## 整体说明

Maven的Assembly插件用于将项目输出及其依赖包，模块，站点文档和其他文件聚合到一个可发布的归档文件中。

可以使用预制装配描述符轻松地构建分布“组件”。 这些描述符处理许多常见操作，例如将项目的工件与生成的文档一起打包到一个zip压缩文件中。 也可以提供自己的描述符，对程序集中的依赖关系，模块，文件集和单个文件的打包方式进行更高级别的控制。

目前可以创建以下格式的发布包：

* zip
* tar
* tar.gz (or tgz)
* tar.bz2 (or tbz2)
* tar.snappy
* tar.xz (or txz)
* jar
* dir
* war

如果想将项目封装在超级jar中，则assembly插件仅提供基本的支持。想进行更多控制，查看[Maven Shade插件](http://maven.apache.org/plugins/maven-shade-plugin/)。

要在Maven中使用Assembly插件，需要下面三步：

* 选择或编写要使用的assembly描述符
* 在项目的pom.xml中配置Assembly插件
* 在项目上运行“mvn assembly：single”

要编写自定义程序集，需要参考[程序集描述符格式](http://maven.apache.org/plugins/maven-assembly-plugin/assembly.html)。

## Assembly(程序集)是什么

“程序集”是一组文件，目录和依赖关系集合，它们被组合成一个存档格式并发布。

Assembly插件提供了一个描述符格式，它允许你任意组装一个项目中的文件和目录，成一个单独的程序集。 例如，假设Maven项目包含“src/main/bin”目录，则可以指示Assembly插件将此目录的内容复制到程序集的“bin”目录，并更改该目录中文件的权限为UNIX模式755.

装配描述符配置了该行为的相关参数并提供给Assembly插件。

## Goals:目标

Assembly插件的主要目标是[single](http://maven.apache.org/plugins/maven-assembly-plugin/single-mojo.html)。 可用于创建所有程序集。

1.single目标

从描述符中组装应用程序包或发布包。 这个目标可以绑定到maven生命周期，或者直接从命令行调用。

Remark：descriptors, descriptorRefs和descriptorSourceDirectory三个参数不相交的。

特性：

* 需要一个maven程序
* 需要在编译阶段解析依赖
* 该目标是线程安全的，支持并行构建
* 在多项目构建中默认情况下不会被继承

必备的参数:

|  |  |  |
| --- | --- | --- |
| 参数名称 | 类型 | 描述 |
| outputDirectory | File | 发布文件的输出目录，默认是：  ${project.build.directory} |
| workDirectory | File | Jar包的解压目录(如果需要),默认为  ${project.build.directory}/assembly/work. |

可选的参数:

|  |  |  |
| --- | --- | --- |
| 参数名称 | 类型 | 描述 |
| appendAssemblyId | boolean | 是否包含AssemblyId到最后的输出文件名中，默认值是true |
| archive | MavenArchiveConfiguration | 对存档生成器的一系列指令，特别是用于构建.jar文件。可以为jar指定一个Manifest文件请参阅[Maven Archiver](http://maven.apache.org/shared/maven-archiver/index.html)。 |
| archiveBaseDirectory |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

<http://maven.apache.org/plugins/maven-assembly-plugin/single-mojo.html>

## Usage:用法

<http://maven.apache.org/plugins/maven-assembly-plugin/usage.html>

1.配置插件

如果要使用预定义的装配描述符，可以通过<descriptorRefs>和<descriptorRef>进行配置，使用自定义的装配描述符，可以通过<descriptors>和<descriptor> 进行配置：

Assembly插件的单个调用实际上可以从多个描述符生成程序集，从而可以生成多个程序集。 创建程序集时，将使用assemblyId作为工件的分类器，并将创建的程序集附加到项目中，以便在安装和部署阶段将其上载到存储库中。

例如，假设项目产生一个JAR。 如果想创建一个包含我们项目依赖关系的程序集二进制文件，可以利用Assembly插件的一个预制描述符: jar-with-dependencies。

<**plugin**>  
 <**artifactId**>maven-assembly-plugin</**artifactId**>  
 <**version**>3.1.0</**version**>  
 <**configuration**>  
 <**descriptorRefs**>  
 <**descriptorRef**>jar-with-dependencies</**descriptorRef**>  
 </**descriptorRefs**>  
 </**configuration**>  
</**plugin**>

也可以在src/main/build目录下创建自己的描述符文件: package.xml,然后指定它：

<**plugin**>  
 <**artifactId**>maven-assembly-plugin</**artifactId**>  
 <**version**>3.1.0</**version**>  
 <**configuration**>  
 <**descriptors**>  
 <**descriptor**>src/main/build/package.xml</**descriptor**>  
 </**descriptors**>

</**configuration**>  
</**plugin**>

更多的配置参数：[here](http://maven.apache.org/guides/mini/guide-configuring-plugins.html)

2. Execution:构建一个程序集

配置好了descriptors 和descriptorRefs之后，就可以构建它们了。

在大多数情况下，需要确保程序集是作为常规构建过程的一部分。 这可以确保程序集可用于安装和部署，并且在项目发布期间创建。 这是由assembly:single处理的。

<**plugin**>  
 <**artifactId**>maven-assembly-plugin</**artifactId**>  
 <**version**>3.1.0</**version**>  
 <**configuration**>  
 <**descriptorRefs**>  
 <**descriptorRef**>jar-with-dependencies</**descriptorRef**>  
 </**descriptorRefs**>  
 <**executions**>  
 <**execution**>  
 *<!-- 用于继承合并 -->* <**id**>make-assembly</**id**>  
 *<!-- 绑定到package阶段 -->* <**phase**>package</**phase**>  
 <**goals**>  
 <**goal**>single</**goal**>  
 </**goals**>  
 </**execution**>  
 </**executions**>  
 </**configuration**>  
</**plugin**>

做好上面的配置之后，可以通过mvn package,便能进行打包了。

生成的jar包名称会带jar-with-dependencies后缀，这是装配描述符的ID。

Remark：在大多数情况下，single目标应该绑定到构建的package阶段。 但是，如果程序集不需要二进制文件，或者如果需要使用一个程序集作为另一个程序集的输入，则可能需要更改此程序集。 尽管可以将single目标分配到构建生命周期的任何阶段，但在创建程序集之前，应该小心确保包含在程序集中的资源已经存在。

3. 创建一个可执行的jar

<**configuration**>  
 *<!-- 配置启动类 -->* <**archive**>  
 <**manifest**>  
 <**mainClass**>org.sample.App</**mainClass**>  
 </**manifest**>  
 </**archive**>

</**configuration**>

## 预定义的描述符

<http://maven.apache.org/plugins/maven-assembly-plugin/descriptor-refs.html>

Assembly插件提供了四个内置的描述符，分别是：bin、jar-with-dependencies、src、project

1.bin

bin将项目编译后的二进制文件打包成tar.gz、tar.bz2、zip三种压缩格式，并且包含项目根路径下的README, LICENSE,和NOTICE文件(如果有的话)。

下面是bin描述符的配置内容：

<**assembly xmlns="http://maven.apache.org/ASSEMBLY/2.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/ASSEMBLY/2.0.0  
 http://maven.apache.org/xsd/assembly-2.0.0.xsd"**>  
 <**id**>bin</**id**>  
 <**formats**>  
 <**format**>tar.gz</**format**>  
 <**format**>tar.bz2</**format**>  
 <**format**>zip</**format**>  
 </**formats**>  
 <**fileSets**>  
 <**fileSet**>  
 <**directory**>${project.basedir}</**directory**>  
 <**outputDirectory**>/</**outputDirectory**>  
 <**includes**>  
 <**include**>README\*</**include**>  
 <**include**>LICENSE\*</**include**>  
 <**include**>NOTICE\*</**include**>  
 </**includes**>  
 </**fileSet**>  
 <**fileSet**>  
 <**directory**>${project.build.directory}</**directory**>  
 <**outputDirectory**>/</**outputDirectory**>  
 <**includes**>  
 <**include**>\*.jar</**include**>  
 </**includes**>  
 </**fileSet**>  
 <**fileSet**>  
 <**directory**>${project.build.directory}/site</**directory**>  
 <**outputDirectory**>docs</**outputDirectory**>  
 </**fileSet**>  
 </**fileSets**>  
</**assembly**>

2. jar-with-dependencies

jar-with-dependencies除了自己项目编译后的二进制文件，还有将项目的所有依赖进行解压，然后重新打包成一个jar。

下面是jar-with-dependencies描述符的配置内容：

<**assembly xmlns="http://maven.apache.org/ASSEMBLY/2.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/ASSEMBLY/2.0.0  
 http://maven.apache.org/xsd/assembly-2.0.0.xsd"**>  
 *<!--* ***TODO: a jarjar format would be better*** *-->* <**id**>jar-with-dependencies</**id**>  
 <**formats**>  
 <**format**>jar</**format**>  
 </**formats**>  
 <**includeBaseDirectory**>false</**includeBaseDirectory**>  
 <**dependencySets**>  
 <**dependencySet**>  
 <**outputDirectory**>/</**outputDirectory**>  
 <**useProjectArtifact**>true</**useProjectArtifact**>  
 <**unpack**>true</**unpack**>  
 <**scope**>runtime</**scope**>  
 </**dependencySet**>  
 </**dependencySets**>  
</**assembly**>

3. src

Src描述符将项目的/src路基下的内容，打包成tar.gz、tar.bz2、zip三种压缩格式，具体的配置内容如下：

<**assembly xmlns="http://maven.apache.org/ASSEMBLY/2.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/ASSEMBLY/2.0.0  
 http://maven.apache.org/xsd/assembly-2.0.0.xsd"**>  
 <**id**>src</**id**>  
 <**formats**>  
 <**format**>tar.gz</**format**>  
 <**format**>tar.bz2</**format**>  
 <**format**>zip</**format**>  
 </**formats**>  
 <**fileSets**>  
 <**fileSet**>  
 <**directory**>${project.basedir}</**directory**>  
 <**includes**>  
 <**include**>README\*</**include**>  
 <**include**>LICENSE\*</**include**>  
 <**include**>NOTICE\*</**include**>  
 <**include**>pom.xml</**include**>  
 </**includes**>  
 <**useDefaultExcludes**>true</**useDefaultExcludes**>  
 </**fileSet**>  
 <**fileSet**>  
 <**directory**>${project.basedir}/src</**directory**>  
 <**useDefaultExcludes**>true</**useDefaultExcludes**>  
 </**fileSet**>  
 </**fileSets**>  
</**assembly**>

4.project

project描述符会将整个项目打包成一个程序集，不包括/target目录下的文件。同样会打包成tar.gz、tar.bz2、zip三种压缩格式，具体的配置内容如下：

<**assembly xmlns="http://maven.apache.org/ASSEMBLY/2.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/ASSEMBLY/2.0.0  
 http://maven.apache.org/xsd/assembly-2.0.0.xsd"**>  
 <**id**>project</**id**>  
 <**formats**>  
 <**format**>tar.gz</**format**>  
 <**format**>tar.bz2</**format**>  
 <**format**>zip</**format**>  
 </**formats**>  
 <**fileSets**>  
 <**fileSet**>  
 <**directory**>${project.basedir}</**directory**>  
 <**outputDirectory**>/</**outputDirectory**>  
 <**useDefaultExcludes**>true</**useDefaultExcludes**>  
 <**excludes**>  
 <**exclude**>\*\*/\*.log</**exclude**>  
 <**exclude**>\*\*/${project.build.directory}/\*\*</**exclude**>  
 </**excludes**>  
 </**fileSet**>  
 </**fileSets**>  
</**assembly**>

## 描述符的格式

assembly descriptors的作用:Mavem Assembly插件会根据描述符的配置执行打包操作，描述符指定要创建的程序集归档的类型，程序集的内容以及依赖项或其模块与程序集捆绑在一起的方式。

描述符的整体格式如下，常用的属性附上了相应解释：

<assembly xmlns="http://maven.apache.org/ASSEMBLY/2.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/ASSEMBLY/2.0.0  
 http://maven.apache.org/xsd/assembly-2.0.0.xsd">  
 <!-- String类型，assembly的ID，发布的时候用作分类器，可以将其附加在发布后的打包文件名字后 -->  
 <id/>  
 <!-- 打包的类型，可选的值有：zip、tar、tar.gz、tar.bz2、jar、dir、war -->  
 <formats/>  
 <!-- 是否在打包文件中包含基目录，默认true -->  
 <includeBaseDirectory/>  
 <!-- 设置打包后的基目录，若该值没设置并且 includeBaseDirectory为true，  
 则使用${project.build.finalName}作为配置值-->  
 <baseDirectory/>  
 <!-- 是否在打包文件中包含站点目录，默认false -->  
 <includeSiteDirectory/>  
 <!-- -->  
 <containerDescriptorHandlers>  
 <containerDescriptorHandler>  
 <handlerName/>  
 <configuration/>  
 </containerDescriptorHandler>  
 </containerDescriptorHandlers>  
 <!-- -->  
 <moduleSets>  
 <moduleSet>  
 <useAllReactorProjects/>  
 <includeSubModules/>  
 <includes/>  
 <excludes/>  
 <sources>  
 <useDefaultExcludes/>  
 <outputDirectory/>  
 <includes/>  
 <excludes/>  
 <fileMode/>  
 <directoryMode/>  
 <fileSets>  
 <fileSet>  
 <useDefaultExcludes/>  
 <outputDirectory/>  
 <includes/>  
 <excludes/>  
 <fileMode/>  
 <directoryMode/>  
 <directory/>  
 <lineEnding/>  
 <filtered/>  
 </fileSet>  
 </fileSets>  
 <includeModuleDirectory/>  
 <excludeSubModuleDirectories/>  
 <outputDirectoryMapping/>  
 </sources>  
 <binaries>  
 <outputDirectory/>  
 <includes/>  
 <excludes/>  
 <fileMode/>  
 <directoryMode/>  
 <attachmentClassifier/>  
 <includeDependencies/>  
 <dependencySets>  
 <dependencySet>  
 <outputDirectory/>  
 <includes/>  
 <excludes/>  
 <fileMode/>  
 <directoryMode/>  
 <useStrictFiltering/>  
 <outputFileNameMapping/>  
 <unpack/>  
 <unpackOptions>  
 <includes/>  
 <excludes/>  
 <filtered/>  
 <lineEnding/>  
 <useDefaultExcludes/>  
 <encoding/>  
 </unpackOptions>  
 <scope/>  
 <useProjectArtifact/>  
 <useProjectAttachments/>  
 <useTransitiveDependencies/>  
 <useTransitiveFiltering/>  
 </dependencySet>  
 </dependencySets>  
 <unpack/>  
 <unpackOptions>  
 <includes/>  
 <excludes/>  
 <filtered/>  
 <lineEnding/>  
 <useDefaultExcludes/>  
 <encoding/>  
 </unpackOptions>  
 <outputFileNameMapping/>  
 </binaries>  
 </moduleSet>  
 </moduleSets>  
 <!-- 指定在程序集中包含哪些目录 -->  
 <fileSets>  
 <fileSet>  
 <useDefaultExcludes/>  
 <!-- 设置输出目录相对于程序集根目录的目录 -->  
 <outputDirectory/>  
 <!-- 指定需要包含的文件 -->  
 <includes/>  
 <!-- 指定需要排除的文件 -->  
 <excludes/>  
 <!-- 设置文件的模式，与UNIX权限类似Read = 4，Write = 2和Execute = 1，默认是0644 -->  
 <fileMode/>  
 <!-- 设置目录的模式，与UNIX权限类似Read = 4，Write = 2和Execute = 1，默认是0755 -->  
 <directoryMode/>  
 <!-- 指定要包含的目录 -->  
 <directory/>  
 <!-- 设置文件的行结束符，可选的值有：keep(保持原有换行符)、unix、lf(\n换行符)、dos、windows、crlf(\r\n换行符) -->  
 <lineEnding/>  
 <!-- 是否在复制文件时过滤符号，默认值false -->  
 <filtered/>  
 </fileSet>  
 </fileSets>  
 <!-- 指定在程序集中包含哪些单个文件 -->  
 <files>  
 <file>  
 <!-- 设置要包含的文件的绝对路径或相对路径 -->  
 <source/>  
 <!-- 设置输出目录相对于程序集根目录的目录 -->  
 <outputDirectory/>  
 <!-- 设置目标文件名。 默认是与源文件相同的名称。 -->  
 <destName/>  
 <!-- 设置文件的模式，与UNIX权限类似Read = 4，Write = 2和Execute = 1，默认是0644 -->  
 <fileMode/>  
 <!-- 设置文件的行结束符，可选的值有：keep(保持原有换行符)、unix、lf(\n换行符)、dos、windows、crlf(\r\n换行符) -->  
 <lineEnding/>  
 <!-- 是否在复制文件时过滤符号，默认值false -->  
 <filtered/>  
 </file>  
 </files>  
 <!-- 指定要包含在程序集中的依赖关系 -->  
 <dependencySets>  
 <dependencySet>  
 <!-- 设置输出目录相对于程序集根目录的目录 -->  
 <outputDirectory/>  
 <!-- 设置包含的jar包坐标，格式为：groupId：artifactId  
 或groupId：artifactId：type [：classifier]：version  
 或使用\*通配符 -->  
 <includes/>  
 <!-- 设置排除的jar包坐标 -->  
 <excludes/>  
 <!-- 设置文件的模式，与UNIX权限类似Read = 4，Write = 2和Execute = 1，默认是0644 -->  
 <fileMode/>  
 <!-- 设置目录的模式，与UNIX权限类似Read = 4，Write = 2和Execute = 1，默认是0755 -->  
 <directoryMode/>  
 <!-- 是否使用严格过滤模式，默认false -->  
 <useStrictFiltering/>  
 <!-- 设置此程序集中包含的所有依赖关系的映射模式 -->  
 <outputFileNameMapping/>  
 <!-- 是否解压依赖包到输出目录，默认false -->  
 <unpack/>  
 <!-- 对解压进行进一步配置 -->  
 <unpackOptions>  
 <includes/>  
 <excludes/>  
 <filtered/>  
 <lineEnding/>  
 <useDefaultExcludes/>  
 <encoding/>  
 </unpackOptions>  
 <!-- 设置依赖项范围，默认runtime -->  
 <scope/>  
 <!-- 确定当前项目构建产生的发布包是否包含在这个依赖集中，默认true -->  
 <useProjectArtifact/>  
 <!-- 确定在当前项目的构建产生的附加构件是否包含在这个依赖集中，默认false -->  
 <useProjectAttachments/>  
 <!-- 是否使用传递依赖，默认true -->  
 <useTransitiveDependencies/>  
 <!-- 是否按排除/包含模式过滤传递依赖，默认false -->  
 <useTransitiveFiltering/>  
 </dependencySet>  
 </dependencySets>  
 <!-- 指定要在程序集中包含哪些存储库文件 -->  
 <repositories>  
 <repository>  
 <!-- 设置输出目录相对于程序集根目录的目录 -->  
 <outputDirectory/>  
 <!-- 设置包含的jar包坐标，格式为：groupId：artifactId  
 或groupId：artifactId：type [：classifier]：version  
 或使用\*通配符 -->  
 <includes/>  
 <!-- 设置排除的jar包坐标 -->  
 <excludes/>  
 <!-- 设置文件的模式，与UNIX权限类似Read = 4，Write = 2和Execute = 1，默认是0644 -->  
 <fileMode/>  
 <!-- 设置目录的模式，与UNIX权限类似Read = 4，Write = 2和Execute = 1，默认是0755 -->  
 <directoryMode/>  
 <!-- 是否包含源数据，默认false -->  
 <includeMetadata/>  
 <groupVersionAlignments>  
 <groupVersionAlignment>  
 <id/>  
 <version/>  
 <excludes/>  
 </groupVersionAlignment>  
 </groupVersionAlignments>  
 <!-- 设置依赖范围，默认runtime -->  
 <scope/>  
 </repository>  
 </repositories>  
 <!-- 指定要包含在程序集中的共享组件xml文件位置 -->  
 <componentDescriptors/>  
</assembly>

## Examples:例子

<http://maven.apache.org/plugins/maven-assembly-plugin/examples/index.html>