

Maven学习总结(三)——使用Maven构建项目 - 孤傲苍狼

maven作为一个高度自动化构建工具，本身提供了构建项目的功能，下面就来体验一下使用maven构建项目的过程。

一、构建Java项目

1.1、创建Java Project

1、使用mvn archetype:generate命令，如下所示：

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=myapp -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

2、使用mvn archetype:create命令，如下所示：

```
mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=myapp -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

使用“mvn archetype:generate”命令和“mvn archetype:create”都可以创建项目，目前没有发现这两者的区别，唯一区别的地方就是发现使用“mvn archetype:generate”命令创建项目时要特别长的时间才能够将项目创建好，而使用“mvn archetype:create”命令则可以很快将项目创建出来。使用“mvn archetype:create”命令创建一个java项目的过程如下图所示：

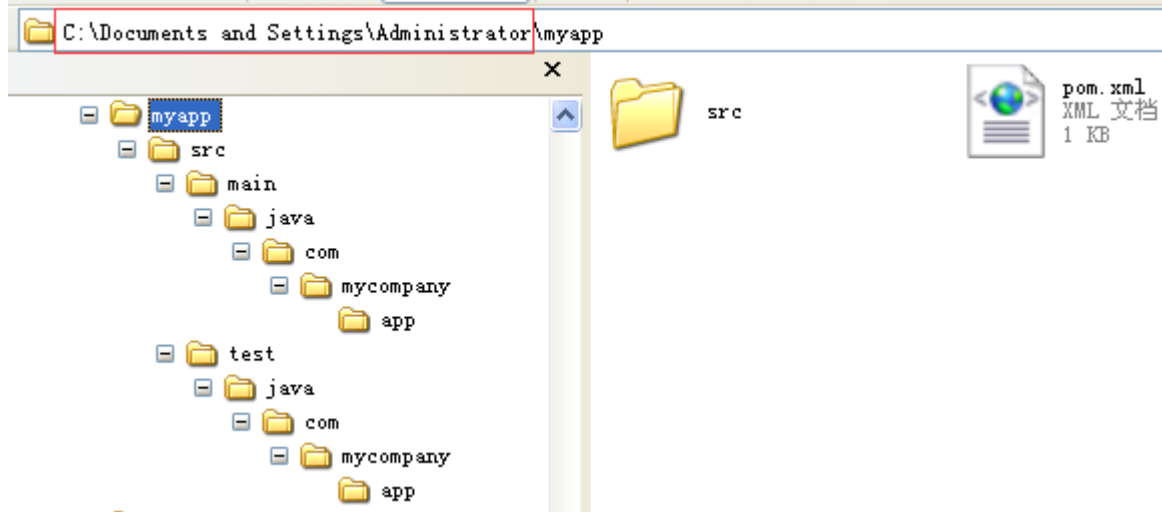
```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator>mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=myapp -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
[INFO]
[INFO] --- maven-archetype-plugin:2.2:create (default-cli) @ standalone-pom ---
[WARNING] This goal is deprecated. Please use mvn archetype:generate instead
[INFO] Defaulting package to group ID: com.mycompany.app
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/maven-metadata.xml
Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/maven-metadata.xml (531 B at 0.1 KB/sec)
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.1/maven-archetype-quickstart-1.1.jar
Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.1/maven-archetype-quickstart-1.1.jar (7 KB at 12.1 KB/sec)
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.1/maven-archetype-quickstart-1.1.pom
Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.1/maven-archetype-quickstart-1.1.pom (2 KB at 3.7 KB/sec)
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-quickstart:RELEASE
[INFO] -----
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: packageName, Value: com.mycompany.app
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: myapp
[INFO] Parameter: basedir, Value: C:\Documents and Settings\Administrator
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Documents and Settings\Administrator\myapp
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.813 s
[INFO] Finished at: 2015-01-22T09:38:26+08:00
[INFO] Final Memory: 11M/26M
[INFO] -----
C:\Documents and Settings\Administrator>
```

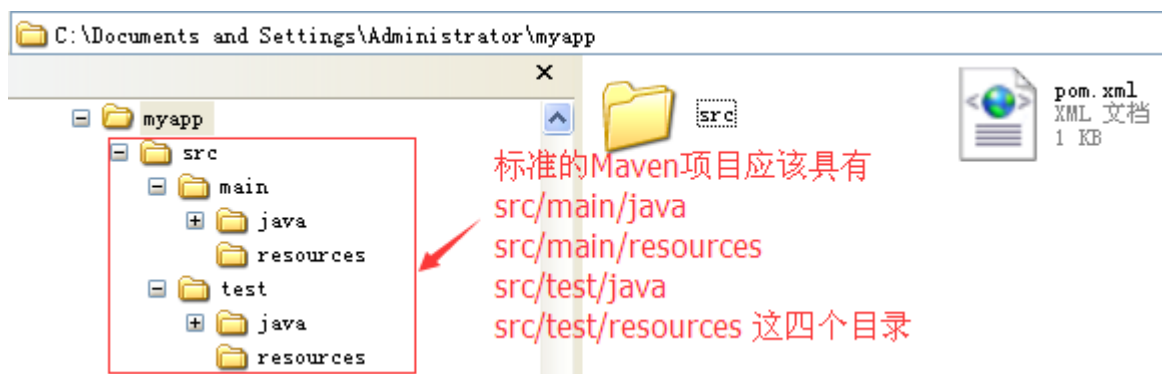
生成的项目默认是存放在当前登录操作系统的用户的目标下

BUILD SUCCESS就表示项目构建成功，当前用户目录下（即C:\Documents and Settings\Administrator）下构建了一个Java Project叫做myapp。

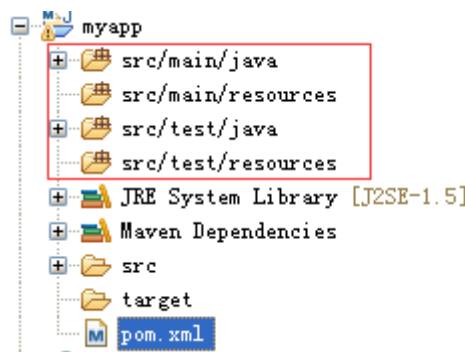
构建好的Java项目的目录结构如下：



可以看到，Maven帮我们创建的项目是一个标准的Maven项目，不过目前Maven只是帮我们生成了src/main/java(存放项目的源代码)和src/test/java(存放测试源代码)这两个目录，但实际项目开发中我们一般都会有配置文件，例如log4j.properties，所以我们还需要手动创建src/main/resources(存放项目开发中用到的配置文件，如存放log4j.properties等)和src/test/resources(存放测试时用到的配置文件)，如下图所示：

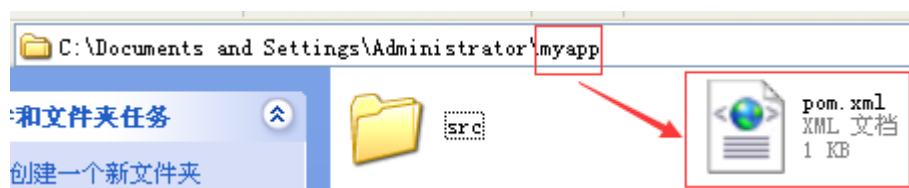


然后我们就可以将创建好的myapp项目导入到Eclipse中进行开发了，如下图所示：



1.2、JavaProject的pom.xml文件说明

通过Maven构建的JavaProject，在项目的根目录下都会存在一个pom.xml文件，进入myapp目录，可以看到有一个pom.xml文件，这个文件是Maven的核心。如下图所示：



- 1、pom意思就是project object model。
 - 2、pom.xml包含了项目构建的信息，包括项目的信息、项目的依赖等。
 - 3、pom.xml文件是可以继承的，大型项目中，子模块的pom.xml一般都会继承于父模块的pom.xml
- pom.xml文件的内容如下：



```

1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4
5   <groupId>com.mycompany.app</groupId>
6   <artifactId>myapp</artifactId>
7   <version>1.0-SNAPSHOT</version>
8   <packaging>jar</packaging>
9
10  <name>myapp</name>
11  <url>http://maven.apache.org</url>
12
13  <properties>
14    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15  </properties>
16
17  <dependencies>
18    <dependency>
19      <groupId>junit</groupId>
20      <artifactId>junit</artifactId>
21      <version>3.8.1</version>
22      <scope>test</scope>
23    </dependency>
24  </dependencies>
25 </project>

```



pom.xml文件的节点元素说明:	<project>	pom文件的顶级节点
<modelVersion>	object model版本, 对Maven2和Maven3来说, 只能是4.0.0	
<groupId>	项目创建组织的标识符, 一般是域名的倒写	
<artifactId>	定义了项目在所属组织的标识符下的唯一标识, 一个组织下可以有多个项目	
<version>	当前项目的版本, SNAPSHOT, 表示是快照版本, 在开发	
中 <packaging>	打包的方式, 有jar、war、ear等	
<name>	项目的名称	
<url>	项目的地址	
<properties>	属性配置, 比如: <project.build.sourceEncoding>UTF-	
8</project.build.sourceEncoding>		
<dependencies>	构建项目依赖的jar	

其中由groupId、artifactId和version唯一的确定了一个项目坐标

1.3、使用Maven编译-测试-打包-安装项目

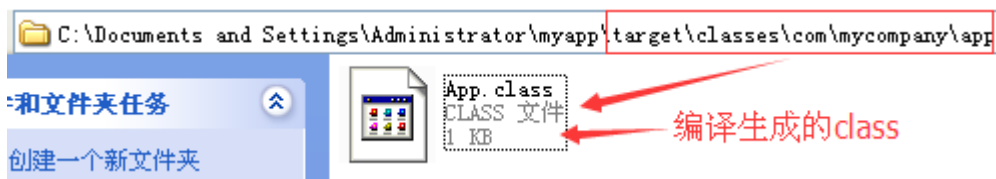
1.3.1、编译

编译源程序, 进入命令行, 切换到myapp目录, 执行命令: mvn clean compile, 如下图所示:

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\myapp>mvn clean compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building myapp 1.0-SNAPSHOT
[INFO] -----
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ myapp ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ myapp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Documents and Settings\Administrator\myapp\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ myapp ---
[INFO] Compiling 1 source file to C:\Documents and Settings\Administrator\myapp\target\classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.313 s
[INFO] Finished at: 2015-01-22T14:31:06+08:00
[INFO] Final Memory: 8M/23M
[INFO] -----
C:\Documents and Settings\Administrator\myapp>
```

编译好的class放在classes目录下

编译成功，在myapp目录下多出一个target目录，target\classes里面存放的就是编译后的class文件，如下图所示：



编译生成的class

1.3.2、测试

进入命令行，切换到myapp目录，执行命令：mvn clean test，如下图所示：

```

C:\Documents and Settings\Administrator\myapp>mvn clean test
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building myapp 1.0-SNAPSHOT
[INFO] -----
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ myapp ---
[INFO] Deleting C:\Documents and Settings\Administrator\myapp\target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ myapp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Documents and Settings\Administrat
or\myapp\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ myapp ---
[INFO] Compiling 1 source file to C:\Documents and Settings\Administrator\myapp\t
arget\classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ my
app ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Documents and Settings\Administrat
or\myapp\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:testCompile (default-testCompile) @ myapp
---
[INFO] Compiling 1 source file to C:\Documents and Settings\Administrator\myapp\t
arget\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ myapp ---
[INFO] Surefire report directory: C:\Documents and Settings\Administrator\myapp\t
arget\surefire-reports

-----
T E S T S
-----
Running com.mycompany.app.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec

Results :

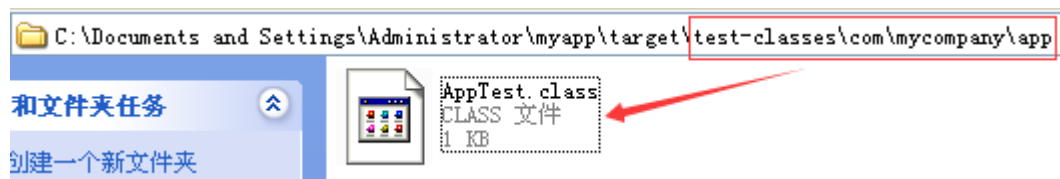
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.516 s
[INFO] Finished at: 2015-01-22T14:37:15+08:00
[INFO] Final Memory: 10M/25M
[INFO] -----

```

← 测试成功

测试成功，在myapp\target目录下会有一个test-classes目录，存放的就是测试代码的class文件，如下图所示：



1.3.3、打包

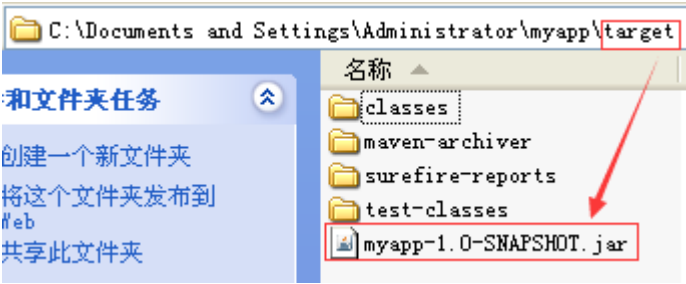
进入命令行，切换到myapp目录，执行命令：mvn clean package，执行打包命令前，会先执行编译和测试命令，如下图所示：

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator\myapp>mvn clean package

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myapp ---
[INFO] Building jar: C:\Documents and Settings\Administrator\myapp\target\myapp-1.0-SNAPSHOT.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 1.719 s
[INFO] Finished at: 2015-01-22T14:45:26+08:00
[INFO] Final Memory: 11M/27M
[INFO]
```

构建成功后，会在target目录下生成myapp-1.0-SNAPSHOT.jar包，如下图所示：



1.3.4、安装

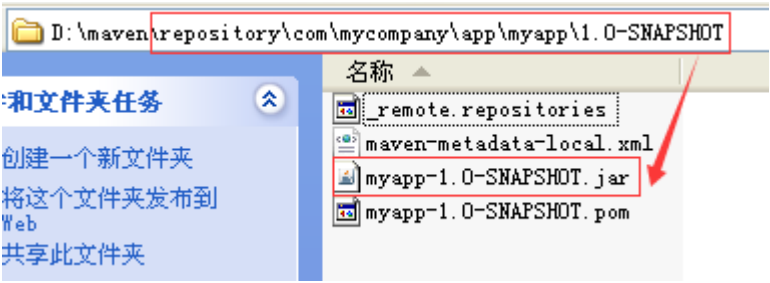
进入命令行，切换到my-app目录，执行命令：mvn clean install，执行安装命令前，会先执行编译、测试、打包命令，如下图所示：

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator\myapp>mvn clean install

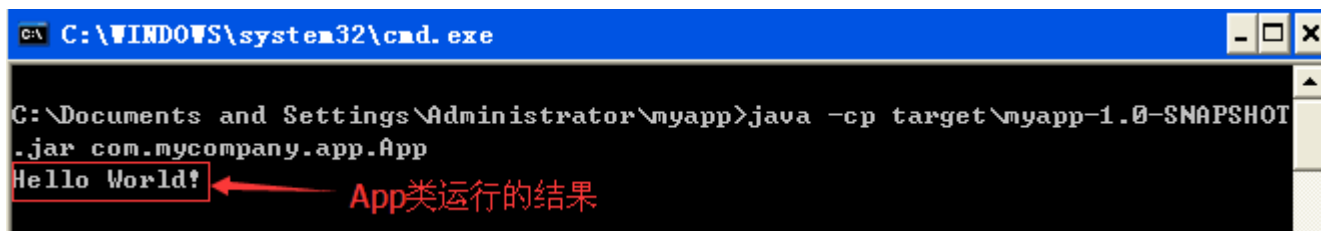
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myapp ---
[INFO] Building jar: C:\Documents and Settings\Administrator\myapp\target\myapp-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ myapp ---
[INFO] Installing C:\Documents and Settings\Administrator\myapp\target\myapp-1.0-SNAPSHOT.jar to D:\maven\repository\com\mycompany\app\myapp\1.0-SNAPSHOT\myapp-1.0-SNAPSHOT.jar
[INFO] Installing C:\Documents and Settings\Administrator\myapp\pom.xml to D:\maven\repository\com\mycompany\app\myapp\1.0-SNAPSHOT\myapp-1.0-SNAPSHOT.pom
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 1.953 s
[INFO] Finished at: 2015-01-22T14:45:41+08:00
[INFO] Final Memory: 10M/24M
[INFO]
C:\Documents and Settings\Administrator\myapp>
```

构建成功，就会将项目的jar包安装到本地仓库，如下图所示：



1.3.5、运行jar包

进入命令行，切换到myapp目录，执行命令：`java -cp target\myapp-1.0-SNAPSHOT.jar com.mycompany.app.App`，如下图所示：



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\myapp>java -cp target\myapp-1.0-SNAPSHOT.jar com.mycompany.app.App
Hello World!
```

App类运行的结果

二、构建JavaWeb项目

2.1、创建JavaWeb项目

1、使用`mvn archetype:generate`命令，如下所示：

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-WebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

使用“`mvn archetype:generate`”命令创建一个javaWeb项目的过程如下图所示：


```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator>mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-WebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
[INFO]
[INFO] >>> maven-archetype-plugin:2.2:generate (default-cli) > generate-sources
@ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.2:generate (default-cli) < generate-sources
@ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom ---
[INFO]
[INFO] Generating project in Batch mode
[INFO] -----
[INFO]
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-webapp:1.0
[INFO] -----
[INFO]
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: packageName, Value: com.mycompany.app
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: my-WebApp
[INFO] Parameter: basedir, Value: C:\Documents and Settings\Administrator
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Documents and Settings\Administrator\my-WebApp
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 48.719 s
[INFO] Finished at: 2015-01-22T10:57:13+08:00
[INFO] Final Memory: 10M/24M
[INFO] -----
C:\Documents and Settings\Administrator>
```

使用“mvn archetype:generate”命令创建一个javaWeb项目的时间非常长，要了40多秒，有时甚至会更久，不知道为啥。

2、使用mvn archetype:create命令，如下所示：

```
mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=myWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

使用“mvn archetype:create”命令创建一个javaWeb项目的过程如下图所示：

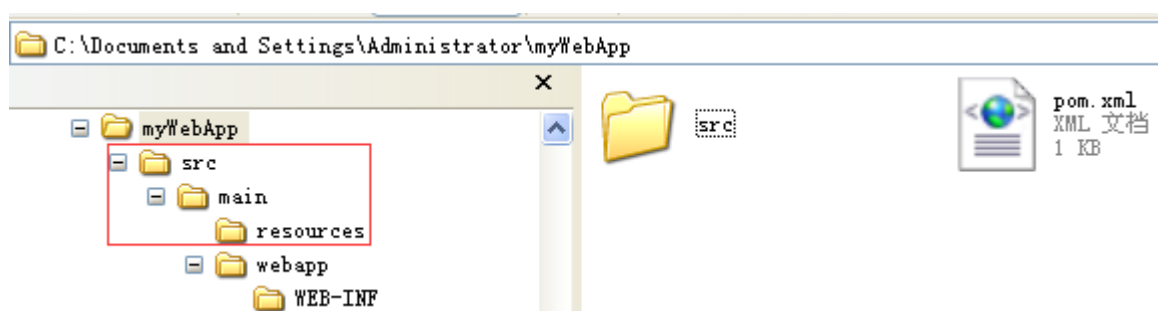
```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrator>mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=myWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
[INFO]
[INFO] --- maven-archetype-plugin:2.2:create (default-cli) @ standalone-pom ---
[WARNING] This goal is deprecated. Please use mvn archetype:generate instead
[INFO] Defaulting package to group ID: com.mycompany.app
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/maven-metadata.xml
Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/maven-metadata.xml (498 B at 0.6 KB/sec)
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/1.0/maven-archetype-webapp-1.0.jar
Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/1.0/maven-archetype-webapp-1.0.jar (4 KB at 8.5 KB/sec)
Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/1.0/maven-archetype-webapp-1.0.pom
Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-webapp/1.0/maven-archetype-webapp-1.0.pom (533 B at 1.0 KB/sec)
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-webapp:RELEASE
[INFO] -----
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: packageName, Value: com.mycompany.app
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: myWebApp
[INFO] Parameter: basedir, Value: C:\Documents and Settings\Administrator
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Documents and Settings\Administrator\myWebApp
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.891 s
[INFO] Finished at: 2015-01-22T10:03:35+08:00
[INFO] Final Memory: 11M/26M
[INFO] -----
C:\Documents and Settings\Administrator>
```

生成的web项目存放位置，存放在当前登录到操作系统的用户的用户目录中

使用“mvn archetype:create”命令创建一个javaWeb项目的时间非常快，几秒钟就可以了。

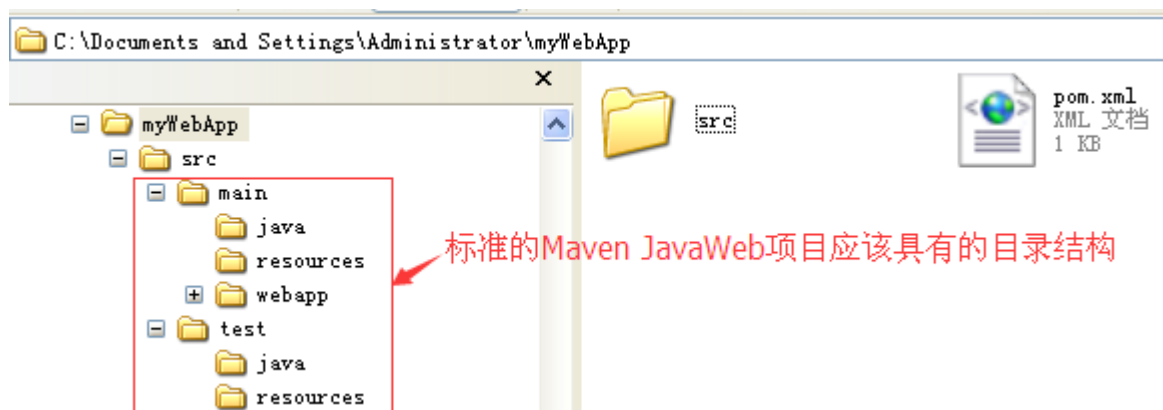
创建好的JavaWeb项目的目录结构如下：



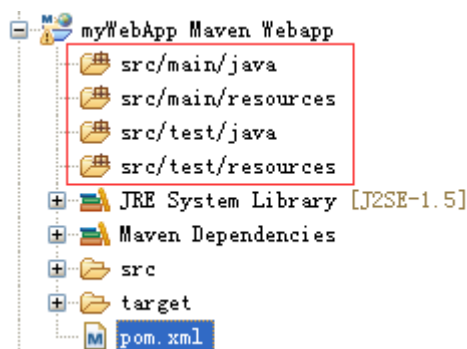
创建好的JavaWeb项目中目前只有src/main/resources目录，因此还需要手动添

加src/main/java、src/test/java、src/test/resources

如下图所示：

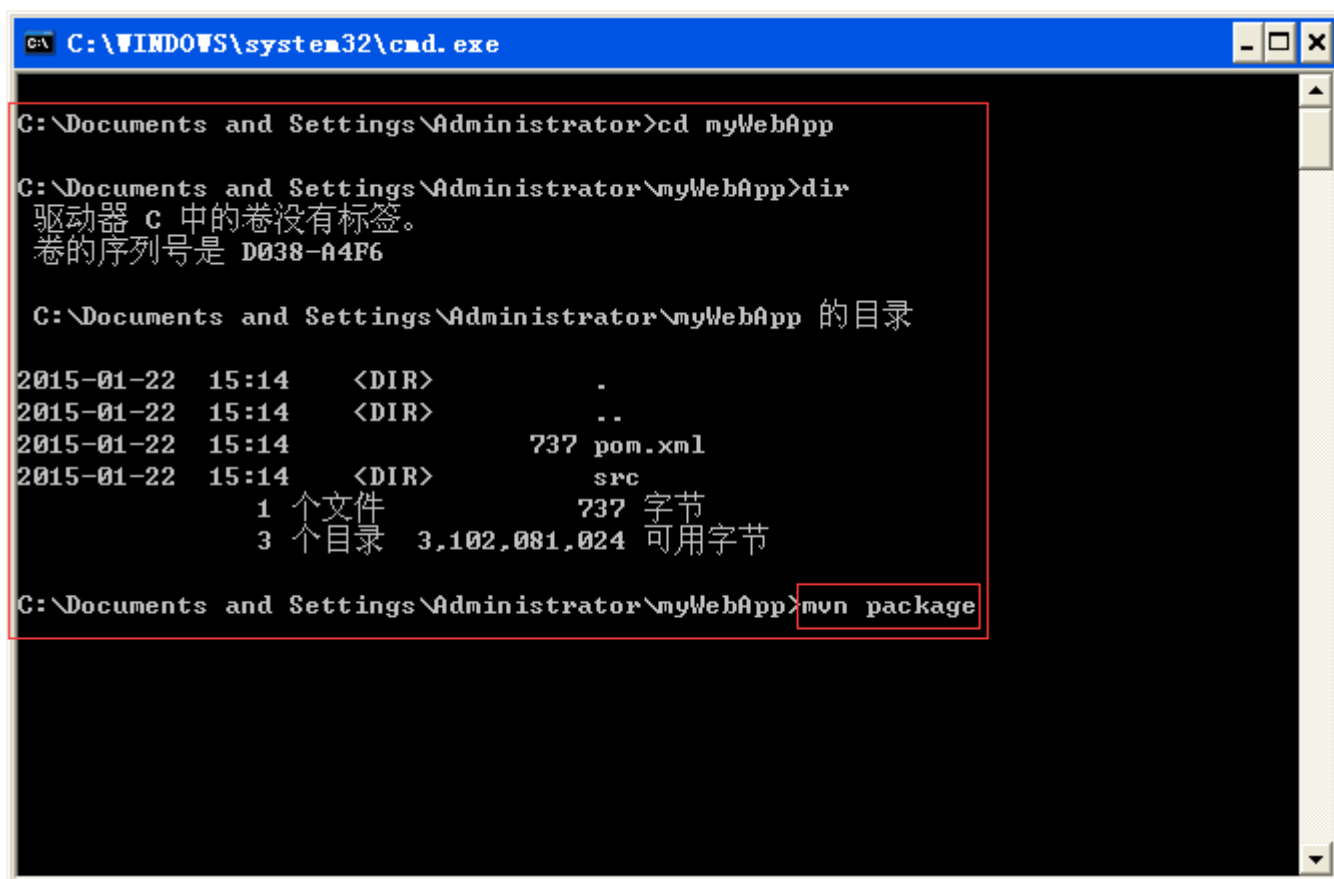


接着我们就可以将创建好的JavaWeb导入Eclipse中进行开发了，如下图所示：



2.2、使用Maven打包发布Web项目

Maven帮我们创建的JavaWeb项目是一个空的项目，只有一个index.jsp页面，我们使用Maven将Web项目打包发布运行。在命令行切换到myWebApp目录，执行：`mvn package`，构建成功后，myWebApp目录目录下多了一个target目录，在这个目录下会打包成myWebApp目录.war，把这个war包拷贝到Tomcat的发布目录下就可以运行了。如下图所示：

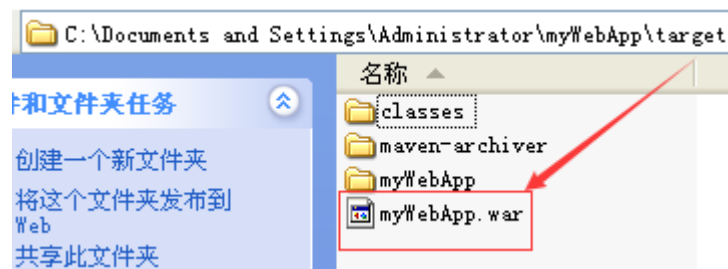


```
C:\WINDOWS\system32\cmd.exe

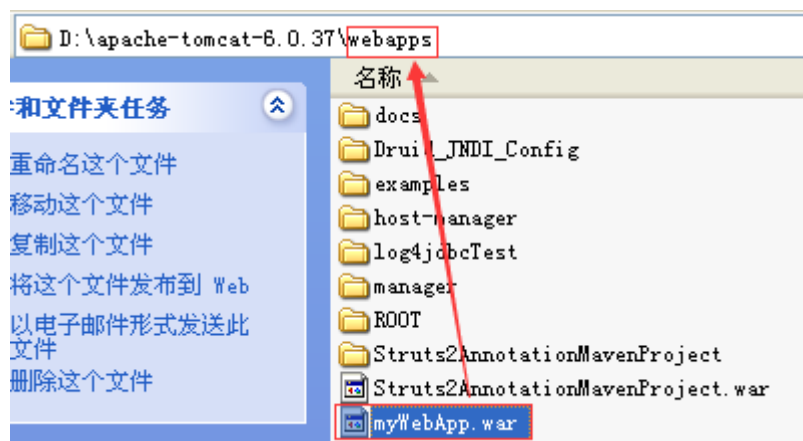
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ myWebApp ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ myWebApp ---
[INFO] Packaging webapp
[INFO] Assembling webapp [myWebApp] in [C:\Documents and Settings\Administrator\myWebApp\target\myWebApp]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\Documents and Settings\Administrator\myWebApp\src\main\webapp]
[INFO] Webapp assembled in [31 msecs]
[INFO] Building war: C:\Documents and Settings\Administrator\myWebApp\target\myWebApp.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.000 s
[INFO] Finished at: 2015-01-22T15:16:08+08:00
[INFO] Final Memory: 7M/19M
[INFO] -----
C:\Documents and Settings\Administrator\myWebApp>
```

在target目录下生成
myWebApp.war

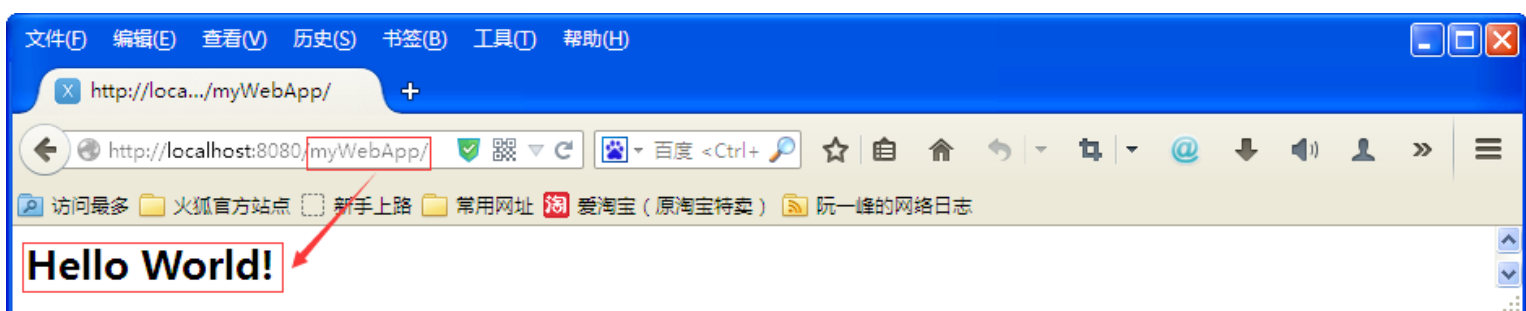
打包成功，在myWebApp\target目录下生成了一个myWebApp.war文件，如下图所示：



将myWebApp.war放到tomcat服务器中运行，如下图所示：



运行效果如下：



除了使用Tomcat服务器运行Web项目之外，我们还可以在Web项目中集成Jetty发布运行，首先在pom.xml文件中配置Jetty插件，如下：



```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>com.mycompany.app</groupId>
5   <artifactId>myWebApp</artifactId>
6   <packaging>war</packaging>
7   <version>1.0-SNAPSHOT</version>
8   <name>myWebApp Maven Webapp</name>
9   <url>http://maven.apache.org</url>
10  <dependencies>
11    <dependency>
12      <groupId>junit</groupId>
13      <artifactId>junit</artifactId>
14      <version>3.8.1</version>
15      <scope>test</scope>
16    </dependency>
17  </dependencies>
18  <build>
19    <finalName>myWebApp</finalName>
20    <pluginManagement>
21      <!--配置Jetty-->
22      <plugins>
23        <plugin>
24          <groupId>org.mortbay.jetty</groupId>
25          <artifactId>maven-jetty-plugin</artifactId>
26        </plugin>
27      </plugins>
28    </pluginManagement>
29  </build>
30 </project>
```



打开命令行窗口，切换到myWebApp目录，然后执行：mvn jetty:run启动Jetty服务器，如下图所示：

```
C:\Documents and Settings\Administrator\myWebApp>mvn jetty:run
```

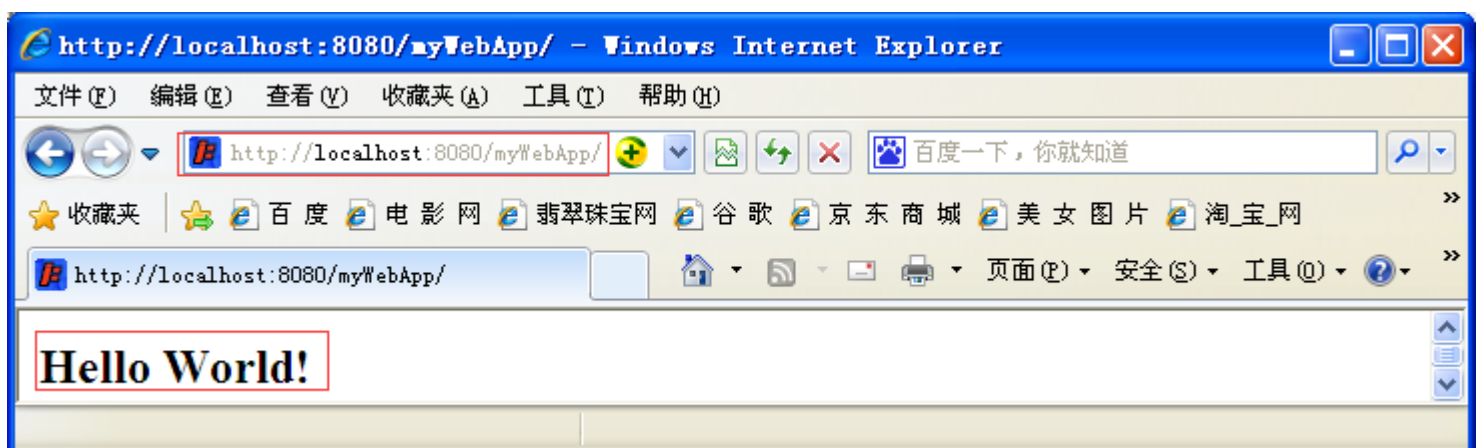
```
C:\WINDOWS\system32\cmd.exe - mvn jetty:run

[INFO]
[INFO] --- maven-jetty-plugin:6.1.26:run (default-cli) @ myWebApp ---
[INFO] Configuring Jetty for project: myWebApp Maven Webapp
[INFO] Webapp source directory = C:\Documents and Settings\Administrator\myWebApp\src\main\webapp
[INFO] Reload Mechanic: automatic
[INFO] Classes = C:\Documents and Settings\Administrator\myWebApp\target\classes

[INFO] Logging to org.slf4j.impl.SimpleLogger(org.morthbay.log) via org.morthbay.log.Slf4jLog
[INFO] Context path = /myWebApp
[INFO] Tmp directory = determined at runtime
[INFO] Web defaults = org/morthbay/jetty/webapp/webdefault.xml
[INFO] Web overrides = none
[INFO] web.xml file = C:\Documents and Settings\Administrator\myWebApp\src\main\webapp\WEB-INF\web.xml
[INFO] Webapp directory = C:\Documents and Settings\Administrator\myWebApp\src\main\webapp
[INFO] Starting jetty 6.1.26 ...
[INFO] jetty-6.1.26
[INFO] No Transaction manager found - if your webapp requires one, please configure one.
[INFO] Started SelectChannelConnector@0.0.0.0:8080
[INFO] Started Jetty Server
```

← Jetty服务器启动成功，使用的是8080端口

接着就可以在8080端口上访问应用了。如下图所示：



三、Maven创建项目的命令说明

mvn archetype:create	或者	mvn archetype:generate	固定写法	-
DgroupId			组织标识（包名）	-
DartifactId			项目名称	-DarchetypeArtifactId
		指定ArchetypeId, maven-archetype-quickstart,	创建一个Java Project;	maven-
archetype-webapp,	创建一个Web Project	-DinteractiveMode		是否使用交互模式

archetype是mvn内置的一个插件，create任务可以创建一个java项目骨架，DgroupId是软件包的名称，DartifactId是项目名，DarchetypeArtifactId是可用的mvn项目骨架，目前可以使用的骨架有：

- maven-archetype-archetype
- maven-archetype-j2ee-simple
- maven-archetype-mojo
- maven-archetype-portlet
- maven-archetype-profiles (currently under development)
- maven-archetype-quickstart

- maven-archetype-simple (currently under development)
- maven-archetype-site
- maven-archetype-site-simple
- maven-archetype-webapp

每一个骨架都会建相应的目录结构和一些通用文件，最常用的是maven-archetype-quickstart和maven-archetype-webapp骨架。maven-archetype-quickstart骨架是用来创建一个Java Project，而maven-archetype-webapp骨架则是用来创建一个JavaWeb Project。

不得不说，Maven的确是一个很好的项目构建工具。掌握好Maven对于项目开发是非常有帮助的。

一、Maven坐标

1.1、什么是坐标？

在平面几何中坐标（x, y）可以标识平面中唯一的一点。

1.2、Maven坐标主要组成

- groupId：组织标识（包名）
- artifactId：项目名称
- version：项目的当前版本
- packaging：项目的打包方式，最为常见的jar和war两种

样例：

```
<groupId>com.mycompany.app</groupId>
<artifactId>myapp</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>

<groupId>com.mycompany.app</groupId>
<artifactId>myWebApp</artifactId>
<packaging>war</packaging>
<version>1.0-SNAPSHOT</version>
```

1.3、Maven为什么使用坐标？

- Maven世界拥有大量构建，我们需要找一个用来唯一标识一个构建的统一规范。
- 拥有了统一规范，就可以把查找工作交给机器。

二、依赖管理

2.1、依赖配置

依赖配置主要包含如下元素：



```
1    <!--添加依赖配置-->
2    <dependencies>
3        <!--项目要使用到junit的jar包，所以在这里添加junit的jar包的依赖-->
```

```

4      <dependency>
5          <groupId>junit</groupId>
6          <artifactId>junit</artifactId>
7          <version>4.9</version>
8          <scope>test</scope>
9      </dependency>
10     <!--项目要使用到Hello的jar包，所以在这里添加Hello的jar包的依赖-->
11     <dependency>
12         <groupId>me.gacl.maven</groupId>
13         <artifactId>Hello</artifactId>
14         <version>0.0.1-SNAPSHOT</version>
15         <scope>compile</scope>
16     </dependency>
17 </dependencies>

```



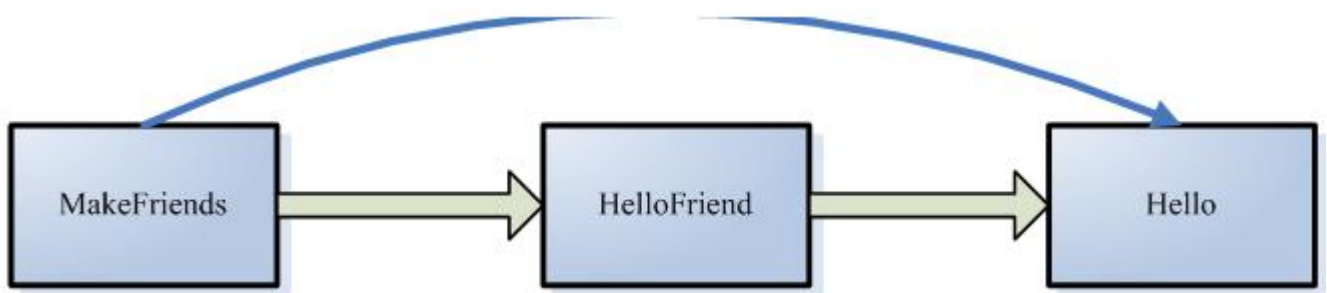
2.2、依赖范围

依赖范围scope用来控制依赖和编译，测试，运行的classpath的关系。主要的是三种依赖关系如下：

- 1.compile：默认编译依赖范围。对于编译，测试，运行三种classpath都有效
- 2.test：测试依赖范围。只对于测试classpath有效
- 3.provided：已提供依赖范围。对于编译，测试的classpath都有效，但对于运行无效。因为由容器已经提供，例如servlet-api
- 4.runtime：运行时提供。例如：jdbc驱动

2.3、传递性依赖

MakeFriends.jar直接依赖于HelloFriends.jar，而HelloFriends.jar又直接依赖于Hello.jar，那么MakeFriends.jar也依赖于Hello.jar，这就是传递性依赖，只不过这种依赖是间接依赖，如下图所示：



2.4、可选依赖

三、仓库管理

3.1、Maven仓库

用来统一存储所有Maven共享构建的位置就是仓库

3.2、Maven仓库布局

根据Maven坐标定义每个构建在仓库中唯一存储路径，大致为：groupId/artifactId/version/artifactId-version.packaging

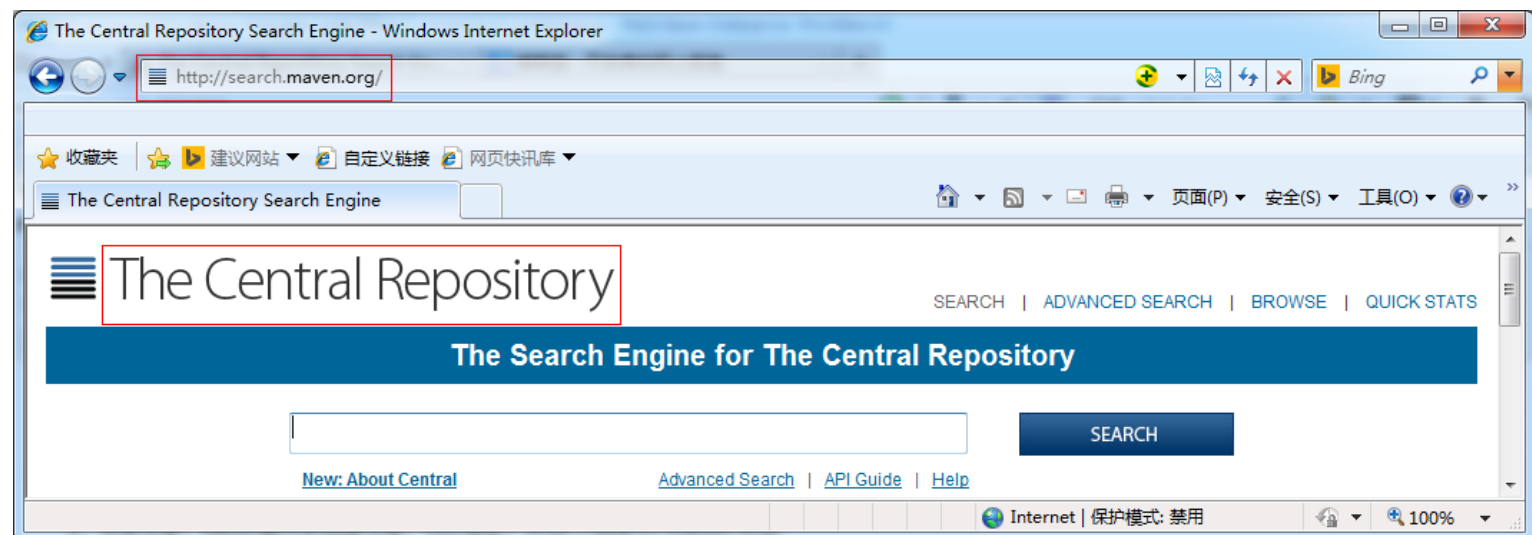
3.3、仓库的分类

3.3.1、本地仓库

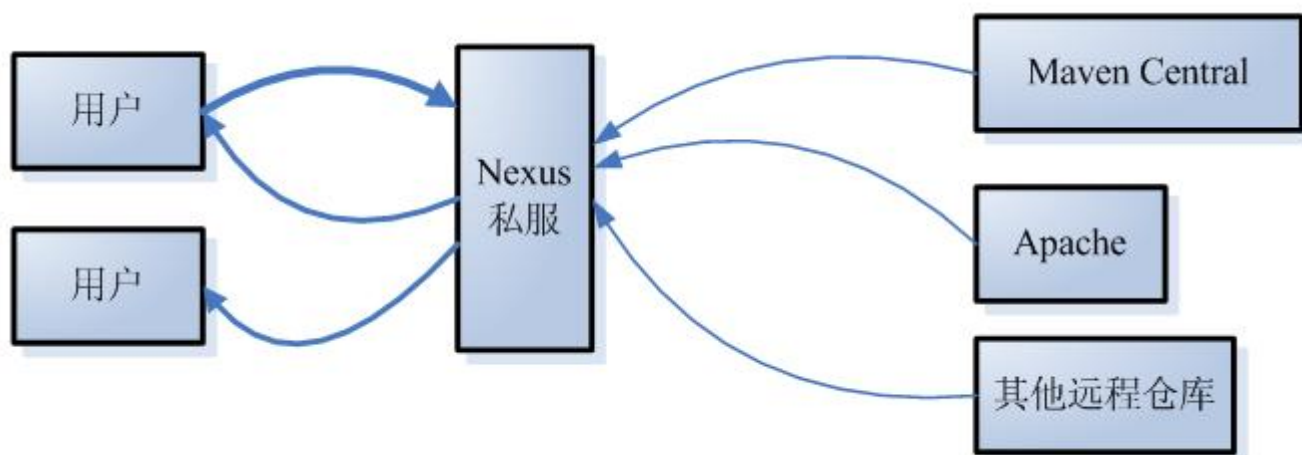
每个用户只有一个本地仓库，默认是在`~/.m2/repository/`，`~`代表的是用户目录

3.3.2、远程仓库

1、中央仓库：Maven默认的远程仓库，URL地址：`http://search.maven.org/`



2、私服：是一种特殊的远程仓库，它是架设在局域网内的仓库



四、生命周期

4.1、何为生命周期？

Maven生命周期就是为对所有的构建过程进行抽象和统一，包括项目清理，初始化，编译，打包，测试，部署等几乎所有构建步骤

4.2、Maven三大生命周期

Maven有三套相互独立的生命周期，请注意这里说的是“三套”，而且“相互独立”，这三套生命周期分别是：

1. Clean Lifecycle 在进行真正的构建之前进行一些清理工作。
2. Default Lifecycle 构建的核心部分，编译，测试，打包，部署等等。
3. Site Lifecycle 生成项目报告，站点，发布站点。

再次强调一下它们是相互独立的，你可以仅仅调用clean来清理工作目录，仅仅调用site来生成站点。当然你也可以直接运行 `mvn clean install site` 运行所有这三套生命周期。

clean生命周期每套生命周期都由一组阶段(Phase)组成，我们平时在命令行输入的命令总会对应于一个特定的阶段。比如，运行`mvn clean`，这个的clean是Clean生命周期的一个阶段。有Clean生命周期，也有clean阶段。Clean生命周期一共包含了三个阶段：

1. `pre-clean` 执行一些需要在clean之前完成的工作
2. `clean` 移除所有上一次构建生成的文件
3. `post-clean` 执行一些需要在clean之后立刻完成的工作

“`mvn clean`”中的clean就是上面的clean，在一个生命周期中，运行某个阶段的时候，它之前的所有阶段都会被运行，也就是说，“`mvn clean`”等同于 `mvn pre-clean clean`，如果我们运行 `mvn post-clean`，那么 `pre-clean`，`clean` 都会被运行。这是Maven很重要的一个规则，可以大大简化命令行的输入。

Site生命周期`pre-site` 执行一些需要在生成站点文档之前完成的工作

1. `site` 生成项目的站点文档
2. `post-site` 执行一些需要在生成站点文档之后完成的工作，并且为部署做准备
3. `site-deploy` 将生成的站点文档部署到特定的服务器上

这里经常用到的是site阶段和site-deploy阶段，用以生成和发布Maven站点，这可是Maven相当强大的功能，Manager比较喜欢，文档及统计数据自动生成，很好看。

Default生命周期Default生命周期是Maven生命周期中最重要的一个，绝大部分工作都发生在这个生命周期中。这里，只解释一些比较重要和常用的阶段：

- `validate`
- `generate-sources`
- `process-sources`
- `generate-resources`
- `process-resources` 复制并处理资源文件，至目标目录，准备打包。
- `compile` 编译项目的源代码。
- `process-classes`
- `generate-test-sources`
- `process-test-sources`
- `generate-test-resources`
- `process-test-resources` 复制并处理资源文件，至目标测试目录。
- `test-compile` 编译测试源代码。
- `process-test-classes`
- `test` 使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。
- `prepare-package`
- `package` 接受编译好的代码，打包成可发布的格式，如 JAR。
- `pre-integration-test`
- `integration-test`
- `post-integration-test`
- `verify`
- `install` 将包安装至本地仓库，以让其它项目依赖。
- `deploy` 将最终的包复制到远程的仓库，以让其它开发人员与项目共享。

运行任何一个阶段的时候，它前面的所有阶段都会被运行，这也就是为什么我们运行`mvn install`的时候，代码会被编译，测试，打包。此外，Maven的插件机制是完全依赖Maven的生命周期的，因此理解生命周期至关重要。

五、Maven插件

1. Maven的核心仅仅定义了抽象的生命周期，具体的任务都是交由插件完成的。
2. 每个插件都能实现多个功能，每个功能就是一个插件目标。
3. Maven的生命周期与插件目标相互绑定，以完成某个具体的构建任务，例如compile就是插件maven-compiler-plugin的一个插件目标。