OpenJDK 17

jlink を使用した Java ランタイム環境のカスタマイズ

ガイド

Last Updated: 2021-12-09

OpenJDK 17 jlink を使用した Java ランタイム環境のカスタマイズ

ガイド

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

OpenJDK 17 は、Red Hat Enterprise Linux プラットフォーム上の Red Hat 製品です。jlink を使用した Java ランタイムイメージのカスタマイズガイドは、Jlink の概要と、jlink を使用してカスタマイズされた Java ランタイムイメージを作成する方法を説明します。

目次

多様性を受け入れるオープンソースの強化

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

RED HAT ドキュメントへのフィードバック

弊社のドキュメントに関するご意見やご感想をお寄せください。フィードバックをお寄せいただくには、ドキュメントのテキストを強調表示し、コメントを追加できます。

本セクションでは、フィードバックの送信方法を説明します。

前提条件

- Red Hat カスタマーポータルにログインしている。
- Red Hat カスタマーポータルで、**マルチページ HTML** 形式でドキュメントを表示します。

手順

フィードバックを提供するには、以下の手順を実施します。

1. ドキュメントの右上隅にある **フィードバック** ボタンをクリックして、既存のフィードバックを確認します。

注記

フィードバック機能は、**マルチページ HTML** 形式でのみ有効です。

2. フィードバックを提供するドキュメントのセクションを強調表示します。
3. ハイライトされたテキスト近くに表示される **Add Feedback** ポップアップをクリックします。ページの右側のフィードバックセクションにテキストボックスが表示されます。
4. テキストボックスにフィードバックを入力し、**Submit** をクリックします。ドキュメントに関する問題が作成されます。
5. 問題を表示するには、フィードバックビューで問題トラッカーリンクをクリックします。

第1章 JLINK の概要

Jlink は、JRE (カスタムの Java ランタイム環境) の生成に使用される Java コマンドラインツールです。カスタマイズした JRE を使用して Java アプリケーションを実行できます。

jlink を使用して、関連するクラスファイルのみが含まれるカスタムのランタイム環境を作成できます。

第2章 モジュール以外のアプリケーション用のカスタム JAVA ランタイム環境の作成

jlink ツールを使用して、非モジュールアプリケーションからカスタム Java ランタイム環境を作成できます。

前提条件

- Installing OpenJDK on RHEL using an archive に従って、インストールします。

注記

最善の結果を得るには、移植可能な Red Hat バイナリーを Jlink ランタイムのベースとして使用します。これらのバイナリーにはバンドルされたライブラリーが含まれます。

手順

1. **Logger** クラスを使用して、単純な Hello World アプリケーションを作成します。
   a. ベースの OpenJDK 17 バイナリーが **jdk-17** フォルダーに存在することを確認します。

```
$ ls jdk-17
bin  conf  demo  include  jmods  legal  lib  man  NEWS  release
$ ./jdk-17/bin/java -version
openjdk version "17.0.10" 2021-01-19 LTS
OpenJDK Runtime Environment 18.9 (build 17.0.10+9-LTS)
OpenJDK 64-Bit Server VM 18.9 (build 17.0.10+9-LTS, mixed mode)
```

   b. アプリケーションのディレクトリーを作成します。

```
$ mkdir -p hello-example/sample
```

   c. 以下の内容を含む **hello-example/sample/HelloWorld.java** ファイルを作成します。

```
package sample;

import java.util.logging.Logger;

public class HelloWorld {
    private static final Logger LOG = Logger.getLogger(HelloWorld.class.getName());
    public static void main(String[] args) {
        LOG.info("Hello World!");
    }
}
```

   d. アプリケーションをコンパイルします。

```
$ ./jdk-17/bin/javac -d . $(find hello-example -name \*.java)
```

   e. カスタム JRE **なし**でアプリケーションを実行します。

```
$ ./jdk-17/bin/java sample.HelloWorld
Mar 09, 2021 10:48:59 AM sample.HelloWorld main
INFO: Hello World!
```

上記の例は、1 つのクラスを実行するために 311 MB を必要とするベース OpenJDK を示しています。

f. **(オプション)** OpenJDK を検査し、アプリケーションの必須でない多くのモジュールを表示できます。

```
$ du -sh jdk-17/
313M jdk-17/
```

```
$ ./jdk-17/bin/java --list-modules
java.base@17.0.1
java.compiler@17.0.1
java.datatransfer@17.0.1
java.desktop@17.0.1
java.instrument@17.0.1
java.logging@17.0.1
java.management@17.0.1
java.management.rmi@17.0.1
java.naming@17.0.1
java.net.http@17.0.1
java.prefs@17.0.1
java.rmi@17.0.1
java.scripting@17.0.1
java.se@17.0.1
java.security.jgss@17.0.1
java.security.sasl@17.0.1
java.smartcardio@17.0.1
java.sql@17.0.1
java.sql.rowset@17.0.1
java.transaction.xa@17.0.1
java.xml@17.0.1
java.xml.crypto@17.0.1
jdk.accessibility@17.0.1
jdk.attach@17.0.1
jdk.charsets@17.0.1
jdk.compiler@17.0.1
jdk.crypto.cryptoki@17.0.1
jdk.crypto.ec@17.0.1
jdk.dynalink@17.0.1
jdk.editpad@17.0.1
jdk.hotspot.agent@17.0.1
jdk.httpserver@17.0.1
jdk.incubator.foreign@17.0.1
jdk.incubator.vector@17.0.1
jdk.internal.ed@17.0.1
jdk.internal.jvmstat@17.0.1
jdk.internal.le@17.0.1
jdk.internal.opt@17.0.1
jdk.internal.vm.ci@17.0.1
jdk.internal.vm.compiler@17.0.1
jdk.internal.vm.compiler.management@17.0.1
```

```
jdk.jartool@17.0.1
jdk.javadoc@17.0.1
jdk.jcmd@17.0.1
jdk.jconsole@17.0.1
jdk.jdeps@17.0.1
jdk.jdi@17.0.1
jdk.jdwp.agent@17.0.1
jdk.jfr@17.0.1
jdk.jlink@17.0.1
jdk.jpackage@17.0.1
jdk.jshell@17.0.1
jdk.jsobject@17.0.1
jdk.jstatd@17.0.1
jdk.localedata@17.0.1
jdk.management@17.0.1
jdk.management.agent@17.0.1
jdk.management.jfr@17.0.1
jdk.naming.dns@17.0.1
jdk.naming.rmi@17.0.1
jdk.net@17.0.1
jdk.nio.mapmode@17.0.1
jdk.random@17.0.1
jdk.sctp@17.0.1
jdk.security.auth@17.0.1
jdk.security.jgss@17.0.1
jdk.unsupported@17.0.1
jdk.unsupported.desktop@17.0.1
jdk.xml.dom@17.0.1
jdk.zipfs@17.0.1
```

サンプル **Hello World** アプリケーションには、非常にいくつかの依存関係があります。jlink を使用して、アプリケーションのカスタムランタイムイメージを作成できます。これらのイメージを使用して、必要な OpenJDK 依存関係のみでアプリケーションを実行できます。

2. **jdeps** コマンドを使用して、アプリケーションのモジュール依存関係を決定します。

```
$ ./jdk-17/bin/jdeps -s ./sample/HelloWorld.class
HelloWorld.class -> java.base
HelloWorld.class -> java.logging
```

3. アプリケーションのカスタム java ランタイムイメージをビルドします。

```
$ ./jdk-17/bin/jlink --add-modules java.base,java.logging --output custom-runtime
$ du -sh custom-runtime
50M custom-runtime/
$ ./custom-runtime/bin/java --list-modules
java.base@17.0.10
java.logging@17.0.10
```

注記

OpenJDK は、313 M ランタイムイメージから 50 M ランタイムイメージに、カスタム Java ランタイムイメージのサイズを縮小します。

4. アプリケーションの縮小ランタイムを確認できます。

```
$ ./custom-runtime/bin/java sample.HelloWorld
Jan 14, 2021 12:13:26 PM HelloWorld main
INFO: Hello World!
```

サンプルアプリケーションで生成された JRE には他の依存関係がありません。

デプロイメントをカスタムランタイムと一緒に配布することができます。

注記

ベース OpenJDK のセキュリティー更新ごとに、アプリケーションのカスタム Java ランタイムイメージを再構築する必要があります。

第3章 モジュラーアプリケーション用のカスタム JAVA ランタイム環境の作成

jlink ツールを使用して、モジュラーアプリケーションからカスタム Java ランタイム環境を作成できます。

前提条件

- Installing OpenJDK on RHEL using an archive に従って、インストールします。

注記

最善の結果を得るには、移植可能な Red Hat バイナリーを Jlink ランタイムのベースとして使用します。これらのバイナリーにはバンドルされたライブラリーが含まれます。

手順

1. **Logger** クラスを使用して、単純な Hello World アプリケーションを作成します。
   a. ベースの OpenJDK 17 バイナリーが **jdk-17** フォルダーに存在することを確認します。

```
$ ls jdk-17
bin  conf  demo  include  jmods  legal  lib  man  NEWS  release
$ ./jdk-17/bin/java -version
openjdk version "17.0.10" 2021-01-19 LTS
OpenJDK Runtime Environment 18.9 (build 17.0.10+9-LTS)
OpenJDK 64-Bit Server VM 18.9 (build 17.0.10+9-LTS, mixed mode)
```

   b. アプリケーションのディレクトリーを作成します。

```
$ mkdir -p hello-example/sample
```

   c. 以下の内容を含む **hello-example/sample/HelloWorld.java** ファイルを作成します。

```
package sample;

import java.util.logging.Logger;

public class HelloWorld {
    private static final Logger LOG = Logger.getLogger(HelloWorld.class.getName());
    public static void main(String[] args) {
        LOG.info("Hello World!");
    }
}
```

   d. **hello-example/module-info.java** という名前のファイルを作成し、以下のコードをファイルに追加します。

```
module sample
{
    requires java.logging;
}
```

e. アプリケーションをコンパイルします。

```
$ ./jdk-17/bin/javac -d example $(find hello-example -name \*.java)
```

f. カスタム JRE **なし**でアプリケーションを実行します。

```
$ ./jdk-17/bin/java -cp example sample.HelloWorld
Mar 09, 2021 10:48:59 AM sample.HelloWorld main
INFO: Hello World!
```

上記の例は、1 つのクラスを実行するために 311 MB を必要とするベース OpenJDK を示しています。

g. **(オプション)** OpenJDK を検査し、アプリケーションの必須でない多くのモジュールを表示できます。

```
$ du -sh jdk-17/
313M jdk-17/
```

```
$ ./jdk-17/bin/java --list-modules
java.base@17.0.1
java.compiler@17.0.1
java.datatransfer@17.0.1
java.desktop@17.0.1
java.instrument@17.0.1
java.logging@17.0.1
java.management@17.0.1
java.management.rmi@17.0.1
java.naming@17.0.1
java.net.http@17.0.1
java.prefs@17.0.1
java.rmi@17.0.1
java.scripting@17.0.1
java.se@17.0.1
java.security.jgss@17.0.1
java.security.sasl@17.0.1
java.smartcardio@17.0.1
java.sql@17.0.1
java.sql.rowset@17.0.1
java.transaction.xa@17.0.1
java.xml@17.0.1
java.xml.crypto@17.0.1
jdk.accessibility@17.0.1
jdk.attach@17.0.1
jdk.charsets@17.0.1
jdk.compiler@17.0.1
jdk.crypto.cryptoki@17.0.1
jdk.crypto.ec@17.0.1
jdk.dynalink@17.0.1
jdk.editpad@17.0.1
jdk.hotspot.agent@17.0.1
jdk.httpserver@17.0.1
jdk.incubator.foreign@17.0.1
jdk.incubator.vector@17.0.1
jdk.internal.ed@17.0.1
```

```
jdk.internal.jvmstat@17.0.1
jdk.internal.le@17.0.1
jdk.internal.opt@17.0.1
jdk.internal.vm.ci@17.0.1
jdk.internal.vm.compiler@17.0.1
jdk.internal.vm.compiler.management@17.0.1
jdk.jartool@17.0.1
jdk.javadoc@17.0.1
jdk.jcmd@17.0.1
jdk.jconsole@17.0.1
jdk.jdeps@17.0.1
jdk.jdi@17.0.1
jdk.jdwp.agent@17.0.1
jdk.jfr@17.0.1
jdk.jlink@17.0.1
jdk.jpackage@17.0.1
jdk.jshell@17.0.1
jdk.jsobject@17.0.1
jdk.jstatd@17.0.1
jdk.localedata@17.0.1
jdk.management@17.0.1
jdk.management.agent@17.0.1
jdk.management.jfr@17.0.1
jdk.naming.dns@17.0.1
jdk.naming.rmi@17.0.1
jdk.net@17.0.1
jdk.nio.mapmode@17.0.1
jdk.random@17.0.1
jdk.sctp@17.0.1
jdk.security.auth@17.0.1
jdk.security.jgss@17.0.1
jdk.unsupported@17.0.1
jdk.unsupported.desktop@17.0.1
jdk.xml.dom@17.0.1
jdk.zipfs@17.0.1
```

サンプル **Hello World** アプリケーションには、非常にいくつかの依存関係があります。jlink を使用して、アプリケーションのカスタムランタイムイメージを作成できます。これらのイメージを使用して、必要な OpenJDK 依存関係のみでアプリケーションを実行できます。

2. アプリケーションモジュールを作成します。

```
$ mkdir sample-module
$ ./jdk-17/bin/jmod create --class-path example/ --main-class sample.HelloWorld --module-version 1.0.0 -p example sample-module/hello.jmod
```

3. 必要なモジュールとアプリケーションのカスタムアプリケーションランチャーを使用して、カスタム JRE を作成します。

```
$ ./jdk-17/bin/jlink --launcher hello=sample/sample.HelloWorld --module-path sample-module --add-modules sample --output custom-runtime
```

4. 生成したカスタム JRE のモジュールを一覧表示します。
元の OpenJDK の分数だけを維持していることを確認します。

```
$ du -sh custom-runtime
50M custom-runtime/
$ ./custom-runtime/bin/java --list-modules
java.base@17.0.10
java.logging@17.0.10
sample@1.0.0
```

注記

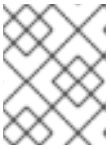
OpenJDK は、313 M ランタイムイメージから 50 M ランタイムイメージに、カスタム Java ランタイムイメージのサイズを縮小します。

5. **hello** launcher を使用してアプリケーションを起動します。

```
$ ./custom-runtime/bin/hello
Jan 14, 2021 12:13:26 PM HelloWorld main
INFO: Hello World!
```

サンプルアプリケーションと生成された JRE には **java.base**、**java.logging**、および **sample** モジュール以外の他の依存関係がありません。

custom-runtime でカスタムランタイムにバンドルされたアプリケーションを配布できます。このカスタムランタイムにはアプリケーションが含まれます。



注記

ベース OpenJDK のセキュリティー更新ごとに、アプリケーションのカスタム Java ランタイムイメージを再構築する必要があります。