Red Hat Fuse 7.8

Apache Karaf へのデプロイ

アプリケーションパッケージを Apache Karaf コンテナーにデプロイします。

Last Updated: 2023-07-25

Red Hat Fuse 7.8 Apache Karaf へのデプロイ

アプリケーションパッケージを Apache Karaf コンテナーにデプロイします。

法律上の通知

Copyright © 2023 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 ® 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 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

本ガイドでは、アプリケーションを Apache Karaf コンテナーにデプロイするオプションを説明します。

目次

パート I. 開発者ガイド 5

第1章 OSGI の概要 6
1.1. 概要 6
1.2. APACHE KARAF のアーキテクチャー 6
1.3. OSGI フレームワーク 6
1.4. OSGI サービス 7
1.5. OSGI バンドル 9

第2章 APACHEKARAF の開始と停止 11
2.1. APACHEKARAF の起動 11
2.2. APACHE KARAF の停止 13

第3章 基本的なセキュリティー 16
3.1. 基本的なセキュリティーの設定 16

第4章 APACHE KARAF のサービスとしてのインストール 19
4.1. 概要 19
4.2. KARAF のサービスとしての実行 19
4.3. SYSTEMD 19
4.4. SYSV 19
4.5. SOLARIS SMF 20
4.6. WINDOWS 20
4.7. KARAF-SERVICE.SH オプション 20

第5章 OSGI バンドルの構築 23
5.1. バンドルプロジェクトの生成 23
5.2. 既存の MAVEN プロジェクトの変更 23
5.3. WEB サービスのバンドルへのパッケージ化 25

第6章 ホットデプロイメントと手動デプロイメント 32
6.1. ホットデプロイメント 32
6.2. バンドルのホットアンデプロイ 32
6.3. 手動デプロイメント 32
6.4. BUNDLE:WATCH を使用したバンドルの自動再デプロイ 33

第7章 ライフサイクル管理 35
7.1. バンドルのライフサイクル状態 35
7.2. バンドルのインストールと解決 35
7.3. バンドルの開始と停止 36
7.4. バンドルの開始レベル 36
7.5. バンドルの開始レベルの指定 36
7.6. システム開始レベル 36

第8章 依存関係のトラブルシューティング 38
8.1. 依存関係が欠落している 38
8.2. 必要な機能またはバンドルがインストールされていない 38
8.3. IMPORT-PACKAGE ヘッダーが不完全 38
8.4. 欠落している依存関係を追跡する方法 38

第9章 機能のデプロイ 41
9.1. 機能の作成 41
9.2. カスタム機能リポジトリーの作成 41
9.3. カスタム機能リポジトリーへの機能追加 41

9.4. ローカルリポジトリー URL の機能サービスへの追加 42
9.5. 機能への依存機能の追加 43
9.6. OSGI 設定を機能追加 43
9.7. OSGI 設定の自動デプロイ 44

第10章 機能のデプロイメント 45
10.1. 概要 45
10.2. コンソールでのインストール 45
10.3. コンソールでのアンインストール 45
10.4. ホットデプロイ 45
10.5. 機能ファイルのホットアンデプロイ 46
10.6. ブート設定への機能追加 46

第11章 プレーン JAR のデプロイ 49
11.1. ラップスキームを使用した JAR の変換 49
ラップおよびインストール 49

第12章 OSGI サービス 51
12.1. BLUEPRINT コンテナー 51
12.2. サービスのエクスポート 54
12.3. サービスのインポート 58
12.4. OSGI サービスの公開 64
12.5. OSGI サービスへのアクセス 68
12.6. APACHE CAMEL との統合 71

第13章 JMS ブローカーを使用したデプロイ 74
13.1. AMQ7 クイックスタート 74
13.2. ARTEMIS コアクライアントの使用 76

第14章 フェイルオーバーのデプロイメント 77
14.1. シンプルなロックファイルシステムの使用 77
14.2. JDBC ロックシステムの使用 77
14.3. コンテナーレベルのロック 80

第15章 URL ハンドラー 82
15.1. ファイル URL ハンドラー 82
15.2. HTTP URL ハンドラー 82
15.3. MVN URL ハンドラー 82
15.4. WRAP URL ハンドラー 85
15.5. WAR URL ハンドラー 86

パート II. ユーザーガイド 89

第16章 APACHE KARAF のデプロイガイドの概要 90
16.1. FUSE 設定の概要 90
16.2. OSGI の設定 90
16.3. 設定ファイル 90
16.4. 高度な UNDERTOW 設定 91
16.5. 設定ファイルの命名規則 92
16.6. JAVA オプションの設定 93
16.7. 設定コンソールのコマンド 93
16.8. JMX CONFIGMBEAN 93
16.9. コンソールの使用 94
16.10. プロビジョニング 102

第17章 リモート接続を使用したコンテナーの管理 117

17.1. リモートアクセスのコンテナーの設定 117
17.2. リモートによる接続および切断 117
17.3. リモートコンテナーの停止 123

第18章 MAVEN でのビルド 125
18.1. MAVEN ディレクトリー構造 125
18.2. APACHE KARAF の BOM ファイル 127

第19章 MAVEN INDEXER プラグイン 129
19.1. LOG 129

第20章 セキュリティー 145
20.1. レルム 145

パート I. 開発者ガイド

このパートには、開発者向けの情報が含まれています。

第1章 OSGI の概要

概要

OSGi 仕様は、複雑なアプリケーションのビルド、デプロイ、および管理を簡素化するランタイムフレームワークを定義することで、モジュラーアプリケーションの開発をサポートします。

1.1. 概要

Apache Karaf は、バンドルをデプロイして管理する OSGi ベースのランタイムコンテナーです。Apache Karaf は、ネイティブオペレーティングシステムの統合も提供し、ライフサイクルがオペレーティングシステムにバインドされるようにオペレーティングシステムに統合できます。

Apache Karaf の構造は次のとおりです。

- Apache Karaf: OSGi コンテナー実装の周りのラッパーレイヤーで、OSGi コンテナーをランタイムサーバーとしてデプロイすることをサポートします。Fuse が提供するランタイム機能には、ホットデプロイメント、運用、管理機能が含まれます。
- OSGi Framework: 依存関係やバンドルのライフサイクルの管理など、OSGi 機能を実装します。

1.2. APACHE KARAF のアーキテクチャー

Apache Karaf は、以下の機能で OSGi レイヤーを拡張します。

- **コンソール** - コンソールはサービスを管理し、アプリケーションとライブラリーをインストールして管理し、Fuse ランタイムと対話します。Fuse のインスタンスを管理するコンソールコマンドを提供します。Apache Karaf コンソールリファレンス を参照してください。
- **ロギング** - ロギングサブシステムは、ログレベルを表示、閲覧、および変更するためのコンソールコマンドを提供します。
- **デプロイメント** - **bundle:install** および **bundle:start** コマンドを使用した OSGi バンドルの手動デプロイメントと、アプリケーションのホットデプロイメントの両方をサポートします。「ホットデプロイメント」 を参照してください。
- **プロビジョニング**: アプリケーションとライブラリーをインストールするための複数のメカニズムを提供します。9章 *機能のデプロイ* を参照してください。
- **設定** - **InstallDir/etc** フォルダーに保存されるプロパティーファイルは継続的に監視され、それらの変更は自動的に設定可能な間隔で関連するサービスに伝播されます。
- **Blueprint**: OSGi コンテナーとの対話を簡素化する依存関係注入フレームワークです。たとえば、OSGi サービスのインポートおよびエクスポート用の標準 XML 要素を提供します。Blueprint 設定ファイルをホットデプロイメントフォルダーにコピーすると、Red Hat Fuse は OSGi バンドルを時に生成し、Blueprint コンテキストをインスタンス化します。

1.3. OSGI フレームワーク

1.3.1. 概要

OSGi Alliance は、 OSGi Service Platform リリース 4 の機能の定義を担当する独立した組織です。OSGi Service Platform は、複雑なソフトウェアアプリケーションのビルド、デプロイ、および管理を簡素化するオープン仕様のセットです。

OSGi テクノロジーは通常、Java の動的モジュールシステムと呼ばれます。OSGi は、バンドルを使用して Java コンポーネントをデプロイし、依存関係、バージョン管理、クラスパス制御、クラ出力ディングを処理する Java のフレームワークです。OSGi のライフサイクル管理により、JVM をシャットダウンせずにバンドルのロード、開始、および停止ができます。

OSGi は、Java に最適なランタイムプラットフォーム、またはサービス用の優れたクラ出力ディングアーキテクチャー、およびレジストリーを提供します。バンドルはサービスをエクスポートしてプロセスを実行し、それらの依存関係を管理できます。各バンドルの要件は OSGi コンテナーで管理できます。

Fuse は Apache Felix をデフォルトの OSGi 実装として使用します。フレームワーク層は、バンドルをインストールするコンテナーを形成します。このフレームワークは、動的でスケーラブルな方法でバンドルのインストールおよび更新を管理し、バンドルとサービス間の依存関係を管理します。

1.3.2. OSGi アーキテクチャー

OSGi フレームワークには、以下が含まれます。

- **バンドル** - アプリケーションを設定する論理モジュール。「OSGi バンドル」 を参照してください。
- **サービスレイヤー**: モジュールおよびその含まれるコンポーネント間の通信を提供します。このレイヤーはライフサイクルレイヤーと密接に統合されています。「OSGi サービス」 を参照してください。
- **ライフサイクルレイヤー**: ベースとなる OSGi フレームワークへのアクセスを提供します。このレイヤーは個別のバンドルのライフサイクルを処理するため、バンドルの開始や停止などのアプリケーションを動的に管理できます。
- **モジュールレイヤー**: バンドルパッケージング、依存関係の解決、クラ出力ディングを管理する API を提供します。
- **実行環境**: JVM の設定。この環境では、バンドルが機能する環境を定義するプロファイルを使用します。
- セキュリティー **レイヤー**: Java 2 セキュリティーに基づく任意のレイヤー。さらに制約と機能強化が追加されています。

フレームワークの各レイヤーは、その下の層によって異なります。たとえば、ライフサイクルレイヤーにはモジュールレイヤーが必要です。モジュールレイヤーは、ライフサイクルおよびサービスレイヤーなしで使用できます。

1.4. OSGI サービス

1.4.1. 概要

OSGi サービスは、名前/値のペアとして定義されたサービスプロパティーを持つ Java クラスまたはサービスインターフェイスです。サービスのプロパティーは、同じサービスインターフェイスでサービスを提供するサービスプロバイダーを区別します。

OSGi サービスは、サービスインターフェイスで意味的に定義され、サービスオブジェクトとして実装されます。サービスの機能は、実装するインターフェイスで定義されます。そのため、異なるアプリケーションが同じサービスを実装できます。

サービスインターフェイスは、バンドルが実装ではなく、インターフェイスをバインドして対話できるようにします。サービスインターフェイスは、できるだけいくつかの実装詳細で指定する必要があります。

1.4.2. OSGi サービスレジストリー

OSGi フレームワークでは、サービスレイヤーがパブリッシュ、検索、バインドサービスモデルを使用して「OSGi バンドル」と含まれるコンポーネント間の通信を提供します。サービスレイヤーには、以下が含まれるサービスレジストリーが含まれます。

- サービスプロバイダーは、他のバンドルで使用されるフレームワークにサービスを登録します。
- サービス要求側がサービスを見つけ、サービスプロバイダーにバインドします。

サービスはバンドルによって所有され、そこから実行されます。バンドルは、サービスの実装を 1 つ以上の Java インターフェイスにあるフレームワークサービスレジストリーに登録します。そのため、サービスの機能はフレームワークの制御下にある他のバンドルで利用でき、他のバンドルはサービスを検索し、使用できます。ルックアップは、Java インターフェイスとサービスプロパティーを使用して実行されます。

各バンドルは、そのインターフェイスとプロパティーの完全修飾名を使用して、サービスレジストリーに複数のサービスを登録できます。バンドルは、LDAP 構文の名前とプロパティーを使用して、サービスレジストリーにサービスを照会します。

バンドルは、公開、検出、バインドなどのランタイムサービスの依存関係管理アクティビティーを担当します。バンドルは、バンドルにバインドされているサービスの動的な可用性 (到着または出発) に起因する変更にも適応できます。

イベント通知

サービスインターフェイスは、バンドルで作成したオブジェクトにより実装されます。バンドルには次のことができます。

- サービスを登録する
- サービスを検索する
- 登録状態が変化したときに通知を受け取る

OSGi フレームワークは、イベント通知メカニズムを提供するため、サービスレジストリーに変更が発生したときに、サービスリクエスターは通知イベントを受信できます。これらの変更には、特定のサービスの公開または取得、およびサービスが登録、変更、または登録解除された場合などが含まれます。

サービス呼び出しモデル

バンドルがサービスを使用する場合に、バンドルはサービスを検索し、通常の Java 呼び出しとして Java オブジェクトを呼び出します。したがって、サービスの呼び出しは同期しており、同じスレッドで発生します。より非同期的な処理にコールバックを使用できます。パラメーターは、Java オブジェクト参照として渡されます。XML の場合のように、マーシャリングや中間の正規形式は必要ありません。OSGi は、サービスが利用できないという問題を解決します。

OSGi フレームワークサービス

独自のサービスに加えて、OSGi フレームワークは、フレームワークの操作を管理するために以下のオプションのサービスを提供します。

- **パッケージ管理サービス**: 管理エージェントが共有パッケージのステータスを調べることで、Java パッケージ共有の管理ポリシーを定義できるようにします。また、管理エージェントがパッケージを更新し、必要に応じてバンドルを停止して再起動できるようにします。このサービスにより、管理エージェントは、エクスポートするバンドルがアンインストールされるか、更新されたときに、共有パッケージに関する決定を行うことができます。
  このサービスは、最後の更新以降に削除または更新されたエクスポート済みパッケージを更新して、特定のバンドルを明示的に解決するメソッドも提供します。このサービスは、実行時にバンドル間の依存関係を追跡することもできるため、アップグレードによって影響を受ける可能性のあるバンドルを確認できます。

- **開始レベルサービス**: 管理エージェントがバンドルの開始順序と停止順序を制御できるようにします。サービスは、各バンドルに開始レベルを割り当てます。管理エージェントは、バンドルの開始レベルを変更し、フレームワークのアクティブな開始レベルを設定できるので、適切にバンドルの開始および停止が行われます。このアクティブな開始レベル以下のバンドルだけを有効にできます。

- **URL ハンドラーサービス**: URL スキームとコンテンツハンドラーを使用して Java ランタイムを動的に拡張し、任意のコンポーネントで URL ハンドラーを追加できるようにします。

- **パーミッション管理サービス**: OSGi フレームワーク管理エージェントが特定のバンドルのパーミッションを管理し、すべてのバンドルにデフォルトを提供できるようにします。バンドルには、パーミッションセットを 1 つ含めることができ、これを使用して特権コードを実行する権限があるかを検証します。ポリシーをその場で変更したり、新しくインストールしたコンポーネントに新しいポリシーを追加したりすることで、アクセス許可を動的に操作できます。ポリシーファイルは、バンドルで実行できる内容の制御に使用されます。

- **条件付きアクセス許可管理サービス**: パーミッションの確認時に特定の条件が true または false の場合に適用できるパーミッションを使用して、アクセス許可管理サービスを拡張します。これらの条件をもとに、パーミッションの適用先のバンドルの選択肢が決まります。パーミッションは、設定後すぐに有効になります。

OSGi フレームワークサービスについては、OSGiAllianceWeb サイトの リリース 4 ダウンロードページ から入手できる **OSGiServicePlatform リリース 4** 仕様の個別の章で詳しく説明されています。

OSGi Compendium サービス

OSGi フレームワークサービスに加えて、OSGi Alliance は、標準化された Compedium サービスのセット (任意) を定義します。OSGi Compedium サービスは、ロギングや設定などのタスクの API を提供します。これらのサービスについては、OSGi Alliance Web サイトの release 4 download page から入手できる **OSGi Service Platform, Service Compendium**に記載されています。

Configuration Admin Compendium サービスは、設定情報を保持し、それを関係者に配布する中央ハブのようなものです。設定管理サービスは、デプロイされたバンドルの設定情報を指定し、バンドルが有効な場合にそのデータを確実に受信するようにします。バンドルの設定データは、名前と値のペアのリストです。「Apache Karaf のアーキテクチャー」 を参照してください。

1.5. OSGI バンドル

概要

OSGi を使用すると、アプリケーションをバンドルにモジュール化できます。各バンドルは、疎結合されており、動的に読み込み可能なクラス、JAR および設定ファイルのコレクションで、外部の依存関係を明示的に宣言します。OSGi では、バンドルが主要なデプロイメント形式です。バンドルは、JAR にパッケージ化されたアプリケーションであり、インストール、開始、停止、更新、および削除できます。

OSGi は、バンドルの開発向けに、動的かつ簡潔で、一貫性のあるプログラミングモデルを提供します。サービスの仕様 (Java インターフェイス) をその実装から切り離すことにより、開発とデプロイメントが簡素化されます。

OSGi バンドルの抽象化により、モジュールは Java クラスを共有できます。これは、再利用の静的な形式です。依存バンドルの開始時に、共有クラスが使用可能である必要があります。

バンドルは、OSGi マニフェストファイルにメタデータが含まれる JAR ファイルです。バンドルには、クラスファイルと、オプションで他のリソースおよびネイティブライブラリーが含まれます。バンドル内のどのパッケージが外部に表示されるか (エクスポートされたパッケージ)、およびバンドルに必要な外部パッケージ (インポートされたパッケージ) を明示的に宣言できます。

モジュールレイヤーは、バンドル間での Java パッケージのパッケージ化と共有、および他のバンドルからのパッケージの非表示を処理します。OSGi フレームワークは、バンドル間の依存関係を動的に解決します。フレームワークは、インポートおよびエクスポートされたパッケージと一致するようにバンドル解決を実行します。また、デプロイされたバンドルの複数のバージョンを管理することもできます。

OSGi でのクラスのロード

OSGi は、ツリーモデル (JVM で使用される) ではなく、グラフモデルをクラスのロードに使用します。バンドルは、ランタイムのクラス読み込みの競合なしに、標準化された方法でクラスを共有および再利用できます。

各バンドルには独自の内部クラスパスがあるため、必要に応じて独立したユニットとして機能できます。

OSGi でのクラス読み込みの利点は次のとおりです。

- バンドル間でクラスを直接共有します。JAR を親クラ出力ダーにプロモートする必要はありません。
- 競合することなく、同じクラスの異なるバージョンを同時にデプロイできます。

第2章 APACHEKARAF の開始と停止

概要

Apache Karaf は、サーバーを起動して停止するためのシンプルなコマンドラインツールを提供します。

2.1. APACHEKARAF の起動

Apache Karaf ランタイムをデプロイするデフォルトの方法として、アクティブなコンソールを備えたスタンドアロンサーバーとしてデプロイすることが挙げられます。コンソールなしでランタイムをバックグラウンドプロセスとしてデプロイすることもできます。

2.1.1. 環境の設定

環境を変更せずに、インストールの **bin** サブディレクトリーから直接 Karaf ランタイムを起動できます。しかし、別のフォルダーで起動する場合は、以下のように Karaf インストールの **bin** ディレクトリーを **PATH** 環境変数に追加する必要があります。

Windows

```
set PATH=%PATH%;InstallDir\bin
```

Linux/UNIX

```
export PATH=$PATH,InstallDir/bin`
```

2.1.2. コンソールモードでランタイムの起動

インストールディレクトリーから Karaf ランタイムを起動する場合は、次のコマンドを使用します。

Windows

```
bin\fuse.bat
```

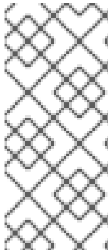
Linux/UNIX

```
./bin/fuse
```

Karaf が正しく起動すると、コンソールに次のように表示されます。

```
Red Hat Fuse starting up. Press Enter to open the shell now...
100%
[========================================================================]

Karaf started in 8s. Bundle stats: 220 active, 220 total


 ____          _   _   _       _     _____
|  _ \ ___  __| | | | | | __ _| |_  |  ___|   _ ___  ___
| |_) / _ \/ _` | | |_| |/ _` | __| | |_ | | | / __|/ _ \
|  _ <  __/ (_| | |  _  | (_| | |_  |  _|| |_| \__ \  __/
|_| \_\___|\__,_| |_| |_|\__,_|\__| |_|   \__,_|___/\___|
```

```
  Fuse (7.x.x.fuse-xxxxxx-redhat-xxxxx)
  http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181/hawtio to access the management console

Hit '<ctrl-d>' or 'shutdown' to shutdown Red Hat Fuse.

karaf@root()>
```



注記

バージョン Fuse 6.2.1 以降、コンソールモードで起動すると、2 つのプロセスが作成されます。それらは、Karaf コンソールを実行している親プロセス **./bin/karaf** と、**java** JVM で Karaf サーバーを実行している子プロセスです。ただし、シャットダウンの動作は以前と同じです。つまり、両方のプロセスを強制終了する Ctrl-D または **osgi:shutdown** のいずれかを使用して、コンソールからサーバーをシャットダウンできます。

2.1.3. サーバーモードでのランタイムの起動

サーバーモードで起動すると、ローカルコンソールなしで Apache Karaf がバックグラウンドで実行されます。次に、リモートコンソールを使用して実行中のインスタンスに接続します。詳しくは、「リモートによる接続および切断」 を参照してください。

Karaf をサーバーモードで起動するには、次のコマンドを実行します

Windows

```
bin\start.bat
```

Linux/UNIX

```
./bin/start
```

2.1.4. クライアントモードでのランタイムの起動

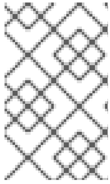
実稼働環境では、ローカルコンソールのみを使用してランタイムインスタンスにアクセスできるようにする場合があります。つまり、SSH コンソールポートを介してランタイムにリモートで接続できません。これを行うには、次のコマンドを使用して、クライアントモードでランタイムを起動します。

Windows

```
bin\fuse.bat client
```

Linux/UNIX

```
./bin/fuse client
```



注記

クライアントモードで起動すると、SSH コンソールポート (通常はポート 8101) のみが抑制されます。他の Karaf サーバーポート (例: JMX 管理 RMI ポート) は通常どおり開かれます。

2.1.5. デバッグモードでの Fuse の実行

Fuse を **デバッグモード** で実行すると、エラーをより効率的に識別して解決するのに役立ちます。このオプションはデフォルトで無効になっています。有効にすると、Fuse はポート **5005** で JDWP ソケットを起動します。

Fuse を **デバッグモード** で実行するには、3 つの方法があります。

- 「Karaf 環境変数の使用」
- 「Fuse debug の実行」
- 「Fuse debugs の実行」

2.1.5.1. Karaf 環境変数の使用

このアプローチにより、**KARAF_DEBUG** 環境変数 (**=1**) が有効になり、次にコンテナーを起動します。

```
$ export KARAF_DEBUG=1
$ bin/start
```

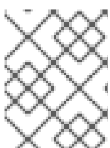
2.1.5.2. Fuse debug の実行

このアプローチでは、**suspend** オプションが **n** (no) に設定されている **debug** を実行します。

```
$ bin/fuse debug
```

2.1.5.3. Fuse debugs の実行

このアプローチでは、**suspend** オプションが **y** (yes) に設定されている **debugs** を実行します。



注記

suspend を yes に設定すると、JVM は **main()** の実行直前に一時停止し、デバッガーがアタッチされると実行を再開します。

```
$ bin/fuse debugs
```

2.2. APACHE KARAF の停止

コンソール内からまたは **stop** スクリプトを使用して、Apache Karaf のインスタンスを停止できます。

2.2.1. ローカルコンソールからのインスタンス停止

fuse または **fuse client** を実行して Karaf インスタンスを起動した場合、**karaf>** プロンプトで以下のいずれかを実行して停止できます。

- **shutdown** を入力します。
- **Ctrl**+**D** を押します。

2.2.2. サーバーモードで実行されているインスタンスの停止

以下のように **InstallDir/bin** ディレクトリーから **stop(.bat)** を呼び出すことで、ローカルに実行中の Karaf インスタンス (root コンテナー) を停止できます。

Windows

```
bin\stop.bat
```

Linux/UNIX

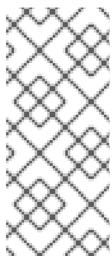
```
./bin/stop
```

Karaf **stop** スクリプトによって呼び出されるシャットダウンメカニズムは、Apache Tomcat に実装されたシャットダウンメカニズムと似ています。Karaf サーバーは、シャットダウン通知の受信に専用のシャットダウンポート (SSH ポートとは**異なる**) を開きます。デフォルトでは、シャットダウンポートはランダムに選択されますが、必要に応じて特定のポートを使用するように設定できます。

必要に応じて、**InstallDir/etc/config.properties** ファイルに以下のプロパティーを設定して、シャットダウンポートをカスタマイズできます。

karaf.shutdown.port

シャットダウンポートとして使用する TCP ポートを指定します。このプロパティーを **-1** に設定すると、ポートが無効になります。デフォルトは **0** (ランダムポート用) です。

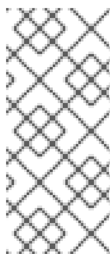


注記

bin/stop スクリプトを使用してリモートホストで実行している Karaf サーバーをシャットダウンする場合は、このプロパティーをリモートホストのシャットダウンポートと同じに設定する必要があります。ただし、この設定が **etc/config.properties** ファイルと同じホストにある Karaf サーバーにも影響を与えることに注意してください。

karaf.shutdown.host

シャットダウンポートがバインドされているホスト名を指定します。この設定は、マルチホームホストで役立つ可能性があります。デフォルトは **localhost** です。



注記

bin/stop スクリプトを使用してリモートホストで実行している Karaf サーバーをシャットダウンする場合は、このプロパティーをリモートホストのホスト名 (または IP アドレス) に設定する必要があります。ただし、この設定が **etc/config.properties** ファイルと同じホストにある Karaf サーバーにも影響を与えることに注意してください。

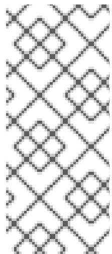
karaf.shutdown.port.file

Karaf インスタンスが起動すると、現在のシャットダウンポートがこのプロパティーで指定されたファイルに書き込まれます。**stop** スクリプトは、このプロパティーで指定されたファイルを読み取り、現在のシャットダウンポートの値を検出します。デフォルトは **${karaf.data}/port** です。

karaf.shutdown.command

シャットダウンのトリガーにシャットダウンポートに送信する必要がある UUID 値を指定します。これにより、UUID 値が公開されていない限り、基本レベルのセキュリティーが提供されます。たとえば、この値が通常のユーザーによって読み取られないように、**etc/config.properties** ファイルの読み取りを保護することができます。

Apache Karaf を初めて起動すると、ランダムな UUID 値が自動的に生成され、この設定が **etc/config.properties** ファイルの最後に書き込まれます。または、**karaf.shutdown.command** がすでに設定されている場合は、Karaf サーバーは既存の UUID 値を使用します (これにより、必要に応じて UUID 設定をカスタマイズできます)。



注記

bin/stop スクリプトを使用してリモートホストで実行している Karaf サーバーをシャットダウンする場合は、このプロパティーをリモートホストの **karaf.shutdown.command** の値と同じに設定する必要があります。ただし、この設定が **etc/config.properties** ファイルと同じホストにある Karaf サーバーにも影響を与えることに注意してください。

2.2.3. リモートインスタンスの停止

「リモートコンテナーの停止」 で説明されているように、リモートホストで実行されているコンテナーインスタンスを停止できます。

第3章 基本的なセキュリティー

この章では、Karaf を初めて起動する前のセキュリティー設定に関する基本的な手順を説明します。デフォルトでは、Karaf は安全ですが、Kafka のサービスにはリモートアクセスできません。この章では、Karaf で公開されているポートへのアクセスをセキュリティーを確保しながら有効にする方法について説明します。

3.1. 基本的なセキュリティーの設定

3.1.1. 概要

Apache Karaf ランタイムは、公開されているすべてのポートにユーザー認証が必要で、最初にユーザーが定義されていないため、デフォルトではネットワーク攻撃から保護されています。つまり、Apache Karaf ランタイムは、デフォルトではリモートでアクセスできません。

ランタイムにリモートでアクセスする場合は、ここで説明するように、最初にセキュリティー設定をカスタマイズする必要があります。

3.1.2. コンテナーを起動する前に

Karaf コンテナーへのリモートアクセスを有効にする場合は、コンテナーを起動する前に、セキュリティー保護された JAAS ユーザーを作成する必要があります。

3.1.3. セキュリティーが保護された JAAS ユーザーの作成

デフォルトでは、コンテナーに JAAS ユーザーが定義されていないため、リモートアクセスが事実上無効になります (ログオン不可)。

セキュアな JAAS ユーザーを作成するには、**InstallDir/etc/users.properties** ファイルを編集し、以下のように新規ユーザーフィールドを追加します。

```
Username=Password,admin
```

Username および **Password** は、新しいユーザー認証情報です。**admin** ロールは、このユーザーに、コンテナーのすべての管理機能にアクセスするための権限を付与します。

数値のユーザー名では、先頭にゼロを使用して定義しないでください。先頭にゼロを指定したユーザー名は常にログイン試行に失敗します。これは、入力が数値であるように見える場合に、コンソールが使用する Karaf シェルにより、先頭のゼロが取り除かれるためです。以下に例を示します。

```
karaf@root> echo 0123
123
karaf@root> echo 00.123
0.123
karaf@root>
```

警告

強力なパスワードを使用してカスタムユーザーの資格情報を定義することを強くお勧めします。

3.1.4. ロールベースのアクセス制御

Karaf コンテナーは、ロールベースのアクセス制御をサポートします。このロールベースのアクセス制御は、JMX プロトコル、Karaf コマンドコンソール、および FuseManagement コンソールを介したアクセスを規制します。ユーザーにロールを割り当てる場合に、標準のロールのセットから選択できます。これにより、表3.1「アクセス制御の標準的なロール」 で説明されているアクセスレベルが提供されます。

表3.1 アクセス制御の標準的なロール

ロール	説明
viewer	コンテナーへの読み取り専用アクセスを許可します。
manager	アプリケーションのデプロイや実行を行う通常のユーザーに、適切なレベルで読み取り/書き込みアクセスを許可します。ただし、機密性の高いコンテナー設定オプションへのアクセスをブロックします。
admin	コンテナーへのアクセスを無制限に許可します。
ssh	SSH ポートを介したリモートコンソールアクセスの許可を付与します。

ロールベースのアクセス制御の詳細は、ロールベースのアクセス制御 を参照してください。

3.1.5. Apache Karaf コンテナーによって公開されたポート

次のポートがコンテナーによって公開されます。

- **コンソールポート**: Apache Karaf シェルコマンドを使用して、コンテナーインスタンスのリモート制御を有効にします。このポートはデフォルトで有効になっており、JAAS 認証と SSH の両方で保護されています。
- **JMX ポート**: JMX プロトコルを介したコンテナーの管理を可能にします。このポートはデフォルトで有効になっており、JAAS 認証によって保護されています。
- **Web コンソールポート**: Web コンソールサーブレットをホストできる組み込みの Undertow コンテナーにアクセスできます。デフォルトでは、Fuse Console は Undertow コンテナーにインストールされています。

3.1.6. リモートコンソールポートの有効化

次の両方の条件が当てはまる場合はいつでも、リモートコンソールポートにアクセスできます。

- JAAS は、少なくとも 1 セットのログイン資格情報で設定されています。
- クライアントモードでは、Karaf ランタイムは開始されて **いません** (クライアントモードはリモートコンソールポートを完全に無効にします)。

たとえば、コンテナーが実行されているのと同じマシンからリモートコンソールポートにログオンするには、次のコマンドを入力します。

```
./client -u Username -p Password
```

ここで、**Username** および **Password** は、**ssh** ロールを持つ JAAS ユーザーのクレデンシャルです。リモートポートを介して Karaf コンソールにアクセスする場合、権限は **etc/users.properties** ファイルでユーザーに割り当てられたロールによって異なります。コンソールコマンドの完全なセットにアクセスする場合は、ユーザーアカウントに **admin** ロールが必要です。

3.1.7. リモートコンソールポートのセキュリティー強化

リモートコンソールポートのセキュリティーを強化するために、次の手段を採用できます。

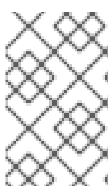
- JAAS ユーザーの資格情報に強力なパスワードが設定されていることを確認する。
- X.509 証明書をカスタマイズする (Java キーストアファイル **InstallDir/etc/host.key** をカスタムキーペアに置き換えます)。

3.1.8. JMX ポートの有効化

JMX ポートはデフォルトで有効になっており、JAAS 認証でセキュリティーが確保されています。JMX ポートにアクセスするには、少なくとも 1 セットのログイン認証情報を使用して JAAS を設定しておく必要があります。JMX ポートに接続するには、JMX クライアント (例: **jconsole**) を開き、以下の JMX URI に接続します。

```
service:jmx:rmi:///jndi/rmi://localhost:1099/karaf-root
```

また、接続するには、有効な JAAS 認証情報を JMX クライアントに渡す必要があります。



注記

一般的に、JMX URI の最後は **/karaf-ContainerName** の形式を取ります。コンテナー名を **root** から他の名前に変更する場合は、それに応じて JMX URI を変更する必要があります。

3.1.9. Fuse Console ポートのセキュリティー強化

Fuse Console は、JAAS 認証ですでに保護されています。SSL セキュリティーを追加するには、Undertow HTTP サーバーのセキュア化 を参照してください。

第4章 APACHE KARAF のサービスとしてのインストール

この章では、提供されているテンプレートを使用して、Apache Karaf インスタンスをシステムサービスとして開始する方法について説明します。

4.1. 概要

サービススクリプトテンプレートを使用すると、オペレーティングシステム固有の初期化スクリプトを使用して Karaf インスタンスを実行できます。これらのテンプレートは、**bin/contrib** ディレクトリーにあります。

4.2. KARAF のサービスとしての実行

karaf-service.sh ユーティリティーは、テンプレートのカスタマイズに役立ちます。このユーティリティーは、オペレーティングシステムとデフォルトの init システムを自動的に識別し、すぐに使用できる init スクリプトを生成します。また、**JAVA_HOME** およびその他の環境変数を設定して、環境に合わせてスクリプトをカスタマイズすることもできます。

生成されたスクリプトは、次の 2 つのファイルで設定されています。

- init スクリプト
- init 設定ファイル

4.3. SYSTEMD

karaf-service.sh ユーティリティーが **systemd** を識別すると、次の 3 つのファイルが生成されます。

- ルート Apache Karaf コンテナーを管理する **systemd** ユニットファイル。
- ルート Apache Karaf コンテナーによって使用される変数が含まれる **systemd** 環境ファイル。
- **(サポート対象外)** Apache Karaf の子コンテナーを管理するための **systemd** テンプレートユニットファイル。

たとえば、**/opt/karaf-4** にインストールされた Karaf インスタンスのサービスを設定するには、サービスに **karaf-4** という名前を付けます。

```
$ ./karaf-service.sh -k /opt/karaf-4 -n karaf-4
Writing service file "/opt/karaf-4/bin/contrib/karaf-4.service"
Writing service configuration file ""/opt/karaf-4/etc/karaf-4.conf"
Writing service file "/opt/karaf-4/bin/contrib/karaf-4@.service"
$ sudo cp /opt/karaf-4/bin/contrib/karaf-4.service /etc/systemd/system
$ sudo systemctl enable karaf-4.service
```

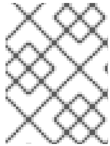
4.4. SYSV

karaf-service.sh ユーティリティーによって SysV システムが特定されると、以下の 2 つのファイルが生成されます。

- ルート Apache Karaf コンテナーを管理するための init スクリプト。
- ルート Apache Karaf コンテナーによって使用される変数が含まれる環境ファイル。

たとえば、**/opt/karaf-4** にインストールされた Karaf インスタンスのサービスを設定するには、サービスに **karaf-4** という名前を付けます。

```
$ ./karaf-service.sh -k /opt/karaf-4 -n karaf-4
Writing service file "/opt/karaf-4/bin/contrib/karaf-4"
Writing service configuration file "/opt/karaf-4/etc/karaf-4.conf"
$ sudo ln -s /opt/karaf-4/bin/contrib/karaf-4 /etc/init.d/
$ sudo chkconfig karaf-4 on
```



注記

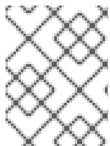
起動時にサービスの開始を有効にするには、オペレーティングのシステム Init ガイドを参照してください。

4.5. SOLARIS SMF

karaf-service.sh ユーティリティーによって Solaris オペレーティングシステムが特定されると、単一のファイルが生成されます。

たとえば、**/opt/karaf-4** にインストールされた Karaf インスタンスのサービスを設定するには、サービスに **karaf-4** という名前を付けます。

```
$ ./karaf-service.sh -k /opt/karaf-4 -n karaf-4
Writing service file "/opt/karaf-4/bin/contrib/karaf-4.xml"
$ sudo svccfg validate /opt/karaf-4/bin/contrib/karaf-4.xml
$ sudo svccfg import /opt/karaf-4/bin/contrib/karaf-4.xml
```



注記

生成された SMF 記述子は一時的なものとして定義されているため、start メソッドを実行できるのは 1 回だけです。

4.6. WINDOWS

Windows サービスとしての Apache Karaf のインストールは、**winsw** を介してサポートされます。

Apache Karaf を Windows サービスとしてインストールするには、次の手順を実行します。

1. **karaf-service-win.exe** ファイルの名前を **karaf-4.exe** に変更します。
2. **karaf-service-win.xml** ファイルの名前を **karaf-4.xml** に変更します。
3. 必要に応じてサービス記述子をカスタマイズします。
4. サービス実行可能ファイルを使用して、サービスをインストール、開始、および停止します。

以下に例を示します。

```
C:\opt\apache-karaf-4\bin\contrib> karaf-4.exe install
C:\opt\apache-karaf-4\bin\contrib> karaf-4.exe start
```

4.7. KARAF-SERVICE.SH オプション

karaf-service.sh ユーティリティーのオプションは、以下のようにコマンドラインオプションとして指定するか、環境変数を設定して指定します。

コマンドラインオプション	環境変数	説明
-k	**KARAF_SERVICE_PATH**	Karaf のインストールパス
-d	**KARAF_SERVICE_DATA**	Karaf データパス (デフォルトは **${KARAF_SERVICE_PATH}/data**)
-c	**KARAF_SERVICE_CONF**	Karaf 設定ファイル (デフォルトは **${KARAF_SERVICE_PATH}/etc/${KARAF_SERVICE_NAME}.conf**)
-t	**KARAF_SERVICE_ETC**	Karaf **etc** パス (デフォルトは **${KARAF_SERVICE_PATH}/etc}**)
-p	**KARAF_SERVICE_PIDFILE**	Karaf PID パス (デフォルトは **${KARAF_SERVICE_DATA}/${KARAF_SERVICE_NAME}.pid**)
-n	**KARAF_SERVICE_NAME**	Karaf サービス名 (デフォルトは **karaf**)
-e	**KARAF_ENV**	サービスの環境変数設定 **NAME=VALUE** を指定します (複数回指定できます)
-u	**KARAF_SERVICE_USER**	Karaf ユーザー
-g	**KARAF_SERVICE_GROUP**	Karaf グループ (デフォルトは **${KARAF_SERVICE_USER}**)
-l	**KARAF_SERVICE_LOG**	Karaf コンソールログ (デフォルトは **${KARAF_SERVICE_DATA}/log/${KARAF_SERVICE_NAME}-console.log**)
-f	**KARAF_SERVICE_TEMPLATE**	使用するテンプレートファイル
-x	**KARAF_SERVICE_EXECUTABLE**	Karaf 実行可能ファイル名 (デフォルトは **karaf –daemon** および **stop** コマンドをサポートする必要があります)

コマンドラインオプション	環境変数	説明
-h		ヘルプメッセージ

第5章 OSGI バンドルの構築

概要

この章では、Maven を使用して OSGi バンドルを構築する方法について説明します。バンドルを構築する場合に、Maven バンドルプラグインは、OSGi バンドルヘッダーの生成を自動化できるため、重要なロールを果たします (そうでない場合は面倒な作業になります)。完全なサンプルプロジェクトを生成する Maven アーキタイプは、バンドルプロジェクトの開始点として使用することもできます。

5.1. バンドルプロジェクトの生成

5.1.1. Maven アーキタイプを使用したバンドルプロジェクトの生成

すばやく開始できるように、Maven アーキタイプを呼び出して、Maven プロジェクトの初期アウトラインを生成できます (Maven アーキタイプはプロジェクトウィザードに類似しています)。次の Maven アーキタイプは、OSGi バンドルを構築するためのプロジェクトを生成します。

5.1.2. Apache Camel アーキタイプ

Apache Camel OSGi アーキタイプは、OSGi コンテナーにデプロイできるルートを構築するためのプロジェクトを作成します。

以下の例は、Maven アーキタイプコマンドに座標 **GroupId:ArtifactId:Version** を使用して **camel-blueprint** プロジェクトを生成する方法を表しています。

```
mvn archetype:generate \
 -DarchetypeGroupId=org.apache.camel.archetypes \
 -DarchetypeArtifactId=camel-archetype-blueprint \
 -DarchetypeVersion=2.23.2.fuse-780036-redhat-00001
```

このコマンドの実行後、Maven は **GroupId**、**ArtifactId**、および **Version** を指定するよう要求します。

5.1.3. バンドルの構築

デフォルトでは、前述のアーキタイプは新しいディレクトリーにプロジェクトを作成します。その名前は、指定されたアーティファクト ID (**ArtifactId**) と同じです。新規プロジェクトで定義されたバンドルをビルドするには、コマンドプロンプトを開いてプロジェクトディレクトリー (つまり **pom.xml** ファイルが含まれるディレクトリー) に移動し、以下の Maven コマンドを入力します。

```
mvn install
```

このコマンドは、すべての Java ソースファイルをコンパイルし、**ArtifactId/target** ディレクトリーの下にバンドル JAR を生成した後、ローカルの Maven リポジトリーで生成された JAR をインストールします。

5.2. 既存の MAVEN プロジェクトの変更

5.2.1. 概要

すでに Maven プロジェクトがあり、OSGi バンドルを生成するように変更する場合は、以下の手順を実行します。

1. 「パッケージタイプのバンドルへの変更」
2. 「バンドルプラグインの POM への追加」.
3. 「バンドルプラグインのカスタマイズ」.
4. 「JDK コンパイラーのバージョンのカスタマイズ」

5.2.2. パッケージタイプのバンドルへの変更

プロジェクトの **pom.xml** ファイルでパッケージタイプを **bundle** に変更して、OSGi バンドルを生成するよう Maven を設定します。以下の例のように、**packaging** 要素の内容を **bundle** に変更します。

```
<project ... >
  ...
  <packaging>bundle</packaging>
  ...
</project>
```

この設定により、Maven バンドルプラグイン **maven-bundle-plugin** を選択され、このプロジェクトのパッケージ化が実行されます。ただし、この設定自体は、バンドルプラグインを POM に明示的に追加するまで効果がありません。

5.2.3. バンドルプラグインの POM への追加

Maven バンドルプラグインを追加するには、以下のサンプル **plugin** 要素をコピーし、プロジェクトの **pom.xml** ファイルの **project/build/plugins** セクションに貼り付けます。

```
<project ... >
  ...
  <build>
    <defaultGoal>install</defaultGoal>
    <plugins>
      ...
    <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>3.3.0</version>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>${project.groupId}.${project.artifactId}
            </Bundle-SymbolicName>
            <Import-Package>*</Import-Package>
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

バンドルプラグインは **instructions** 要素の設定によって設定されます。

5.2.4. バンドルプラグインのカスタマイズ

Apache CXF のバンドルプラグインの設定に関する具体的な推奨事項は、「Web サービスのバンドルへのパッケージ化」 を参照してください。

5.2.5. JDK コンパイラーのバージョンのカスタマイズ

ほとんどの場合、POM ファイルで JDK バージョンを指定する必要があります。コードでジェネリック、静的インポートなどの Java 言語の最新機能を使用しており、POM で JDK バージョンをカスタマイズしていない場合、Maven はソースコードのコンパイルに失敗します。**JAVA_HOME** および **PATH** 環境変数を JDK の正しい値に設定するだけでは**不十分**で、POM ファイルも変更する必要があります。

JDK 1.8 で導入された Java 言語機能を受け入れるように POM ファイルを設定するには、以下の **maven-compiler-plugin** プラグイン設定を POM に追加します (存在しない場合)。

```
<project ... >
  ...
  <build>
    <defaultGoal>install</defaultGoal>
    <plugins>
      ...
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

5.3. WEB サービスのバンドルへのパッケージ化

5.3.1. 概要

このセクションでは、Apache CXF アプリケーションの既存の Maven プロジェクトを変更して、プロジェクトが Red Hat Fuse OSGi コンテナーでのデプロイメントに適した OSGi バンドルを生成する方法について説明します。Maven プロジェクトを変換するには、プロジェクトの POM ファイルとプロジェクトの Blueprint ファイル (**META-INF/spring** にある) を変更する必要があります。

5.3.2. バンドルを生成するための POM ファイルの変更

バンドルを生成するように Maven POM ファイルを設定するには、基本的に 2 つの変更を行う必要があります。1 つ目は、POM のパッケージ型を **bundle** に変更します。2 つ目は、Maven バンドルプラグインを POM に追加します。詳細は、「バンドルプロジェクトの生成」 を参照してください。

5.3.3. 必須のインポートパッケージ

アプリケーションで Apache CXF コンポーネントを使用するには、それらのパッケージをアプリケーションのバンドルにインポートする必要があります。Apache CXF の依存関係は複雑であるため、

Maven バンドルプラグインや **bnd** ツールに依存して必要なインポートを自動的に決定することができません。それらを明示的に宣言する必要があります。

次のパッケージをバンドルにインポートする必要があります。

```
javax.jws
javax.wsdl
javax.xml.bind
javax.xml.bind.annotation
javax.xml.namespace
javax.xml.ws
org.apache.cxf.bus
org.apache.cxf.bus.spring
org.apache.cxf.bus.resource
org.apache.cxf.configuration.spring
org.apache.cxf.resource
org.apache.cxf.jaxws
org.springframework.beans.factory.config
```

5.3.4. Maven バンドルプラグインの手順例

例5.1「必須のインポートパッケージの設定」 は、必須パッケージをインポートするように POM で Maven バンドルプラグインを設定する方法を示しています。必須のインポートパッケージは、**Import-Package** 要素内にコンマ区切りリストとして存在します。リストの最後の要素にワイルドカード (*) があることに注意してください。ワイルドカードにより、現在のバンドルの Java ソースファイルがスキャンされ、インポートする必要のある追加のパッケージが検出されます。

例5.1 必須のインポートパッケージの設定

```
<project ... >
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            ...
            <Import-Package>
              javax.jws,
              javax.wsdl,
              javax.xml.bind,
              javax.xml.bind.annotation,
              javax.xml.namespace,
              javax.xml.ws,
              org.apache.cxf.bus,
              org.apache.cxf.bus.spring,
              org.apache.cxf.bus.resource,
              org.apache.cxf.configuration.spring,
              org.apache.cxf.resource,
              org.apache.cxf.jaxws,
              org.springframework.beans.factory.config,
              *
```

```
                    </Import-Package>
                    ...
                </instructions>
            </configuration>
        </plugin>
    </plugins>
</build>
...
</project>
```

5.3.5. コード生成プラグインの追加

Web サービスプロジェクトでは通常、コードを生成する必要があります。Apache CXF には、JAX-WS フロントエンド用に 2 つの Maven プラグインが同梱されており、コード生成ステップをビルドに統合できます。プラグインの選択は、次のように、Java ファーストアプローチと WSDL ファーストアプローチのどちらを使用してサービスを開発するかによって異なります。

- **Java ファーストのアプローチ – cxf-java2ws-plugin** プラグインを使用します。
- **WSDL ファーストのアプローチ – cxf-codegen-plugin** プラグインを使用します。

5.3.6. OSGi 設定プロパティー

OSGi 設定管理サービスは、設定オプションを OSGi バンドルに渡すためのメカニズムを定義します。設定にこのサービスを使用する必要はありませんが、通常、バンドルアプリケーションを設定するための最も便利な方法です。Blueprint は OSGi 設定のサポートを提供し、OSGi 設定管理サービスから取得した値を使用して Blueprint ファイル内の変数を置き換えることができるようにします。

OSGi 設定プロパティーの使用方法の詳細は、「バンドルプラグインの設定」と「OSGi 設定を機能追加」 を参照してください。

5.3.7. バンドルプラグインの設定

概要

バンドルプラグインを機能させるために必要な情報はほぼありません。必要なすべてのプロパティーは、デフォルト設定を使用して有効な OSGi バンドルを生成します。

デフォルト値のみを使用して有効なバンドルを作成できますが、値の一部を変更することをお勧めします。プラグインの **instructions** 要素内のほとんどのプロパティーを指定できます。

設定プロパティー

一般的に使用される設定プロパティーのいくつかは次のとおりです。

- Bundle-SymbolicName
- Bundle-Name
- Bundle-Version
- Export-Package

- Private-Package
- Import-Package

バンドルのシンボリック名の設定

デフォルトでは、バンドルプラグインは **Bundle-SymbolicName** プロパティーの値を **groupId + "." + artifactId** に設定します。ただし、以下の例外があります。

- **groupId** にセクションが 1 つしかない (ドットがない) 場合には、クラスを含む最初のパッケージ名が返されます。
たとえば、グループ ID が **commons-logging:commons-logging** の場合、バンドルのシンボリック名は **org.apache.commons.logging** です。
- **ArtifactId** が **groupId** の最後のセクションと同じ場合には、**groupId** が使用されます。
たとえば、POM がグループ ID およびアーティファクト ID を **org.apache.maven:maven** として指定すると、バンドルのシンボリック名は **org.apache.maven** になります。
- **ArtifactId** が **groupId** の最後のセクションで始まる場合には、その部分は削除されます。
たとえば、POM がグループ ID およびアーティファクト ID を **org.apache.maven:maven-core** として指定すると、バンドルのシンボリック名は **org.apache.maven.core** になります。

バンドルのシンボリック名に独自の値を指定するには、例5.2「バンドルのシンボリック名の設定」に示すように、プラグインの **instructions** 要素に **Bundle-SymbolicName** の子を追加します。

例5.2 バンドルのシンボリック名の設定

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
     ...
    </instructions>
  </configuration>
</plugin>
```

バンドル名の設定

デフォルトでは、バンドルの名前は **${project.name}** に設定されます。

バンドルの名前に独自の値を指定するには、例5.3「バンドル名の設定」に示すように、プラグインの **instructions** 要素に **Bundle-Name** の子を追加します。

例5.3 バンドル名の設定

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Bundle-Name>JoeFred</Bundle-Name>
```

```
    ...
   </instructions>
  </configuration>
</plugin>
```

バンドルのバージョンの設定

デフォルトでは、バンドルのバージョンは **${project.version}** に設定されます。ダッシュ (**-**) はピリオド (**.**) に置き換えられ、数字は 4 桁に変換されます。たとえば、**4.2-SNAPSHOT** は **4.2.0.SNAPSHOT** になります。

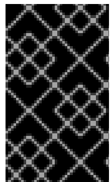
バンドルのバージョンに独自の値を指定するには、例5.4「バンドルのバージョンの設定」に示すように、プラグインの **instructions** 要素に **Bundle-Version** の子を追加します。

例5.4 バンドルのバージョンの設定

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Bundle-Version>1.0.3.1</Bundle-Version>
     ...
   </instructions>
  </configuration>
</plugin>
```

エクスポートされたパッケージの指定

デフォルトでは、OSGi マニフェストの **Export-Package** リストには、ローカルの Java ソースコード (**src/main/java** 下) のすべてのパッケージが反映されます。ただし、デフォルトのパッケージ、**.**、および **.impl** または **.internal** が含まれるすべてのパッケージを **除きます**。



重要

プラグイン設定で **Private-Package** 要素を使用し、エクスポートするパッケージの一覧を指定しない場合、デフォルトの動作ではバンドルの **Private-Package** 要素にリストされているパッケージのみが含まれます。パッケージはエクスポートされません。

デフォルトの動作では、パッケージが非常に大きくなり、非公開にしておく必要のあるパッケージがエクスポートされる可能性があります。エクスポートされるパッケージの一覧を変更するには、**Export-Package** の子をプラグインの **instructions** 要素に追加します。

Export-Package 要素は、バンドルに含まれるパッケージとエクスポートされるパッケージの一覧を指定します。パッケージ名は、***** ワイルドカードシンボルを使用して指定できます。たとえば、エントリー **com.fuse.demo.*** は、プロジェクトのクラスパスにある **com.fuse.demo** で始まるすべてのパッケージが含まれます。

除外するパッケージを指定するには、エントリーに **!** の接頭辞を追加します。たとえば、エントリー **!com.fuse.demo.private** は、パッケージ **com.fuse.demo.private** を除外します。

パッケージを除外する場合に、リスト内のエントリーの順序が重要です。リストは最初から順番に処理され、それ以降の矛盾するエントリーは無視されます。

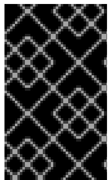
たとえば、パッケージ **com.fuse.demo.private** を除く **com.fuse.demo** で始まるすべてのパッケージを含めるには、以下のようにパッケージのリストを指定します。

```
!com.fuse.demo.private,com.fuse.demo.*
```

ただし、**com.fuse.demo.*,!com.fuse.demo.private** を使用してパッケージを一覧表示すると、**com.fuse.demo.private** は最初のパターンと一致するため、バンドルに含まれます。

プライベートパッケージの指定

バンドルに追加するパッケージの一覧を指定してそれらをエクスポート **しない** 場合、バンドルプラグイン設定に **Private-Package** 命令を追加できます。デフォルトでは、**Private-Package** 命令を指定しないと、ローカルの Java ソースのすべてのパッケージがバンドルに含まれます。



重要

パッケージが **Private-Package** 要素と **Export-Package** 要素の両方のエントリーと一致する場合は、**Export-Package** 要素が優先されます。パッケージがバンドルに追加され、エクスポートされます。

Private-Package 要素は、バンドルに含まれるパッケージの一覧を指定する点で **Export-Package** 要素と同様に機能します。バンドルプラグインは、リストを使用して、バンドルに含まれるプロジェクトのクラスパスにあるすべてのクラスを検索します。これらのパッケージはバンドルにパッケージ化されますが、(**Export-Package** 命令で選択されない限り) エクスポートされません。

例5.5「プライベートパッケージのバンドルへの追加」 はバンドルにプライベートパッケージを含めるための設定です。

例5.5 プライベートパッケージのバンドルへの追加

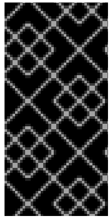
```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Private-Package>org.apache.cxf.wsdlFirst.impl</Private-Package>
     ...
    </instructions>
  </configuration>
</plugin>
```

インポートされたパッケージの指定

デフォルトでは、バンドルプラグインは OSGi マニフェストの **Import-Package** プロパティーに、バンドルのコンテンツによって参照されるすべてのパッケージのリストを反映させます。

ほとんどのプロジェクトでは通常、デフォルトの動作で十分ですが、リストに自動的に追加されないパッケージをインポートする場合があります。デフォルトの動作では、不要なパッケージがインポートされる可能性もあります。

バンドルによってインポートされるパッケージの一覧を指定するには、**Import-Package** の子をプラグインの **instructions** 要素に追加します。パッケージ一覧の構文は、**Export-Package** 要素および **Private-Package** 要素の場合と同じです。



重要

Import-Package 要素を使用する場合、必要なインポートの有無を判断するのにプラグインはバンドルの内容を自動的にスキャンしません。バンドルの内容がスキャンされるようにするには、パッケージリストの最後のエントリーとして * を配置する必要があります。

例5.6「バンドルでインポートされたパッケージの指定」 は、バンドルでインポートされたパッケージを指定する設定です。

例5.6 バンドルでインポートされたパッケージの指定

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Import-Package>javax.jws, javax.wsdl, org.apache.cxf.bus, org.apache.cxf.bus.spring,
org.apache.cxf.bus.resource, org.apache.cxf.configuration.spring, org.apache.cxf.resource,
org.springframework.beans.factory.config, * </Import-Package>
     ...
   </instructions>
  </configuration>
</plugin>
```

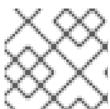
補足情報

バンドルプラグインの設定の詳細には、以下を参照してください。

- olink:OsgiDependencies/OsgiDependencies
- Apache Felix のドキュメント
- Peter Kriens の aQute Software Consultancy Web サイト

5.3.8. OSGI configAdmin ファイルの命名規則

PID 文字列 (シンボリックネーム構文) を使用すると、OSGI 仕様でハイフンを使用できます。ただし、ハイフンは Apache **Felix.fileinstall** および **config:edit** シェルコマンドによって解釈され、マネージドサービスとマネージドサービスファクトリーを区別します。したがって、PID 文字列の他の場所ではハイフンを使用しないことをお勧めします。



注記

設定ファイル名は、PID およびファクトリー PID に関連しています。

第6章 ホットデプロイメントと手動デプロイメント

概要

Fuse は、ファイルのデプロイとして、ホットデプロイメントまたは手動デプロイメントの 2 つの異なるアプローチを提供します。関連するバンドルのコレクションをデプロイする必要がある場合は、単独ではなく、**機能** として一緒にデプロイすることをお勧めします (*9章機能のデプロイ* 参照)。

6.1. ホットデプロイメント

6.1.1. ホットデプロイディレクトリー

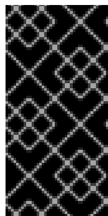
Fuse は **FUSE_HOME/deploy** ディレクトリーのファイルを監視し、このディレクトリーにすべてをホットデプロイします。ファイルがこのディレクトリーにコピーされるたびに、ファイルはランタイムにインストールされて開始されます。その後、**FUSE_HOME/deploy** ディレクトリーのファイルを更新または削除でき、変更は自動的に処理されます。

たとえば、バンドル ProjectDir/**target/foo-1.0-SNAPSHOT.jar** をビルドしたばかりの場合、以下のように FUSE_HOME/**deploy** ディレクトリーにコピーして、このバンドルをデプロイできます (UNIX プラットフォームで作業していると仮定します)。

```
% cp ProjectDir/target/foo-1.0-SNAPSHOT.jar FUSE_HOME/deploy
```

6.2. バンドルのホットアンデプロイ

ホットデプロイディレクトリーからバンドルをアンデプロイするには、**Apache Karaf コンテナーの実行中に**、バンドルファイルを **FUSE_HOME/deploy** ディレクトリーから削除するだけです。



重要

コンテナーのシャットダウン中は、ホットアンデプロイメカニズムは**機能しません。**Karaf コンテナーをシャットダウンしたら、**FUSE_HOME/deploy** ディレクトリーからバンドルファイルを削除してから、Karaf コンテナーを再起動すると、バンドルはコンテナーの再起動後にアンデプロイ**されません。**

bundle:uninstall コンソールコマンドを使用してバンドルをアンデプロイすることもできます。

6.3. 手動デプロイメント

6.3.1. 概要

Fuse コンソールでコマンドを発行して、バンドルを手動でデプロイおよびアンデプロイできます。

6.3.2. バンドルのインストール

bundle:install コマンドを使用して、OSGi コンテナーに 1 つ以上のバンドルをインストールします。このコマンドには、以下の構文があります。

```
bundle:install [-s] [--start] [--help] UrlList
```

ここで、**UrlList** は、デプロイする各バンドルの場所を指定する、空白で区切られた URL のリストです。次のコマンド引数がサポートされています。

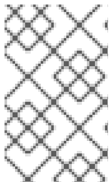
-s
: インストール後にバンドルを開始します。

--start
: **-s** と同じです。

--help
: コマンド構文を表示して説明します。

たとえば、バンドル **ProjectDir/target/foo-1.0-SNAPSHOT.jar** をインストールして起動するには、Karaf コンソールプロンプトで以下のコマンドを入力します。

```
bundle:install -s file:ProjectDir/target/foo-1.0-SNAPSHOT.jar
```



注記

Windows プラットフォームでは、このコマンドの **file** URL に正しい構文を使用するように注意する必要があります。詳しくは、「ファイル URL ハンドラー」 を参照してください。

6.3.3. バンドルのアンインストール

バンドルをアンインストールするには、まず **bundle:list** コマンドを使用してそのバンドル ID を取得する必要があります。次に、**bundle:uninstall** コマンドを使用してバンドルをアンインストールできます (バンドル ID を引数として取ります)。

たとえば、**A Camel OSGi Service Unit** という名前のバンドルをすでにインストールしている場合は、コンソールプロンプトで **bundle:list** を入力すると以下のように出力が生成される可能性があります。

```
...
[ 181] [Resolved   ] [            ] [       ] [   60] A Camel OSGi Service Unit (1.0.0.SNAPSHOT)
```

以下のコンソールコマンドを入力して、ID **181** でバンドルをアンインストールできるようになります。

```
bundle:uninstall 181
```

6.3.4. バンドルの位置を特定する URL スキーム

bundle:install コマンドにロケーション URL を指定する場合、Fuse によってサポートされる URL スキームを使用できます。これには、以下のスキーム型が含まれます。

- 「ファイル URL ハンドラー」
- 「HTTP URL ハンドラー」.
- 「Mvn URL ハンドラー」

6.4. BUNDLE:WATCH を使用したバンドルの自動再デプロイ

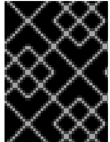
開発者がバンドルを絶えず変更および再構築している開発環境では、通常、バンドルを複数回再インストールする必要があります。**bundle:watch** コマンドを使用すると、ローカルの Maven リポジトリーを

監視し、ローカルの Maven リポジトリーで変更されると、すぐに自動的に特定のバンドルを再インストールするように Karaf に指示することができます。

たとえば、バンドル ID **751** を持つ特定のバンドルの場合、次のコマンドを入力して、自動最デプロイメントを有効にできます。

```
bundle:watch 751
```

これで、Maven アーティファクトをローカルの Maven リポジトリーに再ビルドし、インストールするたびに (Maven プロジェクトで **mvn install** を実行するなど)、Karaf コンテナーは変更した Maven アーティファクトを自動的に再インストールします。詳細は、Apache Karaf コンソールリファレンス を参照してください。



重要

bundle:watch コマンドは、開発環境のみでの使用を目的としています。したがって、実稼働環境での使用は**推奨されません**。

第7章 ライフサイクル管理

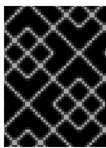
7.1. バンドルのライフサイクル状態

OSGi 環境のアプリケーションは、そのバンドルのライフサイクルの影響を受けます。バンドルには 6 つのライフサイクル状態があります。

1. **インストール済み**: すべてのバンドルはインストール済みの状態で起動します。インストール済み状態のバンドルは、すべての依存関係が解決されるのを待機し、解決されると、バンドルは解決済み状態に移行します。

2. **解決済み**: 次の条件が満たされると、バンドルは解決済み状態に移行します。

   - ランタイム環境は、バンドルで指定された環境を満たすか、それを上回ります。
   - バンドルでインポートされたパッケージはすべて、解決済みの状態にあるか、現在のバンドルと同時に解決済みの状態に移行できるバンドルにより公開されます。
   - 必要なバンドルはすべて解決済みの状態であるか、現在のバンドルと同時に解決できます。

   重要

   アプリケーションを開始する前に、アプリケーションのすべてのバンドルが解決済みの状態になっている必要があります。

   上記の条件のいずれかが満たされない場合には、バンドルはインストール済み状態に戻されます。たとえば、これは、インポートされたパッケージを含むバンドルがコンテナーから削除された場合に発生する可能性があります。

3. **開始**: 開始状態は、解決された状態とアクティブな状態の間の一時的な状態です。バンドルが開始されると、コンテナーはバンドルのリソースを作成する必要があります。また、提供されている場合、コンテナーはバンドルのバンドルアクティベーターの **start()** メソッドも呼び出します。

4. **アクティブ**: アクティブ状態のバンドルは作業に使用できます。アクティブ状態ででのバンドルの動作は、バンドルの内容によって異なります。たとえば、JAX-WS サービスプロバイダーを含むバンドルは、サービスが要求を受け入れる準備ができていることを指します。

5. **停止**: 停止状態は、アクティブ状態と解決済み状態の間の一時的な状態です。バンドルが停止すると、コンテナーはバンドルのリソースを消去する必要があります。また、提供されている場合、コンテナーはバンドルのバンドルアクティベーターの **stop()** メソッドも呼び出します。

6. **アンインストール済み**: バンドルがアンインストールされると、解決済みの状態からアンインストール済みの状態に移行します。この状態のバンドルは、解決済みの状態またはその他の状態に戻すことはできません。明示的に再インストールする必要があります。

アプリケーション開発者にとって最も重要なライフサイクル状態は、開始状態と停止状態です。アプリケーションによって公開されるエンドポイントは、開始状態の間に公開されます。公開されたエンドポイントは、停止状態の間に停止されます。

7.2. バンドルのインストールと解決

bundle:install コマンドを使用してバンドルをインストールする場合 (**-s** フラグなしで) 、カーネルは指

定されたバンドルをインストールし、これを解決済み状態にしようとします。何らかの理由でバンドルの解決に失敗した場合 (たとえば、依存関係の 1 つが満たされていない場合) に、カーネルはバンドルをインストール状態のままにします。

後で (たとえば、不足している依存関係のインストール後など)、以下のように **bundle:resolve** コマンドを実行し、バンドルの解決済み状態への移行を試みることができます。

```
bundle:resolve 181
```

ここでの引数 (この例では **181**) は、解決するバンドルの ID です。

7.3. バンドルの開始と停止

bundle:start コマンドを使用して、(インストール済み状態または解決済み状態から) 1 つ以上のバンドルを開始できます。たとえば、ID が 181、185、および 186 のバンドルを開始するには、次のコンソールコマンドを入力します。

```
bundle:start 181 185 186
```

bundle:stop コマンドを使用して、1 つ以上のバンドルを停止できます。たとえば、ID が 181、185、および 186 のバンドルを停止するには、次のコンソールコマンドを入力します。

```
bundle:stop 181 185 186
```

bundle:restart コマンドを使用して、1 つ以上のバンドルを再起動できます (つまり、開始状態から解決済み状態に移行してから、開始状態に戻ります) 。たとえば、ID が 181、185、および 186 のバンドルを再起動するには、次のコンソールコマンドを入力します。

```
bundle:restart 181 185 186
```

7.4. バンドルの開始レベル

開始レベル はすべてのバンドルに関連付けられています。開始レベルは、バンドルのアクティブ化/開始順序を制御する正の整数値です。開始レベルが低いバンドルは、開始レベルが高いバンドルよりも前に開始されます。したがって、開始レベル **1** のバンドルは最初に開始され、カーネルに属するバンドルの開始レベルは低い傾向にあります。これらのバンドルはほとんどのバンドルを実行する前提条件を提供するためです。

通常、ユーザーバンドルの開始レベルは 60 以上です。

7.5. バンドルの開始レベルの指定

bundle:start-level コマンドを使用して、特定のバンドルの開始レベルを設定します。たとえば、ID が **181** のバンドルに開始レベル **70** を設定するには、次のコンソールコマンドを入力します。

```
bundle:start-level 181 70
```

7.6. システム開始レベル

OSGi コンテナー自体には開始レベルが関連付けられており、この **システム開始レベル** によって、アクティブにできるバンドルとアクティブにできないバンドルが決まります。アクティブにできるのは、開始レベルがシステム開始レベル **以下** のバンドルのみです。

現在のシステム開始レベルを検出するには、以下のようにコンソールで **system:start-level** を入力します。

```
karaf@root()> system:start-level
Level 100
```

システム開始レベルを変更する場合は、以下のように、新しい開始レベルを **system:start-level** コマンドに引数として指定します。

```
system:start-level 200
```

第8章 依存関係のトラブルシューティング

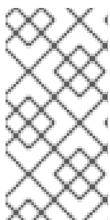
8.1. 依存関係が欠落している

Red Hat Fuse コンテナーへの OSGi バンドルのデプロイ時に発生する可能性のある最も一般的な問題は、依存関係が 1 つ以上欠落していることです。この問題は、OSGi コンテナーでバンドルを解決しようとすると、通常はバンドルを開始した場合の副次的な効果として現れます。バンドルが解決 (または起動) に失敗し、**ClassNotFound** エラーがログに記録されます (ログを表示するには、**log:display** コンソールコマンドを使用するか、**FUSE_HOME/data/log** ディレクトリーのログファイルを確認します)。

依存関係が不足する基本的な原因は 2 つあります。必要な機能またはバンドルがコンテナーにインストールされていない場合と、バンドルの **Import-Package** ヘッダーである場合です。

8.2. 必要な機能またはバンドルがインストールされていない

バンドルを解決する前に、バンドルに必要なすべての機能とバンドルが OSGi コンテナーに **インストール済み** である必要があります。特に、Apache Camel はモジュラーアーキテクチャーを採用しており、各コンポーネントが個別の機能としてインストールされるため、必要なコンポーネントのいずれかをインストールすることを忘れることはよくあります。



注記

バンドルを機能としてパッケージ化することを検討してください。機能を使用すると、バンドルをそのすべての依存関係と一緒にパッケージ化して、それらがすべて同時にインストールされるようにすることができます。詳細は、9章*機能のデプロイ* を参照してください。

8.3. IMPORT-PACKAGE ヘッダーが不完全

すべての必要な機能とバンドルがすでにインストールされ、引き続き **ClassNotFound** エラーが発生する場合、バンドルの **MANIFEST.MF** ファイルの **Import-Package** ヘッダーが不完全であることを意味します。**maven-bundle-plugin** (「既存の Maven プロジェクトの変更」 を参照) は、バンドルの **Import-Package** ヘッダーを生成するときには非常に便利ですが、以下の点に注意してください。

- Maven バンドルプラグイン設定の **Import-Package** 要素に、ワイルドカード * が必ず含まれるようにします。ワイルドカードは、プラグインに Java ソースコードをスキャンするように指示し、パッケージの依存関係のリストを自動的に生成します。
- Maven バンドルプラグインは動的な依存関係を把握 **できません**。たとえば、Java コードが明示的にクラ出力ダーを呼び出してクラスを動的にロードする場合、バンドルプラグインはこれを考慮せず、必要な Java パッケージは生成された **Import-Package** ヘッダーにリストされません。
- Blueprint XML ファイル (例: **OSGI-INF/blueprint** ディレクトリー) を定義する場合、Blueprint XML ファイルからの依存関係はすべて、**ランタイム時に自動的に解決されます**。

8.4. 欠落している依存関係を追跡する方法

欠落している依存関係を追跡するには、次の手順を実行します。

1. **bundle:diag** コンソールコマンドを使用します。このコマンドでは、バンドルがアクティブでない理由に関する情報が得られます。使用法については、Apache Karaf コンソールリファレンス を参照してください。

2. クイックチェックを実行して、必要なすべてのバンドルと機能が実際に OSGi コンテナーにインストールされていることを確認します。**bundle:list** を使用してインストールされたバンドルをチェックでき、**features:list** を使用してどの機能がインストールされているかをチェックできます。

3. **bundle:install** コンソールコマンドを使用して、バンドルをインストールします (ただし、開始しません)。以下に例を示します。

```
karaf@root()> bundle:install MyBundleURL
```

4. **bundle:dynamic-import** コンソールコマンドを使用して、インストールしたばかりのバンドルで動的インポートを有効にします。たとえば、バンドルのバンドル ID が 218 の場合に、次のコマンドを入力して、このバンドルの動的インポートを有効にします。

```
karaf@root()> bundle:dynamic-import 218
```

この設定により、コンテナーにすでにインストールされているバンドルの**いずれか**を使用して依存関係を解決し、通常の依存関係解決メカニズム (**Import-Package** ヘッダーに基づく) を効果的に迂回できます。これはバージョンチェックを飛ばすので、通常のデプロイメントには**お勧めしません**。間違ったバージョンのパッケージを取得しやすく、アプリケーションが誤動作する可能性があります。

5. これで、バンドルを解決できるはずです。たとえば、バンドル ID が 218 の場合に、次のコンソールコマンドを入力します。

```
karaf@root()> bundle:resolve 218
```

6. バンドルが解決されたと仮定した場合 (**bundle:list** でバンドルのステータスを確認)、**package:imports** コマンドを使用して、バンドルに結び付けた全パッケージの完全なリストを取得することができます。たとえば、バンドル ID が 218 の場合に、次のコンソールコマンドを入力します。

```
karaf@root()> package:imports -b 218
```

コンソールウィンドウに依存パッケージのリストが表示されます。

```
Package                              │ Version       │ Optional   │ ID  │ Bundle Name
─────────────────────────────────────┼───────────────┼────────────┼─────┼──────────────────────────────
org.apache.jasper.servlet            │ [2.2.0,3.0.0) │ resolved   │ 217 │ org.ops4j.pax.web.pax-web-runtime
org.jasypt.encryption.pbe            │               │ resolved   │ 217 │ org.ops4j.pax.web.pax-web-runtime
org.ops4j.pax.web.jsp                │ [7.0.0,)      │ resolved   │ 217 │ org.ops4j.pax.web.pax-web-runtime
org.ops4j.pax.web.service.spi.model  │ [7.0.0,)      │            │ 217 │ org.ops4j.pax.web.pax-web-runtime
org.ops4j.pax.web.service.spi.util   │ [7.0.0,)      │            │ 217 │ org.ops4j.pax.web.pax-web-runtime
...
```

7. バンドル JAR ファイルを展開し、**META-INF/MANIFEST.MF** ファイルの **Import-Package** ヘッダーの下に表示されているパッケージを確認します。このリストを前のステップで見つかったパッケージのリストと比較します。ここで、マニフェストの **Import-Package** ヘッダーで欠落しているパッケージの一覧をコンパイルし、これらのパッケージ名をプロジェクトの POM ファイルの Maven バンドルプラグイン設定の **Import-Package** 要素に追加します。

8. 動的インポートオプションをキャンセルするには、OSGi コンテナーから古いバンドルをアンインストールする必要があります。たとえば、バンドル ID が 218 の場合には、次のコマンドを入力します。

```
karaf@root()> bundle:uninstall 218
```

9. これで、インポートされたパッケージの更新されたリストを使用してバンドルを再構築し、OSGi コンテナーでテストできます。

addurl :experimental: :toc: :toclevels:4 :numbered:

第9章 機能のデプロイ

概要

アプリケーションやその他のツールは通常、複数の OSGi バンドルで設定されているため、相互依存または関連するバンドルをより大きなデプロイメントの単位に集約すると便利です。したがって、Red Hat Fuse には、スケーラブルなデプロイメントの単位である **機能** があり、複数のバンドル (およびオプションで他の機能への依存関係) を 1 回のステップでデプロイできます。

9.1. 機能の作成

9.1.1. 概要

基本的に、 **eatures リポジトリー** として知られる特別な種類の XML ファイルに新しい **feature** 要素を追加して機能が作成されます。機能を作成するには、次の手順を実行します。

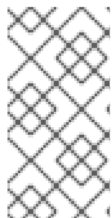
1. 「カスタム機能リポジトリーの作成」
2. 「カスタム機能リポジトリーへの機能追加」.
3. 「ローカルリポジトリー URL の機能サービスへの追加」.
4. 「機能への依存機能の追加」
5. 「OSGi 設定を機能追加」

9.2. カスタム機能リポジトリーの作成

カスタム機能リポジトリーをまだ定義していない場合は、次のように作成できます。ファイルシステム上で機能ポジトリーに便利な場所を選択します。たとえば、**C:\Projects\features.xml** などです。テキストエディターを使用して、以下の行を追加します。

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="CustomRepository">
</features>
```

ここでは、**name** 属性を設定して、リポジトリーの名前 **CustomRepository** を指定する必要があります。



注記

Maven リポジトリーまたは OBR とは対照的に、機能リポジトリーにはバンドルの保管場所は**ありません**。機能リポジトリーは、バンドルへの参照の集合を格納するだけです。バンドル自体は他の場所 (例: ファイルシステムや Maven リポジトリー) に保存されます。

9.3. カスタム機能リポジトリーへの機能追加

カスタム features リポジトリーに機能を追加するには、新しい **feature** 要素をルート **features** 要素の子として挿入します。機能に名前を付ける必要があります。**bundle** 子要素を挿入することで、機能に属するバンドルをいくつでもリストできます。たとえば、単一のバンドル **C:\Projects\camel-bundle\target\camel-bundle-1.0-SNAPSHOT.jar** を含む **example-camel-bundle** という名前の機能を追加するには、次のように **feature** 要素を追加します。

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="MyFeaturesRepo">
  <feature name="example-camel-bundle">
    <bundle>file:C:/Projects/camel-bundle/target/camel-bundle-1.0-SNAPSHOT.jar</bundle>
  </feature>
</features>
```

bundle 要素の内容は有効な URL で、バンドルの場所を示す任意の有効な URL にすることができます (15章*URL ハンドラー* を参照) 。必要に応じて、機能要素で **version** 属性を指定して、機能にゼロ以外のバージョンを割り当てることができます (これにより、バージョンを **features:install** コマンドに任意の引数として指定できます)。

機能サービスが新しい機能エントリーを正しく解析するかどうかを確認するには、以下のコンソールコマンドのペアを入力します。

```
JBossFuse:karaf@root> features:refreshurl
JBossFuse:karaf@root> features:list
...
[uninstalled] [0.0.0                 ] example-camel-bundle               MyFeaturesRepo
...
```

features:list コマンドは、機能のかなり長いリストを生成しますが、リストをスクロールして、新しい機能のエントリーを見つけることができるはずです (ここでは **example-camel-bundle**)。**features:refreshurl** コマンドは、カーネルにすべての features リポジトリーの再読み取りを強制します。このコマンドを実行しないと、カーネルは、リポジトリーに最近加えた変更を認識しません (特に、新しい機能はリストに表示されません)。

機能の長いリストでスクロールしないようにするには、以下のように **example-camel-bundle** 機能の **grep** を使用します。

```
JBossFuse:karaf@root> features:list | grep example-camel-bundle
[uninstalled] [0.0.0                 ] example-camel-bundle               MyFeaturesRepo
```

grep コマンド (標準の UNIX パターン一致ユーティリティー) がシェルに組み込まれるため、このコマンドは Windows プラットフォームでも機能します。

9.4. ローカルリポジトリー URL の機能サービスへの追加

新しい features リポジトリーを Apache Karaf で利用可能にするには、**features:addurl** コンソールを使用して features リポジトリーを追加する必要があります。たとえば、**C:\Projects\features.xml** リポジトリーの内容をカーネルで利用できるようにするには、次のコンソールコマンドを入力します。

```
features:addurl file:C:/Projects/features.xml
```

features:addurl への引数は、サポートされる URL 形式を使用して指定できます (15章*URL ハンドラー* を参照)。

以下のように **features:listUrl** コンソールコマンドを入力して、登録されたすべての features リポジトリーの URL の完全なリストを取得して、リポジトリーの URL が正常に登録されていることを確認します。

```
JBossFuse:karaf@root> features:listUrl
file:C:/Projects/features.xml
mvn:org.apache.ode/ode-jbi-karaf/1.3.3-fuse-01-00/xml/features
```

```
mvn:org.apache.felix.karaf/apache-felix-karaf/1.2.0-fuse-01-00/xml/features
```

9.5. 機能への依存機能の追加

この機能が他の機能に依存している場合は、**feature** 要素を元の **feature** 要素の子として追加することで、これらの依存関係を指定できます。各 **feature** 子要素には、現在の機能が依存する機能の名前が含まれます。依存機能などの機能をデプロイすると、依存メカニズムは、依存機能がコンテナーにインストールされているかどうかを確認します。そうでない場合には、依存関係メカニズムは、欠落している依存関係 (および再帰的な依存関係) を自動的にインストールします。

たとえば、カスタムの Apache Camel 機能 **example-camel-bundle** では、依存する標準の Apache Camel 機能を明示的に指定できます。これには、OSGi コンテナーに必要な機能が事前にデプロイされていない場合でも、アプリケーションを正常にデプロイして実行できるという利点があります。たとえば、以下のように Apache Camel 依存関係で **example-camel-bundle** 機能を定義できます。

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="MyFeaturesRepo">
  <feature name="example-camel-bundle">
    <bundle>file:C:/Projects/camel-bundle/target/camel-bundle-1.0-SNAPSHOT.jar</bundle>
    <feature version="7.8.0.fuse-780038-redhat-00001">camel-core</feature>
    <feature version="7.8.0.fuse-780038-redhat-00001">camel-spring-osgi</feature>
  </feature>
</features>
```

version 属性の指定は任意です。存在する場合、特定のバージョンの機能を選択できます。

9.6. OSGI 設定を機能追加

アプリケーションが **OSGi Configuration Admin** サービスを使用する場合は、機能定義の **config** 子要素を使用して、このサービスの設定を指定できます。たとえば、**prefix** プロパティーの値が **MyTransform** であることを指定するには、以下の **config** 子要素を機能の設定に追加します。

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="MyFeaturesRepo">
  <feature name="example-camel-bundle">
    <config name="org.fusesource.fuseesb.example">
      prefix=MyTransform
    </config>
  </feature>
</features>
```

config 要素の **name** 属性は、プロパティー設定の**永続 ID** を指定します (永続 ID は実質的にプロパティー名の名前スコープとして機能します)。**config** 要素の内容は、Java プロパティーファイル と同じ方法で解析されます。

config 要素の設定は、以下のように、永続 ID にしたがって命名される **InstallDir/etc** ディレクトリーにある Java プロパティーファイルの設定によって、オプションで上書きできます。

```
InstallDir/etc/org.fusesource.fuseesb.example.cfg
```

前述の設定プロパティーを実際に使用する方法の例として、OSGi 設定プロパティーにアクセスする次の BlueprintXML ファイルについて考えてみます。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0">

    <!-- osgi blueprint property placeholder -->
    <cm:property-placeholder id="placeholder"
                             persistent-id="org.fusesource.fuseesb.example">
        <cm:default-properties>
            <cm:property name="prefix" value="DefaultValue"/>
        </cm:default-properties>
    </cm:property-placeholder>

    <bean id="myTransform" class="org.fusesource.fuseesb.example.MyTransform">
      <property name="prefix" value="${prefix}"/>
    </bean>

</blueprint>
```

この Blueprint XML ファイルが **example-camel-bundle** バンドルにデプロイされると、プロパティー参照 **${prefix}** は、 features リポジトリーの **config** 要素で指定される値 **MyTransform** に置き換えられます。

9.7. OSGI 設定の自動デプロイ

configfile 要素を機能に追加すると、この機能がインストールされると同時に OSGi 設定ファイルが **InstallDir/etc** ディレクトリーに追加されるようにすることができます。つまり、機能とそれに関連する設定を同時にインストールでき便利です。

たとえば、**org.fusesource.fuseesb.example.cfg** 設定ファイルが **mvn:org.fusesource.fuseesb.example/configadmin/1.0/cfg** の Maven リポジトリーでアーカイブされている場合、以下の要素を機能に追加するとこの設定ファイルをデプロイできます。

```
<configfile finalname="etc/org.fusesource.fuseesb.example.cfg">
  mvn:org.fusesource.fuseesb.example/configadmin/1.0/cfg
</configfile>
```

第10章 機能のデプロイメント

10.1. 概要

次のいずれかの方法で機能をデプロイできます。

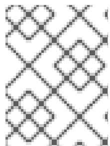
- **features:install** を使用して、コンソールにインストールします。
- ホットデプロイメントを使用します。
- ブート設定を変更します (初回起動のみ)。

10.2. コンソールでのインストール

(features リポジトリーにエントリーを追加し、 features リポジトリーを登録することで) 機能を作成した後、**features:install** コンソールを使用して機能を比較的簡単にデプロイできます。たとえば、**example-camel-bundle** 機能をデプロイするには、以下のコンソールコマンドのペアを入力します。

```
JBossFuse:karaf@root> features:refreshurl
JBossFuse:karaf@root> features:install example-camel-bundle
```

features リポジトリーの機能に対して最近変更が加えられ、カーネルがまだ認識していない場合は、**features:install** を呼び出す前に **features:refreshurl** コマンドを呼び出すことが推奨されます。**features:install** コマンドは、機能名を引数として取ります (オプションで 2 番目の引数として機能バージョンを取ります) 。



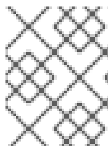
注記

機能はフラットな名前空間を使用します。したがって、機能に名前を付けるときは、既存の機能と名前が重複しないように注意してください。

10.3. コンソールでのアンインストール

機能をアンインストールするには、以下のように **features:uninstall** コマンドを実行します。

```
JBossFuse:karaf@root> features:uninstall example-camel-bundle
```



注記

アンインストール後、**features:list** を呼び出すと機能は引き続き表示されますが、そのステータスには **[uninstalled]** のフラグが付けられます。

10.4. ホットデプロイ

features リポジトリーファイルを **InstallDir/deploy** ディレクトリーにコピーするだけで、**すべて**の機能を features リポジトリーにホットデプロイできます。

機能リポジトリー全体を一度にホットデプロイすることはほとんどないため、デプロイする機能のみを参照する縮小機能リポジトリーまたは **機能記述子** を定義する方が便利な場合がよくあります。機能記述子の構文は機能リポジトリーとまったく同じですが、異なるスタイルで記述されています。違いは、機能記述子が機能リポジトリーからの既存の機能への参照だけで設定されていることです。

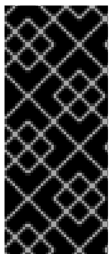
たとえば、以下のように **example-camel-bundle** 機能を読み込む機能記述子を定義できます。

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="CustomDescriptor">
  <repository>RepositoryURL</repository>
  <feature name="hot-example-camel-bundle">
    <feature>example-camel-bundle</feature>
  </feature>
</features>
```

repository 要素は、カスタム機能リポジトリーの場所である **RepositoryURL** を指定します (ここでは、15章*URL ハンドラー* で説明されている URL 形式のいずれかを使用できます)。機能 **hot-example-camel-bundle** は、既存の機能 **example-camel-bundle** への単なる参照です。

10.5. 機能ファイルのホットアンデプロイ

ホットデプロイディレクトリーから features ファイルをアンデプロイするには、**Apache Karaf コンテナーの実行中に**、features ファイルを **InstallDir/deploy** ディレクトリーから削除するだけです。



重要

コンテナーのシャットダウン中は、ホットアンデプロイメカニズムは**機能しません。** Karaf コンテナーをシャットダウンした場合は **deploy/** から機能ファイルを削除してから、Karaf コンテナーを再起動すると、コンテナーの再起動後にその機能はアンデプロ**イされません** (ただし、**features:uninstall** コンソールコマンドを使用してその機能を手作業でアンデプロイできます)。

10.6. ブート設定への機能追加

Apache Karaf のコピーをプロビジョニングして、複数のホストにデプロイする場合は、Apache Karaf の初回起動時にインストールされる機能のコレクションを決定する機能をブート設定に追加することをお勧めします。

インストールディレクトリーの設定ファイル **/etc/org.apache.karaf.features.cfg** には、以下の設定が含まれます。

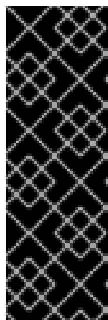
```
...
#
# Comma separated list of features repositories to register by default
#
featuresRepositories = \
    mvn:org.apache-extras.camel-extra.karaf/camel-extra/2.21.0.fuse-000032-redhat-2/xml/features, \
    mvn:org.apache.karaf.features/spring-legacy/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.apache.activemq/artemis-features/2.4.0.amq-710008-redhat-1/xml/features, \
    mvn:org.jboss.fuse.modules.patch/patch-features/7.0.0.fuse-000163-redhat-2/xml/features, \
    mvn:org.apache.karaf.features/framework/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.jboss.fuse/fuse-karaf-framework/7.0.0.fuse-000163-redhat-2/xml/features, \
    mvn:org.apache.karaf.features/standard/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.apache.karaf.features/enterprise/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.apache.camel.karaf/apache-camel/2.21.0.fuse-000055-redhat-2/xml/features, \
    mvn:org.apache.cxf.karaf/apache-cxf/3.1.11.fuse-000199-redhat-1/xml/features, \
    mvn:io.hawt/hawtio-karaf/2.0.0.fuse-000145-redhat-1/xml/features

#
```

```
# Comma separated list of features to install at startup
#
featuresBoot = \
    instance/4.2.0.fuse-000191-redhat-1, \
    cxf-commands/3.1.11.fuse-000199-redhat-1, \
    log/4.2.0.fuse-000191-redhat-1, \
    pax-cdi-weld/1.0.0, \
    camel-jms/2.21.0.fuse-000055-redhat-2, \
    ssh/4.2.0.fuse-000191-redhat-1, \
    camel-cxf/2.21.0.fuse-000055-redhat-2, \
    aries-blueprint/4.2.0.fuse-000191-redhat-1, \
    cxf/3.1.11.fuse-000199-redhat-1, \
    cxf-http-undertow/3.1.11.fuse-000199-redhat-1, \
    pax-jdbc-pool-narayana/1.2.0, \
    patch/7.0.0.fuse-000163-redhat-2, \
    cxf-rs-description-swagger2/3.1.11.fuse-000199-redhat-1, \
    feature/4.2.0.fuse-000191-redhat-1, \
    camel/2.21.0.fuse-000055-redhat-2, \
    jaas/4.2.0.fuse-000191-redhat-1, \
    camel-jaxb/2.21.0.fuse-000055-redhat-2, \
    camel-paxlogging/2.21.0.fuse-000055-redhat-2, \
    deployer/4.2.0.fuse-000191-redhat-1, \
    diagnostic/4.2.0.fuse-000191-redhat-1, \
    patch-management/7.0.0.fuse-000163-redhat-2, \
    bundle/4.2.0.fuse-000191-redhat-1, \
    kar/4.2.0.fuse-000191-redhat-1, \
    camel-csv/2.21.0.fuse-000055-redhat-2, \
    package/4.2.0.fuse-000191-redhat-1, \
    scr/4.2.0.fuse-000191-redhat-1, \
    maven/4.2.0.fuse-000191-redhat-1, \
    war/4.2.0.fuse-000191-redhat-1, \
    camel-mail/2.21.0.fuse-000055-redhat-2, \
    fuse-credential-store/7.0.0.fuse-000163-redhat-2, \
    framework/4.2.0.fuse-000191-redhat-1, \
    system/4.2.0.fuse-000191-redhat-1, \
    pax-http-undertow/6.1.2, \
    camel-jdbc/2.21.0.fuse-000055-redhat-2, \
    shell/4.2.0.fuse-000191-redhat-1, \
    management/4.2.0.fuse-000191-redhat-1, \
    service/4.2.0.fuse-000191-redhat-1, \
    camel-undertow/2.21.0.fuse-000055-redhat-2, \
    camel-blueprint/2.21.0.fuse-000055-redhat-2, \
    camel-spring/2.21.0.fuse-000055-redhat-2, \
    hawtio/2.0.0.fuse-000145-redhat-1, \
    camel-ftp/2.21.0.fuse-000055-redhat-2, \
    wrap/2.5.4, \
    config/4.2.0.fuse-000191-redhat-1, \
    transaction-manager-narayana/5.7.2.Final
```

この設定ファイルには、次の 2 つのプロパティーがあります。

- **featuresRepositories** – 起動時に読み込む features リポジトリーのコンマ区切りリスト。
- **featuresBoot** – 起動時にインストールする機能のコンマ区切りリスト。

設定を変更して、Fuse の起動時にインストールされる機能をカスタマイズできます。機能を事前インストールした Fuse を配布する場合は、この設定ファイルを変更することもできます。



重要

この機能の追加方法は、特定の Apache Karaf インスタンスが **初めて** 起動したときにのみ有効です。その後に **featuresRepositories** 設定および **featuresBoot** 設定に加えられた変更は、コンテナーを再起動しても無視されます。

ただし、**InstallDir/data/cache** の内容全体を削除することで、コンテナーを強制的に初期状態に戻すことができます (これにより、コンテナーのカスタム設定がすべて失われます)。

第11章 プレーン JAR のデプロイ

概要

アプリケーションを Apache Karaf にデプロイする別の方法は、プレーン JAR ファイルを使用することです。これらは通常 **、デプロイメントメタデータを含まない** ライブラリーです。プレーンな JAR は、WAR でも OSGi バンドルでもありません。

プレーン JAR がバンドルの依存関係として発生する場合は、バンドルヘッダーを JAR に追加する必要があります。JAR がパブリック API を公開する場合には、通常、最善の解決策は、既存の JAR をバンドルに変換し、JAR を他のバンドルと共有できるようにすることです。この章の手順を使用して、オープンソースの Bnd ツールで変換プロセスを自動的に実行します。

Bnd ツールの詳細は、Bnd ツールの Web サイト を参照してください。

11.1. ラップスキームを使用した JAR の変換

概要

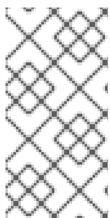
wrap: プロトコルを使用して JAR をバンドルに変換するオプションがあり、既存の URL 形式で使用できます。**wrap:** プロトコルは Bnd ユーティリティーに基づいています。

構文

wrap: プロトコルの基本的な構文は次のとおりです。

```
wrap:LocationURL
```

wrap: プロトコルは、JAR を特定する URL に接頭辞として指定できます。URL の一部を見つけるため、プレーン JAR と **wrap:** プロトコルの URL ハンドラーをの取得に **LocationURL** が使用され、JAR が自動的にバンドルに変換されます。



注記

wrap: プロトコルは、より詳細な構文もサポートします。これにより、Bnd プロパティーファイルを指定するか、URL に個別の Bnd プロパティーを指定して、変換をカスタマイズできます。ただし、通常は、**wrap:** プロトコルがデフォルト設定で使用されます。

デフォルトのプロパティー

wrap: プロトコルは **Bnd** ユーティリティーをベースとしているため、Bnd とまったく同じデフォルトプロパティーを使用してバンドルを生成します

ラップおよびインストール

以下の例は、単一のコンソールコマンドを使用して、リモート Maven リポジトリーからプレーン **commons-logging** JAR をダウンロードし、これを OSGi バンドルに動的に変換してから、OSGi コンテナーで起動する方法を示しています。

```
karaf@root> bundle:install -s wrap:mvn:commons-logging/commons-logging/1.1.1
```

参照資料

wrap: プロトコルは、Pax プロジェクト によって提供されます。これは、さまざまなオープンソースの OSGi ユーティリティー向けの包括的なプロジェクトです。**wrap:** プロトコルに関するドキュメントは、Wrap Protocol リファレンスページを参照してください。

第12章 OSGI サービス

概要

OSGi コアフレームワークは **OSGi サービスレイヤー** を定義します。これは、Java オブジェクトを **OSGi サービスレジストリー** にサービスとして登録して、バンドルによる単純な対話メカニズムを提供します。OSGi サービスモデルの長所の 1 つは、**任意** の Java オブジェクトをサービスとして提供できることです。サービスクラスに適用する必要のある特定の制約、継承ルール、またはアノテーションはありません。この章では、OSGi **Blueprint コンテナー** を使用して OSGi サービスをデプロイする方法について説明します。

12.1. BLUEPRINT コンテナー

概要

Blueprint コンテナーは、OSGi コンテナーとの対話を簡素化する依存関係注入フレームワークです。Blueprint コンテナーは、OSGi サービスレジストリーを使用するための設定ベースのアプローチをサポートします。たとえば、OSGi サービスをインポートおよびエクスポートするための標準 XML 要素を提供します。

12.1.1. Blueprint 設定

JAR ファイル内の Blueprint ファイルの場所

バンドル JAR ファイルのルートを基準にして、Blueprint 設定ファイルの標準の場所は次の相対ディレクトリーです。

```
OSGI-INF/blueprint
```

このディレクトリー下の **.xml** が付いたファイルはすべて、Blueprint 設定ファイルとして解釈されます。つまり、パターン **OSGI-INF/blueprint/*.xml** に一致するファイルがこれに該当します。

Maven プロジェクトでの Blueprint ファイルの場所

Maven プロジェクト **ProjectDir** のコンテキストでは、Blueprint 設定ファイルの標準の場所は次のディレクトリーです。

```
ProjectDir/src/main/resources/OSGI-INF/blueprint
```

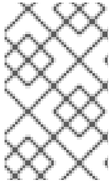
Blueprint の名前空間とルート要素

Blueprint 設定要素は、次の XML 名前空間に関連付けられています。

```
http://www.osgi.org/xmlns/blueprint/v1.0.0
```

Blueprint 設定のルート要素は **blueprint** であるため、通常、Blueprint XML 設定ファイルには以下のようなアウトライン形式になります。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  ...
</blueprint>
```



注記

blueprint ルート要素では、スキーマの場所が Blueprint フレームワークにすでに認識されているため、**xsi:schemaLocation** 属性を使用して Blueprint スキーマの場所を指定する必要はありません。

Blueprint マニフェスト設定

Blueprint 設定の一部の要素は、以下のように JAR のマニフェストファイル **META-INF/MANIFEST.MF** のヘッダーによって制御されます。

- カスタム Blueprint ファイルの場所
- 必須の依存関係

カスタム Blueprint ファイルの場所

標準以外の場所 (つまり **OSGI-INF/blueprint/*.xml** 以外の場所) に Blueprint 設定ファイルを配置する必要がある場合は、マニフェストファイル内の **Bundle-Blueprint** ヘッダーの代替場所をコンマ区切りリストで指定できます。以下に例を示します。

```
Bundle-Blueprint: lib/account.xml, security.bp, cnf/*.xml
```

必須の依存関係

OSGi サービスの依存関係はデフォルトで必須です (ただし、**reference** 要素または **reference-list** 要素の **availability** 属性を **optional** に設定すると変更できます)。依存関係を必須として宣言すると、その依存関係がないとバンドルが正しく機能できず、依存関係が常に利用可能である必要があります。

通常、Blueprint コンテナーは初期化中に **猶予期間** を経過し、その間にすべての必須の依存関係を解決しようとします。この時間内に必須の依存関係を解決できない場合 (デフォルトのタイムアウトは 5 分)、コンテナーの初期化は中止され、バンドルは開始されません。以下の設定を **Bundle-SymbolicName** マニフェストヘッダーに追加して、猶予期間を設定できます。

blueprint.graceperiod

true (デフォルト) の場合、猶予期間が有効になり、Blueprint コンテナーは初期化中に必須の依存関係が解決されるのを待ちます。**false** の場合は、猶予期間が省略され、コンテナーが必須の依存関係が解決されているかどうかを確認しません。

blueprint.timeout

猶予期間のタイムアウトをミリ秒単位で指定します。デフォルトは 300000 (5 分) です。

たとえば、10 秒の猶予期間を有効にするには、マニフェストファイルに以下の **Bundle-SymbolicName** ヘッダーを定義します。

```
Bundle-SymbolicName: org.fusesource.example.osgi-client;
 blueprint.graceperiod:=true;
 blueprint.timeout:= 10000
```

Bundle-SymbolicName ヘッダーの値はセミコロン区切りのリストです。最初の項目は実際のバンドルシンボリック名で、2 番目の項目 **blueprint.graceperiod:=true** は猶予期間を有効にし、3 番目のアイテム **blueprint.timeout:= 10000** は 10 秒のタイムアウトを指定します。

12.1.2. サービス Bean の定義

概要

Blueprint コンテナーを使用すると、**bean** 要素を使用して Java クラスをインスタンス化できます。この方法で、すべてのメインアプリケーションオブジェクトを作成できます。特に、**bean** 要素を使用して、OSGi サービスインスタンスを表す Java オブジェクトを作成できます。

Blueprint Bean 要素

Blueprint **bean** 要素は、Blueprint スキーマ namespace **http://www.osgi.org/xmlns/blueprint/v1.0.0** で定義されています。

サンプル Bean

以下の例は、Blueprint の **bean** 要素を使用して、さまざまな種類の Bean を作成する方法を示しています。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="label" class="java.lang.String">
    <argument value="LABEL_VALUE"/>
  </bean>

  <bean id="myList" class="java.util.ArrayList">
    <argument type="int" value="10"/>
  </bean>

  <bean id="account" class="org.fusesource.example.Account">
    <property name="accountName" value="john.doe"/>
    <property name="balance" value="10000"/>
  </bean>

</blueprint>
```

最後の Bean の例によって参照される **Account** クラスは以下のように定義できます。

```
package org.fusesource.example;

public class Account
{
    private String accountName;
    private int balance;

    public Account () { }

    public void setAccountName(String name) {
        this.accountName = name;
    }

    public void setBalance(int bal) {
        this.balance = bal;
    }
    ...
}
```

参考資料

Blueprint Bean の定義の詳細は、次のリファレンスを参照してください。

- Spring Dynamic Modules リファレンスガイド v2.0、Blueprint の章 。
- OSGi CompendiumServicesR4.2 仕様のセクション 121 **Blueprint コンテナー** 仕様。

12.1.3. プロパティーを使用した Blueprint の設定

概要

このセクションでは、Camel コンテキスト外のファイルに保持されているプロパティーを使用して Blueprint を設定する方法について説明します。

Blueprint Bean の設定

Blueprint Bean は、外部ファイルのプロパティーを置換できる変数を使用して設定できます。**ext** namespace を宣言し、Blueprint xml に **property placeholder** Bean を追加する必要があります。**Property-Placeholder** Bean を使用して、プロパティーファイルの場所を Blueprint に宣言します。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
    xmlns:ext="http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.2.0">

    <ext:property-placeholder>
      <ext:location>file:etc/ldap.properties</ext:location>
    </ext:property-placeholder>
    ...
    <bean ...>
        <property name="myProperty" value="${myProperty}" />
    </bean>
</blueprint>
```

property-placeholder 設定オプションの指定は http://aries.apache.org/schemas/blueprint-ext/blueprint-ext.xsd にあります。

12.2. サービスのエクスポート

概要

このセクションでは、Java オブジェクトを OSGi サービスレジストリーにエクスポートして、OSGi コンテナー内の他のバンドルからサービスとしてアクセスできるようにする方法について説明します。

単一のインターフェイスでのエクスポート

単一のインターフェイス名で OSGi サービスレジストリーにサービスをエクスポートするには、**ref** 属性を使用して関連するサービス Bean を参照する **service** 要素を定義し、**interface** 属性を使用してパブリッシュされたインターフェイスを指定します。

たとえば、例12.1「単一のインターフェイスを使用したサンプルサービスのエクスポート」 に記載されている Blueprint 設定コードを使用して、**org.fusesource.example.Account** インターフェイス名の下にある **SavingsAccountImpl** クラスのインスタンスをエクスポートできます。

例12.1 単一のインターフェイスを使用したサンプルサービスのエクスポート

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="savings" class="org.fusesource.example.SavingsAccountImpl"/>
```

```
  <service ref="savings" interface="org.fusesource.example.Account"/>

</blueprint>
```

ref 属性は、対応する Bean インスタンスの ID を指定し、 **interface** 属性は OSGi サービスレジストリーに登録されているパブリック Java インターフェイスの名前を指定します。この例で使用されているクラスとインターフェイスは、例12.2「サンプルアカウントクラスとインターフェイス」 に示します。

例12.2 サンプルアカウントクラスとインターフェイス

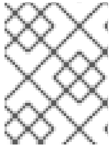
```
package org.fusesource.example

public interface Account { ... }

public interface SavingsAccount { ... }

public interface CheckingAccount { ... }

public class SavingsAccountImpl implements SavingsAccount
{
    ...
}

public class CheckingAccountImpl implements CheckingAccount
{
    ...
}
```

複数のインターフェイスを使用したエクスポート

複数のインターフェイス名の下にある OSGi サービスレジストリーにサービスをエクスポートするには、**ref** 属性を使用して関連するサービス Bean を参照する **service** 要素を定義し、**interfaces** 子要素を使用してパブリッシュされたインターフェイスを指定します。

たとえば、次の Blueprint 設定コードを使用して、パブリック Java インターフェイス **org.fusesource.example.Account** および **org.fusesource.example.SavingsAccount** のリストの下にある **SavingsAccountImpl** クラスのインスタンスをエクスポートできます。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <bean id="savings" class="org.fusesource.example.SavingsAccountImpl"/>
  <service ref="savings">
    <interfaces>
      <value>org.fusesource.example.Account</value>
      <value>org.fusesource.example.SavingsAccount</value>
    </interfaces>
  </service>
  ...
</blueprint>
```



注記

interface 属性と **interfaces** 要素は、同じ **service** 要素で同時に使用できません。どちらか一方を使用する必要があります。

自動エクスポートによるエクスポート

実装された**すべて**のパブリック Java インターフェイス下で、サービスを OSGi サービスレジストリーにエクスポートする場合には、**auto-export** 属性を使用してこれを簡単に実現する方法があります。

たとえば、実装されたすべてのパブリックインターフェイスで **SavingsAccountImpl** クラスのインスタンスをエクスポートするには、以下の Blueprint 設定コードを使用します。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <bean id="savings" class="org.fusesource.example.SavingsAccountImpl"/>
  <service ref="savings" auto-export="interfaces"/>
  ...
</blueprint>
```

auto-export 属性の値 **interfaces** は、Blueprint が **SavingsAccountImpl** によって実装されたすべてのパブリックインターフェイスを登録する必要があることを示しています。**auto-export** 属性には、次の有効な値を指定できます。

disabled

自動エクスポートを無効にします。これはデフォルトになります。

interfaces

実装されているすべてのパブリック Java インターフェイスにサービスを登録します。

class-hierarchy

Object クラスを除き、独自のタイプ (クラス) およびすべてのスーパータイプ (スーパークラス) の下のサービスを登録します。

all-classes

class-hierarchy オプションと同様ですが、実装されたすべてのパブリック Java インターフェイスも含まれます。

サービスプロパティーの設定

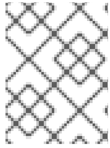
OSGi サービスレジストリーを使用すると、サービス **プロパティー** を登録済みサービスに関連付けることもできます。その後、サービスのクライアントは、サービスプロパティーを使用して、サービスを検索またはフィルターリングできます。サービスプロパティーをエクスポートしたサービスに関連付けるには、1 つ以上の **beans:entry** 要素が含まれる **service-properties** 子要素を追加します (サービスプロパティーごとに **beans:entry** 要素 1 つ)。

たとえば、**bank.name** 文字列プロパティーを普通預金口座サービスに関連付けるには、以下の Blueprint 設定を使用できます。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
        xmlns:beans="http://www.springframework.org/schema/beans"
        ...>
  ...
  <service ref="savings" auto-export="interfaces">
    <service-properties>
      <beans:entry key="bank.name" value="HighStreetBank"/>
    </service-properties>
```

```
  </service>
  ...
</blueprint>
```

bank.name 文字列プロパティーの値が **HighStreetBank** の場合。文字列以外のタイプのサービスプロパティーを定義することができ、プリミティブタイプ、配列、およびコレクションもサポートされます。これらのタイプを定義する方法は、Spring Reference Guide の アドバタイズされたプロパティーのセットの制御 を参照してください。



注記

entry 要素は Blueprint namespace に属している必要があります。Spring の Blueprint 実装での **beans:entry** 要素の使用は標準ではありません。

デフォルトのサービスプロパティー

以下のように、**service** 要素を使用してサービスをエクスポートするときに自動的に設定される可能性がある 2 つのサービスプロパティーがあります。

- **osgi.service.blueprint.compname** は、Bean がインライン (Bean が **service** 要素の子要素として定義される) の場合を除き、サービスの **bean** 要素の **id** を常に設定します。インライン化された Bean は常に匿名です。
- **service.ranking** は、ranking 属性がゼロ以外である場合に自動的に設定されます。

ランキング属性の指定

バンドルがサービスレジストリーでサービスを検索し、一致するサービスが複数見つかった場合は、ランキングを使用して、どのサービスが返されるかを判断できます。このルールは、ルックアップが複数のサービスに一致する場合は常に、最高ランクのサービスが返されるというものです。サービスランクは負ではない整数で、**0** がデフォルトです。**service** 要素に **ranking** 属性を設定して、サービスランクを指定できます。以下に例を示します。

```
<service ref="savings" interface="org.fusesource.example.Account" ranking="10"/>
```

登録リスナーの指定

サービスの登録および登録解除イベントを追跡する場合は、登録および登録解除イベント通知を受信する **登録リスナー** コールバック Bean を定義できます。登録リスナーを定義するには、**registration-listener** 子要素を **service** 要素に追加します。

たとえば、以下の Blueprint 設定は、**registration-listener** 要素によって参照されるリスナー Bean **listenerBean** を定義し、**Account** サービスが登録または登録解除されるたびにリスナー Bean がコールバックを受信するようにします。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" ...>
  ...
  <bean id="listenerBean" class="org.fusesource.example.Listener"/>

  <service ref="savings" auto-export="interfaces">
    <registration-listener
        ref="listenerBean"
        registration-method="register"
        unregistration-method="unregister"/>
```

```
  </service>
  ...
</blueprint>
```

registration-listener 要素の **ref** 属性がリスナー Bean の **id** を参照する場合は、**registration-method** 属性は登録コールバックを受信するリスナーメソッドの名前を指定し、**unregistration-method** 属性は登録解除コールバックを受信するリスナーメソッドの名前を指定します。

以下の Java コードは、登録および登録解除のイベントの通知を受信する **Listener** クラスの定義例を示しています。

```
package org.fusesource.example;

public class Listener
{
    public void register(Account service, java.util.Map serviceProperties) {
        ...
    }

    public void unregister(Account service, java.util.Map serviceProperties) {
        ...
    }
}
```

メソッド名 **register** および **unregister** は、それぞれ **registration-method** 属性および **unregistration-method** 属性によって指定されます。これらのメソッドの署名は、次の構文に準拠している必要があります。

- **最初のメソッド引数**: サービスオブジェクトのタイプから割り当て可能な任意のタイプ T。つまり、サービスクラスのスーパータイプクラス、またはサービスクラスによって実装されるインターフェイス。サービス Bean が **scope** が **prototype** であることを宣言していない限り、この引数にはサービスインスタンスが含まれます。この場合、この引数は **null** です (スコープが **prototype** の場合、登録時に利用可能なサービスインスタンスはありません)。
- **第二の方法引数**は、**java.util.Map** 型や **java.util.Dictionary** 型のいずれかである必要があります。このマップには、このサービス登録に関連付けられたサービスプロパティーが含まれています。

12.3. サービスのインポート

概要

このセクションでは、OSGi サービスレジストリーにエクスポートされた OSGi サービスへの参照を取得して使用する方法について説明します。**reference** 要素または **reference-list** 要素のいずれかを使用して、OSGi サービスをインポートできます。**reference** 要素は**ステートレス**サービスへのアクセスに適していますが、**reference-list** 要素は**ステートフル**サービスへのアクセスに適しています。

サービス参照の管理

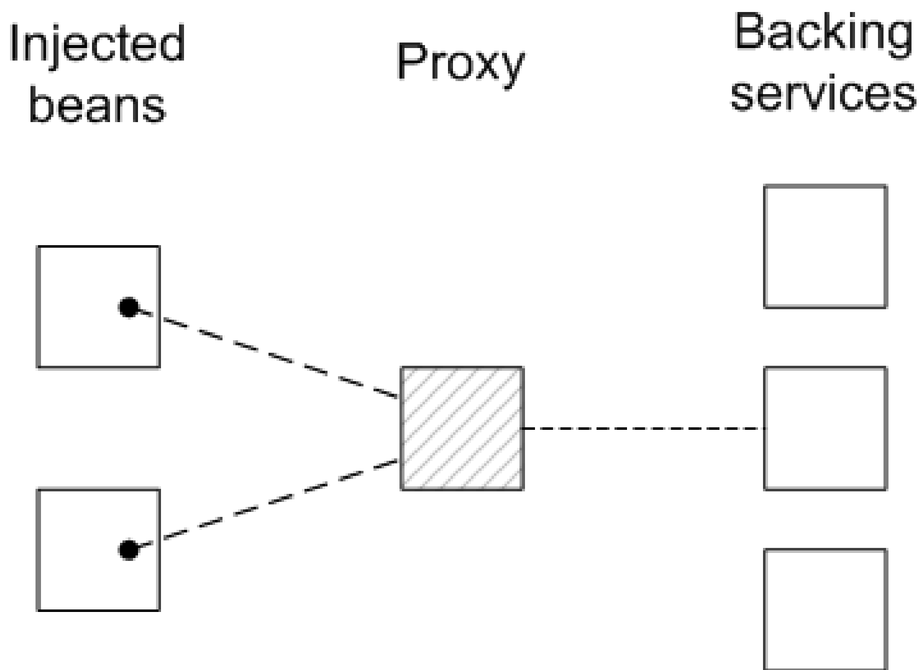
OSGi サービス参照を取得するための以下のモデルがサポートされています。

- 参照マネージャー
- 参照リストマネージャー

参照マネージャー

参照マネージャーインスタンスは、Blueprint **reference** 要素によって作成されます。この要素は、単一のサービス参照を返し、**ステートレス** サービスへのアクセスに推奨される方法です。図12.1「ステートレスサービスへの参照」 は、参照マネージャーを使用してステートレスサービスにアクセスするためのモデルの概要を示しています。

図12.1 ステートレスサービスへの参照



クライアント Blueprint コンテナー内の Bean には、OSGi サービスレジストリーからのサービスオブジェクト (**バッキングサービス**) によってサポートされるプロキシーオブジェクト (**提供されたオブジェクト**) が注入されます。このモデルは、次のように、ステートレスサービスが交換可能であるという事実を明示的に利用しています。

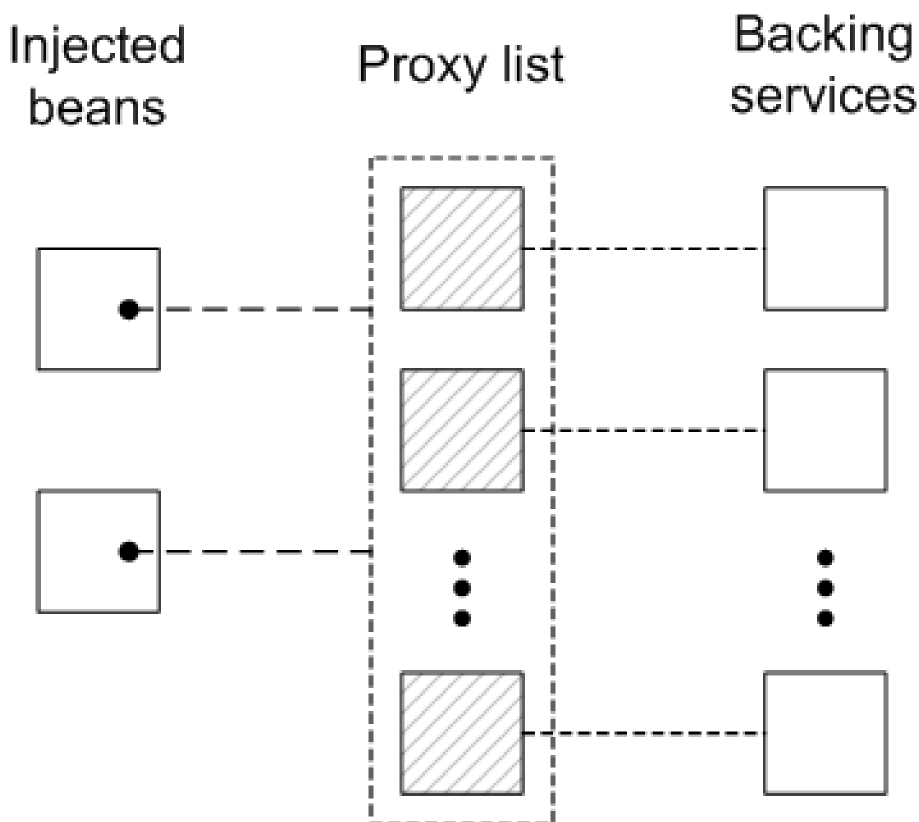
- **reference** 要素の基準と一致する複数のサービスインスタンスが見つかった場合は、参照マネージャーは、バッキングインスタンスのいずれかを任意に選択することができます (交換可能であるため) 。
- バッキングサービスがなくなった場合には、参照マネージャーは、同じタイプの他の利用可能なサービスの 1 つを使用するようにすぐに切り替えることができます。したがって、次のメソッド呼び出しまでの間、プロキシーが同じバッキングサービスに接続されたままであるという保証はありません。

そのため、クライアントとバッキングサービス間のコントラクトは **ステートレス** であり、クライアントは常に同じサービスインスタンスと通信しているとは**限りません**。一致するサービスインスタンスがない場合は、**ServiceUnavailable** 例外を出力する前にプロキシーは一定時間待機します。タイムアウトの長さは、**timeout** 要素に **reference** 属性を設定することで設定できます。

参照リストマネージャー

参照リストマネージャーインスタンスは、Blueprint **reference-list** 要素によって作成されます。この要素は、サービス参照のリストを返し、**ステートフルな** サービスにアクセスするのに推奨される方法です。図12.2「ステートフルサービスへの参照のリスト」 は、参照リストマネージャーを使用してステートフルサービスにアクセスするためのモデルの概要を示しています。

図12.2 ステートフルサービスへの参照のリスト



クライアントの Blueprint コンテナーの Bean は、プロキシーオブジェクトの一覧が含まれる **java.util.List** オブジェクト (**提供済みオブジェクト**) で注入されます。各プロキシーは、OSGi サービスレジストリー内の一意のサービスインスタンスによってサポートされています。ステートレスモデルとは異なり、ここではバッキングサービスは互換性があるとは**見なされません**。実際、リスト内の各プロキシーのライフサイクルは、対応するバッキングサービスのライフサイクルと密接に関連しています。サービスが OSGi レジストリーに登録されると、対応するプロキシーが同時に作成され、プロキシーリストに追加されます。また、サービスが OSGi レジストリーから登録解除されると、対応するプロキシーがプロキシーリストから同時に削除されます。

したがって、プロキシーとそのバッキングサービス間のコントラクトは **ステートフル** であり、クライアントは、特定のプロキシーでメソッドを呼び出すときに、常に **同じ** バッキングサービスと通信していると想定されます。ただし、バッキングサービスが利用できなくなる可能性があり、その場合、プロキシーは古くなります。古いプロキシーでメソッドを呼び出すと、**ServiceUnavailable** 例外が生成されます。

インターフェイスによるマッチング (ステートレス)

ステートレスサービス参照を取得する最も簡単な方法は、**reference** 要素の **interface** 属性を仕様して一致するインターフェイスを指定することです。**interface** 属性の値がサービスのスーパータイプである場合や、属性値がサービスによって実装された Java インターフェイスである場合、サービスは一致すると見なされます (**interface** 属性は Java クラスまたは Java インターフェイスのいずれかを指定できます)。

たとえば、ステートレス **SavingsAccount** サービスを参照するには、以下のように **reference** 要素を定義します (例12.1「単一のインターフェイスを使用したサンプルサービスのエクスポート」 を参照)。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference id="savingsRef"
           interface="org.fusesource.example.SavingsAccount"/>
```

```
  <bean id="client" class="org.fusesource.example.client.Client">
    <property name="savingsAccount" ref="savingsRef"/>
  </bean>

</blueprint>
```

reference 要素によって ID **savingsRef** を持つ参照マネージャー Bean が作成されます。参照されたサービスを使用するには、以下に示すように **savingsRef** Bean をクライアントクラスのいずれかに注入します。

クライアントクラスにインジェクトされた bean プロパティーは、**SavingsAccount** から割り当て可能なあらゆるタイプにすることができます。たとえば、以下のように **Client** クラスを定義できます。

```
package org.fusesource.example.client;

import org.fusesource.example.SavingsAccount;

public class Client {
    SavingsAccount savingsAccount;

    // Bean properties
    public SavingsAccount getSavingsAccount() {
        return savingsAccount;
    }

    public void setSavingsAccount(SavingsAccount savingsAccount) {
        this.savingsAccount = savingsAccount;
    }
    ...
}
```

インターフェイスによるマッチング (ステートフル)

ステートフルサービス参照を取得する最も簡単な方法は、**reference-list** 要素の **interface** 属性を仕様して一致するインターフェイスを指定することです。その後、参照リストマネージャーはすべてのサービスのリストを取得します。**interface** 属性の値はサービスのスーパータイプか、またはサービスによって実装されてた Java インターフェイスのいずれかです (**interface** 属性は Java クラスまたは Java インターフェイスのいずれかを指定できます)。

たとえば、ステートフル **SavingsAccount** サービスを参照するには、以下のように **reference-list** 要素を定義します (例12.1「単一のインターフェイスを使用したサンプルサービスのエクスポート」 を参照)。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference-list id="savingsListRef"
                interface="org.fusesource.example.SavingsAccount"/>

  <bean id="client" class="org.fusesource.example.client.Client">
    <property name="savingsAccountList" ref="savingsListRef"/>
  </bean>

</blueprint>
```

reference-list 要素によって ID **savingsListRef** を持つ参照リストマネージャー Bean が作成されます。参照されたサービスリストを使用するには、以下に示すように **savingsListRef** Bean 参照をクライアントクラスのいずれかに注入します。

デフォルトでは、**savingsAccountList** Bean プロパティーはサービスオブジェクトのリストです (たとえば、**java.util.List<SavingsAccount>**)。クライアントクラスは次のように定義できます。

```
package org.fusesource.example.client;

import org.fusesource.example.SavingsAccount;

public class Client {
    java.util.List<SavingsAccount> accountList;

    // Bean properties
    public java.util.List<SavingsAccount> getSavingsAccountList() {
        return accountList;
    }

    public void setSavingsAccountList(
                java.util.List<SavingsAccount> accountList
    ) {
        this.accountList = accountList;
    }
    ...
}
```

インターフェイスとコンポーネント名によるマッチング

ステートレスサービスのインターフェイスとコンポーネント名 (Bean ID) の両方と一致するには、以下のように **interface** 属性と **component-name** 属性を **reference** 要素に指定します。

```
<reference id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        component-name="savings"/>
```

ステートフルサービスのインターフェイスとコンポーネント名 (Bean ID) の両方と一致するには、以下のように **interface** 属性と **component-name** 属性を **reference-list** 要素に指定します。

```
<reference-list id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        component-name="savings"/>
```

フィルターを使用したサービスプロパティーの照合

フィルターに対してサービスプロパティーを照合して、サービスを選択できます。フィルターは、**reference** 要素または **reference-list** 要素で **filter** 属性を使用して指定されます。**filter** 属性の値は **LDAP フィルター式**である必要があります。たとえば、**bank.name** サービスプロパティーが **HighStreetBank** の場合にマッチするフィルターを定義するには、以下の LDAP フィルター式を使用します。

```
(bank.name=HighStreetBank)
```

2 つのサービスプロパティーの値を一致させるには、論理 **and** で式を組み合わせる **&** 結合を使用できます。たとえば、**foo** プロパティーが **FooValue** と等しく、**bar** プロパティーが **BarValue** と等しいことを必要とするには、以下の LDAP フィルター式を使用できます。

```
(&(foo=FooValue)(bar=BarValue))
```

LDAP フィルター式の完全な構文は、**OSGi コア仕様** のセクション 3.2.7 を参照してください。

フィルターは、**interface** および **component-name** 設定と組み合わせることもできます。この設定では、指定したすべての条件が一致する必要があります。

たとえば、**SavingsAccount** タイプの**ステートレス** サービスを **HighStreetBank** と等しい **bank.name** サービスプロパティーと一致させるには、以下のように **reference** 要素を定義します。

```
<reference id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        filter="(bank.name=HighStreetBank)"/>
```

SavingsAccount タイプの**ステートフル** サービスを **HighStreetBank** と等しい **bank.name** サービスプロパティーと一致させるには、以下のように **reference-list** 要素を定義します。

```
<reference-list id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        filter="(bank.name=HighStreetBank)"/>
```

必須/任意の指定

デフォルトでは、OSGi サービスへの参照は必須であると見なされます (必須の依存関係 参照)。要素に **availability** 属性を設定して、**reference** 要素または **reference-list** 要素の依存関係動作をカスタマイズできます。

availability 属性には、可能な値が 2 つあります。

- **mandatory** (デフォルト) は、通常の Blueprint コンテナーの初期化中に依存関係を解決する**必要がある**ことを意味します。
- **optional** は、依存関係が初期化中に解決される**必要がない**ことを意味します。

以下の **reference** 要素の例では、参照が必須の依存関係であることを明示的に宣言する方法を示しています。

```
<reference id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        availability="mandatory"/>
```

参照リスナーの指定

たとえば、一部のサービス参照に **optional** が使用できることを宣言した場合など、OSGi 環境の動的な性質に対応するには、バッキングサービスがレジストリーにバインドされたときと、レジストリーからバインド解除されたときを追跡すると便利なことがよくあります。サービスバインディングおよびバインドイベントの通知を受信するには、**reference-listener** 要素を **reference** 要素または **reference-list** 要素の子として定義できます。

たとえば、以下の Blueprint 設定は、参照リスナーを、ID **savingsRef** を持つ参照マネージャーの子として定義する方法を示しています。

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference id="savingsRef"
             interface="org.fusesource.example.SavingsAccount"
             >
    <reference-listener bind-method="onBind" unbind-method="onUnbind">
      <bean class="org.fusesource.example.client.Listener"/>
    </reference-listener>
  </reference>

  <bean id="client" class="org.fusesource.example.client.Client">
    <property name="savingsAcc" ref="savingsRef"/>
  </bean>

</blueprint>
```

前述の設定は、**bind** と **unbind** イベントをリッスンし、そのコールバックとして **org.fusesource.example.client.Listener** タイプのインスタンスを登録します。イベントは、**savingsRef** 参照マネージャーのバッキングサービスがバインドまたはバインド解除されるたびに生成されます。

以下の例は、**Listener** クラスの実装例を示しています。

```
package org.fusesource.example.client;

import org.osgi.framework.ServiceReference;

public class Listener {

    public void onBind(ServiceReference ref) {
        System.out.println("Bound service: " + ref);
    }

    public void onUnbind(ServiceReference ref) {
        System.out.println("Unbound service: " + ref);
    }

}
```

メソッド名 **onBind** および **onUnbind** は、それぞれ **bind-method** 属性および **unbind-method** 属性によって指定されます。これらのコールバックメソッドは両方とも **org.osgi.framework.ServiceReference** 引数を取ります。

12.4. OSGI サービスの公開

12.4.1. 概要

このセクションでは、OSGi コンテナーで単純な OSGi サービスの生成、構築、およびデプロイ方法について説明します。このサービスは単純な **HelloWorldJava** クラスであり、OSGi 設定は Blueprint 設定ファイルを使用して定義されます。

12.4.2. 前提条件

Maven クイックスタートアーキタイプを使用してプロジェクトを生成するには、次の前提条件を満たす必要があります。

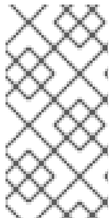
- **Maven インストール**: Maven は Apache の無料のオープンソースビルドツールです。最新バージョンは http://maven.apache.org/download.html からダウンロードできます (最小値は 2.0.9 です)。
- **インターネット接続**: ビルドの実行中、Maven は追加設定を必要とせずに動的に外部リポジトリーを検索し、必要なアーティファクトをダウンロードします。これを機能させるには、ビルドマシンがインターネットに接続されている **必要** があります。

12.4.3. Maven プロジェクトの生成

maven-archetype-quickstart アーキタイプは汎用の Maven プロジェクトを作成し、それを目的に合わせてカスタマイズできます。コーディネート **org.fusesource.example:osgi-service** を使用して Maven プロジェクトを生成するには、次のコマンドを入力します。

```
mvn archetype:create
-DarchetypeArtifactId=maven-archetype-quickstart
-DgroupId=org.fusesource.example
-DartifactId=osgi-service
```

このコマンドの結果は、生成されたプロジェクトのファイルを含むディレクトリー **ProjectDir/osgi-service** です。



注記

既存の製品のグループ ID と競合するアーティファクトのグループ ID を **選択しない** ように **注意** してください。これにより、プロジェクトのパッケージと既存の製品のパッケージが競合する可能性があります (通常、グループ ID はプロジェクトの Java パッケージ名のルートとして使用されるため)。

12.4.4. POM ファイルのカスタマイズ

OSGi バンドルを生成するには、次のように POM ファイルをカスタマイズする必要があります。

1. 「バンドルプロジェクトの生成」 で説明されている POM のカスタマイズ手順に従います。
2. Maven バンドルプラグインの設定で、以下のようにバンドルの手順を変更して **org.fusesource.example.service** パッケージをエクスポートします。

```
<project ... >
  ...
  <build>
    ...
    <plugins>
      ...
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>${pom.groupId}.${pom.artifactId}</Bundle-SymbolicName>
        <Export-Package>org.fusesource.example.service</Export-Package>
```

```
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

12.4.5. サービスインターフェイスの作成

ProjectDir/osgi-service/src/main/java/org/fusesource/example/service サブディレクトリーを作成します。このディレクトリーで、お気に入りのテキストエディターを使用してファイル **HelloWorldSvc.java** を作成し、例12.3「HelloWorldSvc インターフェイス」 からコードを追加します。

例12.3 HelloWorldSvc インターフェイス

```
package org.fusesource.example.service;

public interface HelloWorldSvc
{
    public void sayHello();
}
```

12.4.6. サービスクラスの作成

ProjectDir/osgi-service/src/main/java/org/fusesource/example/service/impl サブディレクトリーを作成します。このディレクトリーで、お気に入りのテキストエディターを使用してファイル **HelloWorldSvcImpl.java** を作成し、例12.4「HelloWorldSvcImpl クラス」 からコードを追加します。

例12.4 HelloWorldSvcImpl クラス

```
package org.fusesource.example.service.impl;

import org.fusesource.example.service.HelloWorldSvc;

public class HelloWorldSvcImpl implements HelloWorldSvc {

    public void sayHello()
    {
        System.out.println( "Hello World!" );
    }

}
```

12.4.7. Blueprint ファイルの書き込み

Blueprint 設定ファイルは、クラスパスの **OSGI-INF/blueprint** ディレクトリーに格納されている XML ファイルです。Blueprint ファイルをプロジェクトに追加するには、最初に次のサブディレクトリーを作成します。

-

```
ProjectDir/osgi-service/src/main/resources
ProjectDir/osgi-service/src/main/resources/OSGI-INF
ProjectDir/osgi-service/src/main/resources/OSGI-INF/blueprint
```

src/main/resources は、すべての JAR リソースの標準的な Maven の場所になります。このディレクトリーの下のリソースファイルは、生成されたバンドル JAR のルートスコープに自動的にパッケージ化されます。

例12.5「サービスをエクスポートするための Blueprint ファイル」 は、**bean** 要素を使用して、**HelloWorldSvc** Bean を作成し、**service** 要素を使用して Bean を OSGi サービスとしてエクスポートする Blueprint ファイルのサンプルを示しています。

ProjectDir/osgi-service/src/main/resources/OSGI-INF/blueprint ディレクトリーの下で、お気に入りのテキストエディターを使用して **config.xml** ファイルを作成し、例12.5「サービスをエクスポートするための Blueprint ファイル」から XML コードを追加します。

例12.5 サービスをエクスポートするための Blueprint ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="hello" class="org.fusesource.example.service.impl.HelloWorldSvcImpl"/>

  <service ref="hello" interface="org.fusesource.example.service.HelloWorldSvc"/>

</blueprint>
```

12.4.8. サービスバンドルの実行

osgi-service プロジェクトをインストールおよび実行するには、以下の手順を実行します。

1. **プロジェクトをビルドします** – コマンドプロンプトを開き、**ProjectDir/osgi-service** ディレクトリーに移動します。Maven を使用して、次のコマンドを入力してデモをビルドします。

   ```
   mvn install
   ```

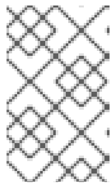
   このコマンドが正常に実行される場合、**ProjectDir/osgi-service/target** ディレクトリーには、バンドルファイル **osgi-service-1.0-SNAPSHOT.jar** が含まれている必要があります。

2. **osgi-service バンドルをインストールして開始する**: Red Hat Fuse コンソールで、次のコマンドを入力します。

   ```
   Jkaraf@root()> bundle:install -s file:ProjectDir/osgi-service/target/osgi-service-1.0-SNAPSHOT.jar
   ```

   ProjectDir は Maven プロジェクトを含むディレクトリーで、**-s** フラグはバンドルをすぐに起動するように指示します。たとえば、Windows マシンのプロジェクトディレクトリーが **C:\Projects** の場合、以下のコマンドを入力します。

   ```
   karaf@root()> bundle:install -s file:C:/Projects/osgi-service/target/osgi-service-1.0-SNAPSHOT.jar
   ```



注記

Windows マシンでは、**file** URL のフォーマット方法に注意してください。**file** URL ハンドラーが認識する構文の詳細は 「ファイル URL ハンドラー」 を参照してください。

3. **サービスが作成されたことを確認する**: バンドルが正常に開始されたことを確認するには、次の Red Hat Fuse コンソールコマンドを入力します。

```
karaf@root()> bundle:list
```

このリストのどこかに、**osgi-service** バンドルの行が表示されるはずです。以下に例を示します。

```
[ 236] [Active     ] [Created     ] [       ] [   60] osgi-service (1.0.0.SNAPSHOT)
```

12.5. OSGI サービスへのアクセス

12.5.1. 概要

このセクションでは、OSGi コンテナーで単純な OSGi クライアントの生成、構築、およびデプロイ方法について説明します。クライアントは、OSGi レジストリーで単純な Hello World サービスを見つけ、そのサービスで **sayHello()** メソッドを呼び出します。

12.5.2. 前提条件

Maven クイックスタートアーキタイプを使用してプロジェクトを生成するには、次の前提条件を満たす必要があります。

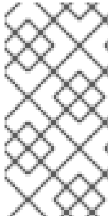
- **Maven インストール**: Maven は Apache の無料のオープンソースビルドツールです。最新バージョンは http://maven.apache.org/download.html からダウンロードできます (最小値は 2.0.9 です)。
- **インターネット接続**: ビルドの実行中、Maven は追加設定を必要とせずに動的に外部リポジトリーを検索し、必要なアーティファクトをダウンロードします。これを機能させるには、ビルドマシンがインターネットに接続されている **必要** があります。

12.5.3. Maven プロジェクトの生成

maven-archetype-quickstart アーキタイプは汎用の Maven プロジェクトを作成し、それを目的に合わせてカスタマイズできます。コーディネート **org.fusesource.example:osgi-client** を使用して Maven プロジェクトを生成するには、次のコマンドを入力します。

```
mvn archetype:create
-DarchetypeArtifactId=maven-archetype-quickstart
-DgroupId=org.fusesource.example
-DartifactId=osgi-client
```

このコマンドの結果は、生成されたプロジェクトのファイルを含むディレクトリー **ProjectDir/osgi-client** です。



注記

既存の製品のグループ ID と競合するアーティファクトのグループ ID を **選択しない** ように **注意** してください。これにより、プロジェクトのパッケージと既存の製品のパッケージが競合する可能性があります (通常、グループ ID はプロジェクトの Java パッケージ名のルートとして使用されるため)。

12.5.4. POM ファイルのカスタマイズ

OSGi バンドルを生成するには、次のように POM ファイルをカスタマイズする必要があります。

1. 「バンドルプロジェクトの生成」 で説明されている POM のカスタマイズ手順に従います。
2. クライアントは **osgi-service** バンドルで定義される **HelloWorldSvc** Java インターフェイスを使用するため、**osgi-service** バンドルに Maven 依存関係を追加する必要があります。 **osgi-service** バンドルの Maven コーディネートが **org.fusesource.example:osgi-service:1.0-SNAPSHOT** であることを想定する場合、以下の依存関係をクライアントの POM ファイルに追加する必要があります。

```
<project ... >
  ...
  <dependencies>
    ...
    <dependency>
        <groupId>org.fusesource.example</groupId>
        <artifactId>osgi-service</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
  ...
</project>
```

12.5.5. Blueprint ファイルの書き込み

Blueprint ファイルをクライアントプロジェクトに追加するには、最初に次のサブディレクトリーを作成します。

```
ProjectDir/osgi-client/src/main/resources
ProjectDir/osgi-client/src/main/resources/OSGI-INF
ProjectDir/osgi-client/src/main/resources/OSGI-INF/blueprint
```

ProjectDir/osgi-client/src/main/resources/OSGI-INF/blueprint ディレクトリーの下で、お気に入りのテキストエディターを使用して **config.xml** ファイルを作成し、例12.6「サービスをインポートするための Blueprint ファイル」から XML コードを追加します。

例12.6 サービスをインポートするための Blueprint ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference id="helloWorld"
      interface="org.fusesource.example.service.HelloWorldSvc"/>

  <bean id="client"
```

```
          class="org.fusesource.example.client.Client"
          init-method="init">
      <property name="helloWorldSvc" ref="helloWorld"/>
    </bean>

  </blueprint>
```

reference 要素は、OSGi レジストリーで **HelloWorldSvc** タイプのサービスを見つける参照マネージャーを作成します。**bean** 要素は **Client** クラスのインスタンスを作成し、サービス参照を Bean プロパティー **helloWorldSvc** として注入します。さらに、**init-method** 属性は、Bean の初期化フェーズ中 (つまり、サービス参照がクライアント Bean に注入された後) に **Client.init()** メソッドが呼び出されるように指定します。

12.5.6. クライアントクラスの作成

ProjectDir/osgi-client/src/main/java/org/fusesource/example/client ディレクトリーの下で、お気に入りのテキストエディターを使用して **Client.java** ファイルを作成し、例12.7「クライアントクラス」から Java コードを追加します。

例12.7 クライアントクラス

```
package org.fusesource.example.client;

import org.fusesource.example.service.HelloWorldSvc;

public class Client {
    HelloWorldSvc helloWorldSvc;

    // Bean properties
    public HelloWorldSvc getHelloWorldSvc() {
        return helloWorldSvc;
    }

    public void setHelloWorldSvc(HelloWorldSvc helloWorldSvc) {
        this.helloWorldSvc = helloWorldSvc;
    }

    public void init() {
        System.out.println("OSGi client started.");
        if (helloWorldSvc != null) {
            System.out.println("Calling sayHello()");
            helloWorldSvc.sayHello();  // Invoke the OSGi service!
        }
    }

}
```

Client クラスは **helloWorldSvc** Bean プロパティーの getter メソッドおよび setter メソッドを定義します。これによりインジェクションによる Hello World サービスへの参照を受け取ることができます。**init()** メソッドは、プロパティーインジェクションの後に Bean 初期化フェーズ中に呼び出されます。つまり、通常、このメソッドのスコープ内で Hello World サービスを呼び出すことができます。

12.5.7. クライアントバンドルの実行

osgi-client プロジェクトをインストールおよび実行するには、以下の手順を実行します。

1. **プロジェクトをビルドします** – コマンドプロンプトを開き、 **ProjectDir/osgi-client** ディレクトリーに移動します。Maven を使用して、次のコマンドを入力してデモをビルドします。

```
mvn install
```

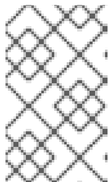
このコマンドが正常に実行される場合、**ProjectDir/osgi-client/target** ディレクトリーには、バンドルファイル **osgi-client-1.0-SNAPSHOT.jar** が含まれている必要があります。

2. **osgi-service バンドルをインストールして開始する**: Red Hat Fuse コンソールで、次のコマンドを入力します。

```
karaf@root()> bundle:install -s file:ProjectDir/osgi-client/target/osgi-client-1.0-SNAPSHOT.jar
```

ProjectDir は Maven プロジェクトを含むディレクトリーで、 **-s** フラグはバンドルをすぐに起動するように指示します。たとえば、Windows マシンのプロジェクトディレクトリーが **C:\Projects** の場合、以下のコマンドを入力します。

```
karaf@root()> bundle:install -s file:C:/Projects/osgi-client/target/osgi-client-1.0-SNAPSHOT.jar
```



注記

Windows マシンでは、**file** URL のフォーマット方法に注意してください。 **file** URL ハンドラーが認識する構文の詳細は「ファイル URL ハンドラー」 を参照してください。

3. **クライアント出力**: クライアントバンドルが正常に開始されると、コンソールに次のような出力がすぐに表示されます。

```
Bundle ID: 239
OSGi client started.
Calling sayHello()
Hello World!
```

12.6. APACHE CAMEL との統合

12.6.1. 概要

Apache Camel は、Bean 言語を使用して OSGi サービスを簡単に呼び出す方法を提供します。この機能は、Apache Camel アプリケーションが OSGi コンテナーにデプロイされ、特別な設定を必要としない場合は常に自動的に使用可能になります。

12.6.2. レジストリーチェーン

Apache Camel ルートが OSGi コンテナーにデプロイされると、**CamelContext** が Bean インスタンスの解決のためにレジストリーチェーンを自動的に設定します。レジストリーチェーンは OSGi レジストリーとそれに続く Blueprint レジストリーで設定されます。ここで、特定の Bean クラスまたは Bean イ

ンスタンスを参照しようとすると、レジストリーは Bean を次のように解決します。

1. 最初に OSGi レジストリーで Bean を検索します。クラス名が指定されている場合は、これを OSGi サービスのインターフェイスまたはクラスと同じにしてください。
2. OSGi レジストリーに一致するものが見つからない場合は、Blueprint レジストリーにフォールバックします。

12.6.3. サンプル OSGi サービスインターフェイス

単一のメソッド **getGreeting()** を定義する以下の Java インターフェイスで定義された OSGi サービスについて考えてみましょう。

```
package org.fusesource.example.hello.boston;

public interface HelloBoston {
    public String getGreeting();
}
```

12.6.4. サンプルサービスのエクスポート

HelloBoston OSGi サービスを実装するバンドルを定義するとき、以下の Blueprint 設定を使用してサービスをエクスポートできます。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="hello" class="org.fusesource.example.hello.boston.HelloBostonImpl"/>

  <service ref="hello" interface="org.fusesource.example.hello.boston.HelloBoston"/>

</blueprint>
```

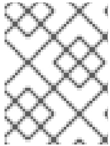
HelloBoston インターフェイスが **HelloBostonImpl** クラス (ここでは示されていない) によって実装されることを前提とします。

12.6.5. JavaDSL からの OSGi サービスの呼び出し

HelloBoston OSGi サービスが含まれるバンドルをデプロイしたら、Java DSL を使用して Apache Camel アプリケーションからサービスを呼び出すことができます。Java DSL では、次のように Bean 言語を介して OSGi サービスを呼び出します。

```
from("timer:foo?period=5000")
  .bean(org.fusesource.example.hello.boston.HelloBoston.class, "getGreeting")
  .log("The message contains: ${body}")
```

bean コマンドでは、最初の引数は OSGi インターフェイスまたはクラスで、これは OSGi サービスバンドルからエクスポートされたインターフェイスと一致する必要があります。2 番目の引数は、呼び出す Bean メソッドの名前です。**bean** コマンド構文の詳細は、Apache Camel Development Guide の Bean Integration を参照してください。



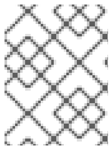
注記

このアプローチを使用すると、OSGi サービスが暗黙的にインポートされます。この場合、OSGi サービスを明示的にインポートする**必要はありません**。

12.6.6. XMLDSL からの OSGi サービスの呼び出し

XML DSL では、Bean 言語を使用して **HelloBoston** OSGi サービスを呼び出すこともできますが、構文は若干異なります。XML DSL では、以下のように **method** 要素を使用して Bean 言語を介して OSGi サービスを呼び出します。

```
<beans ...>
  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri="timer:foo?period=5000"/>
      <setBody>
          <method ref="org.fusesource.example.hello.boston.HelloBoston" method="getGreeting"/>
      </setBody>
      <log message="The message contains: ${body}"/>
    </route>
  </camelContext>
</beans>
```



注記

このアプローチを使用すると、OSGi サービスが暗黙的にインポートされます。この場合、OSGi サービスを明示的にインポートする**必要はありません**。

第13章 JMS ブローカーを使用したデプロイ

概要

Fuse 7.12 はデフォルトの内部ブローカーが含まれていませんが、4 つの外部 JMS ブローカーとインターフェイスするように設計されています。

Fuse 7.12 コンテナーには、サポートされる外部ブローカーのブローカークライアントライブラリーが含まれます。

Fuse 7.12 のメッセージングで使用できる外部ブローカー、クライアント、および Camel コンポーネントの組み合わせに関する詳細は、Red Hat Fuse でサポートされる設定 を参照してください。

13.1. AMQ7 クイックスタート

クイックスタートは、AMQ7 ブローカーを使用したアプリの設定とデプロイメントを紹介するために提示しています。

クイックスタートのダウンロード

すべてのクイックスタートは、FuseSoftwareDownloads ページからインストールできます。

ダウンロードした zip ファイルの内容をローカルフォルダーに展開します (例: **quickstarts** という名前のフォルダー)。

クイックスタートの設定

1. **quickstarts/camel/camel-jms** フォルダーに移動します。
2. **mvn clean install** と入力して、クイックスタートをビルドします。
3. ファイル **org.ops4j.connectionfactory-amq7.cfg** を **/camel/camel-jms/src/main** ディレクトリーから Fuse インストールの **FUSE_HOME/etc** ディレクトリーにコピーします。正しいブローカー URL と認証情報の内容を確認してください。デフォルトでは、ブローカー URL は AMQ 7 の CORE プロトコルに従って tcp://localhost:61616 に設定されています。認証情報は admin/admin に設定されます。外部ブローカーに合わせてこれらの詳細を変更してください。
4. Linux では **./bin/fuse** を実行し、Windows では **bin\fuse.bat** を実行して Fuse を起動します。
5. Fuse コンソールで、次のコマンドを入力します。

   ```
   feature:install pax-jms-pool artemis-jms-client camel-blueprint camel-jms

   install -s mvn:org.jboss.fuse.quickstarts/camel-jms/${project.version}
   ```

   バンドルがデプロイされると、Fuse によりバンドル ID が提供されます。

6. **log:display** を入力して、起動ログ情報を表示します。バンドルが正常にデプロイされたことを確認してください。

```
12:13:50.445 INFO [Blueprint Event Dispatcher: 1] Attempting to start Camel Context jms-example-context
12:13:50.446 INFO [Blueprint Event Dispatcher: 1] Apache Camel 2.21.0.fuse-000030
(CamelContext: jms-example-context) is starting
12:13:50.446 INFO [Blueprint Event Dispatcher: 1] JMX is enabled
```

```
12:13:50.528 INFO [Blueprint Event Dispatcher: 1] StreamCaching is not in use. If using streams then its recommended to enable stream caching. See more details at http://camel.apache.org/stream-caching.html
12:13:50.553 INFO [Blueprint Event Dispatcher: 1] Route: file-to-jms-route started and consuming from: file://work/jms/input
12:13:50.555 INFO [Blueprint Event Dispatcher: 1] Route: jms-cbr-route started and consuming from: jms://queue:incomingOrders?transacted=true
12:13:50.556 INFO [Blueprint Event Dispatcher: 1] Total 2 routes, of which 2 are started
```

クイックスタートの実行

1. Camel ルートが実行されると、**/camel/camel-jms/work/jms/input** ディレクトリーが作成されます。ファイルを **/camel/camel-jms/src/main/data** ディレクトリーから **/camel/camel-jms/work/jms/input** ディレクトリーにコピーします。

2. **…/src/main/data** ファイルは順序ファイルです。1 分待ってから、**/camel/camel-jms/work/jms/output** ディレクトリーを確認します。ファイルは、宛先の国に応じて個別のディレクトリーに分類されます。

   - **/camel/camel-jms/work/jms/output/others/** の **order1.xml**、**order2.xml** および **order4.xml**
   - **/camel/camel-jms/work/jms/output/us** の **order3.xml** および **order5.xml**
   - **/camel/camel-jms/work/jms/output/fr** の **order6.xml**

3. **log:display** を使用してログメッセージを表示します。

```
Receiving order order1.xml
Sending order order1.xml to another country
Done processing order1.xml
```

1. Camel コマンドは、コンテキストに関する詳細を表示します。

camel:context-list を使用して、コンテキストの詳細を表示します。

```
Context              Status             Total #       Failed #     Inflight #   Uptime
-------              ------             -------       --------     ----------   ------
jms-example-context   Started                12            0            0   3 minutes
```

camel:route-list を使用して、コンテキストの Camel ルートを表示します。

```
Context              Route              Status             Total #       Failed #     Inflight #   Uptime
-------              -----              ------             -------       --------     ----------   ------
jms-example-context   file-to-jms-route   Started                6            0            0   3 minutes
jms-example-context   jms-cbr-route       Started                6            0            0   3 minutes
```

camel:route-info を使用してエクスチェンジの統計を表示します。

```
karaf@root()> camel:route-info jms-cbr-route jms-example-context
Camel Route jms-cbr-route
    Camel Context: jms-example-context
    State: Started
    State: Started
```

```
Statistics
   Exchanges Total: 6
   Exchanges Completed: 6
   Exchanges Failed: 0
   Exchanges Inflight: 0
   Min Processing Time: 2 ms
   Max Processing Time: 12 ms
   Mean Processing Time: 4 ms
   Total Processing Time: 29 ms
   Last Processing Time: 4 ms
   Delta Processing Time: 1 ms
   Start Statistics Date: 2018-01-30 12:13:50
   Reset Statistics Date: 2018-01-30 12:13:50
   First Exchange Date: 2018-01-30 12:19:47
   Last Exchange Date: 2018-01-30 12:19:47
```

13.2. ARTEMIS コアクライアントの使用

Artemis コアクライアントは、**qpid-jms-client** ではなく外部ブローカーに接続するために使用できます。

Artemis コアクライアントを使用した接続

1. Artemis コアクライアントを有効にするには、Fuse を起動します。**FUSE_HOME** ディレクトリーに移動し、Linux の場合は **./bin/fuse**、Windows の場合は **bin\fuse.bat** と入力します。
2. コマンド **feature:install artemis-core-client** を使用して、Artemis クライアントを機能として追加します。
3. コードの作成時は、Camel コンポーネントを接続ファクトリーに接続する必要があります。

接続ファクトリーをインポートします。

```
import org.apache.qpid.jms.JmsConnectionFactory;
```

接続を設定します。

```
ConnectionFactory connectionFactory = new JmsConnectionFactory("amqp://localhost:5672");
     try (Connection connection = connectionFactory.createConnection()) {
```

第14章 フェイルオーバーのデプロイメント

概要

Red Hat Fuse は、単純なロックファイルシステムまたは JDBC ロックメカニズムのいずれかを使用してフェイルオーバー機能を提供します。いずれの場合も、コンテナーレベルのロックシステムによって、より高速なフェイルオーバーのパフォーマンスを提供するためにバンドルをセカンダリーカーネルインスタンスに事前ロードできます。

14.1. シンプルなロックファイルシステムの使用

概要

Red Hat Fuse を初めて起動すると、インストールディレクトリーのルートにロックファイルが作成されます。マスターインスタンスに障害が発生した場合に、ロックが同じホストマシンにあるスレーブインスタンスに渡されるようにマスター/スレーブシステムを設定できます。

ロックファイルシステムの設定

ロックファイルのフェイルオーバーデプロイメントを設定するには、プライマリーおよびセカンダリーインストール両方の **etc/system.properties** ファイルを編集し、例14.1「ファイルフェイルオーバー設定のロック」 のプロパティーが含まれるようにします。

例14.1 ファイルフェイルオーバー設定のロック

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.SimpleFileLock
karaf.lock.dir=PathToLockFileDirectory
karaf.lock.delay=10000
```

- **karaf.lock** は、ロックファイルが書き込まれるかどうかを指定します。
- **karaf.lock.class** は、ロックを実装する Java クラスを指定します。単純なファイルロックの場合は、常に **org.apache.karaf.main.SimpleFileLock** である必要があります。
- **karaf.lock.dir** は、ロックファイルが書き込まれるディレクトリーを指定します。マスターとスレーブのインストール両方で同じである**必要があります**。
- **karaf.lock.delay** は、ロックを再取得する試行間の遅延をミリ秒単位で指定します。

14.2. JDBC ロックシステムの使用

概要

JDBC ロックメカニズムは、Red Hat Fuse インスタンスが別々のマシンに存在するフェールオーバーデプロイメントを対象としています。

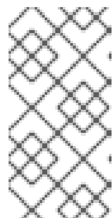
このシナリオでは、プライマリーインスタンスはデータベースでホストされるロッキングテーブルのロックを保持します。マスターでロックが失われた場合、待機しているスレーブプロセスがロックテーブルにアクセスできるようになり、そのコンテナーを完全に起動します。

JDBC ドライバーのクラスパスへの追加

JDBC ロッキングシステムでは、JDBC ドライバーは、マスター/スレーブセットアップの各インスタンスのクラスパス上にある必要があります。次のように、JDBC ドライバーをクラスパスに追加します。

1. JDBC ドライバー JAR ファイルを各 Red Hat Fuse インスタンスの **ESBInstallDir/lib/ext** ディレクトリーにコピーします。
2. **CLASSPATH** 変数に JDBC ドライバー JAR が含まれるように **bin/karaf** 起動スクリプトを変更します。
たとえば、JDBC JAR ファイル **JDBCJarFile.jar** の場合、以下のように起動スクリプトを変更できます (*NIX オペレーティングシステムの場合)。

```
...
# Add the jars in the lib dir
for file in "$KARAF_HOME"/lib/karaf*.jar
do
    if [ -z "$CLASSPATH" ]; then
        CLASSPATH="$file"
    else
        CLASSPATH="$CLASSPATH:$file"
    fi
done
CLASSPATH="$CLASSPATH:$KARAF_HOME/lib/JDBCJarFile.jar"
```



注記

MySQL ドライバー JAR または PostgreSQL ドライバー JAR を追加する場合、**karaf-** 接頭辞を付けてドライバー JAR の名前を変更する必要があります。変更しない場合には、Apache Karaf がハングし、ログに Apache Karaf がドライバーを見つけられないと表示されます。

JDBC ロックシステムの設定

JDBC ロックシステムを設定するには、以下のようにプライマリー/セカンダリーデプロイメント内の各インスタンスの **etc/system.properties** ファイルを更新します。

例14.2 JDBC ロックファイルの設定

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.DefaultJDBCLock
karaf.lock.level=50
karaf.lock.delay=10000
karaf.lock.jdbc.url=jdbc:derby://dbserver:1527/sample
karaf.lock.jdbc.driver=org.apache.derby.jdbc.ClientDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

この例では、sample という名前のデータベースが存在しない場合は作成されます。ロックテーブルを

取得する最初の Red Hat Fuse インスタンスはプライマリーインスタンスです。データベースへの接続が失われた場合、マスターインスタンスは正常なシャットダウンを試行し、データベースサービスが復元する際にスレーブインスタンスをマスターにすることができます。以前のマスターは手動で再起動する必要があります。

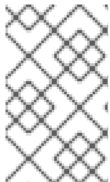
Oracle での JDBC ロックの設定

JDBC ロックシナリオでデータベースとして Oracle を使用している場合、**etc/system.properties** ファイルの **karaf.lock.class** プロパティーは **org.apache.karaf.main.lock.OracleJDBCLock** を指している必要があります。

それ以外の場合は、以下に示すように、設定に **system.properties** ファイルを通常どおりに設定します。

例14.3 Oracle 用の JDBC ロックファイル設定

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.OracleJDBCLock
karaf.lock.jdbc.url=jdbc:oracle:thin:@hostname:1521:XE
karaf.lock.jdbc.driver=oracle.jdbc.OracleDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```



注記

karaf.lock.jdbc.url には、アクティブな Oracle システム ID (SID) が必要です。つまり、この特定のロックを使用する前に、データベースインスタンスを手動で作成する必要があります。

Derby での JDBC ロックの設定

JDBC ロックシナリオでデータベースとして Derby を使用している場合、**etc/system.properties** ファイルの **karaf.lock.class** プロパティーは **org.apache.karaf.main.lock.DerbyJDBCLock** を指している必要があります。たとえば、**system.properties** ファイルを次のように設定できます。

例14.4 Derby の JDBC ロックファイル設定

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.DerbyJDBCLock
karaf.lock.jdbc.url=jdbc:derby://127.0.0.1:1527/dbname
karaf.lock.jdbc.driver=org.apache.derby.jdbc.ClientDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

MySQL での JDBC ロックの設定

JDBC ロックシナリオでデータベースとして MySQL を使用している場合、**etc/system.properties** ファイルの **karaf.lock.class** プロパティーは **org.apache.karaf.main.lock.MySQLJDBCLock** を指している必要があります。たとえば、**system.properties** ファイルを次のように設定できます。

例14.5 MySQL の JDBC ロックファイル設定

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.MySQLJDBCLock
karaf.lock.jdbc.url=jdbc:mysql://127.0.0.1:3306/dbname
karaf.lock.jdbc.driver=com.mysql.jdbc.Driver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

PostgreSQL での JDBC ロックの設定

JDBC ロックシナリオでデータベースとして PostgreSQL を使用している場合、**etc/system.properties** ファイルの **karaf.lock.class** プロパティーは **org.apache.karaf.main.lock.PostgreSQLJDBCLock** を指している必要があります。たとえば、**system.properties** ファイルを次のように設定できます。

例14.6 PostgreSQL の JDBC ロックファイル設定

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.PostgreSQLJDBCLock
karaf.lock.jdbc.url=jdbc:postgresql://127.0.0.1:5432/dbname
karaf.lock.jdbc.driver=org.postgresql.Driver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=0
```

JDBC ロッククラス

Apache Karaf では現在、次の JDBC ロッククラスが提供されています。

```
org.apache.karaf.main.lock.DefaultJDBCLock
org.apache.karaf.main.lock.DerbyJDBCLock
org.apache.karaf.main.lock.MySQLJDBCLock
org.apache.karaf.main.lock.OracleJDBCLock
org.apache.karaf.main.lock.PostgreSQLJDBCLock
```

14.3. コンテナーレベルのロック

概要

コンテナーレベルのロッキングによって、より高速なフェイルオーバーのパフォーマンスを提供するためにバンドルをセカンダリーカーネルインスタンスに事前ロードできます。コンテナーレベルのロックは、単純なファイルと JDBC の両方のロックメカニズムでサポートされています。

コンテナーレベルのロックの設定

コンテナーレベルのロッキングを実装するには、プライマリー/セカンダリー設定の各システムの **etc/system.properties** ファイルに以下を追加します。

例14.7 コンテナーレベルのロック設定

```
karaf.lock=true
karaf.lock.level=50
karaf.lock.delay=10000
```

karaf.lock.level プロパティーでは、Red Hat Fuse インスタンスが OSGi コンテナーを起動するために起動プロセスをどこまで実行するかを示します。同等かそれ以下の開始レベルが割り当てられたバンドルも、その Fuse インスタンスで開始されます。

バンドルの開始レベルは、**BundleName.jar=level** 形式で、**etc/startup.properties** に指定されます。コアシステムバンドルのレベルは 50 未満ですが、ユーザーバンドルのレベルは 50 を超えています。

表14.1 バンドル開始レベル

開始レベル	動作
1	コールドスタンバイインスタンス。コアバンドルはコンテナーにロードされません。スレーブは、ロックが取得されるまで待ち、サーバーを開始します。
<50	ホットスタンバイインスタンス。コアバンドルがコンテナーにロードされます。スレーブは、ロックが取得されるまで待ち、ユーザーレベルバンドルを開始します。コンソールは、このレベルで各スレーブインスタンスに対してアクセス可能です。
>50	ユーザーバンドルが開始されるため、この設定はお勧めしません。

ポートの競合回避

同じホストでホットスペアを使用する場合は、JMX リモートポートを一意の値に設定してバインドの競合を回避する必要があります。**fuse** 起動スクリプト (または子インスタンス上の **karaf** スクリプト) を編集して、以下が含まれるようにすることができます。

```
DEFAULT_JAVA_OPTS="-server $DEFAULT_JAVA_OPTS -
Dcom.sun.management.jmxremote.port=1100 -
Dcom.sun.management.jmxremote.authenticate=false"
```

第15章 URL ハンドラー

多くの Red Hat Fuse のコンテキストには、リソースの場所を指定する URL を提供する必要があります (例: コンソールコマンドの引数として)。一般に、URL の指定時には、Fuse の組み込み URL ハンドラーでサポートされている任意のスキームを使用できます。この付録では、使用可能なすべての URL ハンドラーの構文について説明します。

15.1. ファイル URL ハンドラー

15.1.1. 構文

ファイル URL の構文は **file:PathName** で、**PathName** は、Classpath で利用可能なファイルの相対パスまたは絶対パス名です。指定された **PathName** は、Java の組み込みファイル **URL ハンドラー** によって解析されます。したがって、**PathName** 構文は、Java パス名の通常の規則に従います。特に、Windows では、各バックスラッシュを別のバックスラッシュでエスケープするか、スラッシュに置き換える必要があります。

15.1.2. 例

たとえば、Windows のパス名 **C:\Projects\camel-bundle\target\foo-1.0-SNAPSHOT.jar** について考えてみます。次の例は、Windows でのファイル URL の **正しい** 代替案です。

```
file:C:/Projects/camel-bundle/target/foo-1.0-SNAPSHOT.jar
file:C:\\Projects\\camel-bundle\\target\\foo-1.0-SNAPSHOT.jar
```

次の例は、Windows でのファイル URL の **誤った** 代替案です。

```
file:C:\Projects\camel-bundle\target\foo-1.0-SNAPSHOT.jar      // WRONG!
file://C:/Projects/camel-bundle/target/foo-1.0-SNAPSHOT.jar    // WRONG!
file://C:\\Projects\\camel-bundle\\target\\foo-1.0-SNAPSHOT.jar // WRONG!
```

15.2. HTTP URL ハンドラー

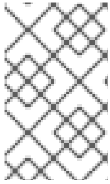
15.2.1. 構文

HTTP URL の標準構文は、**http:Host[:Port]/[Path][#AnchorName][?Query]** です。**https** スキームを使用してセキュアな HTTP URL を指定することもできます。指定の HTTP URL は、Java の組み込み HTTP URL ハンドラーによって解析されるため、HTTP URL は Java アプリケーションに対しては通常の方法で動作します。

15.3. MVN URL ハンドラー

15.3.1. 概要

Maven を使用してバンドルをビルドする場合、または特定のバンドルが Maven リポジトリーから利用可能であることがわかっている場合は、Mvn ハンドラースキームを使用してバンドルを見つけることができます。



注記

Mvn URL ハンドラーがローカルおよびリモートの Maven アーティファクトを確実に検出できるようにするには、Mvn URL ハンドラーの設定をカスタマイズする必要がある場合があります。詳細は、「Mvn URL ハンドラーの設定」 を参照してください。

15.3.2. 構文

Mvn URL の構文は次のとおりです。

```
mvn:[repositoryUrl!]groupId/artifactId[/[version][/[packaging][/[classifier]]]]
```

ここで、**repositoryUrl** は、オプションで Maven リポジトリーの URL を指定します。**groupId**、**artifactId**、**version**、**packaging**、および **classifier** は、Maven アーティファクトを見つけるための標準の Maven コーディネートです。

15.3.3. コーディネートの省略

Mvn URL を指定する場合に、必要なのは **groupId** と **artifactId** のコーディネートのみです。以下は、**groupId** の **org.fusesource.example** と、**artifactId** の **bundle-demo** で Maven バンドルを参照する例です。

```
mvn:org.fusesource.example/bundle-demo
mvn:org.fusesource.example/bundle-demo/1.1
```

最初の例のように、**バージョン**を省略すると、デフォルトは **LATEST** に設定されます。これにより、利用可能な Maven メタデータに基づいて最新バージョンに解決されます。

packaging または **version** 値を指定せずに **classifier** 値を指定するには、Mvn URL 用にスペースを空けることができます。**version** 値なしで **packaging** 値を指定する場合も同様です。以下に例を示します。

```
mvn:groupId/artifactId///classifier
mvn:groupId/artifactId/version//classifier
mvn:groupId/artifactId//packaging/classifier
mvn:groupId/artifactId//packaging
```

15.3.4. バージョン範囲の指定

Mvn URL で **version** 値を指定する場合、単純なバージョン番号の代わりにバージョンの範囲 (標準の Maven バージョン管理構文を使用) を指定できます。角括弧 **[** および **]** を使用して含まれる範囲を示し、括弧 **(** および **)** を使用して除外される範囲を示します。たとえば、範囲 **[1.0.4,2.0)** は、**1.0.4 ⇐ v < 2.0** を満たすすべてのバージョン **v** に一致します。このバージョン範囲は、以下のように Mvn URL で使用できます。

```
mvn:org.fusesource.example/bundle-demo/[1.0.4,2.0)
```

15.3.5. Mvn URL ハンドラーの設定

Mvn URL を初めて使用する前に、以下のように Mvn URL ハンドラー設定をカスタマイズする必要がある場合があります。

1. 「Mvn URL 設定の確認」

2. 「設定ファイルを編集します。」
3. 「ローカルリポジトリーの場所のカスタマイズ」

15.3.6. Mvn URL 設定の確認

Mvn URL ハンドラーは、ローカル Maven リポジトリーへの参照を解決し、リモート Maven リポジトリーのリストを維持します。Mvn URL を解決するとき、ハンドラーは最初にローカルリポジトリーを、その後にリモートリポジトリーを検索して、指定された Maven アニティーを見つけます。Mvn URL の解決に問題がある場合は、最初にハンドラー設定を確認し、URL の解決に使用するローカルリポジトリーおよびリモートリポジトリーを確認することができます。

Mvn URL 設定を確認するには、コンソールで以下のコマンドを入力します。

```
JBossFuse:karaf@root> config:edit org.ops4j.pax.url.mvn
JBossFuse:karaf@root> config:proplist
```

config:edit コマンドは、**config** ユーティリティーの焦点を **org.ops4j.pax.url.mvn** 永続 ID に属するプロパティーに切り替えます。**config:proplist** コマンドは、現在の永続 ID のプロパティー設定をすべて出力します。**org.ops4j.pax.url.mvn** を対象とすると、以下のようなリストが表示されるはずです。

```
   org.ops4j.pax.url.mvn.defaultRepositories = file:/path/to/JBossFuse/jboss-fuse-7.8.0.fuse-780038-redhat-00001/system@snapshots@id=karaf.system,file:/home/userid/.m2/repository@snapshots@id=local,file:path/to/JBossFuse/jboss-fuse-7.8.0.fuse-780038-redhat-00001/local-repo@snapshots@id=karaf.local-repo,file:/path/to/JBossFuse/jboss-fuse-7.8.0.fuse-780038-redhat-00001/system@snapshots@id=child.karaf.system
   org.ops4j.pax.url.mvn.globalChecksumPolicy = warn
   org.ops4j.pax.url.mvn.globalUpdatePolicy = daily
   org.ops4j.pax.url.mvn.localRepository = /path/to/JBossFuse/jboss-fuse-7.8.0.fuse-780038-redhat-00001/data/repository
   org.ops4j.pax.url.mvn.repositories = http://repo1.maven.org/maven2@id=maven.central.repo,
https://maven.repository.redhat.com/ga@id=redhat.ga.repo,
https://maven.repository.redhat.com/earlyaccess/all@id=redhat.ea.repo,
https://repository.jboss.org/nexus/content/groups/ea@id=fuseearlyaccess
   org.ops4j.pax.url.mvn.settings = /path/to/jboss-fuse-7.8.0.fuse-780038-redhat-00001/etc/maven-settings.xml
   org.ops4j.pax.url.mvn.useFallbackRepositories = false
   service.pid = org.ops4j.pax.url.mvn
```

localRepository 設定は、ハンドラーによって現在使用されているローカルリポジトリーの場所を示し、**repositories** の設定は、ハンドラーによって現在使用されているリモートリポジトリーのリストを示します。

15.3.7. 設定ファイルを編集します。

Mvn URL ハンドラーのプロパティー設定をカスタマイズするには、以下の設定ファイルを編集します。

```
InstallDir/etc/org.ops4j.pax.url.mvn.cfg
```

このファイルの設定により、ローカル Maven リポジトリーの場所を明示的に指定し、Maven リポジトリーや Maven プロキシーサーバー設定を削除できるようになります。これらの設定の詳細は、設定ファイルのコメントを参照してください。

15.3.8. ローカルリポジトリーの場所のカスタマイズ

特に、ローカルの Maven リポジトリーがデフォルト以外の場所にある場合は、ローカルにビルドする Maven アーティファクトにアクセスするために、明示的に設定する必要がある場合があります。**org.ops4j.pax.url.mvn.cfg** 設定ファイルで、**org.ops4j.pax.url.mvn.localRepository** プロパティーのコメントを解除し、ローカルの Maven リポジトリーの場所に設定します。以下に例を示します。

```
# Path to the local maven repository which is used to avoid downloading
# artifacts when they already exist locally.
# The value of this property will be extracted from the settings.xml file
# above, or defaulted to:
#     System.getProperty( "user.home" ) + "/.m2/repository"
#
org.ops4j.pax.url.mvn.localRepository=file:E:/Data/.m2/repository
```

15.3.9. 参照資料

mvn URL 構文の詳細は、元の Pax URL の Mvn Protocol ドキュメントを参照してください。

15.4. WRAP URL ハンドラー

15.4.1. 概要

まだバンドルとしてパッケージ化されていない JAR ファイルを参照する必要がある場合は、Wrap URL ハンドラーを使用して動的に変換できます。Wrap URL ハンドラーの実装は、Peter Krien のオープンソース Bnd ユーティリティーに基づいています。

15.4.2. 構文

Wrap URL の構文は次のとおりです。

```
wrap:locationURL[,instructionsURL][$instructions]
```

locationURL は、JAR を検索する任意の URL にすることができます (参照される JAR がバンドルとしてフォーマットされて **いない** 場合)。オプションの **instructionsURL** は、バンドル変換の実行方法を指定する Bnd プロパティーファイルを参照します。任意の **instructions** は、バンドル変換の実行方法を指定する、アンパサンド **&** で区切られた Bnd プロパティーのリストです。

15.4.3. デフォルトの手順

ほとんどの場合、API JAR ファイルをラップするには、デフォルトの Bnd 命令で十分です。デフォルトでは、Wrap は、表15.1「JAR をラッピングするデフォルトの命令」 に示されているようにマニフェストヘッダーを JAR の **META-INF/Manifest.mf** ファイルに追加します。

表15.1 JAR をラッピングするデフォルトの命令

マニフェストヘッダー	デフォルト値
Import-Package	***;resolution:=optional**

マニフェストヘッダー	デフォルト値
Export-Package	ラップされた JAR からのすべてのパッケージ。
Bundle-SymbolicName	JAR ファイルの名前。セット **[a-zA-Z0-9_-]** にない文字はすべてアンダースコア _ に置き換えられます。

15.4.4. 例

以下の Wrap URL は、Maven リポジトリーでバージョン 1.1 の **commons-logging** JAR のを見つけ、デフォルトの Bnd プロパティーを使用してこれを OSGi バンドルに変換します。

```
wrap:mvn:commons-logging/commons-logging/1.1
```

以下の Wrap URL は、ファイル **E:\Data\Examples\commons-logging-1.1.bnd** からの Bnd プロパティーを使用します。

```
wrap:mvn:commons-logging/commons-logging/1.1,file:E:/Data/Examples/commons-logging-1.1.bnd
```

以下の Wrap URL は、**Bundle-SymbolicName** プロパティーと **Bundle-Version** プロパティーを明示的に指定します。

```
wrap:mvn:commons-logging/commons-logging/1.1$Bundle-SymbolicName=apache-comm-
log&Bundle-Version=1.1
```

上記の URL をコマンドライン引数として使用する場合は、以下のようにドル記号を **\$** のようにエスケープしてコマンドラインで処理されないようにする必要がある場合があります。

```
wrap:mvn:commons-logging/commons-logging/1.1\$Bundle-SymbolicName=apache-comm-
log&Bundle-Version=1.1
```

15.4.5. 参照資料

wrap URL ハンドラーの詳細は、以下の参考資料を参照してください。

- Bnd プロパティーと Bnd 命令ファイルの詳細は、Bnd ツールのドキュメント を参照してください。
- 元の Pax URL Wrap Protocol ドキュメント。

15.5. WAR URL ハンドラー

15.5.1. 概要

OSGi コンテナーに WAR ファイルをデプロイする必要がある場合は、ここで説明するように **war:** を WAR URL の先頭に追加し、必要なマニフェストヘッダーを WAR ファイルに自動的に追加できます。

15.5.2. 構文

War URL は、以下の構文のいずれかを使用して指定されます。

```
war:warURL
warref:instructionsURL
```

war スキームを使用する最初の構文は、デフォルトの命令を使用してバンドルに変換される WAR ファイルを指定します。**warURL** には、WAR ファイルを特定する URL を指定できます。

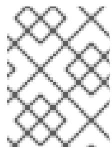
warref スキームを使用する 2 番目の構文は、変換命令 (このハンドラーに固有の命令を含む) が含まれる Bnd プロパティーファイル **instructionsURL** を指定します。この構文では、参照される WAR ファイルの場所は URL に明示的に**表示されません**。代わりに、WAR ファイルは、プロパティーファイルの (必須の) **WAR-URL** プロパティーにより指定されます。

15.5.3. WAR 固有のプロパティー/命令

以下のように、**.bnd** 命令ファイルのプロパティーの一部は、War URL ハンドラーに固有です。

WAR-URL
: (必須) バンドルに変換する War ファイルの場所を指定します。

Web-ContextPath
: Web コンテナー内にデプロイされた後に、この Web アプリケーションへのアクセスに使用される URL パスを指定します。



注記

PAX Web の以前のバージョンでは、現在**非推奨**となっているプロパティー **Webapp-Context** が使用されていました。

15.5.4. デフォルトの手順

デフォルトでは、表15.2「WAR ファイルをラッピングするデフォルトの命令」 に示すように、War URL ハンドラーは WAR の **META-INF/Manifest.mf** ファイルにマニフェストヘッダーを追加します。

表15.2 WAR ファイルをラッピングするデフォルトの命令

マニフェストヘッダー	デフォルト値
Import-Package	**javax.,org.xml.,org.w3c.***
Export-Package	パッケージはエクスポートされません。
Bundle-SymbolicName	WAR ファイルの名前。セット **[a-zA-Z0-9_-\.]** にない文字はすべてピリオド **.** に置き換えられます。
Web-ContextPath	デフォルト値はありません。しかし、WAR エクステンダーは、デフォルトで **Bundle-SymbolicName** の値を使用します。

マニフェストヘッダー	デフォルト値
Bundle-ClassPath	明示的に指定されたクラスパスエントリーの他に、以下のエントリーが自動的に追加されます。 • **.** • **WEB-INF/classes** • **WEB-INF/lib** ディレクトリーからのすべての JAR。

15.5.5. 例

以下の War URL は、Maven リポジトリーの **wicket-examples** WAR のバージョン 1.4.7 を見つけ、これをデフォルトの命令を使用して OSGi バンドルに変換します。

```
war:mvn:org.apache.wicket/wicket-examples/1.4.7/war
```

以下の Wrap URL は、**Web-ContextPath** を明示的に指定します。

```
war:mvn:org.apache.wicket/wicket-examples/1.4.7/war?Web-ContextPath=wicket
```

以下の War URL は、**wicket-examples-1.4.7.bnd** ファイルの **WAR-URL** プロパティーによって参照される WAR ファイルを変換し、**.bnd** ファイルの他の命令を使用して WAR を OSGi バンドルに変換します。

```
warref:file:E:/Data/Examples/wicket-examples-1.4.7.bnd
```

15.5.6. 参照資料

war URL 構文の詳細は、元の Pax URL の War Protocol ドキュメントを参照してください。

パート II. ユーザーガイド

このパートには、Apache Karaf on Red Hat Fuse の設定および準備の情報が含まれています。

第16章 APACHE KARAF のデプロイガイドの概要

概要

Apache Karaf のデプロイのユーザーガイドセクションを使用する前に、Installing on Apache Karaf の手順にしたがって、最新バージョンの Red Hat Fuse をインストールしておく必要があります。

16.1. FUSE 設定の概要

OSGi 設定管理サービスは、デプロイされたサービスの設定情報を指定し、サービスが有効な場合にそのデータを確実に受信するようにします。

16.2. OSGI の設定

設定は、**FUSE_HOME/etc** ディレクトリーの **.cfg** ファイルから読み取られた名前と値のペアのリストです。ファイルは Java プロパティーファイル形式を使用して解釈されます。ファイル名は、設定されるサービスの永続識別子 (PID) にマッピングされます。OSGi では、PID を使用して、コンテナーの再起動後もサービスを特定します。

16.3. 設定ファイル

以下のファイルを使用して Red Hat Fuse ランタイムを設定できます。

表16.1 Fuse 設定ファイル

ファイル名	説明
config.properties	コンテナーの主な設定ファイル。
custom.properties	コンテナー用のカスタムプロパティーの主な設定ファイル。
keys.properties	SSH キーベースプロトコルを使用して Fuse ランタイムにアクセスできるユーザーを一覧表示します。ファイルの内容は、**username=publicKey,role** の形式を取ります。
org.apache.karaf.features.repos.cfg	機能リポジトリーの URL。
org.apache.karaf.features.cfg	Fuse の初回起動に、登録する機能リポジトリーのリストとインストールする機能のリストを設定します。
org.apache.karaf.jaas.cfg	Karaf JAAS ログインモジュールのオプションを設定します。暗号化されたパスワードの設定に主に使用されます (デフォルトでは無効)。
org.apache.karaf.log.cfg	**log** コンソールコマンドの出力を設定します。
org.apache.karaf.management.cfg	JMX システムを設定します。

ファイル名	説明
org.apache.karaf.shell.cfg	リモートコンソールのプロパティーを設定します。
org.ops4j.pax.logging.cfg	ロギングシステムを設定します。
org.ops4j.pax.transx.tm.narayana.cfg	Narayana トランザクションマネージャーを設定します。
org.ops4j.pax.url.mvn.cfg	追加の URL リゾルバーを設定します。
org.ops4j.pax.web.cfg	デフォルトの Undertow コンテナー (Web サーバー) を設定します。Red Hat Fuse Apache CXF セキュリティーガイドの Undertow HTTP サーバーの保護 を参照してください。
startup.properties	コンテナーで起動するバンドルとその開始レベルを指定します。エントリーは、**bundle=start-level** 形式を取ります。
system.properties	Java システムプロパティーを指定します。このファイルで設定されているプロパティーは、**System.getProperties()** を使用して起動時に利用できます。
users.properties	リモートで、または Web コンソールを介して Fuse ランタイムにアクセスできるユーザーを一覧表示します。ファイルの内容は、**username=password,role** の形式を取ります。
setenv または **setenv.bat**	このファイルは、**/bin** ディレクトリーにあります。JVM オプションの設定に使用されます。ファイルの内容は **JAVA_MIN_MEM=512M** の形式を取ります。**512M** は Java メモリーの最小サイズです。詳細は、「Java オプションの設定」 を参照してください。

16.4. 高度な UNDERTOW 設定

16.4.1. IO 設定

PAXWEB-1255 以降、リスナーによって使用される XNIO ワーカーおよびバッファープールの設定を変更できます。undertow.xml テンプレートには、一部の IO 関連のパラメーターのデフォルト値を指定するセクションがあります。

```
<!-- Only "default" worker and buffer-pool are supported and can be used to override the default
values used by all listeners
    buffer-pool:
     - buffer-size defaults to:
```

```
        - when < 64MB of Xmx: 512
        - when < 128MB of Xmx: 1024
        - when >= 128MB of Xmx: 16K - 20
      - direct-buffers defaults to:
        - when < 64MB of Xmx: false
        - when >= 64MB of Xmx: true
     worker:
      - io-threads defaults to Math.max(Runtime.getRuntime().availableProcessors(), 2);
      - task-core-threads and task-max-threads default to io-threads * 8
-->

<!--
<subsystem xmlns="urn:jboss:domain:io:3.0">
    <buffer-pool name="default" buffer-size="16364" direct-buffers="true" />
    <worker name="default" io-threads="8" task-core-threads="64" task-max-threads="64" task-
keepalive="60000" />
</subsystem>
-->
```

以下の **buffer-pool** パラメーターを指定できます。

buffer-size

IO 操作に使用されるバッファーのサイズを指定します。指定のない場合は、利用可能なメモリーに応じてサイズが計算されます。

direct-buffers

java.nio.ByteBuffer#allocateDirect または java.nio.ByteBuffer#allocate を使用するかどうかを決定します。

以下の **worker** パラメーターを指定できます。

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。

task-core-threads

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、最大スレッドの数が CPU の数の 16 倍に設定されます。

16.5. 設定ファイルの命名規則

設定ファイルのファイルの命名規則は、設定が OSGi 管理対象サービス用か、OSGi Managed Service ファクトリーのいずれを対象としているかによって異なります。

OSGi Managed Service の設定ファイルは、以下の命名規則に従います。

```
<PID>.cfg
```

<PID> は、OSGi Managed Service の **永続 ID** です (OSGi Configuration Admin 仕様で定義されます)。永続 ID は通常、**org.ops4j.pax.web** のようにドットで区切られています。

OSGi Managed Service Factory の設定ファイルは、以下の命名規則に従います。

```
<PID>-<InstanceID>.cfg
```

<PID> は、OSGi Managed Service Factory の **永続 ID** です。マネージドサービスファクトリーの **<PID>,** の場合、ハイフンの後に任意のインスタンス ID **<InstanceID>** を追加できます。次に、マネージドサービスファクトリーは、見つかった **<InstanceID>** ごとに一意のサービスインスタンスを作成します。

16.6. JAVA オプションの設定

Java Options は、Linux の **/bin/setenv** ファイルまたは Windows の **bin/setenv.bat** ファイルを使用して設定できます。このファイルを使用して、Java オプションのグループを直接設定します。JAVA_MIN_MEM, JAVA_MAX_MEM, JAVA_PERM_MEM, JAVA_MAX_PERM_MEM.他の Java オプションは、EXTRA_JAVA_OPTS 変数を使用して設定できます。

たとえば、JVM が使用する最小メモリーを割り当てるには、以下を実行します。

```
JAVA_MIN_MEM=512M # Minimum memory for the JVM
```

```
To set a Java option other than the direct options, use
```

```
EXTRA_JAVA_OPTS="Java option"
```

```
For example,
```

```
EXTRA_JAVA_OPTS="-XX:+UseG1GC"
```

16.7. 設定コンソールのコマンド

Fuse 7.12 の設定を変更または問い合わせするために使用できるコンソールコマンドが複数あります。

config: コマンドの詳細は、Apache Karaf コンソールリファレンス の Config セクションを参照してください。

16.8. JMX CONFIGMBEAN

JMX レイヤーでは、**MBean** は設定管理専用です。

ConfigMBean オブジェクト名は **org.apache.karaf:type=config,name=*`** です。

14.1.2.1.属性

config MBean には、すべての設定 PID の一覧が含まれます。

14.1.2.2.操作

表16.2 JMX MBean 操作

操作名	説明
listProperties(pid)	設定 pid のプロパティー (property=value 形式の) 一覧を返します。

操作名	説明
deleteProperty(pid, property)	設定 pid からプロパティーを削除します。
appendProperty(pid, property, value)	設定 pid のプロパティーの値の最後に値を追加します。
setProperty(pid, property, value)	設定 pid のプロパティーの値を設定します。
delete(pid)	pid で識別される設定を削除します。
create(pid)	pid で、空の (プロパティーなし) 設定を作成します。
update(pid, properties)	指定のプロパティーマップを使用して pid で識別される設定を更新します。

16.9. コンソールの使用

16.9.1. 利用可能なコマンド

コンソールで利用可能なコマンドを一覧表示するには、**help** を使用できます。

```
karaf@root()> help
bundle                          Enter the subshell
bundle:capabilities              Displays OSGi capabilities of a given bundles.
bundle:classes                   Displays a list of classes/resources contained in the bundle
bundle:diag                      Displays diagnostic information why a bundle is not Active
bundle:dynamic-import            Enables/disables dynamic-import for a given bundle.
bundle:find-class                Locates a specified class in any deployed bundle
bundle:headers                   Displays OSGi headers of a given bundles.
bundle:id                        Gets the bundle ID.
...
```

すべてのコマンドの一覧と簡単な説明が表示されます。

tab キーを使用して、すべてのコマンドのクイック一覧を取得できます。

```
karaf@root()> Display all 294 possibilities? (y or n)
...
```

16.9.2. サブシェルおよび完了モード

コマンドには、スコープと名前があります。たとえば、**feature:list** コマンドでは、**feature** が範囲で、**list** が名前となります。

Karaf はコマンドをスコープごとにグループ化します。スコープごとにサブシェルが形成されます。

完全修飾名 (scope:name) でコマンドを直接実行できます。

```
karaf@root()> feature:list
...
```

または、サブシェルに入り、コマンドコンテキストをサブシェルに入力します。

```
karaf@root()> feature
karaf@root(feature)> list
```

サブシェル名 (ここでは **feature**) を入力して、サブシェルに直接入力できることに注意してください。サブシェルから別のシェルに直接切り替えることができます。

```
karaf@root()> feature
karaf@root(feature)> bundle
karaf@root(bundle)>
```

プロンプトには、() の間にある現在のサブシェルが表示されます。

exit コマンドは、親サブシェルに移動します。

```
karaf@root()> feature
karaf@root(feature)> exit
karaf@root()>
```

補完モードは、タブキーの動作と help コマンドを定義します。

利用可能なモードは 3 つあります。

- GLOBAL
- FIRST
- SUBSHELL

etc/org.apache.karaf.shell.cfg ファイルの completionMode プロパティーを使用して、デフォルトの補完モードを定義できます。デフォルトには以下が含まれます。

```
completionMode = GLOBAL
```

shell:completion コマンドを使用して、(Karaf シェルコンソールの使用中に) 補完モードを即座に変更することもできます。

```
karaf@root()> shell:completion
GLOBAL
karaf@root()> shell:completion FIRST
karaf@root()> shell:completion
FIRST
```

shell:completion は現在使用されている補完モードについて通知できます。必要な新しい補完モードを指定することもできます。

GLOBAL 補完モードは Karaf 4.0.0 のデフォルトモードです (主に移行の目的で)。

GLOBAL モードは、実際には subshell を使用せず、以前のバージョンの Karaf と同じ動作です。

タブキーを入力すると、どのサブシェルで操作していても、補完ですべてのコマンドとすべてのエイリアスが表示されます。

```
karaf@root()> <TAB>
karaf@root()> Display all 273 possibilities? (y or n)
...
karaf@root()> feature
karaf@root(feature)> <TAB>
karaf@root(feature)> Display all 273 possibilities? (y or n)
```

FIRST 補完モードは、GLOBAL 補完モードの代わりとなります。

ルートレベルのサブシェルでタブキーを入力すると、(GLOBAL モードのように) すべてのサブシェルからのコマンドとエイリアスが表示されます。ただし、サブシェルでタブキーを入力すると、補完として、現在のサブシェルのコマンドのみが表示されます。

```
karaf@root()> shell:completion FIRST
karaf@root()> <TAB>
karaf@root()> Display all 273 possibilities? (y or n)
...
karaf@root()> feature
karaf@root(feature)> <TAB>
karaf@root(feature)>
info install list repo-add repo-list repo-remove uninstall version-list
karaf@root(feature)> exit
karaf@root()> log
karaf@root(log)> <TAB>
karaf@root(log)>
clear display exception-display get log set tail
```

SUBSHELL 補完モードは、実際のサブシェルモードです。

ルートレベルでタブキーを入力すると、補完としてサブシェルコマンド (サブシェルに移動する) とグローバルエイリアスが表示されます。サブシェルで TAB キーを入力すると、補完として、現在のサブシェルのコマンドが表示されます。

```
karaf@root()> shell:completion SUBSHELL
karaf@root()> <TAB>
karaf@root()>
* bundle cl config dev feature help instance jaas kar la ld lde log log:list man package region service
shell ssh system
karaf@root()> bundle
karaf@root(bundle)> <TAB>
karaf@root(bundle)>
capabilities classes diag dynamic-import find-class headers info install list refresh requirements
resolve restart services start start-level stop
uninstall update watch
karaf@root(bundle)> exit
karaf@root()> camel
karaf@root(camel)> <TAB>
karaf@root(camel)>
backlog-tracer-dump backlog-tracer-info backlog-tracer-start backlog-tracer-stop context-info
context-list context-start context-stop endpoint-list route-info route-list route-profile route-reset-stats
route-resume route-show route-start route-stop route-suspend
```

16.9.3. UNIX のような環境

Karaf コンソールは、Unix のような完全な環境を提供します。

16.9.3.1. ヘルプまたは man

利用できるすべてのコマンドを表示する **help** コマンドの使用方法についてすでに説明しました。

ただし、**help** コマンドや、**help** コマンドのエイリアスである **man** コマンドを使用して、コマンドの詳細を取得することもできます。コマンドの **--help** オプションを使用して、別のフォームを使用してコマンドのヘルプを取得することもできます。

以下のコマンドのようになります。

```
karaf@root()> help feature:list
karaf@root()> man feature:list
karaf@root()> feature:list --help
```

以下で、すべて同じヘルプ出力を生成します。

```
DESCRIPTION
        feature:list

        Lists all existing features available from the defined repositories.

SYNTAX
        feature:list [options]

OPTIONS
        --help
                Display this help message
        -o, --ordered
                Display a list using alphabetical order
        -i, --installed
                Display a list of all installed features only
        --no-format
                Disable table rendered output
```

16.9.3.2. 補完

タブキーを入力すると、Karaf は補完しようとします。

- subshell
- commands
- aliases
- コマンド引数
- コマンドオプション

16.9.3.3. エイリアス

エイリアスは、指定したコマンドに関連する別の名前です。

shell:alias コマンドは、新しいエイリアスを作成します。たとえば、実際の **feature:list -i** コマンドの **list-installed-features** エイリアスを作成するには、次の操作を実行できます。

```
karaf@root()> alias "list-features-installed = { feature:list -i }"
karaf@root()> list-features-installed
Name       | Version | Required | State   | Repository     | Description
------------------------------------------------------------------------------------------------------------------------
---
feature    | 4.0.0   | x        | Started | standard-4.0.0 | Features Support
shell      | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Shell
deployer   | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Deployer
bundle     | 4.0.0   | x        | Started | standard-4.0.0 | Provide Bundle support
config     | 4.0.0   | x        | Started | standard-4.0.0 | Provide OSGi ConfigAdmin support
diagnostic | 4.0.0   | x        | Started | standard-4.0.0 | Provide Diagnostic support
instance   | 4.0.0   | x        | Started | standard-4.0.0 | Provide Instance support
jaas       | 4.0.0   | x        | Started | standard-4.0.0 | Provide JAAS support
log        | 4.0.0   | x        | Started | standard-4.0.0 | Provide Log support
package    | 4.0.0   | x        | Started | standard-4.0.0 | Package commands and mbeans
service    | 4.0.0   | x        | Started | standard-4.0.0 | Provide Service support
system     | 4.0.0   | x        | Started | standard-4.0.0 | Provide System support
kar        | 4.0.0   | x        | Started | standard-4.0.0 | Provide KAR (KARaf archive) support
ssh        | 4.0.0   | x        | Started | standard-4.0.0 | Provide a SSHd server on Karaf
management | 4.0.0   | x        | Started | standard-4.0.0 | Provide a JMX MBeanServer and a set of
MBeans in
```

ログイン時に、Apache Karaf コンソールは、エイリアスを作成できる **etc/shell.init.script** ファイルを読み取ります。これは、Unix の bashrc またはプロファイルファイルと似ています。

```
ld = { log:display $args } ;
lde = { log:exception-display $args } ;
la = { bundle:list -t 0 $args } ;
ls = { service:list $args } ;
cl = { config:list "(service.pid=$args)" } ;
halt = { system:shutdown -h -f $args } ;
help = { *:help $args | more } ;
man = { help $args } ;
log:list = { log:get ALL } ;
```

デフォルトでは利用可能なエイリアスをここで確認できます。

- **ld** はログを表示する短縮形式です (**log:display** コマンドのエイリアス)。
- **lde** は例外を表示する短縮形式です (**log:exception-display** コマンドのエイリアス)。
- **la** はすべてのバンドルを一覧表示するための短縮形式です (**bundle:list -t 0** コマンドのエイリアス) 。
- **ls** はすべてのサービスを一覧表示する短縮形式です (**service:list** コマンドのエイリアス)。
- **cl** は全設定を一覧表示するための短縮形式です (**config:list** コマンドのエイリアス)。
- **halt** は Apache Karaf をシャットダウンするための短縮形式です (**system:shutdown -h -f** コマンドへのエイリアス)。
- **help** は、ヘルプを表示する短縮形式です (***:help** コマンドのエイリアス)。

- **man** は help と同じです (**help** コマンドのエイリアス)。
- **log:list** はすべてのロガーおよびレベルを表示します (**log:get ALL** コマンドのエイリアス)。

etc/shell.init.script ファイルで独自のエイリアスを作成できます。

16.9.3.4. キーバインディング

ほとんどの Unix 環境と同様に、Karaf コンソールはいくつかの主要なバインディングをサポートしています。

- 矢印キー (コマンド履歴内で移動する)
- CTRL-D (Karaf をログアウト/シャットダウンする)
- CTRL-R (以前実行したコマンドを検索する)
- CTRL-U (現在の行を削除する)

16.9.3.5. パイプ

あるコマンドの出力を別の入力としてパイプできます。これは、| 文字を使用したパイプです。

```
karaf@root()> feature:list |grep -i war
pax-war                    | 4.1.4                        |          | Uninstalled | org.ops4j.pax.web-4.1.4  | Provide
support of a full WebContainer
pax-war-tomcat             | 4.1.4                        |          | Uninstalled | org.ops4j.pax.web-4.1.4  |
war                        | 4.0.0                        |          | Uninstalled | standard-4.0.0           | Turn Karaf as
a full WebContainer
blueprint-web              | 4.0.0                        |          | Uninstalled | standard-4.0.0           | Provides
an OSGI-aware Servlet ContextListener fo
```

16.9.3.6. grep、more、find など

Karaf コンソールでは、Unix 環境に似たコアコマンドを使用できます。

- **shell:alias** は既存のコマンドのエイリアスを作成します。
- **shell:cat** はファイルまたは URL の内容を表示します。
- **shell:clear** は現在のコンソール表示の消去します。
- **shell:completion** は現在の補完モードを表示または変更します。
- **shell:date** は現在の日付を表示します (必要に応じて形式を使用)。
- **shell:each** は引数のリストでクロージャーを実行します。
- **shell:echo** は引数を標準出力にエコーおよび出力します。
- **shell:edit** は現在のファイルまたは URL のテキストエディターを呼び出します。
- **shell:env** はシェルセッション変数の値を表示または設定します。
- **shell:exec** はシステムコマンドを実行します。

- **shell:grep** は指定のパターンに一致する行を出力します。
- **shell:head** は入力の最初の行を表示します
- **shell:history** はコマンド履歴を出力します。
- **shell:if** はスクリプトで条件 (if、then、else blocks) の使用を可能にします。
- **shell:info** は現在の Karaf インスタンスに関するさまざまな情報を出力します。
- **shell:java** は Java アプリケーションを実行します。
- **shell:less** ファイルページャーです。
- **shell:logout** は現行セッションからシェルの切断します。
- **shell:more** はファイルページャーです
- **shell:new** は新しい Java オブジェクトを作成します。
- **shell:printf** は引数をフォーマットして出力します
- **shell:sleep** は少し眠ってから目を覚ます
- **shell:sort** はすべてのファイルのソートされた連結を標準出力に書き込みます。
- **shell:source** はスクリプトに含まれるコマンドを実行します。
- **shell:stack-traces-print** は、コマンドの実行によって例外が発生すると、コンソールで完全なスタックトレースを出力します。
- **shell:tac** は STDIN をキャプチャーし、文字列として返します。
- **shell:tail** は入力の最後の行を表示します。
- **shell:threads** は現在のスレッドを出力します。
- **shell:watch** は定期的にコマンドを実行し、出力を更新します。
- **shell:wc** は各ファイルの改行、単語、およびバイト数を出力します。
- **shell:while** は条件が true である間にループします。

コマンドの完全修飾名を使用する必要はありませんが、コマンド名が一意であれば、直接使用できます。つまり、shell:head の代わりに head を使用できます。

ここでも、**help** コマンドまたは **--help** オプションを使用して、これらのコマンドの詳細とすべてのオプションを確認できます。

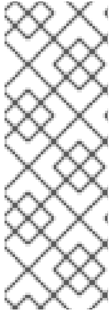
16.9.3.7. スクリプト

Apache Karaf コンソールは、Unix の bash または csh と同様に完全なスクリプト言語をサポートします。

each (**shell:each**) コマンドは、リストで処理を繰り返すことができます。

```
karaf@root()> list = [1 2 3]; each ($list) { echo $it }
```

```
1
2
3
```



注記

同じループを **shell:while** コマンドで記述できます。

```
karaf@root()> a = 0 ; while { %((a+=1) <= 3) } { echo $a }
1
2
3
```

(前述の例のように) 一覧を独自に作成することも、一部のコマンドでもリストを返すこともできます。

コンソールは、**$list** でアクセス可能な **list** という名前のセッション変数を作成したことに注意してください。

$it 変数は、現在のオブジェクトに対応する暗黙的な変数です (ここではリストの現在の反復処理された値)。

[] でリストを作成すると、Apache Karaf コンソールは Java ArrayList を作成します。つまり、ArrayList オブジェクトで利用可能なメソッド (例: get または size) を使用できます。

```
karaf@root()> list = ["Hello" world]; echo ($list get 0) ($list get 1)
Hello world
```

ここで、オブジェクトのメソッドを呼び出すことは、**(object method argument)** を直接使用することに注意してください。**($list get 0)** は **$list.get(0)** を意味します。**$list** は ArrayList です。

class 表記には、オブジェクトの詳細が表示されます。

```
karaf@root()> $list class
...
ProtectionDomain     ProtectionDomain  null
 null
 <no principals>
 java.security.Permissions@6521c24e (
 ("java.security.AllPermission" "<all permissions>" "<all actions>")
)


Signers              null
SimpleName           ArrayList
TypeParameters       [E]
```

指定のタイプに変数をキャストすることができます。

```
karaf@root()> ("hello world" toCharArray)
[h, e, l, l, o,  , w, o, r, l, d]
```

失敗すると、キャストの例外が表示されます。

```
karaf@root()> ("hello world" toCharArray)[0]
Error executing command: [C cannot be cast to [Ljava.lang.Object;
```

shell:source コマンドを使用して、スクリプトを呼び出しすることができます。

```
karaf@root> shell:source script.txt
True!
```

script.txt に含まれる場所:

```
foo = "foo"
if { $foo equals "foo" } {
  echo "True!"
}
```

> **注記**
>
> スペースは、スクリプトの記述時には重要です。たとえば、以下のスクリプトが正しくありません。
>
> ```
> if{ $foo equals "foo" } ...
> ```
>
> そして、次のようなエラーが発生します。
>
> ```
> karaf@root> shell:source script.txt
> Error executing command: Cannot coerce echo "true!"() to any of []
> ```
>
> **if** ステートメントの後にスペースがないためです。

エイリアスについては、**etc/shell.init.script** ファイルで init スクリプトを作成できます。エイリアスを使ってスクリプトに名前を付けることもできます。実際にはエイリアスは単なるスクリプトです。

詳細は、開発者ガイドのスクリプトセクションを参照してください。

16.9.4. セキュリティー

Apache Karaf コンソールは、ロールベースアクセス制御 (RBAC) のセキュリティーメカニズムをサポートします。コンソールに接続するユーザー、ユーザーのグループおよびロール、コマンドの実行パーミッションに応じて定義でき、引数として指定可能な値を制限できます。

コンソールのセキュリティーは、このユーザーガイドの セキュリティーセクション を参照してください。

16.10. プロビジョニング

Apache Karaf は、Karaf 機能の概念を使用してアプリケーションとモジュールのプロビジョニングをサポートします。

16.10.1. アプリケーション

アプリケーションをプロビジョニングすると、すべてのモジュール、設定、および推移的なアプリケーションがインストールされます。

16.10.2. OSGi

OSGi アプリケーションのデプロイメントをネイティブにサポートします。

OSGi アプリケーションは OSGi バンドルのセットです。OSGi バンドルは通常の jar ファイルで、jar MANIFEST に別のメタデータを追加します。

OSGi では、バンドルは他のバンドルに依存します。そのため、OSGi アプリケーションをデプロイする場合には通常、アプリケーションが必要とする他のバンドルを多数、先にデプロイする必要があります。

そのため、まずこれらのバンドルを見つけ、バンドルをインストールする必要があります。ここでも、これらの "dependency" バンドルは、独自の依存関係を満たすために他のバンドルを必要とする場合があります。

さらに、通常、アプリケーションには設定が必要です (ユーザーガイドの [Configuration section|configuration] を参照)。そのため、依存関係バンドルに加えてアプリケーションを起動する前に設定を作成またはデプロイする必要があります。

このように、アプリケーションのプロビジョニングは非常に時間がかかり、煩雑になります。

16.10.3. 機能とリゾルバー

Apache Karaf では、アプリケーションをシンプルで柔軟にプロビジョニングする方法をご利用いただけます。

Apache Karaf では、アプリケーションのプロビジョニングは Apache Karaf の機能です。

機能は、アプリケーションを次のように記述します。

- 名前
- バージョン
- オプションの説明 (最終的に長い説明が含まれる)
- バンドルのセット
- 設定または設定ファイルのセット (任意)
- 依存関係機能のセット (任意)

機能をインストールすると、Apache Karaf はその機能に記述されているすべてのリソースをインストールします。つまり、機能に記述されている全バンドル、設定、および依存関係機能を自動的に解決してインストールします。

機能リゾルバーはサービス要件をチェックし、要件に一致するサービスを提供するバンドルをインストールします。デフォルトモードでは、新しいスタイルの機能リポジトリー (基本的に、スキーマが 1.3.0 以上の機能リポジトリー XML) に対してのみこの動作が有効になります。以前のスタイルの機能リポジトリー (Karaf 2 または 3 から提供) には適用されません。

serviceRequirements プロパティーを使用して、**etc/org.apache.karaf.features.cfg** ファイルのサービス要件の強制モードを変更できます。

```
serviceRequirements=default
```

以下の値を使用できます。

- disable: 古いスタイルと新しいスタイルの両方の機能リポジトリーのサービス要件は完全に無視されます
- default: 古いスタイルの機能リポジトリーのサービス要件が無視され、新しいスタイルの機能リポジトリーのサービス要件が有効になります。
- enforce: 古いスタイルおよび新しいスタイルの機能リポジトリーのサービス要件が常に検証されます。

さらに、機能で要件を定義することもできます。その場合、Karaf は、要件を満たす機能を提供するバンドルまたは機能を自動的に追加できます。

機能には、インストール、開始、停止、更新、アンインストールの完全なライフサイクルがあります。

16.10.4. 機能リポジトリー

機能は、機能 XML 記述子で記述します。この XML ファイルには、一連の機能の記述が含まれています。

機能 XML 記述子は機能リポジトリーと呼ばれます。機能をインストールする前に、機能を提供する features リポジトリーを登録する必要があります (後で説明するように **feature:repo-add** コマンドまたは FeatureMBean を使用します)。

たとえば、以下の XML ファイル (または features リポジトリー) は、**feature1** および **feature2** 機能を記述します。

```
<features xmlns="http://karaf.apache.org/xmlns/features/v1.3.0">
  <feature name="feature1" version="1.0.0">
    <bundle>...</bundle>
    <bundle>...</bundle>
  </feature>
  <feature name="feature2" version="1.1.0">
    <feature>feature1</feature>
    <bundle>...</bundle>
  </feature>
</features>
```

機能 XML にはスキーマがあることに注意してください。詳細は、ユーザーガイドの [Features XML Schema section|provisioning-schema] を参照してください。**feature1** 機能はバージョン **1.0.0** で使用でき、2 つのバンドルが含まれます。**<bundle/>** 要素には、バンドルアーティファクトへの URL が含まれます (詳細はアーティファクトリーポジトリーおよび URL セクションを参照) 。**feature1** 機能をインストールする場合 (後で説明するように **feature:install** または FeatureMBean を使用)、Apache Karaf によって自動的に記述された 2 つのバンドルがインストールされます。**feature2** 機能はバージョン **1.1.0** で利用でき、**feature1** 機能およびバンドルへの参照が含まれます。**<feature/>** 要素には機能の名前が含まれます。特定の機能バージョンは、**<feature/>** 要素 (**<feature version="1.0.0">feature1</feature>**) の **version** 属性を使用して定義できます。**version** 属性が指定されていない場合、Apache Karaf は利用可能な最新バージョンをインストールします。**feature2** 機能をインストールする場合 (後で説明するように **feature:install** または FeatureMBean を使用)、Apache Karaf は **feature1** (すでにインストールされていない場合) およびバンドルを自動的にインストールします。

機能リポジトリーは、機能 XML ファイルへの URL を使用して登録されます。

機能の状態は Apache Karaf キャッシュ (**KARAF_DATA** フォルダー内) に保存されます。Apache Karaf を再起動できます。以前にインストールされた機能はインストールされたままで、再起動後も使用できます。クリーンな再起動を行ったり、Apache Karaf キャッシュを削除する場合 (**KARAF_DATA** フォル

ダーを削除する)、以前に登録された features リポジトリーとインストールされた機能はすべて失われます。手作業で features リポジトリーを再度登録し、機能を再度インストールする必要があります。この動作を防ぐために、機能をブート機能として指定できます。

16.10.5. ブート機能

一部の機能は、ブート機能として記述できます。**feature:install** または FeatureMBean を使用して事前にインストールされていない場合でも、ブート機能は Apache Karaf によって自動的にインストールされます。

Apache Karaf の機能設定は **etc/org.apache.karaf.features.cfg** 設定ファイルにあります。

この設定ファイルには、ブート機能の定義に使用する 2 つのプロパティーが含まれています。

- **featuresRepositories** には features リポジトリー (features XML) URL のリスト (コンマ区切り) が含まれます。
- **featuresBoot** には起動時にインストールする機能の一覧 (コンマ区切り) が含まれます。

16.10.6. 機能のアップグレード

同じ機能を (同じ SNAPSHOT バージョンまたは異なるバージョンで) インストールすることにより、リリースを更新できます。

機能のライフサイクルがあると、機能のステータス (開始、停止など) を制御できます。

シミュレーションを使用して更新の動作を確認することもできます。

16.10.7. オーバーライド

機能で定義されたバンドルは、ファイル etc/overrides.properties を使用してオーバーライドできます。ファイル内の各行で、オーバーライドを 1 つ定義します。構文は <bundle-uri>[;range="[min,max)"] です。上書きするバージョンが上書きされるバンドルのバージョンよりも大きく、範囲が同じ場合には、指定のバンドルは、シンボリック名が同じ機能定義にあるすべてのバンドルを上書きます。範囲が指定されていない場合には、マイクロバージョンレベルでの互換性を想定します。

たとえば、mvn:org.ops4j.pax.logging/pax-logging-service/1.8.5 の上書きは、1.8.6 または 1.7.0 ではなく、pax-logging-service 1.8.3 を上書きします。

16.10.8. 機能バンドル

16.10.8.1. 開始レベル

デフォルトでは、機能によってデプロイされるバンドルの開始レベルは、**karaf.startlevel.bundle** プロパティーの **etc/config.properties** 設定ファイルで定義された値と等しくなります。

この値を、features XML の **<bundle/>** 要素の **start-level** 属性で上書きすることができます。

```
<feature name="my-project" version="1.0.0">
  <bundle start-level="80">mvn:com.mycompany.myproject/myproject-dao</bundle>
  <bundle start-level="85">mvn:com.mycompany.myproject/myproject-service</bundle>
</feature>
```

start-level 属性は、それを使用するバンドルの前に **myproject-dao** バンドルが開始されるようにします。

開始レベルを使用するのではなく、単純に必要なパッケージやサービスを定義して、OSGi フレームワークに依存関係を認識させるものが、ソリューションとしてより優れています。また、開始レベルを設定するよりも堅牢です。

16.10.8.2. シミュレーション、起動、および停止

-t オプションを **feature:install** コマンドに使用して、機能のインストールをシミュレートできます。

バンドルは、起動せずにインストールできます。デフォルトでは、機能のバンドルは自動的に開始されます。

機能は、バンドルが自動的に開始されないように指定できます (バンドルは解決された状態のままになります)。これを行うために、機能は **<bundle/>** 要素で **start** 属性を false に指定できます。

```
<feature name="my-project" version="1.0.0">
  <bundle start-level="80" start="false">mvn:com.mycompany.myproject/myproject-dao</bundle>
  <bundle start-level="85" start="false">mvn:com.mycompany.myproject/myproject-service</bundle>
</feature>
```

16.10.8.3. 依存関係

<bundle/> 要素で **dependency** 属性を true に設定して、バンドルに依存関係としてフラグを付けることができます。

リゾルバーでこの情報を使用して、インストールするバンドルの完全リストを算出することができます。

16.10.9. 依存機能

機能は、他の機能のセットに依存する場合があります。

```
<feature name="my-project" version="1.0.0">
  <feature>other</feature>
  <bundle start-level="80" start="false">mvn:com.mycompany.myproject/myproject-dao</bundle>
  <bundle start-level="85" start="false">mvn:com.mycompany.myproject/myproject-service</bundle>
</feature>
```

other 機能をインストールすると、**my-project** 機能も自動的にインストールされます。

依存機能のバージョン範囲を定義できます。

```
<feature name="spring-dm">
  <feature version="[2.5.6,4)">spring</feature>
  ...
</feature>
```

範囲内で利用可能な最高バージョンの機能がインストールされます。

1 つのバージョンを指定すると、範囲はそのバージョン以降とみなされます。

何も指定されていない場合は、利用可能な最大値がインストールされます。

正確なバージョンを指定するには、**[3.1,3.1]** のように範囲を指定します。

16.10.9.1. 機能の前提条件

必要な機能は、特別な種類の依存関係です。**prerequisite** 属性を依存する機能タグに追加する場合、実際の機能をインストールする前に、依存機能でのバンドルのインストールとアクティベーションが強制されます。これは、特定の機能に登録されたバンドルが **wrap**、**war** などの事前インストールされた URL を使用しない場合に便利です。

16.10.10. 機能設定

features XML の **<config/>** 要素を使用すると、機能は設定を作成したり入力したりできます (設定 PID によって識別)。

```
<config name="com.foo.bar">
  myProperty = myValue
</config>
```

<config/> 要素の **name** 属性は、設定 PID に対応します (詳細は設定セクションを参照してください)。

この機能のインストールは、**com.foo.bar.cfg** という名前のファイルを **etc** フォルダーにドロップするのと同じ効果があります。

<config/> 要素の内容は、key=value 標準に従ったプロパティーのセットになります。

16.10.11. 機能設定ファイル

この機能は、**<config/>** 要素を使用する代わりに **<configfile/>** 要素を指定できます。

```
<configfile finalname="/etc/myfile.cfg" override="false">URL</configfile>
```

(**<config/>** 要素を使用する場合のように) Apache Karaf 設定レイヤーを直接操作する代わりに、**<configfile/>** 要素は URL で指定されたファイルを直接取得し、 **finalname** 属性で指定された場所にファイルをコピーします。

指定しない場合、場所は **KARAF_BASE** 変数からの相対位置になります。${karaf.home}、${karaf.base}、${karaf.etc} などの変数やシステムプロパティーを使用することもできます。

たとえば、以下のようになります。

```
<configfile finalname="${karaf.etc}/myfile.cfg" override="false">URL</configfile>
```

ファイルが任意の場所にすでにあった場合には、その既存のファイルにカスタマイズが含まれる可能性があるので、ファイルはそのまま保持されて、設定ファイルのデプロイメントがスキップされます。この動作は、**override** を true に設定してオーバーライドできます。

ファイル URL は、Apache Karaf でサポートされている任意の URL です (詳細は、ユーザーガイドの [Artifacts repositories and URLs|urls] を参照してください)。

16.10.11.1. 要件

機能では、想定される要件を指定することもできます。機能リゾルバーは、要件を満たそうとします。そのために、機能とバンドル機能をチェックし、要件を満たすためにバンドルを自動的にインストールします。

たとえば、機能には次のものを含めることができます。

```
<requirement>osgi.ee;filter:=&quot;(&amp;(osgi.ee=JavaSE)(!(version&gt;=1.8)))&quot;
</requirement>
```

この要件では、JDK のバージョンが 1.8 でない場合にのみ (基本的に 1.7)、機能が動作することを指定しています。

機能リゾルバーは、オプションの依存関係が満たされたときにバンドルを更新して、オプションのインポートを再設定することもできます。

16.10.12. コマンド

16.10.12.1. feature:repo-list

feature:repo-list コマンドは、登録されたすべての features リポジトリーを一覧表示します。

```
karaf@root()> feature:repo-list
Repository               | URL
--------------------------------------------------------------------------------------
org.ops4j.pax.cdi-0.12.0 | mvn:org.ops4j.pax.cdi/pax-cdi-features/0.12.0/xml/features
org.ops4j.pax.web-4.1.4  | mvn:org.ops4j.pax.web/pax-web-features/4.1.4/xml/features
standard-4.0.0           | mvn:org.apache.karaf.features/standard/4.0.0/xml/features
enterprise-4.0.0         | mvn:org.apache.karaf.features/enterprise/4.0.0/xml/features
spring-4.0.0             | mvn:org.apache.karaf.features/spring/4.0.0/xml/features
```

各リポジトリーには、機能 XML への名前と URL があります。

Apache Karaf は、features リポジトリーの URL を登録する際に機能 XML を解析します (後ほど説明する **feature:repo-add** コマンドまたは FeatureMBean を使用します)。Apache Karaf が features リポジトリーの URL をリロードするよう強制する場合 (たとえば機能定義を更新)、**-r** オプションを使用できます。

```
karaf@root()> feature:repo-list -r
Reloading all repositories from their urls

Repository               | URL
--------------------------------------------------------------------------------------
org.ops4j.pax.cdi-0.12.0 | mvn:org.ops4j.pax.cdi/pax-cdi-features/0.12.0/xml/features
org.ops4j.pax.web-4.1.4  | mvn:org.ops4j.pax.web/pax-web-features/4.1.4/xml/features
standard-4.0.0           | mvn:org.apache.karaf.features/standard/4.0.0/xml/features
enterprise-4.0.0         | mvn:org.apache.karaf.features/enterprise/4.0.0/xml/features
spring-4.0.0             | mvn:org.apache.karaf.features/spring/4.0.0/xml/features
```

16.10.12.2. feature:repo-add

features リポジトリーを登録する (Apache Karaf で新しい機能を利用できるようにする) には、**feature:repo-add** コマンドを使用する必要があります。

feature:repo-add コマンドには、**name/url** 引数が必要です。この引数では、以下が可能です。

- 機能リポジトリーの URL。これは、機能 XML ファイルへの直接の URL です。ユーザーガイドの [Artifacts repositories and URLs section|urls] に記載されている URL はすべてサポートされます。
- **etc/org.apache.karaf.features.repos.cfg** 設定ファイルで定義されている機能リポジトリー名。

etc/org.apache.karaf.features.repos.cfg は、プリインストールされた/利用可能な機能リポジトリーのリストを定義します。

```
################################################################################
#
#    Licensed to the Apache Software Foundation (ASF) under one or more
#    contributor license agreements.  See the NOTICE file distributed with
#    this work for additional information regarding copyright ownership.
#    The ASF licenses this file to You under the Apache License, Version 2.0
#    (the "License"); you may not use this file except in compliance with
#    the License.  You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to in writing, software
#    distributed under the License is distributed on an "AS IS" BASIS,
#    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#    See the License for the specific language governing permissions and
#    limitations under the License.
#
################################################################################

#
# This file describes the features repository URL
# It could be directly installed using feature:repo-add command
#
enterprise=mvn:org.apache.karaf.features/enterprise/LATEST/xml/features
spring=mvn:org.apache.karaf.features/spring/LATEST/xml/features
cellar=mvn:org.apache.karaf.cellar/apache-karaf-cellar/LATEST/xml/features
cave=mvn:org.apache.karaf.cave/apache-karaf-cave/LATEST/xml/features
camel=mvn:org.apache.camel.karaf/apache-camel/LATEST/xml/features
camel-extras=mvn:org.apache-extras.camel-extra.karaf/camel-extra/LATEST/xml/features
cxf=mvn:org.apache.cxf.karaf/apache-cxf/LATEST/xml/features
cxf-dosgi=mvn:org.apache.cxf.dosgi/cxf-dosgi/LATEST/xml/features
cxf-xkms=mvn:org.apache.cxf.services.xkms/cxf-services-xkms-features/LATEST/xml
activemq=mvn:org.apache.activemq/activemq-karaf/LATEST/xml/features
jclouds=mvn:org.apache.jclouds.karaf/jclouds-karaf/LATEST/xml/features
openejb=mvn:org.apache.openejb/openejb-feature/LATEST/xml/features
wicket=mvn:org.ops4j.pax.wicket/features/LATEST/xml/features
hawtio=mvn:io.hawt/hawtio-karaf/LATEST/xml/features
pax-cdi=mvn:org.ops4j.pax.cdi/pax-cdi-features/LATEST/xml/features
pax-jdbc=mvn:org.ops4j.pax.jdbc/pax-jdbc-features/LATEST/xml/features
pax-jpa=mvn:org.ops4j.pax.jpa/pax-jpa-features/LATEST/xml/features
pax-web=mvn:org.ops4j.pax.web/pax-web-features/LATEST/xml/features
pax-wicket=mvn:org.ops4j.pax.wicket/pax-wicket-features/LATEST/xml/features
ecf=http://download.eclipse.org/rt/ecf/latest/site.p2/karaf-features.xml
decanter=mvn:org.apache.karaf.decanter/apache-karaf-decanter/LATEST/xml/features
```

機能のリポジトリー名は、**feature:repo-add** コマンドに直接指定できます。PAX JDBC をインストールするには、以下を実行できます。

```
karaf@root()> feature:repo-add pax-jdbc
Adding feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/LATEST/xml/features
```

オプションの **version** 引数を指定しないと、Apache Karaf は利用可能な features リポジトリーの最新バージョンをインストールします。**version** 引数でターゲットバージョンを指定できます。

```
karaf@root()> feature:repo-add pax-jdbc 1.3.0
Adding feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
```

etc/org.apache.karaf.features.repos.cfg 設定ファイルで定義された features リポジトリー名を提供する代わりに、features リポジトリーの URL を **feature:repo-add** コマンドに直接指定できます。

```
karaf@root()> feature:repo-add mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
Adding feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
```

デフォルトでは、**feature:repo-add** コマンドは機能リポジトリーを登録するだけで、機能はインストールされません。**-i** オプションを指定すると、**feature:repo-add** コマンドは features リポジトリーを登録し、この features リポジトリーに記述されているすべての機能をインストールします。

```
karaf@root()> feature:repo-add -i pax-jdbc
```

16.10.12.3. feature:repo-refresh

Apache Karaf は、登録時に features リポジトリー XML を解析します (**feature:repo-add** コマンドまたは FeatureMBean を使用)。機能リポジトリーの XML が変更された場合は、変更をロードするために機能リポジトリーを更新するように Apache Karaf に指示する必要があります。

feature:repo-refresh コマンドは、機能リポジトリーを更新します。

引数がない場合に、コマンドはすべての機能リポジトリーを更新します。

```
karaf@root()> feature:repo-refresh
Refreshing feature url mvn:org.ops4j.pax.cdi/pax-cdi-features/0.12.0/xml/features
Refreshing feature url mvn:org.ops4j.pax.web/pax-web-features/4.1.4/xml/features
Refreshing feature url mvn:org.apache.karaf.features/standard/4.0.0/xml/features
Refreshing feature url mvn:org.apache.karaf.features/enterprise/4.0.0/xml/features
Refreshing feature url mvn:org.apache.karaf.features/spring/4.0.0/xml/features
```

すべての機能リポジトリーを更新する代わりに、URL または機能リポジトリー名 (およびオプションでバージョン) を指定して、更新する機能リポジトリーを指定できます。

```
karaf@root()> feature:repo-refresh mvn:org.apache.karaf.features/standard/4.0.0/xml/features
Refreshing feature url mvn:org.apache.karaf.features/standard/4.0.0/xml/features
```

```
karaf@root()> feature:repo-refresh pax-jdbc
Refreshing feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/LATEST/xml/features
```

16.10.12.4. feature:repo-remove

feature:repo-remove コマンドは、登録した features リポジトリーから features リポジトリーを削除します。

feature:repo-remove コマンドには引数が必要です。

- features リポジトリー名 (**feature:repo-list** コマンド出力のリポジトリーの列に表示)
- features リポジトリー URL (**feature:repo-list** コマンド出力の URL コラムに表示)

```
karaf@root()> feature:repo-remove org.ops4j.pax.jdbc-1.3.0
```

```
karaf@root()> feature:repo-remove mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
```

デフォルトでは、**feature:repo-remove** コマンドは、登録した機能から features リポジトリーを削除します。これは、features リポジトリーによって提供される機能をアンインストールしません。

-u オプションを使用すると、**feature:repo-remove** コマンドは features リポジトリーによって記述されたすべての機能をアンインストールします。

```
karaf@root()> feature:repo-remove -u org.ops4j.pax.jdbc-1.3.0
```

16.10.12.5. feature:list

feature:list コマンドは、利用可能なすべての機能を一覧表示します (登録された異なる features リポジトリーによって提供) 。

```
Name                    | Version                    | Required | State       | Repository              |
Description
------------------------------------------------------------------------------------------------------------------------
---------------------------------------------
pax-cdi                 | 0.12.0                     |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Provide
CDI support
pax-cdi-1.1             | 0.12.0                     |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.1 support
pax-cdi-1.2             | 0.12.0                     |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.2 support
pax-cdi-weld            | 0.12.0                     |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Weld
CDI support
pax-cdi-1.1-weld        | 0.12.0                     |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld CDI 1.1 support
pax-cdi-1.2-weld        | 0.12.0                     |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld CDI 1.2 support
pax-cdi-openwebbeans    | 0.12.0                     |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
OpenWebBeans CDI support
pax-cdi-web             | 0.12.0                     |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Web
CDI support
pax-cdi-1.1-web         | 0.12.0                     |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Web
CDI 1.1 support
...
```

名前をアルファベット順に並べたい場合は、**-o** オプションを使用できます。

```
karaf@root()> feature:list -o
Name                    | Version                    | Required | State       | Repository              |
```

```
Description
---------------------------------------------------------------------------------------------------------------------
-------------------------------------------
deltaspike-core             | 1.2.1                         |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike core support
deltaspike-data             | 1.2.1                         |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike data support
deltaspike-jpa              | 1.2.1                         |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike jpa support
deltaspike-partial-bean     | 1.2.1                         |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike partial bean support
pax-cdi                     | 0.12.0                        |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Provide
CDI support
pax-cdi-1.1                 | 0.12.0                        |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.1 support
pax-cdi-1.1-web             | 0.12.0                        |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Web
CDI 1.1 support
pax-cdi-1.1-web-weld        | 0.12.0                        |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld Web CDI 1.1 support
pax-cdi-1.1-weld            | 0.12.0                        |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld CDI 1.1 support
pax-cdi-1.2                 | 0.12.0                        |           | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.2 support
...
```

デフォルトでは、**feature:list** コマンドは現在の状態 (インストール済みまたはインストールされていない) に関わらず、すべての機能を表示します。

-i オプションを使用すると、インストールされた機能だけが表示されます。

```
karaf@root()> feature:list -i
Name            | Version | Required | State   | Repository     | Description
-------------------------------------------------------------------------------------------------------------
aries-proxy     | 4.0.0   |          | Started | standard-4.0.0 | Aries Proxy
aries-blueprint | 4.0.0   | x        | Started | standard-4.0.0 | Aries Blueprint
feature         | 4.0.0   | x        | Started | standard-4.0.0 | Features Support
shell           | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Shell
shell-compat    | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Shell Compatibility
deployer        | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Deployer
bundle          | 4.0.0   | x        | Started | standard-4.0.0 | Provide Bundle support
config          | 4.0.0   | x        | Started | standard-4.0.0 | Provide OSGi ConfigAdmin support
diagnostic      | 4.0.0   | x        | Started | standard-4.0.0 | Provide Diagnostic support
instance        | 4.0.0   | x        | Started | standard-4.0.0 | Provide Instance support
jaas            | 4.0.0   | x        | Started | standard-4.0.0 | Provide JAAS support
log             | 4.0.0   | x        | Started | standard-4.0.0 | Provide Log support
package         | 4.0.0   | x        | Started | standard-4.0.0 | Package commands and mbeans
service         | 4.0.0   | x        | Started | standard-4.0.0 | Provide Service support
system          | 4.0.0   | x        | Started | standard-4.0.0 | Provide System support
kar             | 4.0.0   | x        | Started | standard-4.0.0 | Provide KAR (KARaf archive) support
ssh             | 4.0.0   | x        | Started | standard-4.0.0 | Provide a SSHd server on Karaf
management      | 4.0.0   | x        | Started | standard-4.0.0 | Provide a JMX MBeanServer and a set of
MBeans in
wrap            | 0.0.0   | x        | Started | standard-4.0.0 | Wrap URL handler
```

16.10.12.6. feature:install

feature:install コマンドは、機能をインストールします。

feature 引数が必要です。**feature** 引数は、機能の名前または機能の名前/バージョンです。(バージョンではなく) 機能の名前のみが提供されている場合に、使用可能な最新バージョンがインストールされます。

```
karaf@root()> feature:install eventadmin
```

-t または **--simulate** オプションを使用してインストールをシミュレートできます。これは、何が行われるかを表示するだけで、実行はされません。

```
karaf@root()> feature:install -t -v eventadmin
Adding features: eventadmin/[4.0.0,4.0.0]
No deployment change.
  Managing bundle:
    org.apache.felix.metatype / 1.0.12
```

インストールする機能のバージョンを指定できます。

```
karaf@root()> feature:install eventadmin/4.0.0
```

デフォルトでは、**feature:install** コマンドは詳細ではありません。**feature:install** コマンドによって実行されるアクションの詳細を知りたい場合は、**-v** オプションを使用できます。

```
karaf@root()> feature:install -v eventadmin
Adding features: eventadmin/[4.0.0,4.0.0]
No deployment change.
Done.
```

インストール済みのバンドルが機能に含まれている場合に、デフォルトでは、Apache Karaf はこのバンドルを更新します。この更新が原因で、実行中の他のアプリケーションに問題が発生する場合があります。インストールされたバンドルの自動更新を無効にする場合は、**-r** オプションを使用できます。

```
karaf@root()> feature:install -v -r eventadmin
Adding features: eventadmin/[4.0.0,4.0.0]
No deployment change.
Done.
```

-s または **--no-auto-start** オプションを使用して、機能によってインストールされたバンドルを開始しないよう指定できます。

```
karaf@root()> feature:install -s eventadmin
```

16.10.12.7. feature:start

デフォルトでは、機能をインストールすると、その機能は自動的にインストールされます。ただし、**-s** オプションを **feature:install** コマンドに指定できます。

機能をインストールすると (開始されているかどうかに関係なく)、機能で定義されているバンドルによって提供されるすべてのパッケージが使用可能になり、他のバンドルの設定に使用できるようになります。

機能を開始すると、すべてのバンドルが開始されるため、機能はサービスも公開します。

16.10.12.8. feature:stop

機能を停止することもできるので、機能によって提供されるすべてのサービスが停止され、サービスレジストリーから削除されます。ただし、パッケージはワイヤリングで引き続き利用できます (バンドルは解決済み状態です)。

16.10.12.9. feature:uninstall

feature:uninstall コマンドは、機能をアンインストールします。**feature:install** コマンドとして、**feature:uninstall** コマンドには **feature** 引数が必要です。**feature** 引数は、機能の名前または機能の名前/バージョンです。(バージョンではなく) 機能の名前のみが提供されている場合に、使用可能な最新バージョンがインストールされます。

```
karaf@root()> feature:uninstall eventadmin
```

機能リゾルバーは、機能のアンインストール中に使用されます。アンインストールされた機能でインストールされた一時的な機能は、他の機能によって使用されていない場合には、自分でアンインストールできます。

16.10.13. Deployer

deploy フォルダーに直接ファイルをドロップすることで features XML をホットデプロイできます。

Apache Karaf には、機能デプロイヤーが同梱されています。

デプロイフォルダーに features XML をドロップすると、機能デプロイヤー次を行います: * features XML を features リポジトリーとして登録。* **install** 属性が auto に設定された機能は機能デプロイヤーによって自動的にインストールされます。

たとえば、次の XML をデプロイフォルダーにドロップすると、feature1 と feature2 が自動的にインストールされますが、feature3 はインストールされません。

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="my-features" xmlns="http://karaf.apache.org/xmlns/features/v1.3.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://karaf.apache.org/xmlns/features/v1.3.0
http://karaf.apache.org/xmlns/features/v1.3.0">

    <feature name="feature1" version="1.0" install="auto">
        ...
    </feature>

    <feature name="feature2" version="1.0" install="auto">
        ...
    </feature>

    <feature name="feature3" version="1.0">
        ...
    </feature>

</features>
```

16.10.14. JMX FeatureMBean

JMX レイヤーには、機能および機能リポジトリーの管理専用の MBean である Feature MBean があります。

FeatureMBean オブジェクト名は **org.apache.karaf:type=feature,name=*** です。

16.10.14.1. 属性

FeatureMBean は、次の 2 つの属性を提供します。

- **Features** は、利用可能なすべての機能の表形式のデータセットです。
- **Repositories** は、すべての登録済み features リポジトリーの表形式のデータセットです。

Repositories 属性は、以下の情報を提供します。

- **Name** は features リポジトリーの名前です。
- **Uri** はこのリポジトリーの機能 XML に対する URI です。
- **Features** は、この feature リポジトリーによって提供されるすべての機能 (名前およびバージョン) の表形式のデータセットです。
- **Repositories** は、この feature リポジトリーにインポートされた features リポジトリーの表形式のデータセットです。

Features 属性は、以下の情報を提供します。

- **Name** は、機能の名前です。
- **Version** は、機能のバージョンです。
- **Installed** は、ブール値です。true の場合、機能が現在インストールされていることを意味します。
- **Bundles** は、機能に記述されている全バンドル (バンドル URL) の表形式のデータセットです。
- **Configurations** は、機能に記述されているすべての設定の表形式のデータセットです。
- **Configuration Files** は、機能に記述されているすべての設定ファイルの表形式のデータセットです。
- **Dependencies** は、機能に記述されているすべての依存する機能の表形式のデータセットです。

16.10.14.2. 操作

- **addRepository(url)** は、**url** で features リポジトリーを追加します。**url** は、**feature:repo-add** コマンドのように **name** にすることができます。
- **addRepository(url, install)** は **url** で features リポジトリーを追加し、**install** が true の場合はすべてのバンドルを自動的にインストールします。**url** は、**feature:repo-add** コマンドのように **name** にすることができます。
- **removeRepository(url)** は、**url** で features リポジトリーを削除します。**url** は、**feature:repo-remove** コマンドのように **name** にすることができます。

- **installFeature(name)** は、**name** の feature をインストールします。
- **installFeature(name, version)** は、**name** および **version** の feature をインストールします。
- **installFeature(name, noClean, noRefresh)** は、障害発生時にバンドルを消去せず、バンドルをクリーニングせずに **name** で機能をインストールします。
- **installFeature(name, version, noClean, noRefresh) ` は、障害発生時にバンドルを消去せず、バンドルをクリーニングせずに `name** および **version** で機能をインストールします。
- **uninstallFeature(name)** は、**name** の feature をアンインストールします。
- **uninstallFeature(name, version)** は、**name** および **version** の feature をアンインストールします。

16.10.14.3. 通知

FeatureMBean は、次の 2 種類の通知を送信します (サブスクライブおよび対応)。

- features レポジトリーの変更時 (追加または削除)
- 機能の変更時 (インストールまたはアンインストール)。

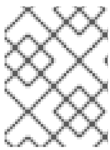
第17章 リモート接続を使用したコンテナーの管理

ローカルのコンソールを使用したコンテナーの管理は、論理的でない場合があります。Red Hat Fuse でのコンテナーのリモート管理にはさまざまな方法があります。リモートコンテナーのコマンドコンソールを使用するか、リモートクライアントを起動します。

17.1. リモートアクセスのコンテナーの設定

17.1.1. 概要

Red Hat Fuse ランタイムをデフォルトモードまたは「サーバーモードでのランタイムの起動」で起動すると、他の Fuse コンソールから SSH 経由でアクセス可能なリモートコンソールが有効になります。リモートコンソールには、ローカルコンソールのすべての機能が含まれており、リモートユーザーがコンテナー内で実行されているコンテナーおよびサービスを完全に制御できます。



注記

「クライアントモードでのランタイムの起動」で実行すると、Fuse ランタイムはリモートコンソールを無効にします。

17.1.2. リモートアクセスのためのスタンドアロンコンテナーの設定

SSH ホスト名とポート番号は **INSTALL_DIR/etc/org.apache.karaf.shell.cfg** 設定ファイルで設定されます。リモートアクセスのポート変更 は、8102 で使用するポートを変更するサンプル設定を示しています。

リモートアクセスのポート変更

```
sshPort=8102
sshHost=0.0.0.0
```

17.2. リモートによる接続および切断

代わりとなる、リモートコンテナーへの接続方法は 2 つあります。Red Hat Fuse コマンドシェルをすでに実行している場合は、コンソールコマンドを呼び出してリモートコンテナーに接続できます。また、コマンドラインでユーティリティーを直接実行して、リモートコンテナーに接続できます。

17.2.1. リモートコンテナーからのスタンドアロンコンテナーへの接続

17.2.1.1. 概要

リモートコンテナーにアクセスするには、任意のコンテナーのコマンドコンソールを使用できます。SSH を使用すると、ローカルコンテナーのコンソールはリモートコンテナーに接続し、リモートコンテナーのコマンドコンソールとして機能します。

17.2.1.2. ssh:ssh コンソールコマンドの使用

ssh:ssh コンソールを使用して、リモートコンテナーのコンソールに接続します。

ssh:ssh コマンド構文

```
ssh:ssh -l username -P password -p port hostname
```

-l

リモートコンテナーへの接続に使用されるユーザー名。**admin** 権限を持つ有効な JAAS ログインクレデンシャルを使用します。

-P

リモートコンテナーへの接続に使用するパスワード。

-p

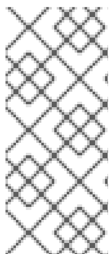
必要なコンテナーのリモートコンソールへのアクセスに使用される SSH ポート。デフォルト値は **8101** です。ポート番号の変更に関する詳細は、「リモートアクセスのためのスタンドアロンコンテナーの設定」を参照してください。

hostname

リモートコンテナーが実行されているマシンのホスト名。ホスト名の変更に関する詳細は、「リモートアクセスのためのスタンドアロンコンテナーの設定」を参照してください。

警告

etc/users.properties ファイルでユーザー名とパスワードをカスタマイズすることが推奨されます。



注記

リモートコンテナーが Oracle VM Server for SPARCインスタンスにデプロイされている場合は、デフォルトの SSH ポート値 **8101** はすでに論理ドメインマネージャーデーモンによって使用されている可能性が高くなります。この場合、「リモートアクセスのためのスタンドアロンコンテナーの設定」の説明に従って、コンテナーの SSH ポートを再設定する必要があります。

正しいコンテナーに接続していることを確認するには、Karaf コンソールプロンプトで **shell:info** を入力します。これにより、現在接続されているインスタンスに関する情報が返されます。

17.2.1.3. リモートコンソールからの切断

リモートコンソールから切断するには、**logout** を入力するか、プロンプトで **Ctrl**+**D** を押します。

リモートコンテナーから切断され、コンソールが再びローカルコンテナーを管理するようになります。

17.2.2. クライアントコマンドラインユーティリティーを使用したコンテナーへの接続

17.2.2.1. リモートクライアントの使用

リモートクライアントを使用すると、完全な Fuse コンテナーをローカルで起動せずに、リモート Red Hat Fuse コンテナーへセキュアに接続できます。

たとえば、同じマシンでサーバーモードで実行されている Fuse インスタンスに即座に接続するには、コマンドプロンプトを開いて、**client[.bat]** スクリプトを実行します (**InstallDir/bin** ディレクトリー内にあります)。

```
client
```

通常は、リモートインスタンスに接続するためのホスト名、ポート、ユーザー名、およびパスワードを指定します。テストスイートなど、大規模なスクリプト内でクライアントを使用している場合は、以下のようにコンソールコマンドを追加できます。

```
client -a 8101 -h hostname -u username -p password shell:info
```

または、**-p** オプションを省略した場合は、パスワードの入力を求められます。

スタンドアロンコンテナーの場合は、**admin** 権限を持つ有効な JAAS ユーザークレデンシャルを使用します。

クライアントで使用できるオプションを表示するには、以下を入力します。

```
client --help
```

Karaf クライアントのヘルプ

```
Apache Felix Karaf client
  -a [port]     specify the port to connect to
  -h [host]     specify the host to connect to
  -u [user]     specify the user name
  -p [password] specify the password
  --help        shows this help message
  -v            raise verbosity
  -r [attempts] retry connection establishment (up to attempts times)
  -d [delay]    intra-retry delay (defaults to 2 seconds)
  [commands]    commands to run
If no commands are specified, the client will be put in an interactive mode
```

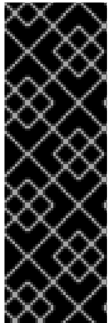
17.2.2.2. リモートクライアントのデフォルトの認証情報

クレデンシャルを指定しなくても、**bin/client** を使用して Karaf コンテナーにログインできることに驚くかもしれません。これは、リモートクライアントプログラムがデフォルトの認証情報を使用するように事前設定されているためです。認証情報を指定しないと、リモートクライアントは以下のデフォルト認証情報を (順番に) 自動的に使用するように試みます。

- **デフォルトの SSH キー**: デフォルトの Apache Karaf SSH キー を使用してログインしようとします。このログインが正常に行われるようにする設定エントリーは、**etc/keys.properties** ファイルでデフォルトでコメントアウトされています。

- **デフォルトのユーザー名/パスワードクレデンシャル** - ユーザー名とパスワードの組み合わせ **admin/admin** を使用してログインを試みます。このログインが正常に行われるようにする設定エントリーは、**etc/users.properties** ファイルでデフォルトでコメントアウトされています。

そのため、**users.properties** でデフォルトの **admin/admin** クレデンシャルをアンコメントして、Karaf コンテナーに新しいユーザーを作成する場合は、クレデンシャルを指定せずに **bin/client** ユーティリティーがログインできます。



重要

セキュリティー上の理由から、Karaf コンテナーの初回インストール時に (コメントアウトして)Fuse のデフォルトのクレデンシャルが無効になっています。ただし、デフォルトのパスワードや SSH 公開鍵を変更 **せず** にこれらのデフォルトの認証情報をアンコメントすると、Karaf コンテナーに、セキュリティーホールが発生します。実稼働環境では、これは実行**しない**でください。クレデンシャルを提供せずに **bin/client** を使用してコンテナーにログインできる場合、**コンテナーには安全ではないため、実稼働環境で修正手順を実行する必要があります。**

17.2.2.3. リモートクライアントコンソールからの切断

リモートクライアントを使用してリモートコンソールを開き、このコマンドを渡すのではなく、リモートクライアントを使用してリモートコンソールを切断する必要があります。リモートクライアントのコンソールから切断するには、**logout** を入力するか、プロンプトで **Ctrl-D** を押します。

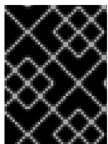
クライアントは切断され、終了します。

17.2.3. SSH コマンドラインユーティリティーを使用したコンテナーへの接続

17.2.3.1. 概要

また、**ssh** コマンドラインユーティリティー (UNIX 系オペレーティングシステムで標準のユーティリティー) を使用して Red Hat Fuse コンテナーにログインすることもできます。この認証メカニズムは、公開鍵の暗号化に基づいています (最初にコンテナーにインストールする必要があります)。たとえば、コンテナーが TCP ポート 8101 をリッスンするように設定されている場合には、以下のようにログインします。

```
ssh -p 8101 jdoe@localhost
```



重要

キーベースのログインは現在スタンドアロンコンテナーでのみサポートされ、Fabric コンテナーでは対応していません。

17.2.3.2. 前提条件

キーベースの SSH ログインを使用するには、以下の前提条件を満たす必要があります。

- このコンテナーは **PublickeyLoginModule** がインストールされているスタンドアロン (Fabric はサポートされません) である必要があります。
- SSH キーペアを作成しておく必要があります (「新しい SSH キーペアの作成」 参照)
- SSH キーペアからの公開鍵をコンテナーにインストールする必要があります (「コンテナーへの SSH 公開鍵のインストール」 を参照)。

17.2.3.3. デフォルトキーの場所

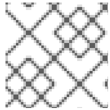
ssh コマンドは、デフォルトのキーの場所で秘密鍵を自動的に検索します。場所を明示的に指定する必要がなくなるので、鍵をデフォルトの場所にインストールすることが推奨されます。

*NIX オペレーティングシステムでは、RSA キーペアのデフォルトの場所は以下のようになります。

```
~/.ssh/id_rsa
~/.ssh/id_rsa.pub
```

Windows オペレーティングシステムでは、RSA キーペアのデフォルトの場所は次のとおりです。

```
C:\Documents and Settings\Username\.ssh\id_rsa
C:\Documents and Settings\Username\.ssh\id_rsa.pub
```



注記

Red Hat Fuse は RSA キーのみをサポートします。DSA キーは機能し **ません**。

17.2.3.4. 新しい SSH キーペアの作成

ssh-keygen ユーティリティーを使用して RSA 鍵ペアを生成します。新しいコマンドプロンプトを開き、以下のコマンドを入力します。

```
ssh-keygen -t rsa -b 2048
```

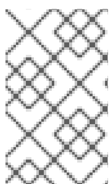
上記のコマンドは、鍵の長さ 2048 ビットを使用して RSA 鍵を生成します。次に、キーペアのファイル名を指定するように求められます。

```
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/Username/.ssh/id_rsa):
```

return を入力して、デフォルトの場所にあるキーペアを保存します。次に、パスフレーズの入力を求めるプロンプトが出されます。

```
Enter passphrase (empty for no passphrase):
```

必要に応じて、パスフレーズを入力するか、または 2 回入力してパスフレーズなしを選択できます。



注記

Fabric コンソールコマンドの実行に同じキーペアを使用する場合は、Fabric が暗号化された秘密鍵の使用をサポートしていないため、**パススルーのフレーズ** を選択しないことが推奨されます。

17.2.3.5. コンテナーへの SSH 公開鍵のインストール

Red Hat Fuse コンテナーにログインするときに SSH キーペアを使用するには、**INSTALL_DIR/etc/keys.properties** ファイルに新しいユーザーエントリーを作成し、コンテナーに SSH 公開鍵をインストールする必要があります。このファイルの各ユーザーエントリーは、以下の形式 1 行に表示されます。

```
Username=PublicKey,Role1,Role2,...
```

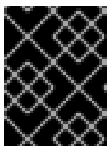
たとえば、公開鍵ファイル **~/.ssh/id_rsa.pub** に以下の内容があるとします。

```
ssh-rsa
AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH7WT2
NWPq/xfW6MPbLm1Vs14E7
```

```
gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnfqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1y
rv8iIDGZ3RSAHHAAAAFQCX
YFCPFSMLzLKSuYKi64QL8Fgc9QAAAnEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0Hgmd
RWVeOutRZT+ZxBxCBgLRJFnEj6Ewo
FhO3zwkyjMim4TwWeotifI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKL
Zl6Ae1UlZAFMO/7PSSoAAACB
AKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskl8wO63OzP/qLmcJM0+JbcRU/53Jj7uyk
31drV2qxhIOsLDC9dGCWj4
7Y7TyhPdXh/0dthTRBy6bqGtRPxGa7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx
jdoe@doemachine.local
```


以下のエントリーを 1 行で **InstallDir/etc/keys.properties** ファイルに追加することで、**admin** ロールで **jdoe** ユーザーを作成できます。


```
jdoe=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH
7WT2NWPq/xfW6MPbLm1Vs14E7
gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnfqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1y
rv8iIDGZ3RSAHHAAAAFQCX
YFCPFSMLzLKSuYKi64QL8Fgc9QAAAnEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0Hgmd
RWVeOutRZT+ZxBxCBgLRJFnEj6Ewo
FhO3zwkyjMim4TwWeotifI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKL
Zl6Ae1UlZAFMO/7PSSoAAACB
AKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskl8wO63OzP/qLmcJM0+JbcRU/53Jj7uyk
31drV2qxhIOsLDC9dGCWj4
7Y7TyhPdXh/0dthTRBy6bqGtRPxGa7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx,admin
```

重要

ここで、**id_rsa.pub** ファイルの内容をすべて挿入しないでください。公開鍵自体を表す記号のブロックだけを挿入します。

17.2.3.6. 公開鍵認証がサポートされていることの確認

コンテナーの起動後、以下のように **jaas:realms** コンソールコマンドを実行して、公開鍵認証がサポートされているかどうかを確認できます。

```
karaf@root()> jaas:realms
Index │ Realm Name │ Login Module Class Name
──────┼────────────┼──────────────────────────────────────────────────────
──────────-
1 │ karaf │ org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
2 │ karaf │ org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
3 │ karaf │ org.apache.karaf.jaas.modules.audit.FileAuditLoginModule
4 │ karaf │ org.apache.karaf.jaas.modules.audit.LogAuditLoginModule
5 │ karaf │ org.apache.karaf.jaas.modules.audit.EventAdminAuditLoginModule
karaf@root()>
```

PublickeyLoginModule がインストールされていることを確認する必要があります。この設定では、ユーザー名/パスワード認証情報または公開鍵の認証情報のいずれかを使用してコンテナーにログインできます。

17.2.3.7. ssh ロールの etc/keys.properties への追加

次の例に示すように、**etc/keys.properties** で定義されている **admingroup** には、**ssh** ロールが含まれている必要があります。

```
#
# For security reason, the default auto-signed key is disabled.
# The user guide describes how to generate/update the key.
#
#karaf=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIl
H7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E
+4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAAAAFQCXYFCPFSMLzLKSuYKi64QL8Fgc9
QAAAIEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6
EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuW
fBpKLZl6Ae1UlZAFMO/7PSSoAAACBAKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskl8
wO63OzP/qLmcJM0+JbcRU/53JjTuyk31drV2qxhIOsLDC9dGCWj47Y7TyhPdXh/0dthTRBy6bqGtRPxG
a7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh
```

ssh ロールが **admingroup** の定義に含まれていない場合は、**etc/keys.properties** を編集して、**ssh** ロールを追加する必要があります。

17.2.3.8. キーベースの SSH を使用したログイン

キーベースの SSH ユーティリティーを使用してコンテナーにログインする準備が整いました。以下に例を示します。

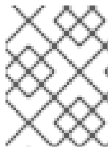
```
$ ssh -p 8101 jdoe@localhost
_____        _  _  _      _    ______
| _ \ ___  __| | | | | | __ _| |_  | ____|   _ ___  ___
| |_) / _ \/ _` | | |_| |/ _` | __| | |_ | | | / __|/ _ \
| _ <  __/ (_| | | _  | (_| | |_  | _|| |_| \__ \  __/
|_| \_\___|\__,_| |_| |_|\__,_|\__| |_|   \__,_|___/\___|

  Fuse (7.x.x.fuse-xxxxxx-redhat-xxxxx)
  http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181/hawtio to access the management console

Hit '<ctrl-d>' or 'shutdown' to shutdown Red Hat Fuse.

karaf@root()>
```

注記

暗号化された秘密鍵を使用している場合は、**ssh** ユーティリティーにより、パスフレーズの入力が求められます。

17.3. リモートコンテナーの停止

ssh:ssh コマンドまたはリモートクライアントを使用してリモートコンソールに接続している場合は、**osgi:shutdown** コマンドを使用してリモートインスタンスを停止できます。



注記

リモートコンソールで **Ctrl+D** を押してリモート接続を閉じ、ローカルシェルに戻ります。

第18章 MAVEN でのビルド

概要

Maven は、Apache Maven プロジェクトで利用できるオープンソースのビルドシステムです。本章では、基本的な Maven の概念の一部と、Red Hat Fuse と連携するように Maven を設定する方法を説明します。原則では、どのビルドシステムでも OSGi バンドルをビルドできます。ただし、Red Hat Fuse による十分なサポートがあるため、Maven の使用を強く推奨しています。

18.1. MAVEN ディレクトリー構造

18.1.1. 概要

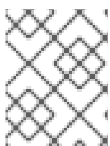
Maven ビルドシステムの最も重要な原則の 1 つは、Maven プロジェクト内のすべてのファイルに **標準的な場所** があることです。この原則には、いくつかの利点があります。利点の 1 つは、Maven プロジェクトに同一のディレクトリーレイアウトがあり、プロジェクトのファイルを簡単に見つけられることです。もう 1 つの利点は、Maven と統合されている各種ツールでは初期設定をほとんど必要とし **ない** ことです。たとえば、Java コンパイラーは、**src/main/java** 下のソースファイルをすべてコンパイルし、結果を **target/classes** に配置する必要があることを認識しています。

18.1.2. 標準のディレクトリーレイアウト

例18.1「標準の Maven ディレクトリーレイアウト」 は、OSGi バンドルプロジェクトのビルドに関連する標準の Maven ディレクトリーレイアウトの要素を示しています。さらに、Blueprint 設定ファイル (Maven によって定義されて **いない**) の標準の場所も紹介しています。

例18.1 標準の Maven ディレクトリーレイアウト

```
ProjectDir/
    pom.xml
    src/
        main/
            java/
                ...
            resources/
                META-INF/

                OSGI-INF/
                    blueprint/
                        *.xml
        test/
            java/
            resources/
    target/
        ...
```



注記

標準のディレクトリーのレイアウトを上書きすることは可能ですが、Maven では推奨され **ません**。

18.1.3. pom.xml ファイル

pom.xml ファイルは現在のプロジェクトのプロジェクトオブジェクトモデル (POM) です。これには、現在のプロジェクトのビルド方法に関する完全な説明が含まれています。**pom.xml** ファイルは完全に自己完結型にすることができますが、これは頻繁に (特に複雑な Maven プロジェクトの場合) **親 POM** ファイルから設定をインポートできます。

プロジェクトをビルドすると、生成された JAR ファイルの以下の場所に **pom.xml** ファイルのコピーが自動的に組み込まれます。

> META-INF/maven/**groupId**/**artifactId**/pom.xml

18.1.4. src とターゲットディレクトリー

src/ ディレクトリーには、プロジェクトの開発中に作業するすべてのコードおよびリソースファイルが含まれます。

target/ ディレクトリーには、ビルドの結果 (通常は JAR ファイル) と、ビルド中に生成されたすべての中間ファイルが含まれます。たとえば、ビルドの実行後、**target/classes/** ディレクトリーにはリソースファイルのコピーとコンパイルした Java クラスが含まれます。

18.1.5. メインおよびテストディレクトリー

src/main/ ディレクトリーには、アーティファクトのビルドに必要なすべてのコードおよびリソースが含まれます。

src/test/ ディレクトリーには、コンパイルされたアーティファクトに対して単体テストを実行するためのすべてのコードおよびリソースが含まれます。

18.1.6. Java ディレクトリー

各 **java/** サブディレクトリーには、標準の Java ディレクトリーレイアウト (ディレクトリーパス名が **.** の代わりに / で Java パッケージ名をミラーリングする) を持つ Java ソースコード (***.java** ファイル) が含まれます。**src/main/java/** ディレクトリーにはバンドルソースコードが含まれ、**src/test/java/** ディレクトリーには単体テストのソースコードが含まれます。

18.1.7. resources ディレクトリー

バンドルに含める設定ファイル、データファイル、または Java プロパティーがある場合は、これらを **src/main/resources/** ディレクトリー下に配置する必要があります。**src/main/resources/** 下のファイルおよびディレクトリーは、Maven ビルドプロセスによって生成される JAR ファイルのルートにコピーされます。

src/test/resources/ 下のファイルは、テストフェーズ中にのみ使用され、生成された JAR ファイルにコピー**されません**。

18.1.8. Blueprint コンテナー

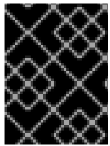
OSGi R4.2 は、Blueprint コンテナーを定義します。Red Hat Fuse には Blueprint コンテナーのサポートが組み込まれています。これは、プロジェクト内に **OSGI-INF/blueprint/*.xml** という Blueprint 設定ファイルを含むだけで有効にできます。Blueprint コンテナーの詳細は、12章*OSGi サービス* を参照してください。

18.2. APACHE KARAF の BOM ファイル

Maven BOM (Bill of Materials) ファイルの目的は、正常に動作する Maven 依存関係バージョンのセットを提供し、各 Maven アーティファクトに対して個別にバージョンを定義する必要をなくすことです。

Apache Karaf の Fuse BOM には以下の利点があります。

- Maven 依存関係のバージョンを定義するため、依存関係を POM に追加するときにバージョンを指定する必要がありません。
- 特定バージョンの Fuse に対して完全にテストされ、完全にサポートする依存関係のセットを定義します。
- Fuse のアップグレードを簡素化します。

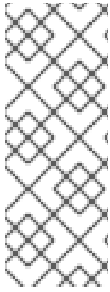


重要

Fuse BOM によって定義される依存関係のセットのみが Red Hat によってサポートされます。

Maven プロジェクトに Maven BOM ファイルを組み込むには、以下の例のように、プロジェクトの **pom.xml** ファイル (または親 POM ファイル内の) **dependencyManagement** 要素を指定します。

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- configure the versions you want to use here -->
    <fuse.version>7.8.0.fuse-sb2-780038-redhat-00001</fuse.version>

  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-karaf-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```



注記

org.jboss.redhat-fuse BOM は Fuse 7 の新機能で、BOM のバージョン管理を簡素化するように設計されています。しかし、新しい BOM を使用するためにリファクタリングされていないので、Fuse クイックスタートおよび Maven アーキタイプは古い BOM を使用します。どちらの BOM も正しく、Maven プロジェクトで使用することができます。今後の Fuse リリースでは、新しい BOM を使用するように、クイックスタートと Maven archetype がリファクタリングされます。

依存関係管理のメカニズムを使用して BOM を指定した後、アーティファクトのバージョンを指定**しなくても**、Maven 依存関係を POM に追加できるようになります。たとえば、 **camel-velocity** コンポーネントの依存関係を追加するには、以下の XML フラグメントを POM の **dependencies** 要素に追加します。

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
</dependency>
```

この依存関係の定義では、**version** 要素が省略されることに注意してください。

第19章 MAVEN INDEXER プラグイン

Maven プラグインがアーティファクトに対して Maven Central を迅速に検索できるようにするには、Maven Indexer プラグインが必要です。

Maven Indexer プラグインをデプロイするには、以下のコマンドを使用します。

前提条件

Maven Indexer プラグインをデプロイする前に、Installing on Apache Karaf の Preparing to Use Maven セクションの手順に従うようにしてください。

Maven Indexer プラグインのデプロイ

1. Karaf コンソールに移動し、以下のコマンドを実行して Maven Indexer プラグインをインストールします。

```
features:install hawtio-maven-indexer
```

2. 以下のコマンドを入力して Maven Indexer プラグインを設定します。

```
config:edit io.hawt.maven.indexer
config:proplist
config:propset repositories 'https://maven.oracle.com'
config:proplist
config:update
```

3. Maven Indexer プラグインがデプロイされるまで待ちます。これには数分の時間がかかる場合があります。以下のようなメッセージがログタブに表示されます。

INFO	org.apache.felix.fileinstall	Creating configuration from io.hawt.maven.indexer.cfg
INFO	io.fabric8.internal.ProfileServiceImpl	updateProfile: Profile[ver=1.0,id=fabric,atts={parents=karaf hawtio}]
INFO	io.fabric8.internal.ProfileServiceImpl	updateProfile: Profile[ver=1.0,id=fabric,atts={parents=karaf hawtio}]

Maven Indexer プラグインがデプロイされると、以下のコマンドを使用して追加の外部 Maven リポジトリーを Maven Indexer プラグイン設定に追加します。

```
config:edit io.hawt.maven.indexer
config:proplist
config:propset repositories external repository
config:proplist
config:update
```

19.1. LOG

Apache Karaf は、動的で強力なロギングシステムを提供します。

以下をサポートします。

- OSGi ログサービス
- Apache Log4j v1 および v2 フレームワーク
- Apache Commons Logging フレームワーク

- Logback フレームワーク
- SLF4J フレームワーク
- ネイティブ Java Util Logging フレームワーク

つまり、アプリケーションはすべてのロギングフレームワークを使用でき、Apache Karaf は中央のログシステムを使用してロガーやアペンダーなどを管理できます。

19.1.1. 設定ファイル

初期ログ設定は、**etc/org.ops4j.pax.logging.cfg** からロードされます。

このファイルは 標準の Log4j2 設定ファイル です。

さまざまな Log4j2 要素があります。

- loggers
- アペンダー
- レイアウト

独自の初期設定を直接ファイルに追加できます。

デフォルト設定は以下のとおりです。

```
#
#  Copyright 2005-2018 Red Hat, Inc.
#
#  Red Hat licenses this file to you under the Apache License, version
#  2.0 (the "License"); you may not use this file except in compliance
#  with the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
#  implied.  See the License for the specific language governing
#  permissions and limitations under the License.
#

#
# Internal Log4j2 configuration
#
log4j2.status = WARN
log4j2.verbose = false
log4j2.dest = out

#
# Common pattern layouts for appenders defined as reusable properties
# See https://logging.apache.org/log4j/2.x/manual/layouts.html#PatternLayout
# references will be replaced by felix.fileinstall
#
log4j2.pattern = %d{DEFAULT} | %-5.5p | %-20.20t | %-32.32c{1.} | %X{bundle.id} -
```

```
%X{bundle.name} - %X{bundle.version} | %m%n
#log4j2.pattern = %d{DEFAULT} %-5.5p {%t} [%C.%M()] (%F:%L) : %m%n

#
# Appenders configuration
#

# JDBC Appender
log4j2.appender.jdbc.type = JDBC
log4j2.appender.jdbc.name = JdbcAppender
log4j2.appender.jdbc.tableName = EVENTS
log4j2.appender.jdbc.cs.type = DataSource
log4j2.appender.jdbc.cs.lazy = true
log4j2.appender.jdbc.cs.jndiName = osgi:service/jdbc/logdb
log4j2.appender.jdbc.c1.type = Column
log4j2.appender.jdbc.c1.name = DATE
log4j2.appender.jdbc.c1.isEventTimestamp = true
log4j2.appender.jdbc.c2.type = Column
log4j2.appender.jdbc.c2.name = LEVEL
log4j2.appender.jdbc.c2.pattern = %level
log4j2.appender.jdbc.c2.isUnicode = false
log4j2.appender.jdbc.c3.type = Column
log4j2.appender.jdbc.c3.name = SOURCE
log4j2.appender.jdbc.c3.pattern = %logger
log4j2.appender.jdbc.c3.isUnicode = false
log4j2.appender.jdbc.c4.type = Column
log4j2.appender.jdbc.c4.name = THREAD_ID
log4j2.appender.jdbc.c4.pattern = %thread
log4j2.appender.jdbc.c4.isUnicode = false
log4j2.appender.jdbc.c5.type = Column
log4j2.appender.jdbc.c5.name = MESSAGE
log4j2.appender.jdbc.c5.pattern = %message
log4j2.appender.jdbc.c5.isUnicode = false


# Console appender not used by default (see log4j2.rootLogger.appenderRefs)
log4j2.appender.console.type = Console
log4j2.appender.console.name = Console
log4j2.appender.console.layout.type = PatternLayout
log4j2.appender.console.layout.pattern = ${log4j2.pattern}

# Rolling file appender
log4j2.appender.rolling.type = RollingRandomAccessFile
log4j2.appender.rolling.name = RollingFile
log4j2.appender.rolling.fileName = ${karaf.data}/log/fuse.log
log4j2.appender.rolling.filePattern = ${karaf.data}/log/fuse-%i.log.gz
# uncomment to not force a disk flush
#log4j2.appender.rolling.immediateFlush = false
log4j2.appender.rolling.append = true
log4j2.appender.rolling.layout.type = PatternLayout
log4j2.appender.rolling.layout.pattern = ${log4j2.pattern}
log4j2.appender.rolling.policies.type = Policies
log4j2.appender.rolling.policies.size.type = SizeBasedTriggeringPolicy
log4j2.appender.rolling.policies.size.size = 16MB
log4j2.appender.rolling.strategy.type = DefaultRolloverStrategy
log4j2.appender.rolling.strategy.max = 20
```

```
# Audit file appender
log4j2.appender.audit.type = RollingRandomAccessFile
log4j2.appender.audit.name = AuditRollingFile
log4j2.appender.audit.fileName = ${karaf.data}/security/audit.log
log4j2.appender.audit.filePattern = ${karaf.data}/security/audit.log.%i
log4j2.appender.audit.append = true
log4j2.appender.audit.layout.type = PatternLayout
log4j2.appender.audit.layout.pattern = ${log4j2.pattern}
log4j2.appender.audit.policies.type = Policies
log4j2.appender.audit.policies.size.type = SizeBasedTriggeringPolicy
log4j2.appender.audit.policies.size.size = 8MB

# OSGi appender
log4j2.appender.osgi.type = PaxOsgi
log4j2.appender.osgi.name = PaxOsgi
log4j2.appender.osgi.filter = *

#
# Loggers configuration
#

# Root logger
log4j2.rootLogger.level = INFO
log4j2.rootLogger.appenderRef.RollingFile.ref = RollingFile
log4j2.rootLogger.appenderRef.PaxOsgi.ref = PaxOsgi
log4j2.rootLogger.appenderRef.Console.ref = Console
log4j2.rootLogger.appenderRef.Console.filter.threshold.type = ThresholdFilter
log4j2.rootLogger.appenderRef.Console.filter.threshold.level = ${karaf.log.console:-OFF}
#log4j2.rootLogger.appenderRef.Sift.ref = Routing

# Spifly logger
log4j2.logger.spifly.name = org.apache.aries.spifly
log4j2.logger.spifly.level = WARN

# Security audit logger
log4j2.logger.audit.name = org.apache.karaf.jaas.modules.audit
log4j2.logger.audit.level = INFO
log4j2.logger.audit.additivity = false
log4j2.logger.audit.appenderRef.AuditRollingFile.ref = AuditRollingFile

# help with identification of maven-related problems with pax-url-aether
#log4j2.logger.aether.name = shaded.org.eclipse.aether
#log4j2.logger.aether.level = TRACE
#log4j2.logger.http-headers.name = shaded.org.apache.http.headers
#log4j2.logger.http-headers.level = DEBUG
#log4j2.logger.maven.name = org.ops4j.pax.url.mvn
#log4j2.logger.maven.level = TRACE
```

デフォルト設定では、**out** ファイルアペンダーを使用して、**INFO** ログレベルで **ROOT** ロガーを定義します。ログレベルは、Log4j2 の有効な値に変更できます。最も詳細度の低いものから高いものまで、TRACE、DEBUG、INFO、ERROR、または FATAL を指定できます。

OSGi アペンダー

osgi:* アペンダーは、ログメッセージを OSGi Log Service に送信する特別なアペンダーです。

stdout アペンダー

stdout コンソールアペンダーは事前設定されていますが、デフォルトでは有効になっていません。このアペンダーを使用すると、ログメッセージを標準出力に直接表示できます。サーバーモードで Apache Karaf を実行する予定 (コンソールなし) の場合は便利です。

有効にするには、**stdout** アペンダーを **rootLogger** に追加する必要があります。

```
log4j2.rootLogger=INFO, out, stdout, osgi:*
```

out アペンダー

out アペンダーは、デフォルトのアペンダーです。これは、1MB ログファイル 10 個を管理してローテーションするローリングファイルアペンダーです。ログファイルはデフォルトで **data/log/fuse.log** にあります。

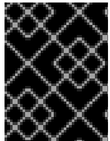
SIFT アペンダー

sift アペンダーは、デフォルトでは有効になっていません。このアペンダーを使用すると、デプロイされたバンドルごとに 1 つのログファイルを作成することができます。デフォルトでは、ログファイルの名前の形式は、バンドルのシンボリック名 (**data/log** フォルダーの) を使用します。このファイルはランタイム時に編集できます。Apache Karaf はファイルを再読み込みし、変更を有効にします。Apache Karaf を再起動する必要はありません。別の設定ファイルは Apache Karaf により使用されます (**etc/org.apache.karaf.log.cfg**)。このファイルは、ログコマンドによって使用されるログサービスを設定します (後で参照してください)。

JDBC アペンダー

jdbc アペンダーには **lazy** フラグがあり、**true** (**有効**) で、データソースが利用できない場合は、ロギングはデータベースに追加されません。ただし、jndi、データソース、または接続が戻ると、ロギングが再起動されます。

```
log4j2.appender.jdbc.cs.lazy = true
```



重要

ログメッセージが失われないようにするには、緊急アペンダーを設定することもできます。

19.1.2. コマンド

Apache Karaf は、**etc/org.ops4j.pax.logging.cfg** ファイルを変更する代わりに、ログ設定を動的に変更してログの内容を確認できる一連のコマンドを提供します。

19.1.2.1. log:clear

log:clear コマンドはログエントリーを消去します。

19.1.2.2. log:display

log:display コマンドは、ログエントリーを表示します。

デフォルトでは、**rootLogger** のログエントリーが表示されます。

```
karaf@root()> log:display
2015-07-01 19:12:46,208 | INFO  | FelixStartLevel  | SecurityUtils                    | 16 -
org.apache.sshd.core - 0.12.0 | BouncyCastle not registered, using the default JCE provider
```

```
2015-07-01 19:12:47,368 | INFO  | FelixStartLevel  | core                     | 68 -
org.apache.aries.jmx.core - 1.1.1 | Starting JMX OSGi agent
```

logger 引数を使用して、特定のロガーからログエントリーを表示することもできます。

```
karaf@root()> log:display ssh
2015-07-01 19:12:46,208 | INFO  | FelixStartLevel  | SecurityUtils                | 16 -
org.apache.sshd.core - 0.12.0 | BouncyCastle not registered, using the default JCE provider
```

デフォルトでは、すべてのログエントリーが表示されます。Apache Karaf コンテナーが長時間にわたる場合には、非常に長い時間がかかる可能性があります。**-n** オプションを使用して、表示するエントリー数を制限することができます。

```
karaf@root()> log:display -n 5
2015-07-01 06:53:24,143 | INFO  | JMX OSGi Agent   | core                     | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.framework.BundleStateMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=bundleState,version=1.7,framework=org.apache.felix.framework,uuid=5335370f-
9dee-449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,150 | INFO  | JMX OSGi Agent   | core                     | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.framework.PackageStateMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=packageState,version=1.5,framework=org.apache.felix.framework,uuid=5335370f-
9dee-449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,150 | INFO  | JMX OSGi Agent   | core                     | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.framework.ServiceStateMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=serviceState,version=1.7,framework=org.apache.felix.framework,uuid=5335370f-
9dee-449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,152 | INFO  | JMX OSGi Agent   | core                     | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering
org.osgi.jmx.framework.wiring.BundleWiringStateMBean to MBeanServer
com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=wiringState,version=1.1,framework=org.apache.felix.framework,uuid=5335370f-9dee-
449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,501 | INFO  | FelixStartLevel  | RegionsPersistenceImpl       | 78 -
org.apache.karaf.region.persist - 4.0.0 | Loading region digraph persistence
```

etc/org.apache.karaf.log.cfg ファイルの **size** プロパティーを使用すると、保存され保持されるエントリーの数を制限することもできます。

```
#
# The number of log statements to be displayed using log:display. It also defines the number
# of lines searched for exceptions using log:display exception. You can override this value
# at runtime using -n in log:display.
#
size = 500
```

デフォルトでは、各ログレベルは異なる色で表示されます。ERROR/FATAL は赤で、DEBUG in purple、INFO in cyan などになります。**--no-color** オプションを使用して色付けを無効にすることができます。

ログエントリーの形式パターンは、**etc/org.ops4j.pax.logging.cfg** ファイルに定義された変換パターンを使用しません。デフォルトでは、**etc/org.apache.karaf.log.cfg** で定義されている **pattern** プロパティーを使用します。

```
#
# The pattern used to format the log statement when using log:display. This pattern is according
# to the log4j2 layout. You can override this parameter at runtime using log:display with -p.
#
pattern = %d{ISO8601} | %-5.5p | %-16.16t | %-32.32c{1} | %X{bundle.id} - %X{bundle.name} -
%X{bundle.version} | %m%n
```

-p オプションを使用して、パターンを動的に変更することもできます (1 回の実行)。

```
karaf@root()> log:display -p "\%d - \%c - \%m\%n"
2015-07-01 07:01:58,007 - org.apache.sshd.common.util.SecurityUtils - BouncyCastle not registered,
using the default JCE provider
2015-07-01 07:01:58,725 - org.apache.aries.jmx.core - Starting JMX OSGi agent
2015-07-01 07:01:58,744 - org.apache.aries.jmx.core - Registering MBean with ObjectName
[osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=6361fc65-
8df4-4886-b0a6-479df2d61c83] for service with service.id [13]
2015-07-01 07:01:58,747 - org.apache.aries.jmx.core - Registering
org.osgi.jmx.service.cm.ConfigurationAdminMBean to MBeanServer
com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=6361fc65-
8df4-4886-b0a6-479df2d61c83
```

このパターンは通常の Log4j2 パターンで、日付に %d、クラスに %c、ログメッセージに %m などのキーワードを使用できます。

19.1.2.3. log:exception-display

log:exception-display コマンドは、最後に発生した例外を表示します。

log:display コマンドの場合、**log:exception-display** コマンドはデフォルトで **rootLogger** を使用しますが、**logger** 引数を使用してロガーを指定できます。

19.1.2.4. log:get

log:get コマンドは、ロガーの現在のログレベルを表示します。

デフォルトでは、表示されるログレベルはルートロガーからのログレベルです。

```
karaf@root()> log:get
Logger                          │ Level
────────────────────────────────┼──────
ROOT                            │ INFO
org.apache.aries.spifly         │ WARN
org.apache.karaf.jaas.modules.audit │ INFO
org.apache.sshd                 │ INFO
```

logger 引数を使用して、特定のロガーを指定できます。

```
karaf@root()> log:get ssh
INFO
```

logger 引数は、**ALL** キーワードを受け入れて、(リストとして) すべてのロガーのログレベルを表示します。

たとえば、**etc/org.ops4j.pax.logging.cfg** ファイルに独自のロガーを次のように定義します。

```
log4j2.logger.my.name = MyLogger
log4j2.logger.my.level = DEBUG
```

ロガーのリストは、対応するログレベルで確認できます。

```
karaf@root()> log:get ALL
Logger                              │ Level
────────────────────────────────────┼──────
MyLogger                            │ DEBUG
ROOT                                │ INFO
org.apache.aries.spifly             │ WARN
org.apache.karaf.jaas.modules.audit │ INFO
org.apache.sshd                     │ INFO
```

log:list コマンドは、**log:get ALL** のエイリアスです。

19.1.2.5. log:log

log:log コマンドを使用すると、ログにメッセージを手動で追加できます。Apache Karaf スクリプトを作成する場合には、以下が役立ちます。

```
karaf@root()> log:log "Hello World"
karaf@root()> log:display
12:55:21.706 INFO [pipe-log:log "Hello World"] Hello World
```

デフォルトでは、ログレベルは INFO ですが、**-l** オプションを使用して別のログレベルを指定できます。

```
karaf@root()> log:clear
karaf@root()> log:log -l ERROR "Hello World"
karaf@root()> log:display
12:55:41.460 ERROR [pipe-log:log "Hello World"] Hello World
```

19.1.2.6. log:set

log:set コマンドは、ロガーのログレベルを設定します。

デフォルトでは、**rootLogger** のログレベルが変更されます。

```
karaf@root()> log:set DEBUG
karaf@root()> log:get
Logger                              │ Level
────────────────────────────────────┼──────
ROOT                                │ DEBUG
...
```

level の後に **logger** 引数を使用して、特定のロガーを指定できます。

```
karaf@root()> log:set INFO my.logger
karaf@root()> log:get my.logger
Logger    | Level
```

```
-----------------
my.logger | INFO
```

level 引数は、Log4j2 ログレベルを受け入れます。TRACE、DEBUG、INFO、WARN、ERROR、または FATAL

また、DEFAULT の特別キーワードも使用できます。

DEFAULT キーワードの目的は、ロガーの現在のレベルを削除して (レベルのみ。アペンダーなどの他のプロパティーは削除されない)、ロガーの親レベル (ロガーは階層的) を使用することです。

たとえば、(**etc/org.ops4j.pax.logging.cfg** ファイルに) 次のロガーを定義しました。

```
rootLogger=INFO,out,osgi:*
my.logger=INFO,appender1
my.logger.custom=DEBUG,appender2
```

my.logger.custom ロガーのレベルを変更できます。

```
karaf@root()> log:set INFO my.logger.custom
```

以下のようになります。

```
rootLogger=INFO,out,osgi:*
my.logger=INFO,appender1
my.logger.custom=INFO,appender2
```

my.logger.custom ロガーで DEFAULT キーワードを使用して、レベルを削除できます。

```
karaf@root()> log:set DEFAULT my.logger.custom
```

以下のようになります。

```
rootLogger=INFO,out,osgi:*
my.logger=INFO,appender1
my.logger.custom=appender2
```

つまり、実行時に **my.logger.custom** ロガーは親 **my.logger** のレベルを使用するため、**INFO** となります。

ここで、**my.logger** ロガーで DEFAULT キーワードを使用するとします。

```
karaf@root()> log:set DEFAULT my.logger
```

以下のようになります。

```
rootLogger=INFO,out,osgi:*
my.logger=appender1
my.logger.custom=appender2
```

したがって、**my.logger.custom** と **my.logger** はどちらも、親の **rootLogger** のログレベルを使用します。

rootLogger で DEFAULT キーワードを使用することはできず、親はありません。

19.1.2.7. log:tail

log:tail は **log:display** と同じですが、ログエントリーが継続的に表示されます。

log:display コマンドと同じオプションと引数を使用できます。

デフォルトでは、**rootLogger** からのエントリーが表示されます。

```
karaf@root()> log:tail
2015-07-01 07:40:28,152 | INFO  | FelixStartLevel  | SecurityUtils                    | 16 -
org.apache.sshd.core - 0.9.0 | BouncyCastle not registered, using the default JCE provider
2015-07-01 07:40:28,909 | INFO  | FelixStartLevel  | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Starting JMX OSGi agent
2015-07-01 07:40:28,928 | INFO  | FelixStartLevel  | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering MBean with ObjectName
[osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=b44a44b7-
41cd-498f-936d-3b12d7aafa7b] for service with service.id [13]
2015-07-01 07:40:28,936 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.service.cm.ConfigurationAdminMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=b44a44b7-
41cd-498f-936d-3b12d7aafa7b
```

log:tail コマンドから終了するには、CTRL-C を入力します。

19.1.3. JMX LogMBean

log:* コマンドで実行可能なすべてのアクションは、LogMBean を使用して実行できます。

LogMBean オブジェクト名は **org.apache.karaf:type=log,name=*** です。

19.1.3.1. 属性

- **Level** 属性は、ROOT ロガーのレベルです。

19.1.3.2. 操作

- **getLevel(logger)** を使用して特定のロガーのログレベルを取得します。この操作は ALL キーワードをサポートするため、各ロガーのレベルで Map を返します。
- **setLevel(level, logger)** を使用して特定のロガーのログレベルを取得します。この操作は、**log:set** コマンドの DEFAULT キーワードをサポートします。

19.1.4. 詳細設定

19.1.4.1. SIFT ロギング

Fuse-Karaf は、$FUSE_HOME/etc/org.ops4j.pax.logging.cfg ファイルで、Log4j2 sift アペンダーの設定例 (デフォルトではコメントアウトして除外されている) と、このアペンダーを使用したロガーを提供します。

```
# Sift appender
```

```
log4j2.appender.mdc.type = Routing
log4j2.appender.mdc.name = SiftAppender
log4j2.appender.mdc.routes.type = Routes
# see: http://logging.apache.org/log4j/2.x/manual/appenders.html#Routes
log4j2.appender.mdc.routes.pattern = $\\{ctx:bundle.name}
log4j2.appender.mdc.routes.sift.type = Route
log4j2.appender.mdc.routes.sift.appender.type = RollingRandomAccessFile
log4j2.appender.mdc.routes.sift.appender.name = RollingFile
log4j2.appender.mdc.routes.sift.appender.fileName = ${karaf.data}/log/sift-$\\{ctx:bundle.name}.log
log4j2.appender.mdc.routes.sift.appender.filePattern = ${karaf.data}/log/sift-$\\{ctx:bundle.name}-
%i.log.gz
log4j2.appender.mdc.routes.sift.appender.append = true
log4j2.appender.mdc.routes.sift.appender.layout.type = PatternLayout
log4j2.appender.mdc.routes.sift.appender.layout.pattern = ${log4j2.pattern}
log4j2.appender.mdc.routes.sift.appender.policies.type = Policies
log4j2.appender.mdc.routes.sift.appender.policies.size.type = SizeBasedTriggeringPolicy
log4j2.appender.mdc.routes.sift.appender.policies.size.size = 16MB
log4j2.appender.mdc.routes.sift.appender.strategy.type = DefaultRolloverStrategy
log4j2.appender.mdc.routes.sift.appender.strategy.max = 20
...
# sample logger using Sift appender
#log4j2.logger.example.name = org.apache.camel
#log4j2.logger.example.level = INFO
#log4j2.logger.example.appenderRef.SiftAppender.ref = SiftAppender
```

この設定は、http://logging.apache.org/log4j/2.x/manual/appenders.html#RoutingAppender で説明されています。

SIFT/Routing アペンダーのパターンプロパティーは、ロギングのターゲットの場所を区別するために使用できます。

さまざまなルックアップを利用できますが、このルックアップについては http://logging.apache.org/log4j/2.x/manual/lookups.html を参照してください。

最も重要なルックアップは ctx で、ThreadContext マップ (MDC とも呼ばれる) の値 (キー) をルックアップします。

Fuse Karaf によって提供されるデフォルト設定は ctx:bundle.name をパターンとして使用します。つまり、以下のようになります。

```
lookup bundle.name key in MDC
```

pax-logging 自体が接頭辞の付いたキーを渡します。以下の異なる 3 つ値から選択できます。

- bundle.name == org.osgi.framework.Bundle.getSymbolicName()
- bundle.id == org.osgi.framework.Bundle.getBundleId()
- bundle.version == org.osgi.framework.Bundle.getVersion().toString()

ただし、Camel コンテキストが (Blueprint XML DSL) を使用して MDC サポートで作成されている場合:

```
<camelContext id="my-context" xmlns="http://camel.apache.org/schema/blueprint"
useMDCLogging="true">
```

MDC/ThreadContext で使用できる鍵は他にもあり、SIFT アペンダー設定のパターンとして使用できます。

- camel.exchangeId - エクスチェンジ ID
- camel.messageId - メッセージ ID
- camel.correlationId: 相関があればエクスチェンジの相関 ID。(例: Splitter EIP のサブメッセージ)。
- camel.transactionKey: トランザクションエクスチェンジのトランザクションの ID。id は一意ではありませんが、指定のトランザクションのトランザクション境界をマークするトランザクションテンプレートの ID であることに注意してください。そのため、このキーに transactionKey という名前を付け、transactionID でこのファクトを参照しないように決定しました。
- camel.routeId - エクスチェンジが現在ルーティングされるルートの ID
- camel.breadcrumbId: トランスポート全体でメッセージを追跡するために使用される一意の ID。
- camel.contextId - 異なる Camel コンテキストからメッセージを追跡するために使用される Camel コンテキスト ID。

https://people.apache.org/~dkulp/camel/mdc-logging.html を参照してください。

たとえば、ロギング宛先ファイルを Camel のルート ID で区別するには、以下を使用します。

```
log4j2.appender.mdc.routes.pattern = $\\{ctx:camel.routeId}
...
log4j2.appender.mdc.routes.sift.appender.fileName = ${karaf.data}/log/sift-$\\{ctx:camel.routeId}.log
log4j2.appender.mdc.routes.sift.appender.filePattern = ${karaf.data}/log/sift-$\\{ctx:camel.routeId}-
%i.log.gz
```

また、アペンダーの設定だけでは不十分であるため、一部のロガーに添付する必要があります。ここでも、設定のサンプルには以下が含まれます。

```
# sample logger using Sift appender
#log4j2.logger.example.name = org.apache.camel
#log4j2.logger.example.level = INFO
#log4j2.logger.example.appenderRef.SiftAppender.ref = SiftAppender
```

(log4j2.logger.example.appenderRef.SiftAppender.ref プロパティーの SiftAppender 値は、アペンダー設定の log4j2.appender.mdc.name の値と一致する必要があることに注意してください)。

ここでは、org.apache.camel はロガー名 (またはカテゴリー名) です。これは、Camel のログ: エンドポイントで使用される値と全く同じです。(Camel ルートの場合) の設定は以下のとおりです。

```
<to uri="log:org.apache.camel" />
```

ロギングは機能します。

もう 1 つの作業設定は以下のようになります。

```
<to uri="log:my-special-logger" />
```

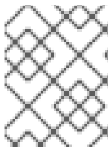
および

```
log4j2.logger.example.name = my-special-logger
log4j2.logger.example.level = DEBUG
log4j2.logger.example.appenderRef.SiftAppender.ref = SiftAppender
```

19.1.4.2. Filters

フィルターはアペンダーに適用できます。フィルターは各ログイベントを評価し、ログに送信するかどうかを決定します。

Log4j2 で、すぐに使用できるフィルターが提供されます。



注記

これらの包括的なビューについては、Log4J サイトの フィルター を参照してください。

19.1.4.3. ネストされたアペンダー

ネストされたアペンダーは、別のアペンダーを使用する特別な種類のアペンダーです。アペンダーのチェーン間である種のルーティングを設定できます。

最も使用される入れ子に準拠するアペンダーは以下のとおりです。

- AsyncAppender(**org.apache.log4j2.AsyncAppender**) は、イベントを非同期でログに記録します。このアペンダーはイベントを収集し、それに接続されているすべてのアペンダーにディスパッチします。
- RewriteAppender (**org.apache.log4j2.rewrite.RewriteAppender**) は、ログイベントを書き直した後、ログイベントを別のアペンダーに転送します。

このアペンダーは、アペンダー定義で **appenders** プロパティーを受け入れます。

```
log4j2.appender.[appender-name].appenders=[comma-separated-list-of-appender-names]
```

たとえば、**async** という名前の AsyncAppender を作成し、ログイベントを JMS アペンダーに非同期にディスパッチできます。

```
log4j2.appender.async=org.apache.log4j2.AsyncAppender
log4j2.appender.async.appenders=jms

log4j2.appender.jms=org.apache.log4j2.net.JMSAppender
...
```

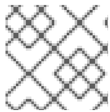
19.1.4.4. エラーハンドラー

アペンダーが失敗する可能性があります。たとえば、**RollingFileAppender** はファイルシステムへの書き込みを試みてもファイルシステムが満杯であったり、JMS アペンダーがメッセージを送信しようとしても JMS ブローカーは利用できない場合などです。

ロギングは極めて重要なので、ログアペンダーに障害があるかどうかを把握することは重要です。

各ログアペンダーはエラー処理をエラーハンドラーに委譲できるため、アペンダーエラーに対応することができます。

- FailoverAppender (**org.apache.log4j2.varia.FailoverAppender**) を使用すると、プライマリーアペンダーが失敗した場合にセカンダリーアペンダーを引き継ぐことができます。エラーメッセージが **System.err** で出力され、セカンダリーアペンダーのログに記録されます。



注記

FailoverAppender の詳細は、Log4j2 の Apppender ページ にアクセスしてください。

アペンダー定義自体で **errorhandler** プロパティーを使用して、各アペンダーに使用するエラーハンドラーを定義できます。

```
log4j2.appender.[appender-name].errorhandler=[error-handler-class]
log4j2.appender.[appender-name].errorhandler.root-ref=[true|false]
log4j2.appender.[appender-name].errorhandler.logger-ref=[logger-ref]
log4j2.appender.[appender-name].errorhandler.appender-ref=[appender-ref]
```

19.1.4.5. OSGi 固有の MDC 属性

routing アペンダーは、MDC (Mapped Diagnostic Context) 属性に基づいてログイベントの分割を可能にする OSGi 指向アペンダーです。

MDC では、ログイベントのさまざまなソースを区別できます。

routing アペンダーは、デフォルトで OSGi 指向の MDC 属性を提供します。

- **bundle.id** はバンドル ID です
- **bundle.name** は、バンドルのシンボリック名です。
- **bundle.version** はバンドルバージョンです

以下の MDC プロパティーを使用すると、バンドルごとにログファイルを作成できます。

```
log4j2.appender.routing.type = Routing
log4j2.appender.routing.name = Routing
log4j2.appender.routing.routes.type = Routes
log4j2.appender.routing.routes.pattern = $$\\{ctx:bundle.name\\}
log4j2.appender.routing.routes.bundle.type = Route
log4j2.appender.routing.routes.bundle.appender.type = RollingRandomAccessFile
log4j2.appender.routing.routes.bundle.appender.name = Bundle-$\\{ctx:bundle.name\}
log4j2.appender.routing.routes.bundle.appender.fileName = ${karaf.data}/log/bundle-$\\
{ctx:bundle.name\\}.log
log4j2.appender.routing.routes.bundle.appender.filePattern = ${karaf.data}/log/bundle-$\\
{ctx:bundle.name\\}.log.%d{yyyy-MM-dd}
log4j2.appender.routing.routes.bundle.appender.append = true
log4j2.appender.routing.routes.bundle.appender.layout.type = PatternLayout
log4j2.appender.routing.routes.bundle.appender.policies.type = Policies
log4j2.appender.routing.routes.bundle.appender.policies.time.type = TimeBasedTriggeringPolicy
log4j2.appender.routing.routes.bundle.appender.strategy.type = DefaultRolloverStrategy
log4j2.appender.routing.routes.bundle.appender.strategy.max = 31

log4j2.rootLogger.appenderRef.Routing.ref = Routing
```

19.1.4.6. OSGi スタックトレースレンダラーの強化

デフォルトでは、Apache Karaf は特殊なスタックトレースレンダラーを提供し、OSGi 固有の情報を追加します。

スタックトレースでは、例外を出力するクラスに加えて、各スタックトレース行の末尾にある **[id:name:version]** パターンを見つけることができます。

- **id** はバンドル ID です
- **name** はバンドル名です
- **version** はバンドルバージョンです

問題源を診断すると非常に役立ちます。

たとえば、以下の IllegalArgumentException スタックトレースでは、例外のソースに関する OSGi の詳細を確認できます。

```
java.lang.IllegalArgumentException: Command not found:  *:foo
 at org.apache.felix.gogo.runtime.shell.Closure.execute(Closure.java:225)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.felix.gogo.runtime.shell.Closure.executeStatement(Closure.java:162)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.felix.gogo.runtime.shell.Pipe.run(Pipe.java:101)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.felix.gogo.runtime.shell.Closure.execute(Closure.java:79)
[21:org.apache.karaf.shell.console:4.0.0]
 at
org.apache.felix.gogo.runtime.shell.CommandSessionImpl.execute(CommandSessionImpl.java:71)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.karaf.shell.console.jline.Console.run(Console.java:169)
[21:org.apache.karaf.shell.console:4.0.0]
 at java.lang.Thread.run(Thread.java:637)[:1.7.0_21]
```

19.1.4.7. カスタムアペンダー

Apache Karaf で独自のアペンダーを使用できます。

これを行う最も簡単な方法は、アペンダーを OSGi バンドルとしてパッケージ化し、**org.ops4j.pax.logging.pax-logging-service** バンドルのフラグメントとしてアタッチすることです。

たとえば、**MyAppender** を作成します。

```
public class MyAppender extends AppenderSkeleton {
...
}
```

MANIFEST を含む OSGi バンドルとしてコンパイルしてパッケージ化すると以下のようになります。

```
Manifest:
Bundle-SymbolicName: org.mydomain.myappender
Fragment-Host: org.ops4j.pax.logging.pax-logging-service
...
```

Apache Karaf **system** フォルダーでバンドルをコピーします。**system** フォルダーは、標準の Maven ディレクトリーレイアウト groupId/artifactId/version を使用します。

etc/startup.properties 設定ファイルで、pax-logging-service バンドルの前にバンドルをリストで定義します。

システムバンドルをリロードするには、クリーンな実行 (**data** フォルダーのパージ) で Apache Karaf を再起動する必要があります。これで、**etc/org.ops4j.pax.logging.cfg** 設定ファイルでアペンダーを直接使用できるようになります。

第20章 セキュリティー

Apache Karaf は、JAAS (Java Authentication and Authorization Service) が搭載する高度なセキュリティーシステムを OSGi に準拠する方法で提供します。

動的セキュリティーシステムを提供します。

Apache Karaf セキュリティーフレームワークは、以下へのアクセスを制御するために内部で使用されます。

- OSGi サービス (開発者ガイドで説明)
- コンソールコマンド
- JMX レイヤー
- WebConsole

アプリケーションはセキュリティーフレームワークを使用することもできます (詳細は、開発者ガイドを参照してください)。

20.1. レルム

Apache Karaf は複数のレルムを管理できます。レルムには、このレルムの認証や承認に使用するログインモジュールの定義が含まれます。ログインモジュールは、レルムの認証および承認を定義します。

jaas:realm-list コマンドは、現在定義されたレルムを一覧表示します。

```
karaf@root()> jaas:realm-list
Index | Realm Name | Login Module Class Name
-----------------------------------------------------------------------------------
1     | karaf      | org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
2     | karaf      | org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
```

Apache Karaf は **karaf** という名前のデフォルトのレルムを提供していることがわかります。

このレルムには 2 つのログインモジュールがあります。

- **PropertiesLoginModule** は、**etc/users.properties** ファイルをユーザー、グループ、ロール、およびパスワードのバックエンドとして使用します。このログインモジュールはユーザーを認証し、ユーザーのロールを返します。
- **PublickeyLoginModule** は、特に SSHd で使用されます。**etc/keys.properties** ファイルを使用します。このファイルには、ユーザーおよび各ユーザーに関連付けられた公開鍵が含まれます。

Apache Karaf は、追加のログインモジュールを提供します (詳細は、開発者ガイドを参照してください)。

- JDBCLoginModule がデータベースをバックエンドとして使用する
- LdapLoginModule が LDAP サーバーをバックエンドとして使用する
- SyncopeLoginModule が Apache Syncope をバックエンドとして使用する
- OsgiConfigLoginModule は設定をバックエンドとして使用する

- Krb5LoginModule が Kerberos サーバーをバックエンドとして使用する
- GSSAPILdapLoginModule は LDAP サーバーをバックエンドとして使用するが、LDAP サーバー認証を他のバックエンド (通常は Krb5LoginModule) に委譲する

jaas:realm-manage コマンドを使用して、既存のレルムとログインモジュールを管理するか、独自のレルムを作成できます。

20.1.1. ユーザー、グループ、ロール、およびパスワード

Apache Karaf はデフォルトで PropertiesLoginModule を使用する

このログインモジュールは、ユーザー、グループ、ロール、およびパスワードのストレージとして **etc/users.properties** ファイルを使用します。

初期の **etc/users.properties** ファイルには次のものが含まれています。

```
################################################################################
#
#    Licensed to the Apache Software Foundation (ASF) under one or more
#    contributor license agreements.  See the NOTICE file distributed with
#    this work for additional information regarding copyright ownership.
#    The ASF licenses this file to You under the Apache License, Version 2.0
#    (the "License"); you may not use this file except in compliance with
#    the License.  You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to in writing, software
#    distributed under the License is distributed on an "AS IS" BASIS,
#    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#    See the License for the specific language governing permissions and
#    limitations under the License.
#
################################################################################

#
# This file contains the users, groups, and roles.
# Each line has to be of the format:
#
# USER=PASSWORD,ROLE1,ROLE2,...
# USER=PASSWORD,_g_:GROUP,...
# _g_\:GROUP=ROLE1,ROLE2,...
#
# All users, grousp, and roles entered in this file are available after Karaf startup
# and modifiable via the JAAS command group. These users reside in a JAAS domain
# with the name "karaf".
#
karaf = karaf,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer
```

デフォルトではユーザーは **karaf** の 1 人のみであることが分かります。デフォルトのパスワードは、**karaf** です。

karaf ユーザーは 1 つのグループ **admingroup** のメンバーです。

グループの先頭には常に **g:** が付けられます。この接頭辞のないエントリーはユーザーです。

グループはロールのセットを定義します。デフォルトでは、**admingroup** は、**group**、**admin**、**manager**、および **viewer** のロールを定義します。

つまり、**karaf** ユーザーは **admingroup** によって定義されたロールを持つことを意味します。

20.1.1.1. コマンド

jaas:* コマンドは、コンソールでレルム、ユーザー、グループ、および ロールを管理します。

20.1.1.1.1. jaas:realm-list

このセクションでは、これまでに **jaas:realm-list** を使用しました。

jaas:realm-list コマンドは、レルムと各レルムのログインモジュールを一覧表示します。

```
karaf@root()> jaas:realm-list
Index | Realm Name | Login Module Class Name
-----------------------------------------------------------------------------------
1     | karaf      | org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
2     | karaf      | org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
```

ここに、2 つのログインモジュール (**PropertiesLoginModule** と **PublickeyLoginModule**) を含む 1 つのレルム (**karaf**) があります。

index コマンドは、管理するレルム/ログインモジュールを簡単に識別するために **jaas:realm-manage** コマンドによって使用されます。

20.1.1.1.2. jaas:realm-manage

jaas:realm-manage コマンドは、レルム/ログインモジュールの編集モードを切り替え、ログインモジュールでユーザー、グループ、およびロールを管理できます。

管理するレルムおよびログインモジュールを識別するには、**--index** オプションを使用します。インデックスは **jaas:realm-list** コマンドによって表示されます。

```
karaf@root()> jaas:realm-manage --index 1
```

もう 1 つの方法は、**--realm** および **--module** オプションを使用することです。**--realm** オプションはレルム名を想定し、**--module** オプションはログインモジュールクラス名を想定します。

```
karaf@root()> jaas:realm-manage --realm karaf --module
org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
```

20.1.1.1.3. jaas:user-list

編集モードの場合は、**jaas:user-list** を使用してログインモジュールのユーザーを一覧表示できます。

```
karaf@root()> jaas:user-list
User Name | Group      | Role
--------------------------------
```

```
karaf     | admingroup | admin
karaf     | admingroup | manager
karaf     | admingroup | viewer
```

ユーザー名およびグループはロールごとに確認できます。

20.1.1.1.4. jaas:user-add

jaas:user-add コマンドは、現在編集中のログインモジュールに新しいユーザー (およびパスワード) を追加します。

```
karaf@root()> jaas:user-add foo bar
```

変更 (ユーザーの追加) をコミットするには、**jaas:update** コマンドを実行します。

```
karaf@root()> jaas:update
karaf@root()> jaas:realm-manage --index 1
karaf@root()> jaas:user-list
User Name | Group      | Role
--------------------------------
karaf     | admingroup | admin
karaf     | admingroup | manager
karaf     | admingroup | viewer
foo       |            |
```

一方、ユーザーの追加をロールバックする場合は、**jaas:cancel** コマンドを使用できます。

20.1.1.1.5. jaas:user-delete

jaas:user-delete コマンドは、現在編集中のログインモジュールからユーザーを削除します。

```
karaf@root()> jaas:user-delete foo
```

jaas:user-add コマンドの場合と同様に、**jaas:update** を使用して変更をコミットする必要があります (または **jaas:cancel** を使用してロールバックする必要があります)。

```
karaf@root()> jaas:update
karaf@root()> jaas:realm-manage --index 1
karaf@root()> jaas:user-list
User Name | Group      | Role
--------------------------------
karaf     | admingroup | admin
karaf     | admingroup | manager
karaf     | admingroup | viewer
```

20.1.1.1.6. jaas:group-add

jaas:group-add コマンドは、現在編集中のログインモジュールでグループをユーザーに割り当てます (最終的にグループを作成)。

```
karaf@root()> jaas:group-add karaf mygroup
```

20.1.1.1.7. jaas:group-delete

jaas:group-delete コマンドは、現在編集中のログインモジュールでグループからユーザーを削除します。

```
karaf@root()> jaas:group-delete karaf mygroup
```

20.1.1.1.8. jaas:group-role-add

jaas:group-role-add コマンドは、現在編集中のログインモジュールでグループのロールを追加します。

```
karaf@root()> jaas:group-role-add mygroup myrole
```

20.1.1.1.9. jaas:group-role-delete

jaas:group-role-delete コマンドは、現在編集中のログインモジュールでグループからロールを削除します。

```
karaf@root()> jaas:group-role-delete mygroup myrole
```

20.1.1.1.10. jaas:update

jaas:update コマンドは、ログインモジュールのバックエンドで変更をコミットします。たとえば、PropertiesLoginModule の場合、**etc/users.properties** は **jaas:update** コマンドの実行後にのみ更新されます。

20.1.1.1.11. jaas:cancel

jaas:cancel コマンドは変更をロールバックし、ログインモジュールのバックエンドを更新しません。

20.1.2. パスワードの暗号化

デフォルトでは、パスワードは **etc/users.properties** ファイルにクリアテキストで保存されます。

etc/org.apache.karaf.jaas.cfg 設定ファイルで暗号化を有効にすることができます。

```
################################################################################
#
#    Licensed to the Apache Software Foundation (ASF) under one or more
#    contributor license agreements.  See the NOTICE file distributed with
#    this work for additional information regarding copyright ownership.
#    The ASF licenses this file to You under the Apache License, Version 2.0
#    (the "License"); you may not use this file except in compliance with
#    the License.  You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to in writing, software
#    distributed under the License is distributed on an "AS IS" BASIS,
#    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#    See the License for the specific language governing permissions and
#    limitations under the License.
```

```
#
########################################################################

#
# Boolean enabling / disabling encrypted passwords
#
encryption.enabled = false

#
# Encryption Service name
#   the default one is 'basic'
#   a more powerful one named 'jasypt' is available
#       when installing the encryption feature
#
encryption.name =

#
# Encryption prefix
#
encryption.prefix = {CRYPT}

#
# Encryption suffix
#
encryption.suffix = {CRYPT}

#
# Set the encryption algorithm to use in Karaf JAAS login module
# Supported encryption algorithms follow:
#   MD2
#   MD5
#   SHA-1
#   SHA-256
#   SHA-384
#   SHA-512
#
encryption.algorithm = MD5

#
# Encoding of the encrypted password.
# Can be:
#   hexadecimal
#   base64
#
encryption.encoding = hexadecimal
```

encryption.enabled が true に設定されている場合、パスワードの暗号化が有効になります。

暗号化を有効にすると、パスワードは、ユーザーの初回ログイン時に暗号化されます。暗号化したパスワードの先頭と末尾に、\{**CRYPT**\} が付きます。パスワードを再暗号化するには、先頭と末尾の \{**CRYPT**\} がない状態でパスワードをリセットできます (**etc/users.properties** ファイルで)。Apache Karaf は、このパスワードが明確であることを検出し (先頭と末尾に \{**CRYPT**\} がないため)、再度暗号化します。

etc/org.apache.karaf.jaas.cfg 設定ファイルを使用すると、高度な暗号化動作を定義できます。

- **encryption.prefix** プロパティーは、パスワードが暗号化されていることをフラグする接頭辞を定義します。デフォルトは \{**CRYPT**\} です。
- **encryption.suffix** プロパティーは、パスワードが暗号化されていることをフラグする接尾辞を定義します。デフォルトは \{**CRYPT**\} です。
- **encryption.algorithm** プロパティーは、暗号化 (digest) に使用するアルゴリズムを定義します。可能な値は、**MD2**、**MD5**、**SHA-1**、**SHA-256**、**SHA-384**、**SHA-512** です。デフォルトは **MD5** です。
- **encryption.encoding** プロパティーは、暗号化されたパスワードのエンコーディングを定義します。可能な値は **hexadecimal** または **base64** です。デフォルト値は **hexadecimal** です。

20.1.3. キーによる認証の管理

SSH レイヤーでは Karaf はキーによる認証をサポートし、パスワードなしでログインできるようにします。

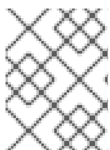
SSH クライアント (Karaf 自体によって提供される bin/client、OpenSSH などの ssh クライアント) は、Karaf SSHD(サーバー側) 上で自身を識別する公開鍵と秘密鍵のペアを使用します。

接続が許可された鍵は、次の形式に従って **etc/keys.properties** ファイルに格納されます。

```
user=key,role
```

デフォルトでは、Karaf は karaf ユーザーのキーを許可します。

```
#
karaf=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH
7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E+
4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAAAAFQCXYFCPFSMLzLKSuYKi64QL8Fgc9Q
AAAIEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6E
woFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWf
BpKLZl6Ae1UlZAFMO/7PSSoAAACBAKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskl8
wO63OzP/qLmcJM0+JbcRU/53JjTuyk31drV2qxhIOsLDC9dGCWj47Y7TyhPdXh/0dthTRBy6bqGtRPxG
a7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx,admin
```



注記

セキュリティー上の理由から、このキーは無効になっています。クライアントごとにキーペアを作成し、**etc/keys.properties** ファイルを更新することが推奨されます。

キーペアを作成する最も簡単な方法は、OpenSSH を使用することです。

以下を使用してキーペアを作成できます。

```
ssh-keygen -t dsa -f karaf.id_dsa -N karaf
```

これで公開鍵と秘密鍵ができました。

```
-rw-------  1 jbonofre jbonofre    771 Jul 25 22:05 karaf.id_dsa
-rw-r--r--  1 jbonofre jbonofre    607 Jul 25 22:05 karaf.id_dsa.pub
```

etc/keys.properties の **karaf.id_dsa.pub** ファイルの内容をコピーできます。

```
karaf=AAAAB3NzaC1kc3MAAACBAJLj9vnEhu3/Q9Cvym2jRDaNWkATgQiHZxmErCmiLRuD5Klfv+HT/
+8WoYdnvj0YaXFP80phYhzZ7fbIO2LRFhYhPmGLa9nSeOsQlFuX5A9kY1120yB2kxSIZI0fU2hy1UCg
mTxdTQPSYtdWBJyvO/vczoX/8I3FziEfss07Hj1NAAAAFQD1dKEzkt4e7rBPDokPOMZigBh4kwAAAIEAi
LnpbGNbKm8SNLUEc/fJFswg4G4VjjngjbPZAjhkYe4+H2uYmynry6V+GOTS2kaFQGZRf9XhSpSwfdxK
tx7vCCaoH9bZ6S5Pe0voWmeBhJXi/Sww8f2stpitW2Oq7V7lDdDG81+N/D7/rKDD5PjUyMsVqc1n9wCT
mfqmi6XPEw8AAACAHAGwPn/Mv7P9Q9+JZRWtGq+i4pL1zs1OluiStCN9e/Ok96t3gRVKPheQ6IwLac
NjC9KkSKrLtsVyepGA+V5j/N+Cmsl6csZilnLvMUTvL/cmHDEEhTIQnPNrDDv+tED2BFqkajQqYLgMWe
GVqXsBU6IT66itZlYtrq4v6uDQG/o=,admin
```

また、クライアントに **karaf.id_dsa** 秘密鍵を使用するように指定します。

```
bin/client -k ~/karaf.id_dsa
```

または ssh を使用します。

```
ssh -p 8101 -i ~/karaf.id_dsa karaf@localhost
```

20.1.4. RBAC

Apache Karaf はロールを使用してリソースへのアクセスを制御します。これは RBAC(Role Based Access Control) システムです。

ロールは、以下を制御するために使用されます。

- OSGi サービスへのアクセス
- コンソールへのアクセス (コマンドの実行の制御)
- JMX へのアクセス (MBean や操作)
- WebConsole へのアクセス

20.1.4.1. OSGi サービス

OSGi サービス RBAC サポートの詳細は、開発者ガイドを参照してください。

20.1.4.2. コンソール

コンソール RBAC のサポートは、OSGi サービス RBAC を特化したものです。実際には、Apache Karaf では、すべてのコンソールコマンドは OSGi サービスとして定義されます。

コンソールのコマンド名は **scope:name** 形式に従います。

ACL(アクセスリスト) は、**etc/org.apache.karaf.command.acl.<scope>.cfg** 設定ファイルで定義されています。**<scope>** はコマンドスコープです。

たとえば、**etc/org.apache.karaf.command.acl.feature.cfg** 設定ファイルを作成することにより、**feature:*** コマンドに対する ACL を定義できます。この **etc/org.apache.karaf.command.acl.feature.cfg** 設定ファイルでは、次のように設定できます。

```
list = viewer
info = viewer
```

```
install = admin
uninstall = admin
```

ここでは、**feature:list** コマンドと **feature:info** コマンドは、**viewer** ロールを持つユーザーが実行できるのに対し、**feature:install** コマンドと **feature:uninstall** コマンドは **admin** ロールを持つユーザーのみが実行できることを定義します。admin グループのユーザーも viewer ロールを持っているため、すべて実行できることに注意してください。

Apache Karaf コマンド ACL は、(指定のコマンドスコープ内で) を使用してアクセスを制御できます。

- コマンド名の正規表現 (例: **name = role**)
- コマンド名とオプションまたは引数の値の正規表現 (例: 100 を超える引数の値でのみ名前を実行する **name[/.[0-9][0-9][0-9]+./] = role**)

コマンド名とオプション/引数はいずれも、完全一致または正規表現の一致をサポートします。

デフォルトでは、Apache Karaf は以下の ACL を定義します。

- **etc/org.apache.karaf.command.acl.bundle.cfg** 設定ファイルは、**bundle:*** コマンドの ACL を定義します。この ACL は、システムバンドルの **bundle:*** コマンドの実行を **admin** ロールが割り当てられたユーザーのみに制限しますが、システム以外のバンドルの **bundle:*** コマンドは、**manager** ロールを持つユーザーが実行できます。
- **etc/org.apache.karaf.command.acl.config.cfg** 設定ファイルは、**config:*** コマンドの ACL を定義します。この ACL は、**jmx.acl.***、**org.apache.karaf.command.acl.***、および **org.apache.karaf.service.acl.*** 設定 PID を使用した **config:*** コマンドの実行を **admin** ロールを持つユーザーに制限します。他の設定 PID の場合、**manager** ロールを持つユーザーは **config:*** コマンドを実行できます。
- **etc/org.apache.karaf.command.acl.feature.cfg** 設定ファイルは、**feature:*** コマンドの ACL を定義します。**admin** ロールを持つユーザーのみが **feature:install** および **feature:uninstall** コマンドを実行できます。その他の **feature:*** コマンドは、任意のユーザーが実行できます。
- **etc/org.apache.karaf.command.acl.jaas.cfg** 設定ファイルは、**jaas:*** コマンドの ACL を定義します。**admin** ロールを持つユーザーのみが **jaas:update** コマンドを実行できます。その他の **jaas:*** コマンドは、任意のユーザーが実行できます。
- **etc/org.apache.karaf.command.acl.kar.cfg** 設定ファイルは、**kar:*** コマンドの ACL を定義します。**admin** ロールを持つユーザーのみが **kar:install** および **kar:uninstall** コマンドを実行できます。その他の **kar:*** コマンドは、任意のユーザーが実行できます。
- **etc/org.apache.karaf.command.acl.shell.cfg** 設定ファイルは、**shell:*** コマンドと"direct"コマンドの ACL を定義します。**admin** ロールを持つユーザーのみが、**shell:edit**、**shell:exec**、**shell:new**、および **shell:java** コマンドを実行できます。その他の **shell:*** コマンドは、任意のユーザーが実行できます。

これらのデフォルト ACL を変更し、追加のコマンド範囲に自身の ACL を追加できます (Apache Karaf Cellar の場合は **etc/org.apache.karaf.command.acl.cluster.cfg**、Apache Camel の場合は **etc/org.apache.karaf.command.acl.camel.cfg** など)。の深度を任意に決定できます。

etc/system.properties の **karaf.secured.services** プロパティーを編集することにより、コマンド RBAC サポートを微調整できます。

```
#
# By default, only Karaf shell commands are secured, but additional services can be
# secured by expanding this filter
```

```
#
karaf.secured.services = (&(osgi.command.scope=*)(osgi.command.function=*))
```

20.1.4.3. JMX

コンソールコマンドと同様に、ACL(AccessList) を JMX レイヤーに定義できます。

JMX ACL は、**etc/jmx.acl<ObjectName>.cfg** 設定ファイルで定義されます。**<ObjectName>** は MBean オブジェクト名です (たとえば、**org.apache.karaf.bundle** は **org.apache.karaf;type=Bundle** MBean を表します)。

etc/jmx.acl.cfg は最も汎用的な設定ファイルであり、特定の設定ファイルが見つからない場合に使用されます。グローバル ACL 定義が含まれます。

JMX ACL は、(JMX MBean 内) を使用してアクセスを制御できます。

- 操作名の正規表現 (例: **operation* = role**)
- 操作引数値正規表現 (例: **operation(java.lang.String, int)[/([1-4])?[0-9]/,/.*/] = role**)

デフォルトでは、Apache Karaf は以下の JMX ACL を定義します。

- **etc/jmx.acl.org.apache.karaf.bundle.cfg** 設定ファイルは、**org.apache.karaf:type=bundle** MBean の ACL を定義します。この ACL は、**admin** ロールを持つユーザーに対してのみ、システムバンドル **setStartLevel()**、**start()**、**stop()**、および **update()** 操作を制限します。その他の操作は、**manager** のロールを持つユーザーが実行できます。
- **etc/jmx.acl.org.apache.karaf.config.cfg** 設定ファイルは、**org.apache.karaf:type=config** MBean の ACL を定義します。この ACL は、**jmx.acl***、**org.apache.karaf.command.acl***、および **org.apache.karaf.service.acl*** 設定 PID の変更を、 **admin** ロールを持つユーザーに制限します。その他の操作は、**manager** のロールを持つユーザーが実行できます。
- **etc/jmx.acl.org.apache.karaf.security.jmx.cfg** 設定ファイルは、**org.apache.karaf:type=security,area=jmx** MBean の ACL を定義します。この ACL は、**viewer** ロールを持つユーザーに **canInvoke()** 操作の呼び出しを制限します。
- **etc/jmx.acl.osgi.compendium.cm.cfg** 設定ファイルは、**osgi.compendium:type=cm** MBean の ACL を定義します。この ACL は、**jmx.acl***、**org.apache.karaf.command.acl***、および **org.apache.karaf.service.acl*** 設定 PID の変更を、 **admin** ロールを持つユーザーに制限します。その他の操作は、**manager** のロールを持つユーザーが実行できます。
- **etc/jmx.acl.java.lang.Memory.cfg** 設定ファイルは、コア JVM Memory MBean の ACL を定義します。この ACL は、**gc** 操作の呼び出しを **manager** ロールを持つユーザーのみに制限します。
- **etc/jmx.acl.cfg** 設定ファイルは最も汎用的なファイルです。ここで定義された ACL は、他の特定の ACL が一致しない場合に使用されます (特定の ACL によって)。これは別の MBean 固有の **etc/jmx.acl.*.cfg** 設定ファイルで定義される ACL です。**list*()**、**get*()**、**is*()** 操作は、**viewer** ロールを持つユーザーが実行できます。**set*()** および他のすべての ***()** 操作は、**admin** ロールを持つユーザーが実行できます。

20.1.4.4. WebConsole

Apache Karaf WebConsole はデフォルトで利用できません。これを有効にするには、**webconsole** 機能をインストールする必要があります。

```
karaf@root()> feature:install webconsole
```

WebConsole では、コンソールや JMX などの粒度の細かい RBAC はサポートされません。

admin ロールを持つすべてのユーザーが WebConsole にログインし、任意の操作を実行できます。

20.1.5. SecurityMBean

Apache Karaf は、現在のユーザーが指定の MBean や操作を呼び出すことができるかどうかをチェックする JMX MBean を提供します。

canInvoke() 操作は現在のユーザーのロールを取得し、最終的に指定された引数値を使用して、ロールが MBean や操作を呼び出しできるかどうかを確認します。

20.1.5.1. 操作

- **canInvoke(objectName)** は、現在のユーザーが **objectName** を使用して MBean を呼び出すことができる場合は **true** を返し、できない場合は **false** を返します。
- **canInvoke(objectName)** は、現在のユーザーが **objectName** を使用して操作 **methodName** を MBean で呼び出すことができる場合は **true** を返し、できない場合は **false** を返します。
- **canInvoke(objectName, methodName, argumentTypes)** は、現在のユーザーが **objectName** を使用して、MBean で引数タイプ **argumentTypes** の配列を使用して操作 **methodName** を呼び出すことができる場合は **true** を返し、それ以外の場合は **false** を返します。
- **canInvoke(bulkQuery)** は、**canInvoke** が **true** または **false** の場合、**bulkQuery** 表形式データの各操作を含む表形式データを返します。

20.1.6. セキュリティープロバイダー

アプリケーションによっては、[BouncyCastle|http://www.bouncycastle.org] などの特定のセキュリティープロバイダーが利用可能である必要があります。

JVM には、署名済みでブートクラスパスで利用可能できる必要があるなど、jar の使用に関するいくつかの制限があります。

これらのプロバイダーをデプロイする方法の 1 つは、**$JAVA_HOME/jre/lib/ext** の JRE フォルダーに配置し、そのプロバイダーを登録するためにセキュリティーポリシー設定 (**$JAVA_HOME/jre/lib/security/java.security**) を変更することです。

このアプローチは問題なく機能しますが、グローバルな効果があり、それに応じてすべてのサーバーを設定する必要があります。

Apache Karaf は、追加のセキュリティープロバイダーを設定する簡単な方法を提供します。* プロバイダー jar を **lib/ext** に配置 * **etc/config.properties** 設定ファイルを変更して以下のプロパティーを追加

```
org.apache.karaf.security.providers = xxx,yyy
```

このプロパティーの値は、登録するプロバイダークラス名のコンマ区切りリストです。

たとえば、bouncycastle セキュリティープロバイダーを追加するには、以下を定義します。

```
org.apache.karaf.security.providers = org.bouncycastle.jce.provider.BouncyCastleProvider
```

さらに、すべてのバンドルがそれらにアクセスできるように、システムバンドルのこれらのプロバイダーからクラスにアクセスできるようにしてください。

これには、同じ設定ファイルの **org.osgi.framework.bootdelegation** プロパティーを変更します。

```
org.osgi.framework.bootdelegation = ...,org.bouncycastle*
```