Red Hat JBoss Enterprise Application Platform 7.1

管理 CLI ガイド

Red Hat JBoss Enterprise Application Platform 7.1 向け

Last Updated: 2018-09-03

Red Hat JBoss Enterprise Application Platform 7.1 管理 CLI ガイド

Red Hat JBoss Enterprise Application Platform 7.1 向け

法律上の通知

Copyright © 2018 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

本ガイドは、管理 CLI を使用して Red Hat JBoss Enterprise Application Platform を管理する方法について取り上げます。

目次

第1章 管理 CLI の概要 5

第2章 管理 CLI の使用 6
2.1. 管理 CLI の起動 6
2.2. サーバーへの接続 6
2.3. 現在のパスの変更 7
2.4. コンテンツの一覧表示 7
2.5. ヘルプ 8
2.6. 管理 CLI の終了 8
2.7. 非対話モードでの実行 9
コマンドを渡す 9
コマンドのファイルを渡す 9

第3章 リクエストの作成および実行 11
操作リクエストの作成 11
3.1. リソース値の表示 13
ランタイム属性の追加 13
子リソースの再帰的な読み取り 14
デフォルト値の除外 14
式の解決 15
3.2. リソースの詳細表示 16
3.3. 属性値の表示 17
3.4. 属性の更新 18
3.5. 属性の定義削除 18
3.6. 操作名の表示 19
3.7. 操作の詳細表示 20
3.8. 特殊文字を用いた値の追加 20
空白文字 21
引用符 21
コンマ 21
かっこ 21
中かっこ 21
角かっこ 22
発音区別符号 22
3.9. 操作ヘッダーの指定 22
3.10. IF-ELSE 制御フローの使用 23
3.11. TRY-CATCH-FINALLY 制御フローの使用 25
3.12. リソースのクエリー 25

第4章 管理対象ドメインでの管理 CLI の使用 28
サブシステム設定のプロファイル指定 28
コア管理およびランタイムコマンドのホスト指定 28
コア管理およびランタイムコマンドのサーバー指定 29

第5章 管理 CLI の設定 30
5.1. プロパティーの置換 32
5.2. エイリアスの作成 33
5.3. .JBOSSCLIRC 設定ファイル 34
.jbossclirc ファイルの例 34
5.4. 変数の使用 34
set コマンドの使用 34
unset コマンドの使用 35

jbossclirc ファイルの使用 35
echo コマンドの使用 35
例 35

第6章 管理 CLI のコマンド履歴 37
管理 CLI コマンド履歴の表示 37
管理 CLI コマンド履歴の消去 37
管理 CLI コマンド履歴の有効化 37
管理 CLI コマンド履歴の無効化 37

第7章 管理 CLI のロギング 38
管理 CLI ロギングの設定 38

第8章 バッチ処理 39
外部ファイルの batch コマンド 39

第9章 オフライン設定でのサーバーの埋め込み 40
埋め込みスタンドアロンサーバーの起動 40
サーバー設定の指定 40
admin-only モードの開始 40
標準出力の制御 41
ブートタイムアウト 41
空の設定での開始 41
埋め込みホストコントローラーの開始 41
ホストコントローラー設定の指定 42
標準出力の制御 42
ブートタイムアウト 42
管理 CLI での非モジュラークラスローティング 42

第10章 ハウツー集 44
10.1. データソースの追加 44
10.2. 拡張機能の追加 44
10.3. JMS キューの追加 44
10.4. JMS トピックの追加 44
10.5. モジュールの追加 44
10.6. サーバーの追加 45
10.7. サーバーグループの追加 45
10.8. システムプロパティーの追加 45
10.9. プロファイルのクローン 45
10.10. 階層プロファイルの作成 45
10.11. アプリケーションの管理対象ドメインへのデプロイ 45
10.12. 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 46
10.13. すべてのアプリケーションを無効化 46
10.14. アクティブなユーザーの表示 46
10.15. 添付内容の表示 46
10.16. スキーマ情報の表示 47
10.17. システムおよびサーバー情報の表示 47
10.18. コマンドタイムアウト値の取得 48
10.19. 無効なデプロイメントをすべて再デプロイ 48
10.20. ホストコントローラーのリロード 48
10.21. ADMIN-ONLY モードでのホストコントローラーのリロード 48
10.22. サーバーグループのすべてのサーバーをリロード 48
10.23. サーバーのリロード 49
10.24. スタンドアロンサーバーのリロード 49

10.25. 拡張機能の削除 49
10.26. モジュールの削除 49
10.27. コマンドのタイムアウト値のリセット 50
10.28. サーバーグループのすべてのサーバーを再起動 50
10.29. サーバーの再起動 50
10.30. 添付内容の保存 51
10.31. コマンドのタイムアウト値の設定 51
10.32. ホストコントローラーのシャットダウン 51
10.33. サーバーのシャットダウン 51
10.34. サーバーグループのすべてのサーバーの起動 52
10.35. サーバーの起動 52
10.36. サーバーグループのすべてのサーバーの停止 52
10.37. サーバーの停止 52
10.38. 設定スナップショットの作成 52
10.39. すべてのアプリケーションのアンデプロイ 52
10.40. 管理対象ドメインからのアプリケーションのアンデプロイ 52
10.41. スタンドアロンサーバーからのアプリケーションのアンデプロイ 53
10.42. ホスト名の更新 53
10.43. 添付のアップロード 53
10.44. サーバーログの表示 53

付録A リファレンス資料 54
A.1. 管理 CLI の起動時の引数 54
A.2. 管理 CLI のバッチモードコマンド 55
A.3. 管理 CLI コマンド 56
A.4. 管理 CLI 操作 59
A.5. リソース属性の詳細 62

第1章 管理 CLI の概要

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

管理 CLI を使用して、サーバーの起動および停止、アプリケーションのデプロイおよびアンデプロイ、システムの設定、他の管理タスクの実行を行います。操作はバッチモードで実行でき、複数のタスクをグループとして実行できます。

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

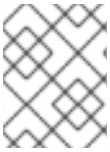
第2章 管理 CLI の使用

管理 CLI は JBoss EAP ディストリビューションに含まれています。管理 CLI を 起動 すると、稼働中のサーバーインスタンスや管理対象ドメインに 接続 して 管理操作を実行 することができます。

2.1. 管理 CLI の起動

管理 CLI を起動するには、JBoss EAP に提供される **jboss-cli** スクリプトを実行します。

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



注記

Windows サーバーでは、**EAP_HOME\bin\jboss-cli.bat** スクリプトを使用して、管理 CLI を起動します。

管理 CLI を起動し、**--connect** 引数を使用して 1 度にサーバーへ接続する方法の詳細は、「サーバーへの接続」を参照してください。

重要

jboss-cli スクリプトは **com.ibm.jsse2.overrideDefaultTLS** プロパティーを **true** に設定します。Elytron によって設定された SSL を使用し、IBM JDK を使用して認証の問題を回避する場合、この設定は重要になります。

たとえば、**EAP_HOME/bin/client/jboss-cli-client.jar** で利用可能なクラスをプログラムで使用する場合など、IBM JDK を使用し、他のメソッドを使用して CLI セッションを開始する場合、このプロパティーを必ず設定するようにしてください。

利用可能なすべての **jboss-cli** スクリプト引数とそれらの目的の完全リストは、**--help** 引数を使用するか、「管理 CLI の起動時の引数」を参照してください。

2.2. サーバーへの接続

connect コマンドを使用すると、稼働中のスタンドアロンサーバーまたは管理対象ドメインに接続できます。

```
connect
```

デフォルトのホストおよびポート設定は **localhost:9990** です。サーバーが別のホストおよびポートをリッスンしている場合、これらを **connect** コマンドで指定する必要があります。

```
connect 192.168.0.1:9991
```

また、**--connect** 引数 (および必要な場合は **--controller** 引数) を使用して、管理 CLI を起動し、サーバーに一度に接続することもできます。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --controller=192.168.0.1:9991
```

JBoss EAP 7.1 で **http-remoting** プロトコルを使用して接続するには、以下を実行します。

```
connect http://192.168.0.1:9999
```

2.3. 現在のパスの変更

別のノードパスに変更するには、**cd** コマンドを使用し、希望のパスを指定します。管理 CLI が最初に起動されると、これはルートレベル (**/**) になります。

```
cd /subsystem=datasources
cd data-source=ExampleDS
```

pwd コマンドを使用すると現在のノードの完全パスを表示できます。

```
pwd
```

```
/subsystem=datasources/data-source=ExampleDS
```

2.4. コンテンツの一覧表示

ls コマンドを使用すると、特定ノードパスのコンテンツをリストすることができます。パスがノード名で終わる場合、そのリソースの属性も表示されます。

以下の例は、**standard-sockets** ソケットバインディンググループを探し、そのコンテンツをリストします。

```
cd /socket-binding-group=standard-sockets
ls -l
```

```
ATTRIBUTE         VALUE                                  TYPE
default-interface public                                 STRING
name              standard-sockets                       STRING
port-offset       ${jboss.socket.binding.port-offset:0} INT

CHILD                                        MIN-OCCURS MAX-OCCURS
local-destination-outbound-socket-binding  n/a        n/a
remote-destination-outbound-socket-binding n/a        n/a
socket-binding                               n/a        n/a
```

ls コマンドにノードパスを指定すると、リソースツリー階層のどこからでも同じ結果を得ることができます。

```
ls -l /socket-binding-group=standard-sockets
```

```
ATTRIBUTE         VALUE                                  TYPE
default-interface public                                 STRING
name              standard-sockets                       STRING
port-offset       ${jboss.socket.binding.port-offset:0} INT

CHILD                                        MIN-OCCURS MAX-OCCURS
local-destination-outbound-socket-binding  n/a        n/a
remote-destination-outbound-socket-binding n/a        n/a
socket-binding                               n/a        n/a
```

また、 **--resolve-expressions** パラメーターを使用して、返された属性の式をサーバーの値に解決することもできます。

```
ls -l /socket-binding-group=standard-sockets --resolve-expressions
```

```
ATTRIBUTE         VALUE            TYPE
default-interface public           STRING
name              standard-sockets STRING
port-offset       0                INT

CHILD                                        MIN-OCCURS MAX-OCCURS
local-destination-outbound-socket-binding  n/a        n/a
remote-destination-outbound-socket-binding n/a        n/a
socket-binding                               n/a        n/a
```

この例では、**port-offset** 属性は式 (**${jboss.socket.binding.port-offset:0}**) の代わりに解決された値 (**0**) を表しています。

2.5. ヘルプ

管理 CLI を使用してヘルプを表示する方法は複数あります。

- 管理 CLI の使用に関する一般的なヘルプの内容を表示します。

  ```
  help
  ```

  起動、移動、および生成操作リクエストに関する詳細なヘルプを表示します。

- 特定コマンドのヘルプを表示します。

  ```
  COMMAND_NAME --help
  ```

  特定コマンドの使用法、説明、および引数を表示します。

- 利用可能なコマンドのリストを表示します。

  ```
  help --commands
  ```

  注記

  スタンドアロンサーバーまたはドメインコントローラーへ接続する必要があるコマンドは、接続が確立されていないとリストには表示されません。

管理 CLI コマンドのリストは、「管理 CLI コマンド」を参照してください。

2.6. 管理 CLI の終了

quit コマンドを入力すると、管理 CLI を終了できます。

```
quit
```

2.7. 非対話モードでの実行

管理 CLI を起動せず、管理 CLI と対話しなくても、管理 CLI コマンドを実行できます。これは、コマンドのバッチ処理や、スクリプトからのコマンド実行に便利です。**jboss-cli** 起動スクリプトに コマンドを渡すか、コマンドが含まれるファイルを渡す ことが可能です。

コマンドを渡す

--command 引数を使用すると、実行する単一の CLI コマンドを指定できます。コマンドの完了後、管理 CLI が終了します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --command="/interface=public:read-attribute(name=inet-address,resolve-expressions=true)"
```

指定された各コマンドの出力は実行時に表示されます。

```
{
    "outcome" => "success",
    "result" => "127.0.0.1"
}
```

--commands 引数を使用して、実行する CLI コマンドのカンマ区切りリストを指定することもできます。

コマンドのファイルを渡す

--file 引数を使用して、実行する CLI コマンドのテキストファイルを渡すことができます。このファイルでは、各コマンドを行ごとに指定します。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/cli_commands.txt
```

ファイルの各コマンドからの出力は、実行時の表示されます。

出力例

```
{
    "outcome" => "success",
    "result" => "NORMAL"
}
helloworld.war
```

--echo-command 引数を使用すると、出力にプロンプトとコマンドを含めることができます。これは、出力と実行したコマンドを照合して障害を解決するときに便利です。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=/path/to/cli_commands.txt --echo-command
```

コマンドと出力は、コマンドが実行されると表示されます。

コマンドエコーによる出力の例

```
[standalone@localhost:9990 /] :read-attribute(name=running-mode)
{
    "outcome" => "success",
    "result" => "NORMAL"
```

```
}
[standalone@localhost:9990 /] ls /deployment
helloworld.war
```

第3章 リクエストの作成および実行

JBoss EAP の設定は、アドレス可能なリソースの階層ツリーとして示され、アドレス可能なリソースはそれぞれ独自の操作セットを提供します。管理 CLI 操作のリクエストによって、管理モデルとの低レベルな対話が可能になり、制御された状態でサーバー設定を編集する方法を提供します。

操作リクエストは以下の書式を使用します。

```
/NODE_TYPE=NODE_NAME:OPERATION_NAME(PARAMETER_NAME=PARAMETER_VALUE)
```

操作リクエストは 3 つの部分で構成されます。

アドレス

アドレスは、操作を実行するリソースノードを指定します。**NODE_TYPE** は要素名にマップし、**NODE_NAME** は設定 XML にあるその要素の **name** 属性にマップします。リソースツリーの各レベルは、スラッシュ (**/**) によって区切られます。

操作名

リソースノードで実行する操作。コロン (**:**) が最初に付けられます。

パラメーター

操作によって異なる必須または任意のパラメーターのセット。これらのパラメーターはかっこ (**( )**) で囲まれます。

操作リクエストの作成

1. アドレスの特定
   XML 設定ファイル (**standalone.xml**、**domain.xml**、または **host.xml**) を参照すると、必要なアドレスを特定するのに便利です。タブ補完を使用して、利用できるアドレスを表示することもできます。

   ルート (**/**) レベルにあるリソースの一般的なアドレスは次のとおりです。

   - **/deployment=DEPLOYMENT_NAME** - デプロイメントの設定。
   - **/socket-binding-group=SOCKET_BINDING_GROUP_NAME** - ソケットバインディングの設定。
   - **/interface=INTERFACE_NAME** - インターフェースの設定。
   - **/subsystem=SUBSYSTEM_NAME** - スタンドアロンサーバー実行時のサブシステム設定。
   - **/profile=PROFILE_NAME/subsystem=SUBSYSTEM_NAME** - 管理対象ドメイン実行時の選択したプロファイルのサブシステム設定。
   - **/host=HOST_NAME** - 管理対象ドメイン実行時に選択したホストのサーバー設定。

   以下は、**ExampleDS** データソースのアドレスになります。

   ```
   /subsystem=datasources/data-source=ExampleDS
   ```

2. 操作の特定
   利用できる操作は、各型のリソースノードによって異なります。リソースアドレス上で **:read-operation-names** 操作を使用すると、利用可能な操作を表示できます。また、タブ補完を使用することも可能です。

リソースに対する特定操作の情報を取得するには、**:read-operation-description** 操作を使用します。

以下の操作は (適切なパラメーターが含まれた場合)、**ExampleDS** データソースの属性の値を設定します。

```
/subsystem=datasources/data-source=ExampleDS:write-attribute
```

3. パラメーターの特定
   各操作には利用できる独自のパラメータのセットがあります。必要なパラメーターを指定せずに操作を実行しようとすると、パラメーターを **null** にできないという内容のエラーメッセージが表示されます。

   複数のパラメーターはコンマ (**,**) で区切られます。操作にパラメーターがない場合、かっこは任意となります。

   :read-operation-description 操作をリソースで使用し、操作名を渡してその操作に必要なパラメーターを特定します。また、タブ補完を使用して利用できるパラメーターを表示することもできます。

   以下の操作は、**enabled** 属性を **false** に設定して、**ExampleDS** データソースを無効にします。

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

入力後、管理インターフェースはサーバー設定で操作リクエストを実行します。操作のリクエストに応じて、操作の出力と結果、または応答が含まれる出力がターミナルに表示されます。

ExampleDS データソースの無効化に対する以下の応答は、操作に成功し、操作の反映にサーバーのリロードが必要であることを表しています。

```
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

read-attribute 操作を使用すると、**ExampleDS** データソースの **enabled** 属性の値を読み取りできます。

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

以下の応答は、操作に成功し、**enabled** の値が **false** であることを表しています。

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

3.1. リソース値の表示

read-resource 操作を使用して、リソースの属性値を表示できます。

```
:read-resource
```

パラメーターを指定して、子リソースに関する完全情報を再帰的に提供することができます。また、パラメーターを指定して、ランタイム属性の追加、式の解決、およびエイリアスの追加を行うこともできます。**read-operation-description(name=read-resource)** を使用すると、**read-resource** に使用できるすべてのパラメーターの説明を表示できます。

以下の例は、デプロイメントの属性を読み取ります。これには、デプロイメント名、有効あるいは無効であるか、最後に有効になった時間などの詳細が含まれます。

```
/deployment=DEPLOYMENT_NAME:read-resource
{
    "outcome" => "success",
    "result" => {
        ...
        "enabled" => true,
        "enabled-time" => 1453929902598L,
        "enabled-timestamp" => "2016-01-27 16:25:02,598 EST",
        "name" => "DEPLOYMENT_NAME",
        "owner" => undefined,
        "persistent" => true,
        "runtime-name" => "DEPLOYMENT_NAME",
        "subdeployment" => undefined,
        "subsystem" => {
            "undertow" => undefined,
            "logging" => undefined
        }
    }
}
```

ランタイム属性の追加

include-runtime パラメーターを使用するとランタイム属性を取得できます。

以下の例は、デプロイメントの属性を読み取ります。永続属性の他に、デプロイメントの状態や最後に無効になった時間などのランタイム属性も含まれます。

```
/deployment=DEPLOYMENT_NAME:read-resource(include-runtime=true)
{
    "outcome" => "success",
    "result" => {
        ...
        "disabled-time" => undefined,
        "disabled-timestamp" => undefined,
        "enabled" => true,
        "enabled-time" => 1453929902598L,
        "enabled-timestamp" => "2016-01-27 16:25:02,598 EST",
        "name" => "DEPLOYMENT_NAME",
        "owner" => undefined,
        "persistent" => true,
        "runtime-name" => "DEPLOYMENT_NAME",
        "status" => "OK",
```

```
        "subdeployment" => undefined,
        "subsystem" => {
            "undertow" => undefined,
            "logging" => undefined
        }
    }
}
```

また、ブール値パラメータに渡すときに **not** 演算子 (**!**) を使用することもできます。

- **:read-resource(include-runtime=false)** を **:read-resource(!include-runtime)** と入力できます。
- **:read-resource(include-runtime=true)** は **:read-resource(include-runtime)** と入力できます。

子リソースの再帰的な読み取り

recursive パラメーターを使用すると、子リソースから再帰的に属性を読み出しできます。

以下の例は、デプロイメントの属性を読み取ります。リソース独自の属性の他に、**undertow** サブシステムなどの子リソースの属性を再帰的に返します。

```
/deployment=DEPLOYMENT_NAME:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        ...
        "enabled" => true,
        "enabled-time" => 1453929902598L,
        "enabled-timestamp" => "2016-01-27 16:25:02,598 EST",
        "name" => "DEPLOYMENT_NAME",
        "owner" => undefined,
        "persistent" => true,
        "runtime-name" => "DEPLOYMENT_NAME",
        "subdeployment" => undefined,
        "subsystem" => {
            "undertow" => {
                "context-root" => "/test",
                "server" => "default-server",
                "virtual-host" => "default-host",
                "servlet" => undefined,
                "websocket" => undefined
            },
            "logging" => {"configuration" => undefined}
        }
    }
}
```

デフォルト値の除外

include-defaults パラメーターを使用するとリソースの属性の読み取り時にデフォルト値を表示または非表示するにすることができます。デフォルトは **true** で、 **read-resource** 操作の使用時にデフォルト値が表示されます。

以下の例は、**undertow** サブシステムで **read-resource** 操作を使用します。

```
/subsystem=undertow:read-resource
{
    "outcome" => "success",
    "result" => {
        "default-security-domain" => "other",
        "default-server" => "default-server",
        "default-servlet-container" => "default",
        "default-virtual-host" => "default-host",
        "instance-id" => expression "${jboss.node.name}",
        "statistics-enabled" => false,
        "buffer-cache" => {"default" => undefined},
        "configuration" => {
            "filter" => undefined,
            "handler" => undefined
        },
        "server" => {"default-server" => undefined},
        "servlet-container" => {"default" => undefined}
    }
}
```

以下の例も、**undertow** サブシステムで **read-resource** 操作を使用しますが、**include-defaults** パラメーターを **false** に設定します。この例では、**statistics-enabled** や **default-server** などの複数の属性がデフォルト値ではなく **undefined** を表示します。

```
/subsystem=undertow:read-resource(include-defaults=false)
{
    "outcome" => "success",
    "result" => {
        "default-security-domain" => undefined,
        "default-server" => undefined,
        "default-servlet-container" => undefined,
        "default-virtual-host" => undefined,
        "instance-id" => undefined,
        "statistics-enabled" => undefined,
        "buffer-cache" => {"default" => undefined},
        "configuration" => {
            "filter" => undefined,
            "handler" => undefined
        },
        "server" => {"default-server" => undefined},
        "servlet-container" => {"default" => undefined}
    }
}
```

式の解決

resolve-expressions パラメーターを使用すると、返された属性の式をサーバーの値に解決することができます。

式を値として持つ属性は、**${PARAMETER:DEFAULT_VALUE}** という形式を使用します。詳細は、『**設定ガイド**』の「プロパティーの置換」を参照してください。

以下の例は、デプロイメントの属性を読み取ります。**instance-id** 属性は、式 (**${jboss.node.name}**) ではなく、解決された値 (**test-name**) を表示します。

```
/subsystem=undertow:read-resource(resolve-expressions=true)
```

```
{
    "outcome" => "success",
    "result" => {
        "default-security-domain" => "other",
        "default-server" => "default-server",
        "default-servlet-container" => "default",
        "default-virtual-host" => "default-host",
        "instance-id" => "test-name",
        "statistics-enabled" => false,
        "buffer-cache" => {"default" => undefined},
        "configuration" => {
            "filter" => undefined,
            "handler" => undefined
        },
        "server" => {"default-server" => undefined},
        "servlet-container" => {"default" => undefined}
    }
}
```

3.2. リソースの詳細表示

read-resource-description 操作を使用すると、リソースおよびその属性に関する詳細を表示できます。

```
:read-resource-description
```

パラメータを指定して、子リソースに関する完全詳細を再帰的に提供することができます。また、パラメーターを指定して、リソースの操作および通知の詳細を含めることもできます。**read-operation-description(name=read-resource-description)** を使用すると、**read-resource-description** に使用できるすべてのパラメーターの詳細を表示できます。

以下の例は、バッファーキャッシュの属性の詳細を表示します。

```
/subsystem=undertow/buffer-cache=default:read-resource-description
{
    "outcome" => "success",
    "result" => {
        "description" => "The buffer cache used to cache static content",
        "attributes" => {
            "buffer-size" => {
                "type" => INT,
                "description" => "The size of an individual buffer",
                "expressions-allowed" => true,
                "nillable" => true,
                "default" => 1024,
                "min" => 0L,
                "max" => 2147483647L,
                "access-type" => "read-write",
                "storage" => "configuration",
                "restart-required" => "resource-services"
            },
            "buffers-per-region" => {
                "type" => INT,
                "description" => "The numbers of buffers in a region",
                "expressions-allowed" => true,
```

```
                "nillable" => true,
                "default" => 1024,
                "min" => 0L,
                "max" => 2147483647L,
                "access-type" => "read-write",
                "storage" => "configuration",
                "restart-required" => "resource-services"
            },
            "max-regions" => {
                "type" => INT,
                "description" => "The maximum number of regions",
                "expressions-allowed" => true,
                "nillable" => true,
                "default" => 10,
                "min" => 0L,
                "max" => 2147483647L,
                "access-type" => "read-write",
                "storage" => "configuration",
                "restart-required" => "resource-services"
            }
        },
        "operations" => undefined,
        "notifications" => undefined,
        "children" => {}
    }
}
```

属性に対して返されたフィールドの詳細については、「リソース属性の詳細」を参照してください。

3.3. 属性値の表示

read-attribute 操作を使用すると、1 つの属性の現在の値を表示できます。

```
:read-attribute(name=ATTRIBUTE_NAME)
```

以下の例は、**level** 属性を読み取って、ルートロガーのログレベルを表示します。

```
/subsystem=logging/root-logger=ROOT:read-attribute(name=level)
{
    "outcome" => "success",
    "result" => "INFO"
}
```

read-attribute 操作を使用する利点の 1 つは、属性の現在のランタイム値を公開できることです。

```
/interface=public:read-attribute(name=resolved-address)
{
    "outcome" => "success",
    "result" => "127.0.0.1"
}
```

resolved-address 属性はランタイム属性です。この属性は、**include-runtime** パラメーターに渡さないと、**read-resource** 操作の使用時に表示されません。表示された場合でも、リソースの他の属性とともに表示されます。

include-defaults および **resolve-expressions** パラメーターを使用することもできます。これらのパラメーターに関する詳細は、「リソース値の表示」を参照してください。

3.4. 属性の更新

write-attribute 操作を使用して、リソースの属性の値を更新できます。

```
:write-attribute(name=ATTRIBUTE_NAME, value=ATTRIBUTE_VALUE)
```

以下の例は、**scan-enabled** 属性を **false** に設定して、デプロイメントスキャナーを無効にします。

```
/subsystem=deployment-scanner/scanner=default:write-attribute(name=scan-
enabled,value=false)
{"outcome" => "success"}
```

操作リクエストからの応答は、この操作が成功したことを表しています。また、**read-attribute** 操作を使用して **scan-enabled** 属性を読み取り (現在は **false** と表示)、結果を確認することもできます。

```
/subsystem=deployment-scanner/scanner=default:read-attribute(name=scan-
enabled)
{
    "outcome" => "success",
    "result" => false
}
```

3.5. 属性の定義削除

属性の値を **undefined** に設定できます。この属性にデフォルト値がある場合、デフォルト値が使用されます。

以下の例は、ルートロガーの **level** 属性の定義を削除します。

```
/subsystem=logging/root-logger=ROOT:undefine-attribute(name=level)
```

level 属性のデフォルト値は **ALL** です。**read-resource** 操作の実行時に、このデフォルトの使用を確認できます。

```
/subsystem=logging/root-logger=ROOT:read-resource
{
    "outcome" => "success",
    "result" => {
        "filter" => undefined,
        "filter-spec" => undefined,
        "handlers" => [
            "CONSOLE",
            "FILE"
        ],
        "level" => "ALL"
    }
}
```

デフォルト値を読み取らずにリソースを表示するには、**include-defaults** パラメーターを **false** に設定する必要があります。そうすると、**level** の値が **undefined** になります。

```
/subsystem=logging/root-logger=ROOT:read-resource(include-defaults=false)
{
    "outcome" => "success",
    "result" => {
        "filter" => undefined,
        "filter-spec" => undefined,
        "handlers" => [
            "CONSOLE",
            "FILE"
        ],
        "level" => undefined
    }
}
```

3.6. 操作名の表示

read-operation-names を使用すると、指定のリソースで使用できる操作を一覧表示できます。

```
:read-operation-names
```

以下の例は、デプロイメントで実行できる操作の一覧を表示します。

```
/deployment=DEPLOYMENT_NAME:read-operation-names
{
    "outcome" => "success",
    "result" => [
        "add",
        "deploy",
        "list-add",
        "list-clear",
        "list-get",
        "list-remove",
        "map-clear",
        "map-get",
        "map-put",
        "map-remove",
        "query",
        "read-attribute",
        "read-attribute-group",
        "read-attribute-group-names",
        "read-children-names",
        "read-children-resources",
        "read-children-types",
        "read-operation-description",
        "read-operation-names",
        "read-resource",
        "read-resource-description",
        "redeploy",
        "remove",
        "undefine-attribute",
        "undeploy",
```

```
        "whoami",
        "write-attribute"
    ]
}
```

read-operation-description 操作を使用して 操作の詳細を表示します。

3.7. 操作の詳細表示

read-operation-description 操作を使用すると、リソースの特定操作の詳細を表示できます。これには、パラメーターの説明と必須のパラメーターが含まれます。

```
:read-operation-description(name=OPERATION_NAME)
```

以下の例は、システムプロパティーにおける **add** 操作の詳細およびパラメーター情報を提供します。

```
/system-property=SYSTEM_PROPERTY:read-operation-description(name=add)
{
    "outcome" => "success",
    "result" => {
        "operation-name" => "add",
        "description" => "Adds a system property or updates an existing
one.",
        "request-properties" => {"value" => {
            "type" => STRING,
            "description" => "The value of the system property.",
            "expressions-allowed" => true,
            "required" => false,
            "nillable" => true,
            "min-length" => 0L,
            "max-length" => 2147483647L
        }},
        "reply-properties" => {},
        "read-only" => false,
        "runtime-only" => false
    }
}
```

3.8. 特殊文字を用いた値の追加

管理 CLI リクエストの作成時、特殊文字が含まれる値を追加する必要があることがあります。管理 CLI リクエストの構文に使われる特殊文字などの一部の特殊文字は、特定の方法で入力する必要があります。

多くの場合、値を二重引用符 (**""**) を囲めば適切に処理されます。使用する特殊文字が許可されるプロパティーであるかどうか分からない場合は、値を追加した後に属性またはリソースを読み取り、適切に保存されたことを確認してください。

以下の特殊文字の処理方法は、該当する項を参照してください。

- 空白文字
- 引用符

- コンマ
- かっこ
- 中かっこ
- 角かっこ
- 発音区別符号

空白文字
デフォルトでは、空白は管理 CLI で追加された値から取り除かれます。値に空白が含まれるようにするには、値を二重引用符 (**""**) または中かっこ (**{}**) で囲むか、バックスラッシュ (\) を使ってエスケープ処理します。

```
/system-property=test1:add(value="Hello World")
/system-property=test2:add(value={Hello World})
/system-property=test3:add(value=Hello\ World)
```

これにより、値が **Hello World** に設定されます。

引用符
値で単一引用符 (**'**) を使用するには、値を二重引用符 (**""**) で囲むか、バックスラッシュ (\) を使用してエスケープ処理します。以下の例は、システムプロパティーの値を **server's** に設定します。

```
/system-property=test1:add(value="server's")
/system-property=test2:add(value=server\'s)
```

値で二重引用符 (**"**) を使用するには、バックスラッシュ (\) を使用してエスケープ処理します。引用符の場所によっては、さらに二重引用符 (**""**) で値を囲む必要がある場合もあります。以下の例は、システムプロパティーの値を **"quote"** に設定します。

```
/system-property=test1:add(value="\"quote\"")
```

コンマ
値でコンマ (**,**) を使用するには、値を二重引用符 (**""**) で囲みます。

```
/system-property=test:add(value="Last,First")
```

これにより、値が **Last,First** に設定されます。

かっこ
値にかっこ (**()**) が含まれるようにするには、値を二重引用符 (**""**) または中かっこ (**{}**) で囲むか、バックスラッシュ (\) を使ってエスケープ処理します。

```
/system-property=test1:add(value="one(1)")
/system-property=test2:add(value={one(1)})
/system-property=test3:add(value=one\(1\))
```

これにより、値が **one(1)** に設定されます。

中かっこ
値に中かっこ (**{}**) が含まれるようにするには、値を二重引用符 (**""**) で囲みます。

```
/system-property=test:add(value="{braces}")
```

これにより、値が **{braces}** に設定されます。

角かっこ
値に角かっこ (**[]**) が含まれるようにするには、値を二重引用符 (**""**) で囲みます。

```
/system-property=test:add(value="[brackets]")
```

これにより、値が **[brackets]** に設定されます。

発音区別符号
ñ、**ř**、**ý** などの発音区別符号は、管理 CLI を使用して値を追加するときに使用できます。

```
/system-property=test1:add(value=Año)
```

ただし、値は二重引用符 (**""**) で囲まないでください。二重引用符で囲むと、発音区別符号が疑問符 (**?**) に置き換えられます。値に空白文字が含まれる場合は、中かっこ (**{}**) で値を囲むか、バックスラッシュ (****) を使ってエスケープ処理します。

```
/system-property=test2:add(value={Dos años})
/system-property=test3:add(value=Dos\ años)
```

これにより、値が **Dos años** に設定されます。

3.9. 操作ヘッダーの指定

操作ヘッダーを指定すると、操作実行方法の特定の内容を制御することができます。以下の操作ヘッダーを使用することができます。

allow-resource-service-restart

操作の変更を反映するために、再起動な必要なランタイムサービスを再起動するかどうか。デフォルトは **false** です。

警告

allow-resource-service-restart=true ヘッダーを使用すると、必要なサービスが再起動するまでエンドユーザーリクエストの処理が中断される可能性があります。

blocking-timeout

操作がロールバックされる前に、操作の完了プロセス中の任意の時点で操作がブロックする最大時間 (秒単位)。デフォルトは **300** 秒です。

roles

操作を呼び出すユーザーに通常関連付けられるロールの代わりに、アクセス制御の決定時に使用される RBAC ロールのリスト。これは、呼び出し側のパーミッションを減らす場合のみ使用され、増やす場合には使用されないことに注意してください。

rollback-on-runtime-failure

永続化設定の変更をランタイムサービスに適用できなかった場合に、設定の変更を元に戻すかどうか。デフォルトは **true** です。

rollout

管理対象ドメインのデプロイメントのロールアウト計画。詳細は、JBoss EAP『**設定ガイド**』の「ロールアウト計画の使用」を参照してください。

例: 操作ヘッダーを使用したアプリケーションのデプロイメント

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

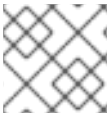
例: 操作ヘッダーを使用したリソースの削除

```
/subsystem=infinispan/cache-container=test:remove() {allow-resource-service-restart=true}
```

セミコロン(**;**) を使用して複数の操作ヘッダーを区切ります。

3.10. IF-ELSE 制御フローの使用

管理 CLI は、条件を基にして実行するコマンドおよび操作のセットを選択可能にする **if-else** 制御フローをサポートします。**if** 条件は、**of** キーワードの後に指定された管理コマンドまたは操作の応答を評価するブール式です。



注記

入れ子の **if-else** ステートメントの使用はサポートされません。

以下の項目をどれでも式に含めることができます。

- 式をグループ化および優先付けするかっこ
- 条件演算子
  - および (and) (**&&**)
  - または (or) (**||**)
- 比較演算子
  - 等しい (**==**)
  - 等しくない (**!=**)
  - より大きい (**>**)
  - より大きいまたは等しい (**>=**)

- より小さい (**<**)
- より小さいまたは等しい (**<=**)
- 正規表現と一致 (**~=**)

重要

正規表現と一致 (**~=**) の演算子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「テクノロジプレビュー機能のサポート範囲」を参照してください。

以下は、正規表現と一致 (**~=**) の演算子を使用して、**features** システムプロパティーの値に **jgroups** が含まれるかどうかをチェックします。

```
if (result ~= ".*jgroups.*") of /:resolve-
expression(expression=${features})
  echo Configuring JGroups
end-if
```

以下の例は、システムプロパティー **test** の読み取りを試みます。**outcome** が **success** でない場合 (プロパティーが存在しないことを意味します)、システムプロパティーが追加され、**true** に設定されます。

```
if (outcome != success) of /system-property=test:read-resource
    /system-property=test:add(value=true)
end-if
```

上記の条件は、**outcome** を使用します。これは、以下のように **of** キーワードの後の CLI コマンドが実行されると返されます。

```
/system-property=test:read-resource
{
    "outcome" => "failed",
    "failure-description" => "JBAS014807: Management resource '[(\"system-
property\" => \"test\")]' not found",
    "rolled-back" => true
}
```

以下の例は、サーバープロセスの起動型 (**STANDALONE** または **DOMAIN**) をチェックし、適切な管理 CLI コマンドを実行して **ExampleDS** データソースを有効にします。

```
if (result == STANDALONE) of /:read-attribute(name=launch-type)
    /subsystem=datasources/data-source=ExampleDS:write-
attribute(name=enabled, value=true)
else
```

```
    /profile=full/subsystem=datasources/data-source=ExampleDS:write-
attribute(name=enabled, value=true)
end-if
```

if-else 制御フローを使用する管理 CLI コマンドをファイルに指定することができ、ファイルには各コマンドを各行に個別に指定します。ファイルを **jboss-cli** スクリプトに渡すと、**--file** パラメーターを使用して非対話的に実行されます。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=CLI_FILE
```

3.11. TRY-CATCH-FINALLY 制御フローの使用

管理 CLI は簡単な **try-catch-finally** 制御フローを提供します。これは、**try**、**catch**、および **finally** ブロックに対応する 3 つの操作およびコマンドのセットで構成されます。**catch** および **finally** ブロックは任意ですが、最低でも 1 つが存在するべきで、1 つの catch ブロックのみを指定できます。

制御フローは **try** バッチの実行で始まります。**try** バッチが正常に完了すると、**catch** バッチはスキップされ、**finally** バッチが実行されます。**java.io.IOException** などが原因で **try** バッチが失敗すると、**try-catch-finally** 制御フローは即座に終了します。利用できる場合は **catch** バッチが実行されます。**finally** バッチは、**try** および **catch** バッチの実行に成功または失敗に関わらず、制御フロードサイドに実行されます。

try-catch-finally 制御フローを定義するコマンドは 4 つあります。

- **try** コマンドは **try** バッチを開始します。**try** バッチは、**catch** または **finally** コマンドの 1 つが実行されるまで継続されます。
- **catch** コマンドは **try** バッチの最後を示します。**try** バッチは引き止められ、**catch** バッチが開始されます。
- **finally** コマンドは **catch** バッチまたは **try** バッチの最後を示し、 **finally** バッチを開始します。
- **end-try** は **catch** または **finally** バッチを終了し、try-catch-finally 制御フローを実行するコマンドです。

以下の例はデータソースを作成または再作成し、有効化します。

```
try
/subsystem=datasources/data-source=myds:add(connection-
url=CONNECTION_URL,jndi-name=java:/myds,driver-name=h2)

catch
/subsystem=datasources/data-source=myds:remove
/subsystem=datasources/data-source=myds:add(connection-
url=CONNECTION_URL,jndi-name=java:/myds,driver-name=h2)

finally
/subsystem=datasources/data-source=myds:enable
end-try
```

3.12. リソースのクエリー

JBoss EAP 管理 CLI は、**query** 操作を提供してリソースをクエリーします。**:read-resource** 操作を使用して、リソースのすべての属性を読み取りできます。選択された属性のみを一覧表示するには、**:query** 操作を使用します。

たとえば、**name** および **enabled** 属性のリストを表示するには、以下のコマンドを実行します。

```
/deployment=jboss-modules.jar:query(select=["name","enabled"])
```

以下の応答は操作の成功を表しています。**name** および **enabled** 属性は **jboss-modules.jar** デプロイメントに対して表示されています。

```
{
    "outcome" => "success",
    "result" => {
        "name" => "jboss-modules.jar",
        "enabled" => true
    }
}
```

ワイルドカードを使用すると、すべてのデプロイメントの **name** および **enabled** 属性の表示など、複数のリソース全体でクエリーを制御できます。

```
/deployment=*:query(select=["name","enabled"])
```

以下の応答は、操作に成功したことを表しています。すべてのデプロイメントの **name** および **enabled** 属性が一覧表示されています。

```
{
     "outcome" => "success",
     "result" => [
         {
             "address" => [("deployment" => "helloworld.war")],
             "outcome" => "success",
             "result" => {
                 "name" => "helloworld.war",
                 "enabled" => true
             }
         },
         {
             "address" => [("deployment" => "kitchensink.war")],
             "outcome" => "success",
             "result" => {
                 "name" => "kitchensink.war",
                 "enabled" => true
             }
         },
         {
             "address" => [("deployment" => "xyz.jar")],
             "outcome" => "success",
             "result" => {
                 "name" => "xyz.jar",
                 "enabled" => false
             }
```

```
            }
        ]
    }
```

:query 操作は、関連するオブジェクトもフィルターします。たとえば、 **enabled** が **true** である、デプロイメントの **name** および **enabled** 属性値を表示します。

```
/deployment=*:query(select=["name","enabled"],where=["enabled","true"])
```

以下の応答は、操作に成功したことを表しています。**enabled** が **true** である、デプロイメントの **name** および **enabled** 属性値が表示されます。

```
{
    "outcome" => "success",
    "result" => [
        {
            "address" => [("deployment" => "helloworld.war")],
            "outcome" => "success",
            "result" => {
                "name" => "helloworld.war",
                "enabled" => true
            }
        },
        {
            "address" => [("deployment" => "kitchensink.war")],
            "outcome" => "success",
            "result" => {
                "name" => "kitchensink.war",
                "enabled" => true
            }
        }
    ]
}
```

第4章 管理対象ドメインでの管理 CLI の使用

管理 CLI を使用すると、スタンドアロンサーバーと管理対象ドメインを両方設定および管理できます。JBoss EAP のドキュメントには、通常はスタンドアロンサーバー設定での管理 CLI コマンドの例が記載されています。管理対象ドメインを実行している場合は、コマンドを調整する必要があります。ここでは、スタンドアロンサーバーの管理 CLI コマンドを管理対象ドメイン設定用に変更する方法について説明します。

サブシステム設定のプロファイル指定

スタンドアロンサーバーのサブシステム設定の管理 CLI コマンドは、**/subsystem=SUBSYSTEM_NAME** で始まります。管理対象ドメインのサブシステム設定では、コマンドの最初に **/profile=PROFILE_NAME/subsystem=SUBSYSTEM_NAME** を追加して、どのプロファイルのサブシステムを設定するか指定する必要があります。

例: logging サブシステム設定の読み取り (スタンドアロンサーバー)

```
/subsystem=logging:read-resource
```

以下の例は、スタンドアロンサーバーの **logging** サブシステムの設定を読み取る方法を表しています。

例: logging サブシステム設定の読み取り (管理対象ドメイン)

```
/profile=default/subsystem=logging:read-resource
```

この例は、管理対象ドメインの **default** プロファイルに対して **logging** サブシステムの設定を読み取る方法を表しています。

コア管理およびランタイムコマンドのホスト指定

管理対象ドメインのコア管理およびランタイムコマンドによっては、コマンドの最初に **/host=HOST_NAME** を追加して、コマンドを適用するホストを指定する必要があるものがあります。

例: 監査ロギングの有効化 (スタンドアロンサーバー)

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

この例は、スタンドアロンサーバーで監査ロギングを有効にする方法を表しています。

例: 監査ロギングの有効化 (管理対象ドメイン)

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

この例は、管理対象ドメインの **master** ホストで監査ロギングを有効にする方法を表しています。

注記

コマンドによっては、**reload --host=HOST_NAME** のように、ホストを引数とする必要があるものがあります。このようなコマンドでホストを指定しないと、**--host** 引数が必要であること伝えるエラーメッセージが出力されます。

コア管理およびランタイムコマンドのサーバー指定

管理対象ドメインのコア管理およびランタイムコマンドによっては、コマンドの最初に **/host=HOST_NAME/server=SERVER_NAME** を追加して、コマンドを適用するホストおよびサーバーを指定する必要があるものがあります。

例: デプロイメントのランタイムメトリックスの表示 (スタンドアロンサーバー)

```
/deployment=test-application.war/subsystem=undertow:read-
attribute(name=active-sessions)
```

この例は、スタンドアロンサーバーデプロイメントのランタイムメトリックスを表示する方法を表しています。

例: デプロイメントのランタイムメトリックスの表示 (管理対象ドメイン)

```
/host=master/server=server-one/deployment=test-
application.war/subsystem=undertow:read-attribute(name=active-sessions)
```

この例は、 **master** ホスト上の **server-one** サーバーにデプロイされる、管理対象ドメインデプロイメントのランタイムメトリックスの表示方法を表しています。

第5章 管理 CLI の設定

管理 CLI の特定の設定は、設定ファイル **jboss-cli.xml** でカスタマイズできます。このファイルは **EAP_HOME/bin** ディレクトリーか、**jboss.cli.config** システムプロパティーで指定されたカスタムディレクトリーに存在する必要があります。

jboss-cli.xml ファイルでは以下の要素を設定できます。

default-protocol

プロコトルの指定がない状態でコントローラーアドレスが提供された場合に使用するデフォルトのプロトコル。デフォルトは **remote+http** です。ポート **9999** が使用され、プロトコルの指定がない場合、**use-legacy-override** 属性が **false** に設定されていない限り、プロトコルは自動的にデフォルトの **remoting** に指定されます。

default-controller

connect コマンドがパラメーターの指定なしで実行された場合に接続するコントローラーの設定。管理 CLI が引数 **--controller=** または **controller=** で開始された場合、引数に指定された値は設定の **default-controller** 定義よりも優先されます。

- **protocol** - コントローラーのプロトコル名。指定のない場合、**default-protocol** の値が使用されます。
- **host** - コントローラーのホスト名。デフォルトは **localhost** です。
- **port** - コントローラーに接続するポート番号。デフォルト値は **9990** です。

controllers

jboss-cli.xml ファイルに接続コントローラーのエイリアスを定義できます。例を以下に示します。

```
<!-- The default controller to connect to when 'connect' command is
executed w/o arguments -->
<default-controller>
    <host>localhost</host>
    <port>9990</port>
</default-controller>
<!-- CLI connection controller aliases -->
<controllers>
    <controller name="ServerOne">
        <protocol>remoting</protocol>
        <host>192.168.3.45</host>
        <port>9999</port>
    </controller>
    <controller name="ServerTwo">
        <protocol>http-remoting</protocol>
        <host>192.168.3.46</host>
    </controller>
</controllers>
```

コントローラー要素の **name** 属性は、たとえば **--controller=ServerTwo** のように、**--controller=** 引数への値として使用する必要があります。

validate-operation-requests

実行のリクエストをコントローラーに送信する前に、操作リクエストのパラメーターリストを検証するかどうか。デフォルトは **true** です。

history

CLI コマンド履歴ログの設定。

- **enabled** - **history** を有効にするかどうか。デフォルトは **true** です。
- **file-name** - 履歴を保存するファイル名。デフォルトは **.jboss-cli-history** です。
- **file-dir** - 履歴が保存されるディレクトリー。デフォルトはユーザーのホームディレクトリーです。
- **max-size** - 履歴ファイルに格納されるコマンドの最大数。デフォルトは **500** です。

resolve-parameter-values

操作のリクエストをコントローラーに送信する前に、コマンド引数 (または操作パラメーター) の値として指定されたシステムプロパティーを解決するかどうか。デフォルトは **false** です。

connection-timeout

コントローラーで接続を確立できる期間 (ミリ秒単位)。デフォルトは **5000** です。

ssl

SSL に使用されるキーストアおよびトラストストアの設定。

警告

Red Hat では、影響するすべてのパッケージで TLSv1.1 または TLSv1.2 を利用するために SSLv2、SSLv3、および TLSv1.0 を明示的に無効化することを推奨しています。

- **vault** - vault の設定。**code** と **module** の指定がない場合、デフォルトの実装が使用されます。 **code** は指定され、**module** は指定されていない場合、Picketbox モジュールで指定されたクラスを探します。**module** と **code** が指定されている場合、「module」によって指定されたモジュールのコードによって指定されたクラスを探します。
- **key-store** - キーストア。
- **key-store-password** - キーストアのパスワード。
- **alias** - エイリアス。
- **key-password** - キーのパスワード。
- **trust-store** - トラストストア。
- **trust-store-password** - トラストストアのパスワード。
- **modify-trust-store** - **true** に設定された場合、ユーザーに認識できない証明書が受信されたことを通知し、それらの証明書はトラストストアに格納することができます。デフォルトは **true** です。

silent

ターミナルに情報およびエラーメッセージを書き込むかどうか。デフォルトは **false** です。

access-control

現在のユーザーに付与されたパーミッションを基にして管理関連のコマンドおよび属性をフィルターすべきかどうか。たとえば、**true** の場合、ユーザーのアクセスが許可されていないコマンドおよび属性はタブ補完では表示されません。デフォルトは **true** です。

echo-command

非対話モードで実行されたコマンドの出力にプロンプトおよびコマンドを含むかどうか。デフォルトは **false** です。

command-timeout

コマンドが完了するまで待機する最大時間。**0** の値はタイムアウトにならないことを意味します。デフォルトではタイムアウトは発生しません。

5.1. プロパティーの置換

JBoss EAP は、管理 CLI での事前設定の要素式およびプロパティー式の使用をサポートします。これらの式は、コマンドの実行中に定義された値に解決されます。

式を以下のプロパティーに置き換えることができます。

- 操作リクエストの操作アドレス部分 (ノード型、名前など)
- 操作名
- 操作パラメーター名
- ヘッダー名および値
- コマンド名
- コマンド引数名

デフォルトでは、管理 CLI は引数またはパラメーターの値以外の各行に対してプロパティーの置換を実行します。引数およびパラメーターの値は起動時にサーバーで解決されます。引数またはパラメーターの値のプロパティー置換を管理 CLI で行い、解決された値をサーバーに送信する必要がある場合は、以下の手順に従います。

1. 管理 CLI 設定ファイル **EAP_HOME/bin/jboss-cli.xml** を編集します。
2. **resolve-parameter-values** パラメーターを **true** に設定します (デフォルトは **false** です)。

```
<resolve-parameter-values>true</resolve-parameter-values>
```

この要素は、操作リクエストのパラメーターの値とコマンド引数の値のみに影響します。他のコマンドラインには影響しません。そのため、パラメーターや引数の値でないコマンドライン上のシステムプロパティーは、**resolve-parameter-values** 要素の値に関係なく、行の解析中に解決されます。

管理 CLI コマンドで使用されるシステムプロパティーの値を解決するには、それらのプロパティーが定義されている必要があります。管理 CLI インスタンスの起動時にプロパティーファイル (**--properties=/path/to/file.properties**) またはプロパティーと値のペア (**-Dkey=value**) に渡す必要があります。プロパティーファイルは標準の **KEY=VALUE** 構文を使用します。

プロパティーキーは、以下の例のように **${MY_VAR}** 構文を使用して管理 CLI コマンドで使用されます。

```
/host=${hostname}/server-config=${servername}:add(group=main-server-group)
```

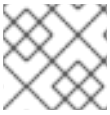
その他の **jboss-cli.xml** 設定オプションは、「管理 CLI の設定」を参照してください。

5.2. エイリアスの作成

alias コマンドを使用すると、CLI セッション中に CLI コマンドおよび操作のエイリアスを定義できます。

以下の例は、**alias** コマンドを使用して **undertow** サブシステムのリソースを読み取る、**read_undertow** という名前の新しい CLI コマンドエイリアスを作成します。

```
alias read_undertow='/subsystem=undertow:read-resource'
```



注記

エイリアス名には、英数字とアンダースコアのみを使用できます。

read_undertow エイリアスの作成をテストするには、管理 CLI にエイリアス名を入力します。

```
read_undertow
```

結果は以下のようになります。

```
{
    "outcome" => "success",
    "result" => {
        "default-security-domain" => "other",
        "default-server" => "default-server",
        "default-servlet-container" => "default",
        "default-virtual-host" => "default-host",
        "instance-id" => expression "${jboss.node.name}",
        "statistics-enabled" => false,
        "buffer-cache" => {"default" => undefined},
        "configuration" => {
            "filter" => undefined,
            "handler" => undefined
        },
        "server" => {"default-server" => undefined},
        "servlet-container" => {"default" => undefined}
    }
}
```

使用できるエイリアスのリストを表示するには、**alias** コマンドを使用します。

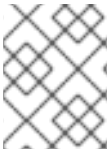
```
alias
```

結果は以下のようになります。

```
alias read_undertow='/subsystem=undertow:read-resource'
```

エイリアスを削除するには、**unalias** コマンドを使用します。

```
unalias read_undertow
```



注記

エイリアスはユーザーのホームディレクトリー内にある **.aesh_aliases** ファイルに格納されます。

5.3. .JBOSSCLIRC 設定ファイル

JBoss EAP には、新規セッションの開始時に環境を初期化できるようにする、ランタイム設定の **.jbossclirc** ファイルが含まれます。このファイルは、**EAP_HOME/bin/** ディレクトリーにあります。ファイルにある例は、ユーザー固有の環境を設定するためのテンプレートとして使用することができます。**.jbossclirc** ファイルはグローバル CLI 変数の格納に適しています。

.jbossclirc ファイルの内容は、CLI がサポートするコマンドおよび操作のリストです。このファイルは、ユーザーに制御を渡す前、新しい管理 CLI セッションの開始時に実行されます。**--properties** 引数で指定されたシステムプロパティーがある場合、**.jbossclirc** ファイルはプロパティーの設定後に実行されます。

.jbossclirc ファイルの例

```
set console=/subsystem=logging/console-handler=CONSOLE
```

注記

--connect または **-c** 引数を使用する場合、**.jbossclirc** はクライアントが実際にサーバーに接続する前に実行されます。

.jbossclirc ファイルを見つけるため、以下の場所が以下の順にチェックされます。

1. システムプロパティー **jboss.cli.rc** が定義済みである場合、その値はファイルへのパスと見なされます。
2. **user.dir** システムプロパティーによって定義されたユーザーの作業ディレクトリー。
3. **EAP_HOME/bin** ディレクトリー。

5.4. 変数の使用

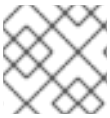
set コマンドの使用

以下のように、**set** コマンドを使用すると、サーバーモデルから変数へのパスを定義できます。

```
set s1=/host=master/server=server-one
```

これは、変数を使用してホストおよびプロファイルへの参照を含めると、異なるサーバー上でスクリプトを簡単に複製できるため、管理対象ドメインで活用できます。例を以下に示します。

```
$s1/subsystem=datasources/data-source=ExampleDS:test-connection-in-pool
```



注記

変数は **$** を使用して参照されます。

unset コマンドの使用

unset コマンドを使用すると変数を削除できます。

```
unset prod_db
```

jbossclirc ファイルの使用

CLI セッション全体で変数を使用するには、これらの変数を **.jbossclirc** ファイルに追加します。このファイルは **EAP_HOME/bin/** ディレクトリーにあります。

例を以下に示します。

```
set s1=/host=master/server=server-one
set s2=/host=master/server=server-two
```

管理 CLI を再起動して、**set** コマンドを実行し、使用できる変数を確認します。

```
set
```

出力は次のようになります。

```
s1=/host=master/server=server-one
s2=/host=master/server=server-two
```

変数はコマンドラインのどこにでも使用され、コマンドラインの解析中に解決されます。この例では、**prod_db** 変数はデータソースに解決されます。

```
$prod_db/statistics=jdbc:read-resource
```

echo コマンドの使用

echo コマンドを使用して変数の値をチェックします。

```
echo $prod_db
```

出力は次のようになります。

```
/subsystem=datasources/data-source=ExampleDS
```

例

以下の例は、変数が表示される場所と、コマンド全体を変数で構成できることを示しています。

```
$prod_db:$op($param=$param_value)
$cmd --$param=$param_value
```

注記

変数は CLI のスクリプトを作成するときに役立ちます。

第6章 管理 CLI のコマンド履歴

管理 CLI にはコマンド履歴の機能があり、これはアプリケーションサーバーインストールではデフォルトで有効になっています。履歴は、記録としてアクティブな CLI セッションの揮発性メモリーに保持され、ユーザーのホームディレクトリーに自動的に保存されるログファイルに **.jboss-cli-history** として追加されます。この履歴ファイルは、デフォルトで最大 **500** 個の CLI コマンドを記録するよう設定されています。履歴ファイルの場所と履歴エントリーの最大数は、**EAP_HOME/bin/jboss-cli.xml** ファイルでカスタマイズできます。

history コマンド自体は現在のセッションの履歴を返します。また、追加の引数を使用するとセッションメモリーの履歴を無効化、有効化、または消去します。また、管理 CLI にはキーボードの矢印キーを使用してコマンドおよび操作の履歴を移動できる機能も含まれています。

管理 CLI コマンド履歴の表示

管理 CLI の起動または履歴消去のコマンドの実行以降にメモリーに保存された CLI コマンドの履歴を表示します。

```
history
```

管理 CLI コマンド履歴の消去

セッションメモリーおよびユーザーのホームディレクトリーにある **.jboss-cli-history** ファイルから CLI コマンドの履歴を消去します。

```
history --clear
```

管理 CLI コマンド履歴の有効化

セッションメモリーと、ユーザーのホームディレクトリにある **.jboss-cli-history** ファイルに CLI コマンドを記録します。

```
history --enable
```

管理 CLI コマンド履歴の無効化

セッションメモリーや、ユーザーのホームディレクトリにある **.jboss-cli-history** ファイルに CLI コマンドを記録しないようにします。

```
history --disable
```

第7章 管理 CLI のロギング

出力やその他の管理 CLI 情報をログファイルに保存できます。デフォルトでは、管理 CLI ロギングは無効になっています。**EAP_HOME/bin/jboss-cli-logging.properties** ファイルを使用すると、ロギングを有効にし、その他のロギングを設定できます。

管理 CLI ロギングの設定

1. **EAP_HOME/bin/jboss-cli-logging.properties** ファイルを編集します。
2. 以下の行をアンコメントまたは追加して、ロギングを有効にします。

```
# uncomment to enable logging to the file
logger.handlers=FILE
```

3. ログレベルを **OFF** から **INFO** や **ALL** などの適切なレベルに変更します。

```
logger.org.jboss.as.cli.level=INFO
```

管理 CLI を再起動すると、出力が **EAP_HOME/bin/jboss-cli.log** ファイルに記録されます。

ロギングプロパティーファイルの他の設定に関する詳細は、JBoss EAP『**開発ガイド**』の「logging.properties の設定」を参照してください。

第8章 バッチ処理

バッチ処理は、複数の操作リクエストを 1 つのシーケンスにグループ化し、1 つのユニットとして一緒に実行できるようにします。シーケンスの操作リクエストのいずれかが失敗すると、操作のグループ全体がロールバックされます。

注記

バッチモードは条件付きステートメントをサポートしません。

1. **batch** 管理 CLI コマンドでバッチモードに変更します。

   ```
   batch
   ```

   バッチモードになると、プロンプトにハッシュ (#) が表示されます。

2. 操作リクエストをバッチに追加します。
   バッチモードでは、通常通りに操作リクエストを入力します。操作リクエストは、入力順にバッチに追加されます。

   バッチコマンドは編集および順序変更が可能です。また、バッチを保存して後で処理することもできます。バッチ処理で使用できるコマンドの完全リストは「管理 CLI のバッチモードコマンド」を参照してください。

3. バッチを実行します。
   操作リクエストのシーケンスをすべて入力したら、**run-batch** コマンドでバッチ処理を実行します。

   ```
   run-batch
   ```

   操作リクエストのシーケンス全体がバッチとして完了し、結果として **The batch executed successfully.** がターミナルに出力されます。

外部ファイルの batch コマンド

頻繁に実行する batch コマンドは外部テキストファイルに保存し、**batch** コマンドへの引数として完全パスをファイルに渡してロードするか、引数として **run-batch** コマンドへ渡して直接実行することができます。

テキストエディターを使用するか、各コマンドを各行に追加すると、batch コマンドファイルを作成できます。

以下のコマンドは、**myscript.txt** ファイルをバッチモードでロードします。その後、このファイルからのコマンドを編集または削除できます。新しいコマンドを挿入することもできます。このバッチセッションでの変更は、**myscript.txt** ファイルに永続化されません。

```
batch --file=myscript.txt
```

以下のコマンドは、**myscript.txt** ファイルに保存された batch コマンドを即座に実行します。

```
run-batch --file=myscript.txt
```

入力された操作リクエストのシーケンスがバッチとして完了します。

第9章 オフライン設定でのサーバーの埋め込み

JBoss EAP スタンドアロンサーバーまたはホストコントローラープロセスを管理 CLI プロセス内に埋め込みすることができます。これにより、ネットワーク上で認識できない場合でもサーバーを設定することが可能になります。これは、一般的にサーバーがオンラインになる前に、セキュリティー関連の設定の管理やポートの競合の回避など、サーバーの初期設定で使用されます。

この、管理 CLI を経由した JBoss EAP インストールの直接的なローカル管理には、ソケットベースの接続が必要ありません。リモート JBoss EAP サーバーとの対話で一貫性を保つよう、埋め込みサーバーを持つ管理 CLI を使用できます。リモートサーバーの管理に使用できる標準の管理 CLI コマンドはすべて利用できます。

埋め込みスタンドアロンサーバーの起動

追加のプロセスを起動したり、ネットワークソケットを開かなくても、管理 CLI を使用してスタンドアロンサーバーをローカルで起動し、スタンドアロン設定を編集することができます。

以下の手順は、管理 CLI の起動、埋め込みスタンドアロンサーバーの開始、設定の変更、および埋め込みサーバーの停止を行います。

1. 管理 CLI を起動します。

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

2. 埋め込みスタンドアロンサーバーを起動します。
   --std-out=echo パラメーターを渡すと、標準出力がターミナルに表示されます。

```
embed-server --std-out=echo
```

3. 希望の操作を実行します。

```
/socket-binding-group=standard-sockets/socket-binding=management-
http:write-attribute(name=port,value=9991)
```

4. 埋め込みサーバーを停止します。

```
stop-embedded-server
```

   これにより、埋め込みサーバーが停止し、管理 CLI セッションが返されます。管理 CLI セッションも終了する場合は、**quit** を使用できます。

サーバー設定の指定

デフォルトでは、埋め込みサーバーは **standalone.xml** 設定ファイルを使用します。**--server-config** パラメーターを使用すると別の設定ファイルを指定できます。

```
embed-server --server-config=standalone-full-ha.xml
```

admin-only モードの開始

デフォルトでは、埋め込みサーバーは **admin-only** モードで起動されます。これは、サーバー管理に関するサービスを起動しますが、他のサービスは起動せず、エンドユーザーの要求も受け入れません。これは、サーバーの初期設定時に便利です。

--admin-only パラメーターを false に設定すると、埋め込みサーバーを通常の実行モードで起動できます。

```
embed-server --admin-only=false
```

また、**reload** コマンドを使用して実行モードを変更することもできます。

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

標準出力の制御
埋め込みサーバーからの標準出力の処理方法を制御することができます。デフォルトでは、標準出力は破棄されますが、サーバーログに出力が記録されます。**--std-out=echo** を渡すと、サーバーの出力を管理 CLI の出力と表示できます。

```
embed-server --std-out=echo
```

ブートタイムアウト
デフォルトでは **embed-server** コマンドは、埋め込みサーバーが完全に起動するまで無期限にブロックされます。**--timeout** パラメーターを使用すると待機時間を秒単位で指定できます。**1** 未満の値の場合、CLI が埋め込みサーバーを管理できる状態になった時点で即座に返されます。

```
embed-server --timeout=30
```

空の設定での開始
埋め込みサーバーを開始するとき、空の設定で開始するように指定できます。これは、管理 CLI コマンドを使用してサーバー全体の設定を構築する場合に便利です。

```
embed-server --server-config=my-config.xml --empty-config
```

ファイルが存在すると、このコマンドは失敗します。これにより、設定ファイルを誤って削除しないようにします。**--remove-existing** パラメーターを渡すと、既存の設定をすべて削除するよう指定できます。

```
embed-server --server-config=my-config.xml --empty-config --remove-
existing
```

埋め込みホストコントローラーの開始

追加のプロセスを起動したり、ネットワークソケットを開かなくても、管理 CLI を使用してホストコントローラーをローカルで起動し、ドメインおよびホストコントローラーの設定を変更することができます。

埋め込みホストコントローラーは、そのサーバーを起動しません。さらに、埋め込みホストコントローラーを開始するときに **--admin-only** パラメーターを使用することもできません。常に **admin-only** モードになるように起動されます。

以下の手順は、管理 CLI の起動、埋め込みホストコントローラーの開始、設定の変更、および埋め込みホストコントローラーの停止を行います。

1. 管理 CLI を起動します。

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

2. 埋め込みホストコントローラーを起動します。
   --std-out=echo パラメーターを渡すと、標準出力がターミナルに表示されます。

```
embed-host-controller --std-out=echo
```

3. 希望の操作を実行します。

```
/host=HOST_NAME:write-attribute(name=name,value=NEW_HOST_NAME)
```

4. 埋め込みホストコントローラーを停止します。

```
stop-embedded-host-controller
```

ホストコントローラー設定の指定

デフォルトでは、埋め込みホストコントローラーはドメイン設定に **domain.xml** を使用し、ホスト設定には **host.xml** を使用します。**--domain-config** および **--host-config** パラメーターを使用すると、別の設定ファイルを指定できます。

```
embed-host-controller --domain-config=other-domain.xml --host-config=host-slave.xml
```

注記

使用する別の設定ファイルによっては、管理 CLI の起動時に特定のプロパティーを設定する必要があることがあります。例を以下に示します。

```
$ EAP_HOME/bin/jboss-cli.sh -Djboss.domain.master.address=127.0.0.1
```

標準出力の制御

埋め込みサーバーからの標準出力の処理方法を制御することができます。デフォルトでは、標準出力は破棄されますが、ホストコントローラーのログに出力が記録されます。**--std-out=echo** を渡すと、ホストコントローラーの出力を管理 CLI の出力と表示できます。

```
embed-host-controller --std-out=echo
```

ブートタイムアウト

デフォルトでは **embed-host-controller** コマンドは、埋め込みホストコントローラーが完全に起動するまで無期限にブロックされます。**--timeout** パラメーターを使用すると待機時間を秒単位で指定できます。**1** 未満の値の場合、CLI が埋め込みホストコントローラーを管理できる状態になった時点で即座に返されます。

```
embed-host-controller --timeout=30
```

管理 CLI での非モジュラークラスローティング

EAP_HOME/bin/jboss-cli.sh スクリプトを使用して管理 CLI を起動すると、モジュラークラスローディング環境が使用されます。**EAP_HOME/bin/client/jboss-cli-client.jar** を使用して管理 CLI を非モジュラークラスローティング環境で実行するには、JBoss EAP インストールのルートディレクトリーを指定する必要があります。

1. 管理 CLI を起動します。

```
$ java -jar EAP_HOME/bin/client/jboss-cli-client.jar
```

2. ルートインストールディレクトリーを指定して、埋め込みサーバーを起動します。

```
embed-server --jboss-home=/path/to/EAP_HOME
```

注記

ホストコントローラーを埋め込みするには、**embed-host-controller** コマンドを使用します。

埋め込みロジックによって、サーバーに適したモジュラークラスローティング環境が設定されます。モジュラークラスローダーのモジュールパスには単一の要素 **EAP_HOME/modules** があります。

管理 CLI を起動する方法に関わらず、埋め込みサーバーはモジュラークラスローディング環境で実行されます。

第10章 ハウツー集

以下の CLI コマンドおよび操作は、特定のタスクを達成する方法の基本的な例になります。手順の詳細は、『**設定ガイド**』やその他の JBoss EAP ガイド の該当する項目を参照してください。

指定のない限り、これらの例はスタンドアロンサーバーとして実行される場合に適用されます。コマンドの使用方法を表示するには、コマンド上で **--help** 引数を使用します。ソースの特定操作の情報を取得するには、**read-operation-description** を使用します。

10.1. データソースの追加

```
data-source add --name=DATASOURCE_NAME --jndi-name=JNDI_NAME --driver-
name=DRIVER_NAME  --connection-url=CONNECTION_URL
```

10.2. 拡張機能の追加

例: 新しい拡張機能を設定に追加する

```
/extension=EXTENSION_NAME:add
```

10.3. JMS キューの追加

```
jms-queue add --queue-address=QUEUE_NAME --entries=JNDI_NAME
```

10.4. JMS トピックの追加

```
jms-topic add --topic-address=TOPIC_NAME --entries=JNDI_NAME
```

10.5. モジュールの追加

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

重要

module 管理 CLI コマンドを使用したモジュールの追加および削除は、テクノロジープレビューとしてのみ提供されます。このコマンドは、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する必要があります。詳細は、JBoss EAP『**設定ガイド**』の「カスタムモジュールの手動作成」および「手作業によるカスタムモジュールの削除」を参照してください。

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「テクノロジプレビュー機能のサポート範囲」を参照してください。

10.6. サーバーの追加

例: 管理対象ドメインで新しいサーバーをホストに追加する

```
/host=HOST_NAME/server-config=SERVER_NAME:add(group=SERVER_GROUP_NAME)
```

10.7. サーバーグループの追加

例: 管理対象ドメインで新しいサーバーグループを追加する

```
/server-group=SERVER_GROUP_NAME:add(profile=PROFILE_NAME, socket-binding-
group=SOCKET_BINDING_GROUP_NAME)
```

10.8. システムプロパティーの追加

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

10.9. プロファイルのクローン

例: 管理対象ドメインでプロファイルをクローンする

```
/profile=PROFILE_TO_CLONE:clone(to-profile=NEW_PROFILE_NAME)
```

10.10. 階層プロファイルの作成

例: 他のプロファイルを継承する新しいプロファイルを作成する

```
/profile=NEW_PROFILE_NAME:add(includes=[PROFILE_1,PROFILE_2])
```

10.11. アプリケーションの管理対象ドメインへのデプロイ

例: 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

```
deploy /path/to/DEPLOYMENT.war --all-server-groups
```

例: 1 つ以上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

```
deploy /path/to/DEPLOYMENT.war --server-
groups=SERVER_GROUP_1,SERVER_GROUP_2
```

10.12. 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

```
deploy /path/to/DEPLOYMENT.war
```

10.13. すべてのアプリケーションを無効化

```
undeploy * --keep-content
```

デプロイメント名にワイルドカード (*****) を使用します。

10.14. アクティブなユーザーの表示

例: 現在のユーザーを表示するコマンド

```
:whoami
```

例: 現在のユーザーの出力

```
{
    "outcome" => "success",
    "result" => {"identity" => {
        "username" => "$local",
        "realm" => "ManagementRealm"
    }}
}
```

10.15. 添付内容の表示

attachment display コマンドを使用すると、管理操作から返された添付の内容を表示することができます。これは、**attached-streams** 応答ヘッダーを返す管理操作に適用されます。

たとえば、以下の操作はストリームとして添付された **server.log** ファイルを返します。

```
/subsystem=logging/log-file=server.log:read-attribute(name=stream)
{
    "outcome" => "success",
    "result" => "f61a27c4-c5a7-43ac-af1f-29e90c9acb3e",
    "response-headers" => {"attached-streams" => [{
        "uuid" => "f61a27c4-c5a7-43ac-af1f-29e90c9acb3e",
```

```
        "mime-type" => "text/plain"
    }]}
}
```

attachment display コマンドを使用すると、この操作からコンソールに返されたストリームの内容を表示できます。

```
attachment display --operation=/subsystem=logging/log-
file=server.log:read-attribute(name=stream)
```

これは、**server.log** ファイルの内容をコンソールに出力します。

```
ATTACHMENT f61a27c4-c5a7-43ac-af1f-29e90c9acb3e:
2017-08-14 15:20:01,472 INFO  [org.jboss.modules] (main) JBoss Modules
version 1.6.0.CR2-redhat-1
2017-08-14 15:20:01,721 INFO  [org.jboss.msc] (main) JBoss MSC version
1.2.7.SP1-redhat-1
2017-08-14 15:20:01,853 INFO  [org.jboss.as] (MSC service thread 1-8)
WFLYSRV0049: JBoss EAP 7.1.0.GA (WildFly Core 3.0.0.Beta30-redhat-1)
starting
...
```

10.16. スキーマ情報の表示

:product-info コマンドのスキーマ情報を表示します。

```
:read-operation-description(name=product-info)
```

スキーマバージョンを表示するには、管理 CLI のルートで **ls** コマンドを実行し、**management-*-version** の値を見つけます。

```
...
management-major-version=4
management-micro-version=0
management-minor-version=1
...
```

10.17. システムおよびサーバー情報の表示

例: システムおよびサーバー情報を表示するコマンド

```
:product-info
```

例: システムおよびサーバー情報の出力

```
{
    "outcome" => "success",
    "result" => [{"summary" => {
        "host-name" => "HOST_NAME",
        "instance-identifier" => "INSTANCE_ID",
        "product-name" => "JBoss EAP",
```

```
        "product-version" => "7.1.0.GA",
        "product-community-identifier" => "Product",
        "product-home" => "EAP_HOME",
        "standalone-or-domain-identifier" => "OPERATING_MODE",
        "host-operating-system" => "OS_NAME",
        "host-cpu" => {
            "host-cpu-arch" => "CPU_ARCH",
            "host-core-count" => CORE_COUNT
        },
        "jvm" => {
            "name" => "JAVA_VM_NAME",
            "java-version" => "JAVA_VERSION",
            "jvm-version" => "JAVA_VM_VERSION",
            "jvm-vendor" => "JAVA_VM_VENDOR",
            "java-home" => "JAVA_HOME"
        }
    }}]
}
```

同様に、管理対象ドメインでは、特定の JBoss EAP ホストまたはサーバーの情報を表示できます。

```
/host=HOST_NAME:product-info
```

```
/host=HOST_NAME/server=SERVER_NAME:product-info
```

10.18. コマンドタイムアウト値の取得

例: CLI コマンドのタイムアウト値を表示する

```
command-timeout get
```

秒単位の値が返されます。**0** はタイムアウトがないことを意味します。

10.19. 無効なデプロイメントをすべて再デプロイ

```
deploy --name=*
```

デプロイメント名にワイルドカード (*****) を使用します。

10.20. ホストコントローラーのリロード

```
reload --host=HOST_NAME
```

10.21. ADMIN-ONLY モードでのホストコントローラーのリロード

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

10.22. サーバーグループのすべてのサーバーをリロード

例: 管理対象ドメインの特定サーバーグループのサーバーをすべてリロードする

```
/server-group=SERVER_GROUP_NAME:reload-servers
```

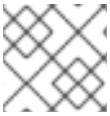
注記

停止状態でサーバーをリロードするには、**start-mode=suspend** 引数を渡します。

10.23. サーバーのリロード

例: 管理対象ドメインでサーバーをリロードする

```
/host=HOST_NAME/server-config=SERVER_NAME:reload
```



注記

停止状態でサーバーをリロードするには、**start-mode=suspend** 引数を渡します。

10.24. スタンドアロンサーバーのリロード

```
reload
```

注記

admin-only モードでサーバーをリロードするには、**--start-mode=admin-only** 引数を渡します。停止状態でサーバーをリロードするには、**start-mode=suspend** 引数を渡します。

10.25. 拡張機能の削除

例: 既存の拡張機能を削除する

```
/extension=EXTENSION_NAME:remove
```

10.26. モジュールの削除

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

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する必要があります。詳細は、JBoss EAP『**設定ガイド**』の「カスタムモジュールの手動作成」および「手作業によるカスタムモジュールの削除」を参照してください。

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「テクノロジプレビュー機能のサポート範囲」を参照してください。

10.27. コマンドのタイムアウト値のリセット

例: コマンドのタイムアウトをデフォルト値にリセットする

```
command-timeout reset default
```

例: コマンドのタイムアウトを CLI 設定によって提供される値にリセットする

```
command-timeout reset config
```

注記

CLI 設定によって提供される値は、**EAP_HOME/bin/jboss-cli.xml** ファイルに設定でき、また管理 CLI の起動時に **--command-timeout** 引数で渡すこともできます。

10.28. サーバーグループのすべてのサーバーを再起動

例: 管理対象ドメインで特定のサーバーグループのサーバーをすべて再起動する

```
/server-group=SERVER_GROUP_NAME:restart-servers
```

注記

停止状態でサーバーを再起動するには、**start-mode=suspend** 引数を渡します。

10.29. サーバーの再起動

例: 管理対象ドメインでサーバーを再起動する

```
/host=HOST_NAME/server-config=SERVER_NAME:restart
```

注記

停止状態でサーバーを再起動するには、**start-mode=suspend** 引数を渡します。

10.30. 添付内容の保存

attachment save コマンドを使用すると、管理操作から返された添付の内容を保存することができます。これは、**attached-streams** 応答ヘッダーを返す管理操作に適用されます。

たとえば、以下の操作はストリームとして添付された **server.log** ファイルを返します。

```
/subsystem=logging/log-file=server.log:read-attribute(name=stream)
{
    "outcome" => "success",
    "result" => "f61a27c4-c5a7-43ac-af1f-29e90c9acb3e",
    "response-headers" => {"attached-streams" => [{
        "uuid" => "f61a27c4-c5a7-43ac-af1f-29e90c9acb3e",
        "mime-type" => "text/plain"
    }]}
}
```

attachment save コマンドを使用すると、この操作からコンソールに返されたストリームの内容を保存できます。

```
attachment save --operation=/subsystem=logging/log-file=server.log:read-
attribute(name=stream) --file=log-output.txt
```

これは、**server.log** ファイルの内容を **EAP_HOME/bin/log-output.txt** に保存します。

10.31. コマンドのタイムアウト値の設定

例: CLI コマンドが完了するまでの最大待機時間を設定する

```
command-timeout set TIMEOUT_VALUE
```

値は秒単位で設定されます。**0** はタイムアウトがないことを意味します。

10.32. ホストコントローラーのシャットダウン

例: 管理対象ドメインでホストコントローラーをシャットダウンする

```
shutdown --host=HOST_NAME
```

10.33. サーバーのシャットダウン

例: スタンドアロンサーバーのシャットダウン

```
shutdown
```

10.34. サーバーグループのすべてのサーバーの起動

例: 管理対象ドメインで特定のサーバーグループのサーバーをすべて起動する

```
/server-group=SERVER_GROUP_NAME:start-servers
```

注記

停止状態でサーバーを起動するには、**start-mode=suspend** 引数を渡します。

10.35. サーバーの起動

例: 管理対象ドメインでサーバーを起動する

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

注記

停止状態でサーバーを起動するには、**start-mode=suspend** 引数を渡します。

10.36. サーバーグループのすべてのサーバーの停止

例: 管理対象ドメインで特定のサーバーグループのサーバーをすべて停止する

```
/server-group=SERVER_GROUP_NAME:stop-servers
```

10.37. サーバーの停止

例: 管理対象ドメインでサーバーを停止する

```
/host=HOST_NAME/server-config=SERVER_NAME:stop
```

10.38. 設定スナップショットの作成

例: 現在の設定のスナップショットを作成する

```
:take-snapshot
```

10.39. すべてのアプリケーションのアンデプロイ

```
undeploy *
```

デプロイメント名にワイルドカード (*****) を使用します。

10.40. 管理対象ドメインからのアプリケーションのアンデプロイ

例: 指定のデプロイメントのサーバーグループすべてからアプリケーションをアンデプロイする

```
undeploy DEPLOYMENT.war --all-relevant-server-groups
```

例: 特定のサーバーグループからアプリケーションをアンデプロイする

```
undeploy DEPLOYMENT.war --server-groups=SERVER_GROUP_NAME --keep-content
```

指定のデプロイメントの他のサーバーグループ向けにコンテンツをリポジトリーに保持するため、**--keep-content** パラメーターが必要になります。

10.41. スタンドアロンサーバーからのアプリケーションのアンデプロイ

```
undeploy DEPLOYMENT.war
```

10.42. ホスト名の更新

例: 管理対象ドメインでホストの名前を更新する

```
/host=EXISTING_HOST_NAME:write-attribute(name=name,value=NEW_HOST_NAME)
reload --host=EXISTING_HOST_NAME
```

変更を反映するには、ホストをリロードする必要があります。

10.43. 添付のアップロード

ファイルストリームを許可する管理操作にローカルファイルを添付としてアップロードすることができます。たとえば、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内容を展開形式 (exploded) の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}]
```

デプロイメントにファイルをアップロードするための詳細については、『**設定ガイド**』の「展開形式のデプロイメントへのコンテンツの追加」を参照してください。

10.44. サーバーログの表示

```
/subsystem=logging/log-file=SERVER_LOG_NAME:read-log-file
```

付録A リファレンス資料

A.1. 管理 CLI の起動時の引数

以下の表には、**jboss-cli** スクリプトに渡して管理 CLI を起動できる引数が記載されています。

表A.1 管理 CLI の引数

引数	説明
--bind	CLI がバインドされるアドレスを指定します。指定のない場合、CLI は必要に応じて自動的にアドレスを選択します。
--command	CLI セッションで実行される単一のコマンドまたは操作を指定します。コマンドまたは操作の実行後、CLI はセッションを即座に終了します。
--command-timeout	コマンドが完了するまで待機する最大時間。**0** の値はタイムアウトにならないことを意味します。デフォルトではタイムアウトは発生しません。
--commands	CLI セッションで実行されるコマンドおよび操作のコンマ区切りリストを指定します。最後のコマンドの実行後またはコマンドが失敗した場合、CLI セッションが終了します。
--connect, -c	CLI が起動時にコントローラーに接続するよう指示します。この引数を使用すると、後で **connect** コマンドを実行する必要がなくなります。
--controller	起動時に **--connect** オプションが指定された場合、または管理 CLI で引数のない **connect** コマンド実行された場合に接続するデフォルトのコントローラーホスト、ポート、およびプロトコル。デフォルトのホストは **localhost**、デフォルトのポートは **9990**、デフォルトのプロトコルは **remote+http** になります。この代わりに、ホスト、ポート、およびプロトコル情報が含まれるコントローラーエイリアスを指定することもできます。
--echo-command	非対話モードで実行されたコマンドの出力にプロンプトおよびコマンドが含まれるようにします。
--error-on-interact	非対話モードでセキュリティー関連の入力のプロンプトを無効にします。CLI プロセスの続行に入力が必要な場合、プロセスは突然エラーによって終了されます。
--file	非対話的に実行されるコマンドおよび操作 (1 行ずつ記述) が含まれるファイルへのパスを指定します。最後のコマンドの実行後またはコマンドが失敗した場合、CLI セッションが終了します。

引数	説明
--gui	コマンドラインインターフェースの上に構築された GUI を起動します。この GUI を使用した JBoss EAP 管理 CLI との対話はサポートされないため、設定の表示や CLI コマンドの構築のみに使用してください。
--help, -h	ヘルプメッセージを表示します。
--no-local-auth	ファイルシステムを使用してトークンの交換によって管理されるサーバーへローカル実行されたことを CLI が実証できる、ローカル認証メカニズムを無効にします。
--password, -p	コントローラーへの接続中に認証のパスワードを指定します。引数の指定がなく、認証が必須である場合、 **`connect`** コマンドの実行時にユーザーはパスワードの入力を求められます。
--properties	プロパティーと値のペアが含まれるプロパティーファイルへのパスを指定し、システムプロパティーを定義します。プロパティーファイルは標準の **`KEY=VALUE`** 構文を使用します。
--timeout	接続に成功するまで待機する時間をミリ秒単位で指定します。デフォルトは **`5000`** です。
--user, -u	コントローラーにユーザー認証が必要な場合にユーザー名を指定します。引数の指定がなく、認証が必須である場合、**`connect`** コマンドの実行時にユーザーはユーザー名の入力を求められます。ユーザーが指定された場合、ローカル認証が自動的に無効になります。
--version	アプリケーションサーバーバージョンと環境情報を表示します。

A.2. 管理 CLI のバッチモードコマンド

以下の表には、管理 CLI で使用できるバッチ処理のコマンドが記載されています。

表A.2 管理 CLI のバッチモードコマンド

コマンド名	説明
clear-batch	現在のアクティブなバッチから既存のコマンドラインをすべて削除します。
discard-batch	現在のアクティブなバッチを破棄し、バッチモードを終了します。
edit-batch-line	編集する行番号と編集されたコマンドを提供して、現在のバッチの行を編集します。例: **`edit-batch-line 2 data-source disable --name=ExampleDS`**

コマンド名	説明
holdback-batch	現在のバッチを延期または格納します。このコマンドを引数なしで使用すると、名前のない保留されたバッチが作成されます。この保留されたバッチに戻るには、CLI コマンドラインで再度 **batch** を入力します。名前のない保留されたバッチは 1 つのみ存在できます。 任意で、**holdback_name** 引数を使用すると格納するバッチの名前を提供できます。名前付きのバッチに戻るには、**heldback_name** を **batch** コマンドに渡します。 保留されたすべてのバッチのリストを表示するには、**batch -l** コマンドを使用します。
list-batch	現在アクティブなバッチのコマンドをすべて表示します。
move-batch-line	最初の引数としたい行番号と 2 つ目の引数としての新たポジションを指定して、バッチの行の順番を変えます。例: **move-batch-line 3 1**
remove-batch-line	指定行のバッチコマンドを削除します。例: **remove-batch-line 3**
run-batch	現在アクティブなバッチを実行します。バッチが正常に実行されると、バッチが破棄され、CLI はバッチモードを終了します。

A.3. 管理 CLI コマンド

以下の表には、管理 CLI コマンドとそれらの目的が記載されています。使用方法や引数の詳細は、特定のコマンドで **--help** 引数を使用します。

表A.3 管理 CLI コマンド

コマンド	説明
alias	**NAME=VALUE** の書式でエイリアスを定義します。引数の指定がない場合、エイリアスのリストが表示されます。
batch	新しいバッチを作成して、バッチモードを開始します。名前のない保留されたバッチがある場合、そのバッチは再アクティベートされます。名前付きの保留されたバッチがある場合は、**heldback_name** を指定して再アクティベートします。
cd	指定されたパスへの変更。
clear	画面を消去します。

コマンド	説明
command	既存の汎用型コマンドの追加、削除、およびリストを可能にします。汎用型コマンドは、特定のノード型に割り当てられ、その型のインスタンスで実行できる操作の実行を可能にします。また、既存のインスタンスでその型によって公開されるプロパティーの編集も可能にします。
connect	管理 CLI の起動時に、指定のプロトコルを使用して指定のホストおよびポート上でコントローラーに接続します。指定のない場合、デフォルトのホストは **`localhost`**、デフォルトのポートは **`9990`**、デフォルトのプロトコルは **`http`** になります。
connection-factory	**`messaging-activemq`** サブシステムで接続ファクトリーを管理します。
connection-info	現在のサーバーへの接続に関する情報を表示します。
data-source	**`datasources`** サブシステムでデータソース設定を管理します。
deploy	**`file_path`** によって指定されたアプリケーションをデプロイするか、リポジトリーに事前に存在するアプリケーションを有効にします。引数の指定がない場合、既存のデプロイメントがすべて表示されます。
deployment-info	個別のデプロイメントに関する情報や、複数のデプロイメントに関する情報を表示します。
deployment-overlay	デプロイメントオーバーレイを管理します。引数の指定がない場合、既存のデプロイメントオーバーレイがすべて表示されます。
echo	指定のテキストをコンソールに出力します。
echo-dmr	コマンドの DMR リクエストまたは引数に渡された操作をビルドし、**`toString()`** 形式で echo を実行します。
help	ヘルプメッセージを表示します。**`--commands`** 引数を使用すると、利用可能なコマンドの一覧を表示することができます。
history	メモリーの CLI コマンド履歴を表示し、履歴の拡張が有効または無効であるかを表示します。引数と使用すると、必要時に履歴を消去、無効化、および有効化できます。
if	if-else 制御フローを開始します。
jdbc-driver-info	インストールされた JDBC ドライバーに関する情報を表示します。
jms-queue	**`messaging-activemq`** サブシステムの JMS キューを管理します。

コマンド	説明
jms-topic	**`messaging-activemq`** サブシステムの JMS トピックを管理します。
ls	ノードパスの内容を表示します。**`-l`** スイッチを使用すると結果を 1 行ずつ表示できます。
module	モジュールを追加または削除します。このコマンドは テクノロジープレビュー としてのみ提供されます。
patch	パッチをサーバーに適用またはロールバックします。
pwd	現在の作業ノードの完全ノードパスを出力します。
quit	コマンドラインインターフェースを終了します。
read-attribute	値を表示し、引数によっては管理されたリソースの属性の詳細も表示します。
read-operation	指定された操作の詳細を表示します。指定がない場合は使用できる操作をすべて表示します。
reload	**`:reload`** 操作リクエストをサーバー/ドメインコントローラーに送信し、コントローラーが接続を閉じるまで待機した後、制御をクライアントに戻します。
rollout-plan	保存されたロールアウトプランを管理します。
run-batch	現在アクティブなバッチをバッチモードの間に実行します。バッチモードでない間は、**`--file`** 引数を使用してファイルの内容をバッチとして実行できます。
set	指定の名前の変数を指定の値で初期化します。
shutdown	**`:shutdown`** 操作リクエストをサーバー/ドメインコントローラーに送信し、コントローラーが接続を閉じるまで待機します。
try	try-catch-finally 制御フローを開始します。
unalias	指定のエイリアスを削除します。
undeploy	指定の名前のアプリケーションをアンデプロイします。引数によってはコンテンツリポジトリーからその内容を削除することもあります。引数の指定がない場合、既存のデプロイメントがすべて表示されます。
unset	指定の名前を持つ既存の変数を削除します。

コマンド	説明
version	アプリケーションサーバーバージョンと環境情報を出力します。
xa-data-source	**datasources** サブシステムで XA データソース設定を管理します。

A.4. 管理 CLI 操作

以下の表には、ルート (/) で使用できる管理 CLI 操作が記載されています。特定のリソースに使用できる実際の操作はリソースごとに異なり、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) によっても異なります。

操作はコロン (:) を使用して呼び出されます。リソースに使用できる操作は、**read-operation-names** 操作を使用するか、コロンの後でタブ補完を使用すると公開できます。操作の詳細は **read-operation-description** 操作を使用すると表示できます。例を以下に示します。

```
:read-operation-description(name=write-attribute)
```

表A.4 管理 CLI 操作

操作名	説明
add-namespace	名前空間接頭辞のマッピングを **namespaces** 属性のマップに追加します。
add-schema-location	スキーマロケーションのマッピングを **schema-locations** 属性のマップに追加します。
clean-obsolete-content	コンテンツリポジトリーから参照されなくなったクリーンなコンテンツ。
delete-snapshot	snapshots ディレクトリーからサーバー設定のスナップショットを削除します。
full-replace-deployment	利用可能なコンテンツの一覧に以前アップロードしたデプロイメントのコンテンツを追加し、ランタイムで同じ名前の既存コンテンツを置き換え、利用可能なコンテンツの一覧からこの置き換えたコンテンツを削除します。
list-add	list 属性へエントリーを追加します。
list-clear	list 属性からすべてのエントリーを消去します。
list-get	list 属性からエントリーを取得します。
list-remove	list 属性からエントリーを削除します。

操作名	説明
list-snapshots	snapshots ディレクトリーに保存されたサーバー設定のスナップショットを一覧表示します。
map-clear	map 属性からすべてのエントリーを消去します。
map-get	map 属性からエントリーを取得します・
map-put	map 属性にエントリーを追加します。
map-remove	map 属性からエントリーを削除します。
product-info	現在のサーバーインストールの概要を返します。
query	リソースをクエリーします。
read-attribute	選択したリソースの属性の値を表示します。
read-attribute-group	選択したグループの属性の値を表示します。
read-attribute-group-names	選択したリソースの配下にある属性グループの名前をすべて表示します。
read-children-names	指定の型を持つ選択したリソースの配下にある子の名前をすべて表示します。
read-children-resources	指定の型を持つリソースの子リソースすべてに関する情報を表示します。
read-children-types	選択したリソースの配下にある子すべての型名を表示します。
read-config-as-xml	現在の設定を XML 形式で表示します。
read-operation-description	指定リソースの操作の詳細を表示します。
read-operation-names	指定リソースに使用できる操作すべての名前を表示します。
read-resource	リソースの属性値と、その子リソースに関する基本情報または完全情報を表示します。
read-resource-description	リソースの属性、子の型、および操作の詳細を表示します。
reload	すべてのサービスを終了し、再起動することでサーバーをリロードします。
reload-servers	ドメインで現在稼働中のサーバーをすべてリロードします。

操作名	説明
remove-namespace	**`namespaces`** 属性マップから名前空間接頭辞のマッピングを削除します。
remove-schema-location	**`schema-locations`** 属性マップからスキーマロケーションのマッピングを削除します。
replace-deployment	ランタイムの既存のコンテンツを新しいコンテンツに置き換えます。新しいコンテンツを事前にデプロイメントコンテンツリポジトリーにアップロードする必要があります。
resolve-expression	式を入力 (または式へ解析できる文字列) として受け入れ、ローカルシステムプロパティーおよび環境変数に対して解決します。
resolve-expression-on-domain	式を入力 (または式へ解析できる文字列) として受け入れ、ドメインのすべてのサーバー上でローカルシステムプロパティーおよび環境変数に対して解決します。
resolve-internet-address	インターフェース解決基準を取り、その基準と一致するローカルマシンの IP アドレスを見つけます。一致する IP アドレスが見つからない場合は失敗します。
restart-servers	このドメインで稼働中のサーバーをすべて再起動します。
resume	中断されたサーバーでの通常操作を再開します。
resume-servers	ドメインのすべてのサーバー上で処理を再開します。
shutdown	**`System.exit(0)`** への呼び出しにより、サーバーをシャットダウンします。
start-servers	管理対象ドメインで現在稼働していない設定済みのサーバーをすべて開始します。
stop-servers	管理対象ドメインで現在稼働しているサーバーをすべて停止します。
suspend	サーバーの操作を正常に中断します。現在のリクエストはすべて通常どおり完了しますが、新しいリクエストは許可されません。
suspend-servers	ドメインのサーバーをすべて中断します。現在の操作はすべて完了し、新しい操作を受け入れません。
take-snapshot	サーバー設定のスナップショットを作成し、snapshots ディレクトリーに保存します。
undefine-attribute	選択したリソースの属性値を **`undefined`** に設定します。

操作名	説明
upload-deployment-bytes	含まれたバイトアレイのデプロイメントコンテンツをデプロイメントコンテンツリポジトリーに追加すべきであることを示します。この操作は、コンテンツがランタイムにデプロイされるべきであることを示すものではありません。
upload-deployment-stream	対象入力ストリームインデックスで利用可能なデプロイメントコンテンツをデプロイメントコンテンツレポジトリーに追加すべきか指定します。この操作は、コンテンツをランタイムにデプロイすべきかは指定していません。
upload-deployment-url	対象の URL で利用可能なデプロイメントコンテンツをデプロイメントコンテンツリポジトリーに追加すべきかを指定します。この操作は、コンテンツをランタイムにデプロイすべきかは指定しません。
validate-address	指定のアドレスを持つリソースが存在するかどうかを確認します。
validate-operation	説明にしたがい、操作が有効であるかを検証します。存在するエラーは、操作の **`failure-description`** に表示されます。
whoami	現在認証されたユーザーの ID を返します。
write-attribute	選択したリソースの属性値を設定します。

A.5. リソース属性の詳細

`read-resource-description` 操作はリソースの属性と、属性の詳細を表示します。以下の表には、属性との関連に応じて返される可能性があるフィールドのリストが記載されています。

表A.5 リソース属性の詳細

フィールド	説明
access-type	属性が読み取り専用、読み書き可能、またはメトリックであるか。有効な値は **`read-only`**、**`read-write`**、および **`metric`** です。**`metric`** は、値が永続設定に保存されず、サーバーの活動によって変更する可能性がある読み取り専用属性です。
allowed	有効な値のリスト。
alternatives	属性間の排他的な関係を定義します。この属性の値が設定されている場合、これらの属性が必要であることを示しても、**`alternatives`** フィールドに表示された属性は未定義である必要があります。
capability-reference	この属性の値が、別のリソースによって提供される指定機能の名前の動的部分を指定すること示します。これは、属性が管理モデルの別領域への参照であることを示します。

フィールド	説明
default	値が提供されなかった場合に属性に使用するデフォルト値。
description	属性のテキストでの詳細。
deprecated	この属性が非推奨となったかどうか。また、非推奨となったバージョンや非推奨となった理由も提供します。
expressions-allowed	属性の値として式を使用できるかどうか。
max	数値属性の最大値。
max-length	**STRING**、**LIST**、または **BYTES** 型の属性の最大長。
min	数値属性の最小値。
min-length	**STRING**、**LIST**、または **BYTES** 型の属性の最小長。
nillable	定義された値のない属性を受け入れるかどうか。値の定義が必要ない場合、または必要でも代替の値が定義されている場合に、属性を未定義にすることができます。このフィールドを使用すると、未定義の値の可能性に対応する必要があるかどうかを簡単に理解することができます。
required	属性に定義された値が必要であるかどうか。**true** の場合、値または代替を定義する必要があります。**false** の場合はを未定義にすることができます。
requires	属性に定義された値がある場合、このリストに定義された属性にも値がなければならないことを示します。
restart-required	**write-attribute** 操作の実行時に再起動しなければならないサービスを定義します。このフィールドには以下の値を使用できます。 • **no-services** - 再起動するサービスはありません。 • **all-services** - すべてのサービスを再起動する必要があります。 • **resource-services** - リソースに関連する一部のサービスを再起動する必要があります。 • **jvm** - JVM 全体を再起動する必要があります。
storage	属性の値が永続設定ファイルに保存されるか、またはリソースの実行中のみ存在するか。値は **configuration** または **runtime** になります。

フィールド	説明
type	属性値の型。許可される値は **BIG_DECIMAL**、**BIG_INTEGER**、**BOOLEAN**、**BYTES**、**DOUBLE**、**INT**、**LIST**、**LONG**、**OBJECT**、**PROPERTY**、および **STRING** です。
value-type	**LIST** または **OBJECT** 型の属性の追加の型情報を定義します。**LIST** 属性の **INT** の **value-type** は Java の **List<Integer>** と似ています。**OBJECT** 属性の **STRING** の **value-type** は Java の **Map<String, String>** と似ています。**OBJECT** 属性のすべての要素が同じ型でない場合、**value-type** はオブジェクトのフィールドと値を定義する完全定義の複合オブジェクトを表します。
unit	属性の値の単位 (該当する場合)。

Revised on 2018-09-02 21:45:24 EDT