=name,value=NEW_HOST_NAME)
```

これにより、**host-slave.xml** ファイルのホスト名属性が以下のように変更されます。

```
<host name="NEW_HOST_NAME" xmlns="urn:jboss:domain:8.0">
```

4. 変更を反映するためにホストコントローラーをリロードします。

```
reload --host=EXISTING_HOST_NAME
```

ホストコントローラーの名前が設定ファイルに設定されていない場合は、起動時にホスト名を渡すこともできます。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -Djboss.host.name=HOST_NAME
```

## 8.4. サーバーの管理

### 8.4.1. サーバークラスタの設定

以下はサーバークラスタ定義の例になります。

```
<server-group name="main-server-group" profile="full">
  <jvm name="default">
    <heap size="64m" max-size="512m"/>
  </jvm>
  <socket-binding-group ref="full-sockets"/>
  <deployments>
    <deployment name="test-application.war" runtime-name="test-application.war"/>
    <deployment name="helloworld.war" runtime-name="helloworld.war" enabled="false"/>
  </deployments>
</server-group>
```

サーバークラスタの設定は、管理 CLI を使用するか、管理コンソールの **Runtime** タブから行います。

サーバークラスタの追加

以下の管理 CLI コマンドを実行すると、サーバークラスタを追加できます。

```
/server-group=SERVER_GROUP_NAME:add(profile=PROFILE_NAME,socket-binding-
group=SOCKET_BINDING_GROUP_NAME)
```

### サーバーグループの更新

以下の管理 CLI コマンドを実行すると、サーバーグループ属性を更新できます。

```
/server-group=SERVER_GROUP_NAME:write-attribute(name=ATTRIBUTE_NAME,value=VALUE)
```

### サーバーグループの削除

以下の管理 CLI コマンドを実行すると、サーバーグループを削除できます。

```
/server-group=SERVER_GROUP_NAME:remove
```

### サーバーグループ属性

サーバーグループには次の属性が必要です。

- **name**: サーバーグループの名前。
- **profile**: サーバーグループプロファイル名。
- **socket-binding-group**: グループのサーバーに使用されるデフォルトのソケットバインディンググループ。サーバーごとに上書きできます。

サーバーグループに含まれる任意の属性は次のとおりです。

- **management-subsystem-endpoint: remoting** サブシステムからエンドポイントを使用してサーバーグループに属するサーバーを再びホストコントローラーに接続するには **true** に設定します。これは、**remoting** サブシステムが存在しないと機能しません。
- **socket-binding-default-interface**: このサーバーのソケットバインディンググループのデフォルトインターフェース。
- **socket-binding-port-offset**: ソケットバインディンググループによって提供されたポート値に追加するデフォルトのオフセット。
- **deployments**: グループのサーバーにデプロイするデプロイメントコンテンツ。
- **jvm**: グループの全サーバーに対するデフォルトの JVM 設定。ホストコントローラーはこれらの設定を **host.xml** の他の設定とマージし、サーバーの JVM を開始するために使用される設定を作成します。
- **deployment-overlays**: 定義されたデプロイメントオーバーレイと、このサーバーグループのデプロイメントとの間をリンクします。
- **system-properties**: グループのサーバーに設定するシステムプロパティ。

## 8.4.2. サーバーの設定

デフォルトの **host.xml** 設定ファイルは 3 つのサーバーを定義します。

```
<servers>
  <server name="server-one" group="main-server-group">
  </server>
  <server name="server-two" group="main-server-group" auto-start="true">
    <socket-bindings port-offset="150"/>
  </server>
  <server name="server-three" group="other-server-group" auto-start="false">
```

```
<socket-bindings port-offset="250"/>
</server>
</servers>
```

**server-one** という名前のサーバーインスタンスは **main-server-group** に関連付けられ、そのサーバーグループによって指定されるサブシステム設定とソケットバインディングを継承します。**server-two** という名前のサーバーインスタンスも **main-server-group** に関連付けられていますが、**server-one** によって使用されるポートの値と競合しないようにソケットバインディングの **port-offset** の値も定義します。**server-three** という名前のサーバーインスタンスは **other-server-group** に関連付けられ、そのグループの設定を使用します。また、**port-offset** の値も定義し、ホストコントローラーの起動時にこのサーバーが起動しないように **auto-start** を **false** に設定します。

サーバーの設定は、管理 CLI を使用するか、管理コンソールの **Runtime** タブから行います。

サーバーの追加

以下の管理 CLI コマンドを実行すると、サーバーを追加できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:add(group=SERVER_GROUP_NAME)
```

サーバーの更新

以下の管理 CLI コマンドを実行すると、サーバー属性を更新できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:write-attribute(name=ATTRIBUTE_NAME,value=VALUE)
```

サーバーの削除

以下の管理 CLI コマンドを実行すると、サーバーを削除できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:remove
```

サーバーの属性

サーバーには以下の属性が必要です。

- **name**: サーバーの名前。
- **group**: ドメインモデルからのサーバーグループの名前。

サーバーには以下の任意の属性が含まれます。

- **auto-start**: ホストコントローラーの起動時にこのサーバーが起動されるかどうか。
- **socket-binding-group**: このサーバーが属するソケットバインディンググループ。
- **socket-binding-port-offset**: このサーバーのソケットバインディンググループによって提供されたポート値に追加されるオフセット。
- **update-auto-start-with-server-status**: サーバーの状態を **auto-start** 属性を更新します。
- **interface**: サーバーで使用できる完全に指定された名前付きのネットワークインターフェースのリスト。
- **jvm**: このサーバーの JVM 設定。宣言されていない場合、設定は親のサーバーグループまたはホストから継承されます。
- **path**: 名前付きのファイルシステムパスのリスト。

- **system-property**: このサーバーに設定するシステムプロパティのリスト。

### 8.4.3. サーバーの起動と停止

管理コンソールから起動、停止、リロードなどの操作をサーバーで実行するには、**Runtime** タブに移動して適切なホストまたはサーバーグループを選択します。

管理 CLI を使用してこれらの操作を実行する場合は以下のコマンドを参照してください。

サーバーの起動

特定のホストで1台のサーバーを起動できます。

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

指定のサーバーグループのサーバーをすべて起動できます。

```
/server-group=SERVER_GROUP_NAME:start-servers
```

サーバーの停止

特定のホストで1台のサーバーを停止できます。

```
/host=HOST_NAME/server-config=SERVER_NAME:stop
```

指定のサーバーグループのサーバーをすべて停止できます。

```
/server-group=SERVER_GROUP_NAME:stop-servers
```

サーバーのリロード

特定のホストで1台のサーバーをリロードできます。

```
/host=HOST_NAME/server-config=SERVER_NAME:reload
```

指定のサーバーグループのサーバーをすべてリロードできます。

```
/server-group=SERVER_GROUP_NAME:reload-servers
```

サーバーの Kill

指定のサーバーグループのすべてのサーバープロセスを kill することができます。

```
/server-group=SERVER_GROUP_NAME:kill-servers
```

## 8.5. ドメインコントローラーの検出およびフェイルオーバー

管理対象ドメインの設定時、ドメインコントローラーに接続するために必要な情報を使用して各ホストコントローラーを設定する必要があります。JBoss EAP では、ドメインコントローラーを検索する [複数のオプション](#) を使用して各ホストコントローラーを設定できます。ホストコントローラーは、適切なオプションが見つかるまでオプションのリストを繰り返し処理します。

プライマリードメインコントローラーに問題がある場合は、バックアップホストコントローラーを [ドメインコントローラーに昇格](#) できます。これにより、昇格後にホストコントローラーを新しいドメインコントローラーへ自動的にフェイルオーバーできます。

### 8.5.1. ドメイン検出オプションの設定

以下は、ドメインコントローラー検索の複数のオプションを持つホストコントローラーを設定する方法の例になります。

例: 複数のドメインコントローラーオプションを持つホストコントローラー

```
<domain-controller>
  <remote security-realm="ManagementRealm">
    <discovery-options>
      <static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"
        host="172.16.81.100" port="${jboss.domain.master.port:9990}"/>
      <static-discovery name="backup" protocol="${jboss.domain.master.protocol:remote}"
        host="172.16.81.101" port="${jboss.domain.master.port:9990}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

静的検出オプションには、以下の必須属性が含まれます。

#### name

このドメインコントローラー検出オプションの名前。

#### host

リモートドメインコントローラーのホスト名。

#### port

リモートドメインコントローラーのポート。

上記の例では、最初の検出オプションが指定されることが予想されます。2つ目のオプションはフェイルオーバーの状況で使用される可能性があります。

### 8.5.2. キャッシュされたドメイン設定を用いたホストコントローラーの起動

**--cached-dc** オプションを使用すると、ドメインコントローラーへ接続していなくてもホストコントローラーを起動できますが、ホストコントローラーは前もってドメインコントローラーからドメイン設定をローカルでキャッシュする必要があります。**--cached-dc** オプションを使用してホストコントローラーを起動すると、ドメインコントローラーからホストコントローラーのドメイン設定がキャッシュされます。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml --cached-dc
```

これにより、このホストコントローラーがドメインコントローラーに接続せずに現在のサーバーを一時的に管理するために必要な情報が含まれる **EAP\_HOME/domain/configuration/** ディレクトリーに **domain.cached-remote.xml** ファイルが作成されます。



#### 注記

デフォルトでは、**--cached-dc** オプションを使用すると、このホストコントローラーによって使用される設定のみがキャッシュされます。そのため、ドメイン全体のドメインコントローラーには昇格されません。ドメイン設定全体をキャッシュしてホストコントローラーをドメインコントローラーとする場合は、「[ドメイン設定のキャッシュ](#)」を参照してください。

**--cached-dc** を使用してこのホストコントローラーを起動したときにドメインコントローラーを利用できない場合、ホストコントローラーは **domain.cached-remote.xml** ファイルに保存されたキャッシュされた設定の使用を開始します。このファイルが存在しないとホストコントローラーは起動しないことに注意してください。

この状態の間、ホストコントローラーはドメイン設定を編集できませんが、サーバーを開始したり、デプロイメントを管理することはできます。

ホストコントローラーがキャッシュされた設定を使用して起動されると、ホストコントローラーは継続してドメインコントローラーへの再接続を試みます。ドメインコントローラーが使用できるようになると、ホストコントローラーは自動的にドメインコントローラーに再接続し、ドメイン設定を同期化します。変更された設定によっては、変更の反映のにホストコントローラーのリロードが必要になることがあります。この場合、警告がホストコントローラーのログに記録されます。

### 8.5.3. ホストコントローラーがドメインコントローラーとして動作するよう昇格

主要のドメインコントローラーに問題が発生した場合に、ホストコントローラーがドメインコントローラーとして動作するよう昇格できます。ホストコントローラーを **昇格** する前に、ドメインコントローラーから **ドメイン設定をローカルでキャッシュ** する必要があります。

ドメイン設定のキャッシュ

ドメインコントローラーとして使用したいホストコントローラーに **--backup** オプションを使用します。

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

これにより、ドメイン設定全体のコピーが含まれる **EAP\_HOME/domain/configuration/** ディレクトリーに **domain.cached-remote.xml** ファイルが作成されます。この設定は、ホストコントローラーがドメインコントローラーとして動作するよう再設定された場合に使用されます。

#### 注記

**ignore-unused-configuration** 属性は、特定のホストがキャッシュする設定の範囲を判断するために使用されます。**true** を指定すると、このホストコントローラーに関する設定のみがキャッシュされ、ドメインコントローラーとして引き継ぐことはできません。**false** を指定すると、ドメイン設定全体がキャッシュされます。

The **--backup** 引数はデフォルトでこの属性の値を **false** とし、ドメイン全体をキャッシュします。しかし、この属性を **host.xml** ファイルで設定した場合は、その値が使用されます。

また、**--cached-dc** オプションのみを使用して、ドメイン設定のコピーを作成できますが、**host.xml** で **ignore-unused-configuration** の値を **false** と設定し、ドメイン全体をキャッシュする必要があります。例を以下に示します。

```
<domain-controller>
  <remote username="$local" security-realm="ManagementRealm" ignore-unused-configuration="false">
    <discovery-options>
      ...
    </discovery-options>
  </remote>
</domain-controller>
```



ホストコントローラーをドメインコントローラーに昇格

1. 元のドメインコントローラーが停止した状態であることを確認します。
2. 管理 CLI を使用して、新しいドメインコントローラーとなるホストコントローラーへ接続します。
3. 以下のコマンドを実行してホストコントローラーを設定し、新しいドメインコントローラーとして動作するようにします。

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

4. 以下のコマンドを実行し、ホストコントローラーをリロードします。

```
reload --host=HOST_NAME
```

このホストコントローラーがドメインコントローラーとして動作するようになります。

## 8.6. 管理対象ドメインの設定

### 8.6.1.1 台のマシンで管理対象ドメインを設定

**jboss.domain.base.dir** プロパティを使用すると 1 台のマシンで複数のホストコントローラーを実行できます。



#### 重要

複数の JBoss EAP ホストコントローラーを 1 台のマシン上でシステムサービスとして設定することはサポートされません。

1. ドメインコントローラーの **EAP\_HOME/domain** ディレクトリーをコピーします。

```
$ cp -r EAP_HOME/domain /path/to/domain1
```

2. ホストコントローラーの **EAP\_HOME/domain** ディレクトリーをコピーします。

```
$ cp -r EAP_HOME/domain /path/to/host1
```

3. **/path/to/domain1** を使用してドメインコントローラーを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-master.xml -  
Djboss.domain.base.dir=/path/to/domain1
```

4. **/path/to/host1** を使用してホストコントローラーを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -  
Djboss.domain.base.dir=/path/to/host1 -Djboss.domain.master.address=IP_ADDRESS -  
Djboss.management.http.port=PORT
```



## 注記

ホストコントローラーの起動時に、**jboss.domain.master.address** プロパティを使用してドメインコントローラーのアドレスを指定する必要があります。

さらに、このホストコントローラーはドメインコントローラーと同じマシンで実行されているため、ドメインコントローラーの管理インターフェースと競合しないように管理インターフェースを変更する必要があります。このコマンドは **jboss.management.http.port** プロパティを設定します。

このように起動された各インスタンスは、ベースインストールディレクトリー (例: **EAP\_HOME/modules**) のその他のリソースを共有しますが、**jboss.domain.base.dir** によって指定されたディレクトリーからドメイン設定を使用します。

### 8.6.2.2 台のマシンで管理対象ドメインを設定



## 注記

この例の実行には、ファイアウォールの設定が必要になることがあります。

1台がドメインコントローラーでもう1台がホストである2台のマシンで管理対象ドメインを作成できます。詳細は、「[ドメインコントローラー](#)」を参照してください。

- **IP1** = ドメインコントローラーの IP アドレス (マシン 1)
- **IP2** = ホストの IP アドレス (マシン 2)

2台のマシンで管理対象ドメインを作成

#### 1. マシン 1 での作業

- ホストがドメインコントローラーによって認証されるよう、管理ユーザーを追加します。**add-user.sh** スクリプトを使用してホストコントローラー **HOST\_NAME** の管理ユーザーを追加します。最後の質問で **yes** を指定し、提供される秘密の値を書き留めてください。この秘密の値はホストコントローラーの設定で使用され、以下のように表示されます。

```
<secret value="SECRET_VALUE" />
```

- ドメインコントローラーを起動します。  
専用のドメインコントローラー用に事前設定された **host-master.xml** 設定ファイルを指定します。さらに、**jboss.bind.address.management** プロパティを設定し、ドメインコントローラーが他のマシンにも表示されるようにします。

```
$ EAP_HOME/bin/domain.sh --host-config=host-master.xml -  
Djboss.bind.address.management=IP1
```

#### 2. マシン 2 での作業

- ユーザー認証情報でホスト設定を更新します。  
**EAP\_HOME/domain/configuration/host-slave.xml** を編集し、ホスト名 **HOST\_NAME** と秘密の値 **SECRET\_VALUE** を設定します。

```
<host xmlns="urn:jboss:domain:8.0" name="HOST_NAME">
```

```
<management>
  <security-realms>
    <security-realm name="ManagementRealm">
      <server-identities>
        <secret value="SECRET_VALUE" />
      </server-identities>
    </security-realm>
  </security-realms>
  ...
</management>
```

- b. ホストコントローラーを起動します。

スレーブホストコントローラー用に事前設定された **host-slave.xml** 設定ファイルを指定します。さらに、**jboss.domain.master.address** プロパティを設定してドメインコントローラーに接続し、**jboss.bind.address** プロパティを設定してホストコントローラーバインドアドレスを設定します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -
  Djboss.domain.master.address=IP1 -Djboss.bind.address=IP2
```

起動時に **--controller** パラメーターを使用してドメインコントローラーアドレスを指定し、管理 CLI からドメインを管理できるようになります。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --controller=IP1
```

また、**http://IP1:9990** で管理コンソールからドメインを管理することもできます。

## 8.7. 複数の JBOSS EAP バージョンの管理

最新バージョンの JBoss EAP は、以前のバージョンの JBoss EAP を実行するサーバーおよびホストを管理できます。管理する JBoss EAP のバージョンに該当する項を参照してください。

- [JBoss EAP 6 インスタンスを管理するよう JBoss EAP 7.x ドメインコントローラーを設定](#)
- [JBoss EAP 7.0 インスタンスを管理するよう JBoss EAP 7.1 ドメインコントローラーを設定](#)

### 8.7.1. JBoss EAP 6 インスタンスを管理するよう JBoss EAP 7.x ドメインコントローラーを設定

JBoss EAP 7.1 ドメインコントローラーは、実行している JBoss EAP 6 のバージョンが JBoss EAP 6.2 以上のホストおよびサーバーを管理できます。



#### 注記

JBoss EAP 7.0 ドメインコントローラーが異なるパッチリリースの JBoss EAP 7.0 ホストを管理する場合は、設定を変更する必要はありません。しかし、JBoss EAP 7.0 ドメインコントローラーは、管理するホストコントローラーと同じパッチリリースバージョンまたはそれ以降のバージョンを実行する必要があります。

JBoss EAP 7 の管理対象ドメインで JBoss EAP 6 インスタンスを正常に管理するには、以下の作業を行います。

1. [Boss EAP 6 の設定を JBoss EAP 7 ドメインコントローラーに追加](#)
2. [JBoss EAP 6 プロファイルの動作を更新します。](#)

- JBoss EAP 6 サーバーのサーバーグループを設定します。
- JBoss EAP 6 インスタンスが JBoss EAP 7 の更新を受け取らないようにします。

これらの作業が終了したら、管理 CLI を使用して JBoss EAP 7 ドメインコントローラーから JBoss EAP 6 サーバーおよび設定を管理できます。JBoss EAP 6 ホストは、バッチ処理などの JBoss EAP 7 の新機能を利用できないことに注意してください。



### 警告

管理コンソールは、最新バージョンの JBoss EAP に対して最適化されているため、管理コンソールを使用して JBoss EAP 6 のホスト、サーバー、およびプロファイルを更新しないでください。JBoss EAP 7 の管理対象ドメインから JBoss EAP 6 の設定を管理する場合は、代わりに管理 CLI を使用してください。

#### 8.7.1.1. Boss EAP 6 の設定を JBoss EAP 7 ドメインコントローラーに追加

ドメインコントローラーが JBoss EAP 6 サーバーを管理できるようにするには、JBoss EAP 7 のドメイン設定に JBoss EAP 6 の設定詳細を提供する必要があります。これには、JBoss EAP 6 のプロファイル、ソケットバインディンググループ、およびサーバーグループを JBoss EAP 7 の **domain.xml** 設定ファイルにコピーします。

JBoss EAP 7 の設定と名前が競合する場合は、リソースの名前を変更する必要があります。さらに、正しく動作するようにするため、追加の調整を行う必要もあります。

以下の手順では、JBoss EAP 6 の **default** プロファイル、**standard-sockets** ソケットバインディンググループ、および **main-server-group** サーバーグループが使用されます。

- JBoss EAP 7 の **domain.xml** 設定ファイルを編集します。このファイルをバックアップしてから編集することが推奨されます。
- 該当する JBoss EAP 6 のプロファイルを JBoss EAP 7 の **domain.xml** ファイルにコピーします。  
この手順では、JBoss EAP 6 の **default** プロファイルをコピーし、名前を **eap6-default** に変更したことを仮定します。

#### JBoss EAP 7 の domain.xml

```
<profiles>
...
<profile name="eap6-default">
...
</profile>
</profiles>
```

- このプロファイルによって使用される必要なエクステンションを追加します。  
JBoss EAP 6 のプロファイルが JBoss EAP 7 には存在しないサブシステムを使用する場合は、JBoss EAP ドメイン設定に適切なエクステンションを追加する必要があります。

#### JBoss EAP 7 の domain.xml

-

```

<extensions>
...
<extension module="org.jboss.as.configadmin"/>
<extension module="org.jboss.as.threads"/>
<extension module="org.jboss.as.web"/>
</extensions>

```

4. 該当する JBoss EAP 6 のソケットバインディンググループを JBoss EAP 7 の **domain.xml** ファイルにコピーします。  
この手順では、JBoss EAP 6 の **standard-sockets** ソケットバインディンググループをコピーし、名前を **eap6-standard-sockets** に変更したことを仮定します。

#### JBoss EAP 7 の domain.xml

```

<socket-binding-groups>
...
<socket-binding-group name="eap6-standard-sockets" default-interface="public">
...
</socket-binding-group>
</socket-binding-groups>

```

5. 該当する JBoss EAP 6 のサーバーグループを JBoss EAP 7 の **domain.xml** ファイルにコピーします。  
この手順では、JBoss EAP 6 の **main-server-group** サーバーグループをコピーし、名前を **eap6-main-server-group** に変更したことを仮定します。また、このサーバーグループを更新して、JBoss EAP 6 のプロファイル **eap6-default** と JBoss EAP 6 のソケットバインディンググループ **eap6-standard-sockets** を使用するようになる必要があります。

#### JBoss EAP 7 の domain.xml

```

<server-groups>
...
<server-group name="eap6-main-server-group" profile="eap6-default">
...
  <socket-binding-group ref="eap6-standard-sockets"/>
</server-group>
</server-groups>

```

### 8.7.1.2. JBoss EAP 6 プロファイルの動作の更新

JBoss EAP のバージョンや必要な動作に応じて、JBoss EAP 6 インスタンスによって使用されるプロファイルを追加更新する必要があります。既存の JBoss EAP 6 インスタンスが使用するサブシステムや設定に応じて、追加の変更が必要になる場合があります。

JBoss EAP 7 ドメインコントローラーを起動して管理 CLI を起動し、以下の更新を実行します。これらの例では、JBoss EAP 6 のプロファイルが **eap6-default** であることを仮定します。

- **bean-validation** サブシステムを削除します。  
JBoss EAP 7 では、bean バリデーション機能が **ee** サブシステムから独自の **bean-validation** サブシステムに移されました。JBoss EAP 7 ドメインコントローラーがレガシーの **ee** サブシステムを発見した場合、新しい **bean-validation** サブシステムを追加します。しかし、JBoss EAP 6 のホストはこのサブシステムを認識しないため、削除する必要があります。

#### JBoss EAP 7 のドメインコントローラー CLI

```
/profile=eap6-default/subsystem=bean-validation:remove
```

- CDI 1.0 の挙動を設定します。  
これは、JBoss EAP 6 サーバーに JBoss EAP 7 で使用されるより新しいバージョンの CDI の挙動ではなく、CDI 1.0 の挙動を適用する場合のみ必要です。CDI 1.0 の挙動を提供する場合は、**weld** サブシステムに以下の更新を追加します。

### JBoss EAP 7 のドメインコントローラー CLI

```
/profile=eap6-default/subsystem=weld:write-attribute(name=require-bean-descriptor,value=true)
```

```
/profile=eap6-default/subsystem=weld:write-attribute(name=non-portable-mode,value=true)
```

- JBoss EAP 6.2 のデータソース統計を有効にします。  
これは、プロファイルが JBoss EAP 6.2 サーバーによって使用される場合のみ必要です。JBoss EAP 6.3 には **statistics-enabled** 属性が導入され、デフォルトでは統計を収集しない **false** に設定されます。しかし、JBoss EAP 6.2 では統計を収集しました。このプロファイルが JBoss EAP 6.2 のホストとそれ以降のバージョンの JBoss EAP を実行するホストによって使用される場合、ホストによって動作が異なることになり、これは許可されません。そのため、JBoss EAP 6.2 ホスト向けのプロファイルは、データソースに以下の変更を追加する必要があります。

### JBoss EAP 7 のドメインコントローラー CLI

```
/profile=eap6-default/subsystem=datasources/data-source=ExampleDS:write-attribute(name=statistics-enabled,value=true)
```

#### 8.7.1.3. JBoss EAP 6 サーバーのサーバーグループの設定

サーバーグループの名前を変更した場合は、JBoss EAP 6 のホスト設定を更新し、JBoss EAP 7 の設定に指定された新しいサーバーグループを使用する必要があります。この例では、JBoss EAP 7 の **domain.xml** に指定された **eap6-main-server-group** サーバーグループを使用します。

#### JBoss EAP 6 の host-slave.xml

```
<servers>
  <server name="server-one" group="eap6-main-server-group"/>
  <server name="server-two" group="eap6-main-server-group">
    <socket-bindings port-offset="150"/>
  </server>
</servers>
```



#### 注記

ホストは、実行している JBoss EAP よりも新しいバージョンで導入された機能や設定を使用できません。

#### 8.7.1.4. JBoss EAP 6 インスタンスが JBoss EAP 7 の更新を取得しないようにする

管理対象ドメインのドメインコントローラーは、設定の更新をホストコントローラーに転送します。**host-exclude** 設定を使用して、特定のバージョンが認識できないようにするリソースを指定する必要があります。事前設定された **host-exclude** のオプションは **EAP62**、**EAP63**、**EAP64**、および

**EAP64z** です。ご使用の JBoss EAP 6 のバージョンに該当するものを選択します。

**host-exclude** 設定の **active-server-groups** 属性は、特定のバージョンによって使用されるサーバーグループのリストを指定します。指定のバージョンのホストは、指定されたサーバーグループとそれらに関連するプロファイル、ソケットバインディンググループ、およびデプロイメントリソースを利用できますが、それ以外は認識しません。

以下の例では、JBoss EAP のバージョンが 6.4.z であることを仮定し、JBoss EAP 6 のサーバーグループ **eap6-main-server-group** をアクティブなサーバーグループとして追加します。

### JBoss EAP 7 のドメインコントローラー CLI

```
/host-exclude=EAP64z:write-attribute(name=active-server-groups,value=[eap6-main-server-group])
```

必要な場合は、**active-socket-binding-groups** 属性を使用して、サーバーによって使用される追加のソケットバインディンググループを指定します。これは、**active-server-groups** に指定されたサーバーグループと関連していないソケットバインディンググループのみに必要です。

## 8.7.2. JBoss EAP 7.0 インスタンスを管理するよう JBoss EAP 7.1 ドメインコントローラーを設定

JBoss EAP 7.1 のドメインコントローラーは、JBoss EAP 7.0 を実行しているホストおよびサーバーを管理できます。



### 注記

JBoss EAP 7.1 ドメインコントローラーが異なるパッチリリースの JBoss EAP 7.1 ホストを管理する場合は、設定を変更する必要はありません。しかし、JBoss EAP 7.1 ドメインコントローラーは、管理するホストコントローラーと同じパッチリリースバージョンまたはそれ以降のバージョンを実行している必要があります。

JBoss EAP 7.1 の管理対象ドメインで JBoss EAP 7.0 インスタンスを正常に管理するには、以下の作業を行います。

1. [Boss EAP 7.0 の設定を JBoss EAP 7.1 ドメインコントローラーに追加](#)
2. [JBoss EAP 7.0 サーバーのサーバーグループを設定します。](#)
3. [JBoss EAP 7.0 インスタンスが JBoss EAP 7.1 の更新を受け取らないようにします。](#)

これらの作業が終了したら、管理 CLI を使用して JBoss EAP 7.1 ドメインコントローラーから JBoss EAP 7.0 サーバーおよび設定を管理できます。JBoss EAP 7.0 ホストは、**elytron** サブシステムなどの JBoss EAP 7.1 の新機能を利用できないことに注意してください。



### 警告

管理コンソールは、最新バージョンの JBoss EAP に対して最適化されているため、管理コンソールを使用して JBoss EAP 7.0 のホスト、サーバー、およびプロファイルを更新しないでください。JBoss EAP 7.1 の管理対象ドメインから JBoss EAP 7.0 の設定を管理する場合は、代わりに管理 CLI を使用してください。

### 8.7.2.1. Boss EAP 7.0 の設定を JBoss EAP 7.1 ドメインコントローラーに追加

ドメインコントローラーが JBoss EAP 7.0 サーバーを管理できるようにするには、JBoss EAP 7.1 のドメイン設定に JBoss EAP 7.0 の設定詳細を提供する必要があります。これには、JBoss EAP 7.0 のプロファイル、ソケットバインディンググループ、およびサーバーグループを JBoss EAP 7.1 の **domain.xml** 設定ファイルにコピーします。

JBoss EAP 7.1 の設定と名前が競合する場合は、リソースの名前を変更する必要があります。

以下の手順では、JBoss EAP 7.0 の **default** プロファイル、**standard-sockets** ソケットバインディンググループ、および **main-server-group** サーバーグループが使用されます。

1. JBoss EAP 7.1 の **domain.xml** 設定ファイルを編集します。このファイルをバックアップしてから編集することが推奨されます。
2. 該当する JBoss EAP 7.0 のプロファイルを JBoss EAP 7.1 の **domain.xml** ファイルにコピーします。  
この手順では、JBoss EAP 7.0 の **default** プロファイルをコピーし、名前を **eap70-default** に変更したことを仮定します。

#### JBoss EAP 7.1 の domain.xml

```
<profiles>
...
<profile name="eap70-default">
...
</profile>
</profiles>
```

3. 該当する JBoss EAP 7.0 のソケットバインディンググループを JBoss EAP 7.1 の **domain.xml** ファイルにコピーします。  
この手順では、JBoss EAP 7.0 の **standard-sockets** ソケットバインディンググループをコピーし、名前を **eap70-standard-sockets** に変更したことを仮定します。

#### JBoss EAP 7.1 の domain.xml

```
<socket-binding-groups>
...
<socket-binding-group name="eap70-standard-sockets" default-interface="public">
...
</socket-binding-group>
</socket-binding-groups>
```

4. 該当する JBoss EAP 7.0 のサーバーグループを JBoss EAP 7.1 の **domain.xml** ファイルにコピーします。  
この手順では、JBoss EAP 7.0 の **main-server-group** サーバーグループをコピーし、名前を **eap70-main-server-group** に変更したことを仮定します。また、このサーバーグループを更新して、JBoss EAP 7.0 のプロファイル **eap70-default** と JBoss EAP 7.0 のソケットバインディンググループ **eap70-standard-sockets** を使用するようになります。

#### JBoss EAP 7.1 の domain.xml

```
<server-groups>
...
<server-group name="eap70-main-server-group" profile="eap70-default">
```



```

...
<socket-binding-group ref="eap70-standard-sockets"/>
</server-group>
</server-groups>

```

### 8.7.2.2. JBoss EAP 7.0 サーバーのサーバーグループの設定

サーバーグループの名前を変更した場合は、JBoss EAP 7.0 のホスト設定を更新し、JBoss EAP 7.1 の設定に指定された新しいサーバーグループを使用する必要があります。この例では、JBoss EAP 7.1 の `domain.xml` に指定された `eap70-main-server-group` サーバーグループを使用します。

#### JBoss EAP 7.0 の host-slave.xml

```

<servers>
  <server name="server-one" group="eap70-main-server-group"/>
  <server name="server-two" group="eap70-main-server-group">
    <socket-bindings port-offset="150"/>
  </server>
</servers>

```



#### 注記

ホストは、実行している JBoss EAP よりも新しいバージョンで導入された機能や設定を使用できません。

### 8.7.2.3. JBoss EAP 7.0 インスタンスが JBoss EAP 7.1 の更新を取得しないようにする

管理対象ドメインのドメインコントローラーは、設定の更新をホストコントローラーに転送します。JBoss EAP 7.0 のホストは JBoss EAP 7.1 固有の設定やリソースを取得しないため、JBoss EAP 7.0 のホストを設定してこれらのリソースを無視する必要があります。これには、JBoss EAP 7.0 のホストで `ignore-unused-configuration` 属性を設定します。



#### 注記

また、`host-exclude` 設定を使用して、特定のバージョンの JBoss EAP を実行するホストが特定のリソースを認識しないよう、ドメインコントローラーに指示することもできます。`host-exclude` 設定の使用例については、「[JBoss EAP 6 インスタンスが JBoss EAP 7 の更新を取得しないようにする](#)」を参照してください。JBoss EAP 7.0 のホストの場合、`host-exclude` のオプションとして `EAP70` を使用します。

JBoss EAP 7.0 ホストコントローラーのリモートドメインコントローラーへの接続設定で、`ignore-unused-configuration` 属性を `true` に設定します。

#### JBoss EAP 7.0 の host-slave.xml

```

<domain-controller>
  <remote security-realm="ManagementRealm" ignore-unused-configuration="true">
    <discovery-options>
      <static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}" port="${jboss.domain.master.port:9990}"/>
    </discovery-options>
  </remote>
</domain-controller>

```

この設定では、このホストは使用するサーバーグループと、それらのサーバーグループに関連するプロファイル、ソケットバインディング、およびデプロイメントリソースのみを利用できます。その他はすべて無視されます。

## 8.8. JBOSS EAP プロファイルの管理

### 8.8.1. プロファイル

JBoss EAP は、プロファイルを使用してサーバーが使用できるサブシステムを整理します。プロファイルは、利用可能なサブシステムと各サブシステムの特定の設定で構成されます。プロファイルのサブシステムの数が多いと、サーバーの機能が多くなります。プロファイルのサブシステムが集中的で数が少ないと、機能が少なくなりますが、フットプリントも少なくなります。

JBoss EAP にはほとんどのユースケースに対応する事前定義されたプロファイルが5つあります。

#### default

**logging**、**security**、**datasources**、**infinispan**、**webservices**、**ee**、**ejb3**、**transactions** など、一般的に使用されるサブシステムが含まれます。

#### ha

**default** プロファイルで提供されるサブシステムと、高可用性向けの **jgroups** および **modcluster** サブシステムが含まれます。

#### full

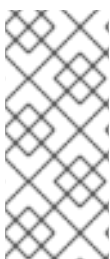
**default** プロファイルで提供されるサブシステムと、**messaging-activemq** および **iiop-openjdk** サブシステムが含まれます。

#### full-ha

**full** プロファイルで提供されるサブシステムと、高可用性向けの **jgroups** および **modcluster** サブシステムが含まれます。

#### load-balancer

ビルトインの `mod_cluster` フロントエンドロードバランサーを使用して他の JBoss EAP インスタンスの負荷を分散するために必要な最低限のサブシステムが含まれます。



#### 注記

JBoss EAP は、既存プロファイルの設定からサブシステムを削除して、エクステンションを無効にしたり、ドライバーやその他のサービスを手作業でアンロードしたりする機能を提供します。ただし、ほとんどの場合、これは必要ありません。JBoss EAP は必要時にサブシステムを動的にロードするため、サーバーまたはアプリケーションがサブシステムを使用しないと、そのサブシステムはロードされません。

既存のプロファイルが必要な機能を提供しない場合、JBoss EAP はカスタムプロファイルを定義する機能も提供します。

### 8.8.2. プロファイルのクローン

JBoss EAP では、既存のプロファイルをクローンして管理対象ドメインで新しいプロファイルを作成することができます。クローンした既存プロファイルの設定およびサブシステムのコピーが作成されます。

管理 CLI を使用してプロファイルをクローンするには、クローンするプロファイルに **clone** 操作を実行します。

```
/profile=full-ha:clone(to-profile=cloned-profile)
```

管理コンソールからプロファイルをクローンするには、クローンするプロファイルを選択し、**Clone** をクリックします。

### 8.8.3. 階層的なプロファイルの作成

管理対象ドメインでは、プロファイルの階層を作成できます。これにより、他のプロファイルが継承できる共通のエクステンションが含まれるベースプロファイルを作成できます。

管理対象ドメインは **domain.xml** の複数のプロファイルを定義します。複数のプロファイルが特定のサブシステムで同じ設定を使用する場合、複数のプロファイルで設定せずに、1つのプロファイルで設定を行うことができます。親プロファイルの値はオーバーライドできません。

さらに、各プロファイルは他に依存しなくてもすむ必要があります。要素やサブシステムが参照される場合、参照されるプロファイルに定義する必要があります。

管理 CLI を使用して **list-add** 操作を実行し、含めるプロファイルを指定すると、プロファイルに階層の別のプロファイルを含めることができます。

```
/profile=new-profile:list-add(name=includes, value=PROFILE_NAME)
```

## 第9章 JVM の設定

Java Virtual Machine (JVM) の設定は、スタンドアロンの JBoss EAP サーバーと管理対象ドメインの JBoss EAP サーバーでは異なります。

スタンドアロン JBoss EAP サーバーインスタンスでは、起動時にサーバー起動プロセスが JVM 設定を JBoss EAP サーバーに渡します。これは、JBoss EAP を起動する前にコマンドラインから宣言できます。また、管理コンソールの **Configuration** にある **System Properties** 画面を使用して宣言することもできます。

管理対象ドメインでは、JVM の設定は **host.xml** および **domain.xml** 設定ファイルで宣言され、ホスト、サーバーグループ、またはサーバーレベルで設定できます。



### 注記

システムプロパティは、起動中に JBoss EAP モジュール (ロギングマネージャーなど) が使用する **JAVA\_OPTS** で設定する必要があります。

### 9.1. スタンドアロンサーバーの JVM 設定

スタンドアロン JBoss EAP サーバーインスタンスの JVM 設定は、サーバーの起動前に **JAVA\_OPTS** 環境変数を設定すると、起動時に宣言できます。

Linux で **JAVA\_OPTS** 環境変数を設定する例は次のとおりです。

```
$ export JAVA_OPTS="-Xmx1024M"
```

Microsoft Windows 環境でも同じ設定を使用できます。

```
set JAVA_OPTS="Xmx1024M"
```

この代わりに、JVM に渡すオプションの例が含まれる **EAP\_HOME/bin** フォルダの **standalone.conf** ファイル (Windows Server の場合は **standalone.conf.bat**) に JVM 設定を追加することもできます。



### 警告

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

### 9.2. 管理対象ドメインの JVM 設定

JBoss EAP 管理対象ドメインでは、複数のレベルで JVM の設定を定義できます。特定ホストのカスタム JVM 設定を定義し、それらの設定をサーバーグループまたは個別のサーバーインスタンスに適用できます。

デフォルトでは、サーバーグループおよび各サーバーは親から JVM 設定を継承しますが、各レベルで JVM 設定をオーバーライドすることもできます。



## 注記

**domain.conf** の JVM 設定 (Windows Server の場合は **domain.conf.bat**) は JBoss EAP ホストコントローラーの Java プロセスに適用されます。ホストコントローラーによって制御される個別の JBoss EAP サーバーインスタンスには適用されません。

### 9.2.1. ホストコントローラーの JVM 設定の定義

ホストコントローラーの JVM 設定を定義し、これらの設定をサーバーグループまたは各サーバーに適用することができます。JBoss EAP には **default** の JVM 設定が含まれますが、以下の管理 CLI コマンドを実行すると、カスタム JVM 設定およびオプションがある **production\_jvm** という名前の JVM 設定が新たに作成されます。

```
/host=HOST_NAME/jvm=production_jvm:add(heap-size=2048m, max-heap-size=2048m, max-permgen-size=512m, stack-size=1024k, jvm-options=["-XX:-UseParallelGC"])
```

使用できるすべてのオプションの説明は「[管理対象ドメインの JVM 設定属性](#)」を参照してください。

また、JBoss EAP 管理コンソールで JVM 設定を作成および編集するには、**Runtime** → **Hosts** と選択し、ホストを選択して **表示** をクリックし、**JVMs** タブを選択します。

これらの設定は **host.xml** の **<jvm>** タグ内に保存されます。

### 9.2.2. JVM 設定のサーバーグループへの適用

サーバーグループの作成時に、グループのすべてのサーバーが使用する JVM 設定を指定できます。以下の管理 CLI コマンドを実行すると、前の例で示された **production\_jvm** JVM 設定を使用する **groupA** という名前のサーバーグループが作成されます。

```
/server-group=groupA:add(profile=default, socket-binding-group=standard-sockets)
/server-group=groupA/jvm=production_jvm:add
```

サーバーグループのすべてのサーバーは、**production\_jvm** から JVM 設定を継承します。

サーバーグループレベルで特定の JVM 設定をオーバーライドすることもできます。たとえば、別のヒープサイズを設定するには、以下のコマンドを使用できます。

```
/server-group=groupA/jvm=production_jvm:write-attribute(name=heap-size,value="1024m")
```

上記コマンドの実行後、サーバーグループ **groupA** は **production\_jvm** から JVM 設定を継承しますが、ヒープサイズの値は **1024m** にオーバーライドされます。

使用できるすべてのオプションの説明は「[管理対象ドメインの JVM 設定属性](#)」を参照してください。

また、JBoss EAP 管理コンソールでサーバーグループの JVM 設定を編集するには、**Runtime** → **Server Groups** と選択し、サーバーグループを選択して **表示** をクリックし、**JVMs** タブを選択します。

サーバーグループの設定は **domain.xml** に保存されます。

### 9.2.3. JVM 設定の個別のサーバーへの適用

デフォルトでは、個別の JBoss EAP サーバーインスタンスは属するサーバーグループの JVM 設定を継承します。しかし、継承した設定をホストコントローラーからの別の JVM 設定定義でオーバーライドしたり、特定の JVM 設定をオーバーライドすることもできます。

たとえば、以下のコマンドは [前の例で示したサーバーグループ](#) の JVM 定義をオーバーライドし、**server-one** の JVM 設定を **default** の JVM 定義に設定します。

```
/host=HOST_NAME/server-config=server-one/jvm=default:add
```

また、サーバーグループと同様に、サーバーレベルで特定の JVM 設定をオーバーライドすることもできます。たとえば、別のヒープサイズを設定するには、以下のコマンドを使用できます。

```
/host=HOST_NAME/server-config=server-one/jvm=default:write-attribute(name=heap-size,value="1024m")
```

使用できるすべてのオプションの説明は「[管理対象ドメインの JVM 設定属性](#)」を参照してください。

また、JBoss EAP 管理コンソールでサーバー JVM 設定を作成編集するには、**Runtime** → **Hosts** と選択し、ホストを選択して **表示** をサーバー上でクリックし、**JVMs** タブを選択します。

各サーバーのこれらの設定は **host.xml** に保存されます。

### 9.3. JVM 状態の表示

ヒープおよびスレッドの使用状況など、スタンドアロンまたは管理対象ドメインサーバーの JVM リソースの状態は管理コンソールから表示できます。統計はリアルタイムでは表示されませんが、**Refresh** をクリックすると JVM リソースの最新の概要を表示できます。

スタンドアロン JBoss EAP サーバーの JVM の状態を表示するには、以下を行います。

- **Runtime** タブに移動し、サーバーを選択して、**状態** を選択します。

管理対象ドメインでの JBoss EAP サーバーの JVM の状態を表示するには、以下を行います。

- **Runtime** → **Hosts** と選択し、ホストとサーバーを選択して **状態** を選択します。

以下のヒープ使用情報が表示されます。

#### Max

メモリー管理に使用できるメモリーの最大量。

#### Used

使用されたメモリーの量。

#### Committed

JVM が使用するために確保されたメモリー量。

JVM のアップタイムやスレッドの使用状況などの他の情報も表示されます。

### 9.4. JVM の調整

JVM のパフォーマンスを最適化するための情報は、『[パフォーマンスチューニングガイド](#)』の「**JVM の調整**」を参照してください。

## 第10章 MAIL サブシステム

### 10.1. MAIL サブシステムの設定

mail サブシステムを使用すると、JBoss EAP でメールセッションを設定でき、JNDI を使用してこれらのセッションをアプリケーションにインジェクトできます。また、Java EE の `@MailSessionDefinition` および `@MailSessionDefinitions` アノテーションを使用する設定もサポートします。

#### アプリケーションで使用する SMTP サーバーの設定

1. 以下の CLI コマンドを使用して SMTP サーバーとアウトバウンドソケットバインディングを設定します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-smtp:add(host=localhost, port=25)
```

```
/subsystem=mail/mail-session=mySession:add(jndi-name=java:jboss/mail/MySession)
```

```
/subsystem=mail/mail-session=mySession/server=smtp:add(outbound-socket-binding-ref=my-smtp, username=user, password=pass, tls=true)
```

2. アプリケーション内で設定されたメールセッションを呼び出します。

```
@Resource(lookup="java:jboss/mail/MySession")  
private Session session;
```

メールセッションおよびサーバーの設定で使用できる属性の完全リストは「[Mail サブシステムの属性](#)」を参照してください。

### 10.2. カスタムトランスポートの設定

POP3 や IMAP などの標準のメールサーバーを使用している場合、メールサーバーには定義できる属性のセットがあります。これらの属性の一部は必須です。最も重要な属性はアウトバウンドメールソケットバインディングの参照である **outbound-socket-binding-ref** で、ホストアドレスとポート番号で定義されます。

**outbound-socket-binding-ref** の定義は、負荷分散の目的でホスト設定に複数のホストを使用するユーザーにとっては最も効率的なソリューションではない場合があります。標準の JavaMail は負荷分散のために複数のホストを使用するホスト設定をサポートしません。そのため、複数のホストを使用するこの設定を持つユーザーはカスタムメールトランスポートを実装する必要があります。カスタムメールトランスポートは **outbound-socket-binding-ref** を必要とせず、カスタムのホストプロパティ形式を許可します。

カスタムのメールトランスポートは管理 CLI から設定できます。

1. 新しいメールセッションを追加し、JNDI 名を指定します。

```
/subsystem=mail/mail-session=mySession:add(jndi-name=java:jboss/mail/MySession)
```

2. アウトバウンドソケットバインディングを追加し、ホストとポートを指定します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-smtp-binding:add(host=localhost, port=25)
```

- SMTP サーバーを追加し、アウトバウンドソケットバインディング、ユーザー名、およびパスワードを指定します。

```
/subsystem=mail/mail-session=mySession/server=smtp:add(outbound-socket-binding-ref=my-smtp-binding, username=user, password=pass, tls=true)
```

### 注記

同様の手順で POP3 または IMAP サーバーを設定できます。

### POP3 サーバー

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-pop3-binding:add(host=localhost, port=110)
/subsystem=mail/mail-session=mySession/server=pop3:add(outbound-socket-binding-ref=my-pop3-binding, username=user, password=pass)
```

### IMAP サーバー

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=my-imap-binding:add(host=localhost, port=143)
/subsystem=mail/mail-session=mySession/server=imap:add(outbound-socket-binding-ref=my-imap-binding, username=user, password=pass)
```

カスタムサーバーを使用するには、アウトバウンドソケットバインディングを指定せずにカスタムメールサーバーを作成します。カスタムメールサーバーのプロパティ定義でホスト情報を指定できます。例を以下に示します。

```
/subsystem=mail/mail-session=mySession/custom=myCustomServer:add(username=user,password=pass, properties={"host" => "myhost", "my-property" => "value"})
```

カスタムプロトコルを定義する場合、ピリオド (.) が含まれるプロパティ名は完全修飾名と見なされ、直接渡されます。**my-property** など、他の形式は **mail.server-name.my-property** という形式に変換されます。

以下の XML は、カスタムサーバーが含まれるメール設定の例になります。

```
<subsystem xmlns="urn:jboss:domain:mail:3.0">
  <mail-session name="default" jndi-name="java:jboss/mail/Default">
    <smtp-server outbound-socket-binding-ref="mail-smtp"/>
  </mail-session>
  <mail-session name="myMail" from="user.name@domain.org" jndi-name="java:/Mail">
    <smtp-server password="password" username="user" tls="true" outbound-socket-binding-ref="mail-smtp"/>
    <pop3-server outbound-socket-binding-ref="mail-pop3"/>
    <imap-server password="password" username="nobody" outbound-socket-binding-ref="mail-
imap"/>
  </mail-session>
```



```

<mail-session name="custom" jndi-name="java:jboss/mail/Custom" debug="true">
  <custom-server name="smtp" password="password" username="username">
    <property name="host" value="mail.example.com"/>
  </custom-server>
</mail-session>
<mail-session name="custom2" jndi-name="java:jboss/mail/Custom2" debug="true">
  <custom-server name="pop3" outbound-socket-binding-ref="mail-pop3">
    <property name="custom-prop" value="some-custom-prop-value"/>
  </custom-server>
</mail-session>
</subsystem>

```

### 10.3. パスワードに認証情報ストアを使用

**mail** サブシステムでクリアテキストのパスワードを提供する他に、認証情報ストアを使用してパスワードを提供することもできます。**elytron** サブシステムを使用すると、認証情報ストアを作成してパスワードをセキュアに保存し、JBoss EAP 全体でパスワードを使用することができます。認証情報ストアの作成および使用に関する詳細は、『[How to Configure Server Security](#)』の「**Credential Store**」を参照してください。

#### 管理 CLI を使用した認証情報ストアの使用

```

/subsystem=mail/mail-session=mySession/server=smtp:add(outbound-socket-binding-ref=my-smtp-binding, username=user, credential-reference={store=exampleCS, alias=mail-session-pw}, tls=true)

```

#### 注記

以下は、クリアテキストパスワードを使用する **credential-reference** 属性を指定する方法の例になります。

```
credential-reference={clear-text="MASK-Ewcyuqd/nP9;A1B2C3D4;351"}
```

#### 管理コンソールを使用した認証情報ストアの使用

1. 管理コンソールにアクセスします。詳細は「[管理コンソール](#)」を参照してください。
2. **Configuration** → **Subsystems** → **Mail** と選択します。
3. 該当するメールセッションを選択し、**表示** をクリックします。
4. **Server** を選択し、該当するメールセッションサーバーを選択します。**Credential Reference** タブで認証情報リファレンスを設定でき、**Attributes** タブでその他の属性を編集できます。

## 第11章 JBOSS EAP を用いたロギング

JBoss EAP は、EAP での内部使用とデプロイされたアプリケーションによる使用の両方で設定可能なロギング機能を提供します。**logging** サブシステムは JBoss LogManager を基盤とし、JBoss Logging だけでなくサードパーティーアプリケーションのロギングフレームワークを複数サポートします。

### 11.1. サーバーロギング

#### 11.1.1. サーバーロギング

デフォルトでは、すべての JBoss EAP ログエントリは **server.log** ファイルに書き込みされます。このファイルの場所は操作するモードによって異なります。

- スタンドアロンサーバーの場合: **EAP\_HOME/standalone/log/server.log**
- 管理対象ドメインの場合: **EAP\_HOME/domain/servers/SERVER\_NAME/log/server.log**

このファイルはサーバーログとも呼ばれます。詳細は「[ルートロガー](#)」を参照してください。

#### 11.1.2. 起動時のロギング

起動中に JBoss EAP は Java 環境と各サービスの起動に関する情報をログに記録します。このログは、トラブルシューティングに役に立ちます。デフォルトでは、すべてのログエントリが[サーバーログ](#)に書き込まれます。

起動時のロギング設定は **logging.properties** 設定ファイルに指定されます。これは、JBoss EAP **logging** が開始され、継承するまでアクティブになります。このファイルの場所は操作するモードによって異なります。

- スタンドアロンサーバーの場合: **EAP\_HOME/standalone/configuration/logging.properties**
- 管理対象ドメインの場合:  
ドメインコントローラーおよびサーバーごとに1つの **logging.properties** ファイルがあります。
  - ドメインコントローラー: **EAP\_HOME/domain/configuration/logging.properties**
  - サーバー: **EAP\_HOME/domain/servers/SERVER\_NAME/data/logging.properties**



#### 警告

**logging.properties** ファイルは、直接編集する必要があるユースケース以外では直接編集しないことが推奨されます。直接編集する前に、[Red Hat カスタマーポータル](#)でサポートケースを作成することが推奨されます。

**logging.properties** ファイルに手動で行った変更は起動時に上書きされます。

##### 11.1.2.1. 起動エラーの表示

JBoss EAP をトラブルシューティングする場合、最初に行うべきことの1つは、起動時に発生したエラーをチェックすることです。提供された情報を使用して原因を診断し、解決します。起動時のエラーをトラブルシューティングする際にサポートが必要な場合はサポートケースを作成してください。

起動時のエラーを表示する方法は2つあり、どちらも利点があります。1つは **server.log** ファイルを確認する方法で、もう1つは **read-boot-errors** 管理 CLI コマンドを使用してブートエラーを確認する方法になります。

サーバーログファイルの確認

**server.log** ファイルを開いて起動中に発生したエラーを確認します。

この方法では、各エラーメッセージおよび関連するメッセージを確認でき、エラーが発生した理由の詳細を知ることができます。また、エラーメッセージをプレーンテキスト形式で表示することもできます。

1. ファイルビューアーで **server.log** を開きます。
2. ファイルの最後に移動します。
3. 最後の起動シーケンスの開始を示す **WFLYSRV0049** メッセージ ID を後方検索します。
4. ログのその位置から **ERROR** を前方検索します。各検索一致箇所には、エラーの説明が示され、関連するモジュールがリストされます。

以下は、**server.log** ログファイルのエラー説明の例です。

```
2016-03-16 14:32:01,627 ERROR [org.jboss.msc.service.fail] (MSC service thread 1-7) MSC000001:
Failed to start service jboss.undertow.listener.default: org.jboss.msc.service.StartException in service
jboss.undertow.listener.default: Could not start http listener
    at org.wildfly.extension.undertow.ListenerService.start(ListenerService.java:142)
    at
org.jboss.msc.service.ServiceControllerImpl$StartTask.startService(ServiceControllerImpl.java:1948)
    at org.jboss.msc.service.ServiceControllerImpl$StartTask.run(ServiceControllerImpl.java:1881)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
    at java.lang.Thread.run(Thread.java:745)
Caused by: java.net.BindException: Address already in use
...
```

管理 CLI からのブートエラーの読み取り

サーバーが起動しても起動中にエラーが報告された場合、**read-boot-errors** 管理 CLI コマンドを使用してエラーを確認できます。

この方法では、サーバーのファイルシステムにアクセスする必要がありません。したがって、エラーを監視する担当者がファイルシステムアクセスを持っていない場合に役に立ちます。これは CLI コマンドであるため、スクリプトで使用できます。たとえば、複数の JBoss EAP インスタンスを起動し、起動時に発生したエラーをチェックするスクリプトを記述できます。

次の管理 CLI コマンドを実行します。

```
/core-service=management:read-boot-errors
```

起動中に発生したすべてのエラーがリストされます。

```
{
  "outcome" => "success",
```

```

"result" => [
  {
    "failed-operation" => {
      "operation" => "add",
      "address" => [
        ("subsystem" => "undertow"),
        ("server" => "default-server"),
        ("http-listener" => "default")
      ]
    },
    "failure-description" => "{\"WFLYCTL0080: Failed services\" =>
    {\"jboss.undertow.listener.default\" => \"org.jboss.msc.service.StartException in service
    jboss.undertow.listener.default: Could not start http listener
    Caused by: java.net.BindException: Address already in use\"}},
    "failed-services" => {\"jboss.undertow.listener.default\" =>
    \"org.jboss.msc.service.StartException in service jboss.undertow.listener.default: Could not start http
    listener
    Caused by: java.net.BindException: Address already in use\"}
  }
  ...
]
}

```

### 11.1.3. ガベッジコレクションのロギング

ガベッジコレクションロギングは、すべてのガベッジコレクションのアクティビティをプレーンテキストのログファイルに記録します。これらのログファイルは診断を行うのに便利です。ガベッジコレクションロギングは、IBM の Java Development Kit を除くすべてのサポート対象設定の JBoss EAP スタンドアロンサーバーではデフォルトで有効になっています。

ガベッジコレクションログの場所は **EAP\_HOME/standalone/log/gc.log.DIGIT.current** です。ガベッジコレクションのログは 3 MB ずつに制限され、最大 5 つのファイルがローテーションされます。

トラブルシューティングに便利で、オーバーヘッドは最低限であるため、ガベッジコレクションのロギングを有効にすることが強く推奨されます。しかし、スタンドアロンサーバーのガベッジコレクションのロギングを無効にする場合は、サーバーを起動する前に **GC\_LOG** 変数を **false** に設定します。例を以下に示します。

```

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

```

### 11.1.4. デフォルトのログファイルの場所

以下のログファイルは、デフォルトのロギング設定に対して作成されます。デフォルトの設定では、periodic ログハンドラーを使用してサーバーログファイルが書き込まれます。

表11.1 スタンドアロンサーバーのデフォルトログファイル

ログファイル	説明
EAP_HOME/standalone/log/server.log	サーバー起動メッセージを含むサーバーログメッセージが含まれます。

ログファイル	説明
EAP_HOME/standalone/log/gc.log.DIGIT.current	ガベージコレクションの詳細が含まれます。

表11.2 管理対象ドメイン用のデフォルトログファイル

ログファイル	説明
EAP_HOME/domain/log/host-controller.log	ホストコントローラーの起動に関連するログメッセージが含まれます。
EAP_HOME/domain/log/process-controller.log	プロセスコントローラーの起動に関連するログメッセージが含まれます。
EAP_HOME/domain/servers/SERVER_NAME/log/server.log	サーバー起動メッセージを含む、名前付きサーバーのログメッセージが含まれます。

### 11.1.5. サーバーのデフォルトロケールの設定

JVM プロパティを適切な起動設定ファイルに設定すると、デフォルトのロケールを設定できます。起動設定ファイルは、スタンドアロンサーバーの場合は **EAP\_HOME/bin/standalone.conf**、管理対象ドメインの場合は **EAP\_HOME/bin/domain.conf** になります。



#### 注記

Windows Server の場合、JBoss EAP 起動設定ファイルは **standalone.conf.bat** と **domain.conf.bat** になります。

国際化または現地化されたログメッセージはこのデフォルトロケールを使用します。JBoss EAP 『開発ガイド』の[国際化されたログメッセージの作成](#)に関する情報を参照してください。

#### 言語の設定

言語を指定するには、**JAVA\_OPTS** 変数を使用して **user.language** プロパティを設定します。たとえば、以下の行を起動設定ファイルに追加し、フランス語のロケールを設定します。

```
JAVA_OPTS="$JAVA_OPTS -Duser.language=fr"
```

国際化および現地化されたログメッセージがフランス語で出力されるようになります。

#### 言語および国の設定

言語の他に、**user.country** プロパティを設定して国を指定することもできます。たとえば、以下の行を起動設定ファイルに追加すると、ポルトガル語のロケールがブラジルに設定されます。

```
JAVA_OPTS="$JAVA_OPTS -Duser.language=pt -Duser.country=BR"
```

国際化および現地化されたログメッセージがブラジルポルトガル語で出力されるようになります。

**org.jboss.logging.locale** プロパティを使用したサーバーロケールの設定

**org.jboss.logging.locale** プロパティを使用すると、Boss EAP からのメッセージや JBoss EAP が所有する依存関係からのメッセージなど、JBoss Logging を使用してログに記録されたメッセージのロケールを上書きできます。JSF などの他の依存関係ではロケールを上書きできません。

JBoss EAP サーバーをシステムデフォルト以外のロケールで起動するには、操作モードに応じて **EAP\_HOME/bin/standalone.conf** または **EAP\_HOME/bin/domain.conf** ファイルを編集し、以下のコマンドを追加して必要なロケールの JVM パラメーターを設定します。プロパティの値は BCP 47 形式で指定する必要があります。たとえば、ブラジルポルトガル語を設定する場合は「pt-BR」を使用します。

```
JAVA_OPTS="$JAVA_OPTS -Dorg.jboss.logging.locale=pt-BR"
```

## 11.2. ログファイルの表示

サーバーやアプリケーションログの確認は、診断エラー、パフォーマンスの問題、およびその他の問題に対応するために重要です。サーバーのファイルシステムで直接ログを表示したいユーザーもいます。直接ファイルシステムにアクセスできないユーザーやグラフィカルインターフェースを使用したいユーザーは JBoss EAP の管理コンソールからログを表示することができます。また、管理 CLI を使用してログを表示することもできます。

管理インターフェースの1つからログにアクセスするには、サーバーの **jboss.server.log.dir** プロパティによって指定されたディレクトリに存在し、File、Periodic rotating、Size rotating、または Periodic size rotating ログハンドラーとして定義される必要があります。RBAC ロール割り当ても考慮されるため、管理コンソールまたは管理 CLI にログインしたユーザーはアクセス権限を持つログのみ表示できます。

### 管理コンソールからのログの表示

管理コンソールから直接ログを表示できます。

1. **Runtime** タブを選択し、該当するサーバーを選択します。
2. **ログファイル** を選択し、リストからログファイルを選択します。
3. **表示** をクリックしてログの内容を表示および検索するか、ドロップダウンリストから **ダウンロード** を選択してログファイルをローカルファイルシステムにダウンロードします。



#### 警告

管理コンソールのログビューアーは、100MB 以上のログファイルなどの非常に大きなログファイルを表示するテキストエディターの代わりとして使用するものではありません。15MB を超えるログファイルを開こうとすると、確認を求められます。管理コンソールで非常に大きなファイルを開くと、ブラウザーがクラッシュする可能性があるため、大きなログファイルは常にローカルにダウンロードし、テキストエディターで開くようにしてください。

### 管理 CLI からのログの表示

**read-log-file** コマンドを使用すると管理 CLI からログファイルの内容を取得できます。デフォルトでは、指定されたログファイルの最後の **10** 行が表示されます。

```
/subsystem=logging/log-file=LOG_FILE_NAME:read-log-file
```



## 注記

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` を追加してください。

以下のパラメーターを使用するとログの出力をカスタマイズできます。

### encoding

ファイルの読み取りに使用される文字エンコーディング。

### lines

ファイルから読み取る行数。-1 はログの行すべてを取得します。デフォルトは **10** です。

### skip

読み取る前にスキップする行数。デフォルトは **0** です。

### tail

ファイルの最後から読み取るかどうか。デフォルト値は **true** です。

たとえば、以下の管理 CLI コマンドは `server.log` ログファイルの最初の **5** 行を読み取ります。

```
/subsystem=logging/log-file=server.log:read-log-file(lines=5,tail=false)
```

このコマンドを実行すると以下が出力されます。

```
{
  "outcome" => "success",
  "result" => [
    "2016-03-24 08:49:26,612 INFO [org.jboss.modules] (main) JBoss Modules version 1.5.1.Final-redhat-1",
    "2016-03-24 08:49:26,788 INFO [org.jboss.msc] (main) JBoss MSC version 1.2.6.Final-redhat-1",
    "2016-03-24 08:49:26,863 INFO [org.jboss.as] (MSC service thread 1-7) WFLYSRV0049: JBoss EAP 7.0.0.GA (WildFly Core 2.0.13.Final-redhat-1) starting",
    "2016-03-24 08:49:27,973 INFO [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0039: Creating http management service using socket-binding (management-http)",
    "2016-03-24 08:49:27,994 INFO [org.xnio] (MSC service thread 1-1) XNIO version 3.3.4.Final-redhat-1"
  ]
}
```

## 11.3. LOGGING サブシステム

JBoss EAP の **logging** サブシステムは **ログカテゴリー** および **ログハンドラー** のシステムを使用して設定されます。ログカテゴリーはキャプチャーするメッセージを定義します。ログハンドラーは、ディスクへの書き込みやコンソールへの送信など、これらのメッセージの対応方法を定義します。

**ロギングプロファイル** は、一意な名前が付けられたロギング設定の作成や、他のロギング設定に関係しないアプリケーションへの割り当てを可能にします。ロギングプロファイルの設定は、メインの **logging** サブシステムとほぼ同じです。

### 11.3.1. ルートロガー

JBoss EAP のルートロガーは、ログカテゴリーによってキャプチャーされないサーバーへ送信されたすべてのログメッセージ (指定のログレベル以上) をキャプチャーします。

デフォルトでは、ルートロガーはコンソールおよび周期ログハンドラーを使用するように設定されます。周期ログハンドラーは、**server.log** ファイルへ書き込むよう設定されます。このファイルはサーバーログとも呼ばれます。

詳細は「[ルートロガーの設定](#)」を参照してください。

### 11.3.2. ログカテゴリー

ログカテゴリーは、キャプチャーするログメッセージのセットと、メッセージを処理する1つ以上のログハンドラーを定義します。

キャプチャーするログメッセージは、指定された元の Java パッケージとログレベルによって定義されます。そのパッケージのクラスおよびそのログレベル以上のメッセージがログカテゴリーによってキャプチャーされ、指定のログハンドラーに送信されます。



#### 注記

通常、ログカテゴリーは Java パッケージとクラス名ですが、**Logger.getLogger(LOGGER\_NAME)** メソッドによって指定された名前をすべて使用できます。

ログカテゴリーは、独自のハンドラーの代わりにルートロガーのログハンドラーを任意で使用することができます。

詳細は「[ログカテゴリーの設定](#)」を参照してください。

### 11.3.3. ログハンドラー

ログハンドラーはキャプチャーしたメッセージの記録方法を定義します。使用できるログハンドラーの種類は **console**、**file**、**periodic**、**size**、**periodic size**、**syslog**、**custom**、および **async** です。



#### 注記

ログハンドラーは、アクティブにするために少なくとも1つのロガーに追加する必要があります。

#### ログハンドラーの種類

##### Console

Console ログハンドラーは、ログメッセージをホストオペレーティングシステムの標準出力 (**stdout**) または標準エラー (**stderr**) ストリームに書き込みます。これらのメッセージは、JBoss EAP がコマンドラインプロンプトから実行された場合に表示されます。オペレーティングシステムで標準出力または標準エラー streams をキャプチャーするように設定されていない限り、Console ログハンドラーからのメッセージは保存されません。

##### File

File ログハンドラーは、ログメッセージを指定のファイルに書き込みます。

##### Periodic

Periodic ログハンドラーは、指定した時間が経過するまで、ログメッセージを指定ファイルに書き込みます。その時間が経過した後、指定のタイムスタンプが追記されてファイルの名前が変更され、ハンドラーは元の名前で新規作成されたログファイルに書き込みを続けます。



## Size

Size ログハンドラーは、指定したファイルが指定したサイズに達するまで、そのファイルにログメッセージを書き込みます。ファイルが指定したサイズに達すると、数値の接尾辞が付いて名前が変更され、ハンドラーは元の名前で新規作成されたログファイルに書き込みを続けます。各サイズログハンドラーは、この方式で保管されるファイルの最大数を指定する必要があります。

## Periodic Size

Periodic Size ログハンドラーは、ファイルが指定のサイズに達するまで、または指定の期間が経過するまで、ログメッセージを名前の付いたファイルに書き込みます。その後、ファイルの名前が変更され、ハンドラーは元の名前で新規作成されたログファイルに書き込みを続けます。これは Periodic と Size ログハンドラーの組み合わせで、組み合わせられた属性をサポートします。

## Syslog

syslog ハンドラーは、リモートのロギングサーバーへメッセージを送信するために使用できます。これにより、複数のアプリケーションが同じサーバーにログメッセージを送信でき、そのサーバーですべてのログメッセージを解析できます。

## Socket

ソケットログハンドラーを使用して、ログメッセージをソケット上でリモートロギングサーバーに送信できます。ソケットは TCP または UDP ソケットになります。

## Custom

カスタムログハンドラーにより、実装された新たなタイプのログハンドラーを設定することができます。カスタムハンドラーは、**java.util.logging.Handler** を拡張する Java クラスとして実装し、モジュール内に格納する必要があります。Log4J アペンダーをカスタムログハンドラーとして使用することもできます。

## Async

Async ログハンドラーは、単一または複数のログハンドラーを対象とする非同期動作を提供するラッパーログハンドラーです。Async ログハンドラーは、待ち時間が長かったり、ネットワークファイルシステムへのログファイルの書き込みなどにパフォーマンス上の問題があるログハンドラーに対して有効です。

各ログハンドラーの設定に関する詳細は、「[ログハンドラーの設定](#)」の項を参照してください。

### 11.3.4. ログレベル

ログレベルとは、ログメッセージの性質と重大度を示す列挙値です。特定のログメッセージのレベルは、そのメッセージを送信するために選択したロギングフレームワークの適切なメソッドを使用して開発者が指定できます。

JBoss EAP は、サポートされるアプリケーションロギングフレームワークによって使用されるすべてのログレベルをサポートします。最も一般的に使用されるログレベルは、ログレベルの低い順に **TRACE**、**DEBUG**、**INFO**、**WARN**、**ERROR** および **FATAL** となります。

ログレベルはログカテゴリとログハンドラーによって使用され、それらが担当するメッセージを限定します。各ログレベルには、他のログレベルに対して相対的な順番を示す数値が割り当てられています。ログカテゴリとハンドラーにはログレベルが割り当てられ、そのレベル以上のログメッセージのみを処理します。たとえば、**WARN** レベルのログハンドラーは、**WARN**、**ERROR**、および **FATAL** のレベルのメッセージのみを記録します。

#### サポート対象のログレベル

ログのレベル	Value	説明
--------	-------	----

ログのレベル	Value	説明
ALL	Integer.MIN_VALUE	すべてのログメッセージを提供します。
FINEST	300	-
FINER	400	-
TRACE	400	<b>TRACE</b> レベルのログメッセージは、実行中のアプリケーションの状態に関する詳細な情報を提供し、デバッグ時のみキャプチャーされます。
DEBUG	500	<b>DEBUG</b> レベルのログメッセージは、個々の要求またはアプリケーションアクティビティの進捗状況を示し、通常はデバッグ時のみキャプチャーされます。
FINE	500	-
CONFIG	700	-
INFO	800	<b>INFO</b> レベルのログメッセージはアプリケーションの全体的な進捗状況を示します。多くの場合、アプリケーションの起動、シャットダウン、およびその他の主要なライフサイクルイベントに使用されます。
WARN	900	<b>WARN</b> レベルのログメッセージはエラーではないが、理想的とは見なされない状況を示します。 <b>WARN</b> ログメッセージは将来エラーが発生する可能性のある状況を示すことがあります。
WARNING	900	-
ERROR	1000	<b>ERROR</b> レベルのログメッセージは、現在のアクティビティや要求の完了を妨げる可能性があるが、アプリケーションの実行を妨げない発生済みエラーを示します。
SEVERE	1000	-
FATAL	1100	<b>FATAL</b> レベルのログメッセージは重大なサービス障害やアプリケーションのシャットダウンを引き起こしたり、JBoss EAP のシャットダウンを引き起こしたりする可能性があるイベントを示します。
OFF	Integer.MAX_VALUE	ログメッセージを表示しません。



## 注記

**ALL** は、最低ログレベルであり、すべてのログレベルのメッセージを含みます。ログインの量は最も多くなります。

**FATAL** は、最大ログレベルであり、そのレベルのメッセージのみを含みます。ログインの量は最も少なくなります。

### 11.3.5. ログフォーマッター

フォーマッターはログメッセージのフォーマットに使用されます。**named-formatter** 属性を使用するとフォーマッターをログハンドラーに割り当てできます。ログハンドラー設定の詳細は「[ログハンドラーの設定](#)」を参照してください。

logging サブシステムには 4 種類のフォーマッターが含まれます。

- [パターンフォーマッター](#)
- [JSON フォーマッター](#)
- [XML フォーマッター](#)
- [カスタムフォーマッター](#)

#### パターンフォーマッター

パターンフォーマッターは、ログメッセージをプレーンテキストでフォーマットするために使用されます。ログハンドラーの **named-formatter** 属性としてフォーマットを使用する他に、あらかじめフォーマッターリソースを作成する必要なく **formatter** 属性として使用することもできます。パターン構文に関する詳細は「[パターンフォーマッターのフォーマット文字](#)」を参照してください。

パターンフォーマッターの設定方法は「[パターンフォーマッターの設定](#)」を参照してください。

#### JSON フォーマッター

JSON フォーマッターは、ログメッセージを JSON 形式でフォーマットするために使用されます。

JSON フォーマッターの設定方法は「[JSON フォーマッターの設定](#)」を参照してください。

#### XML フォーマッター

XML ログフォーマッターは、ログメッセージを XML 形式でフォーマットするために使用されます。

XML フォーマッターの設定方法は「[XML フォーマッターの設定](#)」を参照してください。

#### カスタムフォーマッター

ハンドラーと使用するカスタムフォーマッターです。ほとんどのログレコードは printf 形式でフォーマットされることに注意してください。メッセージを適切にフォーマットするには、**org.jboss.logmanager.ExtLogRecord#getFormattedMessage()** の呼び出しが必要になることがあります。

カスタムログフォーマッターの設定方法は「[カスタムログフォーマッターの設定](#)」を参照してください。

### 11.3.6. フィルター式

フィルター式は **filter-spec** 属性を使用して設定され、さまざまな基準に基づいてログメッセージを記録するために使用されます。フィルターチェックは、常に未フォーマットの raw メッセージに対して行われます。ロガーまたはハンドラーのフィルターを含めることができますが、ハンドラーに配置されたフィルターよりもロガーフィルターが優先されます。



## 注記

ルートロガーに対して指定された **filter-spec** は他のロガーによって継承されません。ハンドラーごとに **filter-spec** を指定する必要があります。

表11.3 ログイングのフィルター式

フィルター式	説明
accept	すべてのログメッセージを許可します。
deny	すべてのログメッセージを拒否します。
not[filter expression]	単一のフィルター式の逆の値を返します。例を以下に示します。 <b>not(match("WFLY"))</b>
all[filter expression]	フィルター式のカンマ区切りリストから連結された値を返します。例を以下に示します。 <b>all(match("WFLY"),match("WELD"))</b>
any[filter expression]	フィルター式のカンマ区切りリストから1つの値を返します。例を以下に示します。 <b>any(match("WFLY"),match("WELD"))</b>
levelChange[level]	指定のレベルでログレコードを更新します。例を以下に示します。 <b>levelChange(WARN)</b>
levels[levels]	レベルのカンマ区切りリストにあるレベルの1つでログメッセージをフィルターします。例を以下に示します。 <b>levels(DEBUG,INFO,WARN,ERROR)</b>
levelRange[minLevel,maxLevel]	指定されたレベル範囲内でログメッセージをフィルターします。【および】は、含まれるレベルを示すために使用されます。(および)は除外されるレベルを示すために使用されます。例を以下に示します。 <ul style="list-style-type: none"> <li>● <b>levelRange[INFO,ERROR]</b> <ul style="list-style-type: none"> <li>○ 最小レベルは <b>INFO</b> 以上で、最大レベルは <b>ERROR</b> 以下でなければなりません。</li> </ul> </li> <li>● <b>levelRange[DEBUG,ERROR]</b> <ul style="list-style-type: none"> <li>○ 最小レベルは <b>DEBUG</b> 以上で、最大レベルは <b>ERROR</b> 未満でなければなりません。</li> </ul> </li> </ul>

フィルター式	説明
match["pattern"]	提供される正規表現を使用してログメッセージをフィルターします。例を以下に示します。  <b>match("WFLY\d+")</b>
substitute["pattern","replacement value"]	最初にパターン (最初の引数) と一致した値を代替テキスト (2 番目の引数) に置き換えるフィルター。例を以下に示します。  <b>substitute("WFLY","EAP")</b>
substituteAll["pattern","replacement value"]	パターン (最初の引数) と一致したすべての値を代替テキスト (2 番目の引数) に置き換えるフィルター。例を以下に示します。  <b>substituteAll("WFLY","EAP")</b>

### 注記

管理 CLI を使用してフィルター式を設定する場合、値が文字列として正しく処理されるよう、フィルターテキストのコンマと引用符を必ずエスケープしてください。コンマと引用符の前にバックスラッシュ (\) を付け、式全体を引用符で囲む必要があります。以下は **substituteAll("WFLY","YLFW")** を適切にエスケープした例になります。

```
/subsystem=logging/console-handler=CONSOLE:write-attribute(name=filter-spec, value="substituteAll(\"WFLY\\\", \"YLFW\\\")")
```

### 11.3.7. 暗黙的なログインの依存関係

JBoss EAP の **logging** サブシステムはデフォルトで暗黙的なログイン API 依存関係をデプロイメントに追加します。**add-logging-api-dependencies** 属性を使用すると、この暗黙的な依存関係をデプロイメントに追加するかどうかを制御できます。この属性はデフォルトでは **true** に設定されています。

管理 CLI を使用して **add-logging-api-dependencies** 属性を **false** に設定すると、暗黙的なログイン API 依存関係がデプロイメントに追加されないようになります。

```
/subsystem=logging:write-attribute(name=add-logging-api-dependencies, value=false)
```

**logging** サブシステムの暗黙的な依存関係については、JBoss EAP 『[開発ガイド](#)』の「暗黙的なモジュール依存関係」の項を参照してください。

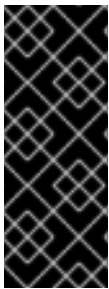
## 11.4. ログカテゴリーの設定

ここでは、管理 CLI を使用してログカテゴリーを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Logging** → **Configuration** と選択し、**表示** をクリックして **Categories** を選択するとログカテゴリーを設定することができます。

ログカテゴリーを設定するために実行する主なタスクは次のとおりです。

- [新しいログカテゴリーの追加](#)。

- ログカテゴリーの設定。
- ログハンドラーのログカテゴリーへの割り当て。



### 重要

ロギングプロファイルにログカテゴリーを設定する場合、コマンドの最初は `/subsystem=logging/` ではなく `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に `/profile=PROFILE_NAME` を付けます。

### ログカテゴリーの追加

ログカテゴリー名は、元の Java パッケージによって定義されます。ログレベルなどのその他の設定に準拠する限り、そのパッケージのクラスからのメッセージはキャプチャーされます。

```
/subsystem=logging/logger=LOG_CATEGORY:add
```

### ログカテゴリーの設定

必要性に応じて、以下のログカテゴリー属性を1つ以上設定する必要がある場合があります。利用できるログカテゴリー属性の完全リストと説明は、「[ログカテゴリーの属性](#)」を参照してください。

- ログレベルを設定します。  
ログカテゴリーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/logger=LOG_CATEGORY:write-attribute(name=level,value=LEVEL)
```

- このカテゴリーがルートロガーのログハンドラーを使用するかどうかを設定します。  
デフォルトでは、ログカテゴリーは独自のハンドラーと、ルートロガーのハンドラーを使用します。ログカテゴリーが割り当てられたハンドラーのみを使用する必要がある場合は **use-parent-handlers** 属性を **false** に設定します。

```
/subsystem=logging/logger=LOG_CATEGORY:write-attribute(name=use-parent-handlers,value=USE_PARENT_HANDLERS)
```

- フィルター式を設定します。  
ログカテゴリーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER\_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\\"WFLY\\"))"** に置き換える必要があります。

```
/subsystem=logging/logger=LOG_CATEGORY:write-attribute(name=filter-spec,value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は「[フィルター式](#)」の項を参照してください。

### ハンドラーの割り当て

ログハンドラーをログカテゴリーに割り当てます。

```
/subsystem=logging/logger=LOG_CATEGORY:add-handler(name=LOG_HANDLER_NAME)
```

## ログカテゴリーの削除

ログカテゴリーは **remove** 操作で削除できます。

```
/subsystem=logging/logger=LOG_CATEGORY:remove
```

## 11.5. ログハンドラーの設定

ログハンドラーはキャプチャーしたメッセージの記録方法を定義します。以下の項を参照して必要なログハンドラーのタイプを設定してください。

- [Console ログハンドラー](#)
- [File ログハンドラー](#)
- [Periodic rotating ログハンドラー](#)
- [Size rotating ログハンドラー](#)
- [Periodic size rotating ログハンドラー](#)
- [Syslog ハンドラー](#)
- [File ログハンドラー](#)
- [Custom ログハンドラー](#)
- [Async ログハンドラー](#)

### 11.5.1. Console ログハンドラーの設定

ここでは、管理 CLI を使用して Console ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Logging** → **Configuration** と選択し、**表示** をクリックして **Handler** → **Console Handler** と選択すると Console ログハンドラーを設定することができます。

Console ログハンドラーを設定するために実行する主なタスクは次のとおりです。

- [新しい Console ログハンドラーの追加。](#)
- [Console ログハンドラーの設定。](#)
- [Console ログハンドラーのロガーへの割り当て。](#)



#### 重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING\_PROFILE\_NAME/** になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に **/profile=PROFILE\_NAME** を付けます。

Console ログハンドラーの追加

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:add
```

## Console ログハンドラーの設定

必要性に応じて、以下の Console ログハンドラー属性を1つ以上設定する必要がある場合があります。利用できる Console ログハンドラー属性の完全リストと説明は、「[Console ログハンドラーの属性](#)」を参照してください。

- ログレベルを設定します。  
ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- ターゲットを設定します。  
ハンドラーのターゲットを設定します。値は **System.out**、**System.err**、**console** のいずれかになります。デフォルト **System.out** です。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=target,value=TARGET)
```

- エンコーディングを設定します。  
ハンドラーのエンコーディングを設定します (例: **utf-8**)。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
```

- ログフォーマッターを設定します。  
ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%n** です。**FORMAT** の値は必ず引用符で囲んでください。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
```



### 注記

保存されたフォーマッターを参照する場合は **named-formatter** 属性を使用します。

- 自動フラッシュを設定します。  
毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- フィルター式を設定します。  
ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER\_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\\"WFLY\\"))"** に置き換える必要があります。



```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は「[フィルター式](#)」の項を参照してください。

#### Console ログハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てする必要があります。

以下の管理 CLI コマンドは Console ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=CONSOLE_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Console ログハンドラーを割り当てます。

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

#### Console ログハンドラーの削除

ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/console-handler=CONSOLE_HANDLER_NAME:remove
```

## 11.5.2. File ログハンドラーの設定

ここでは、管理 CLI を使用して File ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Logging** → **Configuration** と選択し、**表示** をクリックして **Handler** → **File Handler** と選択すると、File ログハンドラーを設定することができます。

File ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- [新しい File ログハンドラーの追加](#)。
- [File ログハンドラーの設定](#)。
- [File ログハンドラーのロガーへの割り当て](#)。

### 重要

ログインプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING\_PROFILE\_NAME/** になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に **/profile=PROFILE\_NAME** を付けます。

#### File ログハンドラーの追加

File ログハンドラーを追加する場合、**path** および **relative-to** 属性で構成される **file** 属性を使用してファイルパスを指定する必要があります。**path** 属性を使用して名前を含むログのファイルパスを設定します (例: **my-log.log**)。オプションで **relative-to** 属性を使用すると **path** が名前付きのパスと相対的になるよう設定できます (例: **jboss.server.log.dir**)。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:add(file={path=FILE_PATH,relative-to=RELATIVE_TO_PATH})
```

### File ログハンドラーの設定

必要性に応じて、以下の File ログハンドラー属性を1つ以上設定する必要がある場合があります。利用できる File ログハンドラー属性の完全リストと説明は、「[File ログハンドラーの属性](#)」を参照してください。

- ログレベルを設定します。  
ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- 追加動作を設定します。  
デフォルトでは、サーバーが再起動されたときに JBoss EAP はログメッセージを同じファイルに追加します。サーバーの再起動時にファイルを上書きする場合は **append** 属性を **false** に設定します。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=append,value=APPEND)
```

- エンコーディングを設定します。  
ハンドラーのエンコーディングを設定します (例: **utf-8**)。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
```

- ログフォーマッターを設定します。  
ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%n** です。 **FORMAT** の値は必ず引用符で囲んでください。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
```



### 注記

保存されたフォーマッターを参照する場合は **named-formatter** 属性を使用します。

- 自動フラッシュを設定します。  
毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- フィルター式を設定します。  
ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な

**FILTER\_EXPRESSION** 変数では、`not(match("WFLY"))` のフィルター式を "`not(match(\"WFLY\"))`" に置き換える必要があります。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:write-attribute(name=filter-spec,
value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は「[フィルター式](#)」の項を参照してください。

**File** ログハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは File ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=FILE_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに File ログハンドラーを割り当てます。

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

**File** ログハンドラーの削除

ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/file-handler=FILE_HANDLER_NAME:remove
```

### 11.5.3. Periodic rotating ログハンドラーの設定

ここでは、管理 CLI を使用して Periodic rotating ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Logging** → **Configuration** と選択し、**表示** をクリックして **Handler** → **Periodic Handler** と選択すると、Periodic ログハンドラーを設定することができます。

Periodic ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- [新しい Periodic ログハンドラーの追加](#)。
- [Periodic ログハンドラーの設定](#)。
- [Periodic ログハンドラーのロガーへの割り当て](#)

#### 重要

ログインプロファイルにこのログハンドラーを設定する場合、コマンドの最初は `/subsystem=logging/` ではなく `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に `/profile=PROFILE_NAME` を付けます。

**Periodic** ログハンドラーの追加

Periodic ログハンドラーを追加する場合、**path** および **relative-to** 属性で構成される **file** 属性を使用してファイルパスを指定する必要があります。**path** 属性を使用して名前を含むログのファイルパスを設定します (例: `my-log.log`)。オプションで **relative-to** 属性を使用すると **path** が名前付きのパスと相対

的になるよう設定できます (例: `jboss.server.log.dir`)。

また、`suffix` 属性を使用してローテーションしたログの接尾辞を設定する必要もあります。これは、`.yyyy-MM-dd-HH` のように `java.text.SimpleDateFormat` が認識できる形式でなければなりません。これは `.yyyy-MM-dd-HH` のように `java.text.SimpleDateFormat` が認識できる形式でなければなりません。ローテーションの周期はこの接尾辞を基に自動的に算出されます。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:add(file={path=FILE_PATH,relative-to=RELATIVE_TO_PATH},suffix=SUFFIX)
```

### Periodic ログハンドラーの設定

必要性に応じて、以下の Periodic ログハンドラー属性を1つ以上設定する必要がある場合があります。利用できる Periodic ログハンドラー属性の完全リストと説明は、「[Periodic ログハンドラーの属性](#)」を参照してください。

- ログレベルを設定します。  
ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- 追加動作を設定します。  
デフォルトでは、サーバーが再起動されたときに JBoss EAP はログメッセージを同じファイルに追加します。サーバーの再起動時にファイルを上書きする場合は `append` 属性を **false** に設定します。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-attribute(name=append,value=APPEND)
```

- エンコーディングを設定します。  
ハンドラーのエンコーディングを設定します (例: `utf-8`)。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
```

- ログフォーマッターを設定します。  
ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は `%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n` です。 **FORMAT** の値は必ず引用符で囲んでください。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
```



### 注記

保存されたフォーマッターを参照する場合は `named-formatter` 属性を使用します。

- 自動フラッシュを設定します。  
毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- フィルター式を設定します。  
ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER\_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\\"WFLY\\"))"** に置き換える必要があります。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は「[フィルター式](#)」の項を参照してください。

#### Periodic ログハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Periodic ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=PERIODIC_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Periodic ログハンドラーを割り当てます。

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

#### Periodic ログハンドラーの削除

ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/periodic-rotating-file-handler=PERIODIC_HANDLER_NAME:remove
```

### 11.5.4. Size rotating ログハンドラーの設定

ここでは、管理 CLI を使用して Size rotating ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Logging** → **Configuration** と選択し、**表示** をクリックして **Handler** → **Size Handler** と選択すると、Size ログハンドラーを設定することができます。

Size ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- [新しい Size ログハンドラーの追加](#)。
- [Size ログハンドラーの設定](#)。
- [Size ログハンドラーのロガーへの割り当て](#)



## 重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は `/subsystem=logging/` ではなく `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に `/profile=PROFILE_NAME` を付けます。

### Size ログハンドラーの追加

Size ログハンドラーを追加する場合、`path` および `relative-to` 属性で構成される `file` 属性を使用してファイルパスを指定する必要があります。`path` 属性を使用して名前を含むログのファイルパスを設定します (例: `my-log.log`)。オプションで `relative-to` 属性を使用すると `path` が名前付きのパスと相対的になるよう設定できます (例: `jboss.server.log.dir`)。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:add(file={path=FILE_PATH,relative-to=RELATIVE_TO_PATH})
```

### Size ログハンドラーの設定

必要性に応じて、以下の Size ログハンドラー属性を1つ以上設定する必要がある場合があります。利用できる Size ログハンドラー属性の完全リストと説明は、「[Size ログハンドラーの属性](#)」を参照してください。

- ログレベルを設定します。  
ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- ローテーションされるログの接尾辞を設定します。  
接尾辞の文字列を設定します。これは `yyyy-MM-dd-HH` のように `java.text.SimpleDateFormat` が認識できる形式でなければなりません。ローテーションの周期はこの接尾辞を基に自動的に算出されます。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=suffix,value=SUFFIX)
```

- ローテーションサイズを設定します。  
ファイルの最大サイズを設定します。この値を超えるとファイルがローテーションされます。デフォルトは2メガバイトを意味する **2m** です。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=rotate-size,value=ROTATE_SIZE)
```

- 保持するバックアップログの最大数の設定  
保持するバックアップの数を設定します。デフォルト値は **1** です。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=max-backup-index,value=MAX_BACKUPS)
```

- 起動時にログをローテーションするかどうかを設定します。

デフォルトでは、サーバーの再起動時に新しいログファイルは作成されません。サーバーの再起動時にログをローテーションするには、**true** に設定します。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=rotate-on-boot, value=ROTATE_ON_BOOT)
```

- 追加動作を設定します。  
デフォルトでは、サーバーが再起動されたときに JBoss EAP はログメッセージを同じファイルに追加します。サーバーの再起動時にファイルを上書きする場合は **append** 属性を **false** に設定します。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=append,value=APPEND)
```

- エンコーディングを設定します。  
ハンドラーのエンコーディングを設定します (例: **utf-8**)。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
```

- ログフォーマッターを設定します。  
ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。FORMAT の値は必ず引用符で囲んでください。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=formatter,value=FORMAT)
```



### 注記

保存されたフォーマッターを参照する場合は **named-formatter** 属性を使用します。

- 自動フラッシュを設定します。  
毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- フィルター式を設定します。  
ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER\_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\\"WFLY\\"))"** に置き換える必要があります。

```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は「[フィルター式](#)」の項を参照してください。

Size ログハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てする必要があります。

以下の管理 CLI コマンドは Size ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=SIZE_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Size ログハンドラーを割り当てます。

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

#### Size ログハンドラーの削除

ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

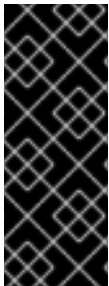
```
/subsystem=logging/size-rotating-file-handler=SIZE_HANDLER_NAME:remove
```

### 11.5.5. Periodic Size rotating ログハンドラーの設定

ここでは、管理 CLI を使用して Periodic Size rotating ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Logging** → **Configuration** と選択し、表示をクリックして **Handler** → **Periodic Size Handler** と選択すると、Periodic Size ログハンドラーを設定することができます。

Periodic Size ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- [新しい Periodic Size ログハンドラーの追加。](#)
- [Periodic Size ログハンドラーの設定。](#)
- [Periodic Size ログハンドラーのロガーへの割り当て](#)



#### 重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING\_PROFILE\_NAME/** になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に **/profile=PROFILE\_NAME** を付けます。

#### Periodic Size ログハンドラーの追加

Periodic Size ログハンドラーを追加する場合、**path** および **relative-to** 属性で構成される **file** 属性を使用してファイルパスを指定する必要があります。**path** 属性を使用して名前を含むログのファイルパスを設定します (例: **my-log.log**)。オプションで **relative-to** 属性を使用すると **path** が名前付きのパスと相対的になるよう設定できます (例: **jboss.server.log.dir**)。

また、**suffix** 属性を使用してローテーションしたログの接尾辞を設定する必要があります。これは、**.yyyy-MM-dd-HH** のように **java.text.SimpleDateFormat** が認識できる形式でなければなりません。ローテーションの周期はこの接尾辞を基に自動的に算出されます。

```
/subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:add(file={path=FILE_PATH,relative-to=RELATIVE_TO_PATH},suffix=SUFFIX)
```



## Periodic Size ログハンドラーの設定

必要性に応じて、以下の Periodic Size ログハンドラー属性を1つ以上設定する必要がある場合があります。利用できる Periodic Size ログハンドラー属性の完全リストと説明は、「[Periodic Size ログハンドラーの属性](#)」を参照してください。

- ログレベルを設定します。  
ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- ローテーションサイズを設定します。  
ファイルの最大サイズを設定します。この値を超えるとファイルがローテーションされます。デフォルトは2メガバイトを意味する **2m** です。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=rotate-size,
value=ROTATE_SIZE)
```

- 保持するバックアップログの最大数の設定  
保持するバックアップの数を設定します。デフォルト値は **1** です。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=max-backup-index,
value=MAX_BACKUPS)
```

- 起動時にログをローテーションするかどうかを設定します。  
デフォルトでは、サーバーの再起動時に新しいログファイルは作成されません。サーバーの再起動時にログをローテーションするには、**true** に設定します。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=rotate-on-boot,
value=ROTATE_ON_BOOT)
```

- 追加動作を設定します。  
デフォルトでは、サーバーが再起動されたときに JBoss EAP はログメッセージを同じファイルに追加します。サーバーの再起動時にファイルを上書きする場合は **append** 属性を **false** に設定します。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-
attribute(name=append,value=APPEND)
```

- エンコーディングを設定します。  
ハンドラーのエンコーディングを設定します (例: **utf-8**)。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-
attribute(name=encoding,value=ENCODING)
```

- ログフォーマッターを設定します。  
ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は `%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n` です。**FORMAT** の値は必ず引用符で囲んでください。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-
attribute(name=formatter,value=FORMAT)
```



### 注記

保存されたフォーマッターを参照する場合は `named-formatter` 属性を使用します。

- 自動フラッシュを設定します。  
毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-
attribute(name=autoflush,value=AUTO_FLUSH)
```

- フィルター式を設定します。  
ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER\_EXPRESSION** 変数では、`not(match("WFLY"))` のフィルター式を `"not(match(\"WFLY\"))"` に置き換える必要があります。

```
/subsystem=logging/periodic-size-rotating-file-
handler=PERIODIC_SIZE_HANDLER_NAME:write-attribute(name=filter-spec,
value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は「[フィルター式](#)」の項を参照してください。

### Periodic Size ログハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Periodic Size ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=PERIODIC_SIZE_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Periodic Size ログハンドラーを割り当てます。

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

### Periodic Size ログハンドラーの削除

ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/periodic-size-rotating-file-handler=PERIODIC_SIZE_HANDLER_NAME:remove
```

### 11.5.6. Syslog ハンドラーの設定

ここでは、管理 CLI を使用して Syslog ハンドラーを設定する方法を説明します。Syslog ハンドラーを使用すると、Syslog プロトコル (RFC-3164 または RFC-5424) をサポートするリモートログインサーバーにメッセージを送信できます。管理コンソールでは、**Configuration** → **Subsystems** → **Logging** → **Configuration** と選択し、表示 をクリックして **Handler** → **Syslog Handler** と選択すると、Syslog ハンドラーを設定することができます。

Syslog ハンドラーを設定するために実行する主なタスクは以下のとおりです。

- [新しい Syslog ハンドラーの追加](#)。
- [Syslog ハンドラーの設定](#)。
- [Syslog ハンドラーのロガーへの割り当て](#)



#### 重要

ログインプロファイルにこのログハンドラーを設定する場合、コマンドの最初は `/subsystem=logging/` ではなく `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に `/profile=PROFILE_NAME` を付けます。

#### Syslog ハンドラーの追加

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:add
```

#### Syslog ハンドラーの設定

必要性に応じて、以下の Syslog ハンドラー属性を 1 つ以上設定する必要がある場合があります。利用できる Syslog ハンドラー属性の完全リストと説明は、「[Syslog ハンドラーの属性](#)」を参照してください。

- ハンドラーのログレベルを設定します。デフォルトのレベルは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- ログに記録するアプリケーションの名前を設定します。デフォルトの名前は **java** です。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=app-name,value=APP_NAME)
```

- Syslog サーバーのアドレスを設定します。デフォルトのアドレスは **localhost** です。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=server-address,value=SERVER_ADDRESS)
```

- syslog サーバーのポートを設定します。デフォルトのポートは **514** です。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=port,value=PORT)
```

- RFC 仕様の定義どおりに syslog 形式を設定します。デフォルトの形式は **RFC5424** です。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:write-attribute(name=syslog-format,value=SYSLOG_FORMAT)
```

#### Syslog ハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Syslog ハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=SYSLOG_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Syslog ハンドラーを割り当てます。

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

#### Syslog ハンドラーの削除

ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/syslog-handler=SYSLOG_HANDLER_NAME:remove
```

### 11.5.7. Socket ログハンドラーの設定

ここでは、管理 CLI を使用して Socket ログハンドラーを設定する方法を説明します。ソケットは TCP または UDP ソケットになります。

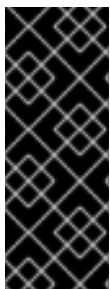


#### 注記

サーバーが管理者専用モードで起動された場合、ログメッセージは破棄されます。

Socket ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- [ソケットバインディングを追加します。](#)
- [ログフォーマッターを追加します。](#)
- [Socket ログハンドラーを追加し、設定を行います。](#)
- [Socket ログハンドラーをロガーに割り当てます。](#)



#### 重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は `/subsystem=logging/` ではなく `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に `/profile=PROFILE_NAME` を付けます。

ソケットバインディングの追加

**remote-destination-outbound-socket-binding** または **local-destination-outbound-socket-binding** を使用する [ソケットバインディング](#) として定義します。

```
/socket-binding-group=SOCKET_BINDING_GROUP/remote-destination-outbound-socket-binding=SOCKET_BINDING_NAME:add(host=HOST, port=PORT)
```

ログフォーマッターの設定

JSON フォーマッターなど、使用する [ログフォーマッター](#) を定義します。

```
/subsystem=logging/json-formatter=FORMATTER:add
```

**Socket** ログハンドラーの追加

Socket ログハンドラーを追加する場合、使用するソケットバインディングとフォーマッターを指定する必要があります。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:add(outbound-socket-binding-ref=SOCKET_BINDING_NAME,named-formatter=FORMATTER)
```

**Socket** ログハンドラーの設定

必要性に応じて、以下の Socket ログハンドラー属性を1つ以上設定する必要がある場合があります。利用できる Socket ログハンドラー属性の完全リストと説明は、「[Socket ログハンドラーの属性](#)」を参照してください。

- プロトコルを設定します。  
使用するプロトコルを設定します。デフォルトは **TCP** です。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=protocol,value=PROTOCOL)
```

- ログレベルを設定します。  
ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```



### 注記

サーバーの起動中、Socket ログハンドラーによって処理されるログメッセージは、ソケットバインディングが設定され、**logging** サブシステムが初期化されるまでキューに置かれます。ログレベルが **TRACE** や **DEBUG** などの低レベルに設定されている場合、起動時に大量のメモリーが消費されることがあります。

- エンコーディングを設定します。  
ハンドラーのエンコーディングを設定します (例: **utf-8**)。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=encoding,value=ENCODING)
```

- 自動フラッシュを設定します。  
毎回書き込みの後に自動的にフラッシュするかどうかを設定します。デフォルト値は **true** です。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=autoflush,value=AUTO_FLUSH)
```

- フィルター式を設定します。  
ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER\_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\"WFLY\"))"** に置き換える必要があります。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は「[フィルター式](#)」の項を参照してください。

### Socket ログハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Socket ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=SOCKET_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Socket ログハンドラーを割り当てます。

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

### Socket ログハンドラーの削除

ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/socket-handler=SOCKET_HANDLER_NAME:remove
```

### SSL/TLS を使用したソケット上でのログメッセージの送信

次の手順は、**SSL\_TCP** プロトコルを使用してソケット上でログメッセージを送信するように Socket ログハンドラーを設定する方法の例を示しています。この例では、Socket ログハンドラーによって使用される **elytron** サブシステムのキーストア、トラストマネージャー、およびクライアント SSL コンテキストを設定します。ルートロガーからのログメッセージは指定のソケット上で送信され、JSON フォーマッターによってフォーマットされます。

Elytron コンポーネントの設定に関する詳細は、JBoss EAP『[How to Configure Server Security](#)』の「[Elytron Subsystem](#)」を参照してください。

1. Elytron を設定します。
  - a. キーストアを追加します。

```
/subsystem=elytron/key-store=log-server-ks:add(path=/path/to/keystore.jks, type=JKS, credential-reference={clear-text=mypassword})
```

- b. トラストマネージャーを追加します。

```
/subsystem=elytron/trust-manager=log-server-tm:add(key-store=log-server-ks)
```

- c. クライアント SSL コンテキストを追加します。

```
/subsystem=elytron/client-ssl-context=log-server-context:add(trust-manager=log-server-tm, protocols=["TLSv1.2"])
```

2. ソケットバインディングを追加します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=log-server:add(host=localhost, port=4560)
```

3. JSON フォーマッターを追加します。

```
/subsystem=logging/json-formatter=json:add
```

4. Socket ログハンドラーを追加します。

```
/subsystem=logging/socket-handler=log-server-handler:add(named-formatter=json, level=INFO, outbound-socket-binding-ref=log-server, protocol=SSL_TCP, ssl-context=log-server-context)
```

5. ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=log-server-handler)
```

### 11.5.8. カスタムログハンドラーの設定

ここでは、管理 CLI を使用してカスタムログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration → Subsystems → Logging → Configuration** と選択し、**表示** をクリックして **Handler → Custom Handler** と選択すると、カスタムログハンドラーを設定することができます。

カスタムログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- [新しいカスタムログハンドラーの追加](#)。
- [カスタムログハンドラーの設定](#)。
- [カスタムログハンドラーのロガーへの割り当て](#)。

#### 重要

ログインプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING\_PROFILE\_NAME/** になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に **/profile=PROFILE\_NAME** を付けます。

#### カスタムログハンドラーの追加

カスタムログハンドラーを追加する場合、ハンドラーの Java クラスとハンドラーが含まれる JBoss EAP モジュールを指定する必要があります。クラスは **java.util.logging.Handler** を拡張する必要があります。



## 注記

すでに、カスタムロガーが含まれる [モジュールが作成](#) されている必要があります。作成されていないと、このコマンドの実行に失敗します。

```
/subsystem=logging/custom-  
handler=CUSTOM_HANDLER_NAME:add(class=CLASS_NAME,module=MODULE_NAME)
```

### カスタムログハンドラーの設定

必要性に応じて、以下のカスタムログハンドラー属性を1つ以上設定する必要がある場合があります。利用できるカスタムログハンドラー属性の完全リストと説明は、「[カスタムログハンドラーの属性](#)」を参照してください。

- ログレベルを設定します。  
ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-  
attribute(name=level,value=LEVEL)
```

- プロパティを設定します。  
ログハンドラーに必要なプロパティを設定します。setter メソッドを使用してプロパティにアクセスできなければなりません。

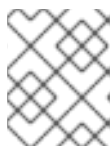
```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-  
attribute(name=properties.PROPERTY_NAME,value=PROPERTY_VALUE)
```

- エンコーディングを設定します。  
ハンドラーのエンコーディングを設定します (例: **utf-8**)。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-  
attribute(name=encoding,value=ENCODING)
```

- ログフォーマッターを設定します。  
ヘッダーの書式設定文字列を設定します。たとえば、デフォルトの書式設定文字列は **%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** です。 **FORMAT** の値は必ず引用符で囲んでください。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-  
attribute(name=formatter,value=FORMAT)
```



## 注記

保存されたフォーマッターを参照する場合は [named-formatter](#) 属性を使用します。

- フィルター式を設定します。  
ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER\_EXPRESSION** 変数では、 **not(match("WFLY"))** のフィルター式を **"not(match(\\"WFLY\\"))"** に置き換える必要があります。



```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は「[フィルター式](#)」の項を参照してください。

カスタムログハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てする必要があります。

以下の管理 CLI コマンドは、カスタムログハンドラーをルートロガーに割り当てます。

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

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Size ログハンドラーを割り当てます。

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

カスタムログハンドラーの削除

ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーまたは Async ログハンドラーに割り当てられている場合は削除できません。

```
/subsystem=logging/custom-handler=CUSTOM_HANDLER_NAME:remove
```

### 11.5.9. Async ログハンドラーの設定

ここでは、管理 CLI を使用して Async ログハンドラーを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Logging** → **Configuration** と選択し、**表示** をクリックして **Handler** → **Async Handler** と選択すると、Async ログハンドラーを設定することができます。

Async ログハンドラーを設定するために実行する主なタスクは以下のとおりです。

- [新しい Async ログハンドラーの追加](#)。
- [サブハンドラーの Async ログハンドラーへの追加](#)。
- [Async ログハンドラーの設定](#)。
- [Async ログハンドラーのロガーへの割り当て](#)



#### 重要

ログインプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING\_PROFILE\_NAME/** になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に **/profile=PROFILE\_NAME** を付けます。

#### Async ログハンドラーの追加

Async ログハンドラーを追加するときにキューの長さを指定する必要があります。これは、キューに保持できるログリクエストの最大数のことです。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:add(queue-length=QUEUE_LENGTH)
```

#### サブハンドラーの追加

1つ以上のハンドラーを Async ログハンドラーのサブハンドラーとして追加できます。ハンドラーが設定に存在しないと、このコマンドの実行に失敗するため注意してください。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:add-handler(name=HANDLER_NAME)
```

#### Async ログハンドラーの設定

必要性に応じて、以下の Async ログハンドラー属性を1つ以上設定する必要がある場合があります。利用できる Async ログハンドラー属性の完全リストと説明は、「[Async ログハンドラーの属性](#)」を参照してください。

- ログレベルを設定します。  
ハンドラーの適切なログレベルを設定します。デフォルトは **ALL** です。利用できるオプションは、「[ログレベル](#)」を参照してください。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:write-attribute(name=level,value=LEVEL)
```

- オーバーフローアクションを設定します。  
オーバーフローが発生したときに行うアクションを設定します。デフォルトの値は **BLOCK** で、キューが満杯になるとスレッドがブロックされます。この値を **DISCARD** に変更すると、キューが満杯になったときにログメッセージが破棄されます。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:write-attribute(name=overflow-action,value=OVERFLOW_ACTION)
```

- フィルター式を設定します。  
ハンドラーのログメッセージをフィルターするために式を設定します。必ずコンマと引用符はエスケープ処理し、引用符で囲むようにしてください。たとえば、以下の置換可能な **FILTER\_EXPRESSION** 変数では、**not(match("WFLY"))** のフィルター式を **"not(match(\\"WFLY\\"))"** に置き換える必要があります。

```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:write-attribute(name=filter-spec, value=FILTER_EXPRESSION)
```

利用可能なフィルター式の詳細は「[フィルター式](#)」の項を参照してください。

#### Async ログハンドラーのロガーへの割り当て

ログハンドラーをアクティブにするには、ロガーに割り当てる必要があります。

以下の管理 CLI コマンドは Async ログハンドラーをルートロガーに割り当てます。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=ASYNC_HANDLER_NAME)
```

以下の管理 CLI コマンドは、名前が **CATEGORY** によって指定されるロガーに Async ログハンドラーを割り当てます。

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

### Async ログハンドラーの削除

ログハンドラーは **remove** 操作で削除できます。ログハンドラーが現在ロガーに割り当てられている場合は削除できません。

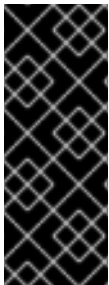
```
/subsystem=logging/async-handler=ASYNC_HANDLER_NAME:remove
```

## 11.6. ルートロガーの設定

ルートロガーは、ログカテゴリによってキャプチャーされないサーバーへ送信されたすべてのログメッセージ (指定のログレベル以上) をキャプチャーします。

ここでは、管理 CLI を使用してルートロガーを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Logging** → **Configuration** と選択し、**表示** をクリックして **Root Logger** を選択すると、ルートロガーを設定することができます。

### ルートロガーの設定



#### 重要

ロギングプロファイルにこのログハンドラーを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING\_PROFILE\_NAME/** になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に **/profile=PROFILE\_NAME** を付けます。

1. ログハンドラーをルートロガーへ割り当てます。  
ログハンドラーを追加します。

```
/subsystem=logging/root-logger=ROOT:add-handler(name=LOG_HANDLER_NAME)
```

ログハンドラーの削除

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

2. ログレベルを設定します。

```
/subsystem=logging/root-logger=ROOT:write-attribute(name=level,value=LEVEL)
```

使用できるルートロガー属性とその説明の完全リストは、「[ルートロガーの属性](#)」を参照してください。

## 11.7. ログフォーマッターの設定

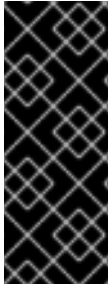
ログフォーマッターはハンドラーでのログメッセージの形式を定義します。**logging** サブシステムでは以下のログフォーマッターを設定できます。

- [パターンフォーマッター](#)
- [JSON ログフォーマッター](#)
- [XML ログフォーマッター](#)

- [カスタムログフォーマッター](#)

### 11.7.1. パターンフォーマッターの設定

ログハンドラーすべてで使用できる名前付きパターンフォーマッターを作成して、ログメッセージをフォーマットすることができます。



#### 重要

ロギングプロファイルにこのログフォーマッターを設定する場合、コマンドの最初は `/subsystem=logging/` ではなく `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に `/profile=PROFILE_NAME` を付けます。

#### パターンフォーマッターの作成

パターンフォーマッターを定義するとき、ログメッセージのフォーマットに使用するパターン文字列を指定します。パターン構文の詳細は、「[パターンフォーマッターのフォーマット文字](#)」を参照してください。

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

たとえば、デフォルトの設定はサーバーログへのロギングメッセージのログフォーマッター文字列として `%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n` を使用します。これにより、以下のようにフォーマットされるログメッセージが作成されます。

```
2016-03-18 15:49:32,075 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin console listening on http://127.0.0.1:9990
```

また、カラーマップを定義してログレベルごとに色を割り当てることもできます。形式は `LEVEL:COLOR` のカンマ区切りリストです。

- 有効なレベル: `finest`、`finer`、`fine`、`config`、`trace`、`debug`、`info`、`warning`、`warn`、`error`、`fatal`、`severe`
- 有効な色: `black`、`green`、`red`、`yellow`、`blue`、`magenta`、`cyan`、`white`、`brightblack`、`brightred`、`brightgreen`、`brightblue`、`brightyellow`、`brightmagenta`、`brightcyan`、`brightwhite`

```
/subsystem=logging/pattern-formatter=PATTERN_FORMATTER_NAME:write-attribute(name=color-map,value="LEVEL:COLOR,LEVEL:COLOR")
```

管理コンソールを使用してパターンログフォーマッターを設定することもできます。

1. ブラウザーで管理コンソールを開きます。
2. **Configuration** → **Subsystems** → **Logging** と選択します。
3. **Configuration** を選択し、**表示** をクリックします。
4. **Formatter** を選択し、**Pattern Formatter** オプションを選択します。

### 11.7.2. カスタムログフォーマッターの設定

JSON 形式でログメッセージをフォーマットする JSON ログフォーマッターを作成できます。



### 重要

ログインプロファイルにこのログフォーマッターを設定する場合、コマンドの最初は `/subsystem=logging/` ではなく `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に `/profile=PROFILE_NAME` を付けます。

### JSON ログフォーマッターの追加

```
/subsystem=logging/json-formatter=JSON_FORMATTER_NAME:add(pretty-print=true, exception-output-type=formatted)
```

これにより、以下のようにフォーマットされるログメッセージが作成されます。

```
{
  "timestamp": "2018-10-18T13:53:43.031-04:00",
  "sequence": 62,
  "loggerClassName": "org.jboss.as.server.logging.ServerLogger_$logger",
  "loggerName": "org.jboss.as",
  "level": "INFO",
  "message": "WFLYSRV0025: JBoss EAP 7.2.0.GA (WildFly Core 6.0.9.Final-redhat-20181015)
started in 5672ms - Started 495 of 679 services (331 services are lazy, passive or on-demand)",
  "threadName": "Controller Boot Thread",
  "threadId": 22,
  "mdc": {
  },
  "ndc": "",
  "hostName": "localhost.localdomain",
  "processName": "jboss-modules.jar",
  "processId": 7461
}
```

### Logstash JSON ログフォーマッターの追加の追加

```
/subsystem=logging/json-formatter=logstash:add(exception-output-type=formatted, key-overrides=[timestamp="@timestamp"], meta-data=[@version=1])
```

JSON フォーマッター属性は以下の説明どおりに使用できます。

- **key-overrides** 属性は、定義されたキーの名前をオーバーライドするために使用できます。
- **exception-output-type** 属性を **formatted** に設定すると、例外をオブジェクトとしてフォーマットできます。
- **exception-output-type** 属性を **detailed** に設定すると、例外スタックトレースを含めることができます。
- **exception-output-type** 属性を **detailed-and-formatted** に設定すると、例外をオブジェクトとしてフォーマットし、スタックトレースを含めることができます。
- **meta-data** 属性を使用すると、メタデータをログレコードに追加できます。

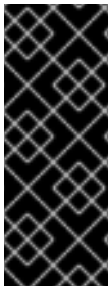
JSON フォーマッター属性の詳細は「[JSON ログフォーマッター属性](#)」を参照してください。

管理コンソールを使用して JSON ログフォーマッターを設定することもできます。

1. ブラウザーで管理コンソールを開きます。
2. **Configuration** → **Subsystems** → **Logging** と選択します。
3. **Configuration** を選択し、**表示** をクリックします。
4. **Formatter** を選択し、**JSON Formatter** オプションを選択します。

### 11.7.3. XML ログフォーマッターの設定

XML 形式でログメッセージをフォーマットする XML ログフォーマッターを作成できます。



#### 重要

ロギングプロファイルにこのログフォーマッターを設定する場合、コマンドの最初は `/subsystem=logging/` ではなく `/subsystem=logging/logging-profile=LOGGING_PROFILE_NAME/` になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に `/profile=PROFILE_NAME` を付けます。

#### XML ログフォーマッターの追加

```
/subsystem=logging/xml-formatter=XML_FORMATTER_NAME:add(pretty-print=true, exception-output-type=detailed-and-formatted)
```

これにより、以下のようにフォーマットされるログメッセージが作成されます。

```
<record>
  <timestamp>2018-10-18T13:55:53.419-04:00</timestamp>
  <sequence>62</sequence>
  <loggerClassName>org.jboss.as.server.logging.ServerLogger_<logger</loggerClassName>
  <loggerName>org.jboss.as</loggerName>
  <level>INFO</level>
  <message>WFLYSRV0025: JBoss EAP 7.2.0.GA (WildFly Core 6.0.9.Final-redhat-20181015)
started in 6271ms - Started 495 of 679 services (331 services are lazy, passive or on-
demand)</message>
  <threadName>Controller Boot Thread</threadName>
  <threadId>22</threadId>
  <mdc>
</mdc>
  <ndc></ndc>
  <hostName>localhost.localdomain</hostName>
  <processName>jboss-modules.jar</processName>
  <processId>7790</processId>
</record>
```

#### キーオーバーライド XML ログフォーマッターの追加

```
/subsystem=logging/xml-formatter=XML_FORMATTER_NAME:add(pretty-print=true, print-namespace=true, namespace-uri="urn:custom:1.0", key-overrides={message=msg, record=logRecord, timestamp=date}, print-details=true)
```

XML フォーマッター属性は以下の説明どおりに使用できます。

- **key-overrides** 属性は、定義されたキーの名前をオーバーライドするために使用できます。
- **exception-output-type** 属性を **formatted** に設定すると、例外をオブジェクトとしてフォーマットできます。
- **exception-output-type** 属性を **detailed** に設定すると、例外スタックトレースを含めることができます。
- **exception-output-type** 属性を **detailed-and-formatted** に設定すると、例外をオブジェクトとしてフォーマットし、スタックトレースを含めることができます。
- **meta-data** 属性を使用すると、メタデータをログレコードに追加できます。

XML フォーマッター属性の詳細は「[XML ログフォーマッター属性](#)」を参照してください。

管理コンソールを使用して XML ログフォーマッターを設定することもできます。

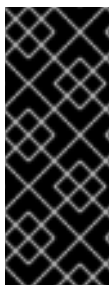
1. ブラウザーで管理コンソールを開きます。
2. **Configuration** → **Subsystems** → **Logging** と選択します。
3. **Configuration** を選択し、**表示** をクリックします。
4. **Formatter** を選択し、**XML Formatter オプション** を選択します。

#### 11.7.4. カスタムログフォーマッターの設定

ログハンドラーすべてで使用できるカスタムログフォーマッターを作成して、ログメッセージをフォーマットすることができます。

ここでは、管理 CLI を使用してカスタムログフォーマッターを設定する方法を説明します。

カスタムログフォーマッターの設定



#### 重要

ログインプロファイルにこのログフォーマッターを設定する場合、コマンドの最初は **/subsystem=logging/** ではなく **/subsystem=logging/logging-profile=LOGGING\_PROFILE\_NAME/** になります。

さらに、管理対象ドメインで実行している場合はコマンドの前に **/profile=PROFILE\_NAME** を付けます。

1. カスタムログフォーマッターを追加します。  
カスタムログフォーマッターを追加する場合、フォーマッターの Java クラスとフォーマッターが含まれる JBoss EAP モジュールを指定する必要があります。クラスは **java.util.logging.Formatter** を拡張する必要があります。



## 注記

すでに、カスタムフォーマッターが含まれるモジュールが作成されている必要があります。作成されていないと、このコマンドの実行に失敗します。

```
/subsystem=logging/custom-formatter=CUSTOM_FORMATTER_NAME:add(class=CLASS_NAME, module=MODULE_NAME)
```

2. ログフォーマッターに必要なプロパティを設定します。  
setter メソッドを使用してプロパティにアクセスできなければなりません。

```
/subsystem=logging/custom-formatter=CUSTOM_FORMATTER_NAME:write-attribute(name=properties.PROPERTY_NAME,value=PROPERTY_VALUE)
```

3. カスタムフォーマッターをログハンドラーに割り当てます。  
以下の管理 CLI コマンドは、Periodic Rotating ファイルハンドラーによって使用されるカスタムフォーマッターを割り当てます。

```
/subsystem=logging/periodic-rotating-file-handler=FILE_HANDLER_NAME:write-attribute(name=named-formatter, value=CUSTOM_FORMATTER_NAME)
```

### カスタム XML フォーマッターの例

以下の例は、カスタム XML フォーマッターを設定します。**org.jboss.logmanager** モジュールに提供される **java.util.logging.XMLFormatter** クラスを使用し、Console ログハンドラーに割り当てます。

```
/subsystem=logging/custom-formatter=custom-xml-formatter:add(class=java.util.logging.XMLFormatter, module=org.jboss.logmanager)
/subsystem=logging/console-handler=CONSOLE:write-attribute(name=named-formatter, value=custom-xml-formatter)
```

このフォーマッターを使用するログメッセージは以下のようにフォーマットされます。

```
<record>
  <date>2016-03-23T12:58:13</date>
  <millis>1458752293091</millis>
  <sequence>93963</sequence>
  <logger>org.jboss.as</logger>
  <level>INFO</level>
  <class>org.jboss.as.server.BootstrapListener</class>
  <method>logAdminConsole</method>
  <thread>22</thread>
  <message>WFLYSRV0051: Admin console listening on http://%s:%d</message>
  <param>127.0.0.1</param>
  <param>9990</param>
</record>
```

管理コンソールを使用したカスタムログフォーマッターの設定  
管理コンソールを使用してカスタムログフォーマッターを設定することもできます。

1. ブラウザーで管理コンソールを開きます。
2. **Configuration** → **Subsystems** → **Logging** と選択します。



3. **Configuration** を選択し、**表示** をクリックします。
4. **Formatter** を選択し、**Custom Formatter** オプションを選択します。

## 11.8. アプリケーションのログイン

アプリケーションのログインは、JBoss EAP の **logging** サブシステムを使用するか、デプロイメントごとに設定できます。

ログメッセージの取得に JBoss EAP ログカテゴリおよびハンドラーを使用する方法は「[Logging サブシステム](#)」を参照してください。

サポートされるアプリケーションログインフレームワークやデプロイメントごとのログイン設定など、アプリケーションログインの詳細は JBoss EAP 『[開発ガイド](#)』の「ログイン」の章を参照してください。

### 11.8.1. デプロイメントごとのログイン

デプロイメントごとのログインを使用すると、開発者はアプリケーションのログイン設定を事前に設定できます。アプリケーションがデプロイされると、定義された設定に従ってログインが開始されます。この設定によって作成されたログファイルにはアプリケーションの動作に関する情報のみが含まれません。



#### 注記

デプロイメントごとのログイン設定が行われない場合、すべてのアプリケーションとサーバーには **logging** サブシステムの設定が使用されます。

この方法では、システム全体のログインを使用する利点と欠点があります。利点は、JBoss EAP インスタンスの管理者がサーバーログイン以外のログインを設定する必要がないことです。欠点は、デプロイメントごとのログイン設定はサーバーの起動時に読み取り専用であるため、実行時に変更できないことです。

アプリケーションでデプロイメントごとのログインを使用する手順については、JBoss EAP 『[開発ガイド](#)』の「[デプロイメントごとのログインをアプリケーションに追加](#)」を参照してください。

#### 11.8.1.1. デプロイメントごとのログインの無効化

以下の方法の1つを使用するとデプロイメントごとのログインを無効にできます。

- **use-deployment-logging-config** 属性を **false** に設定します。  
**use-deployment-logging-config** 属性は、デプロイメントがデプロイメントごとにログインに対してスキャンされるかどうかを制御します。デフォルトは **true** です。デプロイメントごとのログインを無効にするにはこの属性を **false** に設定します。

```
/subsystem=logging:write-attribute(name=use-deployment-logging-config,value=false)
```

- **jboss-deployment-structure.xml** ファイルを使用して **logging** サブシステムを除外します。  
手順については、JBoss EAP 『[開発ガイド](#)』の「[サブシステムをデプロイメントから除外](#)」を参照してください。

### 11.8.2. ログインプロファイル

ロギングプロファイルは、デPLOYされたアプリケーションに割り当てることができる独立したロギング設定のセットです。通常の **logging** サブシステム同様にロギングプロファイルはハンドラー、カテゴリ、およびルートロガーを定義できますが、他のプロファイルや主要な **logging** サブシステムを参照できません。設定が容易である点でロギングプロファイルは **logging** サブシステムと似ています。

ロギングプロファイルを使用すると、管理者は他のロギング設定に影響を与えずに1つ以上のアプリケーションに固有するロギング設定を作成することができます。各プロファイルはサーバー設定で定義されるため、ロギング設定を変更しても影響を受けるアプリケーションを再デプロイする必要はありません。

各ロギングプロファイルには以下の項目を設定できます。

- 一意な名前。この値は必須です。
- 任意の数のログハンドラー。
- 任意の数のログカテゴリ。
- 最大1つのルートロガー。

アプリケーションでは **Logging-Profile** 属性を使用して、**MANIFEST.MF** ファイルで使用するロギングプロファイルを指定できます。

### 11.8.2.1. ロギングプロファイルの設定

ロギングプロファイルは、ログハンドラー、カテゴリ、およびルートロガーで設定できます。ロギングプロファイルの設定には、**logging** サブシステムの設定と同じ構文を使用しますが、以下の点が異なります。

- ルート設定パスが **/subsystem=logging/logging-profile=NAME** になります。
- ロギングプロファイルに他のロギングプロファイルを追加できません。
- **logging** サブシステムには、ロギングプロファイルに使用できない以下の属性があります。
  - **add-logging-api-dependencies**
  - **use-deployment-logging-config**

ロギングプロファイルの作成および設定

以下の手順では、管理 CLI を使用してロギングプロファイルを作成し、ファイルハンドラーとロガーカテゴリを設定します。管理コンソールでは **Configuration** → **Subsystems** → **Logging** → **Logging Profiles** と選択するとロギングプロファイルを設定することができます。

1. ロギングプロファイルを作成します。

```
/subsystem=logging/logging-profile=PROFILE_NAME:add
```

2. ファイルハンドラーを作成します。

```
/subsystem=logging/logging-profile=PROFILE_NAME/file-  
handler=FILE_HANDLER_NAME:add(file={path=>"LOG_NAME.log", "relative-  
to"=>"jboss.server.log.dir"})
```

```
/subsystem=logging/logging-profile=PROFILE_NAME/file-  
handler=FILE_HANDLER_NAME:write-attribute(name="level", value="DEBUG")
```

ファイルハンドラー属性の一覧は、「[File ログハンドラーの属性](#)」を参照してください。

3. ロガーカテゴリを作成します。

```
/subsystem=logging/logging-profile=PROFILE_NAME/logger=CATEGORY_NAME:add(level=TRACE)
```

ログカテゴリ属性のリストは、「[ログカテゴリの属性](#)」を参照してください。

4. ファイルハンドラーをカテゴリに割り当てます。

```
/subsystem=logging/logging-profile=PROFILE_NAME/logger=CATEGORY_NAME:add-handler(name="FILE_HANDLER_NAME")
```

この後、アプリケーションによって使用されるロギングプロファイルを **MANIFEST.MF** ファイルに設定できます。詳細は、JBoss EAP『[開発ガイド](#)』の「アプリケーションでのロギングプロファイルの指定」を参照してください。

### 11.8.2.2. ロギングプロファイル設定の例

この例は、ロギングプロファイルとそれを使用するアプリケーションの設定を表しています。管理 CLI コマンド、結果となる XML、およびアプリケーションの **MANIFEST.MF** が示されています。

ロギングプロファイルの例には次のような特徴があります。

- 名前は **accounts-app-profile** です。
- ログカテゴリは **com.company.accounts.ejbs** です。
- ログレベルは **TRACE** です。
- ログハンドラーは、**ejb-trace.log** ファイルを使用するファイルハンドラーです。

#### 管理 CLI セッション

```
/subsystem=logging/logging-profile=accounts-app-profile:add

/subsystem=logging/logging-profile=accounts-app-profile/file-handler=ejb-trace-file:add(file={path=>"ejb-trace.log", "relative-to"=>"jboss.server.log.dir"})

/subsystem=logging/logging-profile=accounts-app-profile/file-handler=ejb-trace-file:write-attribute(name="level", value="DEBUG")

/subsystem=logging/logging-profile=accounts-app-profile/logger=com.company.accounts.ejbs:add(level=TRACE)

/subsystem=logging/logging-profile=accounts-app-profile/logger=com.company.accounts.ejbs:add-handler(name="ejb-trace-file")
```

#### XML 設定

```
<logging-profiles>
  <logging-profile name="accounts-app-profile">
    <file-handler name="ejb-trace-file">
      <level name="DEBUG"/>
    </file-handler>
  </logging-profile>
</logging-profiles>
```

```

    <file relative-to="jboss.server.log.dir" path="ejb-trace.log"/>
  </file-handler>
  <logger category="com.company.accounts.ejbs">
    <level name="TRACE"/>
    <handlers>
      <handler name="ejb-trace-file"/>
    </handlers>
  </logger>
</logging-profile>
</logging-profiles>

```

## アプリケーションの MANIFEST.MF ファイル

```

Manifest-Version: 1.0
Logging-Profile: accounts-app-profile

```

### 11.8.3. デプロイメントロギング設定の表示

以下の管理 CLI コマンドを使用すると、特定のデプロイメントのロギング設定に関する情報を取得できます。

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

デプロイメントのロギング設定値である **CONFIG** には、以下の 3 つの値の 1 つを指定します。

- デプロイメントが **logging** サブシステムを使用する場合は **default** を指定します。これにより、**logging** サブシステムの設定が出力されます。
- デプロイメントが **logging** サブシステムに定義されている ロギングプロファイルを使用する場合は、**profile-PROFILE\_NAME** を指定します。これにより、ロギングプロファイルの設定が出力されます。
- 使用される設定ファイルへのパス (例: **myear.ear/META-INF/logging.properties**) を指定します。

たとえば、以下の管理 CLI コマンドは、特定のデプロイメントによって使用される **MYPROFILE** ロギングプロファイルの設定を表示します。

```
/deployment=mydeployment.war/subsystem=logging/configuration=profile-MYPROFILE:read-resource(recursive=true,include-runtime=true)
```

以下の情報が出力されます。

```

{
  "outcome" => "success",
  "result" => {
    "error-manager" => undefined,
    "filter" => undefined,
    "formatter" => {
      "MYFORMATTER" => {
        "class-name" => "org.jboss.logmanager.formatters.PatternFormatter",
        "module" => undefined,
        "properties" => {"pattern" => "%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n"}
      }
    }
  }
}

```

```

    },
    "handler" => {
      "MYPERIODIC" => {
        "class-name" => "org.jboss.logmanager.handlers.PeriodicRotatingFileHandler",
        "encoding" => undefined,
        "error-manager" => undefined,
        "filter" => undefined,
        "formatter" => "MYFORMATTER",
        "handlers" => [],
        "level" => "ALL",
        "module" => undefined,
        "properties" => {
          "append" => "true",
          "autoFlush" => "true",
          "enabled" => "true",
          "suffix" => ".yyyy-MM-dd",
          "fileName" => "EAP_HOME/standalone/log/deployment.log"
        }
      }
    },
    "logger" => {"MYCATEGORY" => {
      "filter" => undefined,
      "handlers" => [],
      "level" => "DEBUG",
      "use-parent-handlers" => true
    }},
    "pojo" => undefined
  }
}

```

また、再帰的な **read-resource** 操作を使用して、ログイン設定やデプロイメントに関する他の情報を取得することができます。

```

/deployment=DEPLOYMENT_NAME/subsystem=logging:read-resource(include-runtime=true,
recursive=true)

```

## 11.9. LOGGING サブシステムの調整

**logging** サブシステムのパフォーマンスを監視および最適化するための情報は、『[パフォーマンスチューニングガイド](#)』の「[Logging サブシステムの調整](#)」の項を参照してください。

## 第12章 データソース管理

### 12.1. JBOSS EAP データソース

#### JDBC

JDBC API は、Java アプリケーションがデータベースにアクセスする方法を定義する基準です。アプリケーションは JDBC ドライバーを参照するデータソースを設定します。その後、データベースではなくドライバーに対してアプリケーションを記述できます。ドライバーはコードをデータベース言語に変換します。そのため、適切なドライバーがインストールされていればアプリケーションをサポートされるデータベースで使用できます。

詳細は [JDBC 4.0 の仕様](#) を参照してください。

#### サポートされているデータベース

JBoss EAP 7 によってサポートされる JDBC 対応データベースのリストは、「[JBoss EAP 7 でサポートされる構成](#)」を参照してください。

#### データソースタイプ

リソースの一般的なタイプには、データソースと XA データソースの 2 つのタイプがあります。

##### 非 XA データソース

トランザクションを使用しないアプリケーション、または単一のデータベースでトランザクションを使用するアプリケーションに使用されます。

##### XA データソース

複数のデータベースまたはある XA トランザクションの一部として他の XA リソースを使用するアプリケーションによって使用されます。XA データソースを使用すると追加のオーバーヘッドが発生します。

JBoss EAP 管理インターフェースを使用してデータソースを作成するときに、使用するデータソースのタイプを指定します。

#### ExampleDS データソース

JBoss EAP には、データソースの定義方法を実証するために提供されるデータソース設定例 **ExampleDS** が含まれています。このデータソースは、H2 データベースを使用します。H2 データベースはライトウェイトなリレーショナルデータベース管理システムで、アプリケーションを迅速に構築できる開発者向けの機能を提供します。



#### 警告

**ExampleDS** データソースと H2 データベースは本番環境で使用しないでください。これは、アプリケーションのテストおよび構築に必要なすべての標準をサポートする非常に小さい自己完結型のデータソースであり、本番稼働用として堅牢またはスケーラブルではありません。

### 12.2. JDBC ドライバー

JBoss EAP でアプリケーションが使用するデータソースを定義する前に、最初に適切な JDBC ドライバーをインストールする必要があります。

## 12.2.1. コアモジュールとしての JDBC ドライバーのインストール

JDBC ドライバーをコアモジュールとしてインストールするには、最初に [JDBC ドライバーをコアモジュールとして追加](#) し、[datasources](#) サブシステムで **JDBC** ドライバーを登録する必要があります。

### 12.2.1.1. JDBC ドライバーをコアモジュールとして追加

以下の手順に従うと、管理 CLI を使用して JDBC ドライバーをコアモジュールとしてインストールすることができます。

1. JDBC ドライバーをダウンロードします。  
データベースのベンダーから適切な JDBC ドライバーをダウンロードします。一般的なデータベースの JDBC ドライバーをダウンロードできる場所については、「[JDBC ドライバーのダウンロードできる場所](#)」を参照してください。

JDBC ドライバーの JAR ファイルが ZIP または TAR アーカイブ内に含まれている場合は、必ずそのアーカイブを展開してください。

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

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

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

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

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

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

#### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

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

次に、アプリケーションデータソースによって参照されるよう、JDBC ドライバーとして登録する必要があります。

### 12.2.1.2. JDBC ドライバーの登録

ドライバーが [コアモジュールとしてインストール](#) されたら、以下の管理 CLI コマンドを使用して JDBC ドライバーとして登録する必要があります。管理対象ドメインを実行している場合は、コマンドの前に `/profile=PROFILE_NAME` を付けてください。

```
/subsystem=datasources/jdbc-driver=DRIVER_NAME:add(driver-name=DRIVER_NAME,driver-module-name=MODULE_NAME,driver-xa-datasource-class-name=XA_DATASOURCE_CLASS_NAME, driver-class-name=DRIVER_CLASS_NAME)
```

#### 注記

**driver-class-name** パラメーターは、JDBC ドライバー jar が `/META-INF/services/java.sql.Driver` ファイルで複数の jar を定義する場合のみ必要です。

たとえば、MySQL 5.1.36 JDBC ドライバー JAR の `/META-INF/services/java.sql.Driver` ファイルは、以下の 2 つのクラスを定義します。

- `com.mysql.jdbc.Driver`
- `com.mysql.fabric.jdbc.FabricMySQLDriver`

この場合、**driver-class-name=com.mysql.jdbc.Driver** で渡します。

たとえば、以下のコマンドは MySQL JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-name=mysql,driver-module-name=com.mysql,driver-xa-datasource-class-name=com.mysql.cj.jdbc.MySQLXADataSource, driver-class-name=com.mysql.jdbc.Driver)
```

アプリケーションのデータソースが JDBC ドライバーを参照できる状態になります。

### 12.2.2. JDBC ドライバーの JAR デプロイメントとしてのインストール

管理 CLI または管理コンソールを使用して JDBC ドライバーを JAR デプロイメントとしてインストールできます。JDBC 4 に対応するドライバーは、自動的に認識され、デプロイメント時に JDBC ドライバーとしてインストールされます。

以下の手順は、管理 CLI を使用した JDBC ドライバーのインストール方法になります。

#### 注記

JDBC ドライバーを [コアモジュール](#) としてインストールする方法が推奨されます。

1. JDBC ドライバーをダウンロードします。  
データベースのベンダーから適切な JDBC ドライバーをダウンロードします。一般的なデータベースの JDBC ドライバーをダウンロードできる場所については、「[JDBC ドライバーのダウンロードできる場所](#)」を参照してください。



JDBC ドライバーの JAR ファイルが ZIP または TAR アーカイブ内に含まれている場合は、必ずそのアーカイブを展開してください。

2. JDBC ドライバーが JDBC 4 に対応していない場合は、「[JDBC ドライバー JAR を JDBC 4 対応に更新](#)」の手順を参照してください。
3. JAR を JBoss EAP にデプロイします。

```
deploy PATH_TO_JDBC_JAR
```



#### 注記

管理対象ドメインでは、適切なサーバーグループを指定します。

たとえば、以下のコマンドは MySQL JDBC ドライバーをデプロイします。

```
deploy /path/to/mysql-connector-java-8.0.12.jar
```

JBoss EAP サーバーログにメッセージが表示され、データソースを定義するときに使用されるデプロイされたドライバーの名前が表示されます。

```
WFLYJCA0018: Started Driver service with driver-name = mysql-connector-java-8.0.12.jar
```

アプリケーションのデータソースが JDBC ドライバーを参照できる状態になります。

#### JDBC ドライバー JAR を JDBC 4 対応に更新

JDBC ドライバー JAR が JDBC 4 に対応していない場合、以下の手順に従ってデプロイ可能にすることができます。

1. 空の一時ディレクトリーを作成します。
2. **META-INF** サブディレクトリーを作成します。
3. **META-INF/services** サブディレクトリーを作成します。
4. **META-INF/services/java.sql.Driver** ファイルを作成し、JDBC ドライバーの完全修飾クラス名を示す 1 行を追加します。  
たとえば、MySQL JDBC ドライバーでは以下の行を追加します。

```
com.mysql.jdbc.Driver
```

5. JAR コマンドラインツールを使用して、この新しいファイルを JAR に追加します。

```
jar \-uf jdbc-driver.jar META-INF/services/java.sql.Driver
```

### 12.2.3. JDBC ドライバーをダウンロードできる場所

下表は、JBoss EAP で使用される一般的なデータベースの JDBC ドライバーをダウンロードできる場所を示しています。



## 注記

これらのリンク先は他社の Web サイトであるため、Red Hat は管理しておらず、積極的に監視も行っていません。ご使用のデータベースの最新ドライバーについては、データベースベンダーのドキュメントおよび Web サイトを確認してください。

表12.1 JDBC ドライバーをダウンロードできる場所

ベンダー	ダウンロード場所
MySQL	<a href="http://www.mysql.com/products/connector/">http://www.mysql.com/products/connector/</a>
PostgreSQL	<a href="http://jdbc.postgresql.org/">http://jdbc.postgresql.org/</a>
Oracle	<a href="http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html">http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html</a>
IBM	<a href="http://www-01.ibm.com/support/docview.wss?uid=swg21363866">http://www-01.ibm.com/support/docview.wss?uid=swg21363866</a>
Sybase	jConnect JDBC ドライバーは、SAP ASE インストールの SDK の一部です。現在、このドライバーのみをダウンロードできるサイトはありません。
Microsoft	<a href="http://msdn.microsoft.com/data/jdbc/">http://msdn.microsoft.com/data/jdbc/</a>

### 12.2.4. ベンダー固有クラスへのアクセス

場合によっては、アプリケーションが JDBC API の一部ではないベンダー固有の機能を使用する必要があることがあります。このような場合、そのアプリケーションで依存関係を宣言してベンダー固有の API にアクセスすることができます。



## 警告

これは高度な使用法です。JDBC API に含まれない機能を必要とするアプリケーションのみこれを実装します。



## 重要

このプロセスは、再認証メカニズムを使用し、ベンダー固有のクラスにアクセスする場合に必要です。

**MANIFEST.MF** ファイルまたは **jboss-deployment-structure.xml** ファイルを使用するとアプリケーションの依存関係を定義できます。

JDBC ドライバーをコアモジュールとしてインストールしていない場合は、インストールしてください。

## MANIFEST.MF ファイルの使用

1. アプリケーションの **META-INF/MANIFEST.MF** ファイルを編集します。
2. **Dependencies** 行を追加し、モジュール名を指定します。  
たとえば、以下の行は **com.mysql** モジュールを依存関係として宣言します。

```
Dependencies: com.mysql
```

## jboss-deployment-structure.xml ファイルの使用

1. アプリケーションの **META-INF/** または **WEB-INF/** フォルダで **jboss-deployment-structure.xml** というファイルを作成します。
2. **dependencies** 要素を使用してモジュールを指定します。  
たとえば、以下の **jboss-deployment-structure.xml** ファイル例は **com.mysql** モジュールを依存関係として宣言します。

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="com.mysql"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

以下のコード例は MySQL API にアクセスします。

```
import java.sql.Connection;
import org.jboss.jca.adapters.jdbc.WrappedConnection;

...

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



### 重要

接続は IronJacamar コンテナによって制御されるため、ベンダー固有の API ガイドラインに従ってください。

## 12.3. データソースの作成

データソースは管理コンソールまたは管理 CLI を使用して作成できます。

JBoss EAP 7 では、**enabled** 属性などのデータソース属性値を式で使用することができます。設定で式を使用する場合の詳細は、「[プロパティの置換](#)」の項を参照してください。

### 12.3.1. 非 XA データソースの作成

管理 CLI または管理コンソールを使用して、非 XA データソース作成できます。

管理コンソールを使用した非 XA データソースの定義

1. **Configuration** → **Subsystems** → **Datasources & Drivers** → **Datasources** と選択します。
2. 追加 (+) ボタンをクリックし、**Add Datasource** を選択します。
3. データソースタイプを選択できる **Add Datasource** ウィザードが表示されたら、**Next** をクリックします。これにより、データベースのテンプレートが作成されます。ウィザードの以下のページには、選択したデータソースに固有する値が自動的に入力されています。これにより、データソースの作成プロセスが簡単になります。
4. データソースの作成を終了する前に、**Test Connection** ページで接続をテストできます。
5. 詳細を確認し、**Finish** をクリックしてデータソースを作成します。

管理 CLI を使用した非 XA データソースの定義

**data-source add** 管理 CLI コマンドを使用すると、非 XA データソースを定義できます。

1. JDBC ドライバーをコアモジュールとしてインストールおよび登録していない場合は、「[コアモジュールとしての JDBC ドライバーのインストール](#)」を参照してインストールと登録を行ってください。
2. 適切な引数の値を指定し、**data-source add** コマンドを使用してデータソースを定義します。

```
data-source add --name=DATASOURCE_NAME --jndi-name=JNDI_NAME --driver-name=DRIVER_NAME --connection-url=CONNECTION_URL
```



#### 注記

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

これらのパラメーター値については、以下の「[データソースパラメーター](#)」の項を参照してください。

詳細な例は、サポート対象データベースの「[データソース設定例](#)」を参照してください。

データソースのパラメーター

#### jndi-name

データソースの JNDI 名は、**java:/** または **java:jboss/** で始まる必要があります。たとえば、**java:jboss/datasources/ExampleDS** になります。

#### driver-name

ドライバー名の値は、JDBC ドライバーがコアモジュールまたは JAR デプロイメントとしてインストールされたかによって異なります。

1. コアモジュールでは、ドライバー名の値は登録時に指定した JDBC ドライバーの名前になります。
2. JAR デプロイメントでは、**/META-INF/services/java.sql.Driver** ファイルに1つのクラスのみがある場合はドライバー名が JAR の名前になります。複数のクラスがリストされている場合は値が **JAR\_NAME + "\_" + DRIVER\_CLASS\_NAME + "\_" + MAJOR\_VERSION + "\_" + MINOR\_VERSION** (例: **mysql-connector-java-5.1.36-bin.jar\_com.mysql.jdbc.Driver\_5\_1**) になります。  
また、JDBC JAR がデプロイされると JBoss EAP サーバーログにドライバー名がリストされます。

```
WFLYJCA0018: Started Driver service with driver-name = mysql-connector-java-5.1.36-
bin.jar_com.mysql.jdbc.Driver_5_1
```

### connection-url

サポートされるデータベースの接続 URL 形式の詳細は、「[データソース接続 URL](#)」のリストを参照してください。

利用できるデータソース属性の完全リストは、「[データソース属性](#)」を参照してください。

## 12.3.2. XA データソースの作成

管理 CLI または管理コンソールを使用して XA データソースを作成できます。

管理コンソールを使用した XA データソースの定義

1. **Configuration** → **Subsystems** → **Datasources & Drivers** → **Datasources** と選択します。
2. 追加 (+) ボタンをクリックし、**XA データソースを追加** を選択します。
3. データソースタイプを選択できる **XA データソースを追加** ウィザードが表示されたら、**Next** をクリックします。これにより、データベースのテンプレートが作成されます。ウィザードの以下のページには、選択したデータソースに固有する値が自動的に入力されています。これにより、データソースの作成プロセスが簡単になります。
4. データソースの作成を終了する前に、**Test Connection** ページで接続をテストできます。
5. 詳細を確認し、**Finish** をクリックしてデータソースを作成します。

管理 CLI を使用した XA データソースの定義

**xa-data-source add** 管理 CLI コマンドを使用すると XA データソースを定義できます。



### 注記

管理対象ドメインでは、使用するプロファイルを指定する必要があります。管理 CLI コマンドの形式に応じて、コマンドの前に **/profile=PROFILE\_NAME** を付けるか、**--profile=PROFILE\_NAME** 引数に渡します。

1. JDBC ドライバーをコアモジュールとしてインストールおよび登録していない場合は、「[コアモジュールとしての JDBC ドライバーのインストール](#)」を参照してインストールと登録を行ってください。
2. 適切な引数の値を指定し、**xa-data-source add** コマンドを使用してデータソースを定義します。

```
xa-data-source add --name=XA_DATASOURCE_NAME --jndi-name=JNDI_NAME --driver-
name=DRIVER_NAME --xa-datasource-class=XA_DATASOURCE_CLASS --xa-
datasource-properties=
{"ServerName"=>"HOST_NAME","DatabaseName"=>"DATABASE_NAME"}
```

これらのパラメーター値については、以下の「[データソースパラメーター](#)」の項を参照してください。

3. **XA データソースプロパティ** を設定します。  
XA データソースを定義するときに最低でも1つの **XA データソースプロパティ** が必要になり

ます。XA データソースプロパティがないと、前のステップでデータソースを追加するときにエラーが発生します。XA データソースを定義するときに設定しなかったプロパティは後で個別に設定することができます。

- a. サーバー名を設定します。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME/xa-datasource-properties=ServerName:add(value=HOST_NAME)
```

- b. データベース名を設定します。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME/xa-datasource-properties=DatabaseName:add(value=DATABASE_NAME)
```

詳細な例は、サポート対象データベースの「[データソース設定例](#)」を参照してください。

データソースのパラメーター

#### jndi-name

データソースの JNDI 名は、**java:/** または **java:jboss/** で始まる必要があります。たとえば、**java:jboss/datasources/ExampleDS** になります。

#### driver-name

ドライバー名の値は、JDBC ドライバーがコアモジュールまたは JAR デプロイメントとしてインストールされたかによって異なります。

1. コアモジュールでは、ドライバー名の値は登録時に指定した JDBC ドライバーの名前になります。
2. JAR デプロイメントでは、**/META-INF/services/java.sql.Driver** ファイルに1つのクラスのみがある場合はドライバー名が JAR の名前になります。複数のクラスがリストされている場合は値が **JAR\_NAME + "\_" + DRIVER\_CLASS\_NAME + "\_" + MAJOR\_VERSION + "\_" + MINOR\_VERSION** (例: **mysql-connector-java-5.1.36-bin.jar\_com.mysql.jdbc.Driver\_5\_1**) になります。  
また、JDBC JAR がデプロイされると JBoss EAP サーバーログにドライバー名がリストされます。

```
WFLYJCA0018: Started Driver service with driver-name = mysql-connector-java-5.1.36-bin.jar_com.mysql.jdbc.Driver_5_1
```

#### xa-datasource-class

JDBC ドライバーの **javax.sql.XADataSource** クラスの実装に対する XA データソースクラスを指定します。

#### xa-datasource-properties

XA データソースを定義するときに最低でも1つの XA データソースプロパティが必要になります。XA データソースプロパティがないと、追加するときにエラーが発生します。XA データソースの定義後にプロパティを追加することもできます。

利用できるデータソース属性の完全リストは、「[データソース属性](#)」を参照してください。

## 12.4. データソースの編集

データソースは、管理コンソールまたは管理 CLI を使用して設定できます。

JBoss EAP 7 では、**enabled** 属性などのデータソース属性値を式で使用することができます。設定で式を使用する場合の詳細は、「[プロパティの置換](#)」の項を参照してください。

### 12.4.1. 非 XA データソースの編集

非 XA データソース設定は **data-source** 管理 CLI コマンドを使用して更新できます。管理コンソールでは、**Configuration** → **Subsystems** → **Datasources & Drivers** → **Datasources** と選択するとデータソース属性を更新できます。



#### 注記

非 XA データソースは JTA トランザクションと統合できます。データソースを JTA と統合する場合、必ず **jta** パラメーターを **true** に設定してください。

データソースの設定は、以下の管理 CLI コマンドを使用して更新できます。

```
data-source --name=DATASOURCE_NAME --ATTRIBUTE_NAME=ATTRIBUTE_VALUE
```



#### 注記

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

変更を反映するのにサーバーのリロードが必要になる場合があります。

### 12.4.2. XA データソースの編集

XA データソース設定は **xa-data-source** 管理 CLI コマンドを使用して更新できます。管理コンソールでは、**Configuration** → **Subsystems** → **Datasources & Drivers** → **Datasources** と選択するとデータソース属性を更新できます。

- XA データソースの設定は、以下の管理 CLI コマンドを使用して更新できます。

```
xa-data-source --name=XA_DATASOURCE_NAME -  
-ATTRIBUTE_NAME=ATTRIBUTE_VALUE
```

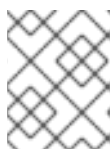


#### 注記

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

- 以下の管理 CLI コマンドを使用すると XA データソースプロパティを追加できます。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME/xa-datasource-  
properties=PROPERTY:add(value=VALUE)
```



#### 注記

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

変更を反映するのにサーバーのリロードが必要になる場合があります。

## 12.5. データソースの削除

データソースは管理コンソールまたは管理 CLI を使用して削除できます。

### 12.5.1. 非 XA データソースの削除

非 XA データソースは **data-source remove** 管理 CLI コマンドを使用して削除できます。管理コンソールでは、**Configuration** → **Subsystems** → **Datasources & Drivers** → **Datasources** と選択するとデータソースを削除できます。

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



#### 注記

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

データソースの削除後にサーバーのリロードが必要になります。

### 12.5.2. XA データソースの削除

XA データソースは **xa-data-source remove** 管理 CLI コマンドを使用して削除できます。管理コンソールでは、**Configuration** → **Subsystems** → **Datasources & Drivers** → **Datasources** と選択するとデータソースを削除できます。

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



#### 注記

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

XA データソースの削除後にサーバーのリロードが必要になります。

## 12.6. データソース接続のテスト

管理 CLI または管理コンソールを使用してデータソースの接続をテストし、設定が適切であることを検証できます。

### 管理 CLI を使用したデータソース接続のテスト

以下の管理 CLI コマンドを実行すると、データソースの接続をテストできます。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:test-connection-in-pool
```



#### 注記

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。XA データソースをテストする場合は、**data-source=DATASOURCE\_NAME** を **xa-data-source=XA\_DATASOURCE\_NAME** に置き換えてください。

### 管理コンソールを使用したデータソース接続のテスト

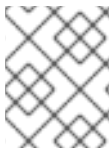


管理コンソールでデータソースの追加ウィザードを使用する場合、データソースの作成前に接続をテストできます。ウィザードのテスト接続画面でテスト接続ボタンをクリックします。

データソースが追加されたら、**Configuration** → **Subsystems** → **Datasources & Drivers** → **Datasources** と選択し、データソースを選択します。ドロップダウンメニューから **テスト接続** を選択すると、接続をテストできます。

## 12.7. データソース接続のフラッシュ

以下の管理 CLI コマンドを使用して、データベースの接続をフラッシュします。



### 注記

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

- プールの接続をすべてフラッシュします。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:flush-all-connection-in-pool
```

- プールの接続をすべて正常にフラッシュします。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:flush-gracefully-connection-in-pool
```

サーバーは、接続がアイドル状態なるまで待ってから接続をフラッシュします。

- プールのアイドル状態の接続をすべてフラッシュします。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:flush-idle-connection-in-pool
```

- プールの無効な接続をすべてフラッシュします。

```
/subsystem=datasources/data-source=DATASOURCE_NAME:flush-invalid-connection-in-pool
```

サーバーは、「データベース接続の検証」で説明した **valid-connection-checker-class-name** または `check-valid-connection-sql` 検証メカニズムなどによって無効と判断されたすべての接続をフラッシュします。

管理コンソールを使用して接続をフラッシュすることもできます。**Runtime** タブでサーバーを選択し、**Datasources** を選択してデータソースを指定します。ドロップダウンメニューを使用してアクションを選択します。

## 12.8. XA データソースのリカバリー

XA データソースは、XA グローバルトランザクションに参加できるデータソースです。XA グローバルトランザクションはトランザクションマネージャーによって調整され、1つのトランザクションで複数のリソースにまたがる可能性があります。参加者の1つが変更のコミットに失敗した場合、他の参加者がトランザクションをアポルトし、トランザクション発生前の状態にリストアします。これにより、一貫性を保持し、データの損失や破損を防ぎます。

XA リカバリーは、リソースやトランザクションの参加者がクラッシュしたり利用できない状態になっても、トランザクションの影響を受けるすべてのリソースが更新またはロールバックされるようにするプロセスです。XA リカバリーはユーザーが関与せずに行われます。

各 XA リソースにはその設定に関連するリカバリーモジュールが必要になります。リカバリーモジュールは、リカバリーの実行中に実行されるコードです。JBoss EAP は JDBC XA リソースのリカバリーモジュールを自動的に登録します。カスタムのリカバリーコードを実装する場合は XA データソースでカスタムモジュールを登録できます。リカバリーモジュールは **com.arjuna.ats.jta.recovery.XAResourceRecovery** クラスを拡張する必要があります。

### 12.8.1. XA リカバリーの設定

ほとんどの JDBC では、リカバリーモジュールがリソースに自動的に関連付けられます。この場合は、リカバリーモジュールがリソースに接続してリカバリーを実行することを許可するオプションのみを設定する必要があります。

以下の表は、XA リカバリーに固有する XA データソースパラメーターを表しています。これらの設定属性はデータソースの作成中または作成後に設定できます。これらは管理コンソールまたは管理 CLI を使用して設定できます。XA データソースのセット [の設定に関する詳細は、「XA データソースの編集」](#) を参照してください。

表12.2 XA リカバリーのデータソースパラメーター

属性	説明
recovery-username	リカバリーでリソースに接続するために使用するユーザー名。指定しないと、データソースのセキュリティー設定が使用されます。
recovery-password	リカバリーのリソースへの接続に使用するパスワード。指定しないと、データソースのセキュリティー設定が使用されます。
recovery-security-domain	リカバリーでリソースに接続するために使用するセキュリティードメイン。
recovery-plugin-class-name	カスタムのリカバリーモジュールを使用する必要がある場合は、この属性をモジュールの完全修飾クラス名に設定します。モジュールはクラス <b>com.arjuna.ats.jta.recovery.XAResourceRecovery</b> を拡張する必要があります。
recovery-plugin-properties	プロパティーを設定する必要があるカスタムのリカバリーモジュールを使用する場合は、この属性をプロパティーに対するコンマ区切りの <b>KEY=VALUE</b> ペアのリストに設定します。

### XA リカバリーの無効化

複数の XA データソースが同じ物理データベースに接続する場合、通常 XA リカバリーは XA データソースの1つのみに設定する必要があります。

以下の管理 CLI コマンドを使用して XA データソースのリカバリーを無効にします。

```
/subsystem=datasources/xa-data-source=XA_DATASOURCE_NAME:write-attribute(name=no-recovery,value=true)
```

## 12.8.2. ベンダー固有の XA リカバリー

### ベンダー固有の設定

一部のデータベースは、JBoss EAP トランザクションマネージャーによって管理される XA トランザクションに対応するために特定の設定が必要になります。詳細な最新情報については、データベースベンダーの資料を参照してください。

### MySQL

特別な設定は必要ありません。詳細は MySQL のドキュメントを参照してください。

### PostgreSQL および Postgres Plus Advanced Server

PostgreSQL による XA トランザクションの処理を可能にするには、**max\_prepared\_transactions** 設定パラメーターを **0** よりも大きい値または **max\_connections** 以上の値に変更します。

### Oracle

必ず Oracle ユーザー **USER** がリカバリーに必要なテーブルにアクセスできるようにしてください。

```
GRANT SELECT ON sys.dba_pending_transactions TO USER;  
GRANT SELECT ON sys.pending_trans$ TO USER;  
GRANT SELECT ON sys.dba_2pc_pending TO USER;  
GRANT EXECUTE ON sys.dbms_xa TO USER;
```

Oracle ユーザーに適切なパーミッションがないと、以下のようなエラーが表示される可能性があります。

```
WARN [com.arjuna.ats.jta.logging.logger18N] [com.arjuna.ats.internal.jta.recovery.xarecovery1]  
Local XARecoveryModule.xaRecovery got XA exception javax.transaction.xa.XAException,  
XAException.XAER_RMERR
```

### Microsoft SQL Server

詳細は、<http://msdn.microsoft.com/en-us/library/aa342335.aspx> を含む Microsoft SQL Server のドキュメントを参照してください。

### IBM DB2

特別な設定は必要ありません。詳細は IBM DB2 のドキュメントを参照してください。

### Sybase

Sybase は、XA トランザクションがデータベース上で有効であることを想定します。XA トランザクションはデータベース設定が正しくないと動作しません。**enable xact coordination** パラメーターは、Adaptive Server トランザクションコーディネーションサービスを有効または無効にします。このパラメーターを有効にすると、リモート Adaptive Server データのアップデートが、確実に元のトランザクションでコミットまたはロールバックされるようになります。

トランザクションコーディネーションを有効にするには、以下を使用します。

```
sp_configure 'enable xact coordination', 1
```

### MariaDB

特別な設定は必要ありません。詳細は MariaDB のドキュメントを参照してください。

### 既知の問題

ここで取り上げる既知の問題は、JBoss EAP 7 でサポートされる特定のデータベースおよび JDBC ドライババージョンの XA トランザクションの処理に関する問題になります。サポートされるデータベースの最新情報は、「[JBoss Enterprise Application Platform \(EAP\) 7 でサポートされる構成](#)」を参照してください。

## MySQL

MySQL は XA トランザクションを完全に処理できません。クライアントと MySQL の接続が切断されると、トランザクションに関する情報がすべて失われます。詳細は [MySQL のバグ](#) を参照してください。この問題は MuSQL 5.7 で修正されました。

## PostgreSQL および Postgres Plus Advanced Server

2 フェーズコミット (2PC) のコミットフェーズ中にネットワークの障害が発生すると、JDBC ドライバーによって **XAER\_RMERR** XAException エラーコードが返されます。このエラーは、トランザクションマネージャーでリカバリー不可能な重大なイベントが発生したことを示しますが、トランザクションはデータベース側で **in-doubt** 状態を維持し、ネットワーク接続の回復後に簡単に修正できます。適切な戻りコードは **XAER\_RMFAIL** または **XAER\_RETRY** になります。誤ったエラーコードにより、トランザクションが JBoss EAP 側で **Heuristic** 状態のままになり、データベースでロックが保持されるため、手動の介入が必要になります。詳細は [PostgreSQL のバグ](#) を参照してください。

1 フェーズコミットの最適化が使用されたときに接続の障害が発生した場合、JDBC ドライバーは **XAER\_RMERR** を返しますが、適切な戻りコードは **XAER\_RMFAIL** になります。そのため、1 フェーズコミット中にデータベースがデータをコミットし、同時に接続が切断されると、クライアントにはトランザクションがロールバックされたと伝えられるため、データの不整合が発生する場合があります。

Postgres Plus JDBC ドライバーは、Postgres Plus Server に存在するすべての準備済みトランザクションの XID を返すため、XID が属するデータベースを判断する方法がありません。JBoss EAP で複数のデータソースを同じデータベースに定義すると、in-doubt トランザクションリカバリーが誤ったアカウントで実行される可能性があります。この場合、リカバリーに失敗します。

## Oracle

一部のユーザー認証情報で設定されたデータソースを使用してリカバリーマネージャーがリカバリーを呼び出すと、JDBC ドライバーはデータベースインスタンスのすべてのユーザーに属する XID を返します。JDBC ドライバーは他のユーザーに属する XID をリカバリーしようとするため、例外 **ORA-24774: cannot switch to specified transaction** が発生します。

この問題を回避するには、リカバリーデータソース設定で認証情報が使用されるユーザーに **FORCE ANY TRANSACTION** 権限を付与します。特権の設定に関する詳細は [http://docs.oracle.com/database/121/ADMIN/ds\\_txnman.htm#ADMIN12259](http://docs.oracle.com/database/121/ADMIN/ds_txnman.htm#ADMIN12259) を参照してください。

## Microsoft SQL Server

2 フェーズコミット (2PC) のコミットフェーズ中にネットワークの障害が発生すると、JDBC ドライバーによって **XAER\_RMERR** XAException エラーコードが返されます。このエラーは、トランザクションマネージャーでリカバリー不可能な重大なイベントが発生したことを示しますが、トランザクションはデータベース側で **in-doubt** 状態を維持し、ネットワーク接続の回復後に簡単に修正できます。適切な戻りコードは **XAER\_RMFAIL** または **XAER\_RETRY** になります。誤ったエラーコードにより、トランザクションが JBoss EAP 側で **Heuristic** 状態のままになり、データベースでロックが保持されるため、手動の介入が必要になります。詳細は [Microsoft SQL Server の問題レポート](#) を参照してください。

1 フェーズコミットの最適化が使用されたときに接続の障害が発生した場合、JDBC ドライバーは **XAER\_RMERR** を返しますが、適切な戻りコードは **XAER\_RMFAIL** になります。そのため、1 フェーズコミット中にデータベースがデータをコミットし、同時に接続が切断されると、クライアントにはトランザクションがロールバックされたと伝えられるため、データの不整合が発生する場合があります。

## IBM DB2

1 フェーズコミット中に接続の障害が発生した場合、JDBC ドライバーは **XAER\_RMERR** を返しますが、適切な戻りコードは **XAER\_RMFAIL** です。そのため、1 フェーズコミット中にデータベースがデータをコミットし、同時に接続が切断されると、クライアントにはトランザクションがロールバックされたと伝えられるため、データの不整合が発生する場合があります。

## Sybase

2 フェーズコミット (2PC) のコミットフェーズ中にネットワークの障害が発生すると、JDBC ドライバーによって **XAER\_RMERR** XAException エラーコードが返されます。このエラーは、トランザクションマネージャーでリカバリー不可能な重大なイベントが発生したことを示しますが、トランザクションはデータベース側で **in-doubt** 状態を維持し、ネットワーク接続の回復後に簡単に修正できます。適切な戻りコードは **XAER\_RMFAIL** または **XAER\_RETRY** になります。誤ったエラーコードにより、トランザクションが JBoss EAP 側で **Heuristic** 状態のままになり、データベースでロックが保持されるため、手動の介入が必要になります。

1 フェーズコミットの最適化が使用されたときに接続の障害が発生した場合、JDBC ドライバーは **XAER\_RMERR** を返しますが、適切な戻りコードは **XAER\_RMFAIL** になります。そのため、1 フェーズコミット中にデータベースがデータをコミットし、同時に接続が切断されると、クライアントにはトランザクションがロールバックされたと伝えられるため、データの不整合が発生する場合があります。

Sybase トランザクションブランチが準備済み状態になる前に、Sybase 15.7 または 16 データベースへの挿入が含まれる XA トランザクションが失敗すると、XA トランザクションの繰り返しや同じ主キーでの同じレコードの挿入に失敗し、エラー **com.sybase.jdbc4.jdbc.SybSQLException: Attempt to insert duplicate key row** が発生します。この例外は、元の終了していない Sybase トランザクションブランチがロールバックするまで発生します。

## MariaDB

MariaDB は XA トランザクションを完全に処理できません。クライアントと MariaDB の接続が切断されると、トランザクションに関する情報がすべて失われます。

### MariaDB Galera クラスタ

MariaDB Galera クラスタでは XA トランザクションはサポートされません。

## 12.9. データベース接続の検証

データベースのメンテナンス、ネットワークの問題、またはその他の障害により、JBoss EAP からデータベースへの接続が失われることがあります。このような状況から回復するために、データソースのデータベース接続検証を有効にすることができます。

データベース接続の検証を設定するには、検証発生時を定義する検証タイミングメソッド、検証の実行方法を決定する検証メカニズム、および例外の処理方法を定義する例外ソーターを指定します。

1. 検証タイミングメソッドを1つ選択します。

### validate-on-match

**validate-on-match** オプションが **true** に設定されている場合は、データ接続が、次の手順で指定された検証メカニズムを使用して接続プールからチェックアウトされるたびに検証されます。

接続が有効でない場合は、警告がログに書き込まれ、プール内の次の接続が取得されます。このプロセスは、有効な接続が見つかるまで続行します。プール内の各接続を繰り返し処理しない場合は、**use-fast-fail** オプションを使用できます。有効な接続がプールにない場合は、新しい接続が作成されます。接続の作成に失敗すると、例外が要求元アプリケーションに返されます。

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

### background-validation

**background-validation** オプションを **true** に設定すると、使用前にバックグラウンドス

レッドで接続が周期的に検証されます。検証の頻度は **background-validation-millis** プロパティによって指定されます。**background-validation-millis** のデフォルト値は **0** で、無効になっています。

以下を考慮して **background-validation-millis** プロパティの値を決定してください。

- この値は **idle-timeout-minutes** 設定とは違う値に設定してください。
- 値が小さいほどプールの検証頻度が高くなり、より迅速に無効な接続がプールから削除されます。
- 値が小さいほど使用されるデータベースリソースが多くなります。値が大きいほど接続検証チェックの頻度が低くなり、データベースリソースの使用量が減りますが、無効な接続が検出されない期間が長くなります。



### 注記

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

## 2. 検証メカニズムを1つ選択します。

### valid-connection-checker-class-name

検証メカニズムとして **valid-connection-checker-class-name** を使用することが推奨されます。これは、使用中のデータベースの接続を検証するために使用される接続チェッカークラスを指定します。JBoss EAP は以下の接続チェッカーを提供します。

- **org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLReplicationValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.novendor.JDBC4ValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.novendor.NullValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker**
- **org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker**

### check-valid-connection-sql

**check-valid-connection-sql** を使用して、接続の検証に使用する SQL ステートメントを提供します。

以下は、Oracle の接続を検証するために使用する SQL ステートメントの例になります。

```
select 1 from dual
```

以下は、MySQL または PostgreSQL の接続を検証するために使用する SQL ステートメントの例になります。

```
select 1
```

### 3. 例外ソータークラス名を設定します。

例外が致命的とマークされた場合、接続はトランザクションに参加していてもすぐに閉じられます。致命的な接続例外を適切に検出およびクリーンアップするには、例外ソータークラスオブションを使用します。データソースタイプに適切な JBoss EAP 例外ソーターを選択します。

- `org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.informix.InformixExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.novendor.NullExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter`
- `org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter`

## 12.10. データソースセキュリティ

データソースセキュリティとは、データソース接続のパスワードを暗号化したり分かりにくくすることを言います。これらのパスワードはプレーンテキストで設定ファイルに保存できますが、セキュリティリスクが高くなります。

データソースセキュリティに使用できるメソッドは複数あります。以下には、各メソッドの例が含まれています。

### セキュリティドメインを使用したデータソースのセキュア化

データソースのセキュリティドメインが定義されます。

```
<security-domain name="DsRealm" cache-type="default">
  <authentication>
    <login-module code="ConfiguredIdentity" flag="required">
      <module-option name="userName" value="sa"/>
      <module-option name="principal" value="sa"/>
      <module-option name="password" value="sa"/>
    </login-module>
  </authentication>
</security-domain>
```



## 注記

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

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

```
<datasources>
<datasource jndi-name="java:jboss/datasources/securityDs"
pool-name="securityDs">
<connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
<driver>h2</driver>
<new-connection-sql>select current_user()</new-connection-sql>
<security>
<security-domain>DsRealm</security-domain>
</security>
</datasource>
</datasources>
```

セキュリティードメインの使用に関する詳細は、『[How to Configure Identity Management](#)』を参照してください。

## パスワード Vault を使用したデータソースのセキュア化

```
<security>
<user-name>admin</user-name>

<password>${VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmEtMW
NIMDMYNDdmNmI2TEIORV9CUkVBS3ZhdWx0}</password>
</security>
```

パスワード vault の使用に関する詳細は、JBoss EAP 『[How to Configure Server Security](#)』の「**Password Vault**」の項を参照してください。

## 認証情報ストアを使用したデータソースのセキュア化

認証情報ストアを使用してパスワードを提供することもできます。**elytron** サブシステムを使用すると、認証情報ストアを作成してパスワードをセキュアに保存し、JBoss EAP 全体でパスワードを使用することができます。認証情報ストアの作成および使用に関する詳細は、JBoss EAP 『[How to Configure Server Security](#)』の「**Credential Store**」を参照してください。

## 認証情報ストア参照を ExampleDS に追加

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=credential-reference,value=
{store=exampleCS, alias=example-ds-pw})
```

## 認証コンテキストを使用したデータソースのセキュア化

Elytron 認証コンテキストを使用して、ユーザー名とパスワードを提供することもできます。

以下の手順にしたがって、データソースセキュリティーの認証コンテキストを設定および使用します。

1. **password** と **user-name** を削除します。





```
/subsystem=datasources/data-source=ExampleDS:undefine-attribute(name=password)
/subsystem=datasources/data-source=ExampleDS:undefine-attribute(name=user-name)
```

2. データソースの Elytron セキュリティーを有効にします。

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

```
reload
```

3. 認証情報の **authentication-configuration** を作成します。  
認証設定には、接続時にデータソースに使用させる認証情報が含まれています。以下の例は、認証情報ストアへの参照を使用しますが、Elytron セキュリティードメインを使用することもできます。

```
/subsystem=elytron/authentication-configuration=exampleAuthConfig:add(authentication-name=sa,credential-reference={clear-text=sa})
```

4. **authentication-context** を作成します。

```
/subsystem=elytron/authentication-context=exampleAuthContext:add(match-rules=[{authentication-configuration=exampleAuthConfig}])
```

5. 認証コンテキストを使用するよう、データソースを更新します。  
以下の例は、認証コンテキストを使用するよう、**ExampleDS** を更新します。

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=authentication-context,value=exampleAuthContext)
```

```
reload
```



### 注記

**authentication-context** が設定されておらず、**elytron-enabled** が **true** に設定された場合、JBoss EAP は現在のコンテキストを使用して認証を行います。

## Kerberos を使用したデータソースの保護

kerberos 認証を使用してデータソースを保護するには、以下の設定が必要です。

- Kerberos がデータベースサーバーに設定されている。
- JBoss EAP ホストサーバーには、データベースサーバーのキータブエントリーがある。

kerberos を使用してデータソースを保護するには、以下を実行します。

1. Kerberos を使用するよう JBoss EAP を設定

```
/system-property=java.security.krb5.conf:add(value="/path/to/krb5.conf")
/system-property=sun.security.krb5.debug:add(value="false")
/system-property=sun.security.spnego.debug:add(value="false")
```

デバッグを行う場合は、**sun.security.krb5.debug** と **sun.security.spnego.debug** の値を **true** に変更します。本番環境では、値を **false** に設定することが推奨されます。

## 2. セキュリティーを設定します。

データソースを保護するには、レガシーセキュリティまたは Elytron セキュリティーを使用できます。

- レガシーのセキュリティで kerberos を使用するには、以下の手順に従います。
  - a. 期限切れのチケットをキャッシュから定期的に削除するように infinispan キャッシュを設定します。

```
batch
/subsystem=infinispan/cache-container=security:add(default-cache=auth-cache)
/subsystem=infinispan/cache-container=security/local-cache=auth-cache:add()
/subsystem=infinispan/cache-container=security/local-cache=auth-cache/expiration=EXPIRATION:add(lifespan=3540000,max-idle=3540000)
/subsystem=infinispan/cache-container=security/local-cache=auth-cache/eviction=EVICTON:add(strategy=LRU, max-entries=1000)
run-batch
```

以下の属性はチケットの有効期限を定義します。

- **lifespan**: KDC から新しい証明書を要求する間隔 (ミリ秒単位)。KDC で定義される lifespan 条件よりも小さい値になるように、**lifespan** 属性の値を設定します。
- **max-idle**: 未使用の場合に、有効なチケットがキャッシュから削除される間隔 (ミリ秒単位)。
- **max-entries**: キャッシュに保持する kerberos チケットの最大コピー数。この値は、データソースで設定された接続の数に対応します。  
エビクションの詳細は、『Infinispan』ユーザーガイドの「[Eviction and Data Container](#)」以下の項を参照してください。

## b. セキュリティードメインを作成します。

```
batch
/subsystem=security/security-domain=KerberosDatabase:add(cache-type=infinispan)
/subsystem=security/security-domain=KerberosDatabase/authentication=classic:add
/subsystem=security/security-domain=KerberosDatabase/authentication=classic/login-module="KerberosDatabase-Module":add(code="org.jboss.security.negotiation.KerberosLoginModule",module="org.jboss.security.negotiation",flag=required, module-options={"debug" => "false", "storeKey" => "false", "useKeyTab" => "true", "keyTab" => "/path/to/eap.keytab", "principal" => "PRINCIPAL@SERVER.COM", "doNotPrompt" => "true", "refreshKrb5Config" => "true", "isInitiator" => "true", "addGSSCredential" => "true", "credentialLifetime" => "-1"})
run-batch
```

- SQL サーバーに Microsoft JDBC ドライバーを使用する場合は、**module-options** に属性と **"wrapsphinx" => "true"** の値を追加します。
- デバッグするには、**module-options** の **debug** 属性の値を **true** に変更します。
- Elytron で kerberos を使用するには、以下を行います。
  - a. Elytron で kerberos ファクトリーを設定します。

```
/subsystem=elytron/kerberos-security-factory=krbsf:add(debug=false,
principal=PRINCIPAL@SERVER.COM, path=/path/to/keytab, request-lifetime=-1,
obtain-kerberos-ticket=true, server=false)
```

- デバッグを行う場合は、属性および **debug = true** の値を追加します。対応している属性の一覧は、『[How to Configure Server Security](#)』の「[Kerberos Security Factory Attributes](#)」を参照してください。

- b. kerberos ファクトリーを使用するよう認証設定を作成します。

```
/subsystem=elytron/authentication-configuration=kerberos-conf:add(kerberos-
security-factory=krbsf)
```

- c. authentication-context を作成します。

```
/subsystem=elytron/authentication-context=ds-context:add(match-rules=
[authentication-configuration=kerberos-conf])
```

3. データソースを kerberos で保護します。

- レガシーのセキュリティーを使用する場合:

- a. セキュリティードメインを使用するようにデータソースを設定します。

```
/subsystem=datasources/data-source=KerberosDS:add(connection-url="URL", min-
pool-size=0, max-pool-size=10, jndi-name="java:jboss/datasource/KerberosDS",
driver-name=<jdbc-driver>.jar, security-domain=KerberosDatabase, allow-multiple-
users=false, pool-prefill=false, pool-use-strict-min=false, idle-timeout-minutes=2)
```

- b. ベンダー固有の接続プロパティを設定します。

```
/subsystem=datasources/data-source=KerberosDS/connection-properties=
<connection-property-name>:add(value="(<kerberos-value>")
```

例: Oracle データベースの接続プロパティ。

```
/subsystem=datasources/data-source=KerberosDS/connection-
properties=oracle.net.authentication_services:add(value="(KERBEROS5)")
```

- Elytron を使用する場合:

- a. 認証コンテキストを使用するようにデータソースを設定します。

```
/subsystem=datasources/data-source=KerberosDS:add(connection-url="URL", min-
pool-size=0, max-pool-size=10, jndi-name="java:jboss/datasource/KerberosDS",
driver-name=<jdbc-driver>.jar, elytron-enabled=true, authentication-context=ds-
context, allow-multiple-users=false, pool-prefill=false, pool-use-strict-min=false, idle-
timeout-minutes=2)
```

- b. ベンダー固有の接続プロパティを設定します。

```
/subsystem=datasources/data-source=KerberosDS/connection-properties=
<connection-property-name>:add(value="(<kerberos-value>")
```

例: Oracle データベースの接続プロパティ

```
/subsystem=datasources/data-source=KerberosDS/connection-
properties=oracle.net.authentication_services:add(value="(KERBEROS5)")
```

kerberos 認証を使用する場合は、データソースに以下の属性と値を使用することが推奨されます。

- **pool-prefill=false**
- **pool-use-strict-min=false**
- **idle-timeout-minutes**

対応している属性のリストは、「[データソース属性](#)」を参照してください。

## 12.11. データソースの統計

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

### 12.11.1. データソース統計の有効化

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

管理 CLI を使用したデータソース統計の有効化

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



#### 注記

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。

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、ポップアップの **Topology** リンクをクリックします。**Topology** タブで該当するサーバーを選択し、**Reload** ドロップダウンオプションを選択してサーバーをリロードします。

## 12.11.2. データソース統計の表示

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

管理 CLI を使用したデータソース統計の表示

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



### 注記

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

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

利用可能な統計の詳細リストは、「[データソースの統計](#)」を参照してください。

管理コンソールを使用したデータソース統計の表示

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

利用可能な統計の詳細リストは、「[データソースの統計](#)」を参照してください。

## 12.12. データソースの調整

**datasources** サブシステムのパフォーマンスを監視および最適化するための情報は、『[パフォーマンスチューニングガイド](#)』の「[データソースおよびリソースアダプターの調整](#)」の項を参照してください。

## 12.13. キャパシティーポリシー

JBoss EAP は、データソースを含む JCA デプロイメントのキャパシティーポリシーの定義をサポートします。キャパシティーポリシーは、キャパシティーのインクリメントと呼ばれるプールの物理接続の作成方法と、キャパシティーのデクリメントと呼ばれる破棄方法を定義します。デフォルトのポリシーは、キャパシティーのインクリメントではリクエストごとに1つの接続を作成し、キャパシティーのデクリメントではアイドル状態のタイムアウトがスケジュールされたときにタイムアウトするとすべての接続が破棄されるよう設定されます。

キャパシティーポリシーを設定するには、キャパシティーインクリメンタークラスかキャパシティーデクリメンタークラスのいずれか、または両方を指定する必要があります。

### 例: キャパシティーポリシーの定義

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

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

指定したキャパシティーインクリメンターまたはデクリメンタークラスにプロパティを設定することもできます。

### 例: キャパシティーポリシーのプロパティ設定

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=capacity-incrementer-
properties.size, value=2)
```

```
/subsystem=datasources/data-source=ExampleDS:write-attribute(name=capacity-decrementer-
properties.size, value=2)
```

### MaxPoolSize インクリメンターポリシー

クラス名: **org.jboss.jca.core.connectionmanager.pool.capacity.MaxPoolSizeIncrementer**

**MaxPoolSize** インクリメンターポリシーは、リクエストごとにプールを最大サイズまでインクリメントします。このポリシーは、常時利用できる接続を最大数維持したい場合に便利です。

### Size インクリメンターポリシー

クラス名: `org.jboss.jca.core.connectionmanager.pool.capacity.SizeIncrementer`

Size インクリメンターポリシーは、リクエストごとにプールを指定の接続数インクリメントします。このポリシーは、次のリクエストにも接続が必要であることを予想する場合に追加の接続数でインクリメントしたい場合に便利です。

表12.3 Size ポリシープロパティ

名前	説明
Size	作成される接続数



#### 注記

これはデフォルトのインクリメントポリシーで、`size` の値は1になります。

### Watermark インクリメンターポリシー

クラス名: `org.jboss.jca.core.connectionmanager.pool.capacity.WatermarkIncrementer`

Watermark インクリメンターポリシーは、リクエストごとにプールを指定の接続数までインクリメントします。このポリシーは、常時プールに指定数の接続を維持したい場合に便利です。

表12.4 Watermark ポリシープロパティ

名前	説明
Watermark	接続数のウォーターマークレベル

### MinPoolSize デクリメンターポリシー

クラス名: `org.jboss.jca.core.connectionmanager.pool.capacity.MinPoolSizeDecrementer`

MinPoolSize デクリメンターポリシーは、リクエストごとにプールの最小サイズまでデクリメントします。このポリシーは、各アイドルタイムアウトリクエストの後に接続の数を制限したい場合に便利です。プールは先入れ先出し (FIFO) で操作します。

### Size デクリメンターポリシー

クラス名: `org.jboss.jca.core.connectionmanager.pool.capacity.SizeDecrementer`

Size デクリメンターポリシーは、アイドルタイムアウトリクエストごとにプールを指定の接続数デクリメントします。

表12.5 Size ポリシープロパティ

名前	説明
Size	破棄されるべき接続の数

このポリシーは、プールの使用度が徐々に減少することが予想されるためアイドルタイムアウトリクエストごとの追加接続数をデクリメントしたい場合に便利です。

プールは先入れ先出し (FIFO) で操作します。

### TimedOut デクリメンターポリシー

クラス名: `org.jboss.jca.core.connectionmanager.pool.capacity.TimedOutDecrementer`

TimedOut デクリメンターポリシーは、アイドルタイムアウトリクエストごとにタイムアウトした接続をすべてプールから削除します。プールは先入れ後出し (FILO) で操作します。



#### 注記

このポリシーはデフォルトのデクリメントポリシーです。

### TimedOut/FIFO デクリメンターポリシー

クラス名: `org.jboss.jca.core.connectionmanager.pool.capacity.TimedOutFIFODecrementer`

TimedOutFIFO デクリメンターポリシーは、アイドルタイムアウトリクエストごとにタイムアウトした接続をすべてプールから削除します。プールは先入れ先出し (FIFO) で操作します。

### Watermark デクリメンターポリシー

クラス名: `org.jboss.jca.core.connectionmanager.pool.capacity.WatermarkDecrementer`

Watermark デクリメンターポリシーは、アイドルタイムアウトリクエストごとにプールを指定の接続数までデクリメントします。このポリシーは、常時プールに指定数の接続を維持したい場合に便利です。プールは先入れ先出し (FIFO) で操作します。

表12.6 Watermark ポリシープロパティ

名前	説明
Watermark	接続数のウォーターマークレベル

## 12.14. エンリストメントトレース

**XAResource** インスタンスのエンリストメント中に発生するエラーを特定できるようにするために、エンリストメントトレースを記録することができます。エンリストメントトレースが有効である場合、必要時に正確なスタックトレースを作成できるように **jca** サブシステムはプール操作ごとに例外オブジェクトを作成しますが、これにはパフォーマンスのオーバーヘッドが発生します。

JBoss EAP 7.1 より、エンリストメントトレースはデフォルトで無効になっています。管理 CLI を使用してデータソースのエンリストメントトレースを有効にするには、**enlistment-trace** 属性を **true** に設定します。

非 XA データソースのエンリストメントトレースを有効にします。

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

XA データソースのエンリストメントトレースを有効にします。

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





## 警告

エンリストメントトレースを有効にするとパフォーマンスに影響することがあります。

## 12.15. データソース設定例

### 12.15.1. MySQL データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる MySQL データソースの設定例になります。

例: MySQL データソース例

```
<datasources>
  <datasource jndi-name="java:jboss/MySqlDS" pool-name="MySqlDS">
    <connection-url>jdbc:mysql://localhost:3306/jbossdb</connection-url>
    <driver>mysql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </datasource>
  <drivers>
    <driver name="mysql" module="com.mysql">
      <driver-class>com.mysql.jdbc.Driver</driver-class>
      <xa-datasource-class>com.mysql.cj.jdbc.MySQLXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

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

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

```
<module name="javax.transaction.api"/>
</dependencies>
</module>
```

### 管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MySQL JDBC ドライバーをコアモジュールとして追加します。

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

### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. MySQL JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-name=mysql,driver-module-
name=com.mysql,driver-xa-datasource-class-name=com.mysql.cj.jdbc.MySQLXADataSource,
driver-class-name=com.mysql.jdbc.Driver)
```

3. MySQL データソースを追加します。

```
data-source add --name=MySqlDS --jndi-name=java:jboss/MySqlDS --driver-name=mysql --
connection-url=jdbc:mysql://localhost:3306/jbossdb --user-name=admin --password=admin --
validate-on-match=true --background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter
```

## 12.15.2. MySQL XA データソースの例

以下は、XA データソースプロパティ、基本のセキュリティ、およびバリデーションオプションが含まれる MySQL XA データソースの設定例になります。

例: MySQL XA データソースの設定

```
<datasources>
<xa-datasource jndi-name="java:jboss/MySqlXADS" pool-name="MySqlXADS">
<xa-datasource-property name="ServerName">
localhost
```

```

</xa-datasource-property>
<xa-datasource-property name="DatabaseName">
  mysql
</xa-datasource-property>
<driver>mysql</driver>
<security>
  <user-name>admin</user-name>
  <password>admin</password>
</security>
<validation>
  <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
  <validate-on-match>true</validate-on-match>
  <background-validation>false</background-validation>
  <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
  </validation>
</xa-datasource>
<drivers>
  <driver name="mysql" module="com.mysql">
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <xa-datasource-class>com.mysql.cj.jdbc.MySQLXADataSource</xa-datasource-class>
  </driver>
</drivers>
</datasources>

```

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

```

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

```

管理 CLI コマンドの例

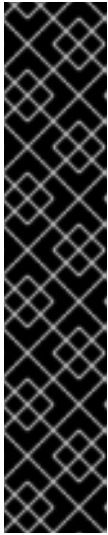
以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MySQL JDBC ドライバーをコアモジュールとして追加します。

```

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

```



## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. MySQL JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-name=mysql,driver-module-name=com.mysql,driver-xa-datasource-class-name=com.mysql.cj.jdbc.MySqlXADataSource,driver-class-name=com.mysql.jdbc.Driver)
```

3. MySQL XA データソースを追加します。

```
xa-data-source add --name=MySqlXADS --jndi-name=java:jboss/MySqlXADS --driver-name=mysql --user-name=admin --password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter --xa-datasource-properties={"ServerName"=>"localhost","DatabaseName"=>"mysqlpdb"}
```

### 12.15.3. PostgreSQL データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる PostgreSQL データソースの設定例になります。

例: PostgreSQL データソースの設定

```
<datasources>
<datasource jndi-name="java:jboss/PostgresDS" pool-name="PostgresDS">
  <connection-url>jdbc:postgresql://localhost:5432/postgresdb</connection-url>
  <driver>postgresql</driver>
  <security>
    <user-name>admin</user-name>
    <password>admin</password>
  </security>
  <validation>
    <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker"/>
    <validate-on-match>true</validate-on-match>
    <background-validation>false</background-validation>
    <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter"/>
  </validation>
```

```

</datasource>
<drivers>
  <driver name="postgresql" module="com.postgresql">
    <xa-datasource-class>org.postgresql.xa.PGXADDataSource</xa-datasource-class>
  </driver>
</drivers>
</datasources>

```

例: PostgreSQL JDBC ドライバーの `module.xml` ファイル

```

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

```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. PostgreSQL JDBC ドライバーをコアモジュールとして追加します。

```

module add --name=com.postgresql --resources=/path/to/postgresql-9.3-1102.jdbc4.jar --
dependencies=javax.api,javax.transaction.api

```

### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. PostgreSQL JDBC ドライバーを登録します。

```

/subsystem=datasources/jdbc-driver=postgresql:add(driver-name=postgresql,driver-module-
name=com.postgresql,driver-xa-datasource-class-
name=org.postgresql.xa.PGXADDataSource)

```

3. PostgreSQL データソースを追加します。

```

data-source add --name=PostgresDS --jndi-name=java:jboss/PostgresDS --driver-

```

```

name=postgresql --connection-url=jdbc:postgresql://localhost:5432/postgresdb --user-
name=admin --password=admin --validate-on-match=true --background-validation=false --
valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker
--exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter

```

#### 12.15.4. PostgreSQL XA データソースの例

以下は、XA データソースプロパティ、基本のセキュリティー、およびバリデーションオプションが含まれる PostgreSQL XA データソースの設定例になります。

例: PostgreSQL XA データソースの例

```

<datasources>
  <xa-datasource jndi-name="java:jboss/PostgresXADS" pool-name="PostgresXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="PortNumber">
      5432
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      postgresdb
    </xa-datasource-property>
    <driver>postgresql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter"/>
    </validation>
  </xa-datasource>
</drivers>
  <driver name="postgresql" module="com.postgresql">
    <xa-datasource-class>org.postgresql.xa.PGXADDataSource</xa-datasource-class>
  </driver>
</drivers>
</datasources>

```

例: PostgreSQL JDBC ドライバーの **module.xml** ファイル

```

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

```

```
<module name="javax.transaction.api"/>
</dependencies>
</module>
```

### 管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. PostgreSQL JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.postgresql --resources=/path/to/postgresql-9.3-1102.jdbc4.jar --
dependencies=javax.api,javax.transaction.api
```

### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. PostgreSQL JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=postgresql:add(driver-name=postgresql,driver-module-
name=com.postgresql,driver-xa-datasource-class-
name=org.postgresql.xa.PGXADDataSource)
```

3. PostgreSQL XA データソースを追加します。

```
xa-data-source add --name=PostgresXADS --jndi-name=java:jboss/PostgresXADS --driver-
name=postgresql --user-name=admin --password=admin --validate-on-match=true --
background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker
--exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter --xa-
datasource-properties=
{"ServerName"=>"localhost","PortNumber"=>"5432","DatabaseName"=>"postgresdb"}
```

### 12.15.5. Oracle データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる Oracle データソースの設定例になります。

例: Oracle データソースの例

```
<datasources>
<datasource jndi-name="java:jboss/OracleDS" pool-name="OracleDS">
```

```

<connection-url>jdbc:oracle:thin:@localhost:1521:XE</connection-url>
<driver>oracle</driver>
<security>
  <user-name>admin</user-name>
  <password>admin</password>
</security>
<validation>
  <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker"/>
  <validate-on-match>true</validate-on-match>
  <background-validation>false</background-validation>
  <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter"/>
</validation>
</datasource>
<drivers>
  <driver name="oracle" module="com.oracle">
    <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
  </driver>
</drivers>
</datasources>

```

例: Oracle JDBC ドライバーの **module.xml** ファイル

```

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

```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Oracle JDBC ドライバーをコアモジュールとして追加します。

```

module add --name=com.oracle --resources=/path/to/ojdbc7.jar --
dependencies=javax.api,javax.transaction.api

```



**重要**

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. Oracle JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=oracle:add(driver-name=oracle,driver-module-name=com.oracle,driver-xa-datasource-class-name=oracle.jdbc.xa.client.OracleXADataSource)
```

3. Oracle データソースを追加します。

```
data-source add --name=OracleDS --jndi-name=java:jboss/OracleDS --driver-name=oracle -
-connection-url=jdbc:oracle:thin:@localhost:1521:XE --user-name=admin --password=admin
--validate-on-match=true --background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter
```

### 12.15.6. Oracle XA データソースの例

**重要**

Oracle XA データソースにアクセスするユーザーは、以下の設定を適用しないと XA リカバリーが適切に操作しません。値 **user** は、JBoss EAP から Oracle に接続するために定義されたユーザーです。

- **GRANT SELECT ON sys.dba\_pending\_transactions TO user;**
- **GRANT SELECT ON sys.pending\_trans\$ TO user;**
- **GRANT SELECT ON sys.dba\_2pc\_pending TO user;**
- **GRANT EXECUTE ON sys.dbms\_xa TO user;**

以下は、XA データソースプロパティ、基本のセキュリティ、およびバリデーションオプションが含まれる Oracle XA データソースの設定例になります。

例: Oracle XA データソースの設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/OracleXADS" pool-name="OracleXADS">
```

```

<xa-datasource-property name="URL">
  jdbc:oracle:thin:@oracleHostName:1521:orcl
</xa-datasource-property>
<driver>oracle</driver>
<xa-pool>
  <is-same-rm-override>>false</is-same-rm-override>
</xa-pool>
<security>
  <user-name>admin</user-name>
  <password>admin</password>
</security>
<validation>
  <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker"/>
  <validate-on-match>true</validate-on-match>
  <background-validation>>false</background-validation>
  <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter"/>
</validation>
</xa-datasource>
<drivers>
  <driver name="oracle" module="com.oracle">
    <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
  </driver>
</drivers>
</datasources>

```

例: Oracle JDBC ドライバーの **module.xml** ファイル

```

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

```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Oracle JDBC ドライバーをコアモジュールとして追加します。

```

module add --name=com.oracle --resources=/path/to/ojdbc7.jar --
dependencies=javax.api,javax.transaction.api

```

## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. Oracle JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=oracle:add(driver-name=oracle,driver-module-name=com.oracle,driver-xa-datasource-class-name=oracle.jdbc.xa.client.OracleXADataSource)
```

3. Oracle XA データソースを追加します。

```
xa-data-source add --name=OracleXADS --jndi-name=java:jboss/OracleXADS --driver-name=oracle --user-name=admin --password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter --same-rm-override=false --xa-datasource-properties={"URL"=>"jdbc:oracle:thin:@oracleHostName:1521:orcl"}
```

### 12.15.7. Microsoft SQL Server データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる Microsoft SQL Server データソースの設定例になります。

例: Microsoft SQL Server データソースの設定

```
<datasources>
<datasource jndi-name="java:jboss/MSSQLDS" pool-name="MSSQLDS">
  <connection-url>jdbc:sqlserver://localhost:1433;DatabaseName=MyDatabase</connection-url>
  <driver>sqlserver</driver>
  <security>
    <user-name>admin</user-name>
    <password>admin</password>
  </security>
  <validation>
    <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker"/>
    <validate-on-match>true</validate-on-match>
    <background-validation>false</background-validation>
    <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter"/>
  </validation>
</datasource>
</datasources>
```

```

</validation>
</datasource>
<drivers>
  <driver name="sqlserver" module="com.microsoft">
    <xa-datasource-class>com.microsoft.sqlserver.jdbc.SQLServerXADataSource</xa-datasource-
class>
  </driver>
</drivers>
</datasources>

```

例: Microsoft SQL Server JDBC ドライバーの **module.xml** ファイル

```

<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.microsoft">
  <resources>
    <resource-root path="sqljdbc42.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
    <module name="javax.xml.bind.api"/>
  </dependencies>
</module>

```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Microsoft SQL Server の JDBC ドライバーをコアモジュールとして追加します。

```

module add --name=com.microsoft --resources=/path/to/sqljdbc42.jar --
dependencies=javax.api,javax.transaction.api,javax.xml.bind.api

```

## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. Microsoft SQL Server の JDBC ドライバーを登録します。

```

/subsystem=datasources/jdbc-driver=sqlserver:add(driver-name=sqlserver,driver-module-
name=com.microsoft,driver-xa-datasource-class-
name=com.microsoft.sqlserver.jdbc.SQLServerXADataSource)

```

## 3. Microsoft SQL Server データソースを追加します。

```
data-source add --name=MSSQLDS --jndi-name=java:jboss/MSSQLDS --driver-
name=sqlserver --connection-
url=jdbc:sqlserver://localhost:1433;DatabaseName=MyDatabase --user-name=admin --
password=admin --validate-on-match=true --background-validation=false --valid-connection-
checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter
```

## 12.15.8. Microsoft SQL Server XA データソースの例

以下は、XA データソースプロパティ、基本のセキュリティー、およびバリデーションオプションが含まれる Microsoft SQL Server XA データソースの設定例になります。

例: Microsoft SQL Server XA データソースの設定

```
<datasources>
<xa-datasource jndi-name="java:jboss/MSSQLXADS" pool-name="MSSQLXADS">
  <xa-datasource-property name="ServerName">
    localhost
  </xa-datasource-property>
  <xa-datasource-property name="DatabaseName">
    mssqldb
  </xa-datasource-property>
  <xa-datasource-property name="SelectMethod">
    cursor
  </xa-datasource-property>
  <driver>sqlserver</driver>
  <xa-pool>
    <is-same-rm-override>>false</is-same-rm-override>
  </xa-pool>
  <security>
    <user-name>admin</user-name>
    <password>admin</password>
  </security>
  <validation>
    <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker"/>
    <validate-on-match>true</validate-on-match>
    <background-validation>false</background-validation>
    <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter"/>
  </validation>
</xa-datasource>
<drivers>
  <driver name="sqlserver" module="com.microsoft">
    <xa-datasource-class>com.microsoft.sqlserver.jdbc.SQLServerXADataSource</xa-datasource-
class>
  </driver>
</drivers>
</datasources>
```

例: Microsoft SQL Server JDBC ドライバーの **module.xml** ファイル

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.microsoft">
  <resources>
    <resource-root path="sqljdbc42.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
    <module name="javax.xml.bind.api"/>
  </dependencies>
</module>
```

### 管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Microsoft SQL Server の JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.microsoft --resources=/path/to/sqljdbc42.jar --
dependencies=javax.api,javax.transaction.api,javax.xml.bind.api
```

### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. Microsoft SQL Server の JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=sqlserver:add(driver-name=sqlserver,driver-module-
name=com.microsoft,driver-xa-datasource-class-
name=com.microsoft.sqlserver.jdbc.SQLServerXADataSource)
```

3. Microsoft SQL Server XA データソースを追加します。

```
xa-data-source add --name=MSSQLXADS --jndi-name=java:jboss/MSSQLXADS --driver-
name=sqlserver --user-name=admin --password=admin --validate-on-match=true --
background-validation=false --background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLExceptionSorter --same-rm-
override=false --xa-datasource-properties=
{"ServerName"=>"localhost","DatabaseName"=>"mssqldb","SelectMethod"=>"cursor"}
```

### 12.15.9. IBM DB2 データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる IBM DB2 データソースの設定例になります。

例: IBM DB2 データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/DB2DS" pool-name="DB2DS">
    <connection-url>jdbc:db2://localhost:50000/ibmdb2db</connection-url>
    <driver>ibmdb2</driver>
    <pool>
      <min-pool-size>0</min-pool-size>
      <max-pool-size>50</max-pool-size>
    </pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter"/>
    </validation>
  </datasource>
</drivers>
  <driver name="ibmdb2" module="com.ibm">
    <xa-datasource-class>com.ibm.db2.jcc.DB2XADataSource</xa-datasource-class>
  </driver>
</drivers>
</datasources>
```

例: IBM DB2 JDBC ドライバーの **module.xml** ファイル

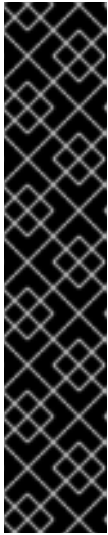
```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm">
  <resources>
    <resource-root path="db2jcc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. IBM DB2 の JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.ibm --resources=/path/to/db2jcc4.jar --
dependencies=javax.api,javax.transaction.api
```



## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. IBM DB2 の JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=ibmdb2:add(driver-name=ibmdb2,driver-module-name=com.ibm,driver-xa-datasource-class-name=com.ibm.db2.jcc.DB2XADataSource)
```

3. IBM DB2 データソースを追加します。

```
data-source add --name=DB2DS --jndi-name=java:jboss/DB2DS --driver-name=ibmdb2 --connection-url=jdbc:db2://localhost:50000/ibmdb2db --user-name=admin --password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter --min-pool-size=0 --max-pool-size=50
```

### 12.15.10. IBM DB2 XA のデータソースの例

以下は、XA データソースプロパティ、基本のセキュリティー、およびバリデーションオプションが含まれる IBM DB2 XA データソースの設定例になります。

例: IBM DB2 XA データソースの設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/DB2XADS" pool-name="DB2XADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      ibmdb2db
    </xa-datasource-property>
    <xa-datasource-property name="PortNumber">
      50000
    </xa-datasource-property>
    <xa-datasource-property name="DriverType">
      4
    </xa-datasource-property>
    <driver>ibmdb2</driver>
    <xa-pool>
      <is-same-rm-override>>false</is-same-rm-override>
    </xa-pool>
  </xa-datasource>
</datasources>
```



```

<security>
  <user-name>admin</user-name>
  <password>admin</password>
</security>
<recovery>
  <recover-plugin class-name="org.jboss.jca.core.recovery.ConfigurableRecoveryPlugin">
    <config-property name="EnableIsValid">
      false
    </config-property>
    <config-property name="IsValidOverride">
      false
    </config-property>
    <config-property name="EnableClose">
      false
    </config-property>
  </recover-plugin>
</recovery>
<validation>
  <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker"/>
  <validate-on-match>true</validate-on-match>
  <background-validation>false</background-validation>
  <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter"/>
</validation>
</xa-datasource>
<drivers>
  <driver name="ibmdb2" module="com.ibm">
    <xa-datasource-class>com.ibm.db2.jcc.DB2XADataSource</xa-datasource-class>
  </driver>
</drivers>
</datasources>

```

例: IBM DB2 JDBC ドライバーの **module.xml** ファイル

```

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

```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. IBM DB2 の JDBC ドライバーをコアモジュールとして追加します。

```

module add --name=com.ibm --resources=/path/to/db2jcc4.jar --
dependencies=javax.api,javax.transaction.api

```



## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. IBM DB2 の JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=ibmdb2:add(driver-name=ibmdb2,driver-module-name=com.ibm,driver-xa-datasource-class-name=com.ibm.db2.jcc.DB2XADataSource)
```

3. IBM DB2 XA データソースを追加します。

```
xa-data-source add --name=DB2XADS --jndi-name=java:jboss/DB2XADS --driver-name=ibmdb2 --user-name=admin --password=admin --validate-on-match=true --background-validation=false --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter --same-rm-override=false --recovery-plugin-class-name=org.jboss.jca.core.recovery.ConfigurableRecoveryPlugin --recovery-plugin-properties={"EnableValid"=>"false","IsValidOverride"=>"false","EnableClose"=>"false"} --xa-datasource-properties={"ServerName"=>"localhost","DatabaseName"=>"ibmdb2db","PortNumber"=>"50000","DriverType"=>"4"}
```

### 12.15.11. Sybase データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる Sybase データソースの設定例になります。

例: Sybase データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/SybaseDB" pool-name="SybaseDB">
    <connection-url>jdbc:sybase:Tds:localhost:5000/DATABASE?
JCONNECT_VERSION=6</connection-url>
    <driver>sybase</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
    </validation>
  </datasource>
</datasources>
```

```

<background-validation>false</background-validation>
<exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter"/>
</validation>
</datasource>
<drivers>
<driver name="sybase" module="com.sybase">
<xa-datasource-class>com.sybase.jdbc4.jdbc.SybXADataSource</xa-datasource-class>
</driver>
</drivers>
</datasources>

```

例: Sybase JDBC ドライバーの **module.xml** ファイル

```

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

```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Sybase JDBC ドライバーをコアモジュールとして追加します。

```

module add --name=com.sybase --resources=/path/to/jconn4.jar --
dependencies=javax.api,javax.transaction.api

```

## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. Sybase JDBC ドライバーを登録します。

```

/subsystem=datasources/jdbc-driver=sybase:add(driver-name=sybase,driver-module-
name=com.sybase,driver-xa-datasource-class-
name=com.sybase.jdbc4.jdbc.SybXADataSource)

```

## 3. Sybase データソースを追加します。

```
data-source add --name=SybaseDB --jndi-name=java:jboss/SybaseDB --driver-
name=sybase --connection-url=jdbc:sybase:Tds:localhost:5000/DATABASE?
JCONNECT_VERSION=6 --user-name=admin --password=admin --validate-on-match=true -
-background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter
```

## 12.15.12. Sybase XA データソースの例

以下は、XA データソースプロパティ、基本のセキュリティー、およびバリデーションオプションが含まれる Sybase XA データソースの設定例になります。

例: Sybase XA データソースの設定

```
<datasources>
  <xa-datasource jndi-name="java:jboss/SybaseXADS" pool-name="SybaseXADS">
    <xa-datasource-property name="ServerName">
      localhost
    </xa-datasource-property>
    <xa-datasource-property name="DatabaseName">
      mydatabase
    </xa-datasource-property>
    <xa-datasource-property name="PortNumber">
      4100
    </xa-datasource-property>
    <xa-datasource-property name="NetworkProtocol">
      Tds
    </xa-datasource-property>
    <driver>sybase</driver>
    <xa-pool>
      <is-same-rm-override>>false</is-same-rm-override>
    </xa-pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter"/>
    </validation>
    </xa-datasource>
  </drivers>
  <driver name="sybase" module="com.sybase">
    <xa-datasource-class>com.sybase.jdbc4.jdbc.SybXADataSource</xa-datasource-class>
  </driver>
</drivers>
</datasources>
```

例: Sybase JDBC ドライバーの **module.xml** ファイル

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

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. Sybase JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=com.sybase --resources=/path/to/jconn4.jar --
dependencies=javax.api,javax.transaction.api
```

### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. Sybase JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=sybase:add(driver-name=sybase,driver-module-
name=com.sybase,driver-xa-datasource-class-
name=com.sybase.jdbc4.jdbc.SybXADataSource)
```

3. Sybase XA データソースを追加します。

```
xa-data-source add --name=SybaseXADS --jndi-name=java:jboss/SybaseXADS --driver-
name=sybase --user-name=admin --password=admin --validate-on-match=true --
background-validation=false --background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter --same-rm-
override=false --xa-datasource-properties=
{"ServerName"=>"localhost","DatabaseName"=>"mydatabase","PortNumber"=>"4100","Netwo
rkProtocol"=>"Tds"}
```

### 12.15.13. MariaDB データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる MariaDB データソースの設定例になります。

例: MariaDB データソースの設定

```
<datasources>
<datasource jndi-name="java:jboss/MariaDBDS" pool-name="MariaDBDS">
  <connection-url>jdbc:mariadb://localhost:3306/jbossdb</connection-url>
  <driver>mariadb</driver>
  <security>
    <user-name>admin</user-name>
    <password>admin</password>
  </security>
  <validation>
    <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
    <validate-on-match>true</validate-on-match>
    <background-validation>>false</background-validation>
    <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
  </validation>
</datasource>
<drivers>
  <driver name="mariadb" module="org.mariadb">
    <driver-class>org.mariadb.jdbc.Driver</driver-class>
    <xa-datasource-class>org.mariadb.jdbc.MySQLDataSource</xa-datasource-class>
  </driver>
</drivers>
</datasources>
```

例: MariaDB JDBC ドライバーの **module.xml** ファイル

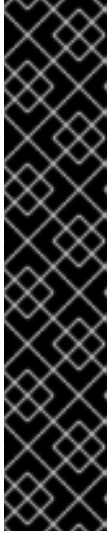
```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="org.mariadb">
  <resources>
    <resource-root path="mariadb-java-client-1.2.3.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MariaDB JDBC ドライバーをコアモジュールとして追加します。

```
module add --name=org.mariadb --resources=/path/to/mariadb-java-client-1.2.3.jar --
dependencies=javax.api,javax.transaction.api
```



## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. MariaDB JDBC ドライバーを登録します。

```
/subsystem=datasources/jdbc-driver=mariadb:add(driver-name=mariadb,driver-module-name=org.mariadb,driver-xa-datasource-class-name=org.mariadb.jdbc.MySQLDataSource,driver-class-name=org.mariadb.jdbc.Driver)
```

3. MariaDB データソースを追加します。

```
data-source add --name=MariaDBDS --jndi-name=java:jboss/MariaDBDS --driver-name=mariadb --connection-url=jdbc:mariadb://localhost:3306/jbossdb --user-name=admin -password=admin --validate-on-match=true --background-validation=false --valid-connection-checker-class-name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --exception-sorter-class-name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter
```

### 12.15.14. MariaDB XA データソースの例

以下は、XA データソースプロパティ、基本のセキュリティー、およびバリデーションオプションが含まれる MariaDB XA データソースの設定例になります。

例: MariaDB XA データソースの設定

```
<datasources>
<xa-datasource jndi-name="java:jboss/MariaDBXADS" pool-name="MariaDBXADS">
  <xa-datasource-property name="ServerName">
    localhost
  </xa-datasource-property>
  <xa-datasource-property name="DatabaseName">
    mariadbdb
  </xa-datasource-property>
  <driver>mariadb</driver>
  <security>
    <user-name>admin</user-name>
    <password>admin</password>
  </security>
  <validation>
    <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
  </validation>
</xa-datasource>
</datasources>
```

```

<validate-on-match>true</validate-on-match>
<background-validation>false</background-validation>
<exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
</validation>
</xa-datasource>
<drivers>
<driver name="mariadb" module="org.mariadb">
<driver-class>org.mariadb.jdbc.Driver</driver-class>
<xa-datasource-class>org.mariadb.jdbc.MySQLDataSource</xa-datasource-class>
</driver>
</drivers>
</datasources>

```

例: MariaDB JDBC ドライバーの **module.xml** ファイル

```

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

```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MariaDB JDBC ドライバーをコアモジュールとして追加します。

```

module add --name=org.mariadb --resources=/path/to/mariadb-java-client-1.2.3.jar --
dependencies=javax.api,javax.transaction.api

```

## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. MariaDB JDBC ドライバーを登録します。



```
/subsystem=datasources/jdbc-driver=mariadb:add(driver-name=mariadb,driver-module-
name=org.mariadb,driver-xa-datasource-class-name=org.mariadb.jdbc.MySQLDataSource,
driver-class-name=org.mariadb.jdbc.Driver)
```

3. MariaDB XA データソースを追加します。

```
xa-data-source add --name=MariaDBXADS --jndi-name=java:jboss/MariaDBXADS --driver-
name=mariadb --user-name=admin --password=admin --validate-on-match=true --
background-validation=false --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --
exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter --xa-
datasource-properties={"ServerName"=>"localhost","DatabaseName"=>"mariadbdb"}
```

### 12.15.15. MariaDB Galera Cluster データソースの例

以下は、接続情報、基本のセキュリティー、およびバリデーションオプションが含まれる MariaDB Galera Cluster データソースの設定例になります。



#### 警告

MariaDB Galera Cluster は XA トランザクションをサポートしません。

例: MariaDB Galera Cluster データソースの設定

```
<datasources>
  <datasource jndi-name="java:jboss/MariaDBGaleraClusterDS" pool-
name="MariaDBGaleraClusterDS">
    <connection-url>jdbc:mariadb://192.168.1.1:3306,192.168.1.2:3306/jbosssdb</connection-url>
    <driver>mariadb</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker"/>
      <validate-on-match>true</validate-on-match>
      <background-validation>false</background-validation>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"/>
    </validation>
  </datasource>
</drivers>
<driver name="mariadb" module="org.mariadb">
  <driver-class>org.mariadb.jdbc.Driver</driver-class>
  <xa-datasource-class>org.mariadb.jdbc.MySQLDataSource</xa-datasource-class>
```

```

</driver>
</drivers>
</datasources>

```

例: MariaDB JDBC ドライバーの **module.xml** ファイル

```

<?xml version='1.0' encoding='UTF-8'?>
<module xmlns="urn:jboss:module:1.1" name="org.mariadb">
  <resources>
    <resource-root path="mariadb-java-client-1.5.4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>

```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MariaDB JDBC ドライバーをコアモジュールとして追加します。

```

module add --name=org.mariadb --resources=/path/to/mariadb-java-client-1.5.4.jar --
dependencies=javax.api,javax.transaction.api

```

### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. MariaDB JDBC ドライバーを登録します。

```

/subsystem=datasources/jdbc-driver=mariadb:add(driver-name=mariadb,driver-module-
name=org.mariadb,driver-xa-datasource-class-name=org.mariadb.jdbc.MySQLDataSource,
driver-class-name=org.mariadb.jdbc.Driver)

```

3. MariaDB Galera Cluster データソースを追加します。

```

data-source add --name=MariaDBGaleraClusterDS --jndi-
name=java:jboss/MariaDBGaleraClusterDS --driver-name=mariadb --connection-
url=jdbc:mariadb://192.168.1.1:3306,192.168.1.2:3306/jbossdb --user-name=admin --
password=admin --validate-on-match=true --background-validation=false --valid-connection-
checker-class-

```

```
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker --  
exception-sorter-class-  
name=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter
```

## 第13章 AGROAL でのデータソース管理

### 13.1. AGROAL データソースサブシステム

新しい **datasources-agroal** サブシステムが JBoss EAP 7.2 に導入されました。これは、Agroal プロジェクトによる、新しい高パフォーマンスな直接接続プールです。現在の JCA ベースの **datasources** サブシステムの代わりに使用することができます。

**datasources-agroal** サブシステムはデフォルトでは使用できません。この新しい実装を **有効** にして使用する必要があります。



#### 重要

**datasources-agroal** サブシステムはテクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

### 13.2. AGROAL データソースサブシステムの有効化

**datasources-agroal** サブシステムは、デフォルトの JBoss EAP 設定では有効になっていません。以下の管理 CLI コマンドを使用して有効にします。

1. 拡張を追加します。

```
/extension=org.wildfly.extension.datasources-agroal:add
```

2. サブシステムを追加します。

```
/subsystem=datasources-agroal:add
```

3. 変更を反映するためにサーバーをリロードします。

```
reload
```

これにより、サーバー設定ファイルに以下の XML が追加されます。

```
<server xmlns="urn:jboss:domain:8.0">
  <extensions>
    ...
    <extension module="org.wildfly.extension.datasources-agroal"/>
    ...
  </extensions>
  ...
  <subsystem xmlns="urn:jboss:domain:datasources-agroal:1.0"/>
  ...
</server>
```

## 13.3. AGROAL データソースに対するコアモジュールとしての JDBC ドライバーのインストール

JBoss EAP でアプリケーションが使用する Agroal データソースを定義する前に、最初に適切な JDBC ドライバーをインストールする必要があります。

Agroal データソースに対して JDBC ドライバーをコアモジュールとしてインストールするには、最初に [JDBC ドライバーをコアモジュールとして追加](#) し、`datasources-agroal` サブシステムで **JDBC** ドライバーを登録する必要があります。

### 13.3.1. JDBC ドライバーをコアモジュールとして追加

以下の手順に従うと、管理 CLI を使用して JDBC ドライバーをコアモジュールとしてインストールすることができます。

1. JDBC ドライバーをダウンロードします。  
データベースのベンダーから適切な JDBC ドライバーをダウンロードします。一般的なデータベースの JDBC ドライバーをダウンロードできる場所については、「[JDBC ドライバーのダウンロードできる場所](#)」を参照してください。

JDBC ドライバーの JAR ファイルが ZIP または TAR アーカイブ内に含まれている場合は、必ずそのアーカイブを展開してください。

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

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

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

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

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

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

#### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

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

次に、アプリケーションデータソースによって参照されるよう、JDBC ドライバーとして登録する必要があります。

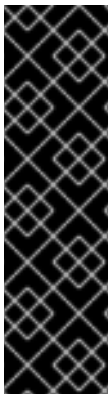
### 13.3.2. Agroal データソースの JDBC ドライバーの登録

ドライバーが **コアモジュールとしてインストール** されたら、以下の管理 CLI コマンドを使用して JDBC ドライバーとして登録する必要があります。管理対象ドメインを実行している場合は、コマンドの前に **/profile=PROFILE\_NAME** を付けてください。

```
/subsystem=datasources-
agroal/driver=DRIVER_NAME:add(module=MODULE_NAME,class=CLASS_NAME)
```

**CLASS\_NAME** は、非 XA データソースの場合は **java.sql.Driver** または **javax.sql.DataSource**、XA データソースの場合は **javax.sql.XADataSource** を実装する完全修飾クラス名である必要があります。

## 13.4. AGROAL 非 XA データソースの設定



### 重要

Agroal データソースの使用は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

### 13.4.1. Agroal データソースの作成

以下の管理 CLI コマンドは Agroal データソースを作成します。管理対象ドメインを実行している場合は、コマンドの前に **/profile=PROFILE\_NAME** を付けてください。

```
/subsystem=datasources-agroal/datasource=DATASOURCE_NAME:add(jndi-
name=JNDI_NAME,connection-factory={driver=DRIVER_NAME,url=URL},connection-pool={max-
size=MAX_POOL_SIZE})
```

利用できる Agroal データソース属性の完全リストは、「[Agroal データソース属性](#)」を参照してください。

Agroal データソース設定の例は、「[MySQL Agroal データソースの例](#)」を参照してください。

### 13.4.2. Agroal データソースの削除

以下の管理 CLI コマンドは Agroal データソースを削除します。管理対象ドメインを実行している場合は、コマンドの前に **/profile=PROFILE\_NAME** を付けてください。

```
/subsystem=datasources-agroal/datasource=DATASOURCE_NAME:remove
```

## 13.5. AGROAL XA データソースの設定



### 重要

Agroal データソースの使用は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

### 13.5.1. Agroal XA データソースの作成

以下の管理 CLI コマンドは Agroal XA データソースを作成します。管理対象ドメインを実行している場合は、コマンドの前に **/profile=PROFILE\_NAME** を付けてください。

```
/subsystem=datasources-agroal/xa-datasource=XA_DATASOURCE_NAME:add(jndi-name=JNDI_NAME,connection-factory={driver=DRIVER_NAME,connection-properties={ServerName=HOST_NAME,PortNumber=PORT,DatabaseName=DATABASE_NAME}},connection-pool={max-size=MAX_POOL_SIZE})
```

利用できる Agroal データソース属性の完全リストは、「[Agroal データソース属性](#)」を参照してください。

Agroal XA データソース設定の例は、「[MySQL Agroal XA データソースの例](#)」を参照してください。

### 13.5.2. Agroal XA データソースの削除

以下の管理 CLI コマンドは Agroal XA データソースを削除します。管理対象ドメインを実行している場合は、コマンドの前に **/profile=PROFILE\_NAME** を付けてください。

```
/subsystem=datasources-agroal/xa-datasource=DATASOURCE_NAME:remove
```

## 13.6. AGROAL データソースの設定例

### 13.6.1. MySQL Agroal データソースの例

以下は、接続設定および基本のセキュリティー設定が含まれる、MySQL Agroal データソースの設定例になります。

例: MySQL Agroal データソース設定

```
<subsystem xmlns="urn:jboss:domain:databases-agroal:1.0">
  <datasource name="ExampleAgroalDS" jndi-name="java:jboss/databases/ExampleAgroalDS">
    <connection-factory driver="mysql" url="jdbc:mysql://localhost:3306/jbossdb" username="admin"
password="admin"/>
    <connection-pool max-size="30"/>
  </datasource>
</subsystem>
```

```

    <driver name="mysql" module="com.mysql" class="com.mysql.cj.jdbc.Driver"/>
  </drivers>
</subsystem>

```

例: MySQL JDBC ドライバー `module.xml` ファイル

```

<?xml version='1.0' encoding='UTF-8'?>

<module xmlns="urn:jboss:module:1.1" name="com.mysql">

  <resources>
    <resource-root path="mysql-connector-java-8.0.12.jar"/>
  </resources>

  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>

```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MySQL JDBC ドライバーをコアモジュールとして追加します。

```

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

```

### 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. MySQL JDBC ドライバーを登録します。

```

/subsystem=datasources-
agroal/driver=mysql:add(module=com.mysql,class=com.mysql.cj.jdbc.Driver)

```

3. MySQL データソースを追加します。

```

/subsystem=datasources-agroal/datasource=ExampleAgroalDS:add(jndi-
name=java:jboss/datasources/ExampleAgroalDS,connection-factory=
{driver=mysql,url=jdbc:mysql://localhost:3306/jbossdb,username=admin,password=admin},conn

```



```
ection-pool={max-size=30})
```

### 13.6.2. MySQL Agroal XA データソースの例

以下は、XA データソースプロパティおよび基本のセキュリティ設定が含まれる、MySQL Agroal XA データソースの設定例になります。

例: MySQL Agroal XA データソース設定

```
<subsystem xmlns="urn:jboss:domain:datasources-agroal:1.0">
  <xa-datasource name="ExampleAgroalXADS" jndi-
name="java:jboss/datasources/ExampleAgroalXADS">
    <connection-factory driver="mysqlXA" username="admin" password="admin">
      <connection-properties>
        <property name="ServerName" value="localhost"/>
        <property name="PortNumber" value="3306"/>
        <property name="DatabaseName" value="jbossdb"/>
      </connection-properties>
    </connection-factory>
    <connection-pool max-size="30"/>
  </xa-datasource>
  <drivers>
    <driver name="mysqlXA" module="com.mysql" class="com.mysql.cj.jdbc.MysqlXADataSource"/>
  </drivers>
</subsystem>
```

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

```
<?xml version='1.0' encoding='UTF-8'?>

<module xmlns="urn:jboss:module:1.1" name="com.mysql">

  <resources>
    <resource-root path="mysql-connector-java-8.0.12.jar"/>
  </resources>

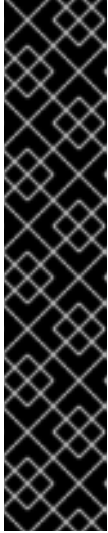
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

管理 CLI コマンドの例

以下の管理 CLI コマンドを使用すると、この設定例を実現できます。

1. MySQL JDBC ドライバーをコアモジュールとして追加します。

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



## 重要

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

テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

2. MySQL XA JDBC ドライバーを登録します。

```
/subsystem=datasources-  
agroal/driver=mysqlXA:add(module=com.mysql,class=com.mysql.cj.jdbc.MySQLXADataSource)
```

3. MySQL XA データソースを追加します。

```
/subsystem=datasources-agroal/xa-datasource=ExampleAgroalXADS:add(jndi-  
name=java:jboss/datasources/ExampleAgroalXADS,connection-factory=  
{driver=mysqlXA,connection-properties=  
{ServerName=localhost,PortNumber=3306,DatabaseName=jbossdb},username=admin,password=admin},connection-pool={max-size=30})
```

## 第14章 TRANSACTIONS サブシステムの設定

**transactions** サブシステムは、タイムアウト値、トランザクションロギング、統計の収集、Java Transaction Service (JTS) を使用するかどうかなど、トランザクションマネージャー (TM) オプションの設定を可能にします。JBoss EAP は、Narayana フレームワークを使用してトランザクションサービスを提供します。このフレームワークは、JTA、JTS、Web Service トランザクション、REST トランザクション、ソフトウェアトランザクションメモリー (STM)、XATMI/TX 仕様など、標準ベースの幅広いトランザクションプロトコルのサポートを利用します。

詳細は、『[JBoss EAP でのトランザクションの管理](#)』を参照してください。

## 第15章 ORB 設定

### 15.1. COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

Common Object Request Broker Architecture (CORBA) は、アプリケーションとサービスが複数の互換性がない言語で記述され、異なるプラットフォームでホストされる場合でも、アプリケーションとサービスが連携することを可能にする標準です。CORBA のリクエストは Object Request Broker (ORB) というサーバーサイドコンポーネントが仲介します。JBoss EAP は、Open JDK ORB コンポーネントを用いて ORB インスタンスを提供します。

ORB は Java Transaction Service (JTS) トランザクションに対して内部的に使用され、ユーザー独自のアプリケーションが使用することもできます。

### 15.2. JTS トランザクション用 ORB の設定

JBoss EAP のデフォルトインストールでは、トランザクションの ORB (Object Request Broker: オブジェクトリクエストブローカー) のサポートが無効になっています。管理 CLI または管理コンソールを使用して **iiop-openjdk** サブシステムの ORB を設定できます。



#### 注記

**iiop-openjdk** サブシステムを利用できるのは、管理対象ドメインで **full** または **full-ha** プロファイルを使用している場合や、スタンドアロンサーバーの **standalone-full.xml** または **standalone-full-ha.xml** 設定ファイルを使用している場合です。

**iiop-openjdk** サブシステムで使用できる設定オプションの一覧は、「[IIOP サブシステムの属性](#)」を参照してください。

#### 管理 CLI を使用した ORB の設定

管理 CLI を使用して ORB を設定できます。これは、JTS と使用するために行う ORB の最低限の設定です。

以下の管理対象ドメインの管理 CLI コマンドは、**full** プロファイルを使用して設定できます。必要な場合は設定に応じてプロファイルを変更してください。スタンドアロンサーバーを使用している場合は、コマンドの **/profile=full** 部分を省略してください。

セキュリティーインターセプターの有効化  
値を **identity** に設定し、**security** 属性を有効にします。

```
/profile=full/subsystem=iiop-openjdk:write-attribute(name=security,value=identity)
```

#### IIOP サブシステムでのトランザクションの有効化

JTS に対して ORB を有効にするには、**transactions** 属性の値をデフォルトの **spec** ではなく **full** に設定します。

```
/profile=full/subsystem=iiop-openjdk:write-attribute(name=transactions, value=full)
```

#### Transactions サブシステムでの JTS の有効化

```
/profile=full/subsystem=transactions:write-attribute(name=jts,value=true)
```



## 注記

JTS をアクティベートするにはリロードでは不十分なため、サーバーを再起動する必要があります。

### 管理コンソールを使用した ORB の設定

1. 管理コンソールの上部で、**Configuration** タブを選択します。管理対象ドメインを使用する場合は、変更するプロファイルを選択する必要があります。
2. **Subsystems** → **IIOp (OpenJDK)** と選択し、**表示** をクリックします。
3. **編集** をクリックし、必要に応じて属性を変更します。
4. **保存** をクリックして変更を保存します。

## 15.3. ELYTRON サブシステムで SSL/TLS を使用するよう IIOp を設定

SSL/TLS/ を使用するよう **iiop-openjdk** サブシステムを設定し、クライアントとサーバー間の通信をセキュア化することができます。**elytron** サブシステムおよびレガシーの **security** サブシステムは、**iiop-openjdk** サブシステムや JBoss EAP 内の他のサブシステムに SSL/TLS を設定するために必要なコンポーネントを提供します。以下の手順にしたがって、SSL/TLS に **elytron** サブシステムを使用するよう **iiop-openjdk** サブシステムを設定します。

1. 以下の管理 CLI コマンドを使用して、**iiop-openjdk** サブシステムの現在のレガシー SSL/TLS 設定を表示します。

```
/subsystem=iiop-openjdk:read-attribute(name=security-domain)
{
  "outcome" => "success",
  "result" => "iiopSSLSecurityDomain"
}
```

**iiop-openjdk** サブシステムは、SSL/TLS にレガシーの **security** サブシステムか **elytron** サブシステムのいずれかを使用する必要があります。両方のサブシステムを同時に使用することはできません。上記のコマンドは、**iiop-openjdk** サブシステムが SSL/TLS の処理にレガシーのセキュリティドメインを使用していることを示しています。SSL/TLS に **elytron** サブシステムを使用するよう **iiop-openjdk** サブシステムを設定する前に、以下の参照を削除する必要があります。

```
/subsystem=iiop-openjdk:undefine-attribute(name=security-realm)
```

**iiop-openjdk** の **security-domain** 属性が定義されていない場合は、手順を続行できません。

2. **server-ssl-context** を作成します。  
**iiop-openjdk** サブシステムで SSL/TLS を使用するには、**server-ssl-context** を定義する必要があります。JBoss EAP は、サーバーへの SSL/TLS 接続を確立するときに、**server-ssl-context** によって提供される設定を使用します。**server-ssl-context** の作成に関する詳細は、『[How to Configure Server Security](#)』の「[Enable One-way SSL/TLS for Applications using the Elytron Subsystem](#)」を参照してください。
3. **client-ssl-context** を作成します。  
**iiop-openjdk** サブシステムで SSL/TLS を使用するには、**client-ssl-context** を定義する必要があります。JBoss EAP は、クライアントへの SSL/TLS 接続を確立するときに、**client-ssl-context** によって提供される設定を使用します。**client-ssl-context** の作成に関する詳細は、

『[How to Configure Server Security](#)』の「[Using a client-ssl-context](#)」を参照してください。

4. **client-ssl-context** および **server-ssl-context** を使用するよう、**iiop-openjdk** サブシステムを設定します。

#### 例: client-ssl-context および server-ssl-context の設定

```
batch

/subsystem=iiop-openjdk:write-attribute(name=client-ssl-context,value=iiopClientSSC)

/subsystem=iiop-openjdk:write-attribute(name=server-ssl-context,value=iiopServerSSC)

run-batch

reload
```

5. **iiop-openjdk** サブシステムを接続先および接続元とする接続の設定  
以下の属性を調整すると、**iiop-openjdk** サブシステムを接続先および接続元とする接続を確立するときに SSL/TLS 接続を要求するかどうかを示すことができます。
  - **iiop-openjdk** サブシステムで SSL のサポートを有効にするには、**support-ssl** を **true** に設定します。デフォルトは **false** です。
  - **iiop-openjdk** サブシステムからの SSL/TLS 接続を要求するには、**client-requires-ssl** を **true** に設定します。デフォルトは **false** です。
  - **iiop-openjdk** サブシステムへの SSL/TLS 接続を要求するには、**server-requires-ssl** を **true** に設定します。デフォルトは **false** です。これを **true** に設定すると、非 SSL IIOP ソケットへの接続をブロックすることに注意してください。
  - **socket-binding** を調整するには、**ssl-socket-binding** を希望のバインディングに設定します。デフォルトは **iiop-ssl** です。

#### 例: IIOP を接続先および接続元とする SSL/TLS 接続の設定

```
/subsystem=iiop-openjdk:write-attribute(name=support-ssl,value=true)

/subsystem=iiop-openjdk:write-attribute(name=client-requires-ssl,value=true)

/subsystem=iiop-openjdk:write-attribute(name=server-requires-ssl,value=true)

/subsystem=iiop-openjdk:write-attribute(name=ssl-socket-binding,value=iiop-ssl)

reload
```

## 第16章 JAVA CONNECTOR ARCHITECTURE (JCA) の管理

### 16.1. JAVA CONNECTOR ARCHITECTURE (JCA)

Java EE Connector Architecture (JCA) は外部にある異種のエンタープライズ情報システム (EIS) に対して Java EE システムの標準アーキテクチャーを定義します。EIS の例として、エンタープライズリソースプランニング (ERP) システム、メインフレームトランザクション処理 (TP)、データベース、メッセージングシステムなどが挙げられます。[リソースアダプター](#)は Java EE Connector API アーキテクチャーを実装するコンポーネントです。

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

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

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

### 16.2. リソースアダプター

リソースアダプターは、Java Connector Architecture (JCA) 仕様を使用して Java EE アプリケーションとエンタープライズ情報システム (EIS) との間の通信を提供するデプロイ可能な Java EE コンポーネントです。EIS ベンダーの製品と Java EE アプリケーションの統合を容易にするため、リソースアダプターは通常 EIS ベンダーによって提供されます。

エンタープライズ情報システムは、組織内における他のあらゆるソフトウェアシステムのことです。例としては、エンタープライズリソースプランニング (ERP) システム、データベースシステム、電子メールサーバー、商用メッセージングシステムなどが挙げられます。

リソースアダプターは、JBoss EAP にデプロイできる Resource Adapter Archive (RAR) ファイルでパッケージ化されます。また、RAR ファイルは、Enterprise Archive (EAR) デプロイメントにも含めることができます。

リソースアダプター自体は、IronJacamar プロジェクトによって提供される **resource-adapters** サブシステム内に定義されます。

### 16.3. JCA サブシステムの設定

**jca** サブシステムは、JCA コンテナおよびリソースアダプターデプロイメントの一般的な設定を制御します。管理コンソールまたは管理 CLI を使用して **jca** サブシステムを設定できます。

設定する主な JCA 要素は次のとおりです。

- [アーカイブバリデーション](#)

- Bean バリデーション
- ワークマネージャー
- 分散ワークマネージャー
- ブートストラップコンテキスト
- キャッシュ接続マネージャー

### 管理コンソールでの JCA の設定

管理コンソールで **jca** サブシステムを設定するには、**Configuration** → **Subsystems** → **JCA** と選択し、**表示** をクリックします。次に、該当するタブを選択します。

- **Configuration:** キャッシュ接続マネージャー、アーカイブバリデーション、および Bean バリデーションの設定が含まれます。これらの項目は独自のタブに含まれます。設定を変更するには、タブを開き、**Edit** リンクをクリックします。
- **Bootstrap Context:** 設定されたブートストラップコンテキストのリストが含まれます。新しいブートストラップコンテキストオブジェクトを追加、削除、および設定できます。各ブートストラップコンテキストをワークマネージャーに割り当てる必要があります。
- **Workmanager:** 設定されたワークマネージャーのリストが含まれます。新しいワークマネージャーを追加および削除でき、スレッドプールをここで設定できます。各ワークマネージャーは短時間実行されるスレッドプールを1つ持つことができ、任意で長時間実行されるスレッドプールを1つ持つことができます。スレッドプール属性を設定するには、選択したワークマネージャーの **Thread Pools** をクリックします。

### 管理 CLI での JCA の設定

`/subsystem=jca` アドレスから管理 CLI で **jca** サブシステムを設定できます。管理対象ドメインでは、コマンドの前に `/profile=PROFILE_NAME` を追加する必要があります。



#### 注記

以下のセクションの表には、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を `EAP_HOME/docs/schema/wildfly-jca_5_0.xsd` のスキーマ定義ファイルで確認してください。

### アーカイブバリデーション

デプロイメントユニットでアーカイブバリデーションを実行するかどうかを決定します。以下の表はアーカイブバリデーションに設定できる属性を表しています。

表16.1 アーカイブバリデーション属性

属性	デフォルト値	説明
enabled	true	アーカイブバリデーションが有効であるかどうかを指定します。
fail-on-error	true	アーカイブバリデーションのエラーレポートによってデプロイメントが失敗するかどうかを指定します



属性	デフォルト値	説明
fail-on-warn	false	アーカイブバリデーションの警告レポートによってデプロイメントが失敗するかどうかを指定します。

アーカイブバリデーションが有効な状態で、アーカイブが JCA 仕様を正しく実装しない場合、デプロイメント中に問題を説明するエラーメッセージが表示されます。例を以下に示します。

Severity: ERROR

Section: 19.4.2

Description: A ResourceAdapter must implement a "public int hashCode()" method.

Code: com.mycompany.myproject.ResourceAdapterImpl

Severity: ERROR

Section: 19.4.2

Description: A ResourceAdapter must implement a "public boolean equals(Object)" method.

Code: com.mycompany.myproject.ResourceAdapterImpl

アーカイブバリデーションが指定されていない場合は、アーカイブバリデーションが指定されているとみなされ、**enabled** 属性のデフォルトが **true** に設定されます。

### Bean バリデーション

この設定は、[JSR-303](#) で定義されている Bean バリデーションがデプロイメントユニットで実行されるかどうかを決定します。以下の表は Bean バリデーションに設定できる属性について表しています。

表16.2 Bean バリデーションの属性

属性	デフォルト値	説明
enabled	true	Bean バリデーションが有効であるかどうかを指定します。

Bean バリデーションが指定されていない場合は、Bean バリデーションが指定されているとみなされ、**enabled** 属性のデフォルトが **true** に設定されます。

### ワークマネージャー

ワークマネージャーには次の 2 種類があります。

#### デフォルトワークマネージャー

デフォルトのワークマネージャーおよびそのスレッドプール。

#### カスタムワークマネージャー

カスタムワークマネージャーの定義およびそのスレッドプール。

ワークマネージャーに設定できる属性は下表のとおりです。

表16.3 ワークマネージャーの属性

属性	説明
name	ワークマネージャーの名前を指定します。

属性	説明
elytron-enabled	この属性は、 <b>workmanager</b> の Elytron セキュリティーを有効にします。

ワークマネージャーには以下の子要素もあります。

表16.4 ワークマネージャーの子要素

子要素	説明
short-running-threads	標準の Work インスタンスのスレッドプール。ワークマネージャーごとに短時間実行されるスレッドプールが1つあります。
long-running-threads	<b>LONG_RUNNING</b> ヒントを設定する JCA 1.7 Work インスタンスのスレッドプール。各ワークマネージャーはオプションの長期スレッドプールを1つ持てます。

ワークマネージャーのスレッドプールに設定できる属性は下表のとおりです。

表16.5 スレッドプールの属性

属性	説明
allow-core-timeout	コアスレッドがタイムアウトするかどうかを決定するブール値の設定。デフォルト値は <b>false</b> です。
core-threads	コアスレッドプールのサイズ。スレッドプールの最大サイズ以下である必要があります。
handoff-executor	タスクが受け入れられない場合、タスクを委譲するエグゼキューター。指定されていない場合、受け入れられないタスクは警告なしに破棄されます。
keepalive-time	ワーク実行後にスレッドプールが保持される期間を指定します。
max-threads	スレッドプールの最大サイズ。
name	スレッドプールの名前を指定します。
queue-length	キューの最大長。
thread-factory	スレッドファクトリーへの参照。

### 分散ワークマネージャー

分散ワークマネージャーは、別のワークマネージャーインスタンスでワークの実行スケジュールを変更できるワークマネージャーです。

以下の管理 CLI コマンドの例は、分散ワークマネージャーを設定します。スタンドアロンサーバーの **standalone-ha.xml** または **standalone-full-ha.xml** 設定ファイルなど、高可用性を提供する設定を使用する必要があります。ご注意ください。

## 例: 分散ワークマネージャーの設定

```
batch
/subsystem=jca/distributed-workmanager=myDistWorkMgr:add(name=myDistWorkMgr)
/subsystem=jca/distributed-workmanager=myDistWorkMgr/short-running-
threads=myDistWorkMgr:add(queue-length=10,max-threads=10)
/subsystem=jca/bootstrap-
context=myCustomContext:add(name=myCustomContext,workmanager=myDistWorkMgr)
run-batch
```



## 注記

**short-running-threads** 要素の名前は **distributed-workmanager** 要素の名前と同じである必要があります。

以下の表は、分散ワークマネージャーに設定できる属性を表しています。

表16.6 分散ワークマネージャーの属性

属性	説明
elytron-enabled	ワークマネージャーの Elytron セキュリティーを有効にします。
name	分散ワークマネージャーの名前。
policy	ワークインスタンスをいつ再分散するかを決定します。使用できる値は次のとおりです。 <ul style="list-style-type: none"> <li>● <b>NEVER</b> - ワークインスタンスを別のノードに分散しません。</li> <li>● <b>ALWAYS</b> - 常にワークインスタンスを別のノードに分散します。</li> <li>● <b>WATERMARK</b> - 現在のノードで使用できる空きのワーカースレッドの数に応じてワーカーインスタンスを別のノードに分散します。</li> </ul>
policy-options	ポリシーのキーバリューペアのオプションリスト。 <b>WATERMARK</b> ポリシーを使用する場合、 <b>watermark</b> ポリシーオプションを使用して、ワークが分散される空きスレッドの数を指定します。例を以下に示します。 <pre>/subsystem=jca/distributed-workmanager=myDistWorkMgr:write-attribute(name=policy-options,value={watermark=3})</pre>
selector	ワークインスタンスを再分散するネットワークのノードを決定します。使用できる値は次のとおりです。 <ul style="list-style-type: none"> <li>● <b>FIRST_AVAILABLE</b> - リストの最初に利用できるノードを選択します。</li> <li>● <b>PING_TIME</b> - ping の時間が最も短いノードを選択します。</li> <li>● <b>MAX_FREE_THREADS</b> - 空きワーカースレッドの数が最も多いノードを選択します。</li> </ul>

属性	説明
selector-options	セレクターのキーバリュペアのオプションリスト。

分散ワークマネージャーには以下の子要素もあります。

表16.7 分散ワークマネージャーの子要素

子要素	説明
long-running-threads	<b>LONG_RUNNING</b> ヒントを設定するワークインスタンスのスレッドプール。各分散ワークマネージャーはオプションで長時間実行スレッドプールを持つことができます。
short-running-threads	標準ワークインスタンスのスレッドプール。各分散ワークマネージャーには短期間実行スレッドプールがある必要があります。

### ブートストラップコンテキスト

カスタムのブートストラップコンテキストを定義するために使用されます。以下の表は、ブートストラップコンテキストに設定できる属性を表しています。

表16.8 ブートストラップコンテキストの属性

属性	説明
name	ブートストラップコンテキストの名前を指定します。
workmanager	このコンテキストに使用するワークマネージャーの名前を指定します。

### キャッシュ接続マネージャー

トランザクションの接続における接続のデバッグおよび Lazy Enlistment のサポートに使用され、アプリケーションによって使用およびリリースされるかどうかを追跡します。以下の表はキャッシュ接続マネージャーに設定できる属性を表しています。

表16.9 キャッシュ接続マネージャーの属性

属性	デフォルト値	説明
debug	false	接続を明示的に閉じるため、障害時に警告を出力します。
error	false	接続を明示的に閉じるため、障害時に例外が発生します。
ignore-unknown-connections	false	未知の接続はキャッシュされないことを指定します。
install	false	キャッシュされた接続マネージャーバルブおよびインターセプターを有効または無効にします。

## 16.4. リソースアダプターの設定

### 16.4.1. リソースアダプターのデプロイ

リソースアダプターは管理 CLI または管理コンソールを使用して他のデプロイメントと同様にデプロイできます。また、スタンドアロンサーバー実行時にアーカイブをデプロイメントディレクトリーにコピーして、デプロイメントスキャナーによって検出されるようにすることもできます。

管理 CLI を使用したリソースアダプターのデプロイ

リソースアダプターをスタンドアロンサーバーへデプロイするには、以下の管理 CLI コマンドを実行します。

```
deploy /path/to/resource-adapter.rar
```

リソースアダプターを管理対象ドメインのすべてのサーバーグループにデプロイするには、以下の管理 CLI コマンドを入力します。

```
deploy /path/to/resource-adapter.rar --all-server-groups
```

管理コンソールを使用したリソースアダプターのデプロイ

1. 管理コンソールにログインし、**Deployments** タブをクリックします。
2. 追加 (+) ボタンをクリックします。管理対象ドメインでは最初に **Content Repository** を選択する必要があります。
3. デプロイメントのアップロード オプションを選択します。
4. リソースアダプターアーカイブを閲覧し、次へ をクリックします。
5. アップロードを確認してから 完了 をクリックします。
6. 管理対象ドメインでは、デプロイメントを該当するサーバーグループにデプロイし、デプロイメントを有効にします。

デプロイメントスキャナーを使用したリソースアダプターのデプロイ

リソースアダプターを手作業でスタンドアロンサーバーにデプロイするには、リソースアダプターアーカイブをサーバーデプロイメントディレクトリー (例: **EAP\_HOME/standalone/deployments/**) にコピーします。これにより、デプロイメントスキャナーによって検出され、デプロイされます。



#### 注記

このオプションは管理対象ドメインでは使用できません。管理コンソールまたは管理 CLI を使用してリソースアダプターをサーバーグループにデプロイする必要があります。

### 16.4.2. リソースアダプターの設定

管理インターフェースを使用してリソースアダプターを設定できます。以下の例は、管理 CLI を使用してリソースアダプターを設定する方法を示しています。サポートされるプロパティーやその他の重要な情報は、リソースアダプターベンダーのマニュアルを参照してください。

リソースアダプター設定の追加

リソースアダプター設定を追加します。

■

```
/subsystem=resource-adapters/resource-adapter=eis.rar:add(archive=eis.rar, transaction-support=XATransaction)
```

リソースアダプターの設定  
必要に応じて以下を設定します。

- **config-properties** を設定します。  
**server** 設定プロパティを追加します。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/config-properties=server:add(value=localhost)
```

**port** 設定プロパティを追加します。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/config-properties=port:add(value=9000)
```

- **admin-objects** を設定します。  
管理オブジェクトを追加します。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/admin-objects=aoName:add(class-name=com.acme.eis.ra.EISAdminObjectImpl, jndi-name=java:/eis/AcmeAdminObject)
```

管理オブジェクト設定プロパティを設定します。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/admin-objects=aoName/config-properties=threshold:add(value=10)
```

- **connection-definitions** を設定します。  
管理接続ファクトリーの接続定義を追加します。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/connection-definitions=cfName:add(class-name=com.acme.eis.ra.EISManagedConnectionFactory, jndi-name=java:/eis/AcmeConnectionFactory)
```

管理接続ファクトリーの設定プロパティを設定します。

```
/subsystem=resource-adapters/resource-adapter=eis.rar/connection-definitions=cfName/config-properties=name:add(value=Acme Inc)
```

エンリストメントトレースを記録するかどうかを設定します。エンリストメントトレースの記録を有効にするには、**enlistment-trace** 属性を **true** に設定します。

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



### 警告

エンリストメントトレースを有効にすると、トランザクションエンリストメント中のエラーを容易に追跡できますが、パフォーマンスに影響します。

リソースアダプターの利用可能なすべてのオプションについては、「[リソースアダプターの属性](#)」を参照してください。

リソースアダプターのアクティベート  
リソースアダプターをアクティベートします。

```
/subsystem=resource-adapters/resource-adapter=eis.rar:activate
```



### 注記

リソースアダプターのキャパシティーポリシーを定義することも可能です。詳細は「[キャパシティーポリシー](#)」の項を参照してください。

## 16.4.3. Elytron サブシステムを使用するようリソースアダプターを設定

IronJacamar では、サーバーとリソースアダプターの間で2種類の通信が発生します。

その通信の1つは、サーバーがリソースアダプター接続を開いた場合に発生します。仕様によって定義されたとおり、コンテナ管理のサインオンでセキュアにすることができます。これには、接続を開いたときにプリンシパルと認証情報で JAAS サブジェクトをリソースアダプターへ伝搬する必要があります。このサインオンは、Elytron に託せます。

IronJacamar はセキュリティーインフローをサポートします。このメカニズムは、ワークをワークマネージャーに提出するときや、同じ JBoss EAP インスタンスにあるエンドポイントにメッセージを配信するときに、リソースアダプターがセキュリティー情報を確立できるようにします。

### コンテナ管理のサインオン

Elytron でコンテナ管理のサインオンを実現するには、**elytron-enabled** 属性を **true** に設定する必要があります。これにより、リソースアダプターへのすべての接続が Elytron によってセキュア化されます。

```
/subsystem=resource-adapters/resource-adapter=RAR_NAME/connection-  
definitions=FACTORY_NAME:write-attribute(name=elytron-enabled,value=true)
```

**resource-adapters** サブシステムの **elytron-enabled** 属性を **true** に設定すると、管理 CLI を使用して **elytron-enabled** 属性を設定できます。デフォルトではこの属性は **false** に設定されています。

**authentication-context** 属性は、サインオンの実行に使用される Elytron 認証コンテキストの名前を定義します。

Elytron の **authentication-context** 属性には1つ以上の **authentication-configuration** 要素を含めることができ、使用を希望する認証情報が含まれます。

**authentication-context** 属性が設定されていない場合、JBoss EAP は現在の **authentication-context** を使用します。これは、接続を開く呼び出し元コードによって使用される **authentication-context** です。

### 例: authentication-configuration の作成

```
/subsystem=elytron/authentication-configuration=exampleAuthConfig:add(authentication-name=sa,credential-reference={clear-text=sa})
```

### 例: 上記設定を使用した authentication-context の作成

```
/subsystem=elytron/authentication-context=exampleAuthContext:add(match-rules=[{authentication-configuration=exampleAuthConfig}])
```

#### セキュリティインフロー

また、ワークマネージャーによって実行される予定のワークを提出するときに、リソースマネージャーはセキュリティ認証情報を流入 (インフロー) することもできます。セキュリティインフローは、実行前にワークがワーク自体を認証できるようにします。認証に成功すると、提出されたワークは結果となる認証コンテキスト下で実行されます。認証に失敗すると、ワークの実行は拒否されます。

Elytron セキュリティインフローを有効にするには、リソースアダプターのワークマネージャーを設定するときに **wm-elytron-security-domain** 属性を設定します。Elytron は指定のドメインを元に認証を実行します。



#### 注記

リソースアダプターのワークマネージャーが Elytron セキュリティドメイン **wm-elytron-security-domain** を使用するよう設定された場合、参照されたワークマネージャーの **elytron-enabled** 属性を **true** に設定する必要があります。

```
/subsystem=jca/workmanager=customWM:add(name=customWM, elytron-enabled=true)
```



#### 注記

**wm-elytron-security-domain** の代わりに **wm-security-domain** 属性が使用された場合、セキュリティインフローはレガシーの **security** サブシステムによって実行されません。

以下の **jca** サブシステムの設定例では、**ra-with-elytron-security-domain** というリソースアダプターの設定を確認できます。このリソースアダプターは、Elytron セキュリティドメインの **wm-realm** を使用するようワークマネージャーセキュリティを設定します。

```
<subsystem xmlns="urn:jboss:domain:jca:5.0">
  <archive-validation enabled="true" fail-on-error="true" fail-on-warn="false"/>
  <bean-validation enabled="true"/>
  <default-workmanager>
    <short-running-threads>
      <core-threads count="50"/>
      <queue-length count="50"/>
      <max-threads count="50"/>
      <keepalive-time time="10" unit="seconds"/>
    </short-running-threads>
  </default-workmanager>
</subsystem>
```



```

<long-running-threads>
  <core-threads count="50"/>
  <queue-length count="50"/>
  <max-threads count="50"/>
  <keepalive-time time="10" unit="seconds"/>
</long-running-threads>
</default-workmanager>
<workmanager name="customWM">
  <elytron-enabled>true</elytron-enabled>
  <short-running-threads>
    <core-threads count="20"/>
    <queue-length count="20"/>
    <max-threads count="20"/>
  </short-running-threads>
</workmanager>
<bootstrap-contexts>
  <bootstrap-context name="customContext" workmanager="customWM"/>
</bootstrap-contexts>
<cached-connection-manager/>
</subsystem>

```

その後、ワークマネージャーは **resource-adapter** サブシステムからブートストラップコンテキストを使用して参照されます。

```

<subsystem xmlns="urn:jboss:domain:resource-adapters:5.0">
  <resource-adapters>
    <resource-adapter id="ra-with-elytron-security-domain">
      <archive>
        ra-with-elytron-security-domain.rar
      </archive>
      <bootstrap-context>customContext</bootstrap-context>
      <transaction-support>NoTransaction</transaction-support>
      <workmanager>
        <security>
          <elytron-security-domain>wm-realm</elytron-security-domain>
          <default-principal>wm-default-principal</default-principal>
          <default-groups>
            <group>
              wm-default-group
            </group>
          </default-groups>
        </security>
      </workmanager>
    </resource-adapter>
  </resource-adapters>
</subsystem>

```

### 例: セキュリティドメインの設定

```

/subsystem=elytron/properties-realm=wm-properties-realm:add(users-properties={path=/security-dir/users.properties, plain-text=true}, groups-properties={path=/security-dir/groups.properties})

```

```

/subsystem=elytron/simple-role-decoder=wm-role-decoder:add(attribute=groups)

```

```

/subsystem=elytron/constant-permission-mapper=wm-permission-mapper:add(permissions={{class-

```

```
name="org.wildfly.security.auth.permission.LoginPermission"]])
```

```
/subsystem=elytron/security-domain=wm-realm:add(default-realm=wm-properties-realm, permission-
mapper=wm-permission-mapper, realms=[{role-decoder=wm-role-decoder, realm=wm-properties-
realm}])
```

**Work** クラスは、指定のドメイン下で Elytron の認証の認証情報を提供する役割があります。これには **javax.resource.spi.work.WorkContextProvider** を実装する必要があります。

```
public interface WorkContextProvider {
    /**
     * Gets an instance of WorkContexts that needs to be used
     * by the WorkManager to set up the execution context while
     * executing a Work instance.
     *
     * @return an List of WorkContext instances.
     */
    List<WorkContext> getWorkContexts();
}
```

このインターフェースは、**Work** クラスが **WorkContext** を使用して、ワークが実行されるコンテキストの一部の機能を設定できるようにします。その機能の1つがセキュリティーインフローです。これについては、**List<WorkContext> getWorkContexts** メソッドは **javax.resource.spi.work.SecurityContext** を提供する必要があります。このコンテキストは、[JSR-196 仕様](#) で定義されている通りに **javax.security.auth.callback.Callback** オブジェクトを使用して認証情報を提供します。

#### 例: コンテキストを使用したコールバックの作成

```
public class ExampleWork implements Work, WorkContextProvider {

    private final String username;
    private final String role;

    public MyWork(TestBean bean, String username, String role) {
        this.principals = null;
        this.roles = null;
        this.bean = bean;
        this.username = username;
        this.role = role;
    }

    public List<WorkContext> getWorkContexts() {
        List<WorkContext> l = new ArrayList<>(1);
        l.add(new MySecurityContext(username, role));
        return l;
    }

    public void run() {
        ...
    }

    public void release() {
        ...
    }
}
```

```

public class ExampleSecurityContext extends SecurityContext {

    public void setupSecurityContext(CallbackHandler handler, Subject executionSubject, Subject
serviceSubject) {
        try {
            List<javax.security.auth.callback.Callback> cbs = new ArrayList<>();
            cbs.add(new CallerPrincipalCallback(executionSubject, new SimplePrincipal(username)));
            cbs.add(new GroupPrincipalCallback(executionSubject, new String[] {role}));
            handler.handle(cbs.toArray(new javax.security.auth.callback.Callback[cbs.size()]));
        } catch (Throwable t) {
            throw new RuntimeException(t);
        }
    }
}

```

上記の例では、**ExampleWork** は **WorkContextProvider** インターフェースを実装し、**ExampleSecurityContext** を提供します。そのコンテキストは、ワークの実行時に Elytron によって認証されるセキュリティー情報の提供に必要なコールバックを作成します。

#### 16.4.4. IBM MQ リソースアダプターのデプロイおよび設定

JBoss EAP の『[Configuring Messaging](#)』には、IBM MQ リソースアダプターのデプロイメントの手順が記載されています。

#### 16.4.5. 汎用 JMS リソースアダプターのデプロイおよび設定

JBoss EAP の『[Configuring Messaging](#)』には、汎用 JMS リソースアダプターの設定の手順が記載されています。

### 16.5. 管理接続プールの設定

JBoss EAP は **ManagedConnectionPool** インターフェースの 3 つの実装を提供します。

#### **org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreConcurrentLinkedQueueManagedConnectionPool**

これは、JBoss EAP 7 のデフォルトの接続プールで、追加設定がない場合のパフォーマンスが最も優れています。

#### **org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreArrayListManagedConnectionPool**

過去の JBoss EAP バージョンではデフォルトの接続プールでした。

#### **org.jboss.jca.core.connectionmanager.pool.mcp.LeakDumperManagedConnectionPool**

この接続プールはデバッグの目的でのみ使用され、シャットダウンやプールのフラッシュ時にリークが報告されます。

以下の管理 CLI コマンドを使用して、データソースの管理接続プール実装を設定できます。

```

/subsystem=datasources/data-source=DATA_SOURCE:write-attribute(name=mcp,value=MCP_CLASS)

```

以下の管理 CLI コマンドを使用して、リソースアダプターの管理接続プール実装を設定できます。

```
/subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-definitions=CONNECTION_DEFINITION:write-attribute(name=mcp,value=MCP_CLASS)
```

以下の管理 CLI コマンドを使用して、メッセージングサーバーの管理接続プール実装を設定できます。

```
/subsystem=messaging-activemq/server=SERVER/pooled-connection-factory=CONNECTION_FACTORY:write-attribute(name=managed-connection-pool,value=MCP_CLASS)
```

## 16.6. 接続統計の表示

`/deployment=NAME.rar` サブツリーから定義された接続の統計を読み取りできます。これは、`standalone.xml` または `domain.xml` 設定に定義されていなくても、すべての RAR の統計にアクセスできるようにするためです。

```
/deployment=NAME.rar/subsystem=resource-adapters/statistics=statistics/connection-definitions=java:\testMe:read-resource(include-runtime=true)
```



### 注記

統計はすべてランタイムのみの情報であるため、必ず `include-runtime=true` 引数を指定してください。

利用できる統計については、「[リソースアダプターの統計](#)」を参照してください。

## 16.7. リソースアダプター接続のフラッシュ

以下の管理 CLI コマンドを使用して、リソースアダプターの接続をフラッシュします。



### 注記

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

- プールの接続をすべてフラッシュします。

```
/subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-definitions=CONNECTION_DEFINITION:flush-all-connection-in-pool
```

- プールの接続をすべて正常にフラッシュします。

```
/subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-definitions=CONNECTION_DEFINITION:flush-gracefully-connection-in-pool
```

サーバーは、接続がアイドル状態なるまで待ってから接続をフラッシュします。

- プールのアイドル状態の接続をすべてフラッシュします。

```
/subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-definitions=CONNECTION_DEFINITION:flush-idle-connection-in-pool
```

- プールの無効な接続をすべてフラッシュします。

```
/subsystem=resource-adapters/resource-adapter=RESOURCE_ADAPTER/connection-definitions=CONNECTION_DEFINITION:flush-invalid-connection-in-pool
```

サーバーは無効であると判断したすべての接続をフラッシュします。

## 16.8. RESOURCE ADAPTERS サブシステムの調整

**resource-adapters** サブシステムのパフォーマンスを監視および最適化するための情報は、『[パフォーマンスチューニングガイド](#)』の「データソースおよびリソースアダプターの調整」を参照してください。

## 第17章 WEB サーバーの設定 (UNDERTOW)

### 17.1. UNDERTOW サブシステムの概要



#### 重要

JBoss EAP 7 では、JBoss EAP 6 の **web** サブシステムの代わりに **undertow** サブシステムが使用されます。

**undertow** サブシステムは、Web サーバーおよびサーブレットコンテナの設定を可能にします。Java Servlet 4.0 仕様や WebSocket を実装します。HTTP のアップグレードや、サーブレットデプロイメントでの高パフォーマンスな非ブロッキングハンドラーの使用もサポートします。**undertow** サブシステムは、mod\_cluster をサポートする高パフォーマンスなリバースプロキシとして動作することも可能です。

**undertow** サブシステム内で設定する主なコンポーネントは5つあります。

- バッファーク্যাッシュ
- server
- サーブレットコンテナ
- handlers
- フィルター



#### 注記

JBoss EAP には、これらの各コンポーネントの設定を更新する機能がありますが、デフォルト設定は妥当なパフォーマンスの設定を提供し、ほとんどのユースケースに適しています。

### デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
  <buffer-cache name="default"/>
  <server name="default-server">
    <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
    <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content"/>
      <http-invoker security-realm="ApplicationRealm"/>
    </host>
  </server>
  <servlet-container name="default">
    <jsp-config/>
    <websockets/>
  </servlet-container>
  <handlers>
```

```
<file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
</handlers>
</subsystem>
```



### 重要

また、**undertow** サブシステムは **io** サブシステムに依存して XNIO ワーカーやバッファプールを提供します。**io** サブシステムは個別に設定され、ほとんどの場合で最適なパフォーマンスを得られるデフォルト設定を提供します。



### 注記

JBoss EAP 6 の **web** サブシステムと比較すると、JBoss EAP 7 の **undertow** サブシステムでは [HTTP メソッドのデフォルト動作](#) が異なります。

## Undertow サブシステムでの Elytron の使用

web アプリケーションがデプロイされると、そのアプリケーションが必要なセキュリティドメインの名前が特定されます。これは、デプロイメント内からで、デプロイメントにセキュリティドメインがない場合は **undertow** サブシステムで定義された **default-security-domain** が仮定されます。デフォルトでは、セキュリティドメインがレガシー security サブシステムで定義された **PicketBox** にマップすることが仮定されます。しかし、アプリケーションが必要なセキュリティドメインの名前から適切な Elytron 設定にマッピングする **undertow** サブシステムに **application-security-domain** リソースを追加することができます。

### 例: マッピングの追加

```
/subsystem=undertow/application-security-domain=MyAppSecurity:add(security-domain=ApplicationDomain)
```

マッピングの追加に成功すると、以下のような結果が出力されます。

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" ... default-security-domain="other">
...
  <application-security-domains>
    <application-security-domain name="MyAppSecurity" security-domain="ApplicationDomain"/>
  </application-security-domains>
...
</subsystem>
```



### 注記

この時点でデプロイメントがすでにデプロイされている場合は、アプリケーションサーバーをリロードして、アプリケーションセキュリティドメインマッピングを有効にする必要があります。

この簡単な例は、**BASIC**、**CLIENT\_CERT**、**DIGEST**、**FORM** など、デプロイメントがサブレット仕様内で定義された標準の HTTP メカニズムを使用する場合に適しています。この場合、認証は **ApplicationDomain** セキュリティドメインに対して実行されます。このフォームは、アプリケーションが認証メカニズムを使用せず、代わりにプログラムによる認証を使用したり、デプロイメントに関連する **SecurityDomain** を取得して直接使用したりする場合にも適しています。

### 例: 高度なマッピング

-

```
/subsystem=undertow/application-security-domain=MyAppSecurity:add(http-authentication-
factory=application-http-authentication)
```

高度なマッピングに成功すると、以下のような結果が出力されます。

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" ... default-security-domain="other">
...
  <application-security-domains>
    <application-security-domain name="MyAppSecurity" http-authentication-factory="application-
http-authentication"/>
  </application-security-domains>
...
</subsystem>
```

このような設定では、セキュリティドメインを参照する代わりに、**http-authentication-factory** が参照されます。これは、認証メカニズムのインスタンスの取得に使用されるファクトリーで、セキュリティドメインと関連付けられます。

カスタム HTTP 認証メカニズムを使用する場合や、プリンシパルトランスフォーマー、認証情報ファクトリー、メカニズムレルムなどのメカニズムに追加の設定を定義する必要がある場合など、**http-authentication-factory** 属性を参照する必要があります。また、**Servlet** 仕様に記載されている 4 つのメカニズム以外を使用する場合は、**http-authentication-factory** 属性を参照した方がよいでしょう。

高度なマッピングが使用される場合、別の設定オプション **override-deployment-config** を使用できます。参照された **http-authentication-factory** は認証メカニズムの完全セットを返すことができます。デフォルトでは、アプリケーションによってリクエストされたメカニズムと一致するするためにフィルターされます。このオプションが **true** に設定された場合、ファクトリーによって提供されたメカニズムは、アプリケーションによってリクエストされたメカニズムをオーバーライドします。

**application-security-domain** リソースには追加のオプション **enable-jacc** もあります。これを **true** に設定すると、このマッピングに一致するすべてのデプロイメントに対して JACC が有効になります。

## ランタイム情報

**application-security-domain** マッピングが使用されている場合、このマッピングに対してデプロイメントが想定どおり一致していることを再度確認すると便利です。リソースが **include-runtime=true** で読み取りされた場合、マッピングに関連するデプロイメントは以下のように表示されます。

```
/subsystem=undertow/application-security-domain=MyAppSecurity:read-resource(include-
runtime=true)
{
  "outcome" => "success",
  "result" => {
    "enable-jacc" => false,
    "http-authentication-factory" => undefined,
    "override-deployment-config" => false,
    "referencing-deployments" => ["simple-webapp.war"],
    "security-domain" => "ApplicationDomain",
    "setting" => undefined
  }
}
```

この出力の **referencing-deployments** 属性は、**simple-webapp.war** デプロイメントがマッピングを使用してデプロイされたことを示しています。



## 17.2. バッファークャッシュの設定

バッファークャッシュは静的リソースをキャッシュするために使用されます。JBoss EAP では複数のキャッシュを設定でき、デプロイメントによる参照が可能であるため、デプロイメントごとに異なるキャッシュサイズを使用することができます。バッファークャッシュは固定サイズで、リージョンで割り当てられます。使用領域の合計は、バッファークャッシュサイズ、リージョンごとのバッファークャッシュ数、およびリージョンの最大数を掛けて算出できます。バッファークャッシュのデフォルトサイズは10MBです。

JBoss EAP はデフォルトで単一のキャッシュを提供します。

### デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
  <buffer-cache name="default"/>
  ...
</subsystem>
```

### 既存のバッファークャッシュの更新

既存のバッファークャッシュを更新するには、以下を指定します。

```
/subsystem=undertow/buffer-cache=default/:write-attribute(name=buffer-size,value=2048)
```

```
reload
```

### 新規バッファークャッシュの作成

新しいバッファークャッシュを作成するには、以下を指定します。

```
/subsystem=undertow/buffer-cache=new-buffer:add
```

### バッファークャッシュの削除

バッファークャッシュを削除するには、以下を指定します。

```
/subsystem=undertow/buffer-cache=new-buffer:remove
```

```
reload
```

バッファークャッシュの設定に使用できる属性の完全リストは、「[Undertow サブシステムの属性](#)」の項を参照してください。

## 17.3. バイトバッファープールの設定

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

これらのバッファークャッシュは IO 操作に使用され、バッファークャッシュサイズはアプリケーションのパフォーマンスに大きく影響します。通常、ほとんどのサーバーでは 16 K が理想のサイズになります。

### 既存のバイトバッファープールの更新

既存のバイトバッファープールを更新するには、以下を指定します。

■

```
/subsystem=undertow/byte-buffer-pool=myByteBufferPool:write-attribute(name=buffer-size,value=1024)
```

```
reload
```

### バイトバッファープールの作成

新しいバイトバッファープールを作成するには、以下を指定します。

```
/subsystem=undertow/byte-buffer-pool=newByteBufferPool:add
```

### バイトバッファープールの削除

バイトバッファープールを削除するには、以下を指定します。

```
/subsystem=undertow/byte-buffer-pool=newByteBufferPool:remove
```

```
reload
```

バイトバッファープールの設定に使用できる属性の完全リストは「[バイトバッファープールの属性](#)」を参照してください。

## 17.4. サーバーの設定

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

- host
- http-listener
- https-listener
- ajp-listener

ホスト要素は仮想ホスト設定を提供し、3つのリスナーはそのタイプの接続を Undertow インスタンスに提供します。

サーバーのデフォルトの動作では、サーバーの起動中にリクエストをキューに置きます。デフォルトの動作を変更するには、ホストで **queue-requests-on-start** 属性を使用します。この属性がデフォルトの **true** に設定されている場合、サーバーの起動時に到達するリクエストはサーバーの準備が整うまで保留されます。この属性が **false** に設定された場合、サーバーが完全に起動する前に到達したリクエストは、デフォルトの応答コードで拒否されます。

属性の値に関わらず、サーバーが完全に起動するまでリクエストの処理は開始されません。

管理コンソールを使用して **queue-requests-on-start** 属性を設定するには、**Configuration** → **Subsystems** → **Web (Undertow)** → **Server** と選択した後、サーバーを選択して **表示** をクリックし、**Hosts** タブを選択します。管理対象ドメインでは、設定するプロファイルを指定する必要があります。



### 注記

複数のサーバーを設定できるため、デプロイメントやサーバーを完全に分離できます。これは、マルチテナント環境などで便利です。

JBoss EAP はデフォルトでサーバーを提供します。

## デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
  <buffer-cache name="default"/>
  <server name="default-server">
    <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
    <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content"/>
      <http-invoker security-realm="ApplicationRealm"/>
    </host>
  </server>
  ...
</subsystem>
```

以下の例は、管理 CLI を使用してサーバーを設定する方法を示しています。管理コンソールを使用してサーバーを設定する場合は、**Configuration** → **Subsystems** → **Web (Undertow)** → **Server** と選択します。

### 既存のサーバーの更新

既存のサーバーを更新するには、以下を設定します。

```
/subsystem=undertow/server=default-server:write-attribute(name=default-host,value=default-host)
```

```
reload
```

### 新規サーバーの作成

新規サーバーを作成するには、以下を指定します。

```
/subsystem=undertow/server=new-server:add
```

```
reload
```

### サーバーの削除

サーバーを削除するには、以下を指定します。

```
/subsystem=undertow/server=new-server:remove
```

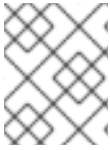
```
reload
```

サーバーの設定に使用できる属性の完全リストは、「[Undertow サブシステムの属性](#)」の項を参照してください。

## 17.5. サーブレットコンテナの設定

サーブレットコンテナは、すべてのサーブレット、JSP、およびソケット関連の設定 (セッションに関連する設定を含む) を提供します。ほとんどのサーバーにはサーブレットコンテナが1つだけ必要ですが、**servlet-container** 要素を追加すると複数のサーブレットコンテナを設定することができま

す。サーブレットコンテナが複数設定されていると、複数のデプロイメントを異なる仮想ホストの同じコンテキストパスにデプロイできるなど、一部の動作を有効にすることができます。



### 注記

サーブレットコンテナによって提供される設定の多くは、デプロイされたアプリケーションが **web.xml** ファイルを使用して個別にオーバーライドできます。

JBoss EAP はデフォルトでサーブレットコンテナを提供します。

## デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0">
  <buffer-cache name="default"/>
  <server name="default-server">
    ...
  </server>
  <servlet-container name="default">
    <jsp-config/>
    <websockets/>
  </servlet-container>
  ...
</subsystem>
```

以下の例は、管理 CLI を使用してサーブレットコンテナを設定する方法を示しています。管理コンソールを使用してサーブレットコンテナを設定する場合は、**Configuration** → **Subsystems** → **Web (Undertow)** → **Servlet Container** と選択します。

### 既存のサーブレットコンテナの更新

既存のサーブレットコンテナを更新するには、以下を指定します。

```
/subsystem=undertow/servlet-container=default:write-attribute(name=ignore-flush,value=true)
```

```
reload
```

### 新規サーブレットコンテナの作成

新規のサーブレットコンテナを作成するには、以下を指定します。

```
/subsystem=undertow/servlet-container=new-servlet-container:add
```

```
reload
```

### サーブレットコンテナの削除

サーブレットコンテナを削除するには、以下を指定します。

```
/subsystem=undertow/servlet-container=new-servlet-container:remove
```

```
reload
```

サーブレットコンテナの設定に使用できる属性の完全リストは、「[Undertow サブシステムの属性](#)」の項を参照してください。

## 17.6. サブレット拡張の設定

サブレット拡張は、サブレットデプロイメントプロセスへのフックや、サブレットデプロイメントの変更を可能にします。これは、追加の認証メカニズムをデプロイメントに追加する必要がある場合や、ネイティブ Undertow ハンドラーをサブレットデプロイメントの一部として使用する必要がある場合などに便利です。

カスタムサブレット拡張を作成するには、`io.undertow.servlet.ServletExtension` インターフェースを実装した後、実装クラスの名前をデプロイメントの **META-INF/services/io.undertow.servlet.ServletExtension** ファイルに追加する必要があります。さらに、**ServletExtension** 実装のコンパイルされたクラスファイルを含める必要があります。Undertow がサブレットをデプロイすると、**deployments** クラスローダーからすべてのサービスをロードし、それらの **handleDeployment** メソッドを呼び出します。

デプロイメントの完全かつ変更可能な記述が含まれる Undertow **DeploymentInfo** 構造は、このメソッドに渡されます。この構造を変更して、デプロイメントの内容を変更することができます。

**DeploymentInfo** 構造は、組み込み API によって使用される構造と同じであるため、**ServletExtension** は Undertow を組み込みモードで使用したときと同じ柔軟性を持ちます。

## 17.7. ハンドラーの設定

JBoss EAP では、2つのタイプのハンドラーを設定できます。

- ファイルハンドラー
- リバースプロキシハンドラー

ファイルハンドラーは静的ファイルに対応します。各ファイルハンドラーは仮想ホストの場所にアタッチされている必要があります。リバースプロキシハンドラーによって、JBoss EAP は高パフォーマンスなリバースプロキシとして機能することができます。

JBoss EAP はデフォルトでファイルハンドラーを提供します。

### デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
  <buffer-cache name="default"/>
  <server name="default-server">
    ...
  </server>
  <servlet-container name="default">
    ...
  </servlet-container>
  <handlers>
    <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
  </handlers>
</subsystem>
```

### 静的リソースに WebDAV を使用

過去のバージョンの JBoss EAP では、**web** サブシステムで WebDAV を使用して (**WebdavServlet** 経由) 静的リソースをホストし、追加の HTTP メソッドを有効にしてこれらのファイルへのアクセスや操作を実行できました。JBoss EAP 7 では、ファイルハンドラーを経由した静的ファイルの対応メカニズ

ムは **undertow** サブシステムによって提供されますが、**undertow** サブシステムは WebDAV をサポートしません。JBoss EAP 7 で WebDAV を使用する場合は、カスタムの WebDav サーブレットを記述してください。

### 既存のファイルハンドラーの更新

既存のファイルハンドラーを更新するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/file=welcome-content:write-attribute(name=case-sensitive,value=true)
```

```
reload
```

### 新規ファイルハンドラーの作成

新規のファイルハンドラーを作成するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/file=new-file-handler:add(path="${jboss.home.dir}/welcome-content")
```



#### 警告

ファイルハンドラーの **path** を直接ディレクトリーではなくファイルに設定した場合、そのファイルハンドラーを参照する **location** 要素の最後にフォワードスラッシュ (/) を付けないでください。最後にフォワードスラッシュが付くと、**404 - Not Found** が返されます。

### ファイルハンドラーの削除

ファイルハンドラーを削除するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/file=new-file-handler:remove
```

```
reload
```

ハンドラーの設定に使用できる属性の完全リストは、「[Undertow サブシステムの属性](#)」の項を参照してください。

## 17.8. フィルターの設定

フィルターはリクエストの一部の変更を可能にし、述語を使用してフィルターの実行時を制御できます。フィルターの一般的なユースケースには、ヘッダーの設定や GZIP 圧縮などがあります。



#### 注記

フィルターの機能は、JBoss EAP 6 で使用されたグローバルバルブと同等です。

以下のタイプのフィルターを定義できます。

- custom-filter

- error-page
- expression-filter
- gzip
- mod-cluster
- request-limit
- response-header
- rewrite

以下の例は、管理 CLI を使用してフィルターを設定する方法を示しています。管理コンソールを使用してフィルターを設定する場合は、**Configuration → Subsystems → Web (Undertow) → Filters** と選択します。

### 既存のフィルターの更新

既存のフィルターを更新するには、以下を指定します。

```
/subsystem=undertow/configuration=filter/response-header=myHeader:write-attribute(name=header-value,value="JBoss-EAP")
```

```
reload
```

### 新規のフィルターの作成

新規のフィルターを作成するには、以下を指定します。

```
/subsystem=undertow/configuration=filter/response-header=new-response-header:add(header-name=new-response-header,header-value="My Value")
```

### フィルターの削除

フィルターを削除するには、以下を指定します。

```
/subsystem=undertow/configuration=filter/response-header=new-response-header:remove
```

```
reload
```

フィルターの設定に使用できる属性の完全リストは「[Undertow サブシステムの属性](#)」の項を参照してください。

## 17.8.1. buffer-request ハンドラーの設定

クライアントまたはブラウザーからのリクエストは、ヘッダーとボディの2つで構成されます。通常の場合、ヘッダーとボディの間に遅延がない状態で JBoss EAP に送信されます。しかし、ヘッダーが最初に送信され、その数秒後にボディが送信されると、完全なリクエストの送信に遅延が発生します。このような場合、JBoss EAP にスレッドが作成され、完全なリクエストの実行を待機していることを示す **waiting** が表示されます。

リクエストのヘッダーおよびボディの送信による遅延は、**buffer-request** ハンドラーを使用して修正できます。**buffer-request** ハンドラーは、ワーカースレッドに割り当てする前に、非ブロッキング IO スレッドからリクエストの消費を試みます。追加された **buffer-request** ハンドラーがない場合、ス

レッドは直接ワーカーレッドに割り当てられます。しかし、**buffer-request** ハンドラーが追加されると、ワーカーレッドに割り当てする前に、ハンドラーは IO スレッドを使用してブロックせずにバッファ処理できる量のデータを読み取ろうとします。

以下の管理 CLI コマンドを使用して **buffer-request** ハンドラーを設定できます。

```
/subsystem=undertow/configuration=filter/expression-filter=buf:add(expression="buffer-request(buffers=1)")
```

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

処理できるバッファリクエストのサイズには制限があります。この制限は、以下の式のとおり、バッファサイズとバッファ合計数の組み合わせで決定されます。

**Total\_size = num\_buffers × buffer\_size**

この式の説明は次のとおりです。

- **Total\_size** は、リクエストがワーカーレッドに送信される前にバッファ処理されるデータのサイズになります。
- **num\_buffers** はバッファの数になります。バッファの数は、ハンドラーの **buffers** パラメーターによって設定されます。
- **buffer\_size** は各バッファのサイズになります。バッファサイズは **io** サブシステムで設定され、デフォルトではリクエストごとに 16KB になります。



#### 警告

メモリー不足になる可能性があるため、サイズが大きすぎるバッファリクエストは設定しないでください。

## 17.9. デフォルトの WELCOME WEB アプリケーションの設定

JBoss EAP には、デフォルトではポート **8080** のルートコンテキストで表示されるデフォルトの **Welcome** アプリケーションが含まれます。

Undertow には、Welcome コンテンツに対応するデフォルトのサーバーが事前設定されています。

### デフォルトの Undertow サブシステムの設定

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
...
  <server name="default-server">
    <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
    <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content"/>
      <http-invoker security-realm="ApplicationRealm"/>
    </host>
  </server>
</subsystem>
```



```

    </host>
  </server>
  ...
  <handlers>
    <file name="welcome-content" path="{jboss.home.dir}/welcome-content"/>
  </handlers>
</subsystem>

```

デフォルトのサーバー **default-server** にはデフォルトのホスト **default-host** が設定されています。デフォルトのホストは、**welcome-content** ファイルハンドラーで **<location>** 要素を使用して、サーバーのルートへのリクエストを処理するよう設定されています。**welcome-content** ハンドラーは **path** プロパティに指定された場所でコンテンツを処理します。

このデフォルトの **Welcome** アプリケーションは、独自の Web アプリケーションで置き換えることができます。これは、以下の2つのいずれかの方法で設定できます。

- [welcome-content ファイルハンドラーの変更](#)
- [default-web-module の変更](#)

Welcome コンテンツを無効にすることもできます。

### welcome-content ファイルハンドラーの変更

1. 新しいデプロイメントを参照する、既存の **welcome-content** ファイルハンドラーのパスを変更します。

```

/subsystem=undertow/configuration=handler/file=welcome-content:write-attribute(name=path,value="/path/to/content")

```



#### 注記

または、サーバーのルートにより使用される異なるファイルハンドラーを作成することもできます。

```

/subsystem=undertow/configuration=handler/file=NEW_FILE_HANDLER:add(
  path="/path/to/content")
/subsystem=undertow/server=default-server/host=default-
host/location=/:write-attribute(name=handler,value=NEW_FILE_HANDLER)

```

2. 変更を反映するためにサーバーをリロードします。

```

reload

```

### default-web-module の変更

1. デプロイされた Web アプリケーションをサーバーのルートにマップします。

```

/subsystem=undertow/server=default-server/host=default-host:write-attribute(name=default-
web-module,value=hello.war)

```

2. 変更を反映するためにサーバーをリロードします。

```

reload

```

-

## デフォルトの Welcome Web アプリケーションの無効化

1. **default-host** の **location** エントリー (/) を削除して welcome アプリケーションを無効にします。

```
/subsystem=undertow/server=default-server/host=default-host/location=/:remove
```

2. 変更を反映するためにサーバーをリロードします。

```
reload
```

## 17.10. HTTPS の設定

Web アプリケーションの HTTPS 設定に関する詳細は、『[How to Configure Server Security](#)』の「[Configure One-way and Two-way SSL/TLS for Applications](#)」を参照してください。

JBoss EAP 管理インターフェースと使用するための HTTPS 設定に関する詳細は、『[How to Configure Server Security](#)』の「[How to Secure the Management Interfaces](#)」を参照してください。

## 17.11. HTTP セッションタイムアウトの設定

HTTP セッションタイムアウトは、HTTP セッションの無効を宣言するために必要な非アクティブな期間を定義します。たとえば、HTTP セッションを作成する JBoss EAP にデプロイされたアプリケーションにユーザーがアクセスしたとします。HTTP セッションタイムアウト後に同じユーザーが同じアプリケーションに再度アクセスしようとする、元の HTTP セッションは無効化され、ユーザーは新しい HTTP セッションの作成を強制されます。これにより、永続化されなかったデータを損失したり、ユーザーを再認証する必要がある場合があります。

HTTP セッションタイムアウトは、アプリケーションの **web.xml** ファイルに設定されますが、デフォルトの HTTP セッションタイムアウトは JBoss EAP 内で指定できます。サーバーのタイムアウト値はデプロイされたすべてのアプリケーションに適用されますが、アプリケーションの **web.xml** はサーバーの値をオーバーライドします。

サーバーの値は、**undertow** サブシステムの **servlet-container** セクションにある **default-session-timeout** プロパティに指定されます。**default-session-timeout** の値は分単位で指定され、デフォルトは **30** です。

### デフォルトのセッションタイムアウトの設定

**default-session-timeout** を設定するには、以下を指定します。

```
/subsystem=undertow/servlet-container=default:write-attribute(name=default-session-timeout,value=60)
```

```
reload
```

## 17.12. HTTP のみのセッション管理クッキーの設定

セッション管理クッキーは、JavaScript などの HTTP API および非 HTTP API の両方によってアクセスされます。JBoss EAP は **HttpOnly** ヘッダーを **Set-Cookie** 応答ヘッダーの一部としてクライアント (通常はブラウザ) に送信します。サポートされるブラウザでこのヘッダーを有効にすると、非 HTTP API を経由してセッション管理クッキーへアクセスしないようにブラウザに通知します。セッ

セッション管理クッキーを HTTP API のみに制限すると、クロスサイトスクリプティングの攻撃によるセッションクッキーの窃盗のリスクを軽減することができます。この動作を有効にするには、**http-only** 属性を **true** に設定する必要があります。



### 重要

**HttpOnly** ヘッダーを使用しても、単にブラウザに通知を行うだけで、クロスサイトスクリプティングによる攻撃を実際に防ぐわけではありません。この動作を反映するには、ブラウザも **HttpOnly** をサポートしている必要があります。



### 重要

**HttpOnly** 属性を使用すると制限がセッション管理クッキーのみに適用され、その他のブラウザクッキーには適用されません。

**http-only** 属性は **undertow** サブシステムの 2 カ所で設定されます。

- セッションクッキー設定としてサブレットコンテナで設定
- 単一のサインオンプロパティとしてサーバーのホストセクションで設定

#### host-only をサブレットコンテナセッションクッキーに設定

**host-only** プロパティをサブレットコンテナセッションクッキーに設定するには、以下を指定します。

```
/subsystem=undertow/servlet-container=default/setting=session-cookie:add
```

```
/subsystem=undertow/servlet-container=default/setting=session-cookie:write-attribute(name=http-only,value=true)
```

```
reload
```

#### host-only をホストシングルサインオンに設定

**host-only** プロパティをホストシングルサインオンに設定するには、以下を指定します。

```
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:add
```

```
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:write-attribute(name=http-only,value=true)
```

```
reload
```

## 17.13. HTTP/2 の設定

Undertow では、**HTTP/2** 標準を使用できます。この標準は、ヘッダーの圧縮と多くのストリームの多重化を同じ TCP 接続で行い、待ち時間を削減します。さらに、リクエストの前にサーバーがリソースをクライアントにプッシュできる機能も提供するため、ページのロードがより速くなります。

HTTP/2 は、HTTP/2 標準をサポートするクライアントとブラウザでのみ機能することに注意してください。

 重要

最新のブラウザは、**h2** と呼ばれるセキュアな TLS 接続上の HTTP/2 を強制します。**h2c** と呼ばれるプレーン HTTP 上の HTTP/2 はサポートしないことがあります。**h2c** で HTTP/2 を使用するように JBoss EAP を設定することが可能です。つまり、HTTPS を使用せずに HTTP のアップグレードでプレーン HTTP のみを使用します。この場合、Undertow の HTTP リスナーで HTTP/2 を有効にします。

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

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

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

HTTPS リスナーの情報や、Web アプリケーションで HTTPS を使用するよう Undertow を設定する方法については、『[How to Configure Server Security](#)』の「[Configure One-way and Two-way SSL/TLS for Applications](#)」を参照してください。

 注記

**elytron** サブシステムで HTTP/2 を使用するには、Undertow の **https-listener** にある設定済みの **ssl-context** が変更可能として設定される必要があります。これには、適切な **server-ssl-context** の **wrap** 属性を **false** に設定します。デフォルトでは **wrap** 属性は **false** に設定されます。これは、ALPN に関する **ssl-context** の変更を行うために Undertow で必要になります。提供された **ssl-context** が書き込み可能でない場合、ALPN は使用できず、接続は HTTP/1.1 にフォールバックします。

## HTTP/2 使用時の ALPN サポート

セキュアな TLS 接続上で HTTP/2 を使用する場合、ALPN TLS プロトコル拡張をサポートする TLS スタックが必要になります。このスタックの取得はインストールされた JDK によって異なります。

- Java 8 を使用する場合、ALPN 実装は Java 内部に依存して直接 JBoss EAP に導入されます。そのため、ALPN 実装は Oracle および OpenJDK でのみ動作します。IBM Java では動作しません。Red Hat は、ALPN 機能を実装する OpenSSL ライブラリーとともに、OpenSSL プロバイダーからの ALPN TLS プロトコル拡張サポートを JBoss EAP で使用することを推奨します。OpenSSL プロバイダーの ALPN TLS プロトコル拡張サポートを使用すると、パフォーマンスが向上します。
- Java 9 より、JDK は ALPN をネイティブでサポートするようになりましたが、Java 9 以上を使用する場合でも OpenSSL プロバイダーからの ALPN TLS プロトコル拡張サポートを使用するとパフォーマンスが向上されるはずですが。

ALPN TLS プロトコル拡張サポートを取得するための OpenSSL のインストール手順は、「[JBoss Core Services からの OpenSSL のインストール](#)」を参照してください。標準のシステム OpenSSL は Red Hat Enterprise Linux 8 でサポートされており、追加の JBoss Core Services OpenSSL は必要ありません。

OpenSSL がインストールされたら、「[OpenSSL を使用するよう JBoss EAP を設定](#)」にある手順にしたがいます。

## HTTP/2 が使用されていることを検証

Undertow が HTTP/2 を使用していることを検証するには、Undertow からのヘッダーを確認する必要があります。https を使用して JBoss EAP インスタンスに移動し (例: <https://localhost:8443>)、ブラウザの開発者ツールを使用してヘッダーを確認します。Google Chrome などの一部のブラウザは、HTTP/2 の使用時には **:path**、**:authority**、**:method**、**:scheme** などの HTTP/2 擬似ヘッダーを表示します。Firefox や Safari などの他のブラウザは、ヘッダーの状態またはバージョンを **HTTP/2.0** と表示します。

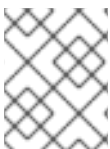
## 17.14. REQUESTDUMPING ハンドラーの設定

**RequestDumping** ハンドラーである `io.undertow.server.handlers.RequestDumpingHandler` は、JBoss EAP 内で Undertow によって処理されるリクエストとその応答オブジェクトの詳細をログに記録します。



### 重要

このハンドラーはデバッグに便利ですが、機密情報がログに記録される可能性があります。この点に留意してこのハンドラーを有効にしてください。



### 注記

**RequestDumping** ハンドラーは、JBoss EAP 6 の **RequestDumperValve** の代わりに使用されます。

**RequestDumping** ハンドラーは、JBoss EAP のサーバーレベルまたは個別のアプリケーション内のいずれかで設定できます。

### 17.14.1. サーバーでの RequestDumping ハンドラーの設定

**RequestDumping** ハンドラーは式フィルターとして設定する必要があります。**RequestDumping** ハンドラーを式フィルターとして設定するには、以下を行う必要があります。

**RequestDumping** ハンドラーで新しい式フィルターを作成する

```
/subsystem=undertow/configuration=filter/expression-
filter=requestDumperExpression:add(expression="dump-request")
```

Undertow Web サーバーで式フィルターを有効にする

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



### 重要

このように **RequestDumping** ハンドラーを式フィルターとして有効にすると、Undertow Web サーバーによって処理されるすべてのリクエストおよびそれらの応答がログに記録されます。

特定 URL に対して **RequestDumping** ハンドラーを設定する

すべてのリクエストをログに記録する他に、特定の URL のリクエストやそれらの応答のみをログに記録するために式フィルターを使用することもできます。これには、**path**、**path-prefix**、**path-suffix** などの述語を式に使用します。たとえば、`/myApplication/test` へのリクエストとそれらの応答をすべて

ログに記録するには、式フィルターの作成時に式 **"dump-request"** の代わりに **"path(/myApplication/test) -> dump-request"** を使用します。これにより、**/myApplication/test** に完全一致するパスを持つリクエストのみが **RequestDumping** ハンドラーに送られます。

### 17.14.2. アプリケーション内での RequestDumping ハンドラーの設定

サーバーで **RequestDumping** ハンドラーを設定する他に、個別のアプリケーション内で設定することもできます。これにより、ハンドラーの範囲がそのアプリケーションのみに制限されます。**RequestDumping** ハンドラーは **WEB-INF/undertow-handlers.conf** で設定する必要があります。

指定のアプリケーションのすべてのリクエストとそれらの応答をログに記録するよう **WEB-INF/undertow-handlers.conf** で **RequestDumping** ハンドラーを設定するには、以下の式を **WEB-INF/undertow-handlers.conf** に追加します。

例: **WEB-INF/undertow-handlers.conf**

```
dump-request
```

指定のアプリケーション内での特定 URL のリクエストやそれらの応答のみをログに記録するよう、**WEB-INF/undertow-handlers.conf** で **RequestDumping** ハンドラーを設定するには、**path**、**path-prefix**、**path-suffix** などの述語を式に使用します。たとえば、アプリケーションの **test** へのリクエストとそれらの応答をすべてログに記録するには、**path** 述語が含まれる以下の式を使用できます。

例: **WEB-INF/undertow-handlers.conf**

```
path(/test) -> dump-request
```



#### 注記

**path**、**path-prefix**、**path-suffix** などの述語をアプリケーションの **WEB-INF/undertow-handlers.conf** に定義された式で使用する場合、使用する値はアプリケーションのコンテキストルートからの相対値になります。たとえば、アプリケーションのコンテキストルートは **myApplication** で、式 **path(/test) -> dump-request** が **WEB-INF/undertow-handlers.conf** に設定されている場合、**/myApplication/test** へのリクエストとそれらの応答のみがログに記録されます。

### 17.15. クッキーセキュリティの設定

**secure-cookie** ハンドラーを使用して、サーバーとクライアント間の接続上で作成されたクッキーのセキュリティを強化できます。この場合、クッキーが設定される接続がセキュアであるとみなされると、クッキーの **secure** 属性が **true** に設定されます。

リスナーを設定したり、HTTPS を使用すると、接続をセキュアにすることができます。**secure-cookie** ハンドラーを設定するには、**undertow** サブシステムの **expression-filter** を定義します。詳細は、「[フィルターの設定](#)」を参照してください。

**secure-cookie** ハンドラーが使用されている場合、セキュアな接続上で設定されたクッキーは暗黙的にセキュアとして設定され、セキュアでない接続上で送信されることはありません。

### 17.16. UNDERTOW サブシステムの調整

**undertow** サブシステムのパフォーマンスを最適化するための情報は、『[パフォーマンスチューニングガイド](#)』の「Undertow サブシステムの調整」を参照してください。

## 第18章 リモータリングの設定

### 18.1. REMOTING サブシステム

**remoting** サブシステムは、ローカルおよびリモートサービスのインバウンドおよびアウトバウンド接続の設定を可能にします。

JBoss Remoting には、設定可能な要素としてエンドポイント、コネクタ、複数のローカルおよびリモート接続 URI が含まれます。独自のアプリケーションにカスタムコネクタを使用する場合を除き、**remoting** のサブシステムの設定は必要でないことがほとんどです。EJB などの、リモータリングクライアントとして動作するアプリケーションには特定のコネクタに接続するための別の設定が必要になります。

#### Remoting サブシステムのデフォルト設定

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
  <endpoint/>
  <http-connector name="http-remoting-connector" connector-ref="default" security-
realm="ApplicationRealm"/>
</subsystem>
```

**remoting** サブシステムで使用できる属性の完全リストは、「[Remoting サブシステムの属性](#)」を参照してください。

#### リモータリングエンドポイント

リモータリングエンドポイントは、**io** サブシステムによって宣言および設定される XNIO ワーカーを使用します。

リモータリングエンドポイントの設定方法の詳細は、「[エンドポイントの設定](#)」を参照してください。

#### コネクタ

コネクタは主なりモータリング設定要素です。複数のコネクタを設定できます。各コネクタは、複数のサブ要素を持つ **<connector>** 要素とその他の属性で構成されます。デフォルトのコネクタは複数の JBoss EAP サブシステムによって使用されます。カスタムコネクタの要素と属性の設定は、アプリケーションによって異なります。詳細は Red Hat グローバルサポートサービスまでお問い合わせください。

コネクタの設定方法の詳細は、「[コネクタの設定](#)」を参照してください。

#### 送信接続

3つのタイプのアウトバウンド接続を指定することができます。

- URI によって指定される [アウトバウンド接続](#)
- ソケットなどのローカルリソースに接続する [ローカルアウトバウンド接続](#)
- リモートリソースに接続し、セキュリティーレームを使用して認証を行う [リモートアウトバウンド接続](#)

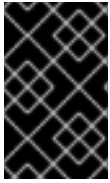
#### 追加設定

リモータリングは、ネットワークインターフェースや IO ワーカーなどの **remoting** サブシステム外部で設定された複数の要素に依存します。

詳細は、「[リモータリングの追加設定](#)」を参照してください。



## 18.2. エンドポイントの設定



### 重要

JBoss EAP 6 では、ワーカーレッドプールは直接 **remoting** サブシステムで設定されてきました。JBoss EAP 7 では、リモータリング **endpoint** 設定が **io** サブシステムからワーカーを参照します。

JBoss EAP は以下の **endpoint** 設定をデフォルトで提供します。

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
  <endpoint/>
  ...
</subsystem>
```

### 既存のエンドポイント設定の更新

```
/subsystem=remoting/configuration=endpoint:write-attribute(name=authentication-retries,value=2)
```

```
reload
```

### 新規エンドポイント設定の作成

```
/subsystem=remoting/configuration=endpoint:add
```

### エンドポイント設定の削除

```
/subsystem=remoting/configuration=endpoint:remove
```

```
reload
```

エンドポイント設定で使用できる属性の完全リストは、「[エンドポイントの設定](#)」を参照してください。

## 18.3. コネクタの設定

コネクタはリモータリングに関する主な設定要素で、追加設定のサブ要素が複数含まれます。

### 既存のコネクタ設定の更新

```
/subsystem=remoting/connector=new-connector:write-attribute(name=socket-binding,value=my-socket-binding)
```

```
reload
```

### 新規コネクタの作成

```
/subsystem=remoting/connector=new-connector:add(socket-binding=my-socket-binding)
```

### コネクタの削除

```
/subsystem=remoting/connector=new-connector:remove
```

```
reload
```

コネクタの設定に使用できる属性の完全リストは「[Remoting サブシステムの属性](#)」の項を参照してください。

## 18.4. HTTP コネクタの設定

HTTP コネクタは、HTTP アップグレードベースのリモートリングコネクタの設定を提供します。JBoss EAP はデフォルトで次の **http-connector** 設定を提供します。

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
  ...
  <http-connector name="http-remoting-connector" connector-ref="default" security-
  realm="ApplicationRealm"/>
</subsystem>
```

デフォルトでは、この HTTP コネクタは **undertow** サブシステムに設定される **default** という名前の HTTP リスナーに接続します。詳細は、「[Web サーバーの設定 \(Undertow\)](#)」を参照してください。

### 既存の HTTP コネクタ設定の更新

```
/subsystem=remoting/http-connector=new-connector:write-attribute(name=connector-ref,value=new-
connector-ref)
```

```
reload
```

### 新規 HTTP コネクタの作成

```
/subsystem=remoting/http-connector=new-connector:add(connector-ref=default)
```

### HTTP コネクタの削除

```
/subsystem=remoting/http-connector=new-connector:remove
```

HTTP コネクタの設定に使用できる属性の完全リストは、「[コネクタの属性](#)」を参照してください。

## 18.5. アウトバウンド接続の設定

アウトバウンド接続は、URI によって完全に指定される汎用のリモートリングアウトバウンド接続です。

### 既存のアウトバウンド接続の更新

```
/subsystem=remoting/outbound-connection=new-outbound-connection:write-
attribute(name=uri,value=http://example.com)
```

### 新規アウトバウンド接続の作成

```
/subsystem=remoting/outbound-connection=new-outbound-connection:add(uri=http://example.com)
```

## アウトバウンド接続の削除

```
/subsystem=remoting/outbound-connection=new-outbound-connection:remove
```

アウトバウンド接続の設定に使用できる属性の完全リストは、「[アウトバウンド接続の属性](#)」を参照してください。

## 18.6. リモートアウトバウンド接続の設定

リモートアウトバウンド接続は、プロトコル、アウトバウンドソケットバインディング、ユーザー名、およびセキュリティーレルムによって指定されます。プロトコルは **remote**、**http-remoting**、**https-remoting** のいずれかになります。

### 既存のリモートアウトバウンド接続の更新

```
/subsystem=remoting/remote-outbound-connection=new-remote-outbound-connection:write-attribute(name=outbound-socket-binding-ref,value=outbound-socket-binding)
```

### 新規リモートアウトバウンド接続の作成

```
/subsystem=remoting/remote-outbound-connection=new-remote-outbound-connection:add(outbound-socket-binding-ref=outbound-socket-binding)
```

### リモートアウトバウンド接続の削除

```
/subsystem=remoting/remote-outbound-connection=new-remote-outbound-connection:remove
```

リモートアウトバウンド接続の設定に使用できる属性の完全リストは、「[リモートアウトバウンド接続の属性](#)」を参照してください。

## 18.7. ローカルアウトバウンド接続の設定

ローカルアウトバウンド接続はプロトコルが **local** のリモータリングアウトバウンド接続で、アウトバウンドソケットバインディングのみによって指定されます。

### 既存のローカルアウトバウンド接続の更新

```
/subsystem=remoting/local-outbound-connection=new-local-outbound-connection:write-attribute(name=outbound-socket-binding-ref,value=outbound-socket-binding)
```

### 新規ローカルアウトバウンド接続の作成

```
/subsystem=remoting/local-outbound-connection=new-local-outbound-connection:add(outbound-socket-binding-ref=outbound-socket-binding)
```

### ローカルアウトバウンド接続の削除

```
/subsystem=remoting/local-outbound-connection=new-local-outbound-connection:remove
```

ローカルアウトバウンド接続の設定に使用できる属性の完全リストは、「[ローカルアウトバウンド接続の属性](#)」を参照してください。

## 18.8. リモーティングの追加設定

**remoting** サブシステム外部に接続されるリモーティング要素が複数あります。

### IO ワーカー

以下のコマンドを使用してリモーティングの IO ワーカーを設定します。

```
/subsystem=remoting/configuration=endpoint:write-attribute(name=worker,
value=WORKER_NAME)
```

IO ワーカーの設定方法に関する詳細は「[ワーカーの設定](#)」を参照してください。

### ネットワークインターフェース

**remoting** サブシステムによって使用されるネットワークインターフェースは **public** インターフェースです。このインターフェースは他のサブシステムによっても使用されるため、変更する場合は十分注意してください。

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

管理対象ドメインでは、**public** インターフェースはホストごとに **host.xml** ファイルで定義されません。

### ソケットバインディング

**remoting** サブシステムによって使用されるデフォルトのソケットバインディングはポート **8080** にバインドされます。

ソケットバインディングおよびソケットバインディンググループの詳細は、「[ソケットバインディング](#)」を参照してください。

### EJB のリモーティングコネクタ参照

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

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

### セキュアなトランスポート設定

リモーティングトランスポートはクライアントの要求があれば STARTTLS を使用してセキュアな接続 (HTTPS、Secure Servlet など) を使用します。セキュアな接続とセキュアでない接続の両方で同じソケットバインディングまたはネットワークポートが使用されるため、サーバー側に追加の設定をする必要はありません。クライアントは必要に応じてセキュアなトランスポートまたはセキュアでないトランスポートを要求します。EJB、ORB、JMS プロバイダーなどのリモーティングを使用する JBoss EAP のコンポーネントはデフォルトでセキュアなインターフェースを使用します。



### 警告

STARTTLS はクライアントの要求があればセキュアな接続を有効にしますが、セキュアでない接続がデフォルトになります。本質的に、StartTLS は攻撃者がクライアントの要求を妨害し、要求を編集してセキュアでない接続を要求する中間者攻撃の対象になりやすい欠点があります。セキュアでない接続が適切なフォールバックである場合を除き、クライアントがセキュアな接続を取得できなかったときに適切に失敗するよう記述する必要があります。

## 第19章 IO サブシステムの設定

### 19.1. IO サブシステムの概要

io サブシステムは、Undertow や Remoting などの他のサブシステムによって使用される XNIO [ワーカー](#) と [バッファプール](#) を定義します。これらのワーカーやバッファプールは、io サブシステムの以下のコンポーネント内で定義されます。

#### IO サブシステムのデフォルト設定

```
<subsystem xmlns="urn:jboss:domain:io:3.0">
  <worker name="default"/>
  <buffer-pool name="default"/>
</subsystem>
```

### 19.2. ワーカーの設定

ワーカーは XNIO ワーカーインスタンスです。XNIO ワーカーインスタンスは、IO およびワーカースレッドの管理や SSL サポートなどの機能を提供する Java NIO API の抽象化レイヤーです。JBoss EAP はデフォルトで **default** という単一のワーカーを提供しますが、複数のワーカーを定義できます。

#### 既存のワーカーの更新

既存のワーカーを更新するには、以下を指定します。

```
/subsystem=io/worker=default:write-attribute(name=io-threads,value=10)
```

```
reload
```

#### 新規ワーカーの作成

新規ワーカーを作成するには、以下を指定します。

```
/subsystem=io/worker=newWorker:add
```

#### ワーカーの削除

ワーカーを削除するには、以下を指定します。

```
/subsystem=io/worker=newWorker:remove
```

```
reload
```

ワーカーの設定に使用できる属性の完全リストは「[IO サブシステムの属性](#)」の項を参照してください。

### 19.3. バッファプールの設定



#### 注記

IO バッファプールは 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で設定されていますが、今後のリリースで [Undertow バイトバッファプール](#) に置き換えられる予定です。

バッファープールはプールされた NIO バッファインスタンスです。バッファサイズの変更はアプリケーションのパフォーマンスに大きく影響します。通常、ほとんどのサーバーでは 16 K が理想のサイズになります。

### 既存のバッファープールの更新

既存のバッファープールを更新するには、以下を指定します。

```
/subsystem=io/buffer-pool=default:write-attribute(name=direct-buffers,value=true)
```

```
reload
```

### バッファープールの作成

新しいバッファープールを作成するには、以下を指定します。

```
/subsystem=io/buffer-pool=newBuffer:add
```

### バッファープールの削除

バッファープールを削除するには、以下を指定します。

```
/subsystem=io/buffer-pool=newBuffer:remove
```

```
reload
```

バッファープールの設定に使用できる属性の完全リストは「[IO サブシステムの属性](#)」の項を参照してください。

## 19.4. IO サブシステムの調整

**io** サブシステムのパフォーマンスを監視および最適化するための情報は、『[パフォーマンスチューニングガイド](#)』の「[IO サブシステムのチューニング](#)」の項を参照してください。

## 第20章 WEB サービスの設定

JBoss EAP では、管理コンソールまたは管理 CLI を使用して **webservices** システム経由でデプロイされた Web サービスの動作を設定できます。パブリッシュされたエンドポイントアドレスやハンドラーチェーンを設定できます。また、Web サービスのランタイム統計収集を有効にすることもできます。

詳細は、JBoss EAP 『[Developing Web Services Applications](#)』の『**Configuring the Web Services Subsystem**』を参照してください。



## 第21章 JAVASERVER FACES (JSF) の設定

### 21.1. JAVASERVER FACES の複数の JSF 実装

The JavaServer Faces (JSF) サブシステムでは、複数の JSF 実装を同じ JBoss EAP サーバーインスタンスにインストールできます。特に、JSF 仕様 2.1 以上を実装するすべてのバージョンの Sun Mojarra または Apache MyFaces をインストールできます。

#### 21.1.1. JSF 実装のインストール

以下の手順は、新しい JSF 実装を手作業でインストールし、デフォルトの実装にする方法になります。

1. JSF 実装 JAR ファイルの追加
2. JSF API JAR ファイルを追加 します。
3. JSF インジェクション JAR ファイルを追加 します。
4. MyFaces をインストールする場合は **commons-digester** JAR ファイルを追加 します。
5. デフォルトの JSF 実装を設定 します。

#### JSF 実装 JAR ファイルの追加

1. JFS 実装の **EAP\_HOME/modules/** ディレクトリーに適切なディレクトリー構造を作成します。

```
$ cd EAP_HOME/modules/
$ mkdir -p com/sun/jsf-impl/JSF_IMPL_NAME-JSF_VERSION
```



#### 注記

たとえば、Mojarra 2.2.11 の場合は **JSF\_IMPL\_NAME-JSF\_VERSION** を **mojarra-2.2.11** に置き換えます。

2. JSF 実装 JAR ファイルを **JSF\_IMPL\_NAME-JSF\_VERSION/** サブディレクトリーにコピーします。
3. **JSF\_IMPL\_NAME-JSF\_VERSION/** サブディレクトリーで、この **Mojarra** テンプレートまたはこの **MyFaces テンプレート** と似た **module.xml** ファイルを作成します。テンプレートを使用する場合は、置き換え可能な変数に適切な値を適用してください。

#### JSF API JAR ファイルの追加

1. JFS 実装の **EAP\_HOME/modules/** ディレクトリーに適切なディレクトリー構造を作成します。

```
$ cd EAP_HOME/modules/
$ mkdir -p javax/faces/api/JSF_IMPL_NAME-JSF_VERSION
```

2. JSF API JAR ファイルを **JSF\_IMPL\_NAME-JSF\_VERSION/** サブディレクトリーにコピーします。

3. **JSF\_IMPL\_NAME-JSF\_VERSION/** サブディレクトリーで、この **Mojarra** テンプレートまたはこの **MyFaces** テンプレート と似た **module.xml** ファイルを作成します。テンプレートを使用する場合は、置き換え可能な変数に適切な値を適用してください。

#### JSF インジェクション JAR ファイルの追加

1. JFS 実装の **EAP\_HOME/modules/** ディレクトリーに適切なディレクトリー構造を作成します。

```
$ cd EAP_HOME/modules/
$ mkdir -p org/jboss/as/jsf-injection/JSF_IMPL_NAME-JSF_VERSION
```

2. **wildfly-jsf-injection** および **weld-core-jsf** JAR ファイルを **EAP\_HOME/modules/system/layers/base/org/jboss/as/jsf-injection/main/** から **JSF\_IMPL\_NAME-JSF\_VERSION/** サブディレクトリーにコピーします。
3. **JSF\_IMPL\_NAME-JSF\_VERSION/** サブディレクトリーで、この **Mojarra** テンプレートまたはこの **MyFaces** テンプレート と似た **module.xml** ファイルを作成します。テンプレートを使用する場合は、置き換え可能な変数に適切な値を適用してください。

#### MyFaces の commons-digester JAR ファイルの追加

1. **commons-digester** JAR の **EAP\_HOME/modules/** ディレクトリーに適切なディレクトリー構造を作成します。

```
$ cd EAP_HOME/modules/
$ mkdir -p org/apache/commons/digester/main
```

2. **commons-digester** JAR ファイルをダウンロードし、**main/** サブディレクトリーにコピーします。
3. **main/** サブディレクトリーで、この テンプレート と似た **module.xml** ファイルを作成します。テンプレートを使用する場合は、置き換え可能な変数に適切な値を適用してください。

#### デフォルト JSF 実装の設定

1. 以下の管理 CLI コマンドを実行して、新しい JSF 実装をデフォルト実装として設定します。

```
/subsystem=jsf:write-attribute(name=default-jsf-impl-slot,value=JSF_IMPL_NAME-JSF_VERSION)
```

2. JBoss EAP サーバーを再起動し、変更を反映します。

### 21.1.2. マルチ JSF 実装のサポート

JBoss EAP 7.2 には、単一の JSF 実装で、Mojarra をベースとした **JSF 2.3** 実装が含まれています。

マルチ JSF は、同じ JBoss EAP サーバーで複数の JSF 実装およびバージョンのインストールを可能にします。この目的は、JSF 実装、MyFaces、および Mojarra すべてと、これらの実装の JSF 2.1 以上のバージョンをすべて使用できるようにすることです。マルチ JSF は、コンテナと完全統合された実装を提供するため、より効率的なアノテーション処理、ライフサイクル処理、およびその他の統合の利点を実現できます。

#### 21.1.2.1. マルチ JSF 実装の仕組み

各 JSF バージョンごとに新しいスロットが **com.sun.jsf-impl**、**javax.faces.api**、および **org.jboss.as.jsf-injection** 下のモジュールパスに作成されることで、マルチ JSF は機能します。**jsf** サブシステムが開始されると、モジュールパスをスキャンしてインストールされた JSF 実装をすべて見つけます。**jsf** サブシステムが指定されたコンテキストパラメーターが含まれる web アプリケーションをデプロイすると、スロットが作成されたモジュールをデプロイメントに追加します。

たとえば、MyFaces 2.2.12 がサーバー上にインストールされていることを仮定して、JSF アプリケーションは MyFaces 2.2.12 を使用すべきであることを示すには、以下のコンテキストパラメーターを追加する必要があります。

```
<context-param>
  <param-name>org.jboss.jbossfaces.JSF_CONFIG_NAME</param-name>
  <param-value>myfaces-2.2.12</param-value>
</context-param>
```

### 21.1.2.2. デフォルトの JSF 実装の変更

マルチ JSF 機能では、**jsf** サブシステムに **default-jsf-impl-slot** 属性が含まれています。この属性を使用すると、以下の手順のようにデフォルトの JSF 実装を変更することができます。

1. **write-attribute** コマンドを使用して、**default-jsf-impl-slot** 属性の値をアクティブな JSF 実装の1つに設定します。

```
/subsystem=jsf:write-attribute(name=default-jsf-impl-slot,value=JSF_IMPLEMENTATION)
```

2. 変更を反映するために、JBoss EAP サーバーを再起動します。

```
reload
```

インストールされている JSF 実装を確認するには、**list-active-jsf-impls** 操作を実行します。

```
/subsystem=jsf:list-active-jsf-impls
{
  "outcome" => "success",
  "result" => [
    "myfaces-2.1.12",
    "mojarra-2.2.0-m05",
    "main"
  ]
}
```

## 21.2. DOCTYPE 宣言の拒否

以下の管理 CLI コマンドを使用すると、JSF デプロイメントの **DOCTYPE** 宣言を拒否することができます。

```
/subsystem=jsf:write-attribute(name=disallow-doctype-decl,value=true)
reload
```

特定の JSF デプロイメントの設定をオーバーライドするには、デプロイメントの **web.xml** ファイルに **com.sun.faces.disallowDoctypeDecl** コンテキストパラメーターを追加します。

```
<context-param>
```

```
<param-name>com.sun.faces.disallowDoctypeDecl</param-name>  
<param-value>>false</param-value>  
</context-param>
```

## 第22章 バッチアプリケーションの設定

JBoss EAP 7には [JSR-352](#) に定義されている Java バッチアプリケーションのサポートが導入されました。バッチアプリケーションを実行するための環境を設定し、**batch-jberet** サブシステムを使用してバッチジョブを管理できます。

バッチアプリケーションの開発に関する詳細は、JBoss EAP 『[開発ガイド](#)』の「[Java バッチアプリケーション開発](#)」を参照してください。

### 22.1. BATCH ジョブの設定

JBeret 実装を基にした **batch-jberet** サブシステムを使用してバッチジョブを設定できます。

デフォルトの **batch-jberet** サブシステム設定は、インメモリージョブリポジトリとデフォルトのスレッドプールの設定を定義します。

```
<subsystem xmlns="urn:jboss:domain:batch-jberet:2.0">
  <default-job-repository name="in-memory"/>
  <default-thread-pool name="batch"/>
  <job-repository name="in-memory">
    <in-memory/>
  </job-repository>
  <thread-pool name="batch">
    <max-threads count="10"/>
    <keepalive-time time="30" unit="seconds"/>
  </thread-pool>
</subsystem>
```

デフォルトでは、サーバーの一時停止中に停止したバッチジョブはサーバーの再開時に再度開始されます。**restart-jobs-on-resume** プロパティを **false** に設定すると **STOPPED** 状態のジョブをそのままにすることができます。

```
/subsystem=batch-jberet:write-attribute(name=restart-jobs-on-resume,value=false)
```

バッチジョブリポジトリおよびスレッドプールを設定することもできます。

#### 22.1.1. バッチジョブリポジトリの設定

本項では、管理 CLI を使用してバッチジョブ情報を保存するインメモリーおよび JDBC ジョブリポジトリを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Batch (JBeret)** と選択し、**表示** をクリックして左側のメニューで **In Memory** または **JDBC** のいずれかを選択すると、ジョブリポジトリを設定することができます。

インメモリージョブリポジトリの追加

バッチジョブ情報をメモリーに保存するジョブリポジトリを追加できます。

```
/subsystem=batch-jberet/in-memory-job-repository=REPOSITORY_NAME:add
```

JDBC ジョブリポジトリ

バッチジョブ情報をデータベースに保存するジョブリポジトリを追加できます。データソースの名前を指定してデータベースに接続する必要があります。

```
/subsystem=batch-jberet/jdbc-job-repository=REPOSITORY_NAME:add(data-source=DATASOURCE)
```

デフォルトのジョブリポジトリの設定

インメモリーまたは JDBC ジョブリポジトリをバッチアプリケーションのデフォルトのジョブリポジトリとして設定できます。

```
/subsystem=batch-jberet:write-attribute(name=default-job-repository,value=REPOSITORY_NAME)
```

サーバーをリロードする必要があります。

```
reload
```

## 22.1.2. バッチスレッドプールの設定

本項では、管理 CLI を使用してバッチジョブに使用するスレッドプールとスレッドファクトリーを設定する方法を説明します。管理コンソールでは、**Configuration** → **Subsystems** → **Batch (JBeret)** と選択し、表示をクリックして左側のメニューで **Thread Factory** または **Thread Pool** のいずれかを選択すると、スレッドプールとスレッドファクトリーを設定できます。

スレッドプールの設定

スレッドプールを追加するときに **max-threads** を指定する必要があります。パーティションのジョブが想定どおりに実行されるように 2 つのスレッドが予約されているため、**3** よりも大きい値を常に設定してください。

1. スレッドプールを追加します。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:add(max-threads=10)
```

2. 必要な場合は **keepalive-time** の値を設定します。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:write-attribute(name=keepalive-time,value={time=60,unit=SECONDS})
```

スレッドファクトリーの使用

1. スレッドファクトリーを追加します。

```
/subsystem=batch-jberet/thread-factory=THREAD_FACTORY_NAME:add
```

2. スレッドファクトリーの属性を設定します。
  - **group-name** - このスレッドファクトリーに作成するスレッドグループの名前。
  - **priority** - 作成されたスレッドの優先度。
  - **thread-name-pattern** - スレッドの名前の作成に使用されるテンプレート。以下のパターンを使用できます。
    - **%%** - パーセント記号
    - **%t** - ファクトリーごとのスレッドシーケンス番号
    - **%g** - グローバルスレッドシーケンス番号

- %f - ファクトリーシーケンス番号
- %i - スレッド ID

3. スレッドファクトリーをスレッドプールに割り当てます。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:write-attribute(name=thread-factory,value=THREAD_FACTORY_NAME)
```

サーバーをリロードする必要があります。

```
reload
```

デフォルトスレッドプールの設定

別のスレッドプールをデフォルトのスレッドプールとして設定できます。

```
/subsystem=batch-jberet:write-attribute(name=default-thread-pool,value=THREAD_POOL_NAME)
```

サーバーをリロードする必要があります。

```
reload
```

スレッドプールの統計表示

**read-resource** 管理 CLI 操作を使用するとバッチスレッドプールのランタイム情報を表示できます。このランタイム情報を表示するには **include-runtime=true** パラメーターを使用する必要があります。

```
/subsystem=batch-jberet/thread-pool=THREAD_POOL_NAME:read-resource(include-runtime=true)
{
  "outcome" => "success",
  "result" => {
    "active-count" => 0,
    "completed-task-count" => 0L,
    "current-thread-count" => 0,
    "keepalive-time" => undefined,
    "largest-thread-count" => 0,
    "max-threads" => 15,
    "name" => "THREAD_POOL_NAME",
    "queue-size" => 0,
    "rejected-count" => 0,
    "task-count" => 0L,
    "thread-factory" => "THREAD_FACTORY_NAME"
  }
}
```

管理コンソールの **Runtime** タブで **Batch** サブシステムを選択して、バッチスレッドプールのランタイム情報を表示することもできます。

## 22.2. バッチジョブの管理

デプロイメントの **batch-jberet** サブシステムリソースを使用すると、バッチジョブを起動、停止、および再起動でき、実行詳細を表示することもできます。バッチジョブは [管理 CLI](#) または [管理コンソール](#) から管理できます。

### 管理 CLI からのバッチジョブの管理

### バッチジョブの再開

**STOPPED** または **FAILED** 状態のジョブを再開するには、実行 ID を指定し、任意でバッチジョブの再開時に使用するプロパティを指定します。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:restart-job(execution-id=EXECUTION_ID,properties={PROPERTY=VALUE})
```

実行 ID はジョブインスタンスが最後に実行された ID である必要があります。

### バッチジョブの開始

バッチジョブを開始するには、ジョブ XML ファイルを指定し、任意でバッチジョブの再開時に使用するプロパティを指定します。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:start-job(job-xml-name=JOB_XML_NAME,properties={PROPERTY=VALUE})
```

### バッチジョブの停止

実行中のバッチジョブを停止するには、実行 ID を指定します。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:stop-job(execution-id=EXECUTION_ID)
```

### バッチジョブ実行詳細の表示

バッチジョブ実行の詳細を表示することができます。このランタイム情報を表示するには、**read-resource** 操作で **include-runtime=true** パラメーターを使用する必要があります。

```
/deployment=DEPLOYMENT_NAME/subsystem=batch-jberet:read-resource(recursive=true,include-runtime=true)
{
  "outcome" => "success",
  "result" => {"job" => {"import-file" => {
    "instance-count" => 2,
    "running-executions" => 0,
    "execution" => {
      "2" => {
        "batch-status" => "COMPLETED",
        "create-time" => "2016-04-11T22:03:12.708-0400",
        "end-time" => "2016-04-11T22:03:12.718-0400",
        "exit-status" => "COMPLETED",
        "instance-id" => 58L,
        "last-updated-time" => "2016-04-11T22:03:12.719-0400",
        "start-time" => "2016-04-11T22:03:12.708-0400"
      },
      "1" => {
        "batch-status" => "FAILED",
        "create-time" => "2016-04-11T21:57:17.567-0400",
        "end-time" => "2016-04-11T21:57:17.596-0400",
        "exit-status" => "Error : org.hibernate.exception.ConstraintViolationException: could not execute statement",
        "instance-id" => 15L,
        "last-updated-time" => "2016-04-11T21:57:17.597-0400",
        "start-time" => "2016-04-11T21:57:17.567-0400"
      }
    }
  }
}
```



```

}
}}
}

```

### 管理コンソールからのバッチジョブの管理

管理コンソールからバッチジョブを管理するには、**Runtime** タブでサーバーを選択し、**Batch (JBeret)** を選択してリストからジョブを選びます。

#### バッチジョブの再開

実行を選択して **Restart** をクリックし、**STOPPED** ジョブを再開します。

#### バッチジョブの開始

ジョブを選択してドロップダウンメニューで **Start** を選択し、バッチジョブの新たな実行を開始します。

#### バッチジョブの停止

実行を選択して **Stop** をクリックし、バッチジョブの実行を停止します。

#### バッチジョブ実行詳細の表示

表にリストされる各実行のジョブ実行詳細が表示されます。

## 22.3. バッチジョブのセキュリティー設定

**batch-jberet** サブシステムを設定すると、Elytron セキュリティードメインでバッチジョブを実行することができます。これにより、バッチジョブをセキュアに一時停止でき、同じセキュアなアイデンティティーで再開できます。たとえば、**batch-jberet** サブシステムを使用して、バッチジョブを開始するためにセキュアな RESTful エンドポイントが作成されます。RESTful エンドポイントと **batch-jberet** サブシステムの両方が同じセキュリティードメインを使用してセキュア化された場合、または **batch-jberet** セキュリティードメインが RESTful エンドポイントのセキュリティードメインドメインを信頼した場合、このように開始されたバッチジョブはセキュアに一時停止され、同じセキュアなアイデンティティーによって再開されます。

以下の管理 CLI コマンドを使用して **security-domain** 属性を更新し、バッチジョブのセキュリティーを設定します。

```
/subsystem=batch-jberet:write-attribute(name=security-domain, value=ExampleDomain)
```

```
reload
```



### 注記

バッチジョブには、**org.wildfly.extension.batch.jberet.deployment.BatchPermission** パーミッションが必要です。これは、**javax.batch.operations.JobOperator** に対応する **start**、**stop**、**restart**、**abandon** および **read** を提供します。**default-permission-mapper** マッパーは **org.wildfly.extension.batch.jberet.deployment.BatchPermission** パーミッションを提供します。

## 第23章 NAMING サブシステムの設定

### 23.1. NAMING サブシステム

**naming** サブシステムは JBoss EAP の JNDI 実装を提供します。このサブシステムを設定して、[グローバル JNDI 名前空間のエントリをバインド](#)することができます。さらに、このサブシステムを設定してリモート JNDI を [アクティブ](#)または [非アクティブ](#)にすることもできます。

以下は、すべての要素と属性が指定された **naming** サブシステムの XML 設定例になります。

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
  <bindings>
    <simple name="java:global/simple-integer-binding" value="100" type="int" />
    <simple name="java:global/jboss.org/docs/url" value="https://docs.jboss.org"
type="java.net.URL" />
    <object-factory name="java:global/foo/bar/factory" module="org.foo.bar"
class="org.foo.bar.ObjectFactory" />
    <external-context name="java:global/federation/ldap/example"
class="javax.naming.directory.InitialDirContext" cache="true">
      <environment>
        <property name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory" />
        <property name="java.naming.provider.url" value="ldap://ldap.example.com:389" />
        <property name="java.naming.security.authentication" value="simple" />
        <property name="java.naming.security.principal" value="uid=admin,ou=system" />
        <property name="java.naming.security.credentials" value="secret" />
      </environment>
    </external-context>
    <lookup name="java:global/new-alias-name" lookup="java:global/original-name" />
  </bindings>
  <remote-naming/>
</subsystem>
```

### 23.2. グローバルバインディングの設定

**naming** サブシステムは、エントリを **java:global**、**java:jboss**、または **java** グローバル JNDI 名前空間へバインドできるようにしますが、標準のポータブルな **java:global** 名前空間を使用することが推奨されます。

グローバルバインディングは **naming** サブシステムの **<bindings>** 要素で設定されます。以下の 4 種類のバインディングがサポートされます。

- [シンプルバインディング](#)
- [オブジェクトファクトリーバインディング](#)
- [外部コンテキストバインディング](#)
- [バインディングルックアップエイリアス](#)

#### シンプルバインディングの設定

**simple** XML 設定要素は、プリミティブまたは **java.net.URL** エントリにバインドします。

- **name** 属性は必須で、エントリのターゲット JNDI 名を指定します。

- **value** 属性は必須で、エントリーの値を定義します。
- 任意の **type** 属性はエントリーの値の型を指定し、デフォルトは **java.lang.String** になります。 **java.lang.String** の他に、 **int** または **java.lang.Integer**、 および **java.net.URL** などのプリミティブ型や対応するオブジェクトラッパークラスを指定できます。

以下に、シンプルバインディングを作成する管理 CLI コマンドの例を示します。

```
/subsystem=naming/binding=java\:global\simple-integer-binding:add(binding-type=simple, type=int, value=100)
```

### 結果の XML 設定

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
  <bindings>
    <simple name="java:global/simple-integer-binding" value="100" type="int"/>
  </bindings>
  <remote-naming/>
</subsystem>
```

以下のコマンドを使用してバインディングを削除します。

```
/subsystem=naming/binding=java\:global\simple-integer-binding:remove
```

### バインディングオブジェクトファクトリー

**object-factory** XML 設定要素は **javax.naming.spi.ObjectFactory** エントリーをバインドします。

- **name** 属性は必須で、エントリーのターゲット JNDI 名を指定します。
- **class** 属性は必須で、オブジェクトファクトリーの Java タイプを定義します。
- **module** 属性は必須で、オブジェクトファクトリーの Java クラスをロードできる JBoss Module ID を指定します。
- 任意の **environment** 子要素は、カスタム環境をオブジェクトファクトリーに提供するために使用できます。

以下に、オブジェクトファクトリーバインディングを作成する管理 CLI コマンドの例を示します。

```
/subsystem=naming/binding=java\:global\foo\bar\factory:add(binding-type=object-factory, module=org.foo.bar, class=org.foo.bar.ObjectFactory, environment=[p1=v1, p2=v2])
```

### 結果の XML 設定

```
<subsystem xmlns="urn:jboss:domain:naming:2.0">
  <bindings>
    <object-factory name="java:global/foo/bar/factory" module="org.foo.bar"
class="org.foo.bar.ObjectFactory">
      <environment>
        <property name="p1" value="v1" />
        <property name="p2" value="v2" />
      </environment>
    </object-factory>
  </bindings>
</subsystem>
```

```

    </object-factory>
  </bindings>
</subsystem>

```

以下のコマンドを使用してバインディングを削除します。

```
/subsystem=naming/binding=java\:global\foo\bar\factory:remove
```

### 外部コンテンツのバインド

LDAP コンテキストなどの外部 JNDI コンテキストのフェデレーションは、**external-context** XML 設定要素を使用して実行されます。

- **name** 属性は必須で、エントリーのターゲット JNDI 名を指定します。
- **class** 属性は必須で、フェデレートされたコンテキストの作成に使用される Java 初期ネーミングコンテキストタイプを示します。このようなタイプには、単一の環境マップ引数を持つコンストラクターが必要なことに注意してください。
- 任意の **module** 属性は、外部 JNDI コンテキストが必要とするすべてのクラスをロードできる JBoss Module ID を指定します。
- オプションの **cache** 属性は外部コンテキストインスタンスをキャッシュする必要があるかどうかを示し、デフォルトは **false** になります。
- 任意の **environment** 子要素は、外部コンテキストを検索するために必要なカスタム環境を提供するために使用されます。

以下に、外部コンテキストバインディングを作成する管理 CLI コマンドの例を示します。

```

/subsystem=naming/binding=java\:global\federation\ldap\example:add(binding-type=external-context, cache=true, class=javax.naming.directory.InitialDirContext, module=org.jboss.as.naming, environment=[java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory, java.naming.provider.url="ldap://ldap.example.com:389", java.naming.security.authentication=simple, java.naming.security.principal="uid=admin,ou=system", java.naming.security.credentials=secret])

```

### 結果の XML 設定

```

<subsystem xmlns="urn:jboss:domain:naming:2.0">
  <bindings>
    <external-context name="java:global/federation/ldap/example" module="org.jboss.as.naming" class="javax.naming.directory.InitialDirContext" cache="true">
      <environment>
        <property name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
        <property name="java.naming.provider.url" value="ldap://ldap.example.com:389"/>
        <property name="java.naming.security.authentication" value="simple"/>
        <property name="java.naming.security.principal" value="uid=admin,ou=system"/>
        <property name="java.naming.security.credentials" value="secret"/>
      </environment>
    </external-context>
  </bindings>
</subsystem>

```

以下のコマンドを使用してバインディングを削除します。

```
/subsystem=naming/binding=java\:global/federation/ldap/example:remove
```

## 注記

JNDI プロバイダーのリソースルックアップが **lookup(Name)** メソッドを適切に実装しないと、「javax.naming.InvalidNameException: Only support CompoundName names」エラーが発生することがあります。

以下のプロパティを追加せずに、外部コンテキスト環境が **lookup(String)** メソッドを使用するように指定すると、この問題を回避できる可能性があります、パフォーマンスが劣化します。

```
<property name="org.jboss.as.naming.lookup.by.string" value="true"/>
```

## ルックアップエイリアスのバインド

**lookup** 要素を使用すると、既存のエントリーを追加の名前またはエイリアスにバインドできます。

- **name** 属性は必須で、エントリーのターゲット JNDI 名を指定します。
- **lookup** 属性は必須で、ソース JNDI 名を示します。

以下に、既存のエントリーをエイリアスにバインドする管理 CLI コマンドの例を示します。

```
/subsystem=naming/binding=java\:global/new-alias-name:add(binding-type=lookup,
lookup=java\:global/original-name)
```

## 結果の XML 設定

```
<lookup name="java\:global/new-alias-name" lookup="java\:global/original-name" />
```

以下のコマンドを使用してバインディングを削除します。

```
/subsystem=naming/binding=java\:global/c:remove
```

## 23.3. JNDI バンディングの動的な変更

JBoss EAO 7.1 には、サーバーのリロードや再起動を強制せずに JNDI バインディングを動的に変更する機能が追加されました。この機能は、バージョンの更新、テストの要件、またはアプリケーション機能の更新によってネットワークサービスのエンドポイントが動的に設定される場合に便利です。

JNDI バインディングを更新するには、**rebind** 操作を使用します。**rebind** 操作は **add** 操作と同じ引数を取ります。このコマンドは、**external-context** バンディングタイプ以下のすべてのバインディングタイプで動作します。外部コンテキストバインディングは、モジュラーサービスコンテナ (MSC) の状態に影響する追加の依存関係を必要とするため、サービスを再起動せずに外部コンテキストバインディングを再起動することはできません。

以下のコマンドは、「[シンプルバインディングの設定](#)」の例で定義した JNDI バインディングを動的に変更します。

```
/subsystem=naming/binding=java\:global/simple-integer-binding:rebind(binding-type=simple,
type=int, value=200)
```

**naming** サブシステムでグローバルバインディングを設定する方法に関する詳細は、「[グローバルバインディングの設定](#)」を参照してください。

## 23.4. リモート JNDI インターフェースの設定

リモート JNDI インターフェースは、クライアントがリモート JBoss EAP インスタンスでエントリをルックアップできるようにします。**naming** サブシステムを設定すると、このインターフェースをアクティブまたは非アクティブにすることができます (デフォルトではアクティブになります)。リモート JNDI インターフェースは **<remote-naming>** 要素を使用して設定されます。

以下の管理 CLI コマンドを使用して、リモート JNDI インターフェースをアクティブまたは非アクティブにします。

```
/subsystem=naming/service=remote-naming:add
```

以下の管理 CLI コマンドを使用して、リモート JNDI インターフェースを非アクティブにします。

```
/subsystem=naming/service=remote-naming:remove
```



### 注記

リモート JNDI 上では **java:jboss/exported** コンテキスト内のエントリのみアクセスできます。

## 第24章 高可用性の設定

### 24.1. 高可用性

JBoss EAP はデプロイされた Java EE アプリケーションの可用性を保証するために以下の高可用性サービスを提供します。

#### ロードバランシング

複数のサーバーにワークロードを分散し、サービスが大量のリクエストを処理できるようにします。リクエストが大量に発生しても、クライアントはサービスからタイムリーに応答を受け取ることができます。

#### フェイルオーバー

ハードウェアやネットワークの障害が発生してもクライアントのサービスへのアクセスが中断しないようにします。サービスに障害が発生すると、別のクラスターメンバーがクライアントのリクエストを引き継ぎ、処理が続行されます。

クラスタリングはこれらすべての機能を包括する言葉です。クラスターのメンバーを設定すると、ロードバランシングと呼ばれるワークロードの共有や、フェイルオーバーと呼ばれる他のクラスターメンバーの障害時におけるクライアント処理の引き継ぎを行うことができます。



#### 注記

選択した JBoss EAP の操作モード (スタンドアロンサーバーまたは管理対象ドメイン) によってサーバーの管理方法が異なることに注意してください。操作モードに関係なく JBoss EAP で高可用性サービスを設定できます。

JBoss EAP は、さまざまなコンポーネントを使用した異なるレベルの高可用性をサポートします。高可用性を実現できるランタイムのコンポーネントおよびアプリケーションの一部は以下のとおりです。

- アプリケーションサーバーのインスタンス
- 内部 JBoss Web Server、Apache HTTP Server、Microsoft IIS、または Oracle iPlanet Web Server と併用される Web アプリケーション
- ステートフルおよびステートレスセッション Enterprise JavaBean (EJB)
- シングルサインオン (SSO) メカニズム
- HTTP セッション
- JMS サービスおよびメッセージ駆動型 Bean (MDB)
- シングルトン MSC サービス
- シングルトンデプロイメント

JBoss EAP では、**jgroups**、**infinispan**、および **modcluster** サブシステムによってクラスタリングが使用できるようになります。**ha** および **full-ha** プロファイルではこれらのシステム有効になっています。JBoss EAP では、これらのサービスは必要に応じて起動およびシャットダウンしますが、分散可能と設定されたアプリケーションがサーバーにデプロイされた場合のみ起動します。

アプリケーションを分散可能とする方法は、JBoss EAP 『[開発ガイド](#)』を参照してください。

## 24.2. JGROUPS を用いたクラスター通信

### 24.2.1. JGroups

JGroups は信頼できるメッセージングのためのツールキットで、お互いにメッセージを送信するノードを持つクラスターを作成するために使用できます。

**jgroups** サブシステムは JBoss EAP で高可用性サービスのグループ通信サポートを提供します。これにより、名前付きのチャンネルおよびプロトコルスタックを設定でき、チャンネルのランタイム統計を表示することもできます。**jgroups** サブシステムは高可用性の機能を提供する設定を使用する場合に使用できます (管理対象ドメインでは **ha** や **full-ha** プロファイル、スタンドアロンサーバーは **standalone-ha.xml** や **standalone-full-ha.xml** 設定ファイルなど)。

JBoss EAP には 2 つの JGroups スタックが事前に設定されています。

#### udp

クラスターのノードは UDP (User Datagram Protocol) マルチキャストを使用してお互いに通信します。これはデフォルトのスタックです。

#### tcp

クラスターのノードは TCP (Transmission Control Protocol) を使用してお互いに通信します。



#### 注記

TCP はエラーのチェック、パケットの順番、および輻輳制御を処理するため、TCP のオーバーヘッドは UDP よりも大きく、速度も遅くなると考えられます。JGroups は UDP のこれらの機能を処理しますが、TCP はこれらの機能を独自で処理します。信頼できないネットワークや輻輳度の高いネットワークで JGroups を使用する場合やマルチキャストが使用できない場合は、TCP を選択するとよいでしょう。

事前設定されたスタックを使用できますが、システムの要件に合うように独自に定義をすることもできます。使用できるプロトコルとそれらの属性については、以下の項を参照してください。

- [JGroups サブシステムの属性](#)
- [JGroups プロトコル](#)

### 24.2.2. デフォルトの JGroups チャンネルが TCP を使用するよう設定

デフォルトでは、クラスターノードは UDP プロトコルを使用して通信します。デフォルトの **ee** JGroups は事前定義された **udp** プロトコルスタックを使用します。

```
<channels default="ee">
  <channel name="ee" stack="udp"/>
</channels>
<stacks>
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp"/>
    <protocol type="PING"/>
    ...
  </stack>
  <stack name="tcp">
    <transport type="TCP" socket-binding="jgroups-tcp"/>
    <protocol type="MPING" socket-binding="jgroups-mping"/>
  </stack>
</stacks>
```



```
...
</stack>
</stacks>
```

一部のネットワークでは TCP のみを使用できます。以下の管理 CLI コマンドを使用して、**ee** チャネルが事前設定された **tcp** スタックを使用するようにします。

```
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=tcp)
```

このデフォルトの **tcp** スタックは、IP マルチキャストを使用して初期クラスターメンバーシップを検出する **MPING** プロトコルを使用します。他のメンバーシップ検出プロトコルに対してスタックを設定する場合は以下の項を参照してください。

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

### 24.2.3. TCCPING の設定

この手順は **TCCPING** プロトコルを使用する新しい JGroups スタックを作成し、静的クラスターメンバーシップのリストを定義します。ベーススクリプトは、**tcpping** スタックを作成し、この新しいスタックを使用するようデフォルトの **ee** チャネルを設定します。このスクリプトの管理 CLI コマンドは環境に合わせてカスタマイズする必要があり、バッチで処理されます。

1. 以下のスクリプトをテキストエディターにコピーし、ローカルファイルシステムに保存します。

```
# Define the socket bindings
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-
binding=jgroups-host-a:add(host=HOST_A,port=7600)
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-
binding=jgroups-host-b:add(host=HOST_B,port=7600)
batch
# Add the tcpping stack
/subsystem=jgroups/stack=tcpping:add
/subsystem=jgroups/stack=tcpping/transport=TCP:add(socket-binding=jgroups-tcp)
/subsystem=jgroups/stack=tcpping/protocol=TCCPING:add(socket-bindings=[jgroups-host-
a,jgroups-host-b])
/subsystem=jgroups/stack=tcpping/protocol=MERGE3:add
/subsystem=jgroups/stack=tcpping/protocol=FD SOCK:add
/subsystem=jgroups/stack=tcpping/protocol=FD_ALL:add
/subsystem=jgroups/stack=tcpping/protocol=VERIFY_SUSPECT:add
/subsystem=jgroups/stack=tcpping/protocol=pbcast.NAKACK2:add
/subsystem=jgroups/stack=tcpping/protocol=UNICAST3:add
/subsystem=jgroups/stack=tcpping/protocol=pbcast.STABLE:add
/subsystem=jgroups/stack=tcpping/protocol=pbcast.GMS:add
/subsystem=jgroups/stack=tcpping/protocol=MFC:add
/subsystem=jgroups/stack=tcpping/protocol=FRAG2:add
# Set tcpping as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=tcpping)
run-batch
reload
```

定義されたプロトコルの順番が重要になることに注意してください。また、**add-index** の値を **add** コマンドに渡すと、特定のインデックスでプロトコルを挿入できます。インデックスはゼロベースであるため、以下の管理 CLI コマンドは **UNICAST3** プロトコルを 7 つ目のプロトコルとして追加します。

```
/subsystem=jgroups/stack=tccping/protocol=UNICAST3:add(add-index=6)
```

2. 環境に合わせてスクリプトを変更します。

- 管理対象ドメインで実行している場合は、**/subsystem=jgroups** コマンドの前に **/profile=PROFILE\_NAME** を追加し、更新するプロファイルを指定する必要があります。
- 以下のプロパティを環境に合わせて調整します。
  - **socket-bindings**: ウェルノウンとして見なされ、最初のメンバーシップの検索に利用できるホストとポートの組み合わせのカンマ区切りリスト。ソケットバインディングの定義に関する詳細は、「[ソケットバインディングの設定](#)」を参照してください
  - **initial\_hosts**: ウェルノウンとして見なされ、最初のメンバーシップの検索に使用できる **HOST[PORT]** という構文を使用したホストとポートの組み合わせのカンマ区切りリスト (例: **host1[1000],host2[2000]**)。
  - **port\_range**: このプロパティは、**initial\_hosts** ポート範囲を指定した値の分拡張するために使用されます。たとえば、**initial\_hosts** を **host1[1000],host2[2000]** に設定し、**port\_range** を 1 に設定した場合 **initial\_hosts** 設定は **host1[1000],host1[1001],host2[2000],host2[2001]** に拡張されます。このプロパティは **initial\_hosts** プロパティと併用した場合のみ有効です。

3. スクリプトファイルを管理 CLI に渡してスクリプトを実行します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/SCRIPT_NAME
```

TCCPING スタックが使用できるようになり、ネットワークの通信に TCP が使用されます。

#### 24.2.4. TCPGOSSIP の設定

この手順は、**TCPGOSSIP** プロトコルを使用する新しい JGroups スタックを作成し、外部ゴシップルーターを使用してクラスターのメンバーを検索します。ベーススクリプトは、**tcpgossip** スタックを作成し、この新しいスタックを使用するようデフォルトの **ee** チャネルを設定します。このスクリプトの管理 CLI コマンドは環境に合わせてカスタマイズする必要があり、バッチで処理されます。

1. 以下のスクリプトをテキストエディターにコピーし、ローカルファイルシステムに保存します。

```
# Define the socket bindings
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=jgroups-host-a:add(host=HOST_A,port=13001)
batch
# Add the tcpgossip stack
/subsystem=jgroups/stack=tcpgossip:add
/subsystem=jgroups/stack=tcpgossip/transport=TCP:add(socket-binding=jgroups-tcp)
/subsystem=jgroups/stack=tcpgossip/protocol=TCPGOSSIP:add(socket-bindings=[jgroups-host-a])
/subsystem=jgroups/stack=tcpgossip/protocol=MERGE3:add
/subsystem=jgroups/stack=tcpgossip/protocol=FD_SOCKET:add
```

```

/subsystem=jgroups/stack=tcpgossip/protocol=FD_ALL:add
/subsystem=jgroups/stack=tcpgossip/protocol=VERIFY_SUSPECT:add
/subsystem=jgroups/stack=tcpgossip/protocol=pbcast.NAKACK2:add
/subsystem=jgroups/stack=tcpgossip/protocol=UNICAST3:add
/subsystem=jgroups/stack=tcpgossip/protocol=pbcast.STABLE:add
/subsystem=jgroups/stack=tcpgossip/protocol=pbcast.GMS:add
/subsystem=jgroups/stack=tcpgossip/protocol=MFC:add
/subsystem=jgroups/stack=tcpgossip/protocol=FRAG2:add
# Set tcpgossip as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=tcpgossip)
run-batch
reload

```

定義されたプロトコルの順番が重要になることに注意してください。また、**add-index** の値を **add** コマンドに渡すと、特定のインデックスでプロトコルを挿入できます。インデックスはゼロベースであるため、以下の管理 CLI コマンドは **UNICAST3** プロトコルを 7 つ目のプロトコルとして追加します。

```

/subsystem=jgroups/stack=tcpgossip/protocol=UNICAST3:add(add-index=6)

```

## 2. 環境に合わせてスクリプトを変更します。

- 管理対象ドメインで実行している場合は、**/subsystem=jgroups** コマンドの前に **/profile=PROFILE\_NAME** を追加し、更新するプロファイルを指定する必要があります。
- 以下のプロパティを環境に合わせて調整します。
  - **socket-bindings**: ウェルノウンとして見なされ、最初のメンバーシップの検索に利用できるホストとポートの組み合わせのカンマ区切りリスト。ソケットバインディングの定義に関する詳細は、「[ソケットバインディングの設定](#)」を参照してください
  - **initial\_hosts**: ウェルノウンとして見なされ、最初のメンバーシップの検索に使用できる **HOST[PORT]** という構文を使用したホストとポートの組み合わせのカンマ区切りリスト (例: **host1[1000],host2[2000]**)。
  - **port\_range**: このプロパティは、**initial\_hosts** ポート範囲を指定した値の分拡張するために使用されます。たとえば、**initial\_hosts** を **host1[1000],host2[2000]** に設定し、**port\_range** を 1 に設定した場合 **initial\_hosts** 設定は **host1[1000],host1[1001],host2[2000],host2[2001]** に拡張されます。このプロパティは **initial\_hosts** プロパティと併用した場合のみ有効です。
  - **reconnect\_interval**: 接続が切断されたスタブがゴシッフルーターへの再接続を試みる間隔 (ミリ秒単位)。
  - **sock\_conn\_timeout**: ソケット作成の最大時間。デフォルトは **1000** ミリ秒です。
  - **sock\_read\_timeout**: 読み取りがブロックされる最大時間 (ミリ秒単位)。0 を値として指定すると無制限にブロックされます。

## 3. スクリプトファイルを管理 CLI に渡してスクリプトを実行します。

```

$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/SCRIPT_NAME

```

TCPGOSSIP スタックが使用できるようになり、ネットワークの通信に TCP が使用されます。このスタックはゴシッフルーターと使用するように設定されるため、JGroups メンバーは他のクラスターメンバーを見つけることができます。

## 24.2.5. JDBC\_PING の設定

**JDBC\_PING** プロトコルを使用して、クラスターでメンバーシップを管理および検出できます。

JDBC\_PING はデータソースに指定されたデータベースを使用して、クラスターのメンバーを一覧表示します。

1. クラスターメンバーシップの管理に使用するデータベースに接続するデータソースを作成します。
2. 以下のスクリプトをテキストエディターにコピーし、ローカルファイルシステムに保存します。

```
batch
# Add the JDBC_PING stack
/subsystem=jgroups/stack=JDBC_PING:add
/subsystem=jgroups/stack=JDBC_PING/transport=TCP:add(socket-binding=jgroups-tcp)
/subsystem=jgroups/stack=JDBC_PING/protocol=JDBC_PING:add(data-
source=ExampleDS)
/subsystem=jgroups/stack=JDBC_PING/protocol=MERGE3:add
/subsystem=jgroups/stack=JDBC_PING/protocol=FD SOCK:add
/subsystem=jgroups/stack=JDBC_PING/protocol=FD_ALL:add
/subsystem=jgroups/stack=JDBC_PING/protocol=VERIFY_SUSPECT:add
/subsystem=jgroups/stack=JDBC_PING/protocol=pbcast.NAKACK2:add
/subsystem=jgroups/stack=JDBC_PING/protocol=UNICAST3:add
/subsystem=jgroups/stack=JDBC_PING/protocol=pbcast.STABLE:add
/subsystem=jgroups/stack=JDBC_PING/protocol=pbcast.GMS:add
/subsystem=jgroups/stack=JDBC_PING/protocol=MFC:add
/subsystem=jgroups/stack=JDBC_PING/protocol=FRAG2:add
# Set JDBC_PING as the stack for the ee channel
/subsystem=jgroups/channel=ee:write-attribute(name=stack,value=JDBC_PING)
run-batch
reload
```

定義されたプロトコルの順番が重要になることに注意してください。また、**add-index** の値を **add** コマンドに渡すと、特定のインデックスでプロトコルを挿入できます。インデックスはゼロベースであるため、以下の管理 CLI コマンドは **UNICAST3** プロトコルを 7 つ目のプロトコルとして追加します。

```
/subsystem=jgroups/stack=JDBC_PING/protocol=UNICAST3:add(add-index=6)
```

3. 環境に合わせてスクリプトを変更します。
  - 管理対象ドメインで実行している場合は、**/subsystem=jgroups** コマンドの前に **/profile=PROFILE\_NAME** を追加し、更新するプロファイルを指定する必要があります。
  - 「**ExampleDS**」を、手順 1 で定義したデータソースの名前に置き換えます。
4. スクリプトファイルを管理 CLI に渡してスクリプトを実行します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/SCRIPT_NAME
```

JDBC\_PING スタックが使用できるようになり、ネットワークの通信に TCP が使用されます。

### 関連情報

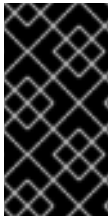
JDBC\_PING: <https://developer.jboss.org/wiki/JDBCPING>

データソースの作成: 「[JBoss EAP データソース](#)」

### 24.2.6. JGroups のネットワークインターフェースへのバインディング

デフォルトでは、JGroups は **private** ネットワークインターフェースのみへバインドし、デフォルト設定でローカルホストへ示されます。クラスタリングのトラフィックはパブリックネットワークインターフェースで公開するべきではないため、セキュリティ上の理由で JGroups は JBoss EAP の起動中に指定された **-b** 引数によって定義されたネットワークインターフェースにはバインドしません。

ネットワークインターフェースの設定方法については「[ネットワークおよびポート設定](#)」の章を参照してください。



#### 重要

セキュリティ上の理由で、JGroups はパブリックではないネットワークインターフェースのみにバインドされる必要があります。また、パフォーマンス上の理由で JGroups トラフィックのネットワークインターフェースは専用の VLAN (Virtual Local Area Network) の一部にすることが推奨されます。

### 24.2.7. クラスターのセキュア化

クラスターをセキュアに実行するには、複数の課題に対応する必要があります。

- 承認されていないノードがクラスターに参加しないようにします。これは[認証](#)を要求して対処します。
- クラスターメンバーがクラスターメンバー以外と通信しないようにします。これは[メッセージの暗号化](#)で対処します。

#### 24.2.7.1. 認証の設定

JGroups の認証は **AUTH** プロトコルによって実行されます。認証されたノードのみがクラスターに参加できるようにすることが目的です。

該当のサーバー設定ファイルに **AUTH** プロトコルと適切なプロパティ設定を追加します。 **AUTH** プロトコルは **pbcast.GMS** プロトコルの直前に設定する必要があります。

以下は、**AUTH** を異なる承認トークンと使用する例になります。

#### AUTH とシンプルトークン

```
...
<protocol type="pbcast.STABLE"/>
<auth-protocol type="AUTH">
  <plain-token>
    <shared-secret-reference clear-text="my_password"/>
  </plain-token>
</auth-protocol>
<protocol type="pbcast.GMS"/>
...
```

#### AUTH とダイジェストアルゴリズムトークン

この形式は、MD5 や SHA-2 など、どのダイジェストアルゴリズムでも使用できます。JBoss EAP 7.2 でのデフォルトのダイジェストアルゴリズムは SHA-256 で、これは JVM によるサポートに必要な最強のダイジェストアルゴリズムです。多くの JVM は、追加で SHA-512 を実装します。

```
...
<protocol type="pbcast.STABLE"/>
<auth-protocol type="AUTH">
  <digest-token algorithm="SHA-512">
    <shared-secret-reference clear-text="my_password"/>
  </digest-token>
</auth-protocol>
<protocol type="pbcast.GMS"/>
...
```

### AUTH と X509 トークン

この例は、**elytron** サブシステムで新しいキーストアを作成し、JGroups の **AUTH** 設定で参照します。

1. キーストアを作成します。

```
$ keytool -genkeypair -alias jgroups_key -keypass my_password -storepass my_password -storetype jks -keystore jgroups.keystore -keyalg RSA
```

2. 管理 CLI を使用して、キーストアを **elytron** サブシステムに追加します。

```
/subsystem=elytron/key-store=jgroups-token-store:add(type=jks,path=/path/to/jgroups.keystore,credential-reference={clear-text=my_password}, required=true)
```

3. JGroups のスタック定義で **AUTH** を設定してキーストアを使用するようにします。

```
...
<protocol type="pbcast.STABLE"/>
<auth-protocol type="AUTH">
  <cipher-token algorithm="RSA" key-alias="jgroups_key" key-store="jgroups-token-store">
    <shared-secret-reference clear-text="my_password"/>
    <key-credential-reference clear-text="my_password"/>
  </cipher-token>
</auth-protocol>
<protocol type="pbcast.GMS"/>
...
```

### 24.2.7.2. 暗号化の設定

JGroups はクラスターのメンバーが共有する秘密鍵を使用してメッセージを暗号化します。送信元は共有する秘密鍵を使用してメッセージを暗号化し、受信先は同じ秘密鍵を使用してメッセージを復号化します。**対称暗号化**は **SYM\_ENCRYPT** プロトコルを使用して設定され、ノードは共有のキーストアを使用して秘密鍵を取得します。**非対称暗号化**は **ASYM\_ENCRYPT** プロトコルを使用して設定され、ノードは **AUTH** を使用して認証された後にクラスターのコーディネーターから秘密鍵を取得します。

#### 対称暗号化の使用

**SYM\_ENCRYPT** を使用するには、各ノードの JGroups 設定で参照されるキーストアを設定する必要があります。

1. キーストアを作成します。

以下のコマンドでは、**VERSION** を適切な JGroups JAR バージョンに置き換え、**PASSWORD** をキーストアパスワードに置き換えます。

```
$ java -cp EAP_HOME/modules/system/layers/base/org/jgroups/main/jgroups-VERSION.jar
org.jgroups.demos.KeyStoreGenerator --alg AES --size 128 --storeName
defaultStore.keystore --storepass PASSWORD --alias mykey
```

これにより、JGroups 設定で参照される **defaultStore.keystore** ファイルが生成されます。

2. キーストアが生成されたら、2つの方法の1つを使用して **SYM\_PROTOCOL** で定義されます。
  - キーストアを [直接設定](#) に定義します。
  - [Elytron サブシステムを使用](#) してキーストアを参照します。



### 注記

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

キーストアを直接参照して対称暗号化を使用

1. **jgroups** サブシステムで **SYM\_ENCRYPT** プロトコルを設定します。  
該当するサーバー設定ファイルに、**SYM\_ENCRYPT** プロトコルと適切なプロパティ設定を追加します。このプロトコルは、以前作成されたキーストアを参照します。**SYM\_ENCRYPT** プロトコルは **pbcast.NAKACK2** の直前に設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:jgroups:6.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCKET"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <protocol type="SYM_ENCRYPT">
        <property name="provider">SunJCE</property>
        <property name="sym_algorithm">AES</property>
        <property name="encrypt_entire_message">>true</property>
        <property name="keystore_name">/path/to/defaultStore.keystore</property>
        <property name="store_password">PASSWORD</property>
        <property name="alias">mykey</property>
      </protocol>
      <protocol type="pbcast.NAKACK2"/>
      <protocol type="UNICAST3"/>
      <protocol type="pbcast.STABLE"/>
      <protocol type="pbcast.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG2"/>
    </stack>
  </stacks>
</subsystem>
```

Elytron と対称暗号化の使用

1. 管理 CLI を使用して、対称暗号化を使用して作成された **defaultStore.keystore** を参照するキーストアを [elytron](#) サブシステムに作成します。

```
/subsystem=elytron/key-store=jgroups-
keystore:add(path=/path/to/defaultStore.keystore,credential-reference={clear-
text=PASSWORD},type=JCEKS)
```

2. **SYM\_ENCRYPT** プロトコルと適切なプロパティ設定を **jgroups** サブシステムに追加します。以下の設定どおり、**SYM\_ENCRYPT** プロトコルは **pbcast.NAKACK2** プロトコルの直前に設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:jgroups:6.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <encrypt-protocol type="SYM_ENCRYPT" key-alias="mykey" key-store="jgroups-
keystore">
        <key-credential-reference clear-text="PASSWORD"/>
        <property name="provider">SunJCE</property>
        <property name="encrypt_entire_message">true</property>
      </encrypt-protocol>
      <protocol type="pbcast.NAKACK2"/>
      <protocol type="UNICAST3"/>
      <protocol type="pbcast.STABLE"/>
      <protocol type="pbcast.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG2"/>
    </stack>
  </stacks>
</subsystem>
```



### 注記

上記の例はクリアテキストのパスワードを使用しますが、認証情報ストアを定義して設定ファイル外部にパスワードを定義することもできます。認証情報ストアの設定に関する詳細は、『[How to Configure Server Security](#)』の「**Credential Store**」を参照してください。

### 非対称暗号化の使用

**ASYM\_ENCRYPT** を使用するには **AUTH** プロトコルを定義する必要があります。**AUTH** プロトコルを **jgroups** サブシステムで設定する手順は、「[認証の設定](#)」を参照してください。

**ASYM\_ENCRYPT** は 2 つの方法の 1 つで設定されます。

- 秘密鍵を生成し、[設定で直接](#) 参照します。
- キーストアを作成し、[Elytron サブシステムを使用して](#) 参照します。

秘密鍵を生成して非対象暗号化を使用



1. **jgroups** サブシステムで **ASYM\_ENCRYPT** プロトコルを設定します。  
 該当のサーバー設定ファイルに **ASYM\_ENCRYPT** プロトコルと適切なプロパティ設定を追加します。以下の設定どおり、**ASYM\_ENCRYPT** プロトコルは **pbcast.NAKACK2** プロトコルの直前に設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:jgroups:6.0">
  <stacks>
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <protocol type="PING"/>
      <protocol type="MERGE3"/>
      <protocol type="FD_SOCK"/>
      <protocol type="FD_ALL"/>
      <protocol type="VERIFY_SUSPECT"/>
      <protocol type="ASYM_ENCRYPT">
        <property name="encrypt_entire_message">true</property>
        <property name="sym_keylength">128</property>
        <property name="sym_algorithm">AES/ECB/PKCS5Padding</property>
        <property name="asym_keylength">512</property>
        <property name="asym_algorithm">RSA</property>
      </protocol>
      <protocol type="pbcast.NAKACK2"/>
      <protocol type="UNICAST3"/>
      <protocol type="pbcast.STABLE"/>
      <!-- Configure AUTH protocol here -->
      <protocol type="pbcast.GMS"/>
      <protocol type="UFC"/>
      <protocol type="MFC"/>
      <protocol type="FRAG2"/>
    </stack>
  </stacks>
</subsystem>
```

### Elytron と非対称暗号化の使用

1. キーペアが含まれるキーストアを作成します。以下のコマンドは、**mykey** というエイリアスでキーストアを作成します。

```
$ keytool -genkeypair -alias mykey -keyalg RSA -keysize 1024 -keystore
defaultKeystore.keystore -dname "CN=localhost" -keypass secret -storepass secret
```

2. 管理 CLI を使用して、**defaultStore.keystore** を参照するキーストアを **elytron** サブシステムに作成します。

```
/subsystem=elytron/key-store=jgroups-
keystore:add(path=/path/to/defaultStore.keystore,credential-reference={clear-
text=PASSWORD},type=JCEKS)
```

3. **ASYM\_ENCRYPT** プロトコルと適切なプロパティ設定を **jgroups** サブシステムに追加します。以下の設定どおり、**ASYM\_ENCRYPT** プロトコルは **pbcast.NAKACK2** プロトコルの直前に設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:jgroups:6.0">
  <stacks>
    <stack name="udp">
```

```

<transport type="UDP" socket-binding="jgroups-udp"/>
<protocol type="PING"/>
<protocol type="MERGE3"/>
<protocol type="FD_SOCK"/>
<protocol type="FD_ALL"/>
<protocol type="VERIFY_SUSPECT"/>
<encrypt-protocol type="ASYM_ENCRYPT" key-alias="mykey" key-store="jgroups-
keystore">
  <key-credential-reference clear-text="secret" />
  <property name="encrypt_entire_message">true</property>
</encrypt-protocol>
<protocol type="pbcast.NAKACK2"/>
<protocol type="UNICAST3"/>
<protocol type="pbcast.STABLE"/>
<!-- Configure AUTH protocol here -->
<protocol type="pbcast.GMS"/>
<protocol type="UFC"/>
<protocol type="MFC"/>
<protocol type="FRAG2"/>
</stack>
</stacks>
</subsystem>

```



### 注記

上記の例はクリアテキストのパスワードを使用しますが、認証情報ストアを定義して設定ファイル外部にパスワードを定義することもできます。認証情報ストアの設定に関する詳細は、『[How to Configure Server Security](#)』の「**Credential Store**」を参照してください。

## 24.2.8. JGroups スレッドプールの設定

**jgroups** サブシステムには **default**、**internal**、**oob**、および **timer** スレッドプールが含まれます。これらのプールはすべての JGroups スタックに設定できます。

以下の表は、各スレッドプールに設定できる属性とデフォルト値を表しています。

スレッドプール名	keepalive-time	max-threads	min-threads	queue-length
default	60000L	300	20	100
internal	60000L	4	2	100
oob	60000L	300	20	0
timer	5000L	4	2	500

以下の構文を使用して、管理 CLI で JGroups スレッドプールを設定します。

```

/subsystem=jgroups/stack=STACK_TYPE/transport=TRANSPORT_TYPE/thread-
pool=THREAD_POOL_NAME:write-attribute(name=ATTRIBUTE_NAME,
value=ATTRIBUTE_VALUE)

```

以下は、**udp** スタックの **default** スレッドプールで **max-threads** の値を **500** に設定する管理 CLI コマンドの例になります。

```
/subsystem=jgroups/stack=udp/transport=UDP/thread-pool=default:write-attribute(name="max-threads", value="500")
```

### 24.2.9. JGroups の送受信バッファの設定

#### バッファサイズ警告の解決

デフォルトでは、JGroups は特定の送受信バッファ値で設定されています。しかし、可能なバッファサイズがオペレーティングシステムによって制限され、JBoss EAP が設定されたバッファ値を使用できないことがあります。このような場合、以下と似た警告が JBoss EAP のログに記録されます。

```
WARNING [org.jgroups.protocols.UDP] (ServerService Thread Pool -- 68)
JGRP000015: the send buffer of socket DatagramSocket was set to 640KB, but the OS only
allocated 212.99KB.
This might lead to performance problems. Please set your max send buffer in the OS correctly (e.g.
net.core.wmem_max on Linux)
WARNING [org.jgroups.protocols.UDP] (ServerService Thread Pool -- 68)
JGRP000015: the receive buffer of socket DatagramSocket was set to 20MB, but the OS only
allocated 212.99KB.
This might lead to performance problems. Please set your max receive buffer in the OS correctly
(e.g. net.core.rmem_max on Linux)
```

これに対応するには、オペレーティングシステムのマニュアルでバッファサイズを増やす方法を参照してください。Red Hat Enterprise Linux システムの場合は、root ユーザーとして **/etc/sysctl.conf** を編集し、システムの再起動後も維持されるバッファサイズの最大値を設定します。例を以下に示します。

```
# Allow a 25MB UDP receive buffer for JGroups
net.core.rmem_max = 26214400
# Allow a 1MB UDP send buffer for JGroups
net.core.wmem_max = 1048576
```

**/etc/sysctl.conf** を編集後、**sysctl -p** を実行して変更を反映します。

#### JGroups バッファサイズの設定

以下のトランスポートプロパティを UDP および TCP JGroups スタックに設定すると、JBoss EAP が使用する JGroups バッファサイズを設定できます。

#### UDP スタック

- **ucast\_recv\_buf\_size**
- **ucast\_send\_buf\_size**
- **mcast\_recv\_buf\_size**
- **mcast\_send\_buf\_size**

#### TCP スタック

- **recv\_buf\_size**

- **send\_buf\_size**

JGroups バッファサイズは、管理コンソールまたは管理 CLI を使用して設定できます。

以下の構文を使用して、管理 CLI で JGroups バッファサイズプロパティを設定します。

```
/subsystem=jgroups/stack=STACK_NAME/transport=TRANSPORT/property=PROPERTY_NAME:add(value=BUFFER_SIZE)
```

以下は、**tcp** スタックで **recv\_buf\_size** プロパティを **20000000** に設定する管理 CLI コマンドの例になります。

```
/subsystem=jgroups/stack=tcp/transport=TRANSPORT/property=recv_buf_size:add(value=20000000)
```

JGroups バッファサイズは管理コンソールを使用して設定することもできます。**Configuration** タブで **JGroups** サブシステムを選択し、**表示** をクリックして **Stack** タブを選択し、該当するスタックを選びます。**Transport** をクリックし、**Properties** フィールドを編集します。

## 24.2.10. JGroups サブシステムの調整

**jgroups** サブシステムのパフォーマンスを監視および最適化するための情報は、『[パフォーマンスチューニングガイド](#)』の「**JGroups サブシステムの調整**」の項を参照してください。

## 24.2.11. JGroups トラブルシューティング

### 24.2.11.1. ノードがクラスターを形成しない

マシンで IP マルチキャストが正しくセットアップされていることを確認します。JBoss EAP には、IP マルチキャストのテストに使用できる **McastReceiverTest** と **McastSenderTest** の 2 つのテストプログラムが含まれています。

ターミナルで **McastReceiverTest** を開始します。

```
$ java -cp EAP_HOME/bin/client/jboss-client.jar org.jgroups.tests.McastReceiverTest -mcast_addr 230.11.11.11 -port 5555
```

別のターミナルウィンドウで **McastSenderTest** を開始します。

```
$ java -cp EAP_HOME/bin/client/jboss-client.jar org.jgroups.tests.McastSenderTest -mcast_addr 230.11.11.11 -port 5555
```

特定のネットワークインターフェースカード (NIC) をバインドする場合は、**-bind\_addr YOUR\_BIND\_ADDRESS** を使用します。**YOUR\_BIND\_ADDRESS** はバインドする NIC の IP アドレスに置き換えます。送信側と受信側の両方にこのパラメーターを使用します。

**McastSenderTest** ターミナルウィンドウで入力すると **McastReceiverTest** ウィンドウに出力が表示されます。表示されない場合は以下の手順に従います。

- 送信側のコマンドに **-ttl VALUE** を追加して、マルチキャストパケットの TTL (time-to-live) を増やします。このテストプログラムによって使用されるデフォルトの値は **32** で、**VALUE** は **255** 以下である必要があります。

- マシンに複数のインターフェースがある場合は、適切なインターフェースを使用していることを確認します。
- システム管理者に連絡し、マルチキャストが選択したインターフェースで動作することを確認します。

クラスターの各マシンでマルチキャストが適切に動作したら、送信側と受信側を別々のマシンに配置し、テストを繰り返します。

### 24.2.11.2. 障害検出での不明なハートビートの原因

ハートビートの確認が一定時間 (**T**) 受信されないと、障害検出 (FD) によってクラスターメンバーが原因として疑われることがあります。T は **timeout** および **max\_tries** によって定義されます。

たとえば、ノード A、B、C、および D のクラスターがあり、A が B、B が C、C が D、D が A を ping する場合、以下のいずれかの理由で C が疑われます。

- B または C が CPU の使用率が 100% の状態で **T** 秒よりも長く稼働している場合。この場合、C がハートビート確認を B に送信しても CPU の使用率が 100% であるため B が確認を処理できないことがあります。
- B または C がガベージコレクションを実行している場合、上記と同じ結果になります。
- 上記 2 件の組み合わせ。
- ネットワークによるパケットの損失が発生する場合。通常、ネットワークに大量のトラフィックがあり、スイッチがパケットを破棄すると発生します (通常は最初にブロードキャスト、次に IP マルチキャスト、そして最後に TCP パケットが破棄されます)。
- B または C がコールバックを処理する場合。C が処理に **T+1** 秒かかるリモートメソッド呼び出しをチャンネル上で受信した場合、C はハートビートを含む他のメッセージを処理できません。そのため、B はハートビート確認を受信せず、C が疑われます。

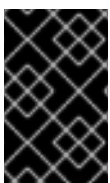
## 24.3. INFINISPAN

### 24.3.1. Infinispan

Infinispan は Java データグリッドプラットフォームで、キャッシュされたデータの管理に [JSR-107](#) 準拠のキャッシュインターフェースを提供します。Infinispan の機能や設定オプションの詳細は [Infinispan のドキュメント](#) を参照してください。

**infinispan** サブシステムは JBoss EAP のキャッシュサポートを提供します。名前付きのキャッシュコンテナーやキャッシュのランタイムメトリクスを設定および表示できます。

高可用性の機能を提供する設定を使用する場合 (管理対象ドメインでは **ha** や **full-ha** プロファイル、スタンドアロンサーバーは **standalone-ha.xml** や **standalone-full-ha.xml** 設定ファイル)、**infinispan** サブシステムはキャッシング、状態のレプリケーション、および状態分散サポートを提供します。高可用性でない設定では、**infinispan** サブシステムはローカルのキャッシュサポートを提供します。



#### 重要

Infinispan は JBoss EAP のプライベートモジュールとして含まれ、JBoss EAP のキャッシュ機能を提供します。アプリケーションによる Infinispan の直接使用はサポートされません。

### 24.3.2. キャッシュコンテナ

キャッシュコンテナはサブシステムによって使用されるキャッシュのリポジトリです。各キャッシュコンテナは、使用されるデフォルトのキャッシュを定義します。

JBoss EAP 7 は以下のデフォルト Infinispan キャッシュコンテナを定義します。

- **server** (シングルトンキャッシング)
- **web** (Web セッションクラスタリング)
- **ejb** (ステートフルセッション Bean クラスタリング)
- **hibernate** (エンティティキャッシング)

#### 例: Infinispan のデフォルト設定

```
<subsystem xmlns="urn:jboss:domain:infinispan:7.0">
  <cache-container name="server" aliases="singleton cluster" default-cache="default"
  module="org.wildfly.clustering.server">
    <transport lock-timeout="60000"/>
    <replicated-cache name="default">
      <transaction mode="BATCH"/>
    </replicated-cache>
  </cache-container>
  <cache-container name="web" default-cache="dist" module="org.wildfly.clustering.web.infinispan">
    <transport lock-timeout="60000"/>
    <distributed-cache name="dist">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <file-store/>
    </distributed-cache>
  </cache-container>
  <cache-container name="ejb" aliases="sfsb" default-cache="dist"
  module="org.wildfly.clustering.ejb.infinispan">
    <transport lock-timeout="60000"/>
    <distributed-cache name="dist">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <file-store/>
    </distributed-cache>
  </cache-container>
  <cache-container name="hibernate" default-cache="local-query" module="org.hibernate.infinispan">
    <transport lock-timeout="60000"/>
    <local-cache name="local-query">
      <eviction size="1000"/>
      <expiration max-idle="100000"/>
    </local-cache>
    <invalidation-cache name="entity">
      <transaction mode="NON_XA"/>
      <eviction size="1000"/>
      <expiration max-idle="100000"/>
    </invalidation-cache>
    <replicated-cache name="timestamps" mode="ASYNC"/>
  </cache-container>
</subsystem>
```

各キャッシュコンテナで定義されたデフォルトのキャッシュに注目してください。この例では、**web** キャッシュコンテナは **dist** 分散キャッシュをデフォルトとして定義します。そのため、Web セッションのクラスタリングでは **dist** キャッシュが使用されます。

デフォルトキャッシュの変更やキャッシュの追加に関する詳細は、「[キャッシュコンテナの設定](#)」を参照してください。



### 重要

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

#### 24.3.2.1. キャッシュコンテナの設定

キャッシュコンテナやキャッシュ属性は、管理コンソールまたは管理 CLI を使用して設定できます。



### 警告

設定の他のコンポーネントが参照する可能性があるため、キャッシュまたはキャッシュコンテナの名前を変更しないでください。

#### 管理コンソールを使用したキャッシュの設定

管理コンソールの **Configuration** タブで **Infinispan** サブシステムを選択するとキャッシュやキャッシュコンテナを設定できます。管理対象ドメインでは設定するプロファイルを選択してください。

- キャッシュコンテナを追加します。  
Cache Container の横にある追加 (+) ボタンをクリックし、Cache Container を追加を選択して新しいキャッシュコンテナの設定を入力します。
- キャッシュコンテナ設定を更新します。  
該当するキャッシュコンテナを選択し、表示 を選択します。必要に応じてキャッシュコンテナの設定を行います。
- キャッシュコンテナトランスポート設定を更新します。  
該当するキャッシュコンテナを選択し、表示 を選択します。Transport タブで、必要に応じてキャッシュコンテナトランスポートを設定します。
- キャッシュを設定します。  
該当するキャッシュコンテナを選択し、表示 を選択します。キャッシュタブ (Replicated Cache など) でキャッシュを追加、更新、および削除できます。

#### 管理 CLI を使用したキャッシュの設定

管理 CLI を使用してキャッシュおよびキャッシュコンテナを設定できます。管理対象ドメインではコマンドの前に **/profile=PROFILE\_NAME** を追加して更新するプロファイルを指定する必要があります。

- キャッシュコンテナを追加します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER:add
```

- レプリケートされたキャッシュを追加します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/replicated-cache=CACHE:add(mode=MODE)
```

- キャッシュコンテナのデフォルトキャッシュを設定します。

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

- レプリケートされたキャッシュのバッチ処理を設定します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/replicated-cache=CACHE/component=transaction:write-attribute(name=mode,value=BATCH)
```

以下の例は、**concurrent** 分散キャッシュを **web** キャッシュコンテナに追加する方法を表しています。このキャッシュ設定はデフォルトキャッシュのロック制約を緩和し、複数の同時リクエストが同じ web セッションに同時にアクセスできるようにします。これは、ロックのない読み取りを許可し、排他的ロックをより頻繁により短期間で取得することで実現されます。

以下の CLI コマンドを使用して **concurrent** 分散キャッシュを **web** キャッシュコンテナに追加し、デフォルトのキャッシュにします。

```
batch
/subsystem=infinispan/cache-container=web/distributed-cache=concurrent:add
/subsystem=infinispan/cache-container=web:write-attribute(name=default-cache,value=concurrent)
/subsystem=infinispan/cache-container=web/distributed-cache=concurrent/store=file:add
run-batch
```

これにより、サーバーが以下のように設定されます。

```
<cache-container name="web" default-cache="concurrent"
module="org.wildfly.clustering.web.infinispan">
...
<distributed-cache name="concurrent">
  <file-store/>
</distributed-cache>
</cache-container>
```

デフォルト EJB キャッシュコンテナの変更

以下のようにキャッシュコンテナを **ejb3** サブシステムで使用できます。

- EJB セッション bean のパッシベーションをサポートするには、**infinispan** サブシステムに定義された **ejb** キャッシュコンテナを使用してセッションを保存できます。
- サーバー上でクラスター化されたデプロイメントに接続するリモート EJB クライアントの場合、クラスタートポロジの情報をこれらのクライアントに提供して、これらのクライアントが対話するノードに障害が発生したときにクラスタの別のノードにフェイルオーバーできるようにする必要があります。

パッシベーションやトポロジ情報の提供をサポートするデフォルトのキャッシュコンテナ **ejb** を変更する場合や、その名前を変更する場合、以下の例のように **cache-container** 属性を **passivation-stores** 要素に追加し、**cluster** 属性を **remote** 要素に追加する必要があります。独自に使用するために新しいキャッシュを追加する場合は、このような変更を加える必要はありません。



```

<subsystem xmlns="urn:jboss:domain:ejb3:5.0">
  <passivation-stores>
    <passivation-store name="infinispan" cache-container="ejb-cltest" max-size="10000"/>
  </passivation-stores>

  <remote cluster="ejb-cltest" connector-ref="http-remoting-connector" thread-pool-name="default"/>
</subsystem>

<subsystem xmlns="urn:jboss:domain:infinispan:7.0">
  ...
  <cache-container name="ejb-cltest" aliases="sfsb" default-cache="dist"
module="org.wildfly.clustering.ejb.infinispan">
</subsystem>

```

**Hibernate** キャッシュコンテナのエビクション機能

**hibernate** キャッシュコンテナのエビクション機能は、メモリーからキャッシュエントリーを削除します。この機能は、サブシステムのメモリー負荷を軽減するのに役立ちます。

**size** 属性は、キャッシュエントリーのエビクションの開始前に保存されるキャッシュエントリーの最大数を設定します。

#### 例：エビクション機能

```

<cache-container name="hibernate" default-cache="local-query" module="org.hibernate.infinispan">
  <transport lock-timeout="60000"/>
  <local-cache name="local-query">
    <eviction size="1000"/>
    <expiration max-idle="100000"/>

```

エビクションはメモリー内でのみ実行されることに注意してください。キャッシュストアは、情報の永続的な損失を防ぐためにエビクトされたキャッシュエントリーを保持します。エビクション機能についての詳細は、『Infinispan ユーザーガイド』の「[エビクションおよびデータコンテナ](#)」セクションを参照してください。

### 24.3.3. クラスタリングモード

JBoss EAP で Infinispan を使用すると、2つの方法でクラスタリングを設定できます。ご使用のアプリケーションに最も適した方法は要件によって異なります。各モードでは可用性、一貫性、信頼性、およびスケーラビリティのトレードオフが発生します。クラスタリングモードを選択する前に、ネットワークで最も重要な点を特定し、これらの要件のバランスを取ることが必要となります。

#### キャッシュモード

##### Replication (レプリケーション)

Replication (レプリケーション) モードはクラスターの新しいインスタンスを自動的に検出し、追加します。これらのインスタンスに加えられた変更は、クラスター上のすべてのノードにレプリケートされます。ネットワークでレプリケートされる情報量が多いため、通常 Replication モードは小型のクラスターでの使用に最も適しています。UDP マルチキャストを使用するよう Infinispan を設定すると、ネットワークトラフィックの輻輳をある程度軽減できます。

##### Distribution (分散)

Distribution (分散) モードでは、Infinispan はクラスターを線形にスケールできます。Distribution モードは一貫性のあるハッシュアルゴリズムを使用して、クラスター内で新しいノードを配置する場所を決定します。保持する情報のコピー数 (または所有者数) は設定可能です。保持するコピー数、データの永続性、およびパフォーマンスにはトレードオフが生じます。保持するコピー数が多

いほどパフォーマンスへの影響が大きくなりますが、サーバーの障害時にデータを損失する可能性は低くなります。ハッシュアルゴリズムは、メタデータのマルチキャストや保存を行わずにエントリーを配置し、ネットワークトラフィックを軽減します。

クラスターサイズが 6-8 ノードを超える場合は Distribution モードをキャッシュストラテジーとして使用することを考慮してください。Distribution モードでは、データはクラスター内のすべてのノードではなくノードのサブセットのみに分散されます。

### Scattered (散在)

Scattered (散在) モードは、一貫したハッシュアルゴリズムを使用して所有者を判断する Distribution モードと似ています。しかし、所有者は 2 名のメンバーに限定され、オリジネーター (指定のセッションのリクエストを受信するノード) は常にロックを調整する所有者やキャッシュエントリーのアップデートを想定します。2 名の所有者を持つ分散キャッシュが頻繁に 2 つの RPC 呼び出しを使用できる場合、Scattered モードで使用されるキャッシュ書き込みアルゴリズムは、書き込み操作の結果が単一の RPC 呼び出しになることを保証します。これは、ロードバランサーのフェイルオーバーがプライマリー以外の所有者やバックアップノードにトラフィックを向ける可能性があるため、分散 Web セッションに便利です。これにより、競合の可能性を下げ、クラスタポートの変更後にパフォーマンスを向上することが可能になります。

Scattered モードは、トランザクションや L1 キャッシュをサポートしません。しかし、一貫したハッシュによると所有者ではない場合でも、指定エントリーのキャッシュの書き込みを開始するノードは一定の期間は継続してそのエントリーの読み取りを継続する、バイアス読み取り (Biased read) をサポートします。バイアス読み取りと L1 キャッシュの設定属性は異なりますが、L1 キャッシュと同様の効果があります。

### 同期および非同期のレプリケーション

レプリケーションは同期または非同期モードで実行でき、選択するモードは要件やアプリケーションモードによって異なります。

#### 同期のレプリケーション

同期のレプリケーションでは、レプリケーションプロセスはユーザーのリクエストを処理するのと同じスレッドで実行されます。セッションレプリケーションは、完了した応答がクライアントに返送された後に開始され、スレッドはレプリケーションの完了後のみに解放されます。同期レプリケーションはクラスターの各ノードからの応答を必要とするため、ネットワークトラフィックに影響を与えます。ただし、クラスターのすべてのノードへ確実に変更が加えられる利点があります。

#### 非同期のレプリケーション

非同期のレプリケーションでは、Infinispan はスレドプールを使用してバックグラウンドでレプリケーションを実行します。送信元はクラスターの他のノードからの返答を待ちません。しかし、同じセッションを読み取るキャッシュはその前のレプリケーションが完了するまでブロックされるため、陳腐データは読み取られません。レプリケーションは時間ベースまたはキューのサイズによって引き起こされます。失敗したレプリケーションはログに書き込まれ、リアルタイムで通知されません。

### 24.3.3.1. キャッシュモードの設定

管理 CLI を使用してデフォルトキャッシュを変更できます。



#### 注記

ここでは、デフォルトが Distribution モードである Web セッションキャッシュの設定に固有する手順を説明します。手順と管理 CLI コマンドは、他のキャッシュコンテナ向けに簡単に調整できます。

### Replication キャッシュモードへの変更

Web セッションキャッシュのデフォルトの JBoss EAP 7 設定には、レプリケートされたキャッシュ **repl** が含まれていません。最初にこのキャッシュを追加する必要があります。



### 注記

以下はスタンドアロンサーバーの管理 CLI コマンドになります。管理対象ドメインで実行している場合は、**/subsystem=infinispan** コマンドの前に **/profile=PROFILE\_NAME** を追加し、更新するプロファイルを指定する必要があります。

1. レプリケーションキャッシュ **repl** を追加し、デフォルトのキャッシュとして設定します。

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

2. サーバーをリロードします。

```
reload
```

### Distribution キャッシュモードへの変更

Web セッションキャッシュのデフォルトの JBoss EAP 7 設定にはすでに **dist** 分散キャッシュが含まれています。



### 注記

以下はスタンドアロンサーバーの管理 CLI コマンドになります。管理対象ドメインで実行している場合は、**/subsystem=infinispan** コマンドの前に **/profile=PROFILE\_NAME** を追加し、更新するプロファイルを指定する必要があります。

1. デフォルトのキャッシュを **dist** 分散キャッシュに変更します。

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

2. 分散キャッシュの所有者数を設定します。以下のコマンドは所有者の数を **5** に設定します。デフォルトは **2** です。

```
/subsystem=infinispan/cache-container=web/distributed-cache=dist:write-attribute(name=owners,value=5)
```

3. サーバーをリロードします。

```
reload
```

### Scattered キャッシュモードの変更

web セッションキャッシュのデフォルトの JBoss EAP 設定には **scattered-cache** が含まれていません。以下の例は、scattered キャッシュを追加し、デフォルトのキャッシュとして設定する管理 CLI コマンドを示しています。



## 注記

以下は、HA プロファイルを使用するスタンドアロンサーバー向けの管理 CLI コマンドになります。管理対象ドメインで実行している場合は、`/subsystem=infinispan` コマンドの前に `/profile=PROFILE_NAME` を追加し、更新するプロファイルを指定する必要があります。

1. 読み取りのバイアスライフスパンがデフォルトの web セッションタイムアウト値 (30 分) と同じになるよう、scattered キャッシュを作成します。

```
/subsystem=infinispan/cache-container=web/scattered-cache=scattered:add(bias-lifespan=1800000)
```

2. **scattered** をデフォルトキャッシュとして設定します。

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

これにより、サーバーが以下のように設定されます。

```
<cache-container name="web" default-cache="scattered"
module="org.wildfly.clustering.web.infinispan">
...
<scattered-cache name="scattered" bias-lifespan="1800000"/>
...
</cache-container>
```

### 24.3.3.2. キャッシュストラテジーのパフォーマンス

**SYNC** キャッシュストラテジーを使用すると、レプリケーションが完了するまでリクエストが完了しないため、レプリケーションのコストを簡単に評価でき、直接応答時間に影響します。

**ASYNC** キャッシュストラテジーの応答時間は **SYNC** キャッシュストラテジーよりも短いと思われがちですが、一定の状況下でのみ短くなります。**ASYNC** キャッシュストラテジーの評価はより困難ですが、リクエスト間の時間がキャッシュ操作を完了できるほど長い場合はパフォーマンスが **SYNC** よりも良くなります。これは、レプリケーションのコストが応答時間に即影響しないためです。

同じセッションのリクエストの作成が早すぎると、先行リクエストからのレプリケーションが完了するまで待機しなければならないため、先行リクエストのレプリケーションコストが後続リクエストの前に移動します。応答の受信直後に後続リクエストが送信され、リクエストが急速に発生する場合、**ASYNC** キャッシュストラテジーのパフォーマンスは **SYNC** よりも劣化します。そのため、同じセッションのリクエストの間には、**SYNC** キャッシュストラテジーのパフォーマンスが **ASYNC** キャッシュストラテジーよりも良くなる期間のしきい値があります。実際の使用状態では、通常同じセッションのリクエストが立て続けに受信されることはありませんが、一般的にリクエストの間に数秒程度の時間が存在します。その代わりに、通常、要求間に数秒以上の時間が経過します。この場合、**ASYNC** キャッシュストラテジーが適切なデフォルトで、応答時間が早くなります。

### 24.3.4. 状態遷移

状態遷移は、基本的なデータグリッドとクラスター化されたキャッシュ機能の両方です。状態遷移がない状態では、ノードがクラスターに追加されたり、クラスターから削除されるとデータが失われます。

状態遷移は、キャッシュメンバーシップの変更に応じて、キャッシュの内部状態を調整します。この変更は、ノードがクラスターに参加または退出するとき、2つ以上のクラスターパーティションがマージするとき、またはこれらのイベントの組み合わせが発生した後に自動的に行われます。新しいキャッシュは、以下のキャッシュのモードを基にして、最大量の状態を受け取る必要があるため、新たに開始されたキャッシュの最初の状態遷移は最も多くのリソースが必要となります。

**timeout** 属性を使用すると、新たに開始されたキャッシュが状態を受け取る待ち時間を制御することができます。**timeout** 属性が正の値である場合、キャッシュはすべての初期状態を受け取るまで待機した後、リクエストに対応できるようになります。状態遷移が指定時間内 (デフォルトは **240000** ミリ秒) に完了しなかった場合、キャッシュによってエラーが発生し、開始がキャンセルされます。**timeout** を **0** に設定するとキャッシュは即座に利用可能になり、バックグラウンド操作中に最初の状態を受け取ります。最初の状態遷移が完了するまで、キャッシュが受け取っていないキャッシュエントリーのリクエストは、リモートノードから取得する必要があります。

**timeout** 属性を **0** に設定するには、以下のコマンドを実行します。

```
/subsystem=infinispan/cache-container=server/CACHE_TYPE=CACHE/component=state-transfer:write-attribute(name=timeout,value=0)
```

状態遷移の挙動はキャッシュのモードによって決まります。

- Replicated (レプリケート) モードでは、キャッシュに参加する新規ノードは既存ノードからキャッシュの状態をすべて受け取ります。ノードがクラスターから退出すると、状態遷移はありません。
- Distribution (分散) モードでは、新しいノードは既存のノードから状態の一部のみを受け取り、既存のノードは一貫したハッシュの決定どおりに、各キーの **owners** コピーを維持するために状態の一部を削除します。ノードがクラスターから退出する場合、分散キャッシュはそのノードに保存されたキーの追加コピーを作成し、各キーの所有者が存続するようにする必要デフォルトがあります。
- Invalidation (インバリデーション) モードでは、最初の状態遷移は Replicated モードと似ていますが、ノードが同じ状態でいられる保証がないことが唯一の違いです。ノードがクラスターから退出すると、状態遷移はありません。

デフォルトでは、状態遷移はインメモリーおよび永続状態の両方を遷移しますが、設定で無効にすることもできます。状態遷移が無効になっている場合、**ClusterLoader** を設定しないと、データをキャッシュにロードせずにノードがキーの所有者またはバックアップ所有者になります。さらに、Distribution モードで状態遷移が無効になっていると、キャッシュでキーのコピーが **owners** コピーよりも少なくなることがあります。

### 24.3.5. Infinispan スレッドプールの設定

**infinispan** サブシステムには **async-operations**、**expiration**、**listener**、**persistence**、**remote-command**、**state-transfer**、および **transport** スレッドプールが含まれます。これらのプールはすべての **Infinispan** キャッシュコンテナに設定できます。

以下の表は、**infinispan** サブシステムの各スレッドプールに設定できる属性とデフォルト値を表しています。

スレッドプール名	keepalive-time	max-threads	min-threads	queue-length
async-operations	60000L	25	25	1000
expiration	60000L	1	該当なし	該当なし
listener	60000L	1	1	100000
persistence	60000L	4	1	0
remote-command	60000L	200	1	0
state-transfer	60000L	60	1	0
transport	60000L	25	25	100000

以下の構文を使用して、管理 CLI で Infinispan スレッドプールを設定します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER_NAME/thread-pool=THREAD_POOL_NAME:write-attribute(name=ATTRIBUTE_NAME, value=ATTRIBUTE_VALUE)
```

以下は、**server** キャッシュコンテナの **persistence** スレッドプールで **max-threads** の値を **10** に設定する管理 CLI コマンドの例になります。

```
/subsystem=infinispan/cache-container=server/thread-pool=persistence:write-attribute(name="max-threads", value="10")
```

### 24.3.6. Infinispan の統計

監視目的で、Infinispan キャッシュやキャッシュコンテナに関する実行時統計を有効にできます。パフォーマンス上の理由で、統計の収集はデフォルトでは無効になっています。

統計収集は、各キャッシュコンテナ、キャッシュ、または両方に対して有効にできます。各キャッシュの統計オプションはキャッシュコンテナのオプションをオーバーライドします。キャッシュコンテナの統計収集を無効または有効にすると、独自の設定が明示的に指定されている場合以外はそのコンテナのすべてのキャッシュが設定を継承します。

#### 24.3.6.1. Infinispan 統計の有効化



#### 警告

Infinispan の統計を有効にすると、**infinispan** サブシステムのパフォーマンスに影響する可能性があります。統計は必要な場合のみ有効にしてください。

管理コンソールまたは管理 CLI を使用すると Infinispan の統計収集を有効または無効にできます。管理コンソールでは、**Configuration** タブで **Infinispan** サブシステム選択してキャッシュまたはキャッシュコンテナを選択し、**Statistics Enabled** 属性を編集します。管理 CLI を使用する場合は以下のコマンドを実行して統計を有効にします。

キャッシュコンテナの統計収集を有効にします。サーバーをリロードする必要があります。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER:write-attribute(name=statistics-enabled,value=true)
```

キャッシュの統計収集を有効にします。サーバーをリロードする必要があります。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/CACHE_TYPE=CACHE:write-attribute(name=statistics-enabled,value=true)
```

### 注記

以下のコマンドを使用すると、キャッシュの **statistics-enabled** 属性の定義が解除され、キャッシュコンテナの **statistics-enabled** 属性の設定を継承します。

```
/subsystem=infinispan/cache-container=CACHE_CONTAINER/CACHE_TYPE=CACHE:undefine-attribute(name=statistics-enabled)
```

## 24.3.7. Infinispan パーティションの処理

**Infinispan** クラスタは、データが保存される複数のノードで構築されます。複数のノードに障害が発生した場合のデータの損失を防ぐため、Infinispan は複数のノードで同じデータをコピーします。このレベルのデータ冗長性は **owners** 属性を使用して設定されます。設定されたノードの数未満のノードが同時にクラッシュしても、Infinispan はデータのコピーを利用できます。

しかし、クラスタから大量のノードが消滅すると最悪の事態を招く可能性があります。

### スプリットブレイン

スプリットブレインはクラスタを独立して動作する 2 つ以上のパーティションまたはサブクラスタに分割します。このような場合、異なるパーティションから読み書きする複数のクライアントが同じキャッシュエントリの異なるバージョンを見ることになり、多くのアプリケーションにとって問題となります。

### 注記

スプリットブレインの発生を軽減する方法には、冗長化ネットワークや **IP ボンディング** などがあります。しかし、これらの方法は問題発生のリードタイムを削減するだけです。

### 複数ノードの連続クラッシュ

複数のノード (所有者の数) が連続してクラッシュし、Infinispan がクラッシュ間の状態を適切に調整する時間がない場合、結果として部分的なデータの損失が発生します。

スプリットブレインや複数ノードの連続クラッシュが原因で、不適切なデータがユーザーに返されないようにすることが大切です。

### 24.3.7.1. スプリットブレイン

スプリットブレインが発生した場合、各ネットワークパーティションが独自の JGroups ビューをインストールし、他のパーティションからノードを削除します。パーティションはお互いを認識しないため、クラスターが 2 つ以上のパーティションに分割されたかどうかを直接判断することはできません。そのため、明示的な脱退メッセージを送信せずに、1 つ以上のノードが JGroups クラスターから消滅した場合にクラスターが分割されたと判断します。

パーティション処理が無効の場合、各パーティションは継続して独立したクラスターとして機能します。各パーティションはデータの一部のみを認識できる可能性があり、競合する更新をキャッシュに書き込む可能性があります。

パーティション処理が有効の場合、スプリットを検出したときに各パーティションは即座にリバランスを行わず、degrade モードにするかどうかを最初にチェックします。

- 1 つ以上のセグメントがすべての所有者を失った場合 (最後に行ったりリバランスが完了した後に指定した所有者の数以上が脱退した場合)、パーティションは degrade モードになります。
- 最新の安定したトポロジーでパーティションに単純多数のノード ( $\text{floor}(\text{numNodes}/2) + 1$ ) が含まれない場合も、パーティションは degrade モードになります。
- その他の場合は、パーティションは通常通り動作し、リバランスを開始します。

安定したトポロジーは、リバランス操作が終了するたびに更新され、コーディネーターによって他のリバランスが必要ないと判断された場合に毎回更新されます。これらのルールは、1 つのパーティションが available モードを維持し、他のパーティションが degraded モードになるようにします。

パーティションが degraded モードの場合、完全に所有されたキーへのアクセスのみを許可します。

- このパーティション内のノード上のコピーをすべて持つエントリーのリクエスト (読み取りおよび書き込み) は許可されます。
- 消滅したノードによって完全所有または一部所有されたエントリーのリクエストは **AvailabilityException** によって拒否されます。

これにより、パーティションが同じキーに異なる値を書き込めないようにし (キャッシュの一貫性を保つ)、さらにパーティションが他のパーティションで更新されたキーを読み取れないようにします (陳腐データをなくす)。



#### 注記

2 つのパーティションは分離して開始できます。これらのパーティションはマージされなければ不整合なデータを読み書きできます。将来的に、この状況に対処できるカスタムの可用性ストラテジーが許可される可能性があります (例: 特定のノードがクラスターの一部であるかを確認、外部のマシンにアクセスできるかどうかを確認など)。

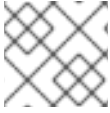
### 24.3.7.2. パーティション処理の設定

現在、パーティションの処理はデフォルトで無効になっています。パーティションの処理を有効にするには、以下の管理 CLI コマンドを使用します。

```
/subsystem=infinispan/cache-container=web/distributed-cache=dist/component=partition-handling:write-attribute(name=enabled, value=true)
```

### 24.3.8. HTTP セッションの Red Hat Data Grid への外部化





## 注記

この機能を使用するには Red Hat Data Grid のサブスクリプションが必要です。

Red Hat Data Grid は、HTTP セッションなどの JBoss EAP のアプリケーション固有データの外部キャッシュコンテナとして使用できます。これにより、アプリケーションとは独立したデータレイヤーのスケールが可能になり、さまざまなドメインに存在する可能性がある異なる JBoss EAP クラスタが同じ Red Hat Data Grid クラスタからデータにアクセスできるようになります。また、他のアプリケーションは Red Hat Data Grid によって提供されたキャッシュと対話できます。

以下の例は、HTTP セッションを外部化する方法を説明しています。これは、JBoss EAP のスタンドアロンインスタンスと管理対象ドメインの両方に適用されますが、管理対象ドメインでは各サーバーグループに一意のリモートキャッシュが設定されている必要があります。複数のサーバーグループは同じ Red Hat Data Grid クラスタを使用できますが、各リモートキャッシュは JBoss EAP サーバーグループに固有となります。

### 1. **infinispan** サブシステムで **remote-cache-container** を作成します。

- a. **socket-binding** を定義します。クラスタの各リモート Red Hat Data Grid インスタンスの必要に応じて、コマンドを繰り返し実行します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=SOCKET_BINDING:add(host=HOSTNAME,port=PORT)
```

- b. 新規作成されたソケットバインディングを参照する **remote-cache-container** を定義します。

```
batch
/subsystem=infinispan/remote-cache-container=CACHE_CONTAINER:add(default-remote-cluster=data-grid-cluster)
/subsystem=infinispan/remote-cache-container=CACHE_CONTAINER/remote-cluster=data-grid-cluster:add(socket-bindings=[SOCKET_BINDING,SOCKET_BINDING_2,...])
run-batch
```

- c. 変更を反映するためにサーバーをリロードします。

```
reload
```

2. HotRod ストアを設定します。HotRod ストアは、JBoss EAP サーバーによって作成された各キャッシュに対して専用のリモートキャッシュを1つ使用します。通常、以下の CLI スクリプトのように、JBoss EAP サーバーで1つのインバリデーションキャッシュが使用されます。



## 注記

Red Hat JDG サーバーでリモートキャッシュを手作業で設定する必要があります。これらのキャッシュに推奨される設定は、楽観的ロックを持つトランザクション分散モードです。キャッシュ名はデプロイメントファイル名に対応する必要があります (例: **test.war**)。

リモートキャッシュコンテナが設定されたら、**hotrod** ストアを設定して既存のストアを置き換えることができます。以下の CLI スクリプトは、インバリデーションキャッシュとともにセッションをオフロードする典型的なユースケースを示しています。

```

batch
/subsystem=infinispan/cache-container=web/invalidation-cache=CACHE_NAME:add()
/subsystem=infinispan/cache-container=web/invalidation-
cache=CACHE_NAME/store=hotrod:add(remote-cache-
container=CACHE_CONTAINER,fetch-
state=false,purge=false,passivation=false,shared=true)
/subsystem=infinispan/cache-container=web/invalidation-
cache=CACHE_NAME/component=transaction:add(mode=BATCH)
/subsystem=infinispan/cache-container=web/invalidation-
cache=CACHE_NAME/component=locking:add(isolation=REPEATABLE_READ)
/subsystem=infinispan/cache-container=web:write-attribute(name=default-
cache,value=CACHE_NAME)
run-batch

```

このスクリプトは新しいインバリデーションキャッシュを設定します。作成後、セッションデータはパフォーマンスの目的でキャッシュに維持され、復元の目的でストアに書き込まれます。

**@Resource** アノテーションを使用すると、HotRod クライアントを直接 Java EE アプリケーションにインジェクトすることができます。**@Resource** アノテーションは **hotrod-client.properties** ファイルで、クラスパスの設定プロパティを検索します。

```

@Resource(lookup = "java:jboss/infinispan/remote-container/web-sessions")
private org.infinispan.client.hotrod.RemoteCacheContainer client;

```

#### 例: hotrod-client.properties ファイル

```

infinispan.client.hotrod.transport_factory =
org.infinispan.client.hotrod.impl.transport.tcp.TcpTransportFactory
infinispan.client.hotrod.server_list = 127.0.0.1:11222
infinispan.client.hotrod.marshaller =
org.infinispan.commons.marshall.jboss.GenericJBossMarshaller
infinispan.client.hotrod.async_executor_factory =
org.infinispan.client.hotrod.impl.async.DefaultAsyncExecutorFactory
infinispan.client.hotrod.default_executor_factory.pool_size = 1
infinispan.client.hotrod.default_executor_factory.queue_size = 10000
infinispan.client.hotrod.hash_function_impl.1 =
org.infinispan.client.hotrod.impl.consistenthash.ConsistentHashV1
infinispan.client.hotrod.tcp_no_delay = true
infinispan.client.hotrod.ping_on_startup = true
infinispan.client.hotrod.request_balancing_strategy =
org.infinispan.client.hotrod.impl.transport.tcp.RoundRobinBalancingStrategy
infinispan.client.hotrod.key_size_estimate = 64
infinispan.client.hotrod.value_size_estimate = 512
infinispan.client.hotrod.force_return_values = false

## below is connection pooling config

maxActive=-1
maxTotal = -1
maxIdle = -1
whenExhaustedAction = 1
timeBetweenEvictionRunsMillis=120000

```

```
minEvictableIdleTimeMillis=300000
testWhileIdle = true
minIdle = 1
```

#### リモートキャッシュコンテナのセキュア化

SSL を使用してリモート Red Hat Data Grid インスタンスへの通信をセキュアにすることが可能です。これを行うには、JBoss EAP インスタンスで **remote-cache-container** を設定し、Red Hat Data Grid インスタンスで hotrod コネクタを調整してアクティブなセキュリティーレルムを使用するようにします。

1. JBoss EAP で **client-ssl-context** を作成します。他の elytron コンポーネントの生成など、**client-ssl-context** 作成のその他の詳細は、JBoss EAP 『[How to Configure Server Security](#)』の「[Using a client-ssl-context](#)」を参照してください。

```
/subsystem=elytron/client-ssl-context=CLIENT_SSL_CONTEXT:add(key-
manager=KEY_MANAGER,trust-manager=TRUST_MANAGER)
```

2. クライアント SSL コンテキストを使用するよう、リモートキャッシュコンテナを設定します。

```
/subsystem=infinispan/remote-cache-
container=CACHE_CONTAINER/component=security:write-attribute(name=ssl-
context,value=CLIENT_SSL_CONTEXT)
```

3. リモート Red Hat Data Grid インスタンスをセキュアにします。各インスタンスの必要に応じて手順を繰り返します。
  - a. **client-ssl-context** で使用されるキーストアをリモート Red Hat Data Grid インスタンスにコピーします。
  - b. **ApplicationRealm** を設定して、このキーストアを使用するようにします。

```
/core-service=management/security-realm=ApplicationRealm/server-
identity=ssl:add(keystore-path="KEYSTORE_NAME",keystore-relative-
to="jboss.server.config.dir",keystore-password="KEYSTORE_PASSWORD")
```

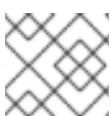
- c. hotrod コネクタを調整して、このセキュリティーレルムを示すようにします。

```
/subsystem=datagrid-infinispan-endpoint/hotrod-connector=hotrod-
connector/encryption=ENCRYPTION:add(require-ssl-client-auth=false,security-
realm="ApplicationRealm")
```

- d. リモート Red Hat Data Grid インスタンスをリロードします。

```
reload
```

### 24.3.9. リモートストアを使用した HTTP セッションの Red Hat Data Grid への外部化



#### 注記

この機能を使用するには Red Hat Data Grid のサブスクリプションが必要です。

この手順は、セッションを外部化する旧式の方法になります。JBoss EAP 7.2 には、**elytron** サブシス

テムと統合する HotRod プロトコルをベースとするカスタムの最適化されたキャッシュストアが導入されました。「**HTTP セッションの JBoss Data Grid への外部化**」の説明にしたがって、新しい **hotrod** ストアを使用することが推奨されます。



## 注記

分散可能なアプリケーションごとに完全に新しいキャッシュを作成する必要があります。新しいキャッシュは **web** などの既存のキャッシュコンテナに作成できます。

HTTP セッションを外部化するには、以下を行います。

1. ネットワーク情報を **socket-binding-group** に追加することにより、リモート Red Hat Data Grid サーバーの場所を定義します。

### 例: リモートソケットバインディングの追加

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-rhdg-server1:add(host=RHDGHostName1, port=11222)
```

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-rhdg-server2:add(host=RHDGHostName2, port=11222)
```

### 結果の XML

```
<socket-binding-group name="standard-sockets" ... >
...
<outbound-socket-binding name="remote-rhdg-server1">
  <remote-destination host="RHDGHostName1" port="11222"/>
</outbound-socket-binding>
<outbound-socket-binding name="remote-rhdg-server2">
  <remote-destination host="RHDGHostName2" port="11222"/>
</outbound-socket-binding>
</socket-binding-group>
```



## 注記

各 Red Hat Data Grid サーバーにリモートソケットバインディングを設定する必要があります。

2. リモートキャッシュコンテナが JBoss EAP の **infinispan** サブシステムで定義されているようにしてください。以下の例では、**remote-store** 要素の **cache** 属性によって、リモート Red Hat Data Grid サーバーのキャッシュ名が定義されます。管理対象ドメインで実行している場合は、コマンドの前に **/profile=PROFILE\_NAME** を追加してください。

### 例: リモートキャッシュコンテナの追加

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

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

```
/subsystem=infinispan/cache-container=web/invalidation-
```

```
cache=rhdg/component=transaction:write-attribute(name=mode,value=BATCH)

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

### 結果の XML

```
<subsystem xmlns="urn:jboss:domain:infinispan:7.0">
  ...
  <cache-container name="web" default-cache="dist"
module="org.wildfly.clustering.web.infinispan" statistics-enabled="true">
    <transport lock-timeout="60000"/>
    <invalidation-cache name="rhdg" mode="SYNC">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <remote-store cache="default" socket-timeout="60000" remote-servers="remote-rhdg-
server1 remote-rhdg-server2" passivation="false" purge="false" shared="true"/>
    </invalidation-cache>
    ...
  </cache-container>
</subsystem>
```

3. キャッシュ情報をアプリケーションの **jboss-web.xml** に追加します。以下の例では、**web** はキャッシュコンテナの名前で、**rhdg** はこのコンテナにある適切なキャッシュの名前になります。

### 例: jboss-web.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-web_10_0.xsd"
  version="10.0">
  <replication-config>
    <replication-granularity>SESSION</replication-granularity>
    <cache-name>web.rhdg</cache-name>
  </replication-config>
</jboss-web>
```

## 24.4. JBOSS EAP をフロントエンドロードバランサーとして設定

バックエンド JBoss EAP サーバーへリクエストをプロキシするフロントエンドロードバランサーとして動作するよう JBoss EAP と **undertow** サブシステムを設定できます。Undertow は非同期 IO を使用するため、リクエストに関与するスレッドは接続用の IO スレッドのみになります。同じスレッドがバックエンドサーバーへの接続に使用されます。

以下のプロトコルを使用できます。

- HTTP/1 および HTTP/2 (**h2c**) をサポートするプレーンテキスト上の HTTP (**http**)
- HTTP/1 および HTTP/2 (**h2c**) をサポートするセキュアな接続上の HTTP (**https**)

- AJP (**ajp**)

静的ロードバランサーを定義して設定でバックエンドホストを指定するか、`mod_cluster` フロントエンドを使用してホストを動的に更新します。

HTTP/2 を使用するよう Undertow を設定する手順は、「[HTTP/2 の設定](#)」を参照してください。

### 24.4.1. `mod_cluster` を使用して Undertow をロードバランサーとして設定

組み込みの `mod_cluster` フロントエンドロードバランサーを使用して、他の EAP インスタンスを負荷分散できます。

この手順では、管理対象ドメインを実行し、以下が設定済みであることを前提としています。

- ロードバランサーとして動作する JBoss EAP サーバー。
  - このサーバーは、**load-balancer-sockets** ソケットバインディンググループにバインドされる **load-balancer** プロファイルを使用します。



#### 注記

このサーバーをフロントエンドロードバランサーとして使用するため、**load-balancer** プロファイルには、ソケットバインディング、`mod-cluster` Undertow フィルター、およびデフォルトホストでのフィルターへの参照がすでに事前設定されています。

- バックエンドサーバーとして動作する 2 つの JBoss EAP サーバー。
  - これらのサーバーはクラスターで実行され、**ha-sockets** ソケットバインディンググループにバインドされる **ha** プロファイルを使用します。
- バックエンドサーバーにデプロイされた負荷分散される分散可能なアプリケーション。

#### `mod_cluster` フロントエンドロードバランサーの設定

以下の手順は、管理対象ドメインのサーバーを負荷分散しますが、手順を変更するとスタンドアロンサーバーのセットに適用することができます。ご使用の環境に応じて管理 CLI コマンドの値を変更してください。

1. `mod_cluster` アドバタイズセキュリティーキーを設定します。  
アドバタイズセキュリティーキーを追加すると、ロードバランサーとサーバーが検出中に認証されます。

以下の管理 CLI コマンドを使用して、`mod_cluster` アドバタイズセキュリティーキーを設定します。

```
/profile=ha/subsystem=modcluster/proxy=default:write-attribute(name=advertise-security-key,value=mypassword)
```

2. `mod_cluster` ロードバランサーのセキュリティーキーを更新します。  
以下の管理 CLI コマンドを使用して、`mod_cluster` フィルターのセキュリティーキーを設定します。

```
/profile=load-balancer/subsystem=undertow/configuration=filter/mod-cluster=load-balancer:write-attribute(name=security-key,value=mypassword)
```



## 重要

mod\_cluster によって使用される管理およびアドバタイズソケットバインディングが内部ネットワークのみで公開され、パブリック IP アドレスで公開されないことが推奨されません。

ロードバランサーである JBoss EAP サーバーが2つのバックエンドである JBoss EAP サーバーを負荷分散できるようになります。

### 複数の mod\_cluster 設定

mod\_cluster サブシステムは名前の付いたプロキシー設定を複数サポートし、デフォルトでない undertow サーバーをリバースプロキシーに登録できます。さらに、単一のアプリケーションサーバーノードを異なるグループのプロキシーサーバーに登録することができます。

以下の例は、ajp-listener、サーバー、および undertow サーバーへのホストを追加します。また、アドバタイズのメカニズムを使用してホストに登録する新しい mod\_cluster 設定も追加します。

```
/socket-binding-group=standard-sockets/socket-binding=ajp-other:add(port=8010)
/subsystem=undertow/server=other-server:add
/subsystem=undertow/server=other-server/ajp-listener=ajp-other:add(socket-binding=ajp-other)
/subsystem=undertow/server=other-server/host=other-host:add(default-web-module=root-other.war)
/subsystem=undertow/server=other-server/host=other-host
/location=other:add(handler=welcome-content)
/subsystem=undertow/server=other-server/host=other-host:write-attribute(name=alias,value=[localhost])

/socket-binding-group=standard-sockets/socket-binding=modcluster-other:add(multicast-address=224.0.1.106,multicast-port=23364)
/subsystem=modcluster/proxy=other:add(advertise-socket=modcluster-other,balancer=other-balancer,connector=ajp-other)

reload
```

## 24.4.2. Undertow を静的ロードバランサーとして設定

Undertow の静的ロードバランサーを設定するには、undertow サブシステムでプロキシーハンドラーを設定する必要があります。Undertow でプロキシーハンドラーを設定するには、静的ロードバランサーとして動作する JBoss EAP インスタンスで以下を行う必要があります。

1. リバースプロキシーハンドラーを追加します。
2. 各リモートホストのアウトバウンドソケットバインディングを定義します。
3. 各リモートホストをリバースプロキシーハンドラーへ追加します。
4. リバースプロキシーの場所を追加します。



## 注記

ロードバランサーの背後でメッセージングを使用するには、静的な Undertow HTTP ロードバランサーを設定する必要があります。詳細は、JBoss EAP『[Configuring Messaging](#)』の「[Using Messaging Behind a Load Balanced](#)」を参照してください。

以下の例は、JBoss EAP インスタンスを静的ロードバランサーとして設定する方法を示しています。

JBoss EAP インスタンスは **lb.example.com** にあり、2つの追加サーバーである **server1.example.com** と **server2.example.com** との間で負荷分散を行います。ロードバランサーは **/app** に逆プロキシを行い、AJP プロトコルを使用します。

1. リバースプロキシハンドラーを追加するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-handler:add
```

2. 各リモートホストのアウトバウンドソケットバインディングを定義するには、以下を指定します。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-host1:add(host=server1.example.com, port=8009)
```

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-host2:add(host=server2.example.com, port=8009)
```

3. 各リモートホストをリバースプロキシハンドラーに追加するには、以下を指定します。

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-handler/host=host1:add(outbound-socket-binding=remote-host1, scheme=ajp, instance-id=myroute1, path=/test)
```

```
/subsystem=undertow/configuration=handler/reverse-proxy=my-handler/host=host2:add(outbound-socket-binding=remote-host2, scheme=ajp, instance-id=myroute2, path=/test)
```

4. リバースプロキシの場所を追加するには、以下を指定します。

```
/subsystem=undertow/server=default-server/host=default-host/location=/test:add(handler=my-handler)
```

**lb.example.com:8080/app** にアクセスすると、**server1.example.com** および **server2.example.com** からプロキシされた内容が表示されるようになります。

## 24.5. 外部 WEB サーバーのプロキシサーバーとしての使用

JBoss EAP は、外部 Web サーバーの設定に応じて、サポートされる HTTP、HTTPS、または AJP プロトコルを使用して外部 Web サーバーからリクエストを許可できます。

各 Web サーバーのサポートされる HTTP コネクタの詳細は、「[HTTP コネクタの概要](#)」を参照してください。使用する Web サーバーと HTTP コネクタを決定したら、適切なコネクタの設定情報の項を参照してください。

- [Apache HTTP Server](#) の場合は、「[mod\\_cluster](#)」、「[mod\\_jk](#)」、または「[mod\\_proxy](#)」の項を参照してください。
- Microsoft IIS の場合は、「[ISAPI コネクタ](#)」の項を参照してください。
- Oracle iPlanet Web Server の場合は、「[NSAPI コネクタ](#)」の項を参照してください。

HTTP コネクタのサポートされる構成に関する最新情報は、「[JBoss EAP でサポートされる構成](#)」を参照してください。



また、JBoss EAP が外部 Web サーバーからのリクエストを許可するよう設定されている必要があります。

### 24.5.1. HTTP コネクターの概要

JBoss EAP には、Apache HTTP Server、Microsoft IIS、Oracle iPlanet などの外部 Web サーバーや Undertow より構築された負荷分散およびクラスタリングメカニズムを使用する機能があります。JBoss EAP はコネクターを使用して Web サーバーと通信します。これらのコネクターは JBoss EAP の **undertow** サブシステム内に設定されます。

Web サーバーには、HTTP リクエストが JBoss EAP ノードにルーティングされる方法を制御するソフトウェアモジュールが含まれています。これらのモジュールの挙動や設定方法はモジュールごとに異なります。モジュールは、複数の JBoss EAP ノード全体でワークロードを分散したり、障害発生時にワークロードを他のサーバーに移動したりするために設定されます。

JBoss EAP は複数のコネクターをサポートします。使用中の Web サーバーや必要な機能に応じてコネクターを選択します。以下の表を参照して、サポートされる構成と、JBoss EAP と互換性のある HTTP コネクターの機能を比較してください。



#### 注記

JBoss EAP 7 をマルチプラットフォームロードバランサーとして使用する場合は「[mod\\_cluster を使用して Undertow をロードバランサーとして設定](#)」を参照してください。

HTTP コネクターのサポートされる構成に関する最新情報は、「[JBoss EAP でサポートされる構成](#)」を参照してください。

表24.1 HTTP コネクターでサポートされる構成

コネクター	Web Server	サポート対象オペレーティングシステム	サポート対象プロトコル
<a href="#">mod_cluster</a>	Red Hat JBoss Core Services の Apache HTTP Server、Red Hat JBoss Web Server の Apache HTTP Server、JBoss EAP (Undertow)	Red Hat Enterprise Linux、Microsoft Windows Server、Oracle Solaris	HTTP、HTTPS、AJP、WebSocket
<a href="#">mod_jk</a>	Red Hat JBoss Core Services の Apache HTTP Server、Red Hat JBoss Web Server の Apache HTTP Server	Red Hat Enterprise Linux、Microsoft Windows Server、Oracle Solaris	AJP
<a href="#">mod_proxy</a>	Red Hat JBoss Core Services の Apache HTTP Server、Red Hat JBoss Web Server の Apache HTTP Server	Red Hat Enterprise Linux、Microsoft Windows Server、Oracle Solaris	HTTP、HTTPS、AJP
<a href="#">ISAPI コネクター</a>	Microsoft IIS	Microsoft Windows Server	AJP
<a href="#">NSAPI コネクター</a>	Oracle iPlanet Web Server	Oracle Solaris	AJP

表24.2 HTTP コネクターの機能

コネクター	スティッキーセッションのサポート	デプロイメント状態への適合
<a href="#">mod_cluster</a>	○	可。アプリケーションのデプロイメントとアンデプロイメントを検出し、アプリケーションがそのサーバーにデプロイされたかどうかに基づいて、サーバーにクライアント要求を送信するかどうかを動的に決定します。
<a href="#">mod_jk</a>	○	不可。アプリケーションの状態に関係なく、コンテナが利用可能な限り、クライアント要求をコンテナに送信します。
<a href="#">mod_proxy</a>	○	不可。アプリケーションの状態に関係なく、コンテナが利用可能な限り、クライアント要求をコンテナに送信します。
ISAPI コネクター	○	不可。アプリケーションの状態に関係なく、コンテナが利用可能な限り、クライアント要求をコンテナに送信します。
NSAPI コネクター	○	不可。アプリケーションの状態に関係なく、コンテナが利用可能な限り、クライアント要求をコンテナに送信します。

## 24.5.2. Apache HTTP Server

スタンドアロン Apache HTTP Server バンドルは、Red Hat JBoss Core Services の個別ダウンロードとして利用できるようになりました。これにより、インストールと設定が容易になり、更新の一貫性が保たれます。

### 24.5.2.1. Apache HTTP Server のインストール

Apache HTTP Server のインストールに関する詳細は、JBoss Core Services の『[Apache HTTP Server Installation Guide](#)』を参照してください。

## 24.5.3. 外部 Web サーバーからのリクエストの許可

JBoss EAP に AJP、HTTP、HTTPS などの適切なプロトコルハンドラーが設定されていれば、プロキシサーバーからのリクエストを許可するための特別な設定は必要ありません。

プロキシサーバーが `mod_jk`、`mod_proxy`、ISAPI、または NSAPI を使用する場合、リクエストを JBoss EAP に送信し、JBoss EAP は応答を返信します。`mod_cluster` の場合、リクエストをどこにルーティングするかを判断できるようにするため、JBoss EAP が現在の負荷、アプリケーションライフサイクルイベント、ヘルス状態などの情報をプロキシサーバーへ送信できるようネットワークを設定する必要もあります。`mod_cluster` プロキシサーバーの設定に関する詳細は「[mod\\_cluster HTTP コネクター](#)」を参照してください。

### JBoss EAP 設定の更新

以下の手順では、例で使用されているプロトコルやポートを実際に設定する必要があるプロトコルやポートに置き換えてください。

1. Undertow の **instance-id** 属性を設定します。

外部 Web サーバーは **instance-id** を使用してコネクター設定の JBoss EAP インスタンスを識別します。以下の管理 CLI コマンドを使用して、Undertow で **instance-id** 属性を設定します。

```
/subsystem=undertow:write-attribute(name=instance-id,value=node1)
```

上記の例では、外部 Web サーバーは現在の JBoss EAP インスタンスを **node1** として識別します。

2. 必要なリスナーを Undertow に追加します。

外部 Web サーバーが JBoss EAP に接続するには、Undertow にリスナーが必要です。各プロトコルにはソケットバインディングに関連する独自のリスナーが必要です。



### 注記

プロトコルやポート設定によってはこの手順は必要でないことがあります。HTTP リスナーはデフォルトの JBoss EAP 設定すべてに設定されており、**ha** または **full-ha** プロファイルを使用している場合、AJP リスナーは設定されています。

デフォルトのサーバー設定を確認すると、必要なリスナーが設定済みであるかどうかを確認できます。

```
/subsystem=undertow/server=default-server:read-resource
```

リスナーを Undertow に追加するには、ソケットバインディングが必要です。ソケットバインディングは、サーバーまたはサーバーグループによって使用されるソケットバインディンググループに追加されます。以下の管理 CLI コマンドを使用すると **ajp** ソケットバインディングが追加され、ポート **8009** と **standard-sockets** ソケットバインディンググループにバインドされます。

```
/socket-binding-group=standard-sockets/socket-binding=ajp:add(port=8009)
```

以下の管理 CLI コマンドを使用すると **ajp** ソケットバインディングを使用して **ajp** リスナーが Undertow に追加されます。

```
/subsystem=undertow/server=default-server/ajp-listener=ajp:add(socket-binding=ajp)
```

## 24.6. MOD\_CLUSTER HTTP コネクター

mod\_cluster コネクターは Apache HTTP Server ベースのロードバランサーです。通信チャネルを使用して、リクエストを Apache HTTP Server からアプリケーションサーバーノードのセットの1つに転送します。

他のコネクターと比べて mod\_cluster コネクターには複数の利点があります。

- mod\_cluster Management Protocol (MCMP) は、JBoss EAP サーバーと mod\_cluster が有効な Apache HTTP Server との間の追加的な接続です。HTTP メソッドのカスタムセットを使用してサーバー側の負荷分散係数およびライフサイクルイベントを Apache HTTP Server サーバーに返信するために JBoss EAP サーバーによって使用されます。
- mod\_cluster がある Apache HTTP Server を動的に設定すると、手動設定を行わずに JBoss EAP サーバーが負荷分散に参加できます。

- JBoss EAP は、mod\_cluster がある Apache HTTP Server に依存せずに負荷分散係数の計算を行います。これにより、負荷分散メトリックが他のコネクタよりも正確になります。
- mod\_cluster コネクタにより、アプリケーションライフサイクルを細かく制御できるようになります。各 JBoss EAP サーバーは Web アプリケーションコンテキストライフサイクルイベントを Apache HTTP Server サーバーに転送し、指定コンテキストのルーティングリクエストを開始または停止するよう通知します。これにより、リソースを使用できないことが原因の HTTP エラーがエンドユーザーに表示されないようになります。
- AJP、HTTP、または HTTPS トランスポートを使用できます。

**modcluster** サブシステムの特定の設定オプションに関する詳細は、「[ModCluster サブシステムの属性](#)」を参照してください。

### 24.6.1. Apache HTTP Server の mod\_cluster の設定

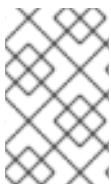
JBoss Core Services Apache HTTP Server をインストールする場合や JBoss Web Server を使用する場合、mod\_cluster モジュールはすでに含まれており、デフォルトでロードされます。



#### 注記

JBoss Web Server バージョン 3.1.0 より、Apache HTTP Server は配布されないようになりました。

以下の手順を参照し、お使いの環境に合った mod\_cluster モジュールを設定します。



#### 注記

Red Hat のお客様は Red Hat カスタマーポータルにある [Load Balancer Configuration Tool](#) を使用して mod\_cluster やその他のコネクタに最適な設定テンプレートを迅速に生成することもできます。このツールを使用するにはログインする必要があります。

#### mod\_cluster の設定

Apache HTTP Server には、mod\_cluster モジュールをロードし、基本設定を提供する mod\_cluster 設定ファイル **mod\_cluster.conf** がすでに含まれています。このファイルに指定されている IP アドレス、ポート、およびその他の設定 (以下参照) は必要に応じて変更できます。

```
# mod_proxy_balancer should be disabled when mod_cluster is used
LoadModule proxy_cluster_module modules/mod_proxy_cluster.so
LoadModule cluster_slotmem_module modules/mod_cluster_slotmem.so
LoadModule manager_module modules/mod_manager.so
LoadModule advertise_module modules/mod_advertise.so
```

```
MemManagerFile cache/mod_cluster
```

```
<IfModule manager_module>
  Listen 6666
  <VirtualHost *:6666>
    <Directory />
      Require ip 127.0.0.1
    </Directory>
    ServerAdvertise on
    EnableMCPMReceive
  <Location /mod_cluster_manager>
```

```

SetHandler mod_cluster-manager
Require ip 127.0.0.1
</Location>
</VirtualHost>
</IfModule>

```

Apache HTTP Server サーバーはロードバランサーとして設定でき、JBoss EAP で実行されている **modcluster** サブシステムと動作します。JBoss EAP が `mod_cluster` を認識するよう [mod\\_cluster ワーカーノードを設定](#)する必要があります。

`mod_cluster` のアドバタイズを無効にし、代わりに静的プロキシリストを設定する場合は「[mod\\_cluster のアドバタイズの無効化](#)」を参照してください。Apache HTTP Server で使用できる `mod_cluster` 設定オプションの詳細は、「[Apache HTTP Server の mod\\_cluster ディレクティブ](#)」を参照してください。

`mod_cluster` の設定に関する詳細は、JBoss Web Server『[HTTP Connectors and Load Balancing Guide](#)』の「[Configure Load Balancing Using Apache HTTP Server and mod\\_cluster](#)」を参照してください。

## 24.6.2. mod\_cluster のアドバタイズの無効化

デフォルトでは、**modcluster** サブシステムのバランサーはマルチキャスト UDP を使用して可用性をバックグラウンドワーカーにアドバタイズします。アドバタイズを無効にし、代わりにプロキシリストを使用する場合は、以下の手順に従ってください。



### 注記

以下の手順の管理 CLI コマンドは、管理対象ドメインで **full-ha** プロファイルを使用していることを前提としています。**full-ha** 以外のプロファイルを使用している場合は、コマンドに適切なプロファイル名を使用してください。スタンドアロンサーバーを実行している場合は `/profile=full-ha` を削除してください。

1. Apache HTTP Server 設定を変更します。

**httpd.conf** Apache HTTP Server 設定ファイルを編集します。**EnableMCPMReceive** ディレクティブを使用して、MCPM リクエストをリスンする仮想ホストに以下の更新を加えてください。

- a. サーバーアドバタイズメントを無効にするディレクティブを追加します。

**ServerAdvertise** ディレクティブを **Off** に設定し、サーバーのアドバタイズを無効にします。

```
ServerAdvertise Off
```

- b. アドバタイズの頻度を無効にします。

設定に **AdvertiseFrequency** が指定されている場合は `#` 文字を使用してコメントアウトします。

```
# AdvertiseFrequency 5
```

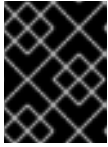
- c. MCPM メッセージの受信機能を有効にします。

Web サーバーがワーカーノードから MCPM メッセージを受信できるようにするため、必ず **EnableMCPMReceive** ディレクティブが存在するようにしてください。

## EnableMCPMReceive

2. **modcluster** サブシステムでアドバタイズを無効にします。  
以下の管理 CLI コマンドを使用してアドバタイズを無効にします。

```
/profile=full-ha/subsystem=modcluster/proxy=default:write-attribute(name=advertise,value=false)
```



### 重要

必ず次のステップでプロキシのリストを提供してください。プロキシのリストが空であるとアドバタイズが無効になりません。

3. JBoss EAP の **modcluster** サブシステムにプロキシのリストを提供します。  
アドバタイズが無効になっていると **modcluster** サブシステムは自動的にプロキシを検出できないため、プロキシのリストを提供する必要があります。

最初に、適切なソケットバインディンググループにアウトバウンドソケットバインディングを定義します。

```
/socket-binding-group=full-ha-sockets/remote-destination-outbound-socket-binding=proxy1:add(host=10.33.144.3,port=6666)
/socket-binding-group=full-ha-sockets/remote-destination-outbound-socket-binding=proxy2:add(host=10.33.144.1,port=6666)
```

次に、プロキシを `mod_cluster` 設定に追加します。

```
/profile=full-ha/subsystem=modcluster/proxy=default:list-add(name=proxies,value=proxy1)
/profile=full-ha/subsystem=modcluster/proxy=default:list-add(name=proxies,value=proxy2)
```

Apache HTTP Server の balancer がその存在をワーカーノードにアドバタイズしなくなり、UDP マルチキャストが使用されないようになります。

### 24.6.3. mod\_cluster ワーカーノードの設定

`mod_cluster` ワーカーノードは、JBoss EAP サーバーで構成されます。このサーバーは、スタンドアロンサーバーまたは管理対象ドメインのサーバーグループの一部になります。個別のプロセスが JBoss EAP 内で実行され、クラスターのワーカーノードをすべて管理します。これはマスターと呼ばれます。

管理対象ドメインのワーカーノードは、サーバーグループ全体で同じ設定を共有します。スタンドアロンサーバーとして実行しているワーカーノードは個別に設定されます。設定手順は同じです。

- スタンドアロンサーバーは **standalone-ha** または **standalone-full-ha** プロファイルで起動する必要があります。
- 管理対象ドメインのサーバーグループは **ha** または **full-ha** プロファイルを使用し、**ha-sockets** または **full-ha-sockets** ソケットバインディンググループを使用する必要があります。JBoss EAP にはこれらの要件を満たし、クラスターが有効になっている **other-server-group** というサーバーグループが含まれます。

ワーカーノードの設定

この手順の管理 CLI コマンドは、**full-ha** プロファイルの管理対象ドメインを使用していることを前提としています。スタンドアロンサーバーを実行している場合は、コマンドの **/profile=full-ha** の部分を削除してください。

### 1. ネットワークインターフェースの設定

デフォルトでは、ネットワークインターフェースがすべて **127.0.0.1** に設定されます。スタンドアロンサーバーまたはサーバーグループ内の1つまたは複数のサーバーをホストする各物理ホストでは、インターフェースが他のサーバーが見つめることができるパブリック IP アドレスを使用するよう設定する必要があります。

以下の管理 CLI コマンドを使用して、ご使用の環境に合うよう **management**、**public**、および **unsecure** インターフェースの外部 IP アドレスを変更します。コマンドの **EXTERNAL\_IP\_ADDRESS** をホストの実際の外部 IP アドレスに置き換えてください。

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

サーバーをリロードします。

```
reload
```

### 2. ホスト名を設定します。

管理対象ドメインに参加する各ホストに一意なホスト名を設定します。この名前はスレーブ全体で一意になる必要があり、スレーブがクラスターを識別するために使用されるため、使用する名前を覚えておくようにしてください。

- a. 適切な **host.xml** 設定ファイルを使用して JBoss EAP のスレーブホストを起動します。

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

- b. 以下の管理 CLI コマンドを使用して、一意なホスト名を設定します。この例では、**slave1** が新しいホスト名として使用されます。

```
/host=EXISTING_HOST_NAME:write-attribute(name=name,value=slave1)
```

ホスト名の設定に関する詳細は、「[ホスト名の設定](#)」を参照してください。

### 3. ドメインコントローラーに接続する各ホストを設定します。



#### 注記

このステップはスタンドアロンサーバーには適用されません。

新たに設定されたホストが管理対象ドメインに参加する必要がある場合、ローカル要素を削除し、ドメインコントローラーを示すリモート要素ホスト属性を追加する必要があります。

- a. 適切な **host.xml** 設定ファイルを使用して JBoss EAP のスレーブホストを起動します。

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

- b. 以下の管理 CLI コマンドを使用して、ドメインコントローラーを設定します。

```
/host=SLAVE_HOST_NAME:write-remote-domain-
controller(host=DOMAIN_CONTROLLER_IP_ADDRESS,port=${jboss.domain.master.port:9990},security-realm="ManagementRealm")
```

これにより、host-slave.xml ファイルの XML が次のように変更されます。

```
<domain-controller>
  <remote host="DOMAIN_CONTROLLER_IP_ADDRESS"
  port="${jboss.domain.master.port:9990}" security-realm="ManagementRealm"/>
</domain-controller>
```

詳細は、「[ホストコントローラーの設定](#)」を参照してください。

4. 各スレーブホストの認証を設定します。

各スレーブサーバーには、ドメインコントローラーまたはスタンドアロンマスターの ManagementRealm で作成されたユーザー名とパスワードが必要です。ドメインコントローラーまたはスタンドアロンマスターで、各ホストに対して **EAP\_HOME/bin/add-user.sh** コマンドを実行します。スレーブのホスト名と一致するユーザー名で、各ホストの管理ユーザーを追加します。

秘密の値が提供されるようにするため、必ず最後の質問 (Is this new user going to be used for one AS process to connect to another AS process?) に **yes** と返答してください。

#### 例: add-user.sh スクリプト出力 (抜粋)

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

```
What type of user do you wish to add?
```

- a) Management User (mgmt-users.properties)
- b) Application User (application-users.properties)

```
(a): a
```

```
Username : slave1
```

```
Password : changeme
```

```
Re-enter Password : changeme
```

```
What groups do you want this user to belong to? (Please enter a comma separated list, or
leave blank for none)[ ]:
```

```
About to add user 'slave1' for realm 'ManagementRealm'
```

```
Is this correct yes/no? yes
```

```
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for
server to server EJB calls.
```

```
yes/no? yes
```

```
To represent the user add the following to the server-identities definition <secret
value="SECRET_VALUE" />
```

ここで出力された Base64 でエンコードされた秘密の値 **SECRET\_VALUE** をコピーします。この値は次のステップで使用することがあります。

詳細は、JBoss EAP 『[How To Configure Server Security](#)』の「[Adding a User to the Master Domain Controller](#)」を参照してください。

5. 新しい認証を使用するようスレーブホストのセキュリティーレームを変更します。



サーバー設定に秘密の値を指定する方法、認証情報ストアまたは vault からパスワードを取得する方法、またはパスワードをシステムプロパティとして渡す方法のいずれかでパスワードを指定できます。

- 管理 CLI を使用して、サーバー設定ファイルに Base64 でエンコードされたパスワード値を指定します。  
以下の管理 CLI コマンドを使用して秘密の値を指定します。必ず **SECRET\_VALUE** を前のステップの add-user 出力から返された秘密の値に置き換えてください。

```
/host=SLAVE_HOST_NAME/core-service=management/security-  
realm=ManagementRealm/server-identity=secret:add(value="SECRET_VALUE")
```

サーバーをリロードする必要があります。--host 引数はスタンドアロンサーバーには適用されません。

```
reload --host=HOST_NAME
```

詳細は、JBoss EAP 『[How To Configure Server Security](#)』の「[Configuring the Slave Controllers to Use the Credential](#)」を参照してください。

- ホストを設定し、認証情報ストアからパスワードを取得します。  
秘密の値を認証情報ストアに保存した場合、以下のコマンドを使用してサーバーの秘密の値が認証情報ストアからの値になるよう設定できます。

```
/host=SLAVE_HOST_NAME/core-service=management/security-  
realm=ManagementRealm/server-identity=secret:add(credential-reference=  
{store=STORE_NAME,alias=ALIAS})
```

サーバーをリロードする必要があります。--host 引数はスタンドアロンサーバーには適用されません。

```
reload --host=HOST_NAME
```

詳細は、JBoss EAP 『[How To Configure Server Security](#)』の「[Credential Store](#)」を参照してください。

- ホストを設定し、vault からパスワードを取得します。
  - a. **EAP\_HOME/bin/vault.sh** スクリプトを使用してマスクされたパスワードを生成します。以下のような **VAULT::secret::password::VAULT\_SECRET\_VALUE** 形式の文字列が生成されます。

```
VAULT::secret::password::ODVmYmJjNGMtZDU2ZC00YmNILWE4ODMtZjQ1NWNm  
NDU4ZDc1TEIORV9CUkVBS3ZhdWx0.
```



### 注記

vault でパスワードを作成する場合、Base64 エンコードではなくプレーンテキストで指定する必要があります。

- b. 以下の管理 CLI コマンドを使用して秘密の値を指定します。必ず **VAULT\_SECRET\_VALUE** を前のステップで生成したマスクされたパスワードに置き換えてください。

■

```
/host=master/core-service=management/security-realm=ManagementRealm/server-identity=secret:add(value="{VAULT::secret::password::VAULT_SECRET_VALUE}")
```

サーバーをリロードする必要があります。 **--host** 引数はスタンドアロンサーバーには適用されません。

```
reload --host=HOST_NAME
```

詳細は、JBoss EAP 『[How To Configure Server Security](#)』の「[Password Vault](#)」を参照してください。

- システムプロパティーとしてパスワードを指定します。  
次の例は、**server.identity.password** をパスワードのシステムプロパティー名として使用します。
  - a. サーバー設定ファイルでパスワードのシステムプロパティーを指定します。  
以下の管理 CLI コマンドを使用して、システムプロパティーを使用する秘密のアイデンティティーを設定します。

```
/host=SLAVE_HOST_NAME/core-service=management/security-realm=ManagementRealm/server-identity=secret:add(value="{server.identity.password}")
```

サーバーをリロードする必要があります。 **--host** 引数はスタンドアロンサーバーには適用されません。

```
reload --host=master
```

- b. サーバーの起動時にシステムプロパティーのパスワードを設定します。  
**server.identity.password** システムプロパティーを設定するには、このプロパティーをコマンドライン引数として渡すか、プロパティーファイルで渡します。
  - i. プレーンテキストのコマンドライン引数として渡します。  
サーバーを起動し、**server.identity.password** プロパティーを渡します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml -Dserver.identity.password=changeme
```



### 警告

パスワードはプレーンテキストで入力する必要があります。  
**ps -ef** コマンドを実行すると、このパスワードを確認できます。

- ii. プロパティーファイルでプロパティーを設定します。  
プロパティーファイルを作成し、キーバリューペアをプロパティーファイルに追加します。例を以下に示します。

```
server.identity.password=changeme
```



### 警告

パスワードはプレーンテキストで、このプロパティファイルにアクセスできるユーザーはパスワードを確認できます。

コマンドライン引数を使用してサーバーを起動します。

```
$ EAP_HOME/bin/domain.sh --host-config=host-slave.xml --
properties=PATH_TO_PROPERTIES_FILE
```

#### 6. サーバーを再起動します。

ホスト名をユーザー名として使用し、暗号化された文字列をパスワードとして使用してスレーブがマスターに対して認証されます。

スタンドアロンサーバーまたは管理対象ドメインのサーバーグループ内のサーバーが `mod_cluster` ワーカーノードとして設定されます。クラスター化されたアプリケーションをデプロイする場合、セッションはフェイルオーバーのためにすべてのクラスターサーバーに複製され、外部 Web サーバーまたはロードバランサーからのリクエストを許可できます。クラスターの各ノードは、デフォルトで自動検出を使用して他のノードを検出します。

### 24.6.4. `mod_cluster` の `fail_on_status` パラメーターの設定

`fail_on_status` パラメーターは、クラスターのワーカーノードによって返されるとそのノードの失敗を意味する HTTP ステータスコードをリストします。ロードバランサーはその後のリクエストをクラスターの別のワーカーノードに送信します。失敗したワーカーノードは、ロードバランサーに **STATUS** メッセージを送信するまで **NOTOK** の状態になります。

`fail_on_status` パラメーターはロードバランサーの `httpd` 設定ファイルに設定する必要があります。`fail_on_status` の HTTP ステータスコードが複数ある場合はカンマで区切って指定します。以下の例は、HTTP ステータスコード **203** および **204** を `fail_on_status` に指定します。

#### 例: `fail_on_status` の設定

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

### 24.6.5. クラスター間のトラフィックの移行

JBoss EAP を使用して新しいクラスターを作成した後、アップグレードプロセスの一部として以前のクラスターから新しいクラスターへトラフィックを移行できます。ここでは、停止時間やダウンタイムを最小限に抑えてトラフィックを移行する方法について説明します。

- 新しいクラスターの設定。このクラスターを **ClusterNEW** と呼びます。

- 不要となった古いクラスターの設定。このクラスターを **ClusterOLD** と呼びます。

### クラスターのアップグレードプロセス - ロードバランシンググループ

1. 前提条件に従って、新しいクラスターを設定します。
2. **ClusterNEW** および **ClusterOLD** の両方で、設定オプション **sticky-session** をデフォルト設定の **true** に設定してください。このオプションを有効にすると、クラスターのクラスターノードへの新しいリクエストはすべてそのクラスターノードに送信されます。

```
/profile=full-ha/subsystem=modcluster/proxy=default:write-attribute(name=sticky-session,value=true)
```

3. **ClusterOLD** のすべてのクラスターノードは **ClusterOLD** ロードバランシンググループのメンバーであることを仮定し、**load-balancing-group** を **ClusterOLD** に設定します。

```
/profile=full-ha/subsystem=modcluster/proxy=default:write-attribute(name=load-balancing-group,value=ClusterOLD)
```

4. 「**mod\_cluster** ワーカーノードの設定」の手順に従って **ClusterNEW** のノードを個別に **mod\_cluster** 設定に追加します。また、この手順を使用してロードバランシンググループを **ClusterNEW** に設定します。  
この時点で、以下の簡易例に似た出力が **mod\_cluster-manager** コンソールに表示されます。

```
mod_cluster/<version>

LBGroup ClusterOLD: [Enable Nodes] [Disable Nodes] [Stop Nodes]
Node node-1-jvmroute (ajp://node1.oldcluster.example:8009):
  [Enable Contexts] [Disable Contexts] [Stop Contexts]
  Balancer: qacluster, LBGroup: ClusterOLD, Flushpackets: Off, ..., Load: 100
  Virtual Host 1:
    Contexts:
      /my-deployed-application-context, Status: ENABLED Request: 0 [Disable]
[Stop]

Node node-2-jvmroute (ajp://node2.oldcluster.example:8009):
  [Enable Contexts] [Disable Contexts] [Stop Contexts]
  Balancer: qacluster, LBGroup: ClusterOLD, Flushpackets: Off, ..., Load: 100
  Virtual Host 1:
    Contexts:
      /my-deployed-application-context, Status: ENABLED Request: 0 [Disable]
[Stop]

LBGroup ClusterNEW: [Enable Nodes] [Disable Nodes] [Stop Nodes]
Node node-3-jvmroute (ajp://node3.newcluster.example:8009):
  [Enable Contexts] [Disable Contexts] [Stop Contexts]
  Balancer: qacluster, LBGroup: ClusterNEW, Flushpackets: Off, ..., Load: 100
  Virtual Host 1:
    Contexts:
      /my-deployed-application-context, Status: ENABLED Request: 0 [Disable]
[Stop]

Node node-4-jvmroute (ajp://node4.newcluster.example:8009):
  [Enable Contexts] [Disable Contexts] [Stop Contexts]
```

```

Balancer: qacluster, LBGroup: ClusterNEW, Flushpackets: Off, ..., Load: 100
Virtual Host 1:
Contexts:
    /my-deployed-application-context, Status: ENABLED Request: 0 [Disable]

```

[Stop]

5. **ClusterOLD** グループ内に古いアクティブなセッションがあり、新しいセッションは **ClusterOLD** または **ClusterNEW** グループ内に作成されます。次に、現在アクティブなクライアントのセッションにエラーが発生しないようにクラスターノードを停止するため、**ClusterOLD** グループ全体を無効にします。  
**mod\_cluster-manager** Web コンソールで LBGroup **ClusterOLD** の **Disable Nodes** リンクをクリックします。

この後、すでに確立されたセッションに属するリクエストのみが **ClusterOLD** ロードバランシンググループのメンバーにルーティングされます。新しいクライアントのセッションは **ClusterNEW** グループのみに作成されます。**ClusterOLD** グループ内にアクティブなセッションがなくなったら、そのメンバーを安全に削除できます。



### 注記

**Stop Nodes** を使用すると、即座にこのドメインへのリクエストのルーティングを停止するようロードバランサーが指示されます。これにより、別のロードバランシンググループへのフェイルオーバーが強制され、**ClusterNEW** と **ClusterOLD** の間にセッションレプリケーションがない場合はクライアントへのセッションデータが損失する原因となります。

### デフォルトのロードバランシンググループ

**mod\_cluster-manager** コンソールの LBGroup を確認して、現在の **ClusterOLD** 設定にロードバランシンググループの設定が含まれていない場合でも、**ClusterOLD** ノードの無効化を利用できます。この場合、各 **ClusterOLD** ノードの **Disable Contexts** をクリックします。これらのノードのコンテンツは無効化され、アクティブなセッションがなくなったら削除することができます。新しいクライアントのセッションは、有効なコンテンツを持つノードのみに作成されます (この例では **ClusterOLD** メンバー)。

### 管理 CLI の使用

**mod\_cluster-manager** web コンソールを使用する他に、JBoss EAP 管理 CLI を使用して特定のコンテキストを停止または無効化することもできます。

### コンテキストの停止

```

/host=master/server=server-one/subsystem=modcluster:stop-context(context=/my-deployed-application-context, virtualhost=default-host, waittime=0)

```

**waittime** を **0** に設定してタイムアウトがない状態でコンテキストを停止すると、すべてのリクエストのルーティングを即座に停止するようバランサーに指示を出し、利用できる他のコンテキストへのフェイルオーバーを強制します。

**waittime** 引数を使用してタイムアウト値を設定すると、このコンテキストでは新しいセッションは作成されませんが、既存のセッションが完了するか、指定のタイムアウト値を経過するまで、既存のセッションはこのノードに引き続き転送されます。**waittime** 引数のデフォルト値は **10** 秒です。

### コンテキストの無効化

```
/host=master/server=server-one/subsystem=modcluster:disable-context(context=/my-deployed-application-context, virtualhost=default-host)
```

コンテキストを無効にすると、バランサーがこのコンテキストで新しいセッションを作成しないよう指示します。

## 24.7. APACHE MOD\_JK HTTP コネクタ

Apache mod\_jk は、互換性の維持を目的に提供される HTTP コネクタです。

JBoss EAP は Apache HTTP プロキシサーバーからのワークロードを許可します。プロキシサーバーは Web フロントエンドからのクライアントリクエストを許可し、ワークを参加する JBoss EAP サーバーへ渡します。スティッキーセッションが有効な場合、同じクライアントリクエストは常に同じ JBoss EAP サーバーに送信されます (同じサーバーが使用できない場合を除く)。

mod\_jk は AJP 1.3 プロトコルを介して通信します。mod\_cluster または mod\_proxy には他のプロトコルを使用できます。詳細は「[HTTP コネクタの概要](#)」を参照してください。



### 注記

mod\_cluster は mod\_jk よりも高度なロードバランサーで、推奨される HTTP コネクタです。mod\_cluster は mod\_jk のすべての機能と、それ以外の追加機能を提供します。JBoss EAP の mod\_cluster HTTP コネクタとは違い、Apache mod\_jk HTTP コネクタはサーバーまたはサーバーグループのデプロイメントの状態を認識せず、ワークの送信先に順応できません。

詳細は、[Apache mod\\_jk ドキュメント](#)を参照してください。

### 24.7.1. Apache HTTP Server での mod\_jk の設定

JBoss Core Services Apache HTTP Server のインストール時または JBoss Web Server の使用時に mod\_jk モジュールである **mod\_jk.so** はすでに含まれていますが、デフォルトではロードされません。



### 注記

JBoss Web Server バージョン 3.1.0 より、Apache HTTP Server は配布されないようになりました。

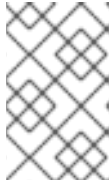
以下の手順に従って、Apache HTTP Server の mod\_jk をロードおよび設定します。この手順では、Apache HTTP Server の **httpd/** ディレクトリーがカレントディレクトリーであることを前提としていますが、このディレクトリーはプラットフォームによって異なります。ご使用のプラットフォームに対応するインストール手順は、JBoss Core Services の『[Apache HTTP Server Installation Guide](#)』を参照してください。



### 注記

Red Hat のお客様は Red Hat カスタマーポータルにある [Load Balancer Configuration Tool](#) を使用して mod\_jk やその他のコネクタに最適な設定テンプレートを迅速に生成することもできます。このツールを使用するにはログインする必要があります。

1. mod\_jk モジュールを設定します。



## 注記

mod\_jk 設定ファイルの例は **conf.d/mod\_jk.conf.sample** にあります。独自のファイルを作成せずにこのファイルを使用するには、**.sample** 拡張子を削除し、必要に応じて内容を変更します。

**conf.d/mod\_jk.conf** という新しいファイルを作成します。以下の設定をファイルに追加し、必要に応じて内容を変更します。

```
# Load mod_jk module
# Specify the filename of the mod_jk lib
LoadModule jk_module modules/mod_jk.so

# Where to find workers.properties
JkWorkersFile conf.d/workers.properties

# Where to put jk logs
JkLogFile logs/mod_jk.log

# Set the jk log level [debug/error/info]
JkLogLevel info

# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"

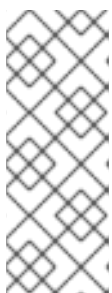
# JkOptions indicates to send SSK KEY SIZE
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories

# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"

# Mount your applications
JkMount /application/* loadbalancer

# Add shared memory.
# This directive is present with 1.2.10 and
# later versions of mod_jk, and is needed for
# for load balancing to work properly
JkShmFile logs/jk.shm

# Add jkstatus for managing runtime data
<Location /jkstatus/>
    JkMount status
    Require ip 127.0.0.1
</Location>
```



## 注記

JkMount ディレクティブは、Apache HTTP Server が mod\_jk モジュールに転送する必要がある URL を指定します。ディレクティブの設定に基づき、mod\_jk は受信した URL を適切なワーカーに送信します。直接静的コンテンツに対応し、Java アプリケーションのロードバランサーのみを使用するには、URL パスは **/application/\*** である必要があります。mod\_jk をロードバランサーとして使用するには、値 **\*** を使用してすべての URL を mod\_jk に転送します。

一般的な `mod_jk` の設定の他に、このファイルは **mod\_jk.so** モジュールをロードするよう指定し、**workers.properties** ファイルの場所を定義します。

2. `mod_jk` ワーカーノードを設定します。



### 注記

ワーカー設定ファイルの例は **conf.d/workers.properties.sample** にあります。独自のファイルを作成せずにこのファイルを使用するには、**.sample** 拡張子を削除し、必要に応じて内容を変更します。

**conf.d/workers.properties** という新しいファイルを作成します。以下の設定をファイルに追加し、必要に応じて内容を変更します。

```
# Define list of workers that will be used
# for mapping requests
worker.list=loadbalancer,status

# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=node1.mydomain.com
worker.node1.type=ajp13
worker.node1.ping_mode=A
worker.node1.lbfactor=1

# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host=node2.mydomain.com
worker.node2.type=ajp13
worker.node2.ping_mode=A
worker.node2.lbfactor=1

# Load-balancing behavior
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1

# Status worker for managing load balancer
worker.status.type=status
```

`mod_jk` **workers.properties** ファイルの構文の詳細およびその他の高度な設定オプションの詳細は、「[mod\\_jk ワーカープロパティ](#)」を参照してください。

3. 任意で `JKMountFile` ディレクティブを指定します。  
**mod-jk.conf** の `JKMount` ディレクティブの他に、`mod_jk` に転送される複数の URL パターンが含まれるファイルを指定できます。
  - a. **uriworkermap.properties** ファイルを作成します。





### 注記

URI ワーカーマップ設定ファイルの例は **conf.d/uriworkermap.properties.sample** にあります。独自のファイルを作成せずにこのファイルを使用するには、**.sample** 拡張子を削除し、必要に応じて内容を変更します。

**conf.d/uriworkermap.properties** という新しいファイルを作成します。以下の例のように、一致する各 URL パターンの行を追加します。

```
# Simple worker configuration file
/*=loadbalancer
```

- b. **uriworkermap.properties** ファイルを示すよう、設定を更新します。**conf.d/mod\_jk.conf** の最後に以下を追加します。

```
# Use external file for mount points.
# It will be checked for updates each 60 seconds.
# The format of the file is: /url=worker
# /examples/*=loadbalancer
JkMountFile conf.d/uriworkermap.properties
```

mod\_jk の設定に関する詳細は、JBoss Web Server 『[HTTP Connectors and Load Balancing Guide](#)』の「[Configuring Apache HTTP Server to Load mod\\_jk](#)」を参照してください。

## 24.7.2. JBoss EAP が mod\_jk と通信するよう設定

JBoss EAP の **undertow** サブシステムは、外部 Web サーバーからのリクエストを許可し、外部 Web サーバーへ返答を返送するために、リスナーを指定する必要があります。mod\_jk は AJP プロトコルを使用するため、AJP リスナーを設定する必要があります。

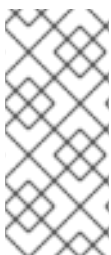
デフォルトの高可用性設定の1つ (**ha** または **full-ha**) を使用している場合は、AJP リスナーはすでに設定されています。

手順は「[外部 Web サーバーからのリクエストの許可](#)」を参照してください。

## 24.8. APACHE MOD\_PROXY HTTP コネクター

Apache **mod\_proxy** は、AJP、HTTP、および HTTPS プロトコルを介して接続をサポートする HTTP コネクターです。mod\_proxy は負荷分散された設定と負荷分散されていない設定が可能で、ステイキーセッションをサポートします。

mod\_proxy モジュールを使用するには、使用するプロトコルに応じて JBoss EAP の **undertow** サブシステムに HTTP、HTTPS または AJP リスナーを設定する必要があります。



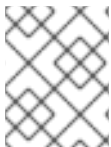
### 注記

**mod\_cluster** は mod\_proxy よりも高度なロードバランサーで、推奨される HTTP コネクターです。mod\_cluster は mod\_proxy のすべての機能と、それ以外の追加機能を提供します。JBoss EAP の mod\_cluster HTTP コネクターとは違い、Apache mod\_proxy HTTP コネクターはサーバーまたはサーバーグループのデプロイメントの状態を認識せず、ワークの送信先に順応できません。

詳細は [Apache mod\\_proxy ドキュメント](#) を参照してください。

### 24.8.1. Apache HTTP Server での mod\_proxy の設定

JBoss Core Services Apache HTTP Server をインストールする場合や JBoss Web Server を使用する場  
合、mod\_proxy モジュールはすでに含まれており、デフォルトでロードされます。



#### 注記

JBoss Web Server バージョン 3.1.0 より、Apache HTTP Server は配布されないようになりました。

基本の [ロードバランシング](#) または [非ロードバランシング](#) プロキシを設定するには、以下のセクションを参照してください。この手順では、Apache HTTP Server の **httpd/** ディレクトリーがカレントディレクトリーであることを前提としていますが、このディレクトリーはプラットフォームによって異なります。ご使用のプラットフォームに対応するインストール手順は、JBoss Core Services の『[Apache HTTP Server Installation Guide](#)』を参照してください。また、この手順は JBoss EAP の **undertow** サブシステムに必要な HTTP リスナーが設定されていることを前提としています。



#### 注記

Red Hat のお客様は Red Hat カスタマーポータルにある [Load Balancer Configuration Tool](#) を使用して mod\_proxy やその他のコネクターに最適な設定テンプレートを迅速に生成することもできます。このツールを使用するにはログインする必要があります。

非ロードバランシングプロキシの追加

以下の設定を、他の **<VirtualHost>** ディレクティブの直下にある **conf/httpd.conf** ファイルに追加します。値を設定に適切な値に置き換えます。

```
<VirtualHost *:80>
# Your domain name
ServerName YOUR_DOMAIN_NAME

ProxyPreserveHost On

# The IP and port of JBoss
# These represent the default values, if your httpd is on the same host
# as your JBoss managed domain or server

ProxyPass / http://localhost:8080/
ProxyPassReverse / http://localhost:8080/

# The location of the HTML files, and access control information
DocumentRoot /var/www
<Directory /var/www>
Options -Indexes
Order allow,deny
Allow from all
</Directory>
</VirtualHost>
```

ロードバランシングプロキシの追加



## 注記

デフォルトの Apache HTTP Server は `mod_cluster` との互換性がないため、**`mod_proxy_balancer.so`** モジュールが無効になっています。この作業を行うには、このモジュールをロードし、`mod_cluster` を無効にする必要があります。

`mod_proxy` をロードバランサーとして使用し、ワークを複数の JBoss EAP インスタンスに送信するには、以下の設定を **`conf/httpd.conf`** ファイルに追加する必要があります。IP アドレスの例は以下のようになります。ご使用の環境に適切な値に置き換えてください。

```
<Proxy balancer://mycluster>

Order deny,allow
Allow from all

# Add each JBoss Enterprise Application Server by IP address and port.
# If the route values are unique like this, one node will not fail over to the other.
BalancerMember http://192.168.1.1:8080 route=node1
BalancerMember http://192.168.1.2:8180 route=node2
</Proxy>

<VirtualHost *:80>
# Your domain name
ServerName YOUR_DOMAIN_NAME

ProxyPreserveHost On
ProxyPass / balancer://mycluster/

# The location of the HTML files, and access control information DocumentRoot /var/www
<Directory /var/www>
Options -Indexes
Order allow,deny
Allow from all
</Directory>

</VirtualHost>
```

上記の例はすべて HTTP プロトコルを使用して通信します。適切な `mod_proxy` モジュールをロードすれば AJP または HTTPS プロトコルを使用することもできます。詳細は [Apache mod\\_proxy ドキュメント](#) を参照してください。

### スティッキーセッションの有効化

スティッキーセッションを使用すると、クライアントリクエストが特定の JBoss EAP ワーカーに送信された場合に、ワーカーが利用不可能にならない限り、後続のリクエストがすべて同じワーカーに送信されます。これは、ほとんどの場合で推奨される動作です。

`mod_proxy` のスティッキーセッションを有効にするには、**`stickysession`** パラメーターを **`ProxyPass`** ステートメントに追加します。

```
ProxyPass / balancer://mycluster stickysession=JSESSIONID
```

追加のパラメーターを **`lbmethod`** や **`nofailover`** などの **`ProxyPass`** ステートメントに指定できます。使用可能なパラメーターの詳細は、[Apache mod\\_proxy ドキュメント](#) を参照してください。

## 24.8.2. JBoss EAP が `mod_proxy` と通信するよう設定

JBoss EAP の **undertow** サブシステムは、外部 Web サーバーからのリクエストを許可し、外部 Web サーバーへ返答を返送するために、リスナーを指定する必要があります。使用するプロトコルによっては、リスナーを設定する必要があることがあります。

HTTP リスナーは JBoss EAP のデフォルト設定に設定されます。デフォルトの高可用性設定である **ha** または **full-ha** のいずれかを使用している場合は、AJP リスナーも事前設定されています。

手順は「[外部 Web サーバーからのリクエストの許可](#)」を参照してください。

## 24.9. MICROSOFT ISAPI コネクタ

Internet Server API (ISAPI) は、Microsoft の IIS (インターネットインフォメーションサービス) などの Web サーバー向けの OLE サーバー拡張機能とフィルターを記述するために使用される API のセットです。**isapi\_redirect.dll** は IIS 向けに調整された **mod\_jk** の拡張機能です。**isapi\_redirect.dll** を使用すると、JBoss EAP インスタンスをワーカーノード、IIS をロードバランサーとして設定できます。



### 注記

Windows Server および IIS のサポートされる構成については、「[JBoss Enterprise Application Platform \(EAP\) 7 でサポートされる構成](#)」を参照してください。

### 24.9.1. Microsoft IIS が ISAPI コネクタを使用するよう設定

Red Hat カスタマーポータルから ISAPI コネクタをダウンロードします。

1. ブラウザーを開き、Red Hat カスタマーポータルで JBoss の [Software Downloads](#) ページにログインします。
2. **Product** ドロップダウンメニューから **Web Connectors** を選択します。
3. **Version** ドロップダウンメニューで最新バージョンの JBoss Core Services を選択します。
4. リストで **Red Hat JBoss Core Services ISAPI Connector** を見つけ、**Download** リンクをクリックします。
5. アーカイブを抽出し、**sbin** ディレクトリーの内容をサーバーの場所にコピーします。以下の手順は、内容が **C:\connectors\** にコピーされたことを前提としています。

IIS マネージャー (IIS 7) を使用して IIS リディレクターを設定するには、以下を行います。

1. **Start** → **Run** とクリックして IIS マネージャーを開き、**inetmgr** と入力します。
2. 左側のツリービューペインで **IIS 7** を展開します。
3. **ISAPI and CGI Registrations** をダブルクリックし、新しいウィンドウで開きます。
4. **Actions** ペインで **Add** をクリックします。**Add ISAPI or CGI Restriction** ウィンドウが開きます。
5. 以下の値を指定します。
  - **ISAPI or CGI Path** **C:\connectors\isapi\_redirect.dll**
  - **Description:** **jboss**
  - **Allow extension path to execute** チェックボックスを選択します。

6. **OK** をクリックして **Add ISAPI or CGI Restriction** ウィンドウを閉じます。
7. JBoss ネイティブ仮想ディレクトリーの定義
  - **Default Web Site** を右クリックし、**Add Virtual Directory** をクリックします。**Add Virtual Directory** ウィンドウが開きます。
  - 以下の値を指定して仮想ディレクトリーを追加します。
    - **Alias: jboss**
    - **Physical Path: C:\connectors\**
  - **OK** をクリックして値を保存し、**Add Virtual Directory** ウィンドウを閉じます。
8. JBoss ネイティブ ISAPI リダイレクトフィルターの定義
  - ツリービューペインで **Sites** → **Default Web Site** と展開します。
  - **ISAPI Filters** をダブルクリックします。**ISAPI Filters Features** ビューが表示されます。
  - **Actions** ペインで **Add** をクリックします。**Add ISAPI Filter** ウィンドウが表示されます。
  - 以下の値を **Add ISAPI Filter** ウィンドウに指定します。
    - **Filter name: jboss**
    - **Executable: C:\connectors\isapi\_redirect.dll**
  - **OK** をクリックして値を保存し、**Add ISAPI Filter** ウィンドウを閉じます。
9. ISAPI-dll ハンドラーの有効化
  - ツリービューペインの **IIS 7** をダブルクリックします。**IIS 7 Home Features View** が開きます。
  - **Handler Mappings** をダブルクリックします。**Handler Mappings Features View** が表示されます。
  - **Group by** コンボボックスで **State** を選択します。**Handler Mappings** が **Enabled and Disabled Groups** に表示されます。
  - **ISAPI-dll** を見つけます。**Disabled** グループにある場合は右クリックし、**Edit Feature Permissions** を選択します。
  - 以下のパーミッションを有効にします。
    - Read
    - Script
    - Execute
  - **OK** をクリックして値を保存し、**Edit Feature Permissions** ウィンドウを閉じます。

これで、ISAPI コネクターを使用するよう Microsoft IIS が設定されます。

## 24.9.2. ISAPI コネクターがクライアントリクエストを JBoss EAP に送信するよう設定

このタスクでは、JBoss EAP サーバーのグループが ISAPI コネクターからのリクエストを受け入れるよう設定します。ロードバランシングまたは高可用性フェイルオーバーの設定は含まれません。

この設定は IIS サーバーで行われ、[外部 Web サーバーからのリクエストを許可](#)するよう JBoss EAP が設定されていることを前提としています。また、IIS への完全な管理者アクセスが必要で、[IIS が ISAPI コネクターを使用するよう設定](#)されている必要があります。

プロパティファイルの作成およびリダイレクトの設定

1. ログ、プロパティファイル、およびロックファイルを格納するディレクトリーを作成します。  
以下の手順では、ディレクトリー **C:\connectors\** の使用を前提としています。異なるディレクトリーを使用する場合は、適切に手順を変更してください。
2. **isapi\_redirect.properties** ファイルを作成します。  
**C:\connectors\isapi\_redirect.properties** という新しいファイルを作成します。このファイルに次の内容をコピーします。

```
# Configuration file for the ISAPI Connector
# Extension uri definition
extension_uri=/jboss/isapi_redirect.dll

# Full path to the log file for the ISAPI Connector
log_file=c:\connectors\isapi_redirect.log

# Log level (debug, info, warn, error or trace)
log_level=info

# Full path to the workers.properties file
worker_file=c:\connectors\workers.properties

# Full path to the uriworkermap.properties file
worker_mount_file=c:\connectors\uriworkermap.properties

#Full path to the rewrite.properties file
rewrite_rule_file=c:\connectors\rewrite.properties
```

**rewrite.properties** ファイルを使用しない場合は、行の先頭に # 文字を記入して最後の行をコメントアウトします。

3. **uriworkermap.properties** ファイルの作成  
**uriworkermap.properties** ファイルには、デプロイされたアプリケーション URL と、それらへの要求を処理するワーカー間のマッピングが含まれます。以下のサンプルファイルはファイルの構文を示しています。**uriworkermap.properties** ファイルを **C:\connectors\** に格納します。

```
# images and css files for path /status are provided by worker01
/status=worker01
/images/*=worker01
/css/*=worker01

# Path /web-console is provided by worker02
# IIS (customized) error page is used for http errors with number greater or equal to 400
# css files are provided by worker01
/web-console/*=worker02;use_server_errors=400
/web-console/css/*=worker01
```

```
# Example of exclusion from mapping, logo.gif won't be displayed
# !/web-console/images/logo.gif=*

# Requests to /app-01 or /app-01/something will be routed to worker01
/app-01|/*=worker01

# Requests to /app-02 or /app-02/something will be routed to worker02
/app-02|/*=worker02
```

#### 4. **workers.properties** ファイルを作成します。

**workers.properties** ファイルには、ワーカーラベルとサーバーインスタンス間のマッピング定義が含まれます。このファイルは、[Apache mod\\_jk ワーカープロパティ](#) 設定で使用される同じファイルの構文を使用します。

以下は **workers.properties** ファイルの例になります。ワーカー名、**worker01**、および **worker02** は、JBoss EAP の **undertow** サブシステムで設定された **instance-id** に一致する必要があります。

このファイルを **C:\connectors** ディレクトリーに格納してください。

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

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

# First JBoss EAP server definition, port 8009 is standard port for AJP in EAP
worker.worker01.host=127.0.0.1
worker.worker01.port=8009
worker.worker01.type=ajp13

# Second JBoss EAP server definition
worker.worker02.host=127.0.0.100
worker.worker02.port=8009
worker.worker02.type=ajp13
```

#### 5. **rewrite.properties** ファイルを作成します。

**rewrite.properties** ファイルには、特定のアプリケーションの単純な URL 書き換えルールが含まれます。以下の例で示されているように、書き換えられたパスは名前と値のペアを使用して指定されます。このファイルを **C:\connectors** ディレクトリーに格納してください。

```
#Simple example
# Images are accessible under abc path
/app-01/abc/=/app-01/images/
```

#### 6. **net stop** および **net start** コマンドを使用して IIS サーバーを再起動します。

```
C:\> net stop was /Y
C:\> net start w3svc
```

アプリケーションごとに、設定した特定の JBoss EAP サーバーにクライアントリクエストを送信するよう IIS サーバーが設定されます。

### 24.9.3. ISAPI コネクターがクライアントリクエストを複数の JBoss EAP サーバーで分散するよう設定

この設定は、指定する JBoss EAP サーバー全体でクライアントリクエストを分散します。この設定は IIS サーバーで行われ、[外部 Web サーバーからのリクエストを許可](#)するよう JBoss EAP が設定されていることを前提としています。また、IIS への完全な管理者アクセスが必要で、[IIS が ISAPI コネクターを使用するよう設定](#)されている必要があります。

複数のサーバー間でのクライアント要求の分散

1. ログ、プロパティファイル、およびロックファイルを格納するディレクトリーを作成します。  
以下の手順では、ディレクトリー **C:\connectors\** の使用を前提としています。異なるディレクトリーを使用する場合は、適切に手順を変更してください。
2. **isapi\_redirect.properties** ファイルを作成します。  
**C:\connectors\isapi\_redirect.properties** という新しいファイルを作成します。このファイルに次の内容をコピーします。

```
# Configuration file for the ISAPI Connector
# Extension uri definition
extension_uri=/jboss/isapi_redirect.dll

# Full path to the log file for the ISAPI Connector
log_file=c:\connectors\isapi_redirect.log

# Log level (debug, info, warn, error or trace)
log_level=info

# Full path to the workers.properties file
worker_file=c:\connectors\workers.properties

# Full path to the uriworkermap.properties file
worker_mount_file=c:\connectors\uriworkermap.properties

#OPTIONAL: Full path to the rewrite.properties file
rewrite_rule_file=c:\connectors\rewrite.properties
```

**rewrite.properties** ファイルを使用しない場合は、行の先頭に # 文字を記入して最後の行をコメントアウトします。

3. **uriworkermap.properties** ファイルを作成します。  
**uriworkermap.properties** ファイルには、デプロイされたアプリケーション URL と、それらへの要求を処理するワーカー間のマッピングが含まれます。以下のサンプルファイルは負荷分散が設定されたファイルの構文を示しています。ワイルドカード (\*) はさまざまな URL サブディレクトリーのすべてのリクエストを router という名前のロードバランサーに送信します。ロードバランサーの設定は次のステップで説明します。

**uriworkermap.properties** ファイルを **C:\connectors\** に格納します。

```
# images, css files, path /status and /web-console will be
# provided by nodes defined in the load-balancer called "router"
/css/*=router
/images/*=router
/status=router
/web-console|/*=router

# Example of exclusion from mapping, logo.gif won't be displayed
# !/web-console/images/logo.gif=*
```



```
# Requests to /app-01 and /app-02 will be routed to nodes defined
# in the load-balancer called "router"
/app-01/*=router
/app-02/*=router

# mapping for management console, nodes in cluster can be enabled or disabled here
/jkmanager/*=status
```

#### 4. **workers.properties** ファイルを作成します。

**workers.properties** ファイルには、ワーカーラベルとサーバーインスタンス間のマッピング定義が含まれます。このファイルは、[Apache mod\\_jk ワーカープロパティ](#) 設定で使用される同じファイルの構文を使用します。

以下は **workers.properties** ファイルの例になります。ロードバランサーはファイルの末尾付近に設定され、ワーカー **worker01** および **worker02** で構成されます。これらのワーカーは、**JBoss EAP** の **undertow** サブシステムで設定された **instance-id** に一致する必要があります。

このファイルを **C:\connectors** ディレクトリーに格納してください。

```
# The advanced router LB worker
worker.list=router,status

# First EAP server definition, port 8009 is standard port for AJP in EAP
#
# lbfactor defines how much the worker will be used.
# The higher the number, the more requests are served
# lbfactor is useful when one machine is more powerful
# ping_mode=A – all possible probes will be used to determine that
# connections are still working

worker.worker01.port=8009
worker.worker01.host=127.0.0.1
worker.worker01.type=ajp13
worker.worker01.ping_mode=A
worker.worker01.socket_timeout=10
worker.worker01.lbfactor=3

# Second EAP server definition
worker.worker02.port=8009
worker.worker02.host=127.0.0.100
worker.worker02.type=ajp13
worker.worker02.ping_mode=A
worker.worker02.socket_timeout=10
worker.worker02.lbfactor=1

# Define the LB worker
worker.router.type=lb
worker.router.balance_workers=worker01,worker02

# Define the status worker for jkmanager
worker.status.type=status
```

## 5. `rewrite.properties` ファイルを作成します。

`rewrite.properties` ファイルには、特定のアプリケーションの単純な URL 書き換えルールが含まれます。以下の例で示されているように、書き換えられたパスは名前と値のペアを使用して指定されます。このファイルを **C:\connectors\** ディレクトリーに格納してください。

```
#Simple example
# Images are accessible under abc path
/app-01/abc/=/app-01/images/
Restart the IIS server.
```

```
Restart your IIS server by using the net stop and net start commands.
C:\> net stop was /Y
C:\> net start w3svc
```

IIS サーバーは、`workers.properties` ファイルで参照された JBoss EAP サーバーにクライアントリクエストを送信し、サーバー間で負荷を **1:3** の比率で分散するよう設定されます。この比率は、各サーバーに割り当てられた負荷分散係数 **lbfactor** から派生します。

## 24.10. ORACLE NSAPI コネクター

Netscape Server API (NSAPI) は、拡張機能をサーバーに実装するために Oracle iPlanet Web Server (旧名 Netscape Web Server) によって提供される API です。これらの拡張機能はサーバープラグインと呼ばれます。NSAPI コネクターは、Oracle iPlanet Web Server 向けに調整された `mod_jk` の拡張機能である **nsapi\_redirector.so** で使用されます。NSAPI コネクターを使用すると、JBoss EAP インスタンスをワーカーノード、Oracle iPlanet Web Server をロードバランサーとして設定できます。



### 注記

Solaris および Oracle iPlanet Web Server のサポートされる構成については、「[JBoss Enterprise Application Platform \(EAP\) 7 でサポートされる構成](#)」を参照してください。

### 24.10.1. iPlanet Web Server が NSAPI コネクターを使用するよう設定

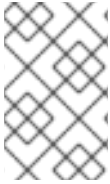
#### 要件

- ワーカーとして動作する各サーバーに JBoss EAP がインストールおよび設定されます。

Red Hat カスタマーポータルから NSAPI コネクターをダウンロードします。

1. ブラウザーを開き、Red Hat カスタマーポータルで JBoss の [Software Downloads](#) ページにログインします。
2. **Product** ドロップダウンメニューから **Web Connectors** を選択します。
3. **Version** ドロップダウンメニューで最新バージョンの JBoss Core Services を選択します。
4. システムのプラットフォームとアーキテクチャーに対応する **Red Hat JBoss Core Services NSAPI Connector** を見つけ、**Download** リンクをクリックします。
5. **lib/** または **lib64/** ディレクトリーにある **nsapi\_redirector.so** ファイルを、**IPLANET\_CONFIG/lib/** または **IPLANET\_CONFIG/lib64/** ディレクトリーに展開します。

NSAPI コネクターを設定します。



## 注記

これらの手順では、**IPLANET\_CONFIG** は Oracle iPlanet の設定ディレクトリーを意味します (通常 **/opt/oracle/webserver7/config/** になります)。Oracle iPlanet 設定ディレクトリーが異なる場合は、適切に手順を変更してください。

1. サブレットマッピングを無効にします。

**IPLANET\_CONFIG/default.web.xml** ファイルを開き、**Built In Server Mappings**という見出しのセクションを見つけます。次の3つのサブレットをXML コメント文字 (`<!--` および `-->`) で囲み、これらのサブレットへのマッピングを無効にします。

- default
- invoker
- jsp

以下の設定例は、無効にされたマッピングを示しています。

```
<!-- ===== Built In Servlet Mappings ===== -->
<!-- The servlet mappings for the built in servlets defined above. -->
<!-- The mapping for the default servlet -->
<!--servlet-mapping>
  <servlet-name>default</servlet-name>
  <url-pattern></url-pattern>
</servlet-mapping-->
<!-- The mapping for the invoker servlet -->
<!--servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping-->
<!-- The mapping for the JSP servlet -->
<!--servlet-mapping>
  <servlet-name>jsp</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping-->
```

ファイルを保存し、終了します。

2. iPlanet Web Server が NSAPI コネクタモジュールをロードするよう設定します。

**IPLANET\_CONFIG/magnus.conf** ファイルの最後に次の行を追加し、設定に合わせてファイルパスを変更します。これらの行は、**nsapi\_redirector.so** モジュールと、ワーカーおよびそのプロパティがリストされた **workers.properties** ファイルの場所を定義します。

```
Init fn="load-modules" funcs="jk_init,jk_service" shlib="/lib/nsapi_redirector.so" shlib_flags="
(global|now)"

Init fn="jk_init" worker_file="IPLANET_CONFIG/connectors/workers.properties"
log_level="info" log_file="IPLANET_CONFIG/connectors/nsapi.log"
shm_file="IPLANET_CONFIG/connectors/tmp/jk_shm"
```

上記の設定は 32 ビットアーキテクチャー向けです。64 ビット Solaris を使用している場合は、文字列 **lib/nsapi\_redirector.so** を **lib64/nsapi\_redirector.so** に変更します。

ファイルを保存し、終了します。

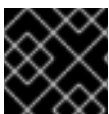
### 3. NSAPI コネクタを設定します。

負荷分散のない基本設定または負荷分散設定向けに NSAPI コネクタを設定できます。以下のいずれかのオプションを選択します。その後、設定が完了します。

- [NSAPI コネクタがクライアントリクエストを JBoss EAP に送信するよう設定](#)
- [NSAPI コネクタがクライアントリクエストを複数の JBoss EAP サーバーで分散するよう設定](#)

## 24.10.2. NSAPI コネクタがクライアントリクエストを JBoss EAP に送信するよう設定

このタスクでは、NSAPI コネクタが負荷分散またはフェイルオーバーなしでクライアントリクエストを JBoss EAP サーバーにリダイレクトするよう設定します。リダイレクトは、デプロイメントごとに（つまり、URL ごとに）行われます。



### 重要

このタスクを続行するには、[NSAPI コネクタが設定](#)されている必要があります。

### 基本的な HTTP コネクタの設定

1. JBoss EAP サーバーにリダイレクトする URL パスを定義します。



### 注記

**IPLANET\_CONFIG/obj.conf** では、前の行から継続する行以外は、行の最初にスペースを挿入しないでください。

**IPLANET\_CONFIG/obj.conf** ファイルを編集します。<Object name="default">で始まるセクションを見つけ、一致する各 URL パターンを次のサンプルファイルで示された形式で追加します。文字列 `jknsapi` は、次の手順で定義される HTTP コネクタを示します。例は、ワイルドカードを使用したパターン一致を示しています。

```
<Object name="default">
[...]
NameTrans fn="assign-name" from="/status" name="jknsapi"
NameTrans fn="assign-name" from="/images(/*)" name="jknsapi"
NameTrans fn="assign-name" from="/css(/*)" name="jknsapi"
NameTrans fn="assign-name" from="/nc(/*)" name="jknsapi"
NameTrans fn="assign-name" from="/jmx-console(/*)" name="jknsapi"
</Object>
```

2. 各パスを提供するワーカーを定義します。

**IPLANET\_CONFIG/obj.conf** ファイルの編集を続行します。編集したセクションの終了タグのすぐ後に、</Object> を追加します。

```
<Object name="jknsapi">
ObjectType fn=force-type type=text/plain
Service fn="jk_service" worker="worker01" path="/status"
Service fn="jk_service" worker="worker02" path="/nc(/*)"
Service fn="jk_service" worker="worker01"
</Object>
```

上記の例は、URL パス `/status` へのリクエストを `worker01` という名前のワーカーにリダイレクトし、`/nc/` 以下のすべての URL パスを `worker02` という名前のワーカーにリダイレクトします。3 番目の行は、前の行で一致しない `jknsapi` オブジェクトに割り当てられたすべての URL が `worker01` に提供されることを示しています。

ファイルを保存し、終了します。

3. ワーカーとその属性を定義します。

**IPLANET\_CONFIG/connectors/** ディレクトリーに **workers.properties** というファイルを作成します。以下の内容をそのファイルに貼り付けし、お使いの環境に合わせて変更します。

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

# Entries that define the host and port associated with these workers
worker.worker01.host=127.0.0.1
worker.worker01.port=8009
worker.worker01.type=ajp13

worker.worker02.host=127.0.0.100
worker.worker02.port=8009
worker.worker02.type=ajp13
```

**workers.properties** ファイルは Apache `mod_jk` と同じ構文を使用します。

ファイルを保存し、終了します。

4. iPlanet Web Server の再起動

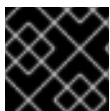
以下のコマンドを実行し、iPlanet Web Server を再起動します。

```
IPLANET_CONFIG/./bin/stopserv
IPLANET_CONFIG/./bin/startserv
```

iPlanet Web Server が、JBoss EAP のデプロイメントに設定した URL へクライアントリクエストを送信します。

### 24.10.3. NSAPI コネクターがクライアントリクエストを複数の JBoss EAP サーバーで分散するよう設定

このタスクは、負荷分散の設定でクライアントリクエストを JBoss EAP サーバーに送信するよう NSAPI コネクターを設定します。



#### 重要

このタスクを続行するには、[NSAPI コネクターが設定](#)されている必要があります。

ロードバランシングのコネクターの設定

1. JBoss EAP サーバーにリダイレクトする URL パスを定義します。



#### 注記

**IPLANET\_CONFIG/obj.conf** では、前の行から継続する行以外は、行の最初にスペースを挿入しないでください。

**IPLANET\_CONFIG/obj.conf** ファイルを編集します。<Object name="default"> で始まるセクションを見つけ、一致する各 URL パターンを次のサンプルファイルで示された形式で追加します。文字列 **jknsapi** は、次の手順で定義される HTTP コネクターを示します。例は、ワイルドカードを使用したパターン一致を示しています。

```
<Object name="default">
[...]
NameTrans fn="assign-name" from="/status" name="jknsapi"
NameTrans fn="assign-name" from="/images(/*)" name="jknsapi"
NameTrans fn="assign-name" from="/css(/*)" name="jknsapi"
NameTrans fn="assign-name" from="/nc(/*)" name="jknsapi"
NameTrans fn="assign-name" from="/jmx-console(/*)" name="jknsapi"
NameTrans fn="assign-name" from="/jkmanager/*" name="jknsapi"
</Object>
```

2. 各パスを提供するワーカーを定義します。

**IPLANET\_CONFIG/obj.conf** ファイルの編集を続行します。前の手順で変更したセクションの終了タグ (</Object>) のすぐ後に、以下の新しいセクションを追加し、必要に応じて変更します。

```
<Object name="jknsapi">
ObjectType fn=force-type type=text/plain
Service fn="jk_service" worker="status" path="/jkmanager(/*)"
Service fn="jk_service" worker="router"
</Object>
```

この **jknsapi** オブジェクトは、**default** オブジェクトの **name="jknsapi"** マッピングにマップされた各パスを提供するために使用されるワーカーノードを定義します。**/jkmanager/\*** に一致する URL 以外のすべてが、**router** という名前のワーカーにリダイレクトされます。

3. ワーカーとその属性を定義します。

**workers.properties** という名前のファイルを **IPLANET\_CONFIG/connector/** で作成します。以下の内容をそのファイルに貼り付けし、お使いの環境に合わせて変更します。

```
# The advanced router LB worker
# A list of each worker
worker.list=router,status

# First JBoss EAP server
# (worker node) definition.
# Port 8009 is the standard port for AJP
#

worker.worker01.port=8009
worker.worker01.host=127.0.0.1
worker.worker01.type=ajp13
worker.worker01.ping_mode=A
worker.worker01.socket_timeout=10
worker.worker01.lbfactor=3

# Second JBoss EAP server
worker.worker02.port=8009
worker.worker02.host=127.0.0.100
worker.worker02.type=ajp13
worker.worker02.ping_mode=A
```

```
worker.worker02.socket_timeout=10
worker.worker02.lbfactor=1

# Define the load-balancer called "router"
worker.router.type=lb
worker.router.balance_workers=worker01,worker02

# Define the status worker
worker.status.type=status
```

**workers.properties** ファイルは Apache mod\_jk と同じ構文を使用します。

ファイルを保存し、終了します。

4. iPlanet Web Server 7.0 を再起動します。

```
IPLANET_CONFIG/./bin/stopserv
IPLANET_CONFIG/./bin/startserv
```

iPlanet Web Server は、設定した URL パターンを負荷分散設定の JBoss EAP サーバーにリダイレクトします。

## 第25章 ECLIPSE MICROPROFILE

### 25.1. ECLIPSE MICROPROFILE CONFIG を使用した設定の管理



#### 重要

Eclipse MicroProfile Config は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

#### 25.1.1. MicroProfile Config SmallRye サブシステム

設定データは動的に変更でき、アプリケーションはサーバーを再起動せずに最新の設定情報にアクセスできる必要があります。[Eclipse MicroProfile Config](#) は設定データのポータブルな外部化を実現します。これにより、変更や再パッケージ化を必要とせずに、アプリケーションとマイクロサービスを複数の環境で実行するよう設定することができます。Eclipse MicroProfile Config 機能は、[SmallRye Config](#) を使用して JBoss EAP に実装され、**microprofile-config-smallrye** サブシステムによって提供されます。このサブシステムはデフォルトの JBoss EAP 7.2 設定に含まれています。

#### 25.1.2. MicroProfile Config SmallRye サブシステムによってサポートされる ConfigSources

形式や元の場所が異なることがある MicroProfile Config 設定プロパティは、[org.eclipse.microprofile.config.spi.ConfigSource](#) インターフェースの実装である ConfigSources によって提供されます。

MicroProfile Config 仕様は、設定値を取得するために、以下のデフォルト **ConfigSource** 実装を提供します。

- **System.getProperties()** の使用。
- **System.getenv()** の使用。
- クラスパス上のすべての **META-INF/microprofile-config.properties** から。

**microprofile-config-smallrye** サブシステムは、設定値を取得するために **ConfigSource** リソースの以下の追加タイプをサポートします。

- **プロパティ** から。
- **ディレクトリーのファイル** から。
- **ConfigSource** クラス から。
- **ConfigSourceProvider** クラス から。





## 注記

以下のセクションでは、管理 CLI を使用して **microprofile-config-smallrye** サブシステムを設定しますが、管理コンソールを使用することもできます。管理コンソールで **Configuration** → **Subsystems** → **MicroProfile Config** と選択し、**表示** をクリックすると、設定できます。

MicroProfile Config SmallRye サブシステムの設定に使用できる属性のリストは「[MicroProfile Config SmallRye サブシステムの属性](#)」を参照してください。

プロパティからの **ConfigSource** 設定の取得

**ConfigSource** のアクセス用にプロパティを直接保存するには、**config-source** 属性を追加し、**properties** をリストとして指定します。

以下の管理 CLI コマンドは **property1** および **property2** プロパティの設定データが含まれる **ConfigSource** を作成します。

```
/subsystem=microprofile-config-smallrye/config-source=props:add(properties={property1=value1,property2=value2})
```

このコマンドを実行すると、**microprofile-config-smallrye** サブシステムに以下の XML 設定が指定されます。

```
<subsystem xmlns="urn:wildfly:microprofile-config-smallrye:1.0">
  <config-source name="props">
    <property name="property1" value="value1"/>
    <property name="property2" value="value2"/>
  </config-source>
</subsystem>
```

ディレクトリーからの **ConfigSource** 設定の取得

**ConfigSource** を作成してディレクトリーのファイルからプロパティを読み取りするには、**config-source** 属性を追加して、ファイルが含まれるディレクトリーを指定します。**dir** ディレクトリーの各ファイルの名前はプロパティの名前で、ファイルの内容はプロパティの値になります。

以下の CLI コマンドは、**/etc/config/numbers-app/** ディレクトリーのファイルから設定プロパティを読み取る **ConfigSource** を作成します。

```
/subsystem=microprofile-config-smallrye/config-source=file-props:add(dir={path=/etc/config/numbers-app})
```

このコマンドを実行すると、**microprofile-config-smallrye** サブシステムに以下の XML 設定が指定されます。

```
<subsystem xmlns="urn:wildfly:microprofile-config-smallrye:1.0">
  <config-source name="file-props">
    <dir path="/etc/config/numbers-app"/>
  </config-source>
</subsystem>
```

この構造は、OpenShift の **ConfigMaps** によって使用される構造に対応します。**dir** の値は、OpenShift または Kubernetes の **ConfigMap** 定義にある **mountPath** に対応します。

JBoss EAP にデプロイされたすべてのアプリケーションは、ディレクトリーに保存されたプロパティーにプログラムを使用してアクセスできます。

`/etc/config/numbers-app/` ディレクトリーに以下の 2 つのファイルが含まれていることを仮定します。

- **num.size**: このファイルには **5** が値として含まれています。
- **num.max**: このファイルには **100** が値として含まれています。

以下のコード例では、**num.size** は **5** を返し、**num.max** は **100** を返します。

```
@Inject
@ConfigProperty(name = "num.size")
int numSize;

@Inject
@ConfigProperty(name = "num.max")
int numMax;
```

**ConfigSource** クラスからの **ConfigSource** 設定の取得

カスタムの [org.eclipse.microprofile.config.spi.ConfigSource](#) 実装クラスを作成および設定して、設定値のソースを提供することができます。

以下の管理 CLI コマンドは、**org.example** という名前の JBoss Module によって提供される、**org.example.MyConfigSource** という名前の実装クラスの **ConfigSource** を作成します。

```
/subsystem=microprofile-config-smallrye/config-source=my-config-source:add(class={name=org.example.MyConfigSource, module=org.example})
```

このコマンドを実行すると、**microprofile-config-smallrye** サブシステムに以下の XML 設定が指定されます。

```
<subsystem xmlns="urn:wildfly:microprofile-config-smallrye:1.0">
  <config-source name="my-config-source">
    <class name="org.example.MyConfigSource" module="org.example"/>
  </config-source>
</subsystem>
```

この **ConfigSource** クラスによって提供されるプロパティーはすべての JBoss EAP デプロイメントで使用できます。

JBoss EAP サーバーにグローバルモジュールを追加する方法は「[グローバルモジュールの定義](#)」を参照してください。

**ConfigSourceProvider** クラスからの **ConfigSource** 設定の取得

複数の **ConfigSource** インスタンスの実装を登録する、カスタムの

**org.eclipse.microprofile.config.spi.ConfigSourceProvider** 実装クラスを作成および設定できます。

以下の管理 CLI コマンドは、**org.example** という名前の JBoss Module によって提供される、**org.example.MyConfigSourceProvider** という名前の実装クラスの **config-source-provider** を作成します。

```
/subsystem=microprofile-config-smallrye/config-source-provider=my-config-source-provider:add(class={name=org.example.MyConfigSourceProvider, module=org.example})
```

このコマンドを実行すると、**microprofile-config-smallrye** サブシステムに以下の XML 設定が指定されます。

```
<subsystem xmlns="urn:wildfly:microprofile-config-smallrye:1.0">
  <config-source-provider name="my-config-source-provider">
    <class name="org.example.MyConfigSourceProvider" module="org.example"/>
  </config-source-provider>
</subsystem>
```

**ConfigSourceProvider** 実装によって提供されるプロパティはすべての JBoss EAP デプロイメントで使用できます。

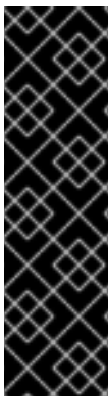
JBoss EAP サーバーにグローバルモジュールを追加する方法は「[グローバルモジュールの定義](#)」を参照してください。

### 25.1.3. ConfigSources にアクセスするアプリケーションのデプロイ

Java コードで MicroProfile Config にアクセスするアプリケーションは、CDI を有効にする必要があります。これには、デプロイメントアーカイブに **META-INF/beans.xml** または **WEB-INF/beans.xml** ファイルが含まれるようにするか、CDI bean アノテーションが含まれるようにします。

CDI の詳細は、JBoss EAP『[開発ガイド](#)』の「コンテキストおよび依存関係の挿入 (CDI)」を参照してください。

## 25.2. MICROPROFILE OPENTRACING SMALLRYE サブシステムでのリクエストのトレース



### 重要

Eclipse Microprofile OpenTracing は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

### 25.2.1. Eclipse MicroProfile OpenTracing

サービス境界全体でリクエストをトレースする機能は、ライフサイクル中にリクエストが複数のサービスを通るマイクロサービス環境で特に重要となります。[Eclipse Microprofile OpenTracing](#) 仕様は、JAX-RS アプリケーション内の Open Tracing 対応の **Tracer** オブジェクトにアクセスするための、動作および API を定義します。動作は、送受信リクエストに対してどのように Open Tracing Spans が自動的に作成されるかを指定します。API は、指定のエンドポイントのトレースをどのように明示的に無効または有効にするかを定義します。

### 25.2.2. MicroProfile OpenTracing SmallRye サブシステム

Eclipse MicroProfile Open Tracing 機能は、[SmallRye OpenTracing](#) コンポーネントを使用して JBoss EAP に実装され、**microprofile-opentracing-smallrye** サブシステムによって提供されます。このサブシステムは、JAX-RS エンドポイントへのリクエストのトレースと CDI bean のサポートが含まれる

Microprofile 1.1 を実装し、デフォルトの JBoss EAP 7.2 設定に含まれています。

**microprofile-opentracing-smallrye** サブシステムには、[Jaeger Java Client](#) がデフォルトのトレーサーとして含まれ、さらに JAX-RS や CDI などの Java EE アプリケーションで一般的に使用されるコンポーネントのインスツルメンテーションライブラリーのセットも含まれます。JBoss EAP サーバーに自動的にデプロイされた各 WAR は、独自の **Tracer** インスタンスを持ちます。EAR 内の各 WAR は個別の WAR として扱われ、各 WAR には独自の **Tracer** インスタンスがあります。デフォルトでは、Jaeger Client と使用されるサービス名はデプロイメントの名前から派生し、通常これは WAR ファイル名になります。

### 25.2.3. MicroProfile OpenTracing SmallRye サブシステムの設定

**microprofile-opentracing-smallrye** サブシステムは、デフォルトの JBoss EAP 7.2 設定に含まれています。有効な OpenTracing にはメモリーまたはパフォーマンスコストが関係するため、このサブシステムを無効にしたい場合があります。

以下の管理 CLI コマンドを使用してサーバー設定からサブシステムを削除し、サーバーインスタンスに対して MicroProfile Open Tracing 機能をグローバルに無効にします。

1. サブシステムを削除します。

```
/subsystem=microprofile-opentracing-smallrye:remove()
```

2. 変更を反映するためにサーバーをリロードします。

```
reload
```

以下の管理 CLI コマンドを使用してサーバー設定にサブシステムを追加し、サーバーインスタンスに対して MicroProfile Open Tracing 機能をグローバルに有効にします。

1. サブシステムを追加します。

```
/subsystem=microprofile-opentracing-smallrye:add()
```

2. 変更を反映するためにサーバーをリロードします。

```
reload
```

**microprofile-opentracing-smallrye** サブシステムに使用できる別の設定オプションはありません。代わりに、システムプロパティまたは環境変数を設定して Jaeger Client を設定します。Jaeger Client の設定方法に関する詳細は、[Jaeger のドキュメント](#) を参照してください。有効なシステムプロパティのリストは Jaeger ドキュメントの「[Configuration via Environment](#)」を参照してください。

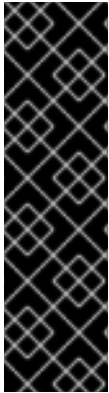
#### 重要

この機能はテクノロジープレビューとして提供されるため、システムプロパティや環境変数を使用した Jaeger Java Client トレーサーの設定に関する現在の設定オプションは特に今後のリリースで互換性のない状態で変更される可能性があります。

また、デフォルトでは Java 用の Jaeger Client に **0.001** に設定されている確率的サンプリングストラテジーがあり、約 1000 トレースに 1 つの割合でのみサンプルが取られることに注意してください。すべてのリクエストのサンプルを取るには、システムプロパティ **JAEGER\_SAMPLER\_TYPE** を **const** に設定し、**JAEGER\_SAMPLER\_PARAM** を **1** に設定します。

デフォルトトレーサーをオーバーライドする方法と、CDI bean のトレース方法に関する詳細は、『[開発ガイド](#)』の「[Eclipse MicroProfile OpenTracing を使用したリクエストのトレース](#)」を参照してください。

## 25.3. ECLIPSE MICROPROFILE HEALTH を使用したサーバー状態の監視



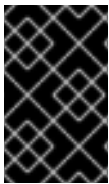
### 重要

Eclipse MicroProfile Health は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

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

### 25.3.1. MicroProfile Health SmallRye サブシステム

JBoss EAP には、[microprofile-health-smallrye](#) サブシステムを使用して [Eclipse MicroProfile Health](#) 機能を提供する **SmallRye Health** コンポーネントが含まれています。このサブシステムは、JBoss EAP インスタンスが想定どおりに応答しているかどうかを判断するために使用され、デフォルトで有効になっています。



### 重要

デフォルトでは、MicroProfile Health SmallRye サブシステムはサーバーが稼働していることのみを確認します。カスタムヘルスチェックを含める場合は、『[開発ガイド](#)』の「[カスタムヘルスチェックの実装](#)」を参照してください。

### 25.3.2. 管理 CLI を使用したヘルスチェックの確認

ヘルスチェックは管理 CLI を使用して確認できます。以下のコマンドは、サーバーの現在の状態を取得します。

```
/subsystem=microprofile-health-smallrye:check
{
  "outcome" => "success",
  "result" => {
    "outcome" => "UP",
    "checks" => []
  }
}
```

### 25.3.3. HTTP エンドポイントを使用したヘルスチェックの確認

ヘルスチェックは自動的に JBoss EAP の **health** コンテキストにデプロイされます。そのため、管理 CLI を使用した確認だけでなく、HTTP エンドポイントを使用して現在のヘルス状態を取得することもできます。管理インターフェースからアクセスできる **/health** エンドポイントのデフォルトアドレスは <http://127.0.0.1:9990/health> です。

HTTP エンドポイントを使用して、サーバーの現在のヘルス状態を取得するには、以下の URL を使用します。

```
http://HOST:PORT/health
```

このコンテキストにアクセスすると、サーバーの状態を示すヘルスチェックが JSON 形式で表示されます。

### HTTP エンドポイントの認証の有効化

**health** コンテキストを設定すると、このコンテキストへのアクセスにユーザーの認証を必要とすることができます。以下の手順は、このエンドポイントで認証を有効にします。

1. **microprofile-health-smallrye** サブシステムで **security-enabled** 属性を **true** に設定します。

```
/subsystem=microprofile-health-smallrye:write-attribute(name=security-enabled,value=true)
```

2. 変更を反映するためにサーバーをリロードします。

```
reload
```

**health** エンドポイントにアクセスしようとする時、認証プロンプトが表示されるようになります。

## 付録A リファレンス資料

### A.1. サーバーランタイム引数

アプリケーションサーバーの起動スクリプトは実行時に引数とスイッチを受け入れます。そのため、**standalone.xml**、**domain.xml**、および **host.xml** 設定ファイルに定義されていない他の設定でサーバーを起動できます。

他の設定には、ソケットバインディングの代替セットを持つサーバーの起動や2次設定が含まれていることがあります。

help スイッチ **-h** または **--help** を起動時に渡すと、利用可能なパラメーターのリストを使用できます。

表A.1 ランタイムスイッチおよび引数

引数またはスイッチ	操作モード	説明
<code>--admin-only</code>	Standalone	サーバーの実行タイプを <b>ADMIN_ONLY</b> に設定します。これにより管理インターフェースが開かれ、管理リクエストが許可されますが、他のランタイムサービスは起動されず、エンドユーザーのリクエストは許可されません。この代わりに <b>--start-mode=admin-only</b> を使用することが推奨されます。
<code>--admin-only</code>	Domain	ホストコントローラーの実行タイプを <b>ADMIN_ONLY</b> に設定します。これにより管理インターフェースが開かれ、管理リクエストが許可されますが、サーバーは起動しません。ホストコントローラーがドメインのマスターである場合はスレーブホストコントローラーからの受信接続が許可されます。
<code>-b=&lt;value&gt;</code> 、 <code>-b &lt;value&gt;</code>	Standalone、 Domain	パブリックインターフェースのバインドアドレスを設定するために使用される <b>jboss.bind.address</b> システムプロパティを設定します。値の指定がない場合は、デフォルトで <b>127.0.0.1</b> が指定されます。他のインターフェースにバインドアドレスを設定するには <b>-b&lt;interface&gt;=&lt;value&gt;</b> エントリーを確認します。
<code>-b&lt;interface&gt;=&lt;value&gt;</code>	Standalone、 Domain	システムプロパティ <b>jboss.bind.address.&lt;interface&gt;</b> を指定の値に設定します。 <b>-bmanagement=IP_ADDRESS</b> など。
<code>--backup</code>	Domain	このホストがドメインコントローラーではない場合でも永続ドメイン設定のコピーを保持します。
<code>-c=&lt;config&gt;</code> 、 <code>-c &lt;config&gt;</code>	Standalone	使用するサーバー設定ファイルの名前。デフォルトは <b>standalone.xml</b> です。

引数またはスイッチ	操作モード	説明
-c<config>、 -c <config>	Domain	使用するサーバー設定ファイルの名前。デフォルトは <b>domain.xml</b> です。
--cached-dc	Domain	ホストがドメインコントローラーではなく、起動時にドメインコントローラーに接続できない場合、ローカルでキャッシュされたドメイン設定のコピーを使用してブートします。
--debug [<port>]	Standalone	オプションの引数を用いてデバッグモードを有効にし、ポートを指定します。起動スクリプトがサポートする場合のみ動作します。
-D<name>[=<value>]	Standalone、 Domain	システムプロパティを設定します。
--domain-config=<config>	Domain	使用するサーバー設定ファイルの名前。デフォルトは <b>domain.xml</b> です。
--git-repo	Standalone	サーバー設定データの管理および永続化に使用される Git リポジトリの場所。これは、ローカルで保存する場合は <b>local</b> を指定し、リモートの場合はリモートリポジトリへの URL を指定します。
--git-branch	Standalone	使用する Git リポジトリのブランチまたはタグ名。ブランチまたはタグ名が存在しないと作成されないため、この引数には既存のブランチまたはタグ名を指定する必要があります。タグ名を使用する場合、リポジトリを detached HEAD 状態にし、今後のコミットがブランチにアタッチされないようにします。タグ名は読み取り専用で、通常複数のノード全体で設定をレプリケートする必要があるときに使用されます。
--git-auth	Standalone	Elytron 設定ファイルへの URL には、リモート Git リポジトリへの接続時に使用される認証情報が含まれています。この引数は、Git リポジトリに認証が必要な場合に必要となります。Git は SSH 認証をサポートしますが、Elytron はサポートしないため、プライベートキーを使用してパスワードのないデフォルトの SSH 認証のみがサポートされます。この引数は <b>local</b> リポジトリとは使用されません。
-h、 --help	Standalone、 Domain	ヘルプメッセージを表示し、終了します。



引数またはスイッチ	操作モード	説明
--host-config=<config>	Domain	使用するホスト設定ファイルの名前。デフォルトは <b>host.xml</b> です。
--interprocess-hc-address=<address>	Domain	ホストコントローラーがプロセスコントローラーからの通信をリッスンしなければならないアドレス。
--interprocess-hc-port=<port>	Domain	ホストコントローラーがプロセスコントローラーからの通信をリッスンしなければならないポート。
--master-address=<address>	Domain	システムプロパティー <b>jboss.domain.master.address</b> を指定の値に設定します。デフォルトのスレーブホストコントローラー設定では、マスターホストコントローラーのアドレスを設定するために使用されます。
--master-port=<port>	Domain	システムプロパティー <b>jboss.domain.master.port</b> を指定の値に設定します。デフォルトのスレーブホストコントローラー設定では、マスターホストコントローラーによるネイティブ管理の通信で使用されるポートを設定するために使用されます。
--read-only-server-config=<config>	Standalone	使用するサーバー設定ファイルの名前。元のファイルは上書きされないため、 <b>--server-config</b> および <b>-c</b> とは異なります。
--read-only-domain-config=<config>	Domain	使用するドメイン設定ファイルの名前。最初のファイルは上書きされないため、 <b>--domain-config</b> および <b>-c</b> とは異なります。
--read-only-host-config=<config>	Domain	使用するホスト設定ファイルの名前。最初のファイルは上書きされないため、 <b>--host-config</b> とは異なります。
-P=<url>、-P <url>、--properties=<url>	Standalone、Domain	該当する URL からシステムプロパティーをロードします。
--pc-address=<address>	Domain	プロセスコントローラーが制御するプロセスからの通信をリッスンするアドレス。
--pc-port=<port>	Domain	プロセスコントローラーが制御するプロセスからの通信をリッスンするポート。
-S<name>[=<value>]	Standalone	セキュリティープロパティーを設定します。

引数またはスイッチ	操作モード	説明
-secmgr	Standalone、Domain	セキュリティーマネージャーがインストールされた状態でサーバーを実行します。
--server-config=<config>	Standalone	使用するサーバー設定ファイルの名前。デフォルトは <b>standalone.xml</b> です。
--start-mode=<mode>	Standalone	サーバーの起動モードを設定します。このオプションは、 <b>--admin-only</b> と併用できません。有効な値は以下のとおりです。 <ul style="list-style-type: none"> <li>● <b>normal</b>: サーバーは通常どおりに起動します。</li> <li>● <b>admin-only</b>: サーバーは管理インターフェースのみを開き、管理リクエストを許可しますが、他のランタイムサービスは起動せず、エンドユーザーのリクエストを許可しません。</li> <li>● <b>suspend</b>: サーバーは中断モードで起動され、再開するまでリクエストに対処しません。</li> </ul>
-u=<value>、-u <value>	Standalone、Domain	設定ファイルの socket-binding 要素のマルチキャストアドレスを設定するために使用される <b>jboss.default.multicast.address</b> システムプロパティを設定します。値の指定がない場合はデフォルトで <b>230.0.0.4</b> が指定されます。
-v、-V、--version	Standalone、Domain	アプリケーションサーバーのバージョンを表示し、終了します。

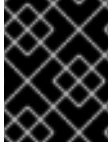


### 警告

JBoss EAP に同梱される設定ファイルは、スイッチ (**-b**、**-u** など) を処理するように設定されます。スイッチによって制御されるシステムプロパティを使用しないよう設定ファイルを変更した場合は、実行するコマンドにスイッチを追加しても効果はありません。

## A.2. RPM サービス設定ファイル

JBoss EAP の RPM インストールには、ZIP またはインストーラーインストールよりも 2 つ多い設定ファイルが含まれています。これらのファイルは、JBoss EAP の起動環境を指定するために、サービス初期化スクリプトによって使用されます。これらのサービス設定ファイルの場所は、Red Hat Enterprise Linux 6 と Red Hat Enterprise Linux 7 以降では異なります。



## 重要

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

表A.2 Red Hat Enterprise Linux 6 の RPM 設定ファイル

File	説明
/etc/sysconfig/eap7-standalone	Red Hat Enterprise Linux 6 のスタンドアロン JBoss EAP サーバーに固有する設定
/etc/sysconfig/eap7-domain	Red Hat Enterprise Linux 6 で管理対象ドメインとして実行されている JBoss EAP に固有する設定

表A.3 Red Hat Enterprise Linux 7 以降の RPM 設定ファイル

File	説明
/etc/opt/rh/eap7/wildfly/eap7-standalone.conf	Red Hat Enterprise Linux 7 以降のスタンドアロン JBoss EAP サーバーに固有する設定
/etc/opt/rh/eap7/wildfly/eap7-domain.conf	Red Hat Enterprise Linux 7 以降で管理対象ドメインとして実行されている JBoss EAP に固有する設定

## A.3. RPM サービス設定プロパティ

以下の表は、JBoss EAP RPM サービスで使用できる設定プロパティと、そのデフォルト値のリストになります。



## 注記

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

表A.4 RPM サービス設定プロパティ

プロパティ	説明
JAVA_HOME	Java Runtime Environment がインストールされたディレクトリー。 デフォルト値: <b>/usr/lib/jvm/jre</b>
JAVAPATH	Java 実行可能ファイルがインストールされたパス。 デフォルト値: <b>\$JAVA_HOME/bin</b>

プロパティ	説明
WILDFLY_STARTUP_WAIT	start または restart コマンドを受け取った後にサーバーが正常に起動されたことを確認するまで、初期化スクリプトが待機する秒数。このプロパティは、Red Hat Enterprise Linux 6 のみに適用されます。  デフォルト値: <b>60</b>
WILDFLY_SHUTDOWN_WAIT	stop または restart コマンドの受信時、続行する前に初期化スクリプトがサーバーのシャットダウンを待機する秒数。このプロパティは、Red Hat Enterprise Linux 6 のみに適用されます。  デフォルト値: <b>20</b>
WILDFLY_CONSOLE_LOG	CONSOLE ログハンドラーがリダイレクトされるファイル。  デフォルト値: スタンドアロンサーバーの場合は <b>/var/opt/rh/eap7/log/wildfly/standalone/console.log</b> 、管理対象ドメインの場合は <b>/var/opt/rh/eap7/log/wildfly/domain/console.log</b>
WILDFLY_SH	JBoss EAP サーバーを起動するために使用されるスクリプト。  デフォルト値: スタンドアロンサーバーの場合は <b>/opt/rh/eap7/root/usr/share/wildfly/bin/standalone.sh</b> 、管理対象ドメインの場合は <b>/opt/rh/eap7/root/usr/share/wildfly/bin/domain.sh</b>
WILDFLY_SERVER_CONFIG	使用するサーバー設定ファイル。  このプロパティにはデフォルト値がありません。開始時に <b>standalone.xml</b> または <b>domain.xml</b> を定義できます。
WILDFLY_HOST_CONFIG	管理対象ドメインでは、このプロパティによってユーザーはホスト設定ファイル ( <b>host.xml</b> など) を指定できます。デフォルトとして設定されている値はありません。
WILDFLY_MODULEPATH	JBoss EAP モジュールディレクトリーのパス。  デフォルト値: <b>/opt/rh/eap7/root/usr/share/wildfly/modules</b>
WILDFLY_BIND	パブリックインターフェースのバインドアドレスを設定するために使用される <b>jboss.bind.address</b> システムプロパティを設定します。値の指定がない場合、 <b>0.0.0.0</b> がデフォルトとして使用されます。
WILDFLY_OPTS	起動時に含む追加の引数。例を以下に示します。  -Dorg.wildfly.openssl.path=PATH_TO_OPENSSL_LIBS

## A.4. JBOSS EAP サブシステムの概要

以下の表は、JBoss EAP のサブシステムを簡単に説明します。

表A.5 JBoss EAP サブシステム

JBoss EAP サブシステム	説明
batch-jberet	バッチアプリケーションを実行する環境を設定し、バッチジョブを管理します。
bean-validation	Java オブジェクトデータを検証するために <a href="#">bean バリデーション</a> を設定します。
core-management	<a href="#">サーバーライフサイクルイベント</a> のリスナーを登録し、 <a href="#">設定の変更</a> を追跡します。
datasources	<a href="#">データソース</a> を作成および設定し、 <a href="#">JDBC データベースドライバー</a> を管理します。
deployment-scanner	アプリケーションがデプロイする特定の場所を監視するために <a href="#">デプロイメントスキャナー</a> を設定します。
ee	<a href="#">グローバルモジュール</a> の定義、 <a href="#">記述子ベースのプロパティ置換</a> の有効化、および <a href="#">デフォルトバインディング</a> の設定など、Java EE プラットフォームで一般的な機能を設定します。
ejb3	セッション Bean やメッセージ駆動型 Bean を含むエンタープライズ JavaBean (EJB) を設定します。  <b>ejb3</b> サブシステムの詳細は、JBoss EAP の『 <a href="#">Developing EJB Applications</a> 』を参照してください。
elytron	サーバーおよびアプリケーションのセキュリティを設定します。  <b>elytron</b> サブシステムの詳細は JBoss EAP の『 <a href="#">セキュリティアーキテクチャー</a> 』を参照してください。
iiop-openjdk	JTS トランザクションの CORBA (Common Object Request Broker Architecture) サービスおよびセキュリティを含むその他の <a href="#">ORB サービス</a> を設定します。JBoss EAP 6 ではこの機能は <b>jacorb</b> サブシステムに含まれていました。
infinispan	JBoss EAP の高可用性サービスの <a href="#">キャッシング</a> 機能を設定します。
io	他のサブシステムによって使用される <a href="#">ワーカー</a> および <a href="#">バッファプール</a> を定義します。
jaxrs	JAX-RS アプリケーションのデプロイメントおよび機能を有効にします。
jca	<a href="#">JCA (Java EE Connector Architecture)</a> コンテナーおよびリソースアダプターデプロイメントの一般設定を行います。

JBoss EAP サブシステム	説明
jdr	トラブルシューティングに役立つ診断データの収集を有効にします。JBoss EAP のサブスクライバーはサポートをリクエストするときにこの情報を Red Hat に提供できます。
jgroups	クラスターのサーバーがお互いに対話するためのプロトコルスタックと <a href="#">通信メカニズム</a> を設定します。
jmx	リモート JMX (Java Management Extensions) のアクセスを設定します。
jpa	JPA (Java Persistence API) 2.1 のコンテナ管理の要件を管理し、永続ユニットの定義、アノテーション、および記述子のデプロイを可能にします。  <b>jpa</b> サブシステムの詳細は JBoss EAP の『 <a href="#">開発ガイド</a> 』を参照してください。
jsf	JSF (JavaServer Faces) 実装を管理します。
jsr77	<a href="#">JSR-77</a> 仕様によって定義された Java EE 管理機能を提供します。
logging	<a href="#">ログカテゴリー</a> および <a href="#">ログハンドラー</a> を介してシステムおよびアプリケーションレベルのロギングを設定します。
mail	<a href="#">メールサーバーの属性</a> と <a href="#">カスタムメールトランスポート</a> を設定して、JBoss EAP へデプロイされたアプリケーションがメールを送信できるメールサービスを作成します。
messaging-activemq	統合メッセージングプロバイダーである Artemis の JMS 宛先、接続ファクトリー、およびその他の設定を設定します。JBoss EAP 6 では、メッセージング機能は <b>messaging</b> サブシステムに含まれていました。  <b>messaging-activemq</b> サブシステムの詳細は、JBoss EAP の『 <a href="#">Configuring Messaging</a> 』を参照してください。

JBoss EAP サブシステム	説明
microprofile-config-smallrye	<p><a href="#">MicroProfile Config SmallRye</a> を使用して、設定データのポータブルな外部化を実現し、サーバーを再起動せずにアプリケーションが最新の設定プロパティにアクセスできるようにします。</p> <div data-bbox="555 371 660 810" style="background-color: black; width: 66px; height: 196px; margin-bottom: 10px;"></div> <p><b>重要</b></p> <p>Eclipse MicroProfile Config は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。</p> <p>テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータル<a href="#">の「テクノロジープレビュー機能のサポート範囲」</a>を参照してください。</p>
microprofile-health-smallrye	<p><a href="#">SmallRye Health</a> を使用してサーバーのヘルス状態を監視します。<a href="#">microprofile-health-smallrye</a> サブシステムに関する詳細やその設定方法は、本書の「<b>Eclipse MicroProfile Health</b> を使用したサーバー状態の監視」を参照してください。カスタムヘルスチェックの作成に関する詳細は『<a href="#">開発ガイド</a>』の「<b>Eclipse MicroProfile Health</b> を使用したサーバー状態の監視」を参照してください。</p> <div data-bbox="555 1151 660 1590" style="background-color: black; width: 66px; height: 196px; margin-bottom: 10px;"></div> <p><b>重要</b></p> <p>Eclipse MicroProfile Health は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。</p> <p>テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータル<a href="#">の「テクノロジープレビュー機能のサポート範囲」</a>を参照してください。</p>

JBoss EAP サブシステム	説明
microprofile-opentracing-smallrye	<p><a href="#">SmallRye OpenTracing</a> を使用して、サービス境界全体でリクエストをトレースします。<a href="#">microprofile-opentracing-smallrye</a> サブシステムに関する詳細と設定方法については、「<b>MicroProfile OpenTracing SmallRye</b> サブシステムでのリクエストのトレース」を参照してください。CDI bean と JAX-RS エンドポイントのトレースをカスタマイズする方法に関しては『<a href="#">開発ガイド</a>』の「<a href="#">Eclipse MicroProfile OpenTracing を使用したリクエストのトレース</a>」を参照してください。</p> <div data-bbox="555 510 662 952" style="background-color: black; width: 67px; height: 197px; margin-bottom: 10px;"></div> <p><b>重要</b></p> <p>Eclipse Microprofile OpenTracing は、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。</p> <p>テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの「<a href="#">テクノロジープレビュー機能のサポート範囲</a>」を参照してください。</p>
modcluster	サーバー側の <a href="#">mod_cluster ワーカーノード</a> を設定します。
naming	エントリーをグローバル JNDI 名前空間にバインドし、リモート JNDI インターフェイスを設定します。
picketlink-federation	PicketLink SAML ベースのシングルサインオン (SSO) を設定します。 <b>picketlink-federation</b> サブシステムの詳細は JBoss EAP の『 <a href="#">How To Set Up SSO with SAML v2</a> 』を参照してください。
picketlink-identity-management	PicketLink アイデンティティ管理サービスを設定します。このサブシステムはサポート対象外です。
pojo	過去のバージョンの JBoss EAP でサポートされたように、JBoss Microcontainer サービスが含まれるアプリケーションのデプロイメントを有効にします。
remoting	ローカルおよび <a href="#">リモートサービス</a> のインバウンドおよびアウトバウンド接続を設定します。
request-controller	<a href="#">サーバーを正常に一時停止およびシャットダウン</a> するよう設定します。
resource-adapters	JCA (Java Connector Architecture) 仕様を使用して Java EE アプリケーションおよび EIS (Enterprise Information System) 間の通信を行うために <a href="#">リソースアダプター</a> を設定および維持します。



JBoss EAP サブシステム	説明
rts	REST-AT のサポート対象外の実装。
sar	過去のバージョンの JBoss EAP でサポートされたように、MBean サービスが含まれる SAR アーカイブのデプロイメントを有効にします。
security	アプリケーションのセキュリティーを設定するレガシーな方法。  <b>security</b> サブシステムの詳細は、JBoss EAP の『 <a href="#">セキュリティーアーキテクチャー</a> 』を参照してください。
security-manager	Java Security Manager によって使用される Java セキュリティーポリシーを設定します。  <b>security-manager</b> サブシステムの詳細は JBoss EAP の『 <a href="#">How to Configure Server Security</a> 』を参照してください。
singleton	シングルトンポリシーを定義して、シングルトンデプロイメントの動作を設定したり、シングルトン MSC サービスを作成したりします。  <b>singleton</b> サブシステムの詳細は JBoss EAP の『 <a href="#">開発ガイド</a> 』を参照してください。
transactions	タイムアウト値やトランザクションロギングなどのトランザクションマネージャー (TM) のオプションと、JTS (Java Transaction Service) を使用するかどうかを設定します。  <b>transactions</b> サブシステムの詳細は、『 <a href="#">JBoss EAP でのトランザクションの管理</a> 』を参照してください。
undertow	JBoss EAP の <a href="#">web サーバー</a> およびサーブレットコンテナを設定します。JBoss EAP 6 では、この機能は <b>web</b> サブシステムに含まれていました。
webservices	パブリッシュされたエンドポイントアドレスおよびエンドポイントハンドラーチェーンを設定します。また、Web サービスプロバイダーのホスト名、ポート、および WSDL アドレスも設定します。  <b>webservices</b> サブシステムの詳細は JBoss EAP の『 <a href="#">Developing Web Services Applications</a> 』を参照してください。
weld	JBoss EAP の CDI (Contexts and Dependency Injection) を設定します。
xts	トランザクションの Web サービスの調整を設定します。

## A.5. ADD-USER ユーティリティー引数

以下の表は、**add-user.sh** または **add-user.bat** スクリプトで使用できる引数を示しています。これらのスクリプトは既定の認証のプロパティファイルに新しいユーザーを追加するためのユーティリティーです。

表A.6 add-user コマンド引数

コマンドライン引数	説明
-a	アプリケーションレルムでユーザーを作成します。省略した場合、デフォルトでは管理レルムでユーザーが作成されます。
-dc <value>	プロパティファイルが含まれるドメイン設定ディレクトリー。省略した場合、デフォルトのディレクトリーは <b>EAP_HOME/domain/configuration/</b> になります。
-sc <value>	プロパティファイルが含まれる代替のスタンドアロンサーバー設定ディレクトリー。省略した場合、デフォルトのディレクトリーは <b>EAP_HOME/standalone/configuration/</b> になります。
-up、--user-properties <value>	代替のユーザープロパティファイルの名前。絶対パスを使用でき、代替の設定ディレクトリーを指定する <b>-sc</b> または <b>-dc</b> 引数と共に使用されるファイル名を使用することもできます。
-g、--group <value>	このユーザーに割り当てるグループのコンマ区切りリスト。
-gp、--group-properties <value>	代替のグループプロパティファイルの名前。絶対パスを使用でき、代替の設定ディレクトリーを指定する <b>-sc</b> または <b>-dc</b> 引数と共に使用されるファイル名を使用することもできます。
-p、--password <value>	ユーザーのパスワード。
-u、--user <value>	ユーザーの名前。ユーザー名には、以下の文字のみを使用できます。文字の数と順番は自由です。 <ul style="list-style-type: none"> <li>● 英数字 (a-z、A-Z、0-9)</li> <li>● ダッシュ (-)、ピリオド (.)、コンマ (,)、アットマーク (@)</li> <li>● バックスラッシュ (\)</li> <li>● 等号 (=)</li> </ul>
-r、--realm <value>	管理インターフェースをセキュアにするために使用されるレルムの名前。省略した場合、デフォルト値は <b>ManagementRealm</b> です。
-s、--silent	コンソールへ出力せずに <b>add-user</b> スクリプトを実行します。
-e、--enable	ユーザーを有効にします。
-d、--disable	ユーザーを無効にします。
-cw、--confirm-warning	対話モードで自動的に警告を確認します。
-h、--help	<b>add-user</b> スクリプトの使用情報を表示します。

コマンドライン引数	説明
-ds、 --display-secret	非対話モードで秘密の値を出力します。

## A.6. 管理監査ロギング属性



### 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-config\_5\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.7 管理監査ロギング: ロガー属性

属性	説明
enabled	監査ロギングが有効になっているかどうか。
log-boot	操作がサーバーの起動時にログに記録されるかどうか。
log-read-only	設定を変更しない操作またはランタイムサービスがログに記録されるかどうか。

表A.8 管理監査ロギング: ログフォーマッター属性

属性	説明
compact	<b>true</b> の場合、JSON を1行でフォーマットします。新しい行が含まれる値が存在する可能性があるため、レコード全体を1行にするのが重要な場合は、 <b>escape-new-line</b> または <b>escape-control-characters</b> を <b>true</b> に設定します。
date-format	<b>java.text.SimpleDateFormat</b> が理解するように使用するデータ形式。 <b>include-date</b> が <b>false</b> に設定されている場合は無視されます。
date-separator	日付と他のフォーマットされたログメッセージのセパレーター。 <b>include-date</b> が <b>false</b> に設定されている場合は無視されます。
escape-control-characters	<b>true</b> の場合、すべての制御文字 (10 進値が <b>32</b> を超える ASCII エントリー) を 8 進の ASCII コードでエスケープします。たとえば、新しい行は <b>#012</b> になります。 <b>true</b> の場合、 <b>escape-new-line=false</b> をオーバーライドします。
escape-new-line	<b>true</b> の場合、新しい行を 8 進の ASCII コードでエスケープします ( <b>#012</b> )。
include-date	フォーマットされたログレコードに日付が含まれるかどうか。

表A.9 管理監査ロギング: ファイルハンドラー属性

属性	説明
disabled-due-to-failure	ロギングの失敗によりこのハンドラーが無効になったかどうか (読み取り専用)。
failure-count	ハンドラーが初期化された後に発生したロギング失敗数 (読み取り専用)。
formatter	ログメッセージのフォーマットに使用される JSON フォーマッター。
max-failure-count	このハンドラーを無効化する前の最大ロギング失敗数。
path	監査ログファイルのパス。
relative-to	以前指定された別のパスの名前、またはシステムによって提供される標準的なパスの1つ。 <b>relative-to</b> が指定された場合、 <b>path</b> 属性の値は、この属性によって指定されたパスへの相対値としてみなされます。
rotate-at-startup	サーバーの起動時に古いログファイルをローテーションするかどうか。

表A.10 管理監査ロギング: Syslog ハンドラー属性

属性	説明
app-name	<b>RFC-5424</b> のセクション 6.2.5 で定義された syslog レコードに追加するアプリケーション名。指定されない場合、デフォルト値は製品の名前になります。
disabled-due-to-failure	ロギングの失敗によりこのハンドラーが無効になったかどうか (読み取り専用)。
facility	<b>RFC-5424</b> のセクション 6.2.1 と <b>RFC-3164</b> のセクション 4.1.1 で定義された syslog ロギングに使用する機能。
failure-count	ハンドラーが初期化された後に発生したロギング失敗数 (読み取り専用)。
formatter	ログメッセージのフォーマットに使用される JSON フォーマッター。
max-failure-count	このハンドラーを無効化する前の最大ロギング失敗数。
max-length	許可される、ヘッダーを含むログメッセージの最大長 (バイト単位)。未定義の場合、デフォルト値は <b>1024</b> バイト ( <b>syslog-format</b> が <b>RFC3164</b> の場合) または <b>2048</b> バイト ( <b>syslog-format</b> が <b>RFC5424</b> の場合) になります。

属性	説明
protocol	syslog ハンドラーに使用するプロトコル。 <b>udp</b> 、 <b>tcp</b> 、または <b>tls</b> のいずれか1つである必要があります。
syslog-format	syslog 形式: <b>RFC5424</b> または <b>RFC3164</b> .
truncate	ヘッダーを含むメッセージの長さ (バイト単位) が <b>max-length</b> 属性の値よりも大きい場合、そのメッセージを省略するかどうか。 <b>false</b> に設定すると、メッセージが分割され、同じヘッダー値で送信されます。



### 注記

syslog サーバーごとに実装が異なるため、すべての設定をすべての syslog サーバーに適用できるとは限りません。テストは **rsyslog** syslog 実装を使用して実行されています。

次の表には、高度な属性のみがリストされています。各属性は設定パラメーターを持ち、一部の属性は個設定パラメーターを持ちます。

## A.7. インターフェース属性



### 注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-config\_5\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.11 インターフェース属性と値

インターフェース要素	説明
any	インターフェースの選択基準の一部は、最低でも基準のネストされたセットの1つ (すべてとは限らない) を満たす必要があることを示す要素。
any-address	このインターフェースを使用するソケットをワイルドカードアドレスにバインドする必要があることを示す空の要素。 <b>java.net.preferIPv4Stack</b> システムプロパティーが true に設定されていない限り、IPv6 ワイルドカードアドレス (:::) が使用されます。 true に設定された場合は、IPv4 ワイルドカードアドレス ( <b>0.0.0.0</b> ) が使用されます。ソケットがデュアルスタックマシンの IPv6 anylocal アドレスにバインドされた場合は、IPv6 および IPv4 トラフィックを受け入れることができます。 IPv4 (IPv4 マッピング) anylocal アドレスにバインドされた場合は、IPv4 トラフィックのみを受け入れることができます。
inet-address	IPv6 または IPv4 のドット区切り表記の IP アドレス、または IP アドレスに解決できるホスト名。

インターフェース要素	説明
link-local-address	インターフェースの選択基準の一部として、関連付けられたアドレスがリンクローカルであるかどうかを示す空の要素。
loopback	インターフェースの選択基準の一部として、ループバックインターフェースであるかどうかを示す空の要素。
loopback-address	マシンのループバックインターフェースで実際には設定できないループバックアドレス。IP アドレスが関連付けられた NIC が見つからない場合であっても該当する値が使用されるため、inet-address タイプとは異なります。
multicast	インターフェースの選択基準の一部として、マルチキャストをサポートするかどうかを示す空の要素。
name	インターフェースの名前。
nic	ネットワークインターフェースの名前 (eth0、eth1、lo など)。
nic-match	使用できるインターフェースを見つけるために、マシンで利用可能なネットワークインターフェースの名前を検索する正規表現。
not	インターフェースの選択基準の一部は、基準のネストされたセットを満たしてはならないことを示す要素。
point-to-point	インターフェースの選択基準の一部として、ポイントツーポイントインターフェースであるかどうかを示す空の要素。
public-address	インターフェースの選択基準の一部として、公開されたルーティング可能なアドレスを持つかどうかを示す空の要素。
site-local-address	インターフェースの選択基準の一部として、関連付けられたアドレスがサイトローカルであるかどうかを示す空の要素。
subnet-match	スラッシュ表記法で記述されたネットワーク IP アドレスとアドレスのネットワーク接頭辞のビット数 (例: <b>192.168.0.0/16</b> )。
up	インターフェースの選択基準の一部として、現在稼動しているかどうかを示す空の要素。
virtual	インターフェースの選択基準の一部として、仮想インターフェースであるかどうかを示す空の要素。

## A.8. ソケットバインディング属性



## 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-config\_5\_0.xsd** のスキーマ定義ファイルで確認してください。

以下の表は、3種類のソケットバインディングそれぞれに設定できる属性を表しています。

- [socket-binding](#)
- [remote-destination-outbound-socket-binding](#)
- [local-destination-outbound-socket-binding](#)

表A.12 インバウンドソケットバインディング (**socket-binding**) の属性

属性	説明
client-mappings	このソケットバインディングのクライアントマッピングを指定します。このソケットへ接続するクライアントは、希望のアウトバウンドインターフェースと一致するマッピングに指定された宛先アドレスを使用する必要があります。これにより、ネットワークアドレスの変換を使用する高度なネットワークポロジーマたは複数のネットワークインターフェースにバインディングを持つ高度なネットワークポロジーマが機能します。各マッピングは宣言された順序で評価される必要があります。最初に一致したマッピングを使用して宛先が決定されます。
fixed-port	ソケットグループの他のソケットに数値のオフセットが適用された場合でもポートの値を固定したままにするかどうか。
interface	ソケットがバインドされる必要があるインターネットの名前、またはマルチキャストソケットの場合はリッスンするインターフェース。宣言されたインターフェースの1つである必要があります。定義されないと、エンクロージングソケットバインディンググループからの <b>default-interface</b> の値が使用されます。
multicast-address	ソケットがマルチキャストトラフィックを受信するマルチキャストアドレス。指定しないと、ソケットがマルチキャストを受信するよう設定されません。
multicast-port	ソケットがマルチキャストトラフィックを受信するポート。 <b>multicast-address</b> が設定されている場合はこれも設定する必要があります。
name	ソケットの名前。ソケット設定情報にアクセスする必要があるサービスは、この名前を使用してソケット設定情報を探します。必須の属性です。
port	ソケットがバインドされる必要があるポートの番号。サーバーによってポートオフセットが適用され、ポートの値がすべて増加または減少される場合、この値は上書きされることに注意してください。

表A.13 リモートアウトバウンドソケットバインディング (**remote-destination-outbound-socket-binding**) の属性

属性	説明
fixed-source-port	数値のオフセットポートがソケットグループの別のアウトバウンドソケットに適用される場合、ポートの値を固定すべきかどうか。
host	このアウトバウンドソケットが接続するリモート宛先のホスト名または IP アドレス。
port	アウトバウンドソケットが接続すべきリモート宛先のポート番号。
source-interface	アウトバウンドソケットのソースアドレスに使用されるインターフェースの名前。
source-port	アウトバウンドソケットのソースポートとして使用されるポート番号。

表A.14 ローカルアウトバウンドソケットバインディング (**local-destination-outbound-socket-binding**) の属性

属性	説明
fixed-source-port	数値のオフセットポートがソケットグループの別のアウトバウンドソケットに適用される場合、ポートの値を固定すべきかどうか。
socket-binding-ref	このアウトバウンドソケットが接続するポートを決定するために使用されるローカルソケットバインディングの名前。
source-interface	アウトバウンドソケットのソースアドレスに使用されるインターフェースの名前。
source-port	アウトバウンドソケットのソースポートとして使用されるポート番号。

## A.9. デフォルトのソケットバインディング

以下の表は、各ソケットバインディンググループのデフォルトのソケットバインディングを示しています。

- [standard-sockets](#)
- [ha-sockets](#)
- [full-sockets](#)
- [full-ha-sockets](#)
- [load-balancer-sockets](#)

表A.15 **standard-sockets**



ソケットバインディング	ポート	説明
ajp	8009	Apache JServ プロトコル。HTTP クラスターリングおよび負荷分散に使用します。
http	8080	デプロイされた Web アプリケーションのデフォルトポート。
https	8443	デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。
management-http	9990	管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993	管理レイヤーを用いた HTTPS 通信に使用されます。
txn-recovery-environment	4712	JTA トランザクションリカバリーマネージャー。
txn-status-manager	4713	JTA / JTS トランザクションマネージャー。

表A.16 ha-sockets

ソケットバインディング	ポート	マルチキャストポート	説明
ajp	8009		Apache JServ プロトコル。HTTP クラスターリングおよび負荷分散に使用します。
http	8080		デプロイされた Web アプリケーションのデフォルトポート。
https	8443		デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。
jgroups-mping		45700	マルチキャスト。HA クラスターでの初期メンバーシップの検出に使用されます。
jgroups-tcp	7600		TCP を使用した、HA クラスター内でのユニキャストピア検出。
jgroups-udp	55200	45688	UDP を使用した、HA クラスター内でのマルチキャストピア検出。
management-http	9990		管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993		管理レイヤーを用いた HTTPS 通信に使用されます。

ソケットバインディング	ポート	マルチキャストポート	説明
modcluster		23364	JBoss EAP と HTTP ロードバランサー間の通信に対するマルチキャストポート。
txn-recovery-environment	4712		JTA トランザクションリカバリーマネージャー。
txn-status-manager	4713		JTA / JTS トランザクションマネージャー。

表A.17 full-sockets

ソケットバインディング	ポート	説明
ajp	8009	Apache JServ プロトコル。HTTP クラスタリングおよび負荷分散に使用します。
http	8080	デプロイされた Web アプリケーションのデフォルトポート。
https	8443	デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。
iiop	3528	JTS トランザクションおよび他の ORB 依存サービス用の CORBA サービス。
iiop-ssl	3529	SSL 暗号化 CORBA サービス。
management-http	9990	管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993	管理レイヤーを用いた HTTPS 通信に使用されます。
txn-recovery-environment	4712	JTA トランザクションリカバリーマネージャー。
txn-status-manager	4713	JTA / JTS トランザクションマネージャー。

表A.18 full-ha-sockets

名前	ポート	マルチキャストポート	説明
ajp	8009		Apache JServ プロトコル。HTTP クラスタリングおよび負荷分散に使用します。

名前	ポート	マルチキャストポート	説明
http	8080		デプロイされた Web アプリケーションのデフォルトポート。
https	8443		デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。
iiop	3528		JTS トランザクションおよび他の ORB 依存サービス用の CORBA サービス。
iiop-ssl	3529		SSL 暗号化 CORBA サービス。
jgroups-mping		45700	マルチキャスト。HA クラスターでの初期メンバーシップの検出に使用されます。
jgroups-tcp	7600		TCP を使用した、HA クラスター内でのユニキャストピア検出。
jgroups-udp	55200	45688	UDP を使用した、HA クラスター内でのマルチキャストピア検出。
management-http	9990		管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993		管理レイヤーを用いた HTTPS 通信に使用されます。
modcluster		23364	JBoss EAP と HTTP ロードバランサー間の通信に対するマルチキャストポート。
txn-recovery-environment	4712		JTA トランザクションリカバリーマネージャー。
txn-status-manager	4713		JTA / JTS トランザクションマネージャー。

表A.19 load-balancer-sockets

名前	ポート	マルチキャストポート	説明
http	8080		デプロイされた Web アプリケーションのデフォルトポート。
https	8443		デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。

名前	ポート	マルチキャストポート	説明
management-http	9990		管理レイヤーを用いた HTTP 通信に使用されます。
management-https	9993		管理レイヤーを用いた HTTPS 通信に使用されます。
mcmp-management	8090		ライフサイクルイベントを送信する MCMP (Mod-Cluster Management Protocol) 接続のポート。
modcluster		23364	JBoss EAP と HTTP ロードバランサー間の通信に対するマルチキャストポート。

## A.10. モジュールコマンド引数

以下の引数は、**module add** 管理 CLI コマンドに渡すことができます。

表A.20 モジュールコマンド引数

引数	説明
--absolute-resources	この引数を使用して、 <b>module.xml</b> ファイルから参照するファイルシステムの絶対パスのリストを指定します。指定されるファイルはモジュールディレクトリーにはコピーされません。  区切り文字の詳細は <b>--resource-delimiter</b> を参照してください。
--allow-nonexistent-resources	この引数を使用して、 <b>--resources</b> によって指定された存在しないリソースの空のディレクトリーを作成します。存在しないリソースがある場合、この引数を使用しないと <b>module add</b> コマンドの実行に失敗します。
--dependencies	この引数を使用して、このモジュールが依存するモジュール名のカンマ区切りリストを提供します。
--export-dependencies	この引数を使用して、エクスポートされた依存関係を指定します。  <pre>module add --name=com.mysql --resources=/path/to/mysql-connector-java-8.0.12.jar --export-dependencies=javafx.api,javafx.transaction.api</pre>
--main-class	この引数を使用して、モジュールのメインメソッドを宣言する完全修飾クラス名を指定します。

引数	説明
--module-root-dir	<p>デフォルトの <b>EAP_HOME/modules/</b> ディレクトリーの代わりに、<b>外部の JBoss EAP モジュールディレクトリーを定義した場合は、この引数を使用します。</b></p> <pre>module add --module-root-dir=<b>path/to</b>/my-external-modules/ --name=com.mysql --resources=<b>path/to</b>/mysql-connector- java-8.0.12.jar --dependencies=javax.api,javax.transaction.api</pre>
--module-xml	<p>この引数を使用して、この新規モジュールに使用する <b>module.xml</b> へのファイルシステムパスを提供します。ます。この引数を指定しないと、<b>module.xml</b> ファイルがモジュールディレクトリーに生成されます。</p>
--name	<p>この引数を使用して、追加するモジュールの名前を提供します。この引数は必須です。</p>
--properties	<p>この引数を使用して、モジュールプロパティーを定義する <b>PROPERTY_NAME=PROPERTY_VALUE</b> ペアのカンマ区切りリストを提供します。</p>
--resource-delimiter	<p>この引数を使用して、<b>--resources</b> または <b>absolute-resources</b> 引数に提供されたリソースリストの、ユーザー定義のファイルパス区切り文字を設定します。設定されていない場合、ファイルパスの区切り文字は Linux ではコロン (:)、Windows ではセミコロン (;) になります。</p>
--resources	<p>この引数を使用して、ファイルシステムパスのリストを提供してこのモジュールのリソースを指定します。ファイルは直接このモジュールディレクトリーにコピーされ、その <b>module.xml</b> ファイルから参照されます。ディレクトリーへのパスを提供した場合、ディレクトリーとその内容はモジュールディレクトリーにコピーされます。シンボリックリンクは保持されず、リンクされたリソースはモジュールディレクトリーにコピーされます。<b>--absolute-resources</b> または <b>--module-xml</b> が提供されている場合を除き、この引数は必須です。</p> <p>区切り文字の詳細は <b>--resource-delimiter</b> を参照してください。</p>
--slot	<p>この引数を使用して、デフォルトの <b>main</b> スロット以外のスロットにモジュールを追加します。</p> <pre>module add --name=com.mysql --slot=8.0 -- resources=<b>path/to</b>/mysql-connector-java-8.0.12.jar -- dependencies=javax.api,javax.transaction.api</pre>

## A.11. デプロイメントスキャナーマーカーファイル

マーカーファイルは、JBoss EAP サーバーインスタンスのデプロイメントディレクトリー内でアプリケーションの状態をマーク付けするためにデプロイメントスキャナーによって使用されます。マーカーファイルの名前はデプロイメントの名前と同じで、ファイル接尾辞はアプリケーションのデプロイメン

トの状態を示します。

たとえば、**test-application.war** のデプロイメントには **test-application.war.deployed** という名前のマーカーファイルがあります。

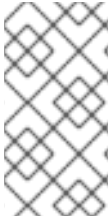
以下の表は、使用できるマーカーファイルタイプとそれらの意味を表しています。

表A.21 マーカーファイルのタイプ

ファイル名接尾辞	生成	説明
.deployed	システムによる生成	コンテンツがデプロイされたことを示します。このファイルが削除された場合、コンテンツはアンデプロイされます。
.dodeploy	ユーザーによる生成	コンテンツをデプロイまたは再デプロイする必要があることを意味します。
.failed	システムによる生成	デプロイメントの失敗を示します。マーカーファイルには、失敗の原因に関する情報が含まれます。マーカーファイルが削除された場合、コンテンツは再度自動デプロイの対象になります。
.isdeploying	システムによる生成	デプロイ中であることを示します。デプロイの完了後、このマーカーファイルは削除されます。
.isundeploying	システムによる生成	<b>.deployed</b> ファイルの削除によってトリガーされ、コンテンツのアンデプロイ中であることを示します。デプロイの完了後、このマーカーファイルは削除されます。
.pending	システムによる生成	デプロイメントスキャナーはコンテンツをデプロイする必要性を認識しているにもかかわらず、現在問題によって自動デプロイメントが実行されないことを意味します (例: コンテンツのコピー中である場合)。このマーカーはグローバルデプロイメントロードブロックとして機能するため、スキャナーはこのマーカーファイルが存在する間、あらゆるコンテンツのデプロイおよびアンデプロイをサーバーに指示しません。
.skipdeploy	ユーザーによる生成	アプリケーションの自動デプロイを無効にします。展開形式のコンテンツの自動デプロイメントを一時的にブロックする方法として役に立ち、不完全なコンテンツの編集がプッシュされないようにします。スキャナーは zip 形式のコンテンツに対する処理中の変更を検出し、完了するまで待機しますが、zip 形式のコンテンツとともに使用できます。
.undeployed	システムによる生成	コンテンツがアンデプロイされたことを示します。このマーカーファイルを削除しても、コンテンツの再デプロイメントには影響ありません。

## A.12. デプロイメントスキャナーの属性

デプロイメントスキャナーには設定可能な以下の属性が含まれます。



## 注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/jboss-as-deployment-scanner\_2\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.22 デプロイメントスキャナーの属性

名前	デフォルト	説明
auto-deploy-exploded	false	<b>.dodeploy</b> マーカーファイルなしで、展開形式のコンテンツの自動デプロイメントを許可します。基本的な開発シナリオに対してのみ推奨され、開発者またはオペレーティングシステムによる変更中に、展開形式のアプリケーションデプロイメントが行われないようにします。
auto-deploy-xml	true	<b>.dodeploy</b> マーカーファイルなしでの XML コンテンツの自動デプロイメントを許可します。
auto-deploy-zipped	true	<b>.dodeploy</b> マーカーファイルなしでの zip 形式のコンテンツの自動デプロイメントを許可します。
deployment-timeout	600	デプロイメントスキャナーでデプロイメントをキャンセルするまでのデプロイメント試行許可時間 (秒単位)。
path	deployments	スキャンされる実際のファイルシステムパス。 <b>relative-to</b> 属性が指定されない限り、絶対パスとして処理され、値はそのパスの相対パスとして処理されます。
relative-to	jboss.server.base.dir	サーバー設定の <b>パス</b> として定義されるファイルシステムパスへの参照。
runtime-failure-causes-rollback	false	デプロイメントのランタイム障害が原因で、スキャン操作の一部としてそのデプロイメントやその他すべてのデプロイメント (関係しないデプロイメントの可能性あり) のロールバックが発生するかどうか。
scan-enabled	true	<b>scan-interval</b> により起動時にアプリケーションの自動スキャンを許可します。
scan-interval	5000	変更に対してレポジトリがスキャンされる間隔 (ミリ秒単位)。1 未満の値を設定すると、初期設定時のみスキャンが行われます。

## A.13. 管理対象ドメインの JVM 設定属性

以下の JVM 設定オプションは、ホスト、サーバーグループ、またはサーバーレベルで管理対象ドメインに設定できます。一部の属性の有効な値は JVM によって異なることに注意してください。詳細は、JDK ベンダーのドキュメントを参照してください。



## 注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-config\_5\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.23 JVM 設定属性

属性	説明
agent-lib	Java エージェントライブラリーを指定する <b>-agentlib java</b> オプションの値を設定します。
agent-path	Java エージェントパスを指定する <b>-agentpath java</b> オプションの値を設定します。
debug-enabled	デバッグを有効にするかどうか。この属性は、サーバーレベルの JVM 設定のみに適用されます。
debug-options	デバッグが有効な場合に JVM オプションを指定します。この属性は、サーバーレベルの JVM 設定のみに適用されます。
env-classpath-ignored	<b>CLASSPATH</b> 環境変数を無視するかどうか。
environment-variables	キーバリューペアの環境変数を指定します。
heap-size	JVM によって割り当てられる初期のヒープサイズを指定する <b>-Xms</b> オプションの値を設定します。
java-agent	Java エージェントを指定する <b>-javaagent java</b> オプションの値を設定します。
java-home	<b>JAVA_HOME</b> 変数の値を設定します。
jvm-options	必要な追加の JVM オプションを指定します。
launch-command	サーバープロセスの起動に使用される <b>java</b> コマンドの前に付けるオペレーティングシステムレベルのコマンドを指定します。たとえば、 <b>sudo</b> コマンドを使用して、別のユーザーとして Java プロセスを実行できます。
max-heap-size	JVM によって割り当てられる最大ヒープサイズを指定する <b>-Xmx</b> オプションの値を設定します。
max-permgen-size	永久生成の最大サイズを設定します。非推奨: JVM は個別の永久生成領域を提供しないようになりました。



属性	説明
permgen-size	永久生成の初期サイズを設定します。 <b>非推奨</b> : JVM は個別の永久生成領域を提供しないようになりました。
stack-size	JVM スタックサイズを指定する <b>-Xss</b> オプションの値を設定します。
type	使用中の JVM を提供したベンダーを指定します。 <b>ORACLE</b> 、 <b>IBM</b> 、 <b>SUN</b> および <b>OTHER</b> を指定できます。

## A.14. MAIL サブシステムの属性

以下の表は、メールセッション と以下のメールサーバータイプ用の **mail** サブシステムの属性を表しています。

- [imap](#)
- [pop3](#)
- [smtp](#)
- [custom](#)



### 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-mail\_3\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.24 メールセッションの属性

属性	説明
debug	JavaMail のデバッグを有効にするかどうか。
from	送信時に送信元が設定されていない場合のデフォルトの送信元 (from) アドレス。
jndi-name	メールセッションのバインド先の JNDI 名。

表A.25 IMAP メールサーバーの属性

属性	説明
credential-reference	サーバー上で認証される、認証情報ストアからの認証情報。
outbound-socket-binding-ref	メールサーバーのアウトバウンドソケットバインディングへの参照。

属性	説明
password	サーバー上で認証するパスワード。
ssl	サーバーが SSL を必要とするかどうか。
tls	サーバーに TLS が必要であるかどうか。
username	サーバー上で認証するユーザー名。

表A.26 POP3 メールサーバーの属性

属性	説明
credential-reference	サーバー上で認証される、認証情報ストアからの認証情報。
outbound-socket-binding-ref	メールサーバーのアウトバウンドソケットバインディングへの参照。
password	サーバー上で認証するパスワード。
ssl	サーバーが SSL を必要とするかどうか。
tls	サーバーに TLS が必要であるかどうか。
username	サーバー上で認証するユーザー名。

表A.27 SMTP メールサーバーの属性

属性	説明
credential-reference	サーバー上で認証される、認証情報ストアからの認証情報。
outbound-socket-binding-ref	メールサーバーのアウトバウンドソケットバインディングへの参照。
password	サーバー上で認証するパスワード。
ssl	サーバーが SSL を必要とするかどうか。
tls	サーバーに TLS が必要であるかどうか。
username	サーバー上で認証するユーザー名。

表A.28 カスタムメールサーバーの属性

属性	説明
credential-reference	サーバー上で認証される、認証情報ストアからの認証情報。
outbound-socket-binding-ref	メールサーバーのアウトバウンドソケットバインディングへの参照。
password	サーバー上で認証するパスワード。
properties	このサーバーの JavaMail プロパティ。
ssl	サーバーが SSL を必要とするかどうか。
tls	サーバーに TLS が必要であるかどうか。
username	サーバー上で認証するユーザー名。

## A.15. ルートロガーの属性



### 注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を [EAP\\_HOME/docs/schema/jboss-as-logging\\_3\\_0.xsd](#) のスキーマ定義ファイルで確認してください。

表A.29 ルートロガーの属性

属性	説明
filter	簡単なフィルタータイプを定義します。 <b>filter-spec</b> が導入されたため非推奨になりました。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> は、パターンに一致しないログエントリを除外するフィルターを定義します。
handlers	ルートロガーによって使用されるログハンドラーの一覧。
level	ルートロガーが記録するログメッセージの最低レベル。



### 注記

ルートロガーに指定された **filter-spec** は他のハンドラーによって継承されません。ハンドラーごとに **filter-spec** を指定する必要があります。

## A.16. ログカテゴリーの属性



## 注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/jboss-as-logging\_6\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.30 ログカテゴリーの属性

属性	説明
category	ログメッセージがキャプチャーされるログカテゴリー。
filter	簡単なフィルタータイプを定義します。 <b>filter-spec</b> が導入されたため非推奨になりました。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> はパターンに一致しないフィルターを定義します。
handlers	ロガーに関連付けられたログハンドラーのリスト。
level	ログカテゴリーが記録するログメッセージの最低レベル。
use-parent-handlers	<b>true</b> に設定した場合、割り当てられた他のハンドラーだけでなく、ルートロガーのログハンドラーを使用します。

## A.17. ログハンドラーの属性



## 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/jboss-as-logging\_6\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.31 Console ログハンドラーの属性

属性	説明
autoflush	<b>true</b> に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	<b>true</b> に設定された場合、ハンドラーが有効になり、通常とおり機能します。 <b>false</b> に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。

属性	説明
filter	簡単なフィルタータイプを定義します。 <b>filter-spec</b> が導入されたため非推奨になりました。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。
name	ログハンドラーの名前。ハンドラーのアドレスに名前が含まれるため非推奨になりました。
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
target	ログハンドラーの出力が送信されるシステム出力ストリーム。以下の1つになります。 <ul style="list-style-type: none"> <li>● <b>System.err</b>: ログハンドラーの出力はシステムエラーストリームに送信されます。</li> <li>● <b>System.out</b>: ログハンドラーの出力は標準出力ストリームに送信されます。</li> <li>● <b>console</b>: ログハンドラーの出力は <b>java.io.PrintWriter</b> クラスに送信されます。</li> </ul>

表A.32 File ログハンドラーの属性

属性	説明
append	<b>true</b> に設定された場合、このハンドラーが書き込んだすべてのメッセージがファイル(すでに存在する場合)に追加されます。 <b>false</b> に設定された場合、アプリケーションサーバーが起動されるたびに、新しいファイルが作成されます。
autoflush	<b>true</b> に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	<b>true</b> に設定された場合、ハンドラーが有効になり、通常とおり機能します。 <b>false</b> に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
file	このログハンドラーの出力が書き込まれるファイルを表すオブジェクト。このオブジェクトには、 <b>relative-to</b> と <b>path</b> の2つの設定プロパティが含まれます。

属性	説明
filter	簡単なフィルタータイプを定義します。 <b>filter-spec</b> が導入されたため非推奨になりました。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。
name	ログハンドラーの名前。ハンドラーのアドレスに名前が含まれるため非推奨になりました。
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。

表A.33 Periodic ログハンドラーの属性

属性	説明
append	<b>true</b> に設定された場合、このハンドラーが書き込んだすべてのメッセージがファイル (すでに存在する場合) に追加されます。 <b>false</b> に設定された場合、アプリケーションサーバーが起動されるたびに、新しいファイルが作成されます。
autoflush	<b>true</b> に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	<b>true</b> に設定された場合、ハンドラーが有効になり、通常とおり機能します。 <b>false</b> に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
file	このログハンドラーの出力が書き込まれるファイルを表すオブジェクト。このオブジェクトには、 <b>relative-to</b> と <b>path</b> の2つの設定プロパティが含まれます。
filter	簡単なフィルタータイプを定義します。 <b>filter-spec</b> が導入されたため非推奨になりました。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。

属性	説明
name	ログハンドラーの名前。ハンドラーのアドレスに名前が含まれるため非推奨になりました。
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
suffix	この文字列はローテーションログに追加される接尾辞に含まれます。 <b>suffix</b> の書式は、ドット(.)の後に <b>SimpleDateFormat</b> クラスが解析できる日付の文字列を追加したものです。

表A.34 Size ログハンドラーの属性

属性	説明
append	<b>true</b> に設定された場合、このハンドラーが書き込んだすべてのメッセージがファイル(すでに存在する場合)に追加されます。 <b>false</b> に設定された場合、アプリケーションサーバーが起動されるたびに、新しいファイルが作成されます。
autoflush	<b>true</b> に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	<b>true</b> に設定された場合、ハンドラーが有効になり、通常とおり機能します。 <b>false</b> に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
file	このログハンドラーの出力が書き込まれるファイルを表すオブジェクト。このオブジェクトには、 <b>relative-to</b> と <b>path</b> の2つの設定プロパティーが含まれます。
filter	簡単なフィルタータイプを定義します。 <b>filter-spec</b> が導入されたため非推奨になりました。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。
max-backup-index	保持するローテーションログの最大数。この数に達すると、古いログが再使用されます。デフォルト値は <b>1</b> です。  <b>suffix</b> 属性が使用された場合は、ローテーションログファイルの接尾辞がローテーションアルゴリズムに含まれます。ログファイルがローテーションされると、名前が <b>name+suffix</b> で始まる最も古いファイルが削除され、残りのローテーションログファイルの数値接尾辞が増分され、新しくローテーションされたログファイルに数値接尾辞 <b>1</b> が提供されます。

属性	説明
name	ログハンドラーの名前。 <b>ハンドラーのアドレスに名前が含まれるため非推奨になりました。</b>
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
rotate-on-boot	<b>true</b> に設定されると、新しいログファイルがサーバーの起動時に作成されます。デフォルトは <b>false</b> です。
rotate-size	ログファイルがローテーションされる前に到達できる最大サイズです。数字に追加された単一の文字はサイズ単位を示します。バイトの場合は <b>b</b> 、キロバイトの場合は <b>k</b> 、メガバイトの場合は <b>m</b> 、ギガバイトの場合は <b>g</b> になります。たとえば、50 メガバイトの場合は、 <b>50m</b> になります。
suffix	この文字列はローテーションログに追加される接尾辞に含まれます。 <b>suffix</b> の書式は、ドット (.) の後に <b>SimpleDateFormat</b> クラスが解析できる日付の文字列を追加したものです。

表A.35 Periodic Size ログハンドラーの属性

属性	説明
append	<b>true</b> に設定された場合、このハンドラーが書き込んだすべてのメッセージがファイル (すでに存在する場合) に追加されます。 <b>false</b> に設定された場合、アプリケーションサーバーが起動されるたびに、新しいファイルが作成されます。
autoflush	<b>true</b> に設定すると、ログメッセージは受信直後にハンドラー割り当てファイルに送信されます。
enabled	<b>true</b> に設定された場合、ハンドラーが有効になり、通常とおり機能します。 <b>false</b> に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	出力に使用する文字エンコーディングスキーム。
file	このログハンドラーの出力が書き込まれるファイルを表すオブジェクト。このオブジェクトには、 <b>relative-to</b> と <b>path</b> の2つの設定プロパティが含まれます。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。



属性	説明
max-backup-index	保持するローテーションログの最大数。この数に達すると、古いログが再使用されます。デフォルト値は <b>1</b> です。  <b>suffix</b> 属性が使用された場合は、ローテーションログファイルの接尾辞がローテーションアルゴリズムに含まれます。ログファイルがローテーションされると、名前が <b>name+suffix</b> で始まる最も古いファイルが削除され、残りのローテーションログファイルの数値接尾辞が増分され、新しくローテーションされたログファイルに数値接尾辞 <b>1</b> が提供されます。
name	ログハンドラーの名前。ハンドラーのアドレスに名前が含まれるため非推奨になりました。
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
rotate-on-boot	<b>true</b> に設定されると、新しいログファイルがサーバーの起動時に作成されます。デフォルトは <b>false</b> です。
rotate-size	ログファイルがローテーションされる前に到達できる最大サイズです。数字に追加された単一の文字はサイズ単位を示します。バイトの場合は <b>b</b> 、キロバイトの場合は <b>k</b> 、メガバイトの場合は <b>m</b> 、ギガバイトの場合は <b>g</b> になります。たとえば、50 メガバイトの場合は、 <b>50m</b> になります。
suffix	この文字列はローテーションログに追加される接尾辞に含まれます。 <b>suffix</b> の書式は、ドット (.) の後に <b>SimpleDateFormat</b> クラスが解析できる日付の文字列を追加したものです。

表A.36 syslog ハンドラーの属性

属性	説明
app-name	メッセージを RFC5424 形式でフォーマットするときに使用されるアプリケーション名。デフォルトのアプリケーション名は <b>java</b> です。
enabled	<b>true</b> に設定された場合、ハンドラーが有効になり、通常とおり機能します。 <b>false</b> に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
facility	RFC-5424 および RFC-3164 によって定義される機能。
hostname	メッセージ送信元のホストの名前。たとえば、アプリケーションサーバーが実行されているホストの名前になります。
level	ログハンドラーが記録するログメッセージの最低レベル。
port	syslog サーバーがリッスンしているポート。

属性	説明
server-address	syslog サーバーのアドレス。
syslog-format	RFC 仕様にしたがってログメッセージをフォーマットします。

表A.37 Socket ログハンドラーの属性

属性	説明
autoflush	毎回書き込み後に自動的にフラッシュを行うかどうか。
block-on-reconnect	<b>true</b> に設定すると、再接続の試行時に書き込みメソッドがブロックされます。非同期ハンドラーの使用時のみ <b>true</b> に設定することが推奨されます。
enabled	<b>true</b> に設定された場合、ハンドラーが有効になり、通常とおり機能します。 <b>false</b> に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
encoding	このハンドラーによって使用される文字エンコーディング。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> はパターンに一致しないフィルターを定義します。
level	ログハンドラーが記録するログメッセージの最低レベル。
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
outbound-socket-binding-ref	ソケット接続のアウトバウンドソケットバインディングへの参照。
protocol	ソケットが通信すべきプロトコル。使用できる値は <b>TCP</b> 、 <b>UDP</b> 、および <b>SSL_TCP</b> です。
ssl-context	定義された SSL コンテキストへの参照。これは、 <b>protocol</b> が <b>SSL_TCP</b> に設定された場合のみ使用されます。

表A.38 Custom ログハンドラーの属性

属性	説明
class	使用されるロギングハンドラークラス。
enabled	<b>true</b> に設定された場合、ハンドラーが有効になり、通常とおり機能します。 <b>false</b> に設定された場合、ログメッセージの処理時にハンドラーが無視されます。

属性	説明
encoding	出力に使用する文字エンコーディングスキーム。
filter	簡単なフィルタータイプを定義します。 <b>filter-spec</b> が導入されたため非推奨になりました。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> はパターンに一致しないフィルターを定義します。
formatter	このログハンドラーで使用するログフォーマッター。
level	ログハンドラーが記録するログメッセージの最低レベル。
module	ロギングハンドラーが依存するモジュール。
name	ログハンドラーの名前。ハンドラーのアドレスに名前が含まれるため非推奨になりました。
named-formatter	ハンドラーで使用する定義されたフォーマッターの名前。
properties	ロギングハンドラーに使用されるプロパティ。

表A.39 Async ログハンドラーの属性

属性	説明
enabled	<b>true</b> に設定された場合、ハンドラーが有効になり、通常とおり機能します。 <b>false</b> に設定された場合、ログメッセージの処理時にハンドラーが無視されます。
filter	簡単なフィルタータイプを定義します。 <b>filter-spec</b> が導入されたため非推奨になりました。
filter-spec	フィルターを定義する式の値。式 <b>not(match("WFLY.*"))</b> はパターンに一致しないフィルターを定義します。
level	ログハンドラーが記録するログメッセージの最低レベル。
name	ログハンドラーの名前。ハンドラーのアドレスに名前が含まれるため非推奨になりました。
overflow-action	キューの長さを超えたときにこのハンドラーがどのように応答するかを示します。これは <b>BLOCK</b> または <b>DISCARD</b> に設定できます。 <b>BLOCK</b> により、キューでスペースが利用可能になるまでロギングアプリケーションが待機します。これは、非同期でないログハンドラーと同じ動作です。 <b>DISCARD</b> により、ロギングアプリケーションは動作し続けますが、ログメッセージは削除されません。

属性	説明
queue-length	サブハンドラーが応答するときに、このハンドラーが保持するログメッセージの最大数。
subhandlers	この async ハンドラーがログメッセージを渡すログハンドラーの一覧。

## A.18. ログフォーマッター属性

表A.40 パターンフォーマッターのフォーマット文字

記号	説明
%c	ロギングイベントのカテゴリ。
%p	ログエントリのレベル (INFO、DEBUG など)
%P	ログエントリのローカライズレベル。
%d	現在の日付/時間 ( <b>yyyy-MM-dd HH:mm:ss,SSS</b> 形式)。
%r	相対時間 (ログ初期化以降のミリ秒単位の時間)。
%z	日付 (%d) の前に指定する必要があるタイムゾーン。例: <b>%z{GMT}%d{HH:mm:ss,SSS}</b>
%k	ログリソースキー (ログメッセージのローカリゼーションに使用)。
%m	ログメッセージ (例外トレースを含む)。
%s	単純なログメッセージ (例外トレースなし)。
%e	例外スタックトレース (拡張モジュール情報なし)。
%E	例外スタックトレース (拡張モジュール情報あり)。
%t	現在のスレッドの名前。
%n	改行文字。
%C	ログメソッドを呼び出すコードのクラス (低速)。
%F	ログメソッドを呼び出すクラスのファイル名 (低速)。
%l	ログメソッドを呼び出すコードのソースロケーション (低速)。
%L	ログメソッドを呼び出すコードの行番号 (低速)。

記号	説明
%M	ログメソッドを呼び出すコードのメソッド (低速)。
%x	ネスト化診断コンテキスト。
%X	メッセージ診断コンテキスト。
%%	リテラルパーセント (%) 記号 (エスケープ)。

表A.41 JSON ログフォーマッター属性

属性	説明
date-format	日付と時刻の形式パターン。有効な <b>java.time.format.DateTimeFormatter.ofPattern()</b> パターンである必要があります。デフォルトのパターンはオフセットのある ISO-8601 拡張形式です。
exception-output-type	<p>ある場合はログに記録されたメッセージの原因が JSON 出力に追加される方法を示します。使用できる値は次のとおりです。</p> <ul style="list-style-type: none"> <li>● <b>detailed</b></li> <li>● <b>formatted</b></li> <li>● <b>detailed-and-formatted</b></li> </ul>
key-overrides	JSON プロパティのキー名のオーバーライドを許可します。
meta-data	JSON フォーマッターで使用されるメタデータを設定します。
pretty-print	フォーマット時にプリティープリントを使用すべきかどうか。
print-details	<p>詳細を出力するかどうか。詳細には、ソースクラス名、ソースファイル名、ソースメソッド名、ソースモジュール名、ソースモジュールバージョン、およびソース行番号が含まれます。</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 2;"> <p><b>注記</b></p> <p>値は呼び出し元から取得されるため、詳細の出力はリソースを多く使用する可能性があります。</p> </div> </div>
record-delimiter	レコードの最後を示すために使用される値。null を設定すると、レコードの最後に区切り文字が使用されません。デフォルト値は改行です。
zone-id	日付と時刻をフォーマットするためのゾーン ID です。定義のない場合はシステムのデフォルトが使用されます。

表A.42 XML ログフォーマッター属性

属性	説明
date-format	日付と時刻の形式パターン。有効な <b>java.time.format.DateTimeFormatter.ofPattern()</b> パターンである必要があります。デフォルトのパターンはオフセットのある ISO-8601 拡張形式です。
exception-output-type	ある場合はログに記録されたメッセージの原因が XML 出力に追加される方法を示します。使用できる値は次のとおりです。 <ul style="list-style-type: none"> <li>● <b>detailed</b></li> <li>● <b>formatted</b></li> <li>● <b>detailed-and-formatted</b></li> </ul>
key-overrides	XML プロパティのキー名のオーバーライドを許可します。
meta-data	XML 形式で使用されるメタデータを設定します。プロパティは各ログメッセージに追加されます。
namespace-uri	print-namespace 属性が true の場合、各レコードに使用される名前空間 URI を設定します。定義されている namespace-uri がなく、オーバーライドされたキーがある場合、 <b>print-namespace</b> 属性が true に設定されている場合でも書き込まれるネームスペースはありません。
pretty-print	フォーマット時にプリティープリントを使用すべきかどうか。
print-details	<p>詳細を出力するかどうか。詳細には、ソースクラス名、ソースファイル名、ソースメソッド名、ソースモジュール名、ソースモジュールバージョン、およびソース行番号が含まれます。</p> <p> <b>注記</b></p> <p>値は呼び出し元から取得されるため、詳細の出力はリソースを多く使用する可能性があります。</p>
record-delimiter	レコードの最後を示すために使用される値。null の場合、レコードの最後に区切り文字が使用されません。デフォルト値は改行です。
zone-id	日付と時刻をフォーマットするためのゾーン ID です。定義のない場合はシステムのデフォルトが使用されます。

## A.19. データソース接続 URL

表A.43 データソース接続 URL

データソース	接続 URL
IBM DB2	jdbc:db2://SERVER_NAME:PORT/DATABASE_NAME
MariaDB	jdbc:mariadb://SERVER_NAME:PORT/DATABASE_NAME
MariaDB Galera クラスター	jdbc:mariadb://SERVER_NAME:PORT,SERVER_NAME:PORT/DATABASE_NAME
Microsoft SQL Server	jdbc:sqlserver://SERVER_NAME:PORT;DatabaseName=DATABASE_NAME
MySQL	jdbc:mysql://SERVER_NAME:PORT/DATABASE_NAME
Oracle	jdbc:oracle:thin:@SERVER_NAME:PORT:ORACLE_SID
PostgreSQL	jdbc:postgresql://SERVER_NAME:PORT/DATABASE_NAME
Sybase	jdbc:sybase:Tds:SERVER_NAME:PORT/DATABASE_NAME

## A.20. データソースの属性



### 注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/jboss\_as\_logging\_5\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.44 データソースの属性

属性	データソースタイプ	説明
allocation-retry	非 XA、XA	例外が発生する前に接続の割り当てを試行する回数。デフォルトは <b>0</b> で、初回の割り当て失敗で例外が発生します。
allocation-retry-wait-millis	非 XA、XA	接続の割り当てを再試行する間隔 (ミリ秒単位)。デフォルトは <b>5000</b> ミリ秒です。
allow-multiple-users	非 XA、XA	複数のユーザーが <b>getConnection(user, password)</b> メソッドを使いデータソースへアクセスするかどうか、およびこの挙動が内部プールタイプによるものであるかどうか。

属性	データソースタイプ	説明
authentication-context	非 XA、XA	プールの接続を区別するために使用される <b>javax.security.auth.Subject</b> を定義する Elytron 認証コンテキスト。
background-validation	非 XA、XA	接続がバックグラウンドスレッドで検証されるべきか、または使用前に検証されるか。通常バックグラウンド検証は <b>validate-on-match</b> とともに使用されません。使用されると冗長チェックが行われず、バックグラウンド検証では、検証時とクライアントへ渡す時の間で接続が不良になる可能性があるため、アプリケーションはこの可能性に対応する必要があります。
background-validation-millis	非 XA、XA	バックグラウンド検証を実行する頻度 (ミリ秒単位)。
blocking-timeout-wait-millis	非 XA、XA	例外が発生する前に接続を待機している間にブロックする最大時間 (ミリ秒単位)。この時間を超過すると、例外が発生します。これは、接続のロックを待っている間のみブロックし、新規接続の作成に長時間要している場合は例外は発生しません。
capacity-decrementer-class	非 XA、XA	プールの接続をデクリメントするポリシーを定義するクラス。
capacity-decrementer-properties	非 XA、XA	プールの接続をデクリメントするポリシーを定義するクラスにインジェクトされるプロパティ。
capacity-incrementer-class	非 XA、XA	プールの接続をインクリメントするポリシーを定義するクラス。
capacity-incrementer-properties	非 XA、XA	プールの接続をインクリメントするポリシーを定義するクラスにインジェクトされるプロパティ。
check-valid-connection-sql	非 XA、XA	プール接続の妥当性を確認する SQL ステートメント。これは、プールから管理接続を取得するときに呼び出される場合があります。
connectable	非 XA、XA	CMR の使用を有効にします。これは、ローカルリソースが信頼して XA トランザクションに参加できることを意味します。



属性	データソースタイプ	説明
connection-listener-class	非 XA、XA	<b>org.jboss.jca.adapters.jdbc.spi.listener.ConnectionListener</b> を拡張するクラス名を指定します。このクラスは、接続がアプリケーションまたはプールに返される前にアクションを実行するために接続のアクティベーションおよびパッシベーションをリッスンします。「 <a href="#">コアモジュールとしての JDBC ドライバーのインストール</a> 」のとおり、2つのリソース JAR を使用して1つのモジュールで指定されたクラスを JDBC ドライバーとバンドルするか、「 <a href="#">グローバルモジュールの定義</a> 」のとおり、指定されたクラスを個別のグローバルモジュールでバンドルする必要があります。
connection-listener-property	非 XA、XA	<b>connection-listener-class</b> で指定されたクラスにインジェクトされるプロパティ。インジェクトされるプロパティは JavaBeans の慣例に対応します。たとえば、 <b>foo</b> という名前のプロパティを指定する場合、接続リスナークラスには <b>String</b> を引数として許可するメソッド <b>setFoo</b> が必要です。
connection-properties	非 XA のみ	<b>Driver.connect(url, props)</b> メソッドに渡す任意の文字列名と値のペアである接続プロパティ。
connection-url	非 XA のみ	JDBC ドライバーの接続 URL。
credential-reference	非 XA、XA	データソース上で認証される、認証情報ストアからの認証情報。
datasource-class	非 XA のみ	JDBC データソースクラスの完全修飾名。
driver-class	非 XA のみ	JDBC ドライバークラスの完全修飾名。
driver-name	非 XA、XA	データソースが使用する JDBC ドライバーを定義します。インストールされたドライバーに一致するシンボリック名になります。ドライバーが JAR としてデプロイされた場合、名前はデプロイメントの名前になります。
elytron-enabled	非 XA、XA	接続の認証の処理に Elytron セキュリティーを有効にします。指定されたコンテキストがない場合、使用される Elytron の <b>authentication-context</b> が現在のコンテキストになります。詳細は <b>authentication-context</b> を参照してください。
enabled	非 XA、XA	データソースを有効にするかどうか。

属性	データソースタイプ	説明
enlistment-trace	非 XA、XA	エンリストメントトレースを記録するかどうか。デフォルトは <b>false</b> です。
exception-sorter-class-name	非 XA、XA	例外がエラーをブロードキャストする場合に検証するメソッドを提供する <b>org.jboss.jca.adapters.jdbc.ExceptionSorter</b> のインスタンス。
exception-sorter-properties	非 XA、XA	例外ソータープロパティ。

属性	データソースタイプ	説明
flush-strategy	非 XA、XA	<p>エラーの場合にプールをフラッシュする方法を指定します。有効な値は以下のとおりです。</p> <p><b>FailingConnectionOnly</b> 問題のある接続のみが削除されます。これはデフォルト設定です。</p> <p><b>InvalidIdleConnections</b> 同じ認証情報を共有し <a href="#">ValidatingManagedConnectionFactory.getInvalidConnections(...)</a> メソッドによって無効として返される、問題のある接続とアイドル状態の接続が削除されます。</p> <p><b>IdleConnections</b> 同じ認証情報を共有する問題のある接続とアイドル状態の接続が削除されます。</p> <p><b>Gracefully</b> 同じ認証情報を共有する問題のある接続とアイドル状態の接続が削除されます。同じ認証情報を共有するアクティブな接続は、プールに戻された時点で破棄されます。</p> <p><b>EntirePool</b> 同じ認証情報を共有する問題のある接続、アイドル状態の接続、およびアクティブな接続が削除されます。この設定は本番環境のシステムには推奨されません。</p> <p><b>AllInvalidIdleConnections</b> <a href="#">ValidatingManagedConnectionFactory.getInvalidConnections(...)</a> メソッドによって無効として返される、問題のある接続とアイドル状態の接続が削除されます。</p> <p><b>AllIdleConnections</b> 問題のある接続と、アイドル状態の接続すべてが削除されます。</p> <p><b>AllGracefully</b> 問題のある接続と、アイドル状態の接続すべてが削除されます。アクティブな接続は、プールに戻された時点で破棄されます。</p> <p><b>AllConnections</b> 問題のある接続、アイドル状態の接続すべて、およびアクティブな接続すべてが削除されます。この設定は本番環境のシステムには推奨されません。</p>
idle-timeout-minutes	非 XA、XA	<p>接続が閉じられるまでのアイドル最大時間 (分単位) を指定します。指定がない場合、デフォルトは <b>30</b> 分になります。実際の最大時間は、IdleRemover スキャン時間 (プールの最小 idle-timeout-minutes の半分) にも左右されます。</p>
initial-pool-size	非 XA、XA	<p>プールが保持する最初の接続数。</p>

属性	データソースタイプ	説明
interleaving	XA のみ	XA 接続のインターリービングを有効にするかどうか。
jndi-name	非 XA、XA	データソースの一意の JNDI 名。
jta	非 XA のみ	JTA の統合を有効にします。
max-pool-size	非 XA、XA	プールが保持可能な最大接続数。
mcp	非 XA、XA	<b>ManagedConnectionPool</b> 実装。例: <b>org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreArrayListManagedConnectionPool</b>
min-pool-size	非 XA、XA	プールが保持可能な最小接続数。
new-connection-sql	非 XA、XA	接続が接続プールに追加されたときに実行する SQL ステートメント。
no-recovery	XA のみ	リカバリーから接続プールが除外されるかどうか。
no-tx-separate-pool	XA のみ	各コンテキストに対して個別のサブプールを作成するかどうか。これは、JTA トランザクションの内部と外部の両方で XA 接続を使用できない一部の Oracle データソースで必要になることがあります。このオプションを使用すると、実際のプールが作成されるため、合計プールサイズが <b>max-pool-size</b> の倍になります。
pad-xid	XA のみ	Xid のパディングを行うかどうかを指定します。
password	非 XA、XA	新しい接続の作成時に使用するパスワード。
pool-fair	非 XA、XA	プールが fair であるかを定義します。この設定は、JCA の接続プールを管理するために使用される <b>Semaphore</b> クラスの一部で、リースする接続の順番が必要でない一部のユースケースでパフォーマンスが向上されます。
pool-prefill	非 XA、XA	プールをプレフィルするかどうか。
pool-use-strict-min	非 XA、XA	<b>min-pool-size</b> を厳格に考慮するかどうか。
prepared-statements-cache-size	非 XA、XA	LRU (Least Recently Used) キャッシュにある接続毎の準備済みステートメントの数。

属性	データソースタイプ	説明
query-timeout	非 XA、XA	クエリーのタイムアウト (秒単位)。デフォルトではタイムアウトはありません。
reauth-plugin-class-name	非 XA、XA	物理接続の再認証する再認証プラグイン実装の完全修飾クラス名。
reauth-plugin-properties	非 XA、XA	再認証プラグインのプロパティー。
recovery-authentication-context	XA のみ	プールの接続を区別するために使用される <b>javax.security.auth.Subject</b> を定義する Elytron 認証コンテキスト。
recovery-credential-reference	XA のみ	データソース上で認証される、認証情報ストアからの認証情報。
recovery-elytron-enabled	XA のみ	リカバリーに対する接続の認証の処理に Elytron セキュリティーを有効にします。指定された <b>authentication-context</b> がない場合、使用される Elytron の <b>authentication-context</b> が現在のコンテキストになります。詳細は <b>authentication-context</b> を参照してください。
recovery-password	XA のみ	リカバリーのリソースへの接続に使用するパスワード。
recovery-plugin-class-name	XA のみ	リカバリープラグイン実装の完全修飾クラス名。
recovery-plugin-properties	XA のみ	リカバリープラグインのプロパティー。
recovery-security-domain	XA のみ	リカバリーでリソースに接続するために使用するセキュリティドメイン。
recovery-username	XA のみ	リカバリーでリソースに接続するために使用するユーザー名。
same-rm-override	XA のみ	<b>javax.transaction.xa.XAResource.isSameRM(XAResource)</b> クラスが <b>true</b> または <b>false</b> を返すかどうか。
security-domain	非 XA、XA	認証処理を行う JAAS security-manager の名前。この名前は、JAAS ログイン設定の application-policy/name 属性に相関します。

属性	データソースタイプ	説明
set-tx-query-timeout	非 XA、XA	トランザクションがタイムアウトするまでの残り時間を基にクエリーのタイムアウトを設定するかどうかを指定します。トランザクションが存在しない場合は設定済みのクエリーのタイムアウトが使用されます。
share-prepared-statements	非 XA、XA	アプリケーションに提供されたラッパーがアプリケーションコードによって閉じられたときに、JBoss EAP が基盤の物理ステートメントを閉じたり終了せずに、キャッシュするかどうか。デフォルトは <b>false</b> です。
spy	非 XA、XA	JDBC レイヤーでスパイ機能を有効にします。この機能は、データソースへの JDBC トラフィックをすべてログに記録します。ロギングカテゴリーの <b>jboss.jdbc.spy</b> も <b>logging</b> サブシステムのログレベルである <b>DEBUG</b> に設定する必要があることに注意してください。
stale-connection-checker-class-name	非 XA、XA	<b>isStaleConnection(SQLException)</b> メソッドを提供する <b>org.jboss.jca.adapters.jdbc.StaleConnectionChecker</b> のインスタンス。このメソッドが <b>true</b> を返す場合、例外は <b>org.jboss.jca.adapters.jdbc.StaleConnectionException</b> でラップされます。
stale-connection-checker-properties	非 XA、XA	陳腐接続チェッカーのプロパティー。
statistics-enabled	非 XA、XA	ランタイム統計が有効になっているかどうか。デフォルトは <b>false</b> です。
track-statements	非 XA、XA	接続がプールへ返され、ステートメントが準備済みステートメントキャッシュへ返された時に、閉じられていないステートメントをチェックするかどうか。false の場合、ステートメントは追跡されません。有効な値は次のとおりです。 <ul style="list-style-type: none"> <li>● <b>true</b>: ステートメントと結果セットが追跡され、ステートメントが閉じられていない場合は警告が出力されます。</li> <li>● <b>false</b>: ステートメントと結果セットのいずれも追跡されません。</li> <li>● <b>nowarn</b>: ステートメントは追跡されますが、警告は出力されません (デフォルト)。</li> </ul>

属性	データソースタイプ	説明
tracking	非 XA、XA	トランザクション境界にまたがる接続ハンドルを追跡するかどうか。
transaction-isolation	非 XA、XA	<p><b>java.sql.Connection</b> トランザクション分離レベル。有効な値は次のとおりです。</p> <ul style="list-style-type: none"> <li>● <b>TRANSACTION_READ_UNCOMMITTED</b></li> <li>● <b>TRANSACTION_READ_COMMITTED</b></li> <li>● <b>TRANSACTION_REPEATABLE_READ</b></li> <li>● <b>TRANSACTION_SERIALIZABLE</b></li> <li>● <b>TRANSACTION_NONE</b></li> </ul>
url-delimiter	非 XA、XA	高可用性 (HA) データソースの connection-url にある URL の区切り文字。
url-property	XA のみ	<b>xa-datasource-property</b> 値の URL プロパティの プロパティ。
url-selector-strategy-class-name	非 XA、XA	<b>org.jboss.jca.adapters.jdbc.URLSelectorStrategy</b> を実装するクラス。
use-ccm	非 XA、XA	キャッシュ接続マネージャーを有効にします。
use-fast-fail	非 XA、XA	true の場合、接続が無効であれば最初に接続を割り当てしようとした時点で失敗します。false の場合、プールが枯渇するまで再試行します。
use-java-context	非 XA、XA	データソースをグローバル JNDI にバインドするかどうか。
use-try-lock	非 XA、XA	内部ロックのタイムアウト値。ロックが使用できない場合に即座に失敗するのではなく、タイムアウトする前に設定された秒数間ロックの取得を試みます。 <b>tryLock()</b> の代わりに <b>lock()</b> を使用します。
user-name	非 XA、XA	新しい接続の作成時に使用するユーザー名。

属性	データソースタイプ	説明
valid-connection-checker-class-name	非 XA、XA	<b>SQLException.isValidConnection(Connection)</b> メソッドを提供して接続を検証する <b>org.jboss.jca.adapters.jdbc.ValidConnectionChecker</b> の実装。接続が破棄されると例外が発生します。存在する場合、 <b>check-valid-connection-sql</b> 属性が上書きされます。
valid-connection-checker-properties	非 XA、XA	有効な接続チェッカープロパティ。
validate-on-match	非 XA、XA	接続ファクトリーが管理された接続への一致を試みたときに接続の検証が実行されるかどうか。これは、使用前にクライアントの接続を検証する必要がある場合に使用する必要があります。通常、Validate-on-match は <b>background-validation</b> と使用しません。使用すると冗長チェックが行われません。
wrap-xa-resource	XA のみ	<b>org.jboss.tm.XAResourceWrapper</b> インスタンスで XAResource をラップするかどうか。
xa-datasource-class	XA のみ	<b>javax.sql.XADataSource</b> 実装クラスの完全修飾名。
xa-datasource-properties	XA のみ	XA データソースプロパティの文字列名と値のペア。
xa-resource-timeout	XA のみ	ゼロ以外の値は <b>XAResource.setTransactionTimeout</b> メソッドに渡されます。

表A.45 JDBC ドライバー属性

属性	データソースタイプ	説明
datasource-class-info	非 XA、XA	<b>jdbc-driver</b> の <b>datasource-class</b> および <b>xa-datasource-class</b> に使用できるプロパティ。 <b>datasource-class</b> および <b>xa-datasource-class</b> 属性は、 <b>javax.sql.DataSource</b> または <b>javax.sql.XADataSource</b> クラスを実装する完全修飾クラス名を定義します。定義されたクラスはさまざまなプロパティのセッターを持つことができます。 <b>datasource-class-info</b> 属性は、クラスに設定できるこれらのプロパティをリストします。



## A.21. データソースの統計

表A.46 コアプールの統計

名前	説明
ActiveCount	アクティブな接続の数。各接続はアプリケーションによって使用されているか、プールで使用可能な状態であるかのいずれかになります。
AvailableCount	プールの使用可能な接続の数。
AverageBlockingTime	プールの排他ロックの取得をブロックするために費やされた平均時間。値はミリ秒単位です。
AverageCreationTime	接続の作成に費やされた平均時間。値はミリ秒単位です。
AverageGetTime	接続の取得に費やされた平均時間。
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。
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。

名前	説明
TotalBlockingTime	プールの排他ロックの待機に費やされた合計時間。値はミリ秒単位です。
TotalCreationTime	接続の作成に費やされた合計時間。値はミリ秒単位です。
TotalGetTime	接続の取得に費やされた合計時間。
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。
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。

名前	説明
XARrecoverAverageTime	XAResource recover 呼び出しの平均時間。
XARrecoverCount	XAResource recover 呼び出しの数。
XARrecoverMaxTime	XAResource recover 呼び出しの最大時間。
XARrecoverTotalTime	すべての XAResource recover 呼び出しの合計時間。
XARrollbackAverageTime	XAResource rollback 呼び出しの平均時間。
XARrollbackCount	XAResource rollback 呼び出しの数。
XARrollbackMaxTime	XAResource rollback 呼び出しの最大時間。
XARrollbackTotalTime	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.47 JDBC の統計

名前	説明
PreparedStatementCacheAccessCount	ステートメントキャッシュがアクセスされた回数。
PreparedStatementCacheAddCount	ステートメントキャッシュに追加されたステートメントの数。
PreparedStatementCacheCurrentSize	ステートメントキャッシュに現在キャッシュされた準備済みおよび呼び出し可能ステートメントの数。
PreparedStatementCacheDeleteCount	キャッシュから破棄されたステートメントの数。
PreparedStatementCacheHitCount	キャッシュからのステートメントが使用された回数。
PreparedStatementCacheMissCount	ステートメント要求がキャッシュのステートメントと一致しなかった回数。

## A.22. AGROAL データソースの属性



## 注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-agroal\_1\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.48 Agroal データソースの属性

属性	説明
connectable	このデータソースで CMR (Commit Markable Resource) 機能を有効にするかどうか。これは非 XA データソースのみに適用されます。
jndi-name	データソースの JNDI 名 を指定します。
jta	JTA 統合を有効にするかどうか。これは非 XA データソースのみに適用されます。
statistics-enabled	このデータソース統計を有効にするかどうか。デフォルトは <b>false</b> です。

表A.49 Agroal データソース接続ファクトリーの属性

属性	説明
authentication-context	<b>elytron</b> サブシステムの認証コンテキストへの参照。
connection-properties	接続の作成時に JDBC ドライバーに渡されるプロパティ。
credential-reference	認証に使用する認証情報ストアからの認証情報。
driver	JDBC ドライバーへの一意な参照。
new-connection-sql	作成後に接続で実行される SQL ステートメント。
password	データベースとの基本認証に使用するパスワード。
transaction-isolation	使用する <b>java.sql.Connection</b> トランザクション分離レベルを設定します。
url	JDBC ドライバーの接続 URL。
username	データベースとの基本認証に使用するユーザー名。

表A.50 Agroal データソース接続プールの属性

属性	説明
background-validation	バックグラウンドで実行されるバリデーションの合間 (ミリ秒単位)。
blocking-timeout	例外が発生する前に接続を待機している間にブロックする最大時間 (ミリ秒単位)。
idle-removal	削除する前に接続がアイドル状態である必要がある期間 (分単位)。
initial-size	プールが保持する最初の接続数。
leak-detection	漏えいを警告する前に接続を保持しなければならない期間 (ミリ秒単位)。
max-size	プールの接続の最大数。
min-size	プールが保持すべき最小接続数。

## A.23. トランザクションマネージャーの設定オプション



### 注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-txn\_5\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.51 Transactions サブシステムの属性

属性	説明
default-timeout	デフォルトのトランザクションタイムアウトです。デフォルトでは <b>300</b> 秒に設定されています。トランザクションごとにプログラムで上書きできます。
enable-statistics	<b>statistics-enabled</b> が導入されたため非推奨になりました。
enable-tsm-status	アウトオブプロセスのリカバリーに使用される TSM (トランザクションステータスマネージャー) サービスを有効にするかどうか。アウトオブプロセスのリカバリーマネージャーを実行してメモリー内ではなく異なるプロセスから <b>ActionStatusService</b> にアクセスすることはサポートされないため、このオプションはサポート対象外になります。
hornetq-store-enable-async-io	<b>journal-store-enable-async-io</b> が導入されたため非推奨になりました。

属性	説明
jdbc-action-store-drop-table	JDBC アクションストアがテーブルをドロップするか。デフォルトは <b>false</b> です。
jdbc-action-store-table-prefix	設定された JDBC アクションストアにトランザクションログを書き込むために使用されるテーブルのオプションの接頭辞。
jdbc-communication-store-drop-table	JDBC コミュニケーションストアがテーブルをドロップするか。デフォルトは <b>false</b> です。
jdbc-communication-store-table-prefix	設定された JDBC コミュニケーションストアにトランザクションログを書き込むために使用されるテーブルのオプションの接頭辞。
jdbc-state-store-drop-table	JDBC ステートストアがテーブルをドロップするか。デフォルトは <b>false</b> です。
jdbc-state-store-table-prefix	設定された JDBC ステートストアにトランザクションログを書き込むために使用されるテーブルのオプションの接頭辞。
jdbc-store-datasource	使用される非 XA データソースの名前。データソースは <b>datasources</b> サブシステムで定義する必要があります。
journal-store-enable-async-io	ジャーナルストアに対して <b>AsyncIO</b> を有効にするかどうか。デフォルトは <b>false</b> です。この設定を有効にするにはサーバーを再起動する必要があります。
jts	Java Transaction Service (JTS) トランザクションを使用するかどうかを指定します。デフォルト値は <b>false</b> で、JTA トランザクションのみを使用します。
maximum-timeout	トランザクションに無制限のタイムアウトを意味する <b>0</b> がトランザクションタイムアウトとして設定されている場合、トランザクションマネージャーは代わりにこの属性によって設定された値を使用します。デフォルトは <b>31536000</b> 秒 (365 日) です。

属性	説明
node-identifier	<p>トランザクションマネージャーのノード識別子。このオプションが設定されていないと、サーバーの起動時に警告が表示されます。このオプションは以下の場合に必要なになります。</p> <ul style="list-style-type: none"> <li>● JTS 対 JTS の通信</li> <li>● 2つのトランザクションマネージャーが共有のリソースマネージャーにアクセスする場合</li> <li>● 2つのトランザクションマネージャーが共有のオブジェクトマネージャーにアクセスする場合</li> </ul> <p>リカバリー中にデータの整合性を維持する必要があるため、各トランザクションマネージャーの node-identifier は一意である必要があります。複数のノードが同じリソースマネージャーと対話したり、トランザクションオブジェクトストアを共有したりするため、node-identifier は JTA に対しても一意である必要があります。</p>
object-store-path	<p>トランザクションマネージャーオブジェクトストアがデータを格納するファイルシステムの相対または絶対パスです。デフォルトは <b>object-store-relative-to</b> パラメーターに相対する値になります。<b>object-store-relative-to</b> を空の文字列に設定すると、この値は絶対パスとして処理されます。</p>
object-store-relative-to	<p>ドメインモデルのグローバルなパス設定を参照します。デフォルト値は、JBoss EAP のデータディレクトリーで、<b>jboss.server.data.dir</b> プロパティーの値です。デフォルトは、管理対象ドメインの場合は <b>EAP_HOME/domain/data/</b>、スタンドアロンサーバーインスタンスの場合は <b>EAP_HOME/standalone/data/</b> です。オブジェクトストアの <b>object-store-path</b> トランザクションマネージャー属性の値はこのパスに相対的です。<b>object-store-path</b> が絶対パスとして処理されるようにするには、この属性を空の文字列に設定します。</p>
process-id-socket-binding	<p>トランザクションマネージャーがソケットベースのプロセス ID を使用する必要がある場合に使用するソケットバインディング設定の名前。<b>process-id-uuid</b> が <b>true</b> に設定された場合は <b>undefined</b> になります。それ以外の場合は、設定する必要があります。</p>
process-id-socket-max-ports	<p>トランザクションマネージャーは、各トランザクションログに対し一意の識別子を作成します。一意の識別子を生成するメカニズムは2種類あります。ソケットベースのメカニズムとプロセスのプロセス識別子をベースにしたメカニズムです。</p> <p>ソケットベースの識別子の場合、あるソケットを開くと、そのポート番号が識別子に使用されます。ポートがすでに使用されている場合は、空きのポートが見つかるまで次のポートがプローブされます。process-id-socket-max-ports は失敗するまでトランザクションマネージャーが試行するソケットの最大数を表します。デフォルト値は <b>10</b> です。</p>

属性	説明
process-id-uuid	<b>true</b> に設定すると、プロセス識別子を使用して各トランザクションに一意的な識別子が作成されます。そうでない場合は、ソケットベースのメカニズムが使用されます。デフォルト値は <b>true</b> です。詳細は、 <b>process-id-socket-max-ports</b> を参照してください。 <b>process-id-socket-binding</b> を有効にするには、 <b>process-id-uuid</b> を <b>false</b> に設定します。
recovery-listener	トランザクションリカバリーのプロセスがネットワークソケットをリスンするかどうかを指定します。デフォルトは <b>false</b> です。
socket-binding	<b>recovery-listener</b> が <b>true</b> に設定されている場合にトランザクション周期リカバリリスナーによって使用されるソケットバインディングの名前を指定します。
statistics-enabled	統計を有効にするべきか。デフォルトは <b>false</b> です。
status-socket-binding	トランザクションステータスマネージャーに使用するソケットバインディングを指定します。この設定オプションはサポートされません。
use-hornetq-store	<b>use-journal-store</b> が導入されたため非推奨になりました。
use-jdbc-store	トランザクションログの書き込みに JDBC ストアを使用します。有効にする場合は <b>true</b> 、デフォルトのログストアタイプを使用する場合は <b>false</b> を設定します。
use-journal-store	ファイルベースのストレージの代わりに Apache ActiveMQ Artemis のジャーナルストレージメカニズムをトランザクションログに使用します。デフォルトでは無効になっていますが、I/O パフォーマンスが改善されます。別々のトランザクションマネージャーで JTS トランザクションを使用することは推奨されません。このオプションの変更を反映するには <b>shutdown</b> コマンドを使用してサーバーを再起動する必要があります。

表A.52 ログストアの属性

属性	説明
expose-all-logs	すべてのログを公開するかどうか。デフォルトは <b>false</b> で、トランザクションログのサブセットのみが公開されます。
type	ロギングストアの実装タイプを指定します。デフォルトは <b>default</b> です。

表A.53 CMR (Commit Markable Resource) の属性



属性	説明
batch-size	この CMR リソースのバッチサイズ。デフォルトは <b>100</b> です。
immediate-cleanup	このCMR リソースの即時クリーンアップを実行するかどうか。デフォルトは <b>true</b> です。
jndi-name	この CMR リソースの JNDI 名。
name	XID を格納するテーブルの名前。デフォルトは <b>xids</b> です。

## A.24. IIOP サブシステムの属性



### 注記

この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-iiop-openjdk\_3\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.54 IIOP サブシステムの属性

属性	説明
add-component-via-interceptor	SSL コンポーネントが IOR インターセプターによって追加されるべきかどうかを示します。 <b>非推奨</b> 。
auth-method	認証方法。有効な値は <b>none</b> および <b>username_password</b> です。
authentication-context	セキュリティーイニシャライザーが <b>elytron</b> に設定されたときに使用する認証コンテキストの名前。
caller-propagation	呼び出し元の ID を SAS コンテキストで伝播する必要があるかどうかを示します。有効な値は <b>none</b> および <b>supported</b> です。
client-requires	クライアント SSL がパラメーターを必要とすることを示す値。有効な値は <b>None</b> 、 <b>ServerAuth</b> 、 <b>ClientAuth</b> 、および <b>MutualAuth</b> です。 <b>非推奨</b> のため、代わりに <b>client-requires-ssl</b> を使用してください。
client-requires-ssl	サーバーからの IIOP 接続に SSL が必要であるかどうかを示します。
client-ssl-context	クライアント側の SSL ソケットの作成に使用される SSL コンテキストの名前。

属性	説明
client-supports	クライアント SSL がサポートされるパラメーターを示す値。有効な値は <b>None</b> 、 <b>ServerAuth</b> 、 <b>ClientAuth</b> 、および <b>MutualAuth</b> です。非推奨のため、代わりに <b>client-requires-ssl</b> を使用してください。
confidentiality	トランスポートに機密性の保護が必要であるかどうかを示します。有効な値は <b>none</b> 、 <b>supported</b> 、および <b>required</b> です。非推奨のため、代わりに <b>server-requires-ssl</b> を使用してください。
detect-misordering	トランスポートに間違った順序の検出が必要であるかどうかを示します。有効な値は <b>none</b> 、 <b>supported</b> 、および <b>required</b> です。非推奨のため、代わりに <b>server-requires-ssl</b> を使用してください。
detect-replay	トランスポートにリプレイの検出が必要であるかどうかを示します。有効な値は <b>none</b> 、 <b>supported</b> 、および <b>required</b> です。非推奨のため、代わりに <b>server-requires-ssl</b> を使用してください。
export-corballoc	ルートコンテキストを <b>corbaloc::address:port/NameService</b> としてエクスポートすべきかどうかを示します。
giop-version	使用される GIOP バージョン。
high-water-mark	TCP 接続キャッシュパラメーター。接続数がこの値を超えるたびに、ORB は接続を再利用しようとします。再利用される接続の数は、 <b>number-to-reclaim</b> プロパティによって指定されます。このプロパティが設定されていない場合は、デフォルトの OpenJDK ORB が使用されます。
integrity	トランスポートに整合性の保護が必要であるかどうかを示します。有効な値は <b>none</b> 、 <b>supported</b> 、および <b>required</b> です。非推奨のため、代わりに <b>server-requires-ssl</b> を使用してください。
number-to-reclaim	TCP 接続キャッシュパラメーター。接続数が <b>high-water-mark</b> プロパティを超えるたびに、ORB は接続を再利用しようとします。再利用される接続の数は、このプロパティによって指定されます。このプロパティが設定されていない場合は、デフォルトの OpenJDK ORB が使用されます。

属性	説明
persistent-server-id	サーバーの永続 ID。永続オブジェクト参照はサーバーの多くのアクティベーションで有効であり、このプロパティーを使用してこの ID を識別します。結果として、同じサーバーの多くのアクティベーションではこのプロパティーを同じ値に設定し、同じホストで実行されている異なる複数のサーバーインスタンスに異なるサーバー ID を割り当てる必要があります。
properties	汎用キー/値のプロパティー一覧。
realm	認証サービスのレルム名。
required	認証が必要であるかどうかを示します。
root-context	ネーミングサービスのルートコンテキスト
security	セキュリティインターセプターをインストールするかどうかを示します。有効な値は <b>client</b> 、 <b>identity</b> 、 <b>elytron</b> 、および <b>none</b> です。
security-domain	SSL 接続の確立に使用されるキーストアとトラストストアを保持するセキュリティドメインの名前。
server-requires	サーバーSSL がパラメーターを必要とすることを示す値。有効な値は <b>None</b> 、 <b>ServerAuth</b> 、 <b>ClientAuth</b> 、および <b>MutualAuth</b> です。非推奨のため、代わりに <b>server-requires-ssl</b> を使用してください。
server-requires-ssl	サーバーへの IIOP 接続に SSL が必要であるかどうかを示します。
server-ssl-context	サーバー側の SSL ソケットの作成に使用される SSL コンテキストの名前。
server-supports	サーバーSSL がサポートされるパラメーターを示す値。有効な値は <b>None</b> 、 <b>ServerAuth</b> 、 <b>ClientAuth</b> 、および <b>MutualAuth</b> です。非推奨のため、代わりに <b>server-requires-ssl</b> を使用してください。
socket-binding	ORB ポートを指定するソケットバインディング設定名
ssl-socket-binding	ORB SSL を指定するソケットバインディング設定名
support-ssl	SSL がサポートされるかどうかを示します。

属性	説明
transactions	トランザクションインターセプターがインストールされるかどうかを示します。有効な値は <b>full</b> 、 <b>spec</b> 、および <b>none</b> です。値が <b>full</b> の場合は JTS が有効になり、 <b>spec</b> の場合は受信トランザクションコンテキストを拒否する JTS でない仕様に準拠するモードを有効にします。
trust-in-client	トランスポートの確立にクライアントの信頼が必要であるかどうかを示します。有効な値は <b>none</b> 、 <b>supported</b> 、および <b>required</b> です。非推奨のため、代わりに <b>server-requires-ssl</b> を使用してください。
trust-in-target	トランスポートの確立にターゲットの信頼が必要であるかどうかを示します。有効な値は <b>none</b> および <b>supported</b> です。非推奨のため、代わりに <b>server-requires-ssl</b> を使用してください。

## A.25. リソースアダプターの属性

以下の表はリソースアダプターの属性を示しています。



### 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-resource-adapters\_5\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.55 主な属性

属性	説明
archive	リソースアダプターアーカイブ。
beanvalidationgroups	使用する必要がある Bean バリデーショングループ。
bootstrap-context	使用する必要があるブートストラップコンテキストの一意的な名前。
config-properties	カスタム定義の設定プロパティ。
module	リソースアダプターがロードされるモジュール。
statistics-enabled	ランタイム統計が有効になっているかどうか。
transaction-support	リソースアダプターのトランザクションサポートレベル。有効な値は <b>NoTransaction</b> 、 <b>LocalTransaction</b> 、または <b>XATransaction</b> です。

属性	説明
wm-elytron-security-domain	使用される必要がある Elytron セキュリティドメインの名前を定義します
wm-security	このリソースアダプターの <b>wm.security</b> をオンまたはオフにします。false の場合、デフォルトを含む <b>wm-security-*</b> パラメーターが無視されます。
wm-security-default-groups	使用された <b>Subject</b> インスタンスに追加する必要があるデフォルトのグループリスト。
wm-security-default-principal	使用された <b>Subject</b> インスタンスに追加する必要があるデフォルトのプリンシパルリスト。
wm-security-domain	使用する必要のあるセキュリティドメインの名前。
wm-security-mapping-groups	グループマッピングのリスト。
wm-security-mapping-required	セキュリティー認証情報にマッピングが必要かどうかを定義します。
wm-security-mapping-users	ユーザーマッピングのリスト。



### 注記

リソースアダプターが **bootstrap-context** とともに、**elytron-enabled** が **true** に設定されているワークマネージャーを使用している場合、セキュリティドメインの仕様に **wm-security-domain** 属性ではなく **wm-elytron-security-domain** 属性を使用する必要があります。

表A.56 admin-objects Attributes

属性	説明
class-name	管理オブジェクトの完全修飾クラス名。
enabled	管理オブジェクトを有効にする必要があるかを指定します。
jndi-name	管理オブジェクトの JNDI 名。
use-java-context	false に設定するとオブジェクトがグローバル JNDI にバインドされません。

表A.57 connection-definitions Attributes

属性	説明
allocation-retry	接続の割り当てを再試行する回数を示します。この回数を超えると例外が発生します。
allocation-retry-wait-millis	接続の割り当てを再試行する間隔(ミリ秒単位)。
authentication-context	プールの接続を区別するために使用される <b>javax.security.auth.Subject</b> を定義する Elytron 認証コンテキスト。
authentication-context-and-application	プール内の接続を区別するために、 <b>getConnection(user, pw)</b> や <b>Subject</b> などからのアプリケーションによって提供されたパラメータを使用するかどうかを示します。これらのパラメータは、設定済みの <b>authentication-context</b> を使用するとき Elytron によって認証後に提供されます。
background-validation	接続をバックグラウンドで検証するか、または使用前に検証するかを指定します。値の変更後にサーバーを再起動する必要があります。
background-validation-millis	バックグラウンド検証を実行する期間(ミリ秒単位)。値の変更後にサーバーを再起動する必要があります。
blocking-timeout-wait-millis	例外が発生する前に接続を待機している間にブロックする最大時間(ミリ秒単位)。この時間を超過すると、例外が発生します。これは、接続のロックを待っている間のみブロックし、新規接続の作成に長時間要している場合は例外は発生しません。
capacity-decrementer-class	プールの接続をデクリメントするポリシーを定義するクラス。
capacity-decrementer-properties	プールの接続をデクリメントするポリシーを定義するクラスにインジェクトするプロパティ。
capacity-incrementer-class	プールの接続をインクリメントするポリシーを定義するクラス。
capacity-incrementer-properties	プールの接続をインクリメントするポリシーを定義するクラスにインジェクトするプロパティ。
class-name	管理された接続ファクトリーまたは管理オブジェクトの完全修飾クラス名。
connectable	CMR の使用を有効にします。この機能により、ローカルリソースが信頼して XA トランザクションに参加できます。
elytron-enabled	接続の認証の処理に Elytron セキュリティーを有効にします。指定されたコンテキストがない場合、使用される Elytron の <b>authentication-context</b> が現在のコンテキストになります。詳細は <b>authentication-context</b> を参照してください。
enabled	リソースアダプターを有効にするべきかどうかを指定します。

属性	説明
enlistment	リソースアダプターによってサポートされる場合に lazy enlistment (レイジーエンリストメント) が使用されるべきかどうかを指定します。
enlistment-trace	JBoss EAP または IronJacamar がエンリストメントトレースを記録すべきかどうかを指定します。デフォルトは <b>false</b> です。
flush-strategy	<p>エラーの場合にプールをフラッシュする方法を指定します。有効な値は以下のとおりです。</p> <p><b>FailingConnectionOnly</b> 問題のある接続のみが削除されます。これはデフォルト設定です。</p> <p><b>InvalidIdleConnections</b> 同じ認証情報を共有し <a href="#">ValidatingManagedConnectionFactory.getInvalidConnections(...)</a> メソッドによって無効として返される、問題のある接続とアイドル状態の接続が削除されます。</p> <p><b>IdleConnections</b> 同じ認証情報を共有する問題のある接続とアイドル状態の接続が削除されます。</p> <p><b>Gracefully</b> 同じ認証情報を共有する問題のある接続とアイドル状態の接続が削除されます。同じ認証情報を共有するアクティブな接続は、プールに戻された時点で破棄されます。</p> <p><b>EntirePool</b> 同じ認証情報を共有する問題のある接続、アイドル状態の接続、およびアクティブな接続が削除されます。この設定は本番環境のシステムには推奨されません。</p> <p><b>AllInvalidIdleConnections</b> <a href="#">ValidatingManagedConnectionFactory.getInvalidConnections(...)</a> メソッドによって無効として返される、問題のある接続とアイドル状態の接続が削除されます。</p> <p><b>AllIdleConnections</b> 問題のある接続と、アイドル状態の接続すべてが削除されます。</p> <p><b>AllGracefully</b> 問題のある接続と、アイドル状態の接続すべてが削除されます。アクティブな接続は、プールに戻された時点で破棄されます。</p> <p><b>AllConnections</b> 問題のある接続、アイドル状態の接続すべて、およびアクティブな接続すべてが削除されます。この設定は本番環境のシステムには推奨されません。</p>
idle-timeout-minutes	接続が閉じられるまでのアイドル最大時間(分単位)を指定します。実際の最大時間は、 <b>IdleRemover</b> スキャン時間(プールの最小 <b>idle-timeout-minutes</b> の半分)に基づきます。値の変更後にサーバーを再起動する必要があります。
initial-pool-size	プールが保持する最初の接続数。
interleaving	XA 接続のインターリーピングを有効にするかどうかを指定します。

属性	説明
jndi-name	接続ファクトリーの JNDI 名。
max-pool-size	プールの最大接続数。各サブプールではこの値を超える接続は作成されません。
mcp	<b>ManagedConnectionPool</b> 実装。例: <b>org.jboss.jca.core.connectionmanager.pool.mcp.Semaphore ArrayListManagedConnectionPool</b>
min-pool-size	プールの最小接続数。
no-recovery	接続プールがリカバリーから除外されるべきかどうかを指定します。
no-tx-separate-pool	Oracle では、XA 接続を JTA トランザクションの内部と外部の両方で使用することが推奨されません。この問題を回避するには、異なるコンテキストに個別のサブプールを作成します。
pad-xid	Xid のパディングを行うべきかどうかを指定します。
pool-fair	プールの使用が公正であるべきかどうかを指定します。
pool-prefill	プールをプレフィルすべきかどうかを指定します。値の変更後にサーバーを再起動する必要があります。
pool-use-strict-min	<b>min-pool-size</b> を厳格に考慮するかどうかを指定します。
recovery-authentication-context	リカバリーに使用される Elytron 認証コンテキスト。 <b>authentication-context</b> の指定がない場合、現在の認証コンテキストが使用されます。
recovery-credential-reference	接続のリカバリーで認証される、認証情報ストアからの認証情報。
recovery-elytron-enabled	Elytron 認証コンテキストがリカバリーに使用されることを示します。デフォルトは <b>false</b> です。
recovery-password	リカバリーに使用されるパスワード。
recovery-plugin-class-name	リカバリープラグイン実装の完全修飾クラス名。
recovery-plugin-properties	リカバリープラグインのプロパティ。
recovery-security-domain	リカバリーに使用されるセキュリティドメイン。
recovery-username	リカバリーに使用されるユーザー名。



属性	説明
same-rm-override	<b>javax.transaction.xa.XAResource.isSameRM(XAResource)</b> が true または false を返すかどうかを無条件に設定します。
security-application	プール内の接続を区別するために、アプリケーションにより提供されたパラメーター ( <b>getConnection(user, pw)</b> からなど) を使用することを指定します。
security-domain	プール内の接続を区別するために使用される <b>javax.security.auth.Subject</b> を定義するセキュリティドメイン。
security-domain-and-application	プール内の接続を区別するために、 <b>getConnection(user, pw)</b> または <b>Subject</b> など、セキュリティドメインからのアプリケーションが提供するパラメーターを使用するかどうかを示します。
sharable	共有可能な接続の使用を有効にします。サポートされる場合はレイジーアソシエーションが有効になります。
tracking	IronJacamar がトランザクション境界にまたがって接続ハンドルを追跡するかどうかを指定します。
use-ccm	キャッシュされた接続マネージャーの使用を有効にします。
use-fast-fail	<b>true</b> に設定すると、無効な場合に接続の割り当てが初回で失敗します。 <b>false</b> に設定すると、プールの潜在的な接続をすべて使い果たすまで試行を継続します。
use-java-context	<b>false</b> に設定するとオブジェクトがグローバル JNDI にバインドされません。
validate-on-match	接続ファクトリーが管理された接続への一致を試みたときに接続の検証を実行すべきかどうかを指定します。通常、バックグラウンド検証の使用のみに限定されます。
wrap-xa-resource	<b>XAResource</b> インスタンスを <b>org.jboss.tm.XAResourceWrapper</b> インスタンスでラップするかどうかを指定します。
xa-resource-timeout	値は <b>XAResource.setTimeout()</b> に渡されます (秒単位)。デフォルトは <b>0</b> です。

## A.26. リソースアダプターの統計

表A.58 リソースアダプターの統計

名前	説明
----	----

名前	説明
ActiveCount	アクティブな接続の数。各接続はアプリケーションによって使用されているか、プールで使用可能な状態であるかのいずれかになります。
AvailableCount	プールの使用可能な接続の数。
AverageBlockingTime	プールの排他ロックの取得をブロックするために費やされた平均時間。値はミリ秒単位です。
AverageCreationTime	接続の作成に費やされた平均時間。値はミリ秒単位です。
CreatedCount	作成された接続の数。
DestroyedCount	破棄された接続の数。
InUseCount	現在使用中の接続の数。
MaxCreationTime	接続の作成にかかった最大時間。値はミリ秒単位です。
MaxUsedCount	使用される接続の最大数。
MaxWaitCount	同時に接続を待機する要求の最大数。
MaxWaitTime	プールの排他ロックの待機に費やされた最大時間。
TimedOut	タイムアウトした接続の数。
TotalBlockingTime	プールの排他ロックの待機に費やされた合計時間。値はミリ秒単位です。
TotalCreationTime	接続の作成に費やされた合計時間。値はミリ秒単位です。
WaitCount	接続を待機する必要がある要求の数。

## A.27. UNDERTOW サブシステムの属性

**undertow** サブシステムのさまざまな要素の属性は以下の表を参照してください。



### 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-undertow\_4\_0.xsd** のスキーマ定義ファイルで確認してください。

- [主な属性](#)
- [アプリケーションセキュリティドメインの属性](#)

- バッファークャッシュの属性
- バイトバッファープールの属性
- サーブレットコンテナの属性
- フィルターの属性
- ハンドラーの属性
- サーバーの属性

表A.59 undertow の主な属性

属性	デフォルト	説明
default-security-domain	other	Web デプロイメントによって使用されるデフォルトのセキュリティドメイン。
default-server	default-server	デプロイメントに使用するデフォルトのサーバー。
default-servlet-container	default	デプロイメントに使用するデフォルトのサーブレットコンテナ。
default-virtual-host	default-host	デプロイメントに使用するデフォルトの仮想ホスト。
instance-id	<code>boss.node.name</code>	クラスターインスタンス ID。
statistics-enabled	false	統計を有効にするかどうか。

### アプリケーションセキュリティドメインの属性

アプリケーションセキュリティドメインの属性の構造は以下のとおりです。

- **application-security-domain**
  - **setting**
    - **single-sign-on**

### application-security-domain 属性

表A.60 application-security-domain 属性

属性	デフォルト	説明
enable-jacc	false	JACC を使用して承認を有効にします。
http-authentication-factory		マップされたセキュリティドメインを参照するデプロイメントで使用される HTTP 認証ファクトリー。

属性	デフォルト	説明
override-deployment-config	false	デプロイメントの認証設定がファクトリーによってオーバーライドされるべきかどうか。
referencing-deployments		現在このマッピングを参照しているデプロイメント。
security-domain		デプロイメントによって使用される <b>SecurityDomain</b> 。

## single-sign-on 属性

表A.61 single-sign-on 属性

属性	デフォルト	説明
client-ssl-context		バックチャンネルログアウト接続のセキュア化に使用される SSL コンテキストへの参照。
cookie-name	JSESSIONIDS SO	クッキーの名前。
credential-reference		プライベートキーエントリを復号化するための認証情報参照。
domain		使用されるクッキードメイン。
http-only	false	クッキーの httpOnly 属性を設定します。
key-alias		バックチャンネルログアウト接続の署名および検証に使用されるプライベートキーエントリのエイリアス。
key-store		プライベートキーエントリが含まれるキーストアへの参照。
path	/	クッキーのパス。
secure	false	クッキーの secure 属性を設定します。

## バッファークッシュの属性

表A.62 buffer-cache 属性

属性	デフォルト	説明
----	-------	----

属性	デフォルト	説明
buffer-size	1024	バッファのサイズ。バッファが小さいと領域をより効率的に使用できます。
buffers-per-region	1024	リージョンごとのバッファの数。
max-regions	10	リージョンの最大数。キャッシングに使用できる最大メモリ容量を制御します。

## バイトバッファープールの属性

表A.63 byte-buffer-pool 属性

属性	デフォルト	説明
buffer-size		<p>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。</p> <ul style="list-style-type: none"> <li>● RAM が 64 MB 未満の場合は 512 バイト</li> <li>● RAM が 64 - 128 MB の場合は 1024 バイト (1 KB)</li> <li>● RAM が 128 MB を超える場合は 16384 バイト (16 KB)</li> </ul> <p>この属性のパフォーマンス調整は、JBoss EAP の『パフォーマンスチューニングガイド』の「<a href="#">バッファープールの設定</a>」を参照してください。</p>
direct		<p>このバッファがダイレクトまたはヒーププールであることを示すブール値。指定のない場合は、システムで利用できる RAM を基にして値が設定されます。</p> <ul style="list-style-type: none"> <li>● 利用できる RAM が &lt; 64MB の場合、値は <b>false</b> に設定されます。</li> <li>● 利用できる RAM が &gt;= 64MB の場合、値は <b>true</b> に設定されます。</li> </ul> <p>ダイレクトプールには対応するヒーププールがあることにも注意してください。</p>
leak-detection-percent	0	漏えい検出に割り当てられるバッファの割合 (パーセント)。
max-pool-size		プールの保持するバッファの最大数。この制限を超えるバッファを割り当てすることができますが、プールが満杯の場合は保持されません。

属性	デフォルト	説明
thread-local-cache-size	12	スレッドごとのキャッシュのサイズ。これは最大サイズで、スレッドが実際にバッファを割り当てる場合にスマートサイズを使用してバッファをスレッドにのみ保持します。

### サーブレットコンテナの属性

サーブレットコンテナコンポーネントの構造は次のとおりです。

- [servlet-container](#)
  - [mime-mapping](#)
  - [setting](#)
    - [crawler-session-management](#)
    - [jsp](#)
    - [persistent-sessions](#)
    - [session-cookie](#)
    - [websockets](#)
  - [welcome-file](#)

servlet-container 属性

表A.64 servlet-container 属性

属性	デフォルト	説明
allow-non-standard-wrappers	false	標準のラッパークラスを拡張しないリクエストおよび応答ラッパーが使用可能かどうか。
default-buffer-cache	default	静的リソースのキャッシュに使用するバッファークッシュ。
default-cookie-version	0	アプリケーションによって作成されるクッキーに使用するデフォルトのクッキーバージョン。
default-encoding		デプロイされたすべてのアプリケーションに使用するデフォルトのエンコード。
default-session-timeout	30	コンテナにデプロイされたすべてのアプリケーションに対するデフォルトのセッションタイムアウト (分単位)。

属性	デフォルト	説明
directory-listing		デフォルトのサブレットにディレクトリーリスティングを有効にするかどうか。
disable-caching-for-secured-pages	true	ヘッダーを設定してセキュア化されたページのキャッシュを無効にするかどうか。無効にすると機密性の高いページが中間者によってキャッシュされる可能性があるため、セキュリティー上の問題が発生することがあります。
disable-file-watch-service	false	<b>true</b> に設定した場合、展開形式のデプロイメントの変更を監視するためにファイルウォッチサービスは使用されません。この属性は、 <b>io.undertow.disable-file-system-watcher</b> システムプロパティをオーバーライドします。
disable-session-id-reuse	false	<b>true</b> に設定した場合、未知のセッション ID は再使用されず、新しいセッション ID が生成されます。 <b>false</b> に設定した場合、同じサーバー上のアプリケーション間で同じ ID を共有できるようにするために別のデプロイメントのセッションマネージャーに存在する場合のみセッション ID が再使用されます。
eager-filter-initialization	false	最初にリクエストされたときではなく、デプロイメントの開始時に filter init() を呼び出すかどうか。
ignore-flush	false	サブレット出力ストリームでのフラッシュを無視します。ほとんどの場合でパフォーマンスに悪影響を与えます。
max-sessions		1度にアクティブにできるセッションの最大数。
proactive-authentication	true	プロアクティブ認証を使用すべきかどうか。 <b>true</b> の場合、認証情報のあるユーザーは常に認証されます。
session-id-length	30	生成されたセッション ID の長さ。セッション ID が長いほどセキュアになります。
stack-trace-on-error	local-only	エラーの発生時にスタックトレースのあるエラーページを生成するかどうか。値は <b>all</b> 、 <b>none</b> 、および <b>local-only</b> です。
use-listener-encoding	false	リスナーで定義されたエンコードを使用します。

## mime-mapping 属性

表A.65 mime-mapping 属性

属性	デフォルト	説明
value		このマッピングの mime タイプ。

### crawler-session-management 属性

クローラーボット (crawler bot) に特別なセッション処理を設定します。



#### 注記

管理 CLI を使用して **crawler-session-management** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。例を以下に示します。

```
/subsystem=undertow/servlet-container=default/setting=crawler-session-management:add
/subsystem=undertow/servlet-container=default/setting=crawler-session-management:read-resource
```

表A.66 crawler-session-management 属性

属性	デフォルト	説明
session-timeout		クローラーが所有するセッションのセッションタイムアウト (秒単位)。
user-agents		クローラーのユーザーエージェントの一致に使用される正規表現。

### jsp 属性



#### 注記

管理 CLI を使用して **jsp** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。例を以下に示します。

```
/subsystem=undertow/servlet-container=default/setting=jsp:read-resource
```

表A.67 jsp 属性

属性	デフォルト	説明
check-interval	0	バックグラウンドスレッドを使用して JSP 更新の間隔をチェックします。ファイルシステム通知 API を使用して JSP 変更通知が処理されるほとんどのデプロイメントでは効果はありません。ファイル監視サービスが無効になっている場合のみ有効です。
development	false	JSP のリロードをオンザフライで有効にする開発モードを有効にします。



属性	デフォルト	説明
disabled	false	JSP コンテナを有効にします。
display-source-fragment	true	ランタイムエラーの発生時に、対応する JSP ソースの断片の表示を試行します。
dump-smap	false	SMAP データをファイルに書き込みます。
error-on-use-bean-invalid-class-attribute	false	useBean で不適切なクラスを使用するときにエラーを有効にします。
generate-strings-as-char-arrays	false	String 定数を char 配列として生成します。
java-encoding	UTF8	Java ソースに使用するエンコーディングを指定します。
keep-generated	true	生成されたサーブレットを保持します。
mapped-file	true	JSP ソースへマップします。
modification-test-interval	4	更新の 2 つのテスト間の最小時間 (秒単位)。
optimize-scriptlets	false	文字列連結の削除に JSP スクリプトレットを最適化するかどうか。
recompile-on-fail	false	各リクエストで失敗した JSP のコンパイルを再試行します。
scratch-dir		別のワークディレクトリを指定します。
smap	true	SMAP を有効にします。
source-vm	1.8	コンパイルのソース VM レベル。
tag-pooling	true	タブプーリングを有効にします。
target-vm	1.8	コンパイルのターゲット VM レベル。
trim-spaces	false	生成されたサーブレットから一部の領域をトリミングします。
x-powered-by	true	x-powered-by で JSP エンジンのアドバタイズを有効にします。

**persistent-sessions** 属性



### 注記

管理 CLI を使用して **persistent-sessions** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。例を以下に示します。

```
/subsystem=undertow/servlet-container=default/setting=persistent-sessions:add
/subsystem=undertow/servlet-container=default/setting=persistent-sessions:read-resource
```

表A.68 persistent-sessions 属性

属性	デフォルト	説明
path		永続セッションデータディレクトリーへのパス。null の場合、セッションがメモリーに保存されます。
relative-to		相対パスの起点となるディレクトリー。

### session-cookie 属性



### 注記

管理 CLI を使用して **session-cookie** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。例を以下に示します。

```
/subsystem=undertow/servlet-container=default/setting=session-cookie:add
/subsystem=undertow/servlet-container=default/setting=session-cookie:read-resource
```

表A.69 session-cookie 属性

属性	デフォルト	説明
comment		クッキーのコメント。
domain		クッキーのドメイン。
http-only		クッキーが http 専用であるかどうか。
max-age		クッキーの最大有効期間。
name		クッキーの名前。
secure		クッキーがセキュアであるかどうか。

### websockets 属性



## 注記

管理 CLI を使用して **websockets** 要素を管理する場合、**servlet-container** 要素の **settings** 下で使用できます。例を以下に示します。

```
/subsystem=undertow/servlet-container=default/setting=websockets:read-resource
```

表A.70 websockets 属性

属性	デフォルト	説明
buffer-pool	default	websocket デプロイメントに使用するバッファープール。
deflater-level	0	DEFLATE アルゴリズムの圧縮レベルを設定します。
dispatch-to-worker	true	コールバックがワーカースレッドにディスパッチされるべきかどうか。 <b>false</b> の場合、IO スレッドで実行され、より高速になりますが、ブロッキング操作を実行しないように注意する必要があります。
per-message-deflate	false	websocket のメッセージごとの圧縮拡張機能を有効にします。
worker	default	websocket デプロイメントに使用するワーカー。

## welcome-file 属性

ウェルカムファイルを定義し、オプションはありません。

## フィルターの属性

これらのコンポーネントは **/subsystem=undertow/configuration=filter** にあります。

## custom-filter フィルター

表A.71 custom-filter 属性

属性	デフォルト	説明
class-name		HttpHandler のクラス名。
module		クラスをロードできるモジュール名。
parameters		フィルターのパラメーター。

## error-page フィルター エラーページ。

表A.72 error-page 属性

属性	デフォルト	説明
code		エラーページコード。
path		エラーページパス。

### expression-filter フィルター

Undertow 式言語から解析されたフィルター。

表A.73 expression-filter 属性

属性	デフォルト	説明
expression		フィルターを定義する式。
module		フィルター定義のロードに使用するモジュール。

### gzip フィルター

gzip フィルターを定義し、属性はありません。

### mod-cluster フィルター

mod-cluster フィルターコンポーネントの構造は次のとおりです。

- **mod-cluster**
  - **balancer**
    - **load-balancing-group**
    - **node**
      - **context**

表A.74 mod-cluster 属性

属性	デフォルト	説明
advertise-frequency	10000	ネットワーク上で mod_cluster 自体がアドバタイズする頻度 (ミリ秒単位)。
advertise-path	/	このパス以下に mod-cluster が登録されます。
advertise-protocol	http	使用中のプロトコル。
advertise-socket-binding		アドバタイズに使用されるマルチキャストグループ。
broken-node-timeout	60000	この期間の経過後に破損したノードがテーブルから削除されます。

属性	デフォルト	説明
cached-connections-per-thread	5	永遠にキープアライブを使用する接続の数。
connection-idle-timeout	60	接続がアイドル状態でいられる期間。この期間を経過すると接続が閉じられます。プールサイズが設定された最小値に達すると ( <b>cached-connections-per-thread</b> によって設定) 接続はタイムアウトしません。
connections-per-thread	10	IO スレッドごとにバックエンドサーバーに保持される接続の数。
enable-http2	false	ロードバランサーがバックエンド接続の HTTP/2 へのアップグレードを試行すべきかどうか。HTTP/2 がサポートされていない場合は通常どおり HTTP または HTTPS が使用されます。
failover-strategy	LOAD_BALANCED	セッションアフィニティーが実行されるノードが利用できない場合にフェイルオーバーノードの選択方法を判断する属性。
health-check-interval	10000	バックエンドノードへのヘルスチェック ping の頻度。
http2-enable-push	true	HTTP/2 接続に対してプッシュを有効にするべきかどうか。
http2-header-table-size	4096	HPACK 圧縮に使用されるヘッダーテーブルのサイズ (バイト単位)。このメモリー量が圧縮のために接続ごとに割り当てられます。より大きな値はより多くのメモリーを使用しますが、圧縮が向上されます。
http2-initial-window-size	65535	クライアントがサーバーにデータを送信できる速度を制御するフロー制御ウィンドウサイズ (バイト単位)。
http2-max-concurrent-streams		単一の接続上でいつでもアクティブな状態になれる HTTP/2 の最大数。
http2-max-frame-size	16384	HTTP/2 の最大フレームサイズ (バイト単位)。
http2-max-header-list-size		サーバーが許可する用意があるリクエストヘッダーの最大サイズ (バイト単位)。
management-access-predicate		mod_cluster 管理コマンドを実行できるかどうかを判断するために受信リクエストに適用される述語。 <b>management-socket-binding</b> からのリクエストの管理を制限することで、提供されるセキュリティに追加のセキュリティを提供します。

属性	デフォルト	説明
management-socket-binding		mod_cluster 管理ポートのソケットバインディング。mod_cluster を使用する場合、リクエストを処理するパブリックの HTTP リスナーと、mod_cluster コマンドを処理するための内部ネットワークにバインドされた HTTP リスナーの 2 つの HTTP リスナーを定義する必要があります。このソケットバインディングは内部リスナーと対応する必要があります、公的にアクセスできない必要があります。
max-ajp-packet-size	8192	AJP パケットの最大サイズ (バイト単位)。この値を大きくすると、AJP はヘッダーの量が多いリクエストおよび応答に対応できます。ロードバランサーとバックエンドサーバーで同じにする必要があります。
max-request-time	-1	バックエンドノードへのリクエストの送信にかかる最大期間。この期間を超えるとリクエストが Kill されます。
max-retries	1	リクエストの失敗時に、リクエストの再試行を実行する回数。   <b>注記</b> リクエストがべき等とみなされない場合、バックエンドサーバーに送信されなかったことをプロキシが確信できる場合のみ再試行が行われます。
request-queue-size	10	接続プールが満杯の場合にキューに置けるリクエストの数。この数を超えるリクエストは拒否され、503 エラーが発生します。
security-key		mod_cluster グループに使用されるセキュリティーキー。すべてのメンバーが同じセキュリティーキーを使用する必要があります。
security-realm		SSL 設定を提供するセキュリティーレルム。 <b>非推奨: ssl-context 属性を使用して設定済みの SSLContext を直接参照してください。</b>
ssl-context		フィルターによって使用される <b>SSLContext</b> への参照。
use-alias	false	エイリアスチェックが実行されるかどうか。
worker	default	アドバタイズ通知の送信に使用される XNIO ワーカー。

表A.75 balancer 属性

属性	デフォルト	説明
max-attempts		リクエストをバックエンドサーバーへ送信する試行回数。
sticky-session		スティッキーセッションが有効であるかどうか。
sticky-session-cookie		セッションクッキー名。
sticky-session-force		<b>true</b> の場合、リクエストがスティッキーノードヘルレーティングできないとエラーが返されます。その他の場合は別のノードにルーティングされます。
sticky-session-path		スティッキーセッションクッキーのパス。
sticky-session-remove		リクエストを正しいホストヘルレーティングできない場合、セッションクッキーを削除します。
wait-worker		利用可能なワーカーを待つ秒数。

### load-balancing-group 属性

ロードバランシンググループを定義し、オプションはありません。

表A.76 node 属性

属性	デフォルト	説明
aliases		ノードのエイリアス。
cache-connections		永遠にキープアライブを使用する接続の数。
elected		選択 (elected) 数。
flush-packets		受信したデータを即座にフラッシュするかどうか。
load		このノードの現在の負荷。
load-balancing-group		このノードが属するロードバランシンググループ。
max-connections		IO スレッドごとの最大接続数。
open-connections		現在開かれている接続の数。
ping		ノードの ping。

属性	デフォルト	説明
queue-new-requests		リクエストが受信され、即座に使用できるワーカーがない場合にキューに置くかどうか。
read		ノードから読み取るバイト数。
request-queue-size		リクエストキューのサイズ。
status		このノードの現在の状態。
timeout		リクエストのタイムアウト。
ttl		接続の数が <b>cache-connections</b> よりも多い場合に、閉じられる前にリクエストがない状態で接続にキープアライブが使用される時間。
uri		ロードバランサーがノードへの接続に使用する URI。
written		ノードに転送されたバイト数。

表A.77 context 属性

属性	デフォルト	説明
requests		このコンテキストに対するリクエストの数。
status		このコンテキストの状態。

## request-limit フィルター

表A.78 request-limit 属性

属性	デフォルト	説明
max-concurrent-requests		同時リクエストの最大数。
queue-size		キューに置くリクエスト数。この数を超えるリクエストは拒否されます。

## response-header フィルター

response-header フィルターはカスタムヘッダーの追加を可能にします。

表A.79 response-header 属性



属性	デフォルト	説明
header-name		ヘッダー名。
header-value		ヘッダーの値

### rewrite フィルター

表A.80 rewrite 属性

属性	デフォルト	説明
redirect	false	再書き込みの代わりにリダイレクトが行われるかどうか。
target		ターゲットを定義する式。定数ターゲットにリダイレクトを行う場合は、値を単一引用符で囲みます。

### ハンドラーの属性

これらのコンポーネントは `/subsystem=undertow/configuration=handler` にあります。

### file 属性

表A.81 file 属性

属性	デフォルト	説明
cache-buffer-size	1024	バッファのサイズ。
cache-buffers	1024	バッファの数。
case-sensitive	true	大文字と小文字を区別してファイル进行处理するかどうか。 <b>false</b> (区別しない) に設定すると、基盤のファイルシステムが大文字と小文字を区別しない場合のみ動作します。
directory-listing	false	ディレクトリーの一覧を有効にするかどうか。
follow-symlink	false	シンボリックリンクのフォローを有効にするかどうか。
path		ファイルハンドラーがリソースに対応する場所からのファイルシステム上のパス。
safe-symlink-paths		シンボリックリンクのターゲットとして安全なパス。

### 静的リソースに WebDAV を使用

過去のバージョンの JBoss EAP では、**web** サブシステムで WebDAV を使用して (**WebdavServlet** 経

由) 静的リソースをホストし、追加の HTTP メソッドを有効にしてこれらのファイルへのアクセスや操作を実行できました。JBoss EAP 7 では、ファイルハンドラーを経由した静的ファイルの対応メカニズムは **undertow** サブシステムによって提供されますが、**undertow** サブシステムは WebDAV をサポートしません。JBoss EAP 7 で WebDAV を使用する場合は、カスタムの WebDav サブレットを記述してください。

### reverse-proxy 属性

reverse-proxy ハンドラーコンポーネントの構造は以下のとおりです。

- **reverse-proxy**
  - **host**

表A.82 reverse-proxy 属性

属性	デフォルト	説明
cached-connections-per-thread	5	永遠にキープアライブを使用する接続の数。
connection-idle-timeout	60	接続がアイドル状態でいられる期間。この期間を経過すると接続が閉じられます。プールサイズが設定された最小値に達すると (cached-connections-per-thread によって設定) 接続はタイムアウトしません。
connections-per-thread	40	IO スレッドごとにバックエンドサーバーに保持される接続の数。
max-request-time	-1	プロキシーリクエストがアクティブな状態でいられる最大時間。この値を超えるとリクエストは kill されます。デフォルトは unlimited (無制限) です。
max-retries	1	リクエストの失敗時に、リクエストの再試行を実行する回数。  <div style="display: flex; align-items: center;">  <div> <p><b>注記</b></p> <p>リクエストがべき等とみなされない場合、バックエンドサーバーに送信されなかったことをプロキシーが確信できる場合のみ再試行が行われず。</p> </div> </div>
problem-server-retry	30	ダウンしたサーバーへの再接続を試みる前に待機する時間 (秒単位)。
request-queue-size	10	接続プールが満杯の場合にキューに置けるリクエストの数。この数を超えるリクエストは拒否され、503 エラーが発生します。
session-cookie-names	JSESSIONID	セッションクッキー名のカンマ区切りリスト。通常は JSESSIONID。

表A.83 host 属性

属性	デフォルト	説明
enable-http2	false	<b>true</b> の場合、プロキシは HTTP/2 を使用してバックエンドへの接続を試行します。サポートされていない場合は、HTTP/1.1 にフォールバックします。
instance-id		スティッキーセッションを有効にするために使用されるインスタンス ID または JVM ルート。
outbound-socket-binding		このホストのアウトバウンドソケットバインディング。
path	/	ホストがルート以外のリソースを使用する場合のオプションのパス。
scheme	http	使用されるスキームの種類。
security-realm		ホストへの接続の SSL 設定を提供するセキュリティレルム。
ssl-context		このハンドラーによって使用される SSLContext への参照。

### サーバーの属性

server コンポーネントの構造は次のとおりです。

- **server**
  - **ajp-listener**
  - **host**
    - **filter-ref**
    - **location**
      - **filter-ref**
    - **setting**
      - **access-log**
      - **http-invoker**
      - **single-sign-on**
  - **http-listener**
  - **https-listener**

server 属性

表 A.84 server 属性

属性	デフォルト	説明
default-host	default-host	サーバーのデフォルトの仮想ホスト。
servlet-container	default	サーバーのデフォルトのサーブレットコンテナ。

## ajp-listener 属性

表A.85 ajp-listener 属性

属性	デフォルト	説明
allow-encoded-slash	false	リクエストにエンコードされた文字 (例: <b>%2F</b> ) が含まれる場合にデコードするかどうか。
allow-equals-in-cookie-value	false	引用符で囲まれていないクッキー値のエスケープされていない等号記号を許可するかどうか。引用符で囲まれていないクッキー値に等号記号が含まれないことがあります。等号記号が含まれると、等号の前で値が終了します。残りのクッキー値は破棄されます。
allow-unescaped-characters-in-url	false	URL でエスケープされていない文字を許可するかどうか。 <b>true</b> に設定されていると、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を処理します。 <b>false</b> に設定されている場合、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を <b>HTTP Bad Request 400</b> 応答コードで拒否します。
always-set-keep-alive	true	仕様が厳密に必要としない場合でも Connection: keep-alive ヘッダーが応答に追加されるかどうか。
buffer-pipelined-data	false	パイプライン化されたリクエストをバッファリングするかどうか。
buffer-pool	default	AJP リスナーのバッファプール。
decode-url	true	<b>true</b> の場合、選択された文字エンコーディング (デフォルトでは UTF-8) を使用してパーサーが URL およびクエリーパラメーターをデコードします。 <b>false</b> の場合はデコードされません。これにより、ハンドラーによる希望の文字セットへのデコードが可能になります。
disallowed-methods	["TRACE"]	許可されない HTTP メソッドのカンマ区切りリスト。

属性	デフォルト	説明
enabled	true	リスナーが有効であるかどうか。 <b>非推奨: 属性を有効にすると、設定の一貫性の保持に問題が生じます。</b>
max-ajp-packet-size	8192	AJP パケットがサポートされる最大サイズ。変更する場合は、ロードバランサーとバックエンドサーバーで増やす必要があります。
max-buffered-request-size	16384	バッファ済みのリクエストの最大サイズ (バイト単位)。リクエストは通常バッファされませんが、バッファされる最も一般的なケースが POST リクエストの SSL 再ネゴシエーションを実行する場合があります。再ネゴシエーションを実行するには、POST データを完全にバッファする必要があります。
max-connections		同時接続の最大数。サーバー設定で値が設定されないと、同時接続の数は <b>Integer.MAX_VALUE</b> に制限されます。
max-cookies	200	解析されるクッキーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-header-size	1048576	HTTP リクエストヘッダーの最大サイズ (バイト単位)。
max-headers	200	解析されるヘッダーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-parameters	1000	解析されるパラメーターの最大数。ハッシュの脆弱性に対して保護するために使用されます。クエリーパラメーターと POST データの両方に適用されますが累積されません。たとえば、max-parameters の 2 倍をパラメーターの合計数とすることができます。
max-post-size	10485760	許可される最大 POST サイズ。
no-request-timeout	60000	接続がアイドル状態でいられる期間 (ミリ秒単位)。この期間を超えると接続がコンテナによって閉じられます。
read-timeout		ソケットの読み取りタイムアウトを設定します (ミリ秒単位)。読み取りに成功しないまま指定の時間が経過すると、ソケットの次の読み取りによって <b>ReadTimeoutException</b> が発生します。
receive-buffer		受信バッファサイズ。

属性	デフォルト	説明
record-request-start-time	false	リクエストの開始時間を記録し、リクエスト時間がログに記録されるようにするかどうか。パフォーマンスへの影響は小さいながら、ある程度の影響を与えます。
redirect-socket		このリスナーが SSL でないリクエストをサポートし、リクエストが一致する必要がある SSL トランスポートに対して受信された場合、リクエストをここに指定されたソケットバインディングポートに自動的にリダイレクトするかどうか。
request-parse-timeout		リクエストの解析に費やすことができる最大時間 (ミリ秒単位)。
resolve-peer-address	false	ホストの DNS ルックアップを有効にします。
scheme		リスナースキーム (HTTP または HTTPS)。デフォルトでは、スキームは受信 AJP リクエストから取得されます。
secure	false	<b>true</b> の場合、リクエストが HTTPS を使用しなくてもこのリスナーから送信されるリクエストはセキュアであるとマーク付けされます。
send-buffer		送信バッファサイズ。
socket-binding		AJP リスナーのソケットバインディング。
tcp-backlog		指定のバックログでサーバーを設定します。
tcp-keep-alive		実装に依存して TCP キープアライブメッセージを送信するようチャンネルを設定します。
url-charset	UTF-8	URL の文字セット。
worker	default	リスナーの XNIO ワーカー。
write-timeout		ソケットの書き込みタイムアウトを設定します (ミリ秒単位)。書き込みに成功しないまま指定の時間が経過すると、ソケットの次の書き込みによって <b>WriteTimeoutException</b> が発生します。

## host 属性

表A.86 host 属性

属性	デフォルト	説明
alias		ホストのエイリアスのカンマ区切りリスト。
default-response-code	404	設定した場合、サーバー上に要求されたコンテキストが存在しない場合に設定した応答コードが返信されます。
default-web-module	ROOT.war	デフォルトの Web モジュール。
disable-console-redirect	false	<b>true</b> に設定すると、 <b>/console</b> リダイレクトはこのホストに対して有効になりません。
queue-requests-on-start	true	<b>true</b> に設定すると、リクエストはこのホストに対して開始時にキューに置かれます。 <b>false</b> に設定された場合、代わりにデフォルトの応答コードが返されます。

### filter-ref 属性

表A.87 filter-ref 属性

属性	デフォルト	説明
predicate		predicate は交換を基に true または false の決定を行う簡単な方法です。多くのハンドラーには条件によって適用される要件があり、predicate は条件を指定する一般的な方法を提供します。
priority	1	フィルターの順序を定義します。小さい数字は、同じコンテキストの他の大きな数字よりも先にサーバーがハンドラーチェーンに含まれるよう指示します。値の範囲は、フィルターが最初に処理されることを示す <b>1</b> からフィルターが最後に処理されることを示す <b>2147483647</b> までになります。

### location 属性

表A.88 location 属性

属性	デフォルト	説明
handler		この場所のデフォルトのハンドラー。

### filter-ref 属性

表A.89 filter-ref 属性

属性	デフォルト	説明
predicate		predicate は交換を基に true または false の決定を行う簡単な方法です。多くのハンドラーには条件によって適用される要件があり、predicate は条件を指定する一般的な方法を提供します。
priority	1	フィルターの順序を定義します。1以上を設定する必要があります。同じコンテキスト下で数字が大きいほどサーバーのハンドラーチェーンでの順序が早くなるよう指示します。

## access-log 属性



### 注記

管理 CLI を使用して **access-log** 要素を管理する場合、**host** 要素の **settings** 下で使用できます。例を以下に示します。

```
/subsystem=undertow/server=default-server/host=default-host/setting=access-log:add
/subsystem=undertow/server=default-server/host=default-host/setting=access-log:read-resource
```

表A.90 access-log 属性

属性	デフォルト	説明
directory	<code>\${jboss.server.log.dir}</code>	ログを保存するディレクトリー。
extended	false	ログが拡張されたログファイル形式を使用するかどうか。



属性	デフォルト	説明
pattern	common	<p>アクセスログパターン。</p>  <p><b>注記</b></p> <p>リクエストの処理にかかった時間を出力するよう <b>pattern</b> を設定した場合、該当するリスナーの <b>record-request-start-time</b> 属性も有効にする必要があります。そうしないと、アクセスログに時間が適切に記録されません。例を以下に示します。例を以下に示します。</p> <pre>/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=record-request-start-time,value=true)</pre>
predicate		リクエストをログに記録するかどうかを判断する述語。
prefix	access_log	ログファイル名の接頭辞。
relative-to		相対パスの起点となるディレクトリー。
rotate	true	アクセスログを毎日ローテーションするかどうか。
suffix	log	ログファイル名の接尾辞。
use-server-log	false	ログが個別のファイルではなくサーバーログに書き込まれるかどうか。
worker	default	ロギングに使用するワーカーの名前。

## http-invoker Attributes

表A.91 http-invoker Attributes

属性	デフォルト	説明
http-authentication-factory		認証に使用する HTTP 認証ファクトリー。
path	wildfly-services	サービスがインストールされるパス。
security-realm		認証に使用するレガシーのセキュリティーレルム。

## single-sign-on 属性



## 注記

管理 CLI を使用して **single-sign-on** 要素を管理する場合は、**host** 要素の **settings** 下で使用できます。例を以下に示します。

```
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:add
/subsystem=undertow/server=default-server/host=default-host/setting=single-sign-on:read-resource
```



## 重要

配布されるシングルサインオンは、アプリケーションの観点からではこれまでのバージョンの JBoss EAP と変わりありませんが、JBoss EAP 7 では認証情報のキャッシュおよび配布の処理が異なります。JBoss EAP 7 で **ha** プロファイルを実行する場合、デフォルトではホストは関連するセッションおよび SSO クッキー情報が保存される独自の Infinispan キャッシュを持ちます。このキャッシュは Web キャッシュコンテナのデフォルトキャッシュがベースになります。また、JBoss EAP はホストすべての個別のキャッシュ間で情報の伝搬を処理します。

表A.92 single-sign-on 属性

属性	デフォルト	説明
cookie-name	JSESSIONIDS SO	クッキーの名前。
domain		使用されるクッキードメイン。
http-only	false	クッキーの httpOnly 属性を設定します。
path	/	クッキーのパス。
secure	false	クッキーの secure 属性を設定します。

## http-listener 属性

表A.93 http-listener 属性

属性	デフォルト	説明
allow-encoded-slash	false	リクエストにエンコードされた文字 (例: <b>%2F</b> ) が含まれる場合にデコードするかどうか。

属性	デフォルト	説明
allow-equals-in-cookie-value	false	引用符で囲まれていないクッキー値のエスケープされていない等号記号を許可するかどうか。引用符で囲まれていないクッキー値に等号記号が含まれないことがあります。等号記号が含まれると、等号の前で値が終了します。残りのクッキー値は破棄されます。
allow-unescaped-characters-in-url	false	URL でエスケープされていない文字を許可するかどうか。true に設定されていると、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を処理します。false に設定されている場合、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を <b>HTTP Bad Request 400</b> 応答コードで拒否します。
always-set-keep-alive	true	仕様が厳密に必要なとしない場合でも Connection: keep-alive ヘッダーが応答に追加されるかどうか。
buffer-pipelined-data	false	パイプライン化されたリクエストをバッファリングするかどうか。
buffer-pool	default	リスナーのバッファプール。
certificate-forwarding	false	証明書の転送を有効にするかどうか。有効な場合、リスナーは <b>SSL_CLIENT_CERT</b> 属性から証明書を取得します。これらのヘッダーを常に設定するようプロキシが設定されている場合のみ有効にする必要があります。
decode-url	true	選択された文字エンコーディング (デフォルトでは UFT-8) を使用してパーサーが URL およびクエリーパラメーターをデコードするかどうか。false の場合はデコードされません。これにより、ハンドラーによる希望の文字セットへのデコードが可能になります。
disallowed-methods	["TRACE"]	許可されない HTTP メソッドのカンマ区切りリスト。
enable-http2	false	このリスナーの HTTP/2 サポートを有効にするかどうか。
enabled	true	リスナーが有効であるかどうか。非推奨: 属性を有効にすると、設定の一貫性の保持に問題が生じます。
http2-enable-push	true	この接続に対してサーバープッシュが有効であるかどうか。

属性	デフォルト	説明
http2-header-table-size	4096	HPACK 圧縮で使用されるヘッダーテーブルのサイズ (バイト単位)。このメモリー量が圧縮のために接続ごとに割り当てられます。より大きな値はより多くのメモリーを使用しますが、圧縮が向上されます。
http2-initial-window-size	65535	クライアントがサーバーにデータを送信できる速度を制御するフロー制御ウィンドウサイズ (バイト単位)。
http2-max-concurrent-streams		単一の接続上でいつでもアクティブな状態になれる HTTP/2 の最大数。
http2-max-frame-size	16384	HTTP/2 の最大フレームサイズ (バイト単位)。
http2-max-header-list-size		サーバーが許可する用意があるリクエストヘッダーの最大サイズ。
max-buffered-request-size	16384	バッファー済みのリクエストの最大サイズ (バイト単位)。リクエストは通常バッファーされませんが、バッファーされる最も一般的なケースが POST リクエストの SSL 再ネゴシエーションを実行する場合があります。再ネゴシエーションを実行するには、POST データを完全にバッファーする必要があります。
max-connections		同時接続の最大数。サーバー設定で値が設定されないと、同時接続の数は <b>Integer.MAX_VALUE</b> に制限されます。
max-cookies	200	解析されるクッキーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-header-size	1048576	HTTP リクエストヘッダーの最大サイズ (バイト単位)。
max-headers	200	解析されるヘッダーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-parameters	1000	解析されるパラメーターの最大数。ハッシュの脆弱性に対して保護するために使用されます。クエリーパラメーターと POST データの両方に適用されますが累積されません。たとえば、max-parameters の 2 倍をパラメーターの合計数とすることができます。
max-post-size	10485760	許可される最大 POST サイズ。

属性	デフォルト	説明
no-request-timeout	60000	接続がアイドル状態でいられる期間 (ミリ秒単位)。この期間を超えると接続がコンテナによって閉じられます。
proxy-address-forwarding	false	x-forwarded-host および同様のヘッダーを有効にし、リモート IP アドレスおよびホスト名を設定するかどうか。
proxy-protocol	false	PROXY プロトコルを使用して接続情報を送信するかどうか。true に設定された場合、リスナーは <a href="#">The PROXY protocol Versions 1 &amp; 2</a> 仕様によって定義される PROXY プロトコルバージョン 1 を使用します。このオプションは、同じプロトコルをサポートするロードバランサーの背後になるリスナーにのみ有効にする必要があります。
read-timeout		ソケットの読み取りタイムアウトを設定します (ミリ秒単位)。読み取りに成功しないまま指定の時間が経過すると、ソケットの次の読み取りによって <b>ReadTimeoutException</b> が発生します。
receive-buffer		受信バッファサイズ。
record-request-start-time	false	リクエストの開始時間を記録し、リクエスト時間がログに記録されるようにするかどうか。パフォーマンスへの影響は小さいながら、ある程度の影響を与えます。
redirect-socket		このリスナーが SSL でないリクエストをサポートし、リクエストが一致する必要がある SSL トランスポートに対して受信された場合、リクエストをここに指定されたソケットバインディングポートに自動的にリダイレクトするかどうか。
request-parse-timeout		リクエストの解析に費やすことができる最大時間 (ミリ秒単位)。
require-host-http11	false	すべての HTTP/1.1 リクエストに <b>Host</b> ヘッダーが必要になります。リクエストにこのヘッダーが含まれないと、403 エラーにより拒否されます。
resolve-peer-address	false	ホストの DNS ルックアップを有効にします。
secure	false	true の場合、リクエストが HTTPS を使用しなくてもこのリスナーから送信されるリクエストはセキュアであるとマーク付けされます。

属性	デフォルト	説明
send-buffer		送信バッファサイズ。
socket-binding		リスナーのソケットバインディング。
tcp-backlog		指定のバックログでサーバーを設定します。
tcp-keep-alive		実装に依存して TCP キープアライブメッセージを送信するようチャンネルを設定します。
url-charset	UTF-8	URL の文字セット。
worker	default	リスナーの XNIO ワーカー。
write-timeout		ソケットの書き込みタイムアウトを設定します (ミリ秒単位)。書き込みに成功しないまま指定の時間が経過すると、ソケットの次の書き込みによって <b>WriteTimeoutException</b> が発生します。

## https-listener 属性

表A.94 https-listener 属性

属性	デフォルト	説明
allow-encoded-slash	false	リクエストにエンコードされた文字 (例: <b>%2F</b> ) が含まれる場合にデコードするかどうか。
allow-equals-in-cookie-value	false	引用符で囲まれていないクッキー値のエスケープされていない等号記号を許可するかどうか。引用符で囲まれていないクッキー値に等号記号が含まれないことがあります。等号記号が含まれると、等号の前で値が終了します。残りのクッキー値は破棄されます。
allow-unescaped-characters-in-url	false	URL でエスケープされていない文字を許可するかどうか。 <b>true</b> に設定されていると、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を処理します。 <b>false</b> に設定されている場合、リスナーはエスケープ処理されていない ASCII でない文字が含まれる URL を <b>HTTP Bad Request 400</b> 応答コードで拒否します。
always-set-keep-alive	true	仕様が厳密に必要としない場合でも Connection: keep-alive ヘッダーが応答に追加されるかどうか。
buffer-pipelined-data	false	パイプライン化されたリクエストをバッファリングするかどうか。

属性	デフォルト	説明
buffer-pool	default	リスナーのバッファプール。
certificate-forwarding	false	証明書の転送を有効にすべきかどうか。有効な場合、リスナーは <b>SSL_CLIENT_CERT</b> 属性から証明書を取得します。これらのヘッダーを常に設定するようプロキシが設定されている場合のみ有効にする必要があります。
decode-url	true	選択された文字エンコーディング (デフォルトでは UFT-8) を使用してパーサーが URL およびクエリーパラメーターをデコードするかどうか。false の場合はデコードされません。これにより、ハンドラーによる希望の文字セットへのデコードが可能になります。
disallowed-methods	["TRACE"]	許可されない HTTP メソッドのカンマ区切りリスト。
enable-http2	false	このリスナーの HTTP/2 サポートを有効にします。
enable-spdy	false	このリスナーの SPDY サポートを有効にします。非推奨: SPDY は HTTP/2 に置き換えられました。
enabled	true	リスナーが有効であるかどうか。非推奨: 属性を有効にすると、設定の一貫性の保持に問題が生じます。
enabled-cipher-suites		有効な SSL 暗号を設定します。非推奨: SSLContext が参照される場合は、暗号スイートで設定してサポートされるようにする必要があります。
enabled-protocols		SSL プロトコルを設定します。非推奨: SSLContext が参照される場合は、暗号スイートで設定してサポートされるようにする必要があります。
http2-enable-push	true	この接続に対してサーバープッシュが有効であるかどうか。
http2-header-table-size	4096	HPACK 圧縮で使用されるヘッダーテーブルのサイズ (バイト単位)。このメモリー量が圧縮のために接続ごとに割り当てられます。より大きな値はより多くのメモリーを使用しますが、圧縮が向上されます。
http2-initial-window-size	65535	クライアントがサーバーにデータを送信できる速度を制御するフロー制御ウィンドウサイズ (バイト単位)。
http2-max-concurrent-streams		単一の接続上でいつでもアクティブな状態になれる HTTP/2 の最大数。

属性	デフォルト	説明
http2-max-frame-size	16384	HTTP/2 の最大フレームサイズ (バイト単位)。
http2-max-header-list-size		サーバーが許可する用意があるリクエストヘッダーの最大サイズ。
max-buffered-request-size	16384	バッファ済みリクエストの最大サイズ (バイト単位)。リクエストは通常バッファされませんが、バッファされる最も一般的なケースが POST リクエストの SSL 再ネゴシエーションを実行する場合があります。再ネゴシエーションを実行するには、POST データを完全にバッファする必要があります。
max-connections		同時接続の最大数。サーバー設定で値が設定されないと、同時接続の数は <b>Integer.MAX_VALUE</b> に制限されます。
max-cookies	100	解析されるクッキーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-header-size	1048576	HTTP リクエストヘッダーの最大サイズ (バイト単位)。
max-headers	200	解析されるヘッダーの最大数。ハッシュの脆弱性に対して保護するために使用されます。
max-parameters	1000	解析されるパラメーターの最大数。ハッシュの脆弱性に対して保護するために使用されます。クエリーパラメーターと POST データの両方に適用されますが累積されません。たとえば、max-parameters の 2 倍をパラメーターの合計数とすることができます。
max-post-size	10485760	許可される最大 POST サイズ。
no-request-timeout	60000	接続がアイドル状態でいられる期間 (ミリ秒単位)。この期間を超えると接続がコンテナーによって閉じられます。
proxy-address-forwarding	false	x-forwarded-host ヘッダー (およびその他の x-forwarded-* ヘッダー) の処理を有効にし、このヘッダー情報を使用してリモートアドレスを設定します。これらのヘッダーを設定する信頼されたプロキシの背後でのみ使用する必要があります。そうでないと、リモートユーザーによる IP アドレスの偽装が可能になります。



属性	デフォルト	説明
proxy-protocol	false	PROXY プロトコルを使用して接続情報を送信するかどうか。true に設定された場合、リスナーは <a href="#">The PROXY protocol Versions 1 &amp; 2</a> 仕様によって定義される PROXY プロトコルバージョン 1 を使用します。このオプションは、同じプロトコルをサポートするロードバランサーの背後になるリスナーにのみ有効にする必要があります。
read-timeout		ソケットの読み取りタイムアウトを設定します (ミリ秒単位)。読み取りに成功しないまま指定の時間が経過すると、ソケットの次の読み取りによって <b>ReadTimeoutException</b> が発生します。
receive-buffer		受信バッファサイズ。
record-request-start-time	false	リクエストの開始時間を記録し、リクエスト時間がログに記録されるようにするかどうか。パフォーマンスへの影響は小さいながら、ある程度の影響を与えます。
request-parse-timeout		リクエストの解析に費やすことができる最大時間 (ミリ秒単位)。
require-host-http11	false	すべての HTTP/1.1 リクエストに Host ヘッダーが必要になります。リクエストにこのヘッダーが含まれないと、403 エラーにより拒否されます。
resolve-peer-address	false	ホストの DNS ルックアップを有効にします。
secure	false	<b>true</b> の場合、リクエストが HTTPS を使用しなくてもこのリスナーから送信されるリクエストはセキュアであると見なされます。
security-realm		リスナーのセキュリティーレルム。非推奨: <b>ssl-context</b> 属性を使用して設定済みの <b>SSLContext</b> を直接参照してください。
send-buffer		送信バッファサイズ。
socket-binding		リスナーのソケットバインディング。
ssl-context		このリスナーによって使用される <b>SSLContext</b> への参照。
ssl-session-cache-size		アクティブな SSL セッションの最大数。非推奨: <b>Elytron セキュリティーコンテキスト</b> 上で設定できるようになりました。

属性	デフォルト	説明
ssl-session-timeout		SSL セッションのタイムアウト (秒単位)。非推奨: Elytron セキュリティーコンテキスト上で設定できるようになりました。
tcp-backlog		指定のバックログでサーバーを設定します。
tcp-keep-alive		実装に依存して TCP キープアライブメッセージを送信するようチャンネルを設定します。
url-charset	UTF-8	URL の文字セット。
verify-client	NOT_REQUIRED	SSL チャンネルの希望の SSL クライアント認証モード。非推奨: SSLContext が参照される場合は、クライアント検証の必要なモードに対して直接設定する必要があります。
worker	default	リスナーの XNIO ワーカー。
write-timeout		ソケットの書き込みタイムアウトを設定します (ミリ秒単位)。書き込みに成功しないまま指定の時間が経過すると、ソケットの次の書き込みによって <b>WriteTimeoutException</b> が発生します。

## A.28. UNDERTOW サブシステムの統計

表A.95 ajp-listener の統計

名前	説明
bytes-received	このリスナーによって受信されたバイト数。
bytes-sent	このリスナーによって送信されたバイト数。
error-count	このリスナーによって送信された 500 応答コードの数。
max-processing-time	このリスナーのリクエストによる最大処理時間。
processing-time	このリスナーによって処理されるすべてのリクエストの合計処理時間。
request-count	このリスナーが対応したリクエストの数。

表A.96 http-listener の統計

名前	説明
bytes-received	このリスナーによって受信されたバイト数。
bytes-sent	このリスナーによって送信されたバイト数。
error-count	このリスナーによって送信された 500 応答コードの数。
max-processing-time	このリスナーのリクエストによる最大処理時間。
processing-time	このリスナーによって処理されるすべてのリクエストの合計処理時間。
request-count	このリスナーが対応したリクエストの数。

表A.97 https-listener の統計

名前	説明
bytes-received	このリスナーによって受信されたバイト数。
bytes-sent	このリスナーによって送信されたバイト数。
error-count	このリスナーによって送信された 500 応答コードの数。
max-processing-time	このリスナーのリクエストによる最大処理時間。
processing-time	このリスナーによって処理されるすべてのリクエストの合計処理時間。
request-count	このリスナーが対応したリクエストの数。

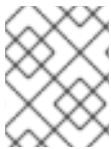
## A.29. HTTP メソッドのデフォルトの動作

JBoss EAP 7.2 の **undertow** サブシステムは、これまでの JBoss EAP リリースの **web** サブシステムとは HTTP メソッドのデフォルト動作が異なります。以下の表は、JBoss EAP 7.2 のデフォルト動作を表しています。

表A.98 HTTP メソッドのデフォルト動作

HTTP メソッド	JSP	静的 HTML	ファイルハンドラーによる静的 HTML
GET	OK	OK	OK
POST	OK	NOT_ALLOWED	OK
HEAD	OK	OK	OK

HTTP メソッド	JSP	静的 HTML	ファイルハンドラーによる静的 HTML
PUT	NOT_ALLOWED	NOT_ALLOWED	NOT_ALLOWED
TRACE	NOT_ALLOWED	NOT_ALLOWED	NOT_ALLOWED
DELETE	NOT_ALLOWED	NOT_ALLOWED	NOT_ALLOWED
OPTIONS	NOT_ALLOWED	OK	NOT_ALLOWED



### 注記

サーブレットでは、実装によってデフォルトの動作が異なりますが、デフォルトの動作が **NOT\_ALLOWED** である **TRACE** メソッドは例外です。

## A.30. REMOTING サブシステムの属性



### 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-undertow\_4\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.99 remoting 属性

属性	デフォルト	説明
worker-read-threads	1	リモートイングワーカーに対して作成する読み取りスレッドの数。
worker-task-core-threads	4	リモートイングワーカーのタスクスレッドプールに対するコアスレッドの数。
worker-task-keepalive	60	コアでないリモートイングワーカーのタスクスレッドにキープアライブを使用する期間 (ミリ秒単位)。
worker-task-limit	16384	許可するリモートイングワーカータスクの最大数。この数を超えるリモートイングワーカータスクは拒否されます。
worker-task-max-threads	16	リモートイングワーカーのタスクスレッドプールに対するスレッドの最大数。
worker-write-threads	1	リモートイングワーカーに作成する書き込みスレッドの数。



## 重要

**remoting** 要素の上記の属性は非推奨になりました。これらの属性は **io** サブシステムを使用して設定する必要があります。

表A.100 endpoint 属性

属性	デフォルト	説明
auth-realm		認証の CallbackHandler が指定されていない場合に使用する認証レルム。
authentication-retries	3	接続を閉じる前にクライアントが認証を再試行できる回数を指定します。
authorize-id		SASL 認証 ID。指定されている認証 CallbackHandler がなく、選択した SASL メカニズムがユーザー名を要求する場合に認証ユーザー名として使用されます。
buffer-region-size		割り当てられたバッファリージョンのサイズ。
heartbeat-interval	2147483647	接続ハートビートに使用する間隔 (ミリ秒単位)。この時間アウトバウンド方向の接続がアイドル状態であると、ping メッセージが送信され、対応する応答メッセージがトリガーされます。
max-inbound-channels	40	チャンネルの同時インバウンドメッセージの最大数。
max-inbound-message-size	92233720368 54775807	許可されるインバウンドメッセージの最大サイズ。メッセージがこのサイズを超えると、読み取り側および書き込み側に例外が発生します。
max-inbound-messages	80	接続に対してサポートされるインバウンドチャンネルの最大数。
max-outbound-channels	40	チャンネルの同時アウトバウンドメッセージの最大数。
max-outbound-message-size	92233720368 54775807	送信するアウトバウンドメッセージの最大サイズ。これよりも大きいサイズのメッセージは送信されません。送信しようとする、書き込み側に例外が発生します。
max-outbound-messages	65535	接続に対してサポートされるアウトバウンドチャンネルの最大数。
receive-buffer-size	8192	接続上でこのエンドポイントが許可する最大バッファサイズ。

属性	デフォルト	説明
receive-window-size	131072	接続チャンネルの受信方向の最大ウィンドウサイズ (バイト単位)。
sasl-protocol	<b>remote</b>	<b>SaslServer</b> または <b>SaslClient</b> が作成された場合、デフォルトで指定されるプロトコルは <b>remote</b> になります。この属性を使用すると、このプロトコルをオーバーライドできます。
send-buffer-size	8192	接続上でこのエンドポイントが送信する最大バッファサイズ。
server-name		最初の案内応答で接続のサーバー側はその名前をクライアントに渡します。デフォルトでは、名前は接続のローカルアドレスから自動的に検索されますが、これを使用して名前を上書きできます。
transmit-window-size	131072	接続チャンネルの受信方向の最大ウィンドウサイズ (バイト単位)。
worker	<b>default</b>	使用するワーカー



### 注記

**endpoint** 要素の更新に管理 CLI を使用する場合は、**remoting** 要素の **configuration** 下で利用できます。例: `/subsystem=remoting/configuration=endpoint/`

### コネクターの属性

connector コンポーネントの構造は次のとおりです。

- [connector](#)
  - [property](#)
  - [security](#)
    - [sasl](#)
      - [property](#)
      - [sasl-policy](#)
        - [policy](#)

表A.101 connector 属性

属性	デフォルト	説明
----	-------	----

属性	デフォルト	説明
authentication-provider		<b>authentication-provider</b> 要素には、受信接続に使用する認証プロバイダーの名前が含まれます。
sasl-authentication-factory		このコネクターをセキュアにする SASL 認証ファクトリーへの参照。
sasl-protocol	<b>remote</b>	認証に使用する SASL メカニズムに渡すプロトコル。
security-realm		このコネクターの認証に使用する、関連するセキュリティレルム。
server-name		最初のメッセージ交換で送信され、SASL ベースの認証用のサーバー名。
socket-binding		アタッチするソケットバインディングの名前。
ssl-context		このコネクターに使用する SSL コンテキストへの参照。

表A.102 property 属性

属性	デフォルト	説明
value		プロパティの値。

### セキュリティの属性

**security** コンポーネントはコネクターのセキュリティを設定できるようにしますが、直接の設定属性は含まれていません。これは、**sasl** などのネストされたコンポーネントを使用して設定できます。

表A.103 sasl 属性

属性	デフォルト	説明
include-mechanisms		オプションのネストされた <b>include-mechanisms</b> 要素には、許可される SASL メカニズム名のホワイトリストが含まれます。このリストにないメカニズムは許可されません。

属性	デフォルト	説明
qop		<p>オプションのネストされた <b>qop</b> 要素には、QOP (保護品質) の値が希望順に並ぶカンマ区切りリストが含まれます。</p> <p>このリストの QOP の値は次のとおりです。</p> <ul style="list-style-type: none"> <li>● <b>auth</b>: 認証のみ</li> <li>● <b>auth-int</b>: 認証と整合性保護</li> <li>● <b>auth-conf</b>: 認証、整合性保護および機密性保護</li> </ul>
reuse-session	false	<p>オプションのネストされた <b>reuse-session</b> ブール値要素は、以前認証したセッション情報をサーバーが再使用するかどうかを指定します。メカニズムによってはこのような再使用をサポートしない可能性があり、その他の要因によって再使用できないこともあります。</p>
server-auth	false	<p>オプションのネストされた <b>server-auth</b> ブール値要素は、サーバーがクライアントに対して認証されるかどうかを指定します。メカニズムによってはこの設定をサポートしない可能性があります。</p>
strength		<p>オプションのネストされた <b>strength</b> 要素には希望順に並ぶ暗号強度の値のカンマ区切りリストが含まれます。</p> <p>このリストの暗号強度の値は次のとおりです。</p> <ul style="list-style-type: none"> <li>● <b>high</b></li> <li>● <b>medium</b></li> <li>● <b>low</b></li> </ul>

## sasl-policy 属性

**sasl-policy** コンポーネントでは、利用できるメカニズムのセットを限定するために使用する任意のポリシーを指定できますが、直接の設定属性は含まれていません。[policy](#) などのネストされたコンポーネントを使用して設定できます。

表A.104 policy 属性

属性	デフォルト	説明
----	-------	----



属性	デフォルト	説明
forward-secrecy	true	オプションのネストされた <b>forward-secrecy</b> 要素には、セッション間で Forward Secrecy を実装するメカニズムを指定するブール値が含まれます。Forward Secrecy とは、あるセッションが侵入されてもその後のセッションに侵入するための情報が自動的に提供されないことを意味します。
no-active	true	オプションのネストされた <b>no-active</b> 要素には、能動的攻撃 (辞書攻撃でない) を受けやすいメカニズムを許可するかどうかを指定するブール値が含まれます。許可する場合は <b>false</b> 、拒否する場合は <b>true</b> を設定します。
no-anonymous	true	オプションのネストされた <b>no-anonymous</b> 要素には、匿名のログインを許可するメカニズムを許可するかどうかを指定するブール値が含まれます。許可する場合は <b>false</b> 、拒否する場合は <b>true</b> を設定します。
no-dictionary	true	オプションのネストされた <b>no-dictionary</b> 要素には、受動的な辞書攻撃を受けやすいメカニズムを許可するかどうかを指定するブール値が含まれます。許可する場合は <b>false</b> 、拒否する場合は <b>true</b> を設定します。
no-plain-text	true	オプションのネストされた <b>no-plain-text</b> 要素には、平文の受動的攻撃 (例: <b>PLAIN</b> ) を受けやすいメカニズムを拒否するかどうかを指定するブール値が含まれます。拒否する場合は <b>true</b> 、許可する場合は <b>false</b> を設定します。
pass-credentials	true	オプションネストされた <b>pass-credentials</b> 要素には、クライアントの認証情報を渡すメカニズムが必要であるかどうかを指定するブール値が含まれます。

## HTTP コネクタの属性

http-connector コンポーネントの構造は次のとおりです。

- [http-connector](#)
  - [property \(connector と同じ\)](#)
  - [security \(connector と同じ\)](#)
    - [sasl \(connector と同じ\)](#)
      - [property \(connector と同じ\)](#)

- [sasl-policy \(connector と同じ\)](#)
  - [policy \(connector と同じ\)](#)

表A.105 http-connector 属性

属性	デフォルト	説明
authentication-provider		<b>authentication-provider</b> 要素には、受信接続に使用する認証プロバイダーの名前が含まれます。
connector-ref		接続する <b>undertow</b> サブシステムのコネクターの名前。
sasl-authentication-factory		このコネクターをセキュアにする SASL 認証ファクトリーへの参照。
sasl-protocol	<b>remote</b>	認証に使用する SASL メカニズムに渡すプロトコル。
security-realm		このコネクターの認証に使用する、関連するセキュリティレルム。
server-name		最初のメッセージ交換で送信され、SASL ベースの認証用のサーバー名。

### アウトバウンド接続の属性

**outbound-connection** コンポーネントの構造は次のとおりです。

- [outbound-connection](#)
  - [property](#)

表A.106 outbound-connection 属性

属性	デフォルト	説明
uri		アウトバウンド接続の接続 URI。

表A.107 property 属性

属性	デフォルト	説明
value		プロパティの値。



#### 注記

上記の **property** 属性は、接続の作成中に使用される XNIO 操作に関連します。

### リモートアウトバウンド接続

**remote-outbound-connection** コンポーネントの構造は次のとおりです。

- [remote-outbound-connection](#)
  - [property \(outbound-connection と同じ\)](#)

表A.108 remote-outbound-connection 属性

属性	デフォルト	説明
authentication-context		アウトバウンド接続の設定が含まれる認証コンテキストインスタンスへの参照。
outbound-socket-binding-ref		接続の宛先アドレスとポートの判断に使用される <b>outbound-socket-binding</b> の名前。
protocol	<b>http-remoting</b>	リモート接続に使用するプロトコル。デフォルトは <b>http-remoting</b> です。非推奨: アウトバウンドセキュリティ設定は <b>authentication-context</b> 定義に移行する必要があります。
security-realm		パスワードと SSL 設定の取得に使用するセキュリティレルムへの参照。非推奨: アウトバウンドセキュリティ設定は <b>authentication-context</b> 定義に移行する必要があります。
username		リモートサーバーに対して認証する際に使用するユーザー名。非推奨: アウトバウンドセキュリティ設定は <b>authentication-context</b> 定義に移行する必要があります。

### ローカルアウトバウンド接続の属性

**local-outbound-connection** コンポーネントの構造は次のとおりです。

- [local-outbound-connection](#)
  - [property \(outbound-connection と同じ\)](#)

表A.109 local-outbound-connection 属性

属性	デフォルト	説明
outbound-socket-binding-ref		接続の宛先アドレスとポートの判断に使用される <b>outbound-socket-binding</b> の名前。

## A.31. IO サブシステムの属性




## 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-io\_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.110 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、 <b>MaxFileDescriptorCount</b> JMX プロパティを考慮して (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.111 buffer-pool の属性

属性	デフォルト	説明
		<p><b>注記</b></p> <p>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で設定されていますが、今後のリリースで <a href="#">Undertow バイトバッファープール</a> に置き換えられる予定です。バイトバッファープールの属性リストは「<a href="#">バイトバッファープールの属性</a>」を参照してください。</p>

属性	デフォルト	説明
buffer-size		<p>各バッファースライスサイズ (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。</p> <ul style="list-style-type: none"> <li>● RAM が 64 MB 未満の場合は 512 バイト</li> <li>● RAM が 64 - 128 MB の場合は 1024 バイト (1 KB)</li> <li>● RAM が 128 MB を超える場合は 16384 バイト (16 KB)</li> </ul> <p>この属性のパフォーマンス調整は、JBoss EAP の『パフォーマンスチューニングガイド』の「<a href="#">バッファープールの設定</a>」を参照してください。</p>
buffers-per-slice		<p>大型のバッファ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に割り当てするよりもメモリ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数が設定されます。</p> <ul style="list-style-type: none"> <li>● RAM が 128 MB 未満の場合は 10</li> <li>● RAM が 128 MB を超える場合は 20</li> </ul>
direct-buffers		<p>バッファープールが直接バッファを使用するかどうか。多くの場合で NIO を使用するとより高速になります。直接バッファ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。</p>

## A.32. JSF モジュールテンプレート

以下は、JBoss EAP に異なる JSF バージョンをインストールするときに必要な JSF モジュールに使用するテンプレートの例になります。手順の詳細は、「[JSF 実装のインストール](#)」を参照してください。

### 例: Mojarra JSF 実装 JAR の module.xml



#### 注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- JSF\_IMPL\_NAME
- JSF\_VERSION

```
<module xmlns="urn:jboss:module:1.8" name="com.sun.jsf-impl:JSF_IMPL_NAME-JSF_VERSION">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>
```

```

<dependencies>
  <module name="javax.faces.api:JSF_IMPL_NAME-JSF_VERSION"/>
  <module name="javaee.api"/>
  <module name="javax.servlet.jstl.api"/>
  <module name="org.apache.xerces" services="import"/>
  <module name="org.apache.xalan" services="import"/>
  <module name="org.jboss.weld.core"/>
  <module name="org.jboss.weld.spi"/>
  <module name="javax.xml.rpc.api"/>
  <module name="javax.rmi.api"/>
  <module name="org.omg.api"/>
</dependencies>

<resources>
  <resource-root path="jsf-impl-JSF_VERSION.jar"/>
</resources>
</module>

```

### 例: MyFaces JSF 実装 JAR の module.xml



#### 注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **JSF\_IMPL\_NAME**
- **JSF\_VERSION**

```

<module xmlns="urn:jboss:module:1.8" name="com.sun.jsf-impl:JSF_IMPL_NAME-JSF_VERSION">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <dependencies>
    <module name="javax.faces.api:JSF_IMPL_NAME-JSF_VERSION">
      <imports>
        <include path="META-INF/**"/>
      </imports>
    </module>
    <module name="javaee.api"/>
    <module name="javax.servlet.jstl.api"/>
    <module name="org.apache.xerces" services="import"/>
    <module name="org.apache.xalan" services="import"/>

    <!-- extra dependencies for MyFaces -->
    <module name="org.apache.commons.collections"/>
    <module name="org.apache.commons.codec"/>
    <module name="org.apache.commons.beanutils"/>
    <module name="org.apache.commons.digester"/>

    <!-- extra dependencies for MyFaces 1.1 -->
    <module name="org.apache.commons.logging"/>
    <module name="org.apache.commons.el"/>
    <module name="org.apache.commons.lang"/> -->
    <module name="javax.xml.rpc.api"/>
  </dependencies>
</module>

```

```

    <module name="javax.rmi.api"/>
    <module name="org.omg.api"/>
</dependencies>

<resources>
    <resource-root path="JSF_IMPL_NAME-impl-JSF_VERSION.jar"/>
</resources>
</module>

```

### 例: Mojarra JSF API JAR の module.xml



#### 注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **JSF\_IMPL\_NAME**
- **JSF\_VERSION**

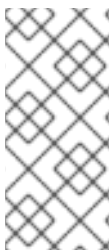
```

<module xmlns="urn:jboss:module:1.8" name="javax.faces.api:JSF_IMPL_NAME-JSF_VERSION">
  <dependencies>
    <module name="com.sun.jsf-impl:JSF_IMPL_NAME-JSF_VERSION"/>
    <module name="javax.enterprise.api" export="true"/>
    <module name="javax.servlet.api" export="true"/>
    <module name="javax.servlet.jsp.api" export="true"/>
    <module name="javax.servlet.jstl.api" export="true"/>
    <module name="javax.validation.api" export="true"/>
    <module name="org.glassfish.javax.el" export="true"/>
    <module name="javax.api"/>
    <module name="javax.websocket.api"/>
  </dependencies>

  <resources>
    <resource-root path="jsf-api-JSF_VERSION.jar"/>
  </resources>
</module>

```

### 例: MyFaces JSF API JAR の module.xml



#### 注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **JSF\_IMPL\_NAME**
- **JSF\_VERSION**

```

<module xmlns="urn:jboss:module:1.8" name="javax.faces.api:JSF_IMPL_NAME-JSF_VERSION">
  <dependencies>
    <module name="javax.enterprise.api" export="true"/>
    <module name="javax.servlet.api" export="true"/>
    <module name="javax.servlet.jsp.api" export="true"/>
    <module name="javax.servlet.jstl.api" export="true"/>
    <module name="javax.validation.api" export="true"/>
  </dependencies>

```

```

<module name="org.glassfish.javax.el" export="true"/>
<module name="javax.api"/>

<!-- extra dependencies for MyFaces 1.1
<module name="org.apache.commons.logging"/>
<module name="org.apache.commons.el"/>
<module name="org.apache.commons.lang"/> -->
</dependencies>

<resources>
  <resource-root path="myfaces-api-JSF_VERSION.jar"/>
</resources>
</module>

```

### 例: Mojarra JSF インジェクション JAR の module.xml



#### 注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- **JSF\_IMPL\_NAME**
- **JSF\_VERSION**
- **INJECTION\_VERSION**
- **WELD\_VERSION**

```

<module xmlns="urn:jboss:module:1.8" name="org.jboss.as.jsf-
injection:JSF_IMPL_NAME-JSF_VERSION">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <resources>
    <resource-root path="wildfly-jsf-injection-INJECTION_VERSION.jar"/>
    <resource-root path="weld-core-jsf-WELD_VERSION.jar"/>
  </resources>

  <dependencies>
    <module name="com.sun.jsf-impl:JSF_IMPL_NAME-JSF_VERSION"/>
    <module name="java.naming"/>
    <module name="java.desktop"/>
    <module name="org.jboss.as.jsf"/>
    <module name="org.jboss.as.web-common"/>
    <module name="javax.servlet.api"/>
    <module name="org.jboss.as.ee"/>
    <module name="org.jboss.as.jsf"/>
    <module name="javax.enterprise.api"/>
    <module name="org.jboss.logging"/>
    <module name="org.jboss.weld.core"/>
    <module name="org.jboss.weld.api"/>

```



```

    <module name="javax.faces.api:JSF_IMPL_NAME-JSF_VERSION"/>
  </dependencies>
</module>

```

#### 例: MyFaces JSF インジェクション JAR の module.xml



#### 注記

テンプレートの置き換え可能な変数に適切な値を使用するようにしてください。

- JSF\_IMPL\_NAME
- JSF\_VERSION
- INJECTION\_VERSION
- WELD\_VERSION

```

<module xmlns="urn:jboss:module:1.8" name="org.jboss.as.jsf-
injection:JSF_IMPL_NAME-JSF_VERSION">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <resources>
    <resource-root path="wildfly-jsf-injection-INJECTION_VERSION.jar"/>
    <resource-root path="weld-jsf-WELD_VERSION.jar"/>
  </resources>

  <dependencies>
    <module name="com.sun.jsf-impl:JSF_IMPL_NAME-JSF_VERSION"/>
    <module name="javax.api"/>
    <module name="org.jboss.as.web-common"/>
    <module name="javax.servlet.api"/>
    <module name="org.jboss.as.jsf"/>
    <module name="org.jboss.as.ee"/>
    <module name="org.jboss.as.jsf"/>
    <module name="javax.enterprise.api"/>
    <module name="org.jboss.logging"/>
    <module name="org.jboss.weld.core"/>
    <module name="org.jboss.weld.api"/>
    <module name="org.wildfly.security.elytron"/>

    <module name="javax.faces.api:JSF_IMPL_NAME-JSF_VERSION"/>
  </dependencies>
</module>

```

#### 例: MyFaces commons-digester JAR の module.xml



#### 注記

テンプレートの置き換え可能な **VERSION** 変数に適切な値を使用するようにしてください。

```
<module xmlns="urn:jboss:module:1.5" name="org.apache.commons.digester">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <resources>
    <resource-root path="commons-digester-VERSION.jar"/>
  </resources>

  <dependencies>
    <module name="javax.api"/>
    <module name="org.apache.commons.collections"/>
    <module name="org.apache.commons.logging"/>
    <module name="org.apache.commons.beanutils"/>
  </dependencies>
</module>
```

### A.33. JGROUPS サブシステムの属性

**jgroups** サブシステムのさまざまな要素の属性は以下の表を参照してください。

- [主な属性](#)
- [チャンネルの属性](#)
- [スタックの属性](#)



#### 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/jboss-as-jgroups\_5\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.112 jgroups の主な属性

属性	デフォルト	説明
default-channel	ee	デフォルトの JGroups チャンネル。
default-stack		デフォルトの JGroups プロトコルスタック。

#### チャンネルの属性

channel 要素の構造は次のとおりです。

- **channel**
  - **fork**
    - **protocol**
  - **protocol**

## channel 属性

表A.113 channel 属性

属性	デフォルト	説明
cluster		JGroups チャンネルのクラスター名。定義されていない場合はチャンネルの名前が使用されます。
module	org.wildfly.clustering.server	チャンネルサービスをロードするモジュール。
stack		JGroups チャンネルのプロトコルスタック。
statistics-enabled	false	統計を有効にするかどうか。
stats-enabled	false	統計を有効にするかどうか。 <b>非推奨:代わりに statistics-enabled 属性を使用してください。</b>

## スタックの属性

stack 要素の構造は次のとおりです。

- **stack**
  - **protocol**
  - **relay**
    - **remote-site**
  - **transport**
    - **thread-pool**

## stack 属性

表A.114 stack 属性

属性	デフォルト	説明
statistics-enabled	false	スタックのすべてのプロトコルが統計を収集するかどうかを示します。

## protocol 属性

一般的に使用されるプロトコルのリストは、「[JGroups プロトコル](#)」の項を参照してください。

表A.115 protocol 属性

属性	デフォルト	説明
module	org.jgroups	プロトコルタイプを解決するモジュール。

属性	デフォルト	説明
properties		このプロトコルのプロパティ。
statistics-enabled	false	スタック設定をオーバーライドして、このプロトコルが統計を収集するかどうかを示します。

## relay 属性

表A.116 relay 属性

属性	デフォルト	説明
module	org.jgroups	プロトコルタイプを解決するモジュール。
properties		このプロトコルのプロパティ。
site		ローカルサイトの名前。
statistics-enabled	false	スタック設定をオーバーライドして、このプロトコルが統計を収集するかどうかを示します。

## remote-site 属性

表A.117 remote-site 属性

属性	デフォルト	説明
channel		このリモートサイトと通信するために使用されるブリッジチャンネルの名前。
cluster		このリモートサイトへのブリッジチャンネルのクラスター名。 <b>非推奨</b> : 代わりに明示的に定義された <b>channel</b> を使用してください。
stack		このリモートサイトへのブリッジを作成するスタック。 <b>非推奨</b> : 代わりに明示的に定義された <b>channel</b> を使用してください。

## transport 属性

表A.118 transport 属性

属性	デフォルト	説明
default-executor		受信メッセージを処理するスレッドプールエグゼキューター。 <b>非推奨</b> : 代わりに事前定義された <b>default</b> スレッドプールを設定してください。

属性	デフォルト	説明
diagnostics-socket-binding		このプロトコル層の診断ソケットバインディング仕様。通信用の IP インターフェースとポートを指定するために使用されます。
machine		このノードのマシン (ホスト) 識別子。Infinispan のトポロジー認識のコンシステントハッシュ法によって使用されます。
module	org.jgroups	プロトコルタイプを解決するモジュール。
oob-executor		受信 OOB (out of band) メッセージを処理するスレッドプールエグゼキューター。 <b>非推奨: 代わりに事前定義された oob スレッドプールを設定してください。</b>
properties		このトランスポートのプロパティー。
rack		サーバーラックなど、このノードのラック識別子。Infinispan のトポロジー認識のコンシステントハッシュ法によって使用されます。
shared	false	<b>true</b> の場合、このスタックを使用するすべてのチャンネルによって基盤のトランスポートが共有されます。 <b>非推奨: 代わりにチャンネルの fork を設定してください。</b>
site		データセンターなど、このノードのサイト識別子。Infinispan のトポロジー認識のコンシステントハッシュ法によって使用されます。
socket-binding		このプロトコル層のソケットバインディング仕様。通信用の IP インターフェースおよびポートを指定するために使用されます。
statistics-enabled	false	スタック設定をオーバーライドして、このプロトコルが統計を収集するかどうかを示します。
thread-factory		非同期のトランスポート固有のタスクを処理するために使用するスレッドファクトリー。 <b>非推奨: 代わりに事前定義された internal スレッドプールを設定してください。</b>
timer-executor		プロトコル関連のタイミングタスクを処理するスレッドプールエグゼキューター。 <b>非推奨: 代わりに事前定義された timer スレッドプールを設定してください。</b>

## thread-pool 属性

表A.119 thread-pool 属性

属性	デフォルト	説明
keepalive-time	5000L	アイドル時にプールスレッドの実行を継続すべき時間(ミリ秒単位)。指定されていない場合は、スレッドはエグゼキューターが終了するまで実行します。
max-threads	4	スレッドプールの最大サイズ。
min-threads	2	<b>max-threads</b> よりも小さいコアスレッドプールサイズ。定義されていない場合、コアスレッドプールサイズは <b>max-threads</b> と同じになります。
queue-length	500	キューの長さ。

## A.34. JGROUPS プロトコル

プロトコル	プロトコルタイプ	説明
ASYM_ENCRYPT	暗号化	クラスタのコーディネーターに保存されるシークレットキーを使用して、クラスタメンバー間のメッセージを暗号化します。
AUTH	認証	認証のレイヤーをクラスタメンバーに提供します。
azure.AZURE_PIN G	検出	Microsoft Azure の blob ストレージを使用したノード検出をサポートします。
FD_ALL	障害検出	簡単なハートビートプロトコルを基にした障害検出を提供します。
FD_SOCKET	障害検出	クラスタメンバー間で作成された TCP ソケットのリングを基にした障害検出を提供します。
JDBC_PING	検出	メンバーがアドレスを書き込む共有データベースを使用してクラスタメンバーを検出します。
MERGE3	マージ	クラスタの分割が発生したときにサブクラスタをマージします。
MFC	フロー制御	送信元とすべてのクラスタメンバーとの間でマルチキャストフロー制御を提供します。
MPING	検出	IP マルチキャストでクラスタメンバーを検出します。

プロトコル	プロトコルタイプ	説明
pbcast.GMS	グループメンバーシップ	新規メンバーのクラスター参加、既存メンバーによる離脱リクエスト、およびクラッシュしたメンバーの <b>SUSPECT</b> メッセージを含む、グループメンバーシップを処理します。
pbcast.NAKACK2	メッセージの送信	メッセージの信頼性および順序を確実にし、1つの送信元から送信されたすべてのメッセージが送信順に受信されることを保証します。
pbcast.STABLE	メッセージの安定性	すべてのメンバーによって送信されたメッセージを削除します。
PING	検出	クラスターメンバーの動的検出のサポートによる、メンバーの最初の検出。
SASL	認証	SASL メカニズムを使用してクラスターメンバーに認証のレイヤーを提供します。
SYM_ENCRYPT	暗号化	共有キーストアを使用して、クラスターメンバー間のメッセージを暗号化します。
S3_PING	検出	Amazon S3 を使用して最初のメンバーを検出します。
TCPGOSSIP	検出	外部のゴシップルーターを使用してクラスターメンバーを検出します。
TCPPING	検出	クラスターを形成するクラスターメンバーのアドレスの静的リストが含まれます。
UFC	フロー制御	送信元とすべてのクラスターメンバーとの間でユニキャストフロー制御を提供します。
UNICAST3	メッセージの送信	ユニキャストメッセージのメッセージの信頼性および順序を確実にし、1つの送信元から送信されたすべてのメッセージが送信順に受信されることを保証します。
VERIFY_SUSPECT	障害検出	疑わしいメンバーをエビクトする前に最終的に ping してそのメンバーが不能になったことを検証します。

### 汎用プロトコルの属性

すべてのプロトコルは以下の属性にアクセスできます。

表A.120 protocol 属性

属性	デフォルト	説明
module	org.jgroups	プロトコルタイプを解決するモジュール。

属性	デフォルト	説明
properties		このプロトコルのプロパティ。
statistics-enabled	false	統計を有効にするかどうか。

## 認証プロトコル

認証プロトコルは、認証を実行するために使用されます。主な役目は、認証されたメンバーのみがクラスターに参加できるようにすることです。これらのプロトコルは、**GMS** プロトコルの下に位置するため、クラスター参加のリクエストをリッスンする可能性があります。

- [AUTH](#)
- [SASL](#)

### AUTH 属性

**AUTH** プロトコルにはその他の属性は含まれませんが、トークンが子要素として定義されている必要があります。



### 注記

このプロトコルを定義するとき、**protocol** 要素の代わりに **auth-protocol** 要素が使用されます。

### トークンタイプ

Elytron をセキュリティーに使用する場合、以下の認証トークンの1つを使用することが推奨されます。これらの認証トークンは、Elytron と使用する目的で設計されており、レガシーのセキュリティー設定とは使用できない可能性があります。

表A.121 Elytron トークンタイプ

トークン	説明
cipher-token	共有の秘密が変換される認証トークン。変換に使用されるデフォルトのアルゴリズムは RSA です。
digest-token	共有の秘密が変換される認証トークン。変換に使用されるデフォルトのアルゴリズムは SHA-256 です。
plain-token	共有の秘密に追加の変換を行わない認証トークン。

以下の認証トークンは JGroups から継承されます。認証が望ましいすべての設定で使用の対象となります。

表A.122 JGroups トークンタイプ

トークン	説明
------	----



トークン	説明
MD5Token	共有の秘密が MD5 または SHA ハッシュを使用して暗号化される認証トークン。この暗号化に使用されるデフォルトのアルゴリズムは MD5 です。
SimpleToken	共有の秘密に追加の変換を行わない認証トークン。このトークンは大文字と小文字を区別せず、文字列の一致を判断するときに大文字と小文字を考慮しません。
X509Token	共有の秘密が X509 証明書を使用して暗号化される認証トークン。

## SASL の属性

表A.123 SASL の属性

属性	デフォルト	説明
client_callback_handler		ノードがクライアントとして動作するとき使用する <b>CallbackHandler</b> のクラス名。
client_name		ノードがクライアントとして動作するとき使用する名前。JAAS ログインモジュールを使用する場合、この名前はサブジェクトの取得にも使用されます。
client_password		ノードがクライアントとして動作するとき使用するパスワード。JAAS ログインモジュールを使用する場合、このパスワードはサブジェクトの取得にも使用されます。
login_module_name		SASL クライアントおよびサーバーを作成するためにサブジェクトとして使用する JAAS ログインモジュールの名前。この属性は、GSSAPI などの一部の <b>mech</b> 値でのみ必要です。
mech		SASL 認証メカニズムの名前。この名前にはローカル SASL プロバイダーがサポートするすべてのメカニズムを指定でき、JDK はデフォルトで <b>CRAM-MD5</b> 、 <b>DIGEST-MD5</b> 、 <b>GSSAPI</b> 、および <b>NTLM</b> を提供します。
sasl_props		定義された <b>mech</b> のプロパティ。
server_callback_handler		ノードがサーバーとして動作するとき使用する <b>CallbackHandler</b> のクラス名。
server_name		完全修飾サーバー名。
timeout	5000	チャレンジへの応答を待つ時間 (ミリ秒単位)。

## 検出プロトコル

以下のプロトコルはクラスターの最初のメンバーシップを見つけるために使用され、その後現在のコーディネーターを決定するために使用できます。検出プロトコルのリストは次のとおりです。

- [AZURE\\_PING](#)
- [JDBC\\_PING](#)
- MPING
- PING
- [S3\\_PING](#)
- [TCPGOSSIP](#)
- [TCPPING](#)

### AZURE\_PING の属性

表A.124 AZURE\_PING の属性

属性	デフォルト	説明
container		<b>PING</b> データに使用する blob コンテナの名前。有効な DNS 名を指定する必要があります。
storage_access_key		ストレージアカウントのシークレットアクセスキー。
storage_account_name		blob コンテナが含まれる Microsoft Azure ストレージアカウントの名前。

### JDBC\_PING の属性

表A.125 JDBC\_PING の属性

属性	デフォルト	説明
data-source		接続および JNDI ルックアッププロパティの代わりに使用されるデータソース参照。



#### 注記

**JDBC\_PING** プロトコルを定義するとき、**protocol** 要素の代わりに **jdbc-protocol** 要素が使用されます。

### S3\_PING の属性

表A.126 S3\_PING の属性

属性	デフォルト	説明
access_key		S3 バケットのアクセスに使用される Amazon S3 アクセスキー。
host	s3.amazonaws.com	S3 web サービスの宛先。
location		使用する Amazon S3 バケットの名前。バケットが存在する必要があり、一意な名前を使用する必要があります。
pre_signed_delete_url		DELETE 操作に使用する事前署名付き URL。
port	<ul style="list-style-type: none"> <li>● <b>use_ssl</b> が <b>true</b> の場合は 443。</li> <li>● <b>use_ssl</b> が <b>false</b> の場合は 80。</li> </ul>	web サービスがリッスンしているポート。
pre_signed_put_url		PUT 操作に使用する事前署名付き URL。
prefix		設定され、 <b>location</b> が設定されている場合、バケット名を <b>PREFIX-LOCATION</b> として定義します。設定され、指定の <b>PREFIX-LOCATION</b> にバケットが存在しない場合は、 <b>PREFIX</b> の後に無作為の UUID が続くバケット名になります。
secret_access_key		S3 バケットのアクセスに使用される Amazon S3 のシークレットアクセスキー。
use_ssl	true	ホストとポートの組み合わせと通信するときに SSL が使用されるかどうかを決定します。

## TCPGOSSIP の属性

表A.127 TCGOSSIP の属性

属性	デフォルト	説明
----	-------	----

属性	デフォルト	説明
socket-binding		このプロトコル層のソケットバインディング仕様。非推奨: 代わりに <b>socket-bindings</b> を使用してください。
socket-bindings		このプロトコルのアウトバウンドソケットバインディング。



### 注記

**TCPGOSSIP** プロトコルを定義するとき、**protocol** 要素の代わりに **socket-discovery-protocol** 要素が使用されます。

## TCPPING の属性

表A.128 TCPPING の属性

属性	デフォルト	説明
socket-binding		このプロトコル層のソケットバインディング仕様。非推奨: 代わりに <b>socket-bindings</b> を使用してください。
socket-bindings		このプロトコルのアウトバウンドソケットバインディング。



### 注記

**TCPPING** プロトコルを定義するとき、**protocol** 要素の代わりに **socket-discovery-protocol** 要素が使用されます。

## 暗号化プロトコル

以下のプロトコルは通信スタックをセキュアにするために使用されます。暗号化はクラスターのすべてのメンバーが持つ共有のシークレットキーを基にします。このキーは、**SYM\_ENCRYPT** を使用している場合は共有のキーストアから取得され、**ASYM\_ENCRYPT** を使用している場合はパブリック/プライベートキーの交換から取得されます。以下のプロトコルのいずれかを定義すると、結果となる XML に **encrypt-protocol** 要素が作成されます。



### 注記

**ASYM\_ENCRYPT** を使用している場合は、同じスタックに **AUTH** プロトコルが定義されている必要があります。**SYM\_ENCRYPT** を使用している場合、**AUTH** プロトコルは任意です。

- [ASYM\\_ENCRYPT](#)
- [SYM\\_ENCRYPT](#)

## ASYM\_ENCRYPT の属性

表A.129 ASYM\_ENCRYPT の属性

属性	デフォルト	説明
key-alias		指定されたキーストアから取得される暗号化キーのエイリアス。
key-credential-reference		キーストアから暗号化キーを取得するために必要な認証情報。
key-store		暗号化キーが含まれるキーストアへの参照。

## SYM\_ENCRYPT の属性

表A.130 SYM\_ENCRYPT の属性

属性	デフォルト	説明
key-alias		指定されたキーストアから取得される暗号化キーのエイリアス。
key-credential-reference		キーストアから暗号化キーを取得するために必要な認証情報。
key-store		暗号化キーが含まれるキーストアへの参照。

## 障害検出プロトコル

以下のプロトコルは、クラスターのメンバーを調査して生存の有無を判断するために使用されます。これらのプロトコルには、汎用属性以外の属性はありません。

- FD\_ALL
- FD\_SOCKET
- VERIFY\_SUSPECT

## フロー制御プロトコル

以下のプロトコルは、メッセージの送信側の速度を最も遅い受信側に合わせて処理するフロー制御を行います。送信側が送信するメッセージの速度が継続して受信側よりも速い場合、受信側はメッセージをキューに置くか破棄するため、再送信が発生します。これらのプロトコルには、汎用属性以外の属性はありません。

- MFC - マルチキャストフロー制御
- UFC - ユニキャストフロー制御

## グループメンバーシッププロトコル

**pbcast.GMS** プロトコルは、新規メンバーのクラスターへの参加、既存メンバーによるクラスターからの離脱、およびクラッシュした疑いのあるメンバーに対応します。このプロトコルには、汎用属性以外の属性はありません。

## マージプロトコル

クラスターが分割された場合、**MERGE3** プロトコルによってサブクラスターが元どおりにマージされます。このプロトコルはクラスターメンバーを元どおりにマージする役割を果たしますが、クラスターの状態はマージしません。マージ状態のコールバックはアプリケーションが対応します。このプロトコルには、汎用属性以外の属性はありません。

### メッセージの安定性

**pbcast.STABLE** プロトコルは、クラスターのすべてのメンバーが受け取ったメッセージをガベージコレクションで処理します。このプロトコルは、ダイジェストと呼ばれるメンバーのメッセージ番号が含まれる安定したメッセージを送信します。クラスターのすべてのメンバーが他のメンバーのダイジェストを受信した後、メッセージを再送信テーブルから削除することができます。このプロトコルには、汎用属性以外の属性はありません。

### 信頼できるメッセージの送信

以下のプロトコルは、信頼できるメッセージの配信と、クラスターのすべてのノードに送信されたメッセージの FIFO プロパティを提供します。信頼できる配信では、すべてのメッセージに番号が付けられ、シーケンス番号を受信されなかった場合に再送信リクエストが送信されるため、送信元が送ったメッセージを損失することはありません。これらのプロトコルには、汎用属性以外の属性はありません。

- pbcast.NAKACK2
- pbcast.UNICAST3

### 非推奨となったプロトコル

以下のプロトコルは非推奨となり、クラス名のみが含まれるプロトコルによって置き換えられました。たとえば、**org.jgroups.protocols.ASYM\_ENCRYPT** を指定する代わりに、プロトコル名は **ASYM\_ENCRYPT** になります。

- org.jgroups.protocols.ASYM\_ENCRYPT
- org.jgroups.protocols.AUTH
- org.jgroups.protocols.JDBC\_PING
- org.jgroups.protocols.SYM\_ENCRYPT
- org.jgroups.protocols.TCPGOSSIP
- org.jgroups.protocols.TCPPING

## A.35. MICROPROFILE CONFIG SMALLRYE 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-microprofile-config-smallrye\_1\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.131 MicroProfile Config SmallRye サブシステム: **config-source** 属性

属性	説明
----	----

属性	説明
class	ロードする <b>ConfigSource</b> クラス。 <b>ConfigSource</b> は、設定値のソースを提供します。
dir	スキャンするディレクトリー。このディレクトリーにはプロパティファイルが含まれ、各ファイル名がプロパティ名で、ファイルの内容がその値となります。
ordinal	<b>ConfigSource</b> の序数值または優先度。ordinal の値が大きい ConfigSources によって提供された値は、ordinal の値が小さい ConfigSources によって提供された値をオーバーライドします。
properties	<b>ConfigSource</b> に直接保存するプロパティ。

表A.132 MicroProfile Config SmallRye サブシステム:config-source-provider 属性

属性	説明
class	ロードする <b>ConfigSourceProvider</b> クラス。 <b>ConfigSourceProvider</b> は、複数の <b>ConfigSource</b> インスタンスの実装を登録します。

## A.36. APACHE HTTP SERVER の MOD\_CLUSTER ディレクティブ

mod\_cluster コネクターは Apache HTTP Server ベースのロードバランサーです。通信チャネルを使用して、リクエストを Apache HTTP Server からアプリケーションサーバーノードのセットの1つに転送します。mod\_cluster の設定には以下のディレクティブを指定できます。



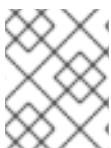
### 注記

mod\_cluster は Apache HTTP Server に転送しなければならない URL を自動的に設定するため、ProxyPass ディレクティブを使用する必要はありません。

表A.133 mod\_cluster ディレクティブ

ディレクティブ	説明	値
---------	----	---

ディレクティブ	説明	値
CreateBalancers	balancer が Apache HTTP Server の VirtualHosts でどのように作成されるかを定義します。 <b>ProxyPass /balancer://mycluster1/</b> のようなディレクティブを許可します。	<ul style="list-style-type: none"> <li>● <b>0</b>: Apache HTTP Server に定義されるすべての VirtualHosts を作成します。</li> <li>● <b>1</b>: balancer を作成しません (balancer 名を定義するため最低でも1つの <b>ProxyPass</b> または <b>ProxyMatch</b> が必要です)。</li> <li>● <b>2</b>: メインサーバーのみ作成します (デフォルト)。</li> </ul>
UseAlias	エイリアスがサーバー名に対応することを確認します。	<ul style="list-style-type: none"> <li>● <b>0</b>: エイリアスを無視します (デフォルト)。</li> <li>● <b>1</b>: エイリアスをチェックします。</li> </ul>
LBstatusRecalTime	負荷分散ロジックがノードの状態を再計算する間隔 (秒単位)。	デフォルト: 5 秒
WaitBeforeRemove	削除されたノードを httpd が記憶しなくなるまでの時間 (分単位)。	デフォルト: 10 秒
ProxyPassMatch/ProxyPass	ProxyPassMatch および ProxyPass は、バックエンド URL の代わりに「!」を使用するとパスのリバースプロキシを防ぐ <b>mod_proxy</b> ディレクティブです。これは、Apache HTTP Server が静的なコンテンツに対応できるようにするために使用されます。例: <b>ProxyPassMatch ^(/.*\.(gif))\$ !</b> この例では、Apache HTTP Server は直接 .gif ファイルに対応できます。	



## 注記

JBoss EAP 7 のセッションのパフォーマンス最適化により、ホットスタンバイノードの設定はサポートされません。

mod\_manager



mod\_manager ディレクティブのコンテキストは、指定がある場合を除きすべて VirtualHost になります。server config コンテキストは、ディレクティブが VirtualHost 設定の外部になければならないことを示します。そうでない場合、エラーメッセージが表示され、Apache HTTP Server が開始しません。

表A.134 mod\_manager ディレクティブ

ディレクティブ	説明	値
EnableMCPMReceive	VirtualHost がノードから MCPM を受信できるようにします。mod_cluster が動作するようにするため、Apache HTTP Server 設定に EnableMCPMReceive が含まれます。VirtualHost のアドバタイズを設定する場所に保存します。	
MemManagerFile	設定の保存、共有メモリーまたはロックされたファイルのキー生成に mod_manager が使用する名前のベース名。絶対パス名である必要があります。ディレクトリーは必要な場合に作成されます。これらのファイルは NFS 共有ではなくローカルドライブに格納することが推奨されます。コンテキスト: server config	<b>\$server_root/logs/</b>
Maxcontext	mod_cluster によってサポートされるコンテキストの最大数。コンテキスト: server config	デフォルト: <b>100</b>
Maxnode	mod_cluster によってサポートされるノードの最大数。コンテキスト: server config	デフォルト: <b>20</b>
Maxhost	mod_cluster によってサポートされるホスト (エイリアス) の最大数。バランサーの最大数も含まれます。コンテキスト: server config	デフォルト: <b>20</b>
Maxsessionid	mod_cluster-manager ハンドラーにアクティブなセッションの数を提供するために保存されるアクティブ sessionid の数。5 分以内に mod_cluster がセッションから情報を受信しないとセッションは非アクティブになります。コンテキスト: server config。このフィールドはデモおよびデバッグの目的のみで使用されます。	<b>0</b> : ロジックはアクティベートされません。

ディレクティブ	説明	値
MaxMCMPMaxMessSize	他の Max ディレクティブからの MCMP メッセージの最大サイズ。	他の Max ディレクティブより計算されます。最小: <b>1024</b>
ManagerBalancerName	JBoss EAP インスタンスが balancer 名を提供しない場合に使用される balancer の名前。	<b>mycluster</b>
PersistSlots	ファイルのノード、エイリアス、およびコンテキストを保持するように mod_slotmem に伝えます。コンテキスト: server config	オフ
CheckNonce	<b>mod_cluster-manager</b> ハンドラーを使用する際に <b>nonce</b> のチェックを切り替えます。	on/off、デフォルト: on - Nonce をチェック
AllowDisplay	<b>mod_cluster-manager</b> メインページの追加表示を切り替えます。	on/off、デフォルト: off - バージョンのみを表示
AllowCmd	<b>mod_cluster-manager</b> URL を使用するコマンドを許可します。	on/off、デフォルト: on - コマンドを許可
ReduceDisplay	メインの <b>mod_cluster-manager</b> ページに表示される情報を減らし、ページ上により多くのノードを表示できるようにします。	on/off、デフォルト: off - 情報をすべて表示
SetHandler mod_cluster-manager	mod_cluster がクラスターから可視できるノードの情報を表示します。情報には一般的な情報が含まれ、追加でアクティブなセッションの数を調べます。  <pre>&lt;Location /mod_cluster-manager&gt;   SetHandler mod_cluster-manager   Require ip 127.0.0.1 &lt;/Location&gt;</pre>	on/off、デフォルト: off



## 注記

`httpd.conf` に定義された場所にアクセスする場合:

- Transferred: バックエンドサーバーに送信された POST データに対応。
- Connected: `mod_cluster` の状態ページが要求されたときに処理された要求の数に対応。
- Num\_sessions: `mod_cluster` がアクティブと報告するセッションの数に対応 (過去 5 分以内に要求があった場合)。Maxsessionid がゼロの場合、このフィールドは存在しません。このフィールドはデモおよびデバッグの目的でのみ使用されます。

## A.37. MODCLUSTER サブシステムの属性

`modcluster` サブシステムの構造は次のとおりです。

- `proxy`
  - `dynamic-load-provider`
    - `custom-load-metric`
    - `load-metric`
  - `ssl`



## 注記

これらの表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を `EAP_HOME/docs/schema/jboss-as-mod-cluster_3_0.xsd` のスキーマ定義ファイルで確認してください。

表A.135 proxy 設定オプション

属性	デフォルト	説明
<code>advertise</code>	<code>true</code>	マルチキャストベースのアドバタイズメカニズムを有効にするかどうか。
<code>advertise-security-key</code>		<code>httpd</code> インスタンスと、 <code>httpd</code> インスタンスからのアドバタイズをリッスンする JBoss EAP サーバーとの間の共有の秘密です。
<code>advertise-socket</code>		登録するリバースプロキシの balancer の名前。
<code>auto-enable-contexts</code>	<code>true</code>	<b>false</b> に設定された場合、コンテキストは無効としてリバースプロキシと登録されます。 <b>enable-context</b> 操作を使用するか、 <code>mod_cluster_manager</code> コンソールを使用するとコンテキストを有効にできます。

属性	デフォルト	説明
balancer		登録するリバースプロキシのバランサーの名前。設定されていない場合、値は <b>ManagerBalancerName</b> ディレクティブで Apache HTTP Server 側に設定され、デフォルトでは <b>mycluster</b> になります。
connector		mod_cluster のリバースプロキシが接続する Undertow リスナーの名前。
excluded-contexts		リバースプロキシでの登録から除外されるコンテキストのリスト。ホストの指定がない場合、 <b>localhost</b> がホストと仮定されます。 <b>ROOT</b> は web アプリケーションのルートコンテキストを示します。
flush-packets	false	web サーバーへのパケットのフラッシングを有効にするかどうか。
flush-wait	-1	httpd でパケットをフラッシュする前に待機する期間。最大値は <b>2,147,483,647</b> 。
listener		リバースプロキシと登録される Undertow リスナーの名前。
load-balancing-group		設定された場合、リクエストはロードバランサーの指定のロードバランシンググループに送信されます。
max-attempts	1	リバースプロキシが指定リクエストのワーカーへの送信を試みる回数。この回数試行した後に送信を断念します。
node-timeout	-1	ワーカーへのプロキシ接続のタイムアウト (秒単位)。mod_cluster はこの期間バックエンド応答を待ち、その経過後にエラーを返します。 <b>node-timeout</b> 属性が未定義である場合、httpd <b>ProxyTimeout</b> ディレクティブが使用されます。 <b>ProxyTimeout</b> が未定義である場合、httpd <b>Timeout</b> ディレクティブが使用され、デフォルトは 300 秒になります。
ping	10	ping に対する pong 返答を待つ時間 (秒単位)。
proxies		<b>socket-binding-group</b> の <b>outbound-socket-binding</b> によって定義される登録する mod_cluster のプロキシのリスト。

属性	デフォルト	説明
proxy-list		プロキシのリスト。形式は <b>HOST_NAME:PORT</b> で、カンマで区切られます。 <b>非推奨になりました。proxies の使用が推奨されます。</b>
proxy-url	/	MCMP リクエストのベース URL。
session-draining-strategy	DEFAULT	<p>Web アプリケーションのアンデプロイメント中に使用されるセッションドレインストラテジー。有効な値は <b>DEFAULT</b>、<b>ALWAYS</b>、または <b>NEVER</b> です。</p> <p><b>DEFAULT</b> web アプリケーションが分散可能でない場合のみ、web アプリケーションのアンデプロイ前にセッションをドレインします。</p> <p><b>ALWAYS</b> web アプリケーションが分散可能であっても、web アプリケーションのアンデプロイ前に常にセッションをドレインします。</p> <p><b>NEVER</b> web アプリケーションのアンデプロイ前にセッションをドレインしません。</p>
simple-load-provider		動的ロードプロバイダーが存在しない場合に使用する簡単なロードプロバイダー。各クラスターメンバーに負荷係数 <b>1</b> を割り当て、負荷分散アルゴリズムを適用せずに作業を均等に分散します。
smax	-1	httpd の soft maximum アイドル接続数。
socket-timeout	20	タイムアウトの前およびプロキシをエラーのように警告する前に、httpd プロキシから MCMP コマンドへの応答を待つ秒数。
ssl-context		mod_cluster によって使用される <b>SSLContext</b> への参照。
status-interval	10	STATUS メッセージがアプリケーションサーバーからリバースプロキシへ送信される秒数。 <b>-1</b> から <b>2,147,483,647</b> までの値が許可されます。
sticky-session	true	あるセッションの後続リクエストを可能な限り同じノードヘルレーティングするべきかどうか。
sticky-session-force	false	balancer がリクエストをスタックしたノードヘルレーティングできない場合に、リバースプロキシがエラーを返すかどうか。スティッキーセッションが無効な場合は無視されます。

属性	デフォルト	説明
sticky-session-remove	false	フェイルオーバー時にセッション情報を削除します。
stop-context-timeout	10	分散可能なコンテキストの場合は、コンテキストが保留中のリクエストを処理するのを待つ最大時間 (秒単位)。分散可能でないコンテキストの場合は、コンテキストがアクティブなセッションを破棄するのを待つ最大時間 (秒単位)。
ttl	-1	smax を超えるアイドル接続の TTL (Time To Live、秒単位)。-1 から <b>2,147,483,647</b> までの値が許可されます。
worker-timeout	-1	httpd で利用可能なワーカーによるリクエスト処理の待機時間のタイムアウト値。-1 から <b>2,147,483,647</b> までの値が許可されます。

表A.136 dynamic-load-provider 設定オプション

属性	デフォルト	説明
decay	2	減退 (Decay)。
history	9	履歴。

表A.137 custom-load-metric 属性オプション

属性	デフォルト	説明
capacity	1.0	メトリックの容量
class		カスタムメトリックのクラス名。
property		メトリックのプロパティ。
weight	1	メトリックの重さ。

表A.138 load-metric 属性オプション

属性	デフォルト	説明
capacity	1.0	メトリックの容量
property		メトリックのプロパティ。

属性	デフォルト	説明
type		メトリックのタイプ。有効な値は <b>cpu</b> 、 <b>mem</b> 、 <b>heap</b> 、 <b>sessions</b> 、 <b>receive-traffic</b> 、 <b>send-traffic</b> 、 <b>requests</b> 、または <b>busyness</b> です。
weight	1	メトリックの重さ。

表A.139 ssl 属性オプション

属性	デフォルト	説明
ca-certificate-file		認証局。
ca-revocation-url		認証局の失効リスト。
certificate-key-file	\${user.home}/.keystore	証明書のキーファイル。
cipher-suite		許可された暗号スイート。
key-alias		キーエイリアス。
password	changeit	パスワード。
protocol	TLS	有効な SSL プロトコル。

## A.38. MOD\_JK ワーカープロパティ

**workers.properties** ファイルは mod\_jk がクライアント要求を渡すワーカーの動作を定義します。**workers.properties** ファイルは、異なるサブレットコンテナが存在する場所と、ワークロードをアプリケーションサーバーすべてで分散する方法を定義します。

プロパティの一般的な構造は **worker.WORKER\_NAME.DIRECTIVE** です。**WORKER\_NAME** は、JBoss EAP **undertow** サブシステムで設定された **instance-id** と一致しなければならない一意な名前です。**DIRECTIVE** はワーカーに適用される設定です。

### Apache mod\_jk ロードバランサーの設定リファレンス

テンプレートはデフォルトのロードバランサーごとの設定を指定します。ロードバランサーの設定内でテンプレートを上書きできます。

表A.140 グローバルプロパティ

プロパティ	説明
worker.list	mod_jk によって使用されるワーカー名のカンマ区切りリスト。

表A.141 必須ディレクティブ

プロパティ	説明
type	ワーカーのタイプ。デフォルトのタイプは <b>ajp13</b> です。他の可能な値は <b>ajp14</b> 、 <b>lb</b> 、 <b>status</b> です。これらのディレクティブの詳細は、 <a href="https://tomcat.apache.org/connectors-doc/reference/workers.html">https://tomcat.apache.org/connectors-doc/reference/workers.html</a> の『Apache Tomcat Connectors Reference』を参照してください。

表A.142 負荷分散ディレクティブ

プロパティ	説明
balance_workers	ロードバランサーが管理する必要があるワーカーノードを指定します。同じロードバランサーにディレクティブを複数回使用できます。カンマ区切りのワーカーノード名のリストで構成されます。
sticky_session	同じセッションからのリクエストを常に同じワーカーにルーティングするかどうかを指定します。デフォルトは <b>1</b> で、スティッキーセッションが有効になります。スティッキーセッションを無効にするには <b>0</b> を設定します。すべてのリクエストが実際にステートレスである場合を除き、スティッキーセッションは通常有効にする必要があります。

表A.143 接続ディレクティブ

プロパティ	説明
host	バックエンドサーバーのホスト名または IP アドレス。バックエンドサーバーは <b>ajp</b> プロトコルスタックをサポートする必要があります。デフォルト値は <b>localhost</b> です。
port	定義されたプロトコルリクエストをリッスンしているバックエンドサーバーインスタンスのポート番号。デフォルトの値は、AJP13 ワーカーのデフォルトのリッスンポートである <b>8009</b> です。AJP14 ワーカーのデフォルト値は <b>8011</b> です。



プロパティ	説明
ping_mode	<p>ネットワークの状態に対して接続がプローブされる条件。プローブはCPing に空の AJP13 パケットを使用し、応答で CPong を想定します。ディレクティブフラグの組み合わせを使用して条件を指定します。フラグはコンマまたはスペースで区切られません。ping_mode は C、P、I、および A の任意の組み合わせです。</p> <ul style="list-style-type: none"> <li>● C - Connect (接続)。サーバーへの接続後に1回接続をプローブします。connect_timeout の値を使用してタイムアウトを指定します。指定がないと、ping_timeout の値が使用されます。</li> <li>● P - Prepost (プレポスト)。各リクエストをサーバーに送信する前に接続をプローブします。prepost_timeout ディレクティブを使用してタイムアウトを指定します。指定がないと、ping_timeout の値が使用されます。</li> <li>● I - Interval (間隔)。connection_ping_interval で指定された間隔で接続をプローブします (指定がある場合)。指定がないと、ping_timeout の値が使用されます。</li> <li>● A - All (すべて)。すべての接続プローブを使用することを指定する CPI のショートカットです。</li> </ul>
ping_timeout、connect_timeout、prepost_timeout、connection_ping_interval	<p>上記の接続プローブ設定のタイムアウト値。値はミリ秒単位で指定され、<b>ping_timeout</b> のデフォルト値は <b>10000</b> です。</p>
lbfactor	<p>各バックエンドサーバーインスタンスの負荷分散係数を指定します。より強力なサーバーにより多くのワークロードを割り当てる場合に便利です。ワーカーにデフォルトの3倍の負荷を割り当てるには、<b>worker.my_worker.lbfactor=3</b> のように <b>3</b> を設定します。</p>

以下の例は、ポート **8009** でリッスンする **node1** および **node2** の2つのワーカーノードの間でステッキセッションを用いて負荷を分散します。

#### 例: workers.properties ファイル

```
# Define list of workers that will be used for mapping requests
worker.list=loadbalancer,status

# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=node1.mydomain.com
worker.node1.type=ajp13
worker.node1.ping_mode=A
worker.node1.lbfactor=1

# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host= node2.mydomain.com
```

```

worker.node2.type=ajp13
worker.node2.ping_mode=A
worker.node2.lbfactor=1

# Load-balancing behavior
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1

# Status worker for managing load balancer
worker.status.type=status

```

Apache mod\_jk の設定の詳細は、本書の範囲外です。[Apache のドキュメント](#)を参照してください。

## A.39. SECURITY MANAGER サブシステム

**security-manager** サブサブシステム自体には設定可能な属性はありませんが、**deployment-permissions=default** という設定可能な属性を持つ子リソースが1つあります。



### 注記

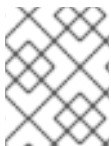
この表は、管理モデルで使用される属性名を示しています (管理 CLI を使用している場合など)。XML で使用される名前は管理モデルの名前と異なる場合があるため、XML で使用される要素を **EAP\_HOME/docs/schema/wildfly-security-manager\_1\_0.xsd** のスキーマ定義ファイルで確認してください。

表A.144 deployment-permissions 設定オプション

属性	説明
maximum-permissions	デプロイメントまたは JAR に付与できる最大パーミッションセット。
minimum-permissions	デプロイメントまたは JAR に付与できる最小パーミッションセット。

## A.40. JBOSS CORE SERVICES からの OPENSSL のインストール

JBoss Core Services OpenSSL のファイルは、[ZIP](#) または [RPM](#) ディストリビューションからインストールできます。インストールの方法に応じて、以下の手順にしたがってください。



### 注記

Red Hat Enterprise Linux 8 では、標準のシステム OpenSSL に対応しているため、JBoss Core Services からの OpenSSL のインストールは必要ありません。

### JBoss Core Services OpenSSL ZIP ファイルディストリビューションの使用



## 注記

ZIP アーカイブの **libs/** ディレクトリーへのパスは、Linux の場合は **jbcs-openssl-VERSION/openssl/lib(64)**、Windows の場合は **jbcs-openssl-VERSION/openssl/bin** になります。

1. お使いのオペレーティングシステムとアーキテクチャーに該当する OpenSSL パッケージを [Software Downloads](#) ページからダウンロードします。
2. ダウンロードした ZIP ファイルをインストールディレクトリーで展開します。
3. OpenSSL ライブラリーのある場所を JBoss EAP に通知します。  
これを行うには、以下の方法の1つを使用します。以下のコマンドでは、必ず **JBCS\_OPENSSL\_PATH** を JBoss Core Services OpenSSL ライブラリーへのパスに置き換えてください (例: `/opt/rh/jbcs-httpd24/root/usr/lib64`)。
  - 以下の引数を使用すると、OpenSSL パスを **standalone.conf** または **domain.conf** 設定ファイルの **JAVA\_OPTS** 変数に追加できます。

```
JAVA_OPTS="$JAVA_OPTS -Dorg.wildfly.openssl.path=JBCS_OPENSSL_PATH
```

- 以下の管理 CLI コマンドを使用すると、OpenSSL パスを指定するシステムプロパティーを定義できます。

```
/system-property=org.wildfly.openssl.path:add(value=JBCS_OPENSSL_PATH)
```



## 重要

使用する方法に関係なく、サーバーを再起動して **JAVA\_OPTS** の値またはシステムプロパティーを有効にする必要があります。サーバーをリロードするだけでは有効にできません。

## JBoss Core Services OpenSSL RPM ディストリビューションの使用

1. システムが JBoss Core Services チャンネルに登録されていることを確認してください。
  - a. オペレーティングシステムバージョンとアーキテクチャーの JBoss Core Services の CDN リポジトリー名を決定します。
    - **RHEL 6**: `jb-coreservices-1-for-rhel-6-server-rpms`
    - **RHEL 7**: `jb-coreservices-1-for-rhel-7-server-rpms`
  - b. システムでリポジトリーを有効にします。

```
# subscription-manager repos --enable REPO_NAME
```

- c. 以下のメッセージが表示されたことを確認してください。

```
Repository REPO_NAME is enabled for this system.
```

2. このチャンネルから OpenSSL をインストールします。

```
# yum install jbcs-httpd24-openssl
```

3. インストールの完了後、`/opt/rh/jbcs-httpd24/root/usr/lib64` の JBCS OpenSSL ライブラリーを利用できます。x86 アーキテクチャーでは `/opt/rh/jbcs-httpd24/root/usr/lib` になります。
4. OpenSSL ライブラリーのある場所を JBoss EAP に通知します。  
これを行うには、以下の方法の1つを使用します。以下のコマンドでは、必ず **JBCS\_OPENSSL\_PATH** を JBoss Core Services OpenSSL ライブラリーへのパスに置き換えてください (例: `/opt/rh/jbcs-httpd24/root/usr/lib64`)。
  - サービス設定ファイルの **eap7-standalone** または **eap7-domain** 設定の **WILDFLY\_OPTS** 変数を更新できます。

```
WILDFLY_OPTS="$WILDFLY_OPTS -
Dorg.wildfly.openssl.path=JBCS_OPENSSL_PATH"
```

- 以下の管理 CLI コマンドを使用すると、OpenSSL パスを指定するシステムプロパティーを定義できます。

```
/system-property=org.wildfly.openssl.path:add(value=JBCS_OPENSSL_PATH)
```



### 重要

使用する方法に関係なく、サーバーを再起動して **WILDFLY\_OPTS** の値またはシステムプロパティーを有効にする必要があります。サーバーをリロードするだけでは有効にできません。

## A.41. OPENSSL を使用するよう JBOSS EAP を設定

JBoss EAP が OpenSSL を使用するよう設定する方法は複数あります。

- **elytron** サブシステムを再設定し、OpenSSL の優先度を上げて、OpenSSL がデフォルトですべての場合で使用されるようにすることができます。



### 注記

OpenSSL は **elytron** サブシステムにインストールされていますが、デフォルトの TLS プロバイダーではありません。

```
/subsystem=elytron:write-attribute(name=initial-providers, value=combined-providers)
/subsystem=elytron:undefine-attribute(name=final-providers)

reload
```

- **elytron** サブシステムでは、OpenSSL プロバイダーは **ssl-context** リソースでも指定できます。これにより、デフォルトの優先度を使用せずに、OpenSSL プロトコルを状況に応じて選択できます。  
**ssl-context** リソースを作成し、Elytron ベースの SSL/TLS 設定で OpenSSL ライブラリーを使用するには、以下のコマンドを使用します。

```
/subsystem=elytron/server-ssl-context=httpsSSC:add(key-manager=localhost-manager,
trust-manager=ca-manager, provider-name=openssl)

reload
```

- レガシーの **security** サブシステムの SSL/TLS 設定で OpenSSL ライブラリーを使用するには、以下のコマンドを使用します。

```
/core-service=management/security-realm=ApplicationRealm/server-identity=ssl:write-attribute(name=protocol,value=openssl.TLSv1.2)

reload
```

使用可能な OpenSSL プロトコルは次のとおりです。

- openssl.TLS
- openssl.TLSv1
- openssl.TLSv1.1
- openssl.TLSv1.2

JBoss EAP は自動的にシステム上の OpenSSL ライブラリーの検索を行い、それを使用します。JBoss EAP の起動中に **org.wildfly.openssl.path** プロパティーを使用すると、カスタム OpenSSL ライブラリーの場所を指定することもできます。JBoss Core Services によって提供される OpenSSL ライブラリーのバージョン 1.0.2 以上のみがサポートされます。

OpenSSL が適切にロードされると、JBoss EAP の起動中に以下と似たメッセージが **server.log** に出力されます。

```
15:37:59,814 INFO [org.wildfly.openssl.SSL] (MSC service thread 1-7) WFOPENSSL0002 OpenSSL Version OpenSSL 1.0.2k-fips 23 Mar 2017
```

## A.42. JAVA 8 向けに提供されるプラットフォームモジュール

- java.base**: この依存関係は提供されるモジュールローダーに常に含まれます。
- java.compiler**
- java.datatransfer**
- java.desktop**
- java.instrument**
- java.jnlp**
- java.logging**
- java.management**
- java.management.rmi**
- java.naming**
- java.prefs**
- java.rmi**
- java.scripting**

- **java.se**: このモジュールエイリアスは以下のベースモジュールのセットを集約します。
  - **java.compiler**
  - **java.datatransfer**
  - **java.desktop**
  - **java.instrument**
  - **java.logging**
  - **java.management**
  - **java.management.rmi**
  - **java.naming**
  - **java.prefs**
  - **java.rmi**
  - **java.scripting**
  - **java.security.jgss**
  - **java.security.sasl**
  - **java.sql**
  - **java.sql.rowset**
  - **java.xml**
  - **java.xml.crypto**
- **java.security.jgss**
- **java.security.sasl**
- **java.smartcardio**
- **java.sql**
- **java.sql.rowset**
- **java.xml**
- **java.xml.crypto**
- **javafx.base**
- **javafx.controls**
- **javafx.fxml**
- **javafx.graphics**

- `javafx.media`
- `javafx.swing`
- `javafx.web`
- `jdk.accessibility`
- `jdk.attach`
- `jdk.compiler`
- `jdk.httpserver`
- `jdk.jartool`
- `jdk.javadoc`
- `jdk.jconsole`
- `jdk.jdi`
- `jdk.jfr`
- `jdk.jsobject`
- `jdk.management`
- `jdk.management.cmm`
- `jdk.management.jfr`
- `jdk.management.resource`
- `jdk.net`
- `jdk.plugin.dom`
- `jdk.scripting.nashorn`
- `jdk.sctp`
- `jdk.security.auth`
- `jdk.security.jgss`
- `jdk.unsupported`
- `jdk.xml.dom`

