Springboot 可执行 Jar 的格式

spring-boot-loader 模块让 Spring Boot 支持可执行的 jar 和 war 文件。如果使用 Maven 或 Gradle 插件，则会自动生成可执行的 jar，您通常不需要了解它们的工作原理。
如果您需要从不同的 build 系统创建可执行 jar，或者您只是对底层技术感到好奇，本附录提供了一些背景知识。

1. 嵌套 JAR

Java 没有提供任何标准的方法来加载嵌套的 jar 文件（jar 文件本身包含在另一个 jar 中）。如果您需要分发可以从命令行运行而无需解包的自包含应用程序，这可能会出现问题。如果您需要分发一个可以从命令行运行而不需要解包的自包含的应用程序，那么这可能会有问题。

为了解决这个问题，许多开发人员使用“shaded” jars。shaded jar 将所有 jar 中的所有类打包到一个超级 jar 中。它的问题在于，很难查看应用程序中实际使用了哪些库。如果在多个 jar 中使用相同的文件名（但内容不同），也会出现问题。Spring Boot 采用了不同的方法，让您实际上可以直接嵌套 jar。

1.1. 可执行 Jar 文件结构

Spring Boot Loader 兼容的 jar 文件应按以下方式构建：

```
example.jar
 |
 +-META-INF
 |  +-MANIFEST.MF
 +-org
 |  +-springframework
 |     +-boot
 |        +-loader
```

```
|               +-<spring boot loader classes>
+-BOOT-INF
    +-classes
    |   +-mycompany
    |       +-project
    |           +-YourClasses.class
    +-lib
        +-dependency1.jar
        +-dependency2.jar
```

应用程序类应放置在嵌套 的 `BOOT-INF/classes` 目录中。依赖项应该放在嵌套的 `BOOT-INF/lib` 目录中。

1.2. 可执行 War 文件结构

Spring Boot Loader 兼容的 war 文件应按以下方式构建：

```
example.war
|
+-META-INF
|   +-MANIFEST.MF
+-org
|   +-springframework
|       +-boot
```

```
|           +-loader
|               +-<spring boot loader classes>
+-WEB-INF
    +-classes
    |   +-com
    |       +-mycompany
    |           +-project
    |               +-YourClasses.class
    +-lib
    |   +-dependency1.jar
    |   +-dependency2.jar
    +-lib-provided
        +-servlet-api.jar
        +-dependency3.jar
```

依赖项应该放在嵌套的 `WEB-INF/lib` 目录中。运行嵌入式时需要但部署到传统 Web 容器时不需要的任何依赖项都应放在 `WEB-INF/lib-provided`.

1.3. 索引文件

Spring Boot Loader 兼容的 jar 和 war 档案可以在 `BOOT-INF/`目录下包含额外的索引文件。 jars 和 wars 都可以有 `classpath.idx` 文件，它提供了 jars 应该被添加到 classpath 的顺序。`layers.idx` 文件只能用于 jar，它允许将 jar 拆分为逻辑层以创建 Docker/OCI 映像。

索引文件遵循兼容 YAML 的语法，以便第三方工具可以轻松解析它们。但是，在内部这些文件不会被解析为 YAML，它们必须完全按照下面描述的格式编写才能使用。

1.4. Classpath 索引

classpath 索引文件可以在 `BOOT-INF/classpath.idx`， 它提供了一个 jar 名称列表（包括目录），按照它们应该被添加到 classpath 中的顺序。每行必须以破折号空格（`"- "`）开头，名称必须用双引号括起来。

例如，给定以下 jar：

```
example.jar
 |
 +-META-INF
 |  +-...
 +-BOOT-INF
    +-classes
    |  +-...
    +-lib
        +-dependency1.jar
        +-dependency2.jar
```

索引文件如下所示：

```
- "BOOT-INF/lib/dependency2.jar"
- "BOOT-INF/lib/dependency1.jar"
```

1.5. Layer 索引

Layer 索引文件可以在 BOOT-INF/layers.idx。 它提供了一个层列表以及应该包含在其中的 jar 部分。层是按照应该添加到 Docker/OCI 镜像的顺序编写的。Layer 名称是带双引号的字符串，前缀为破折号空格（"- "），后缀为冒号（":"）。Layer 内容是一个文件名或目录名，是以空格空格破折号空格("   - ")作为前缀的带双引号的字符串 。目录名以 / 结尾，文件名不用。当使用目录名称时，意味着该目录中的所有文件都在同一层中。

Layer 索引的典型示例是：

```
- "dependencies":

  - "BOOT-INF/lib/dependency1.jar"

  - "BOOT-INF/lib/dependency2.jar"

- "application":

  - "BOOT-INF/classes/"

  - "META-INF/"
```

2. Spring Boot 的 JarFile 类

用于支持加载嵌套 jar 的核心类是 org.springframework.boot.loader.jar.JarFile. 它允许您从标准 jar 文件或嵌套的子 jar 数据中加载 jar 的内容。首次加载时，每个 JarEntry 的位置都被映射到外部 jar 的物理文件偏移量，如下例所示：

```
我的应用程序

+------------------------------+-----------------------+

| /BOOT-INF/classes | /BOOT-INF/lib/mylib.jar |

|+-----------------+||+-----------+---------+|

|| A.class ||| B.class | C.class ||

|+-----------------+||+-----------+---------+|

+------------------------------+-----------------------+
```

```
^ ^ ^

0063  3452  3980
```

上面的示例显示了如何在 myapp.jar 的 /BOOT-INF/classes 中找到位置为 0063 的 A.class。 嵌套 jar 的 B.class 实际上可以在 myapp.jar 的 3452 位置处找到，C.class 在 3980 处。

有了这些信息，我们就可以通过寻找外部 jar 的适当部分来加载特定的嵌套条目。我们不需要解压存档，也不需要将所有条目数据读入内存。

2.1. 与标准 Java JarFile 的兼容性

Spring Boot Loader 努力保持与现有代码和库的兼容。 org.springframework.boot.loader.jar.JarFile 继承自 java.util.jar.JarFile 并应作为替代品。getURL() 方法返回一个 URL ，其打开一个与 java.net.JarURLConnection 兼容的连接，并可与 Java 的 URLClassLoader 一起使用。

3. 启动可执行的 jars

org.springframework.boot.loader.Launcher 是作为一个可执行 JAR 的主入口点一个特殊的启动类。它是 jar 文件中的实际 Main-Class，用于设置适当的 URLClassLoader 并最终调用您的 main() 方法。

有三个 Launcher 子类（JarLauncher，WarLauncher，和 PropertiesLauncher）。它们的目的是从嵌套的 jar 文件或目录中的 war 文件中加载资源（.class 文件等），而不是显式地在 classpath 上加载资源。在 JarLauncher 和 WarLauncher 的情况下，嵌套路径是固定的。

- JarLauncher ：在 BOOT-INF/lib/ 中查找。
- WarLauncher 在 WEB-INF/lib/ 和 WEB-INF/lib-provided/ 查找。
- PropertiesLauncher：默认情况下，在 BOOT-INF/lib/ 中查找。您可以添加其他位置，通过设置名为 LOADER_PATH 的环境变量，或在 loader.properties 文件中设置 loader.path 属性 （这是一个以逗号分隔的目录、档案或档案中的目录的列表）。

3.1. Launcher Manifest

您需要指定一个适当的 `Launcher` 作为 的 `META-INF/MANIFEST.MF` 文件的 `Main-Class` 属性。您要启动的实际类（即包含 `main` 方法的应用程序类）应在 `Start-Class` 属性中指定。

以下示例显示了一个典型的可执行 jar 文件的 `MANIFEST.MF`：

```
Main-Class: org.springframework.boot.loader.JarLauncher

Start-Class: com.mycompany.project.MyApplication
```

对于 war 文件，它将如下所示：

```
Main-Class: org.springframework.boot.loader.WarLauncher

Start-Class: com.mycompany.project.MyApplication
```

4. PropertiesLauncher 功能

`PropertiesLauncher` 有一些可以通过外部属性（系统属性、环境变量、manifest entries 或 `loader.properties` 文件）启用的特殊功能。下表描述了这些属性：

Key	Purpose
`loader.path`	逗号分隔的类路径，例如 `lib,${HOME}/app/lib`。 前面的条目优先，类似于 `javac` 命令行 的`-classpath`。
`loader.home`	用于解析 `loader.path` 中的相对路径。 例如，给定 `loader.path=lib`，则 `${loader.home}/lib` 是一个 `classpath` 位置（以及该目录中的所有 `jar` 文件）。此属性也用于定位 `loader.properties` 文件，如下例中`/opt/app` 默认为 `${user.dir}`。
`loader.args`	`main` 方法的默认参数（空格分隔）。
`loader.main`	要启动的主类的名称（例如，`com.app.Application`）。
`loader.config.name`	属性文件的名称（例如，`launcher`）。默认为 `loader`。

Key	Purpose
loader.config.location	属性文件的路径（例如，classpath:loader.properties）。默认为 loader.properties。
loader.system	布尔标志，指示应将所有属性添加到系统属性。默认为 false。

当指定为环境变量或 manifest entries 时，应使用以下名称：

Key	Manifest entry	Environment variable
loader.path	Loader-Path	LOADER_PATH
loader.home	Loader-Home	LOADER_HOME
loader.args	Loader-Args	LOADER_ARGS
loader.main	Start-Class	LOADER_MAIN
loader.config.location	Loader-Config-Location	LOADER_CONFIG_LOCATION
loader.system	Loader-System	LOADER_SYSTEM

Tip

构建插件会在构建 fat jar 时自动将 Main-Class 属性移动到 Start-Class。如果使用它，请使用 Main-Class 属性指定要启动的类的名称并省略 Start-Class。

以下规则适用于使用 PropertiesLauncher：

- 首先在 loader.home 中搜索 loader.properties，然后在 classpath 的根中搜索，然后在 classpath:/BOOT-INF/classes 中。 使用最先找到的文件。
- loader.home 是仅当 loader.config.location 未指定时附加属性文件的目录位置（覆盖默认值）。
- loader.path 可以包含目录（对 jar 和 zip 文件进行递归扫描）、archive 路径、archive 中扫描 jar 文件的目录（例如，dependencies.jar!/lib）或通配符模式（对于默认的 JVM 行为）。archive 路径可以相对于 loader.home。或者带有 jar:file:前缀的文件系统中的任何位置。
- loader.path（如果为空）默认为 BOOT-INF/lib（意味着本地目录，或嵌套目录，如果从存档运行）。因此，未提供其他配置的 PropertiesLauncher 的行为与 JarLauncher 的行为相同。
- loader.path 不能用于配置 loader.properties 的位置（用于搜索后者的 classpath 是 PropertiesLauncher 启动时的 JVM classpath ）。

- 在使用所有值之前，从系统和环境变量加上属性文件本身进行占位符替换。
- 属性的搜索顺序（在多个地方查看是有意义的）是环境变量、系统属性、loader.properties、分解的 archive manifest 和 archive manifest。

可执行 Jar 的限制

在使用 Spring Boot Loader 打包的应用程序时，您需要考虑以下限制：

- Zip entry compression：必须使用 ZipEntry.STORED 方法保存嵌套 jar 的 ZipEntry。这是必需的，以便我们可以直接查找嵌套 jar 中的单个内容。嵌套 jar 文件本身的内容仍然可以压缩，就像外部 jar 中的任何其他条目一样。

- System classLoader：启动的应用程序在加载类时应该使用 Thread.getContextClassLoader()（大多数库和框架默认是这样做的）。尝试使用 ClassLoader.getSystemClassLoader() 加载嵌套的 jar 会失败。 java.util.Logging 始终使用系统类加载器。因此，您应该考虑不同的日志记录实现。

6. 其他的单个 Jar 的解决方案

如果上述限制意味着您无法使用 Spring Boot Loader，请考虑以下替代方案：

- Maven Shade Plugin
- JarClassLoader
- OneJar
- Gradle Shadow Plugin

原文

- The Executable Jar Format