


将jar文件加到Maven的local repository中


对于Maven项目来说，日常使用的多数第三方java库文件都可以从Maven的Central Repository中自动下载，但是如果我们需要的jar文件不在Central Repository中，那么我们就需要手动将自己下载的jar文件加入到Maven的local repository中了，此时我们需要向Maven提供用于识别jar文件（可能多个）的groupId, artifactId和version等信息。



我并不打算讲怎么将一个下载的jar库加入到local repository中，我们将自己建立一个jar库，比如我们有一个最简单的HelloWorld类HelloWorld.java：

```
package com.thoughtworks.davenkin;

public class HelloWorld
{
    public void sayHello()
    {
        System.out.println("Hello, World");
    }
}
```



我们希望将HelloWorld.java打包成jar文件安装在Maven的local repository中以便其它程序使用。

编译打包hello-world.jar后，为了符合Maven的规定，需要给hello-world.jar一个版本号，故将hello-world.jar改名为hello-world-1.0.jar，此后便可以用mvn来安装此包到Maven的local repository中了：

```
mvn    install:install-file    -Dfile=path/to/hello-world-1.0.jar    -
DgroupId=com.thoughtworks.davenkin    -DartifactId=hello-world    -
Dversion=1.0 -Dpackaging=jar
```

其中，-Dfile选项应给出需要安装jar文件的路径，在Linux/Mac下，jar文件将被安装在以下目录：

```
~/m2/repository/com/thoughtworks/davenkin/hello-world/1.0/hello-world-1.0.jar
```


现在，我们的hello-world-1.0.jar便可以被其它Maven项目所使用了，为此创建一个Maven工程：

```
mvn archetype:generate -DgroupId=com.thoughtworks.davenkin.demo -DartifactId=helloworld-demo -DarchetypeArtifactId=maven-archetype-quickstart -Dversion=1.0
```

此时将在当前目录下自动创建helloworld-demo子目录，切换到helloworld-demo目录，删除已有的App.java，并创建自己的Main.java文件

```
rm src/main/java/com/thoughtworks/davenkin/demo/App.java
touch src/main/java/com/thoughtworks/davenkin/demo/Main.java
```


将以下内容加入到Main.java文件中：




```
package com.thoughtworks.davenkin.demo;

import com.thoughtworks.davenkin>HelloWorld;

public class Main
{
    public static void main(String[] args)
    {
        new HelloWorld().sayHello();
    }
}
```



接下来是最重要的一步，修改pom.xml文件以加入对HelloWorld类的依赖：



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
<groupId>com.thoughtworks.davenkin.demo</groupId>
<artifactId>demo</artifactId>
<version>1.0</version>
<packaging>jar</packaging>

<name>demo</name>
<url>http://maven.apache.org</url>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>com.thoughtworks.davenkin</groupId>
    <artifactId>hello-world</artifactId>
    <version>1.0</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
</project>
```



以上高亮部分为我们手动加入的，Maven会根据artifactId和version拼出所依赖jar包的名字，即artifactId-version.jar，对于我们的例子，artifactId为hello-world，version为1.0，所得到的jar文件为hello-world-1.0.jar，这也是为什么我们在一开始时需要将hello-world.jar的名字改为hello-world-1.0.jar的原因。

接下来便可以编译我们的Main.java了：

```
mvn compile
```

编译结果会放在target文件夹下。

到现在，我们的例子便可以运行了，在工程根目录下（该例为hello-world-demo）输入以下命令：

```
mvn exec:java -Dexec.mainClass="com.thoughtworks.davenkin.demo.Main"
```

在笔者的机器上输出为：



```
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building demo 1.0
[INFO] -----
[INFO]
[INFO] >>> exec-maven-plugin:1.2.1:java (default-cli) @ demo >>>
[INFO]
[INFO] <<< exec-maven-plugin:1.2.1:java (default-cli) @ demo <<<
[INFO]
[INFO] --- exec-maven-plugin:1.2.1:java (default-cli) @ demo ---
Hello, World
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.423s
[INFO] Finished at: Wed Feb 15 21:33:47 CST 2012
[INFO] Final Memory: 5M/81M
[INFO] -----
-----
```



以上高亮部分即为我们期望的程序输出，当然你也可以用传统的java命令来运行，此时需要将hello-world-1.0.jar加入到classpath中，输入：

```
java -cp ~/.m2/repository/com/thoughtworks/davenkin/hello-world/1.0/hello-world-1.0.jar:target/classes/com.thoughtworks.davenkin.demo.Main
```

输出为：

```
Hello, World
```