

# Maven学习总结(二)——Maven项目构建过程练习 - 孤傲苍狼

上一篇只是简单介绍了一下maven入门的一些相关知识，这一篇主要是体验一下Maven高度自动化构建项目的过程

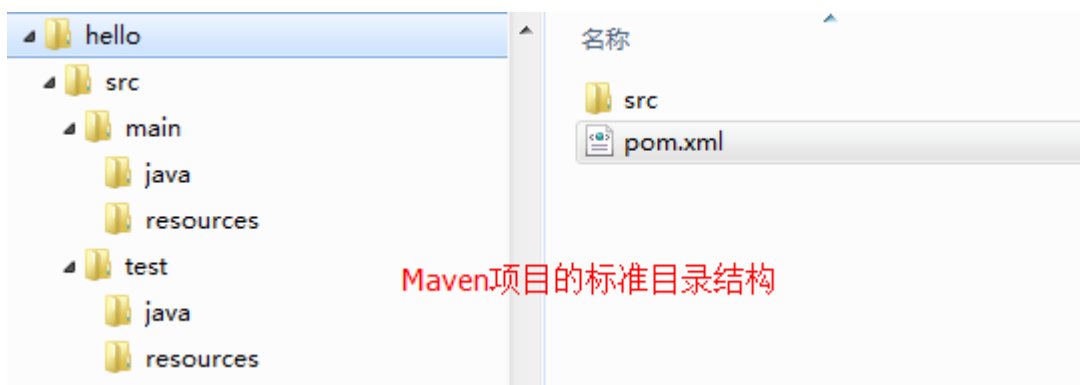
## 一、创建Maven项目

### 1.1、建立Hello项目

1、首先建立Hello项目，同时建立Maven约定的目录结构和pom.xml文件

Hello

```
| --src
|   |--main
|   |   |--java
|   |   |--resources
|   |--test
|   |   |--java
|   |   |--resources
|--pom.xml
```



2、编辑项目Hello根目录下的pom.xml，添加如下的代码：

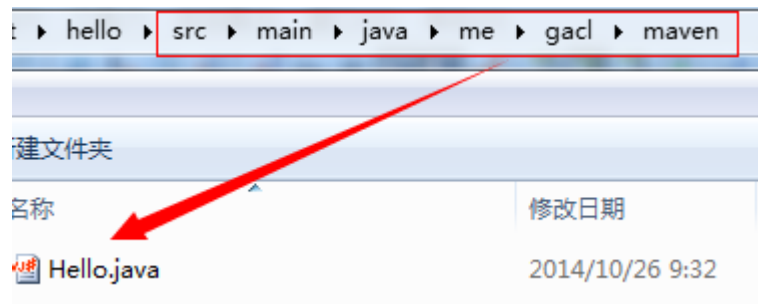


```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
2 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>me.gacl.maven</groupId>
5   <artifactId>Hello</artifactId>
6   <version>0.0.1-SNAPSHOT</version>
7   <name>Hello</name>
8
9   <!--添加依赖的jar包-->
10  <dependencies>
11    <!--项目要使用到junit的jar包，所以在这里添加junit的jar包的依赖-->
12    <dependency>
13      <groupId>junit</groupId>
14      <artifactId>junit</artifactId>
15      <version>4.9</version>
16      <scope>test</scope>
```

```
17     </dependency>
18
19 </dependencies>
20 </project>
```



3、在src/main/java/me/gacl/maven目录下新建文件Hello.java



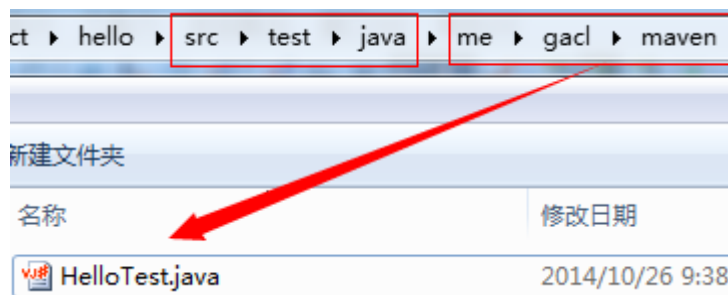
Hello.java的代码如下：



```
1 package me.gacl.maven;
2
3 public class Hello {
4
5     public String sayHello(String name){
6         return "Hello "+name+"!";
7     }
8 }
```



4、在/src/test/java/me/gacl/maven目录下新建测试文件HelloTest.java



HelloTest.java的代码如下：



```
1 package me.gacl.maven;
2 //导入junit的包
3 import org.junit.Test;
4 import static junit.framework.Assert.*;
5
6 public class HelloTest {
7
8     @Test
9     public void testHello(){
10         Hello hello = new Hello();
```

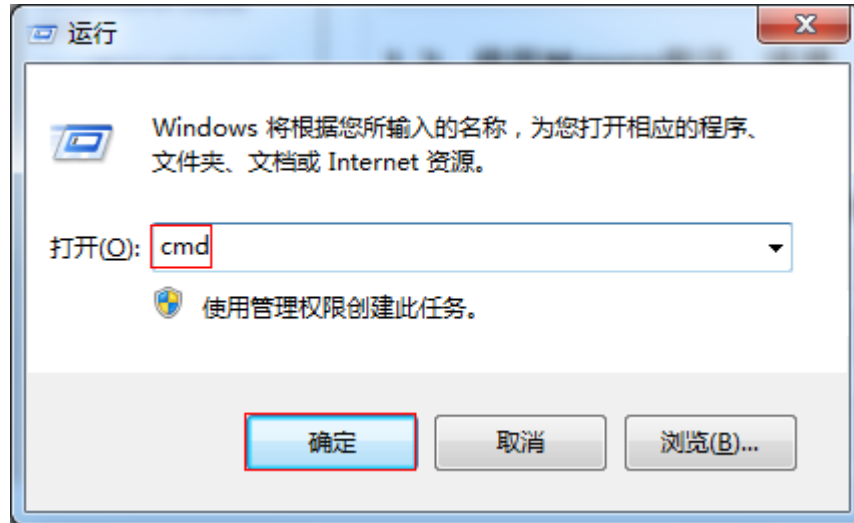
```
11     String results= hello.sayHello("gacl");
12     assertEquals("Hello gacl!",results);
13 }
14 }
```



## 1.2、使用Maven编译、清理、测试、打包项目

1、使用Maven编译项目，编译项目的命令是：“mvn compile”

打开cmd命令行，



进入Hello项目根目录执行“mvn compile”命令编译项目的java类

```
C:\Windows\system32\cmd.exe
C:\Users\gacl>E:
E:\>cd MavenProject\hello
E:\MavenProject\hello>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Hello 0.0.1-SNAPSHOT
[INFO] -----
Downloading: https://repo.maven.apache.org/maven2/junit/junit/4.9/junit-4.9.pom
Downloaded: https://repo.maven.apache.org/maven2/junit/junit/4.9/junit-4.9.pom <3 KB at 0.8 KB/sec>
Downloading: https://repo.maven.apache.org/maven2/org/hamcrest/hamcrest-core/1.1/hamcrest-core-1.1.pom
Downloaded: https://repo.maven.apache.org/maven2/org/hamcrest/hamcrest-core/1.1/hamcrest-core-1.1.pom <481 B at 0.7 KB/sec>
Downloading: https://repo.maven.apache.org/maven2/org/hamcrest/hamcrest-parent/1.1/hamcrest-parent-1.1.pom
Downloaded: https://repo.maven.apache.org/maven2/org/hamcrest/hamcrest-parent/1.1/hamcrest-parent-1.1.pom <6 KB at 9.0 KB/sec>
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Hello ---
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Hello ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding GBK, i.e. build is platform dependent!
[INFO] Compiling 1 source file to E:\MavenProject\hello\target\classes
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 10.176 s
[INFO] Finished at: 2014-10-26T09:46:02+08:00
[INFO] Final Memory: 10M/24M
[INFO]
E:\MavenProject\hello>
```

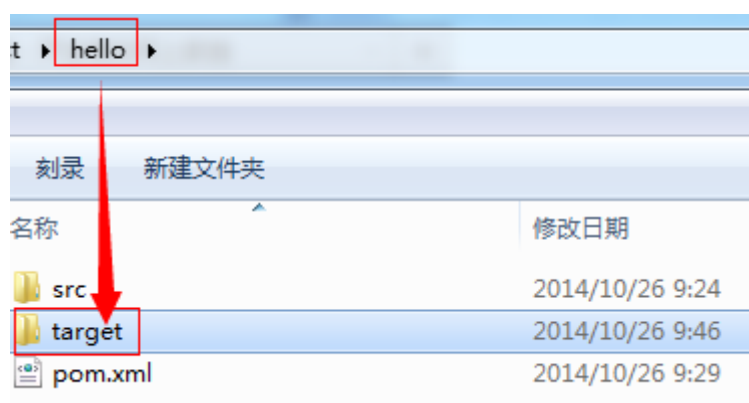
进入hello项目根目录

执行maven的编译命令

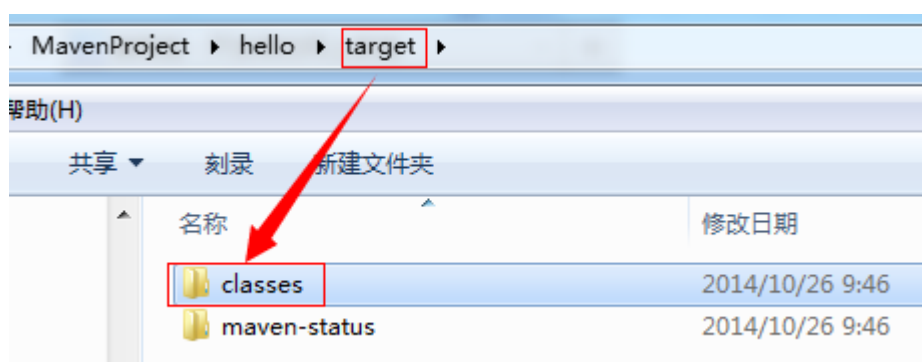
在项目的pom.xml文件中添加了对junit这个jar包的依赖，所以Maven会自动去下载junit这个jar包以及junit这个jar包的相关依赖jar

项目编译成功

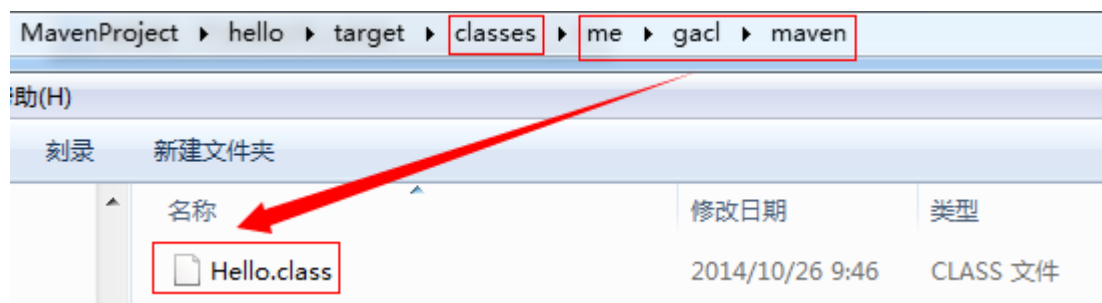
编译成功之后，可以看到hello项目的根目录下多了一个【target】文件夹，这个文件夹就是编译成功之后Maven帮我们生成的文件夹，如下图所示：



打开【target】文件夹，可以看到里面有一个【classes】文件夹，如下图所示：



【classes】文件夹中存放的就是Maven我们编译好的java类，如下图所示：



这就是使用Maven自动编译项目的过程。

## 2、使用Maven清理项目，清理项目的命令是：“mvn clean”

进入Hello项目根目录执行“mvn clean”命令清理项目，清理项目的过程就是把执行“mvn compile”命令编译项目时生成的target文件夹删掉，如下图所示：

```
E:\MavenProject\hello>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Hello 0.0.1-SNAPSHOT
[INFO] -----
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ Hello ---
[INFO] Deleting E:\MavenProject\hello\target
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.436 s
[INFO] Finished at: 2014-10-26T10:05:02+08:00
[INFO] Final Memory: 4M/15M
[INFO] -----
```

## 3、使用Maven测试项目，测试项目的命令是：“mvn test” 进入Hello项目根目录执行“mvn test”命令测试项目，如下图所示：

```
管理员: C:\Windows\system32\cmd.exe
E:\MavenProject\hello>mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Hello 0.0.1-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Hello ---
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform
dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Hello ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding GBK, i.e. build is platform depend
ent!
[INFO] Compiling 1 source file to E:\MavenProject\hello\target\classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ Hello ---
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform
dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ Hello ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding GBK, i.e. build is platform depend
ent!
[INFO] Compiling 1 source file to E:\MavenProject\hello\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ Hello ---
[INFO] Surefire report directory: E:\MavenProject\hello\target\surefire-reports

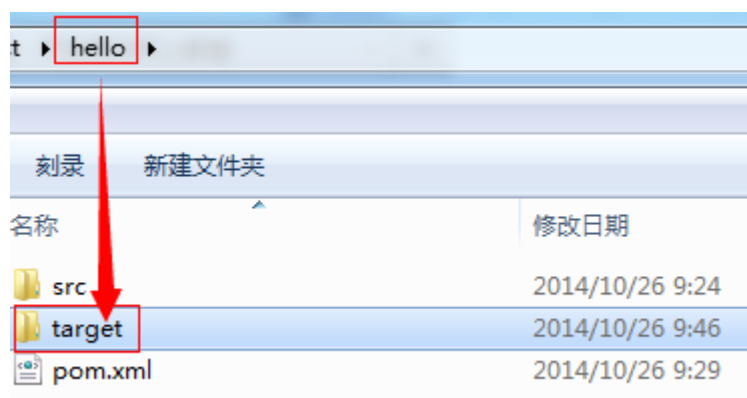
-----
T E S T S
-----

Running me.gacl.maven.HelloTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.039 sec

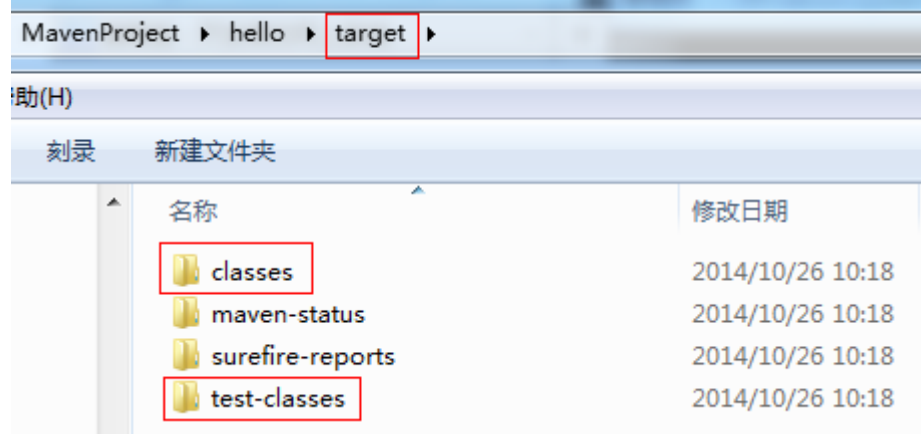
Results :

半:
```

测试成功之后，可以看到hello项目的根目录下多了一个【target】文件夹，这个文件夹就是测试成功之后Maven帮我们生成的文件夹，如下图所示：



打开【target】文件夹，可以看到里面有一个【classes】和【test-classes】文件夹，如下图所示：



也就是说，我们执行执行“mvn test”命令测试项目时，Maven先帮我们编译项目，然后再执行测试代码。

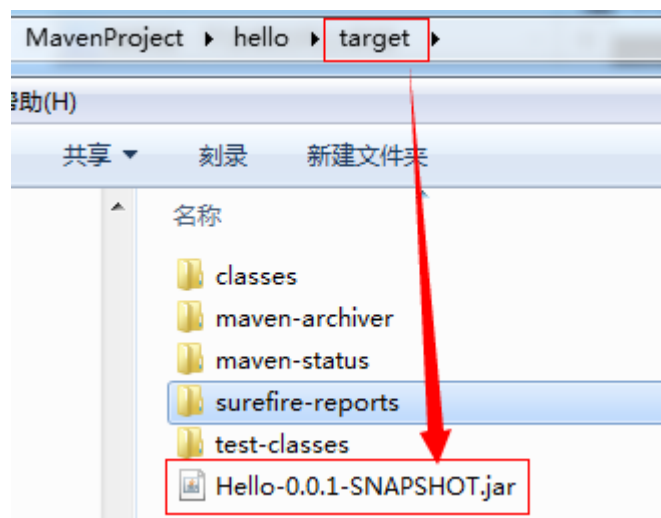
4、使用Maven打包项目，打包项目的命令是：“mvn package” 进入Hello项目根目录执行“mvn package”命令测试项目，如下图所示：

```
E:\MavenProject\hello>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Hello 0.0.1-SNAPSHOT
[INFO] -----
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-jar-plugin/2.4/maven-jar-plugin-2.4.pom
半：
```

执行Maven的打包项目命令

```
C:\Windows\system32\cmd.exe
-components-1.1.19.pom (3 KB at 4.6 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/3.0.1/plexus-3.0.1.pom
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/3.0.1/plexus-3.0.1.pom (19 KB at 22.0 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interpolation/1.15/plexus-interpolation-1.15.pom
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interpolation/1.15/plexus-interpolation-1.15.pom (1018 B at 2.1 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/commons-lang/commons-lang/2.1/commons-lang-2.1.pom
Downloaded: https://repo.maven.apache.org/maven2/commons-lang/commons-lang/2.1/commons-lang-2.1.pom (10 KB at 8.8 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.jar
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.jar (22 KB at 18.5 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.0.2/plexus-io-2.0.2.jar
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.0.2/plexus-io-2.0.2.jar (57 KB at 28.9 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.jar
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.jar (181 KB at 26.9 KB/sec)
[INFO] Building jar: E:\MavenProject\hello\target\Hello-0.0.1-SNAPSHOT.jar
[INFO] BUILD SUCCESS
[INFO] Total time: 01:34 min
[INFO] Finished at: 2014-10-26T10:30:05+08:00
[INFO] Final Memory: 8M/19M
[INFO]
E:\MavenProject\hello>
```

打包成功之后，可以看到hello项目的根目录下的【target】文件夹中多了一个Hello-0.0.1-SNAPSHOT.jar，这个Hello-0.0.1-SNAPSHOT.jar就是打包成功之后Maven帮我们生成的jar文件，如下图所示：



5、使用Maven部署项目，部署项目的命令是：“mvn install”

进入Hello项目根目录执行“mvn install”命令测试项目，如下图所示：



```
E:\MavenProject\hello>mvn clean
```

```
[INFO] Scanning for projects...
```

```
[INFO]
```

```
[INFO]
```

```
[INFO] Building Hello 0.0.1-SNAPSHOT
```

```
[INFO]
```

```
[INFO]
```

```
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ Hello ---
```

```
[INFO] Deleting E:\MavenProject\hello\target
```

```
[INFO]
```

```
[INFO] BUILD SUCCESS
```

```
[INFO]
```

```
[INFO] Total time: 0.241 s
```

```
[INFO] Finished at: 2014-10-26T10:40:11+08:00
```

```
[INFO] Final Memory: 4M/15M
```

```
[INFO]
```

```
E:\MavenProject\hello>mvn install
```

← 执行Maven的部署项目命令

```
[INFO] Scanning for projects...
```

```
[INFO]
```

```
[INFO]
```

```
[INFO] Building Hello 0.0.1-SNAPSHOT
```

```
[INFO]
```

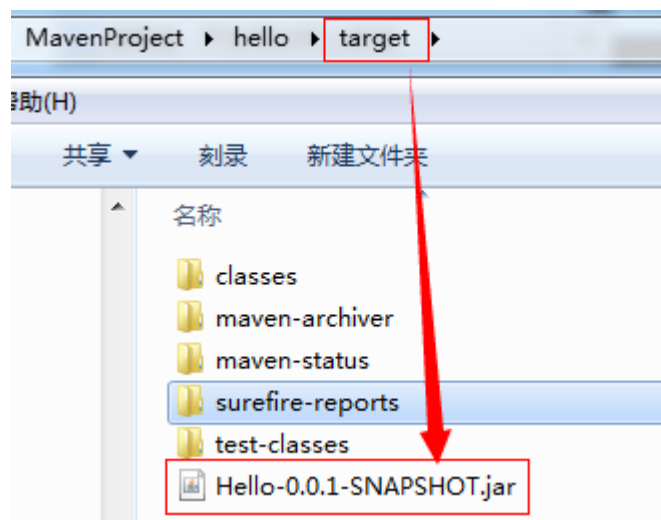
```
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-install-plugin/2.4/maven-install-plugin-2.4.pom
```

```
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-install-plugin/2.4/maven-install-plugin-2.4.pom (7 KB at 2.8 KB/sec)
```

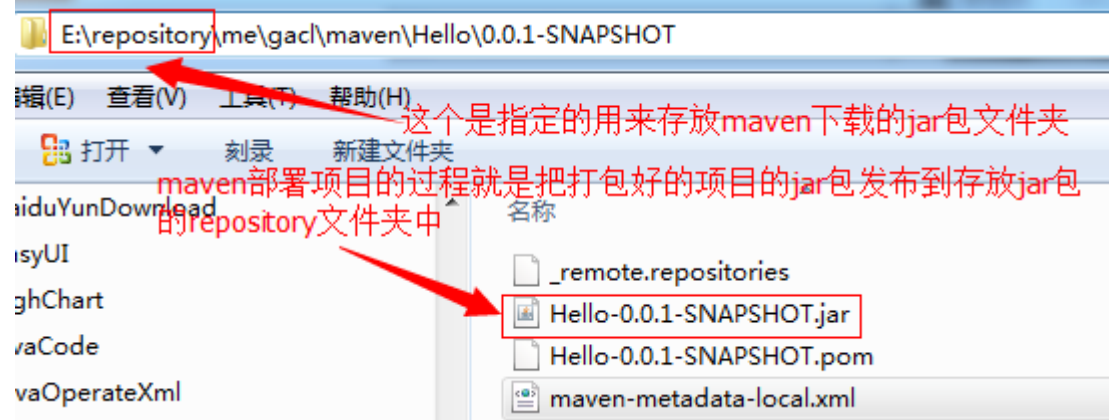
```
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-install-plugin/2.4/maven-install-plugin-2.4.jar
```

```
C:\Windows\system32\cmd.exe
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/1.0.8/plexus-1.0.8.pom
十月 26, 2014 10:41:55 上午 org.apache.maven.wagon.providers.http.httpclient.impl.execchain.RetryExec
c execute
信息: I/O exception (org.apache.maven.wagon.providers.http.httpclient.NoHttpResponseException) caught
when processing request: The target server failed to respond
十月 26, 2014 10:41:55 上午 org.apache.maven.wagon.providers.http.httpclient.impl.execchain.RetryExec
c execute
信息: Retrying request
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/1.0.8/plexus-1.0.8.pom (
8 KB at 0.2 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-8/plexus-container-default-1.0-alpha-8.pom
十月 26, 2014 10:42:39 上午 org.apache.maven.wagon.providers.http.httpclient.impl.execchain.RetryExec
c execute
信息: I/O exception (org.apache.maven.wagon.providers.http.httpclient.NoHttpResponseException) caught
when processing request: The target server failed to respond
十月 26, 2014 10:42:39 上午 org.apache.maven.wagon.providers.http.httpclient.impl.execchain.RetryExec
c execute
信息: Retrying request
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-8/plexus-container-default-1.0-alpha-8.pom (8 KB at 0.4 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.jar
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.jar
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.jar (226 KB at 27.5 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.jar (12 KB at 1.0 KB/sec)
[INFO] Installing E:\MavenProject\hello\target\Hello-0.0.1-SNAPSHOT.jar to E:\repository\me\gac\maven\Hello\0.0.1-SNAPSHOT\Hello-0.0.1-SNAPSHOT.jar
[INFO] Installing E:\MavenProject\hello\pom.xml to E:\repository\me\gac\maven\Hello\0.0.1-SNAPSHOT\Hello-0.0.1-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:30 min
[INFO] Finished at: 2014-10-26T10:43:00+08:00
[INFO] Final Memory: 12M/28M
[INFO] -----
E:\MavenProject\hello>
半:
```

安装成功之后，首先会在hello项目的根目录下生成【target】文件夹，打开【target】文件夹，可以看到里面会有Hello-0.0.1-SNAPSHOT.jar，这个Hello-0.0.1-SNAPSHOT.jar就是安装成功之后Maven帮我们生成的jar文件，如下图所示：



除此之外，在我们存放Maven下载下来的jar包的仓库也会有一个Hello-0.0.1-SNAPSHOT.jar，所以Maven安装项目的过程，实际上就是把项目进行【清理】→【编译】→【测试】→【打包】，再把打包好的jar放到我们指定的存放jar包的Maven仓库中，如下图所示：



所以使用“mvn install”命令，就把maven构建项目的【清理】→【编译】→【测试】→【打包】的这几个过程都做了，同时将打包好的jar包发布到本地的Maven仓库中，所以maven最常用的命令还是“mvn install”，这个命令能够做的事情最多。

## 1.2、组合使用Maven的命令

maven的编译，清理，测试，打包，部署命令是可以几个命令同时组合起来使用的，常用的命令组合如下：

1、先清理再编译：“mvn clean compile”，如下所示：

```
管理员: C:\Windows\system32\cmd.exe
E:\MavenProject\hello>mvn clean compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Hello 0.0.1-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ Hello ---
[INFO] Deleting E:\MavenProject\hello\target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Hello ---
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Hello ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding GBK, i.e. build is platform dependent!
[INFO] Compiling 1 source file to E:\MavenProject\hello\target\classes
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 1.079 s
[INFO] Finished at: 2014-10-26T11:00:28+08:00
[INFO] Final Memory: 9M/22M
[INFO]
半:
```

还有的就是“mvn clean test”，“mvn clean package”，“mvn clean install”，这些组合命令都比较常用。

以上就是关于Maven构建项目的各个过程演示。

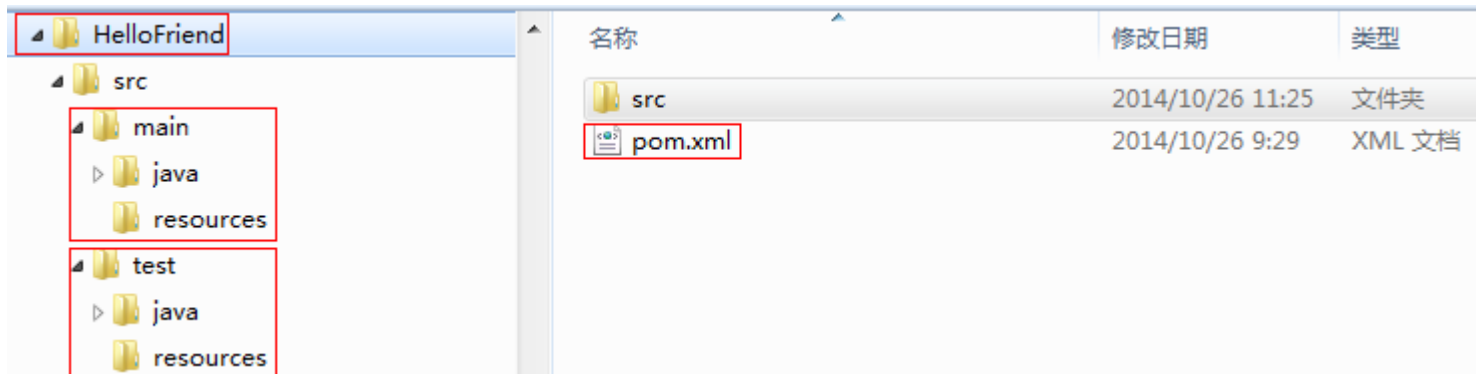
## 二、在别的项目中使用通过Maven安装生成的项目的jar包

在上面，我们使用mvn install命令将hello这个项目打包成了Hello-0.0.1-SNAPSHOT.jar包并且发布到本地的maven仓库E:\repository\me\gac\maven\Hello\0.0.1-SNAPSHOT中，下面我们来看看如何在别的项目中使用Hello-0.0.1-SNAPSHOT.jar

1、新建HelloFriend项目，同时建立Maven约定的目录结构和pom.xml文件

```
HelloFriend
|
|--src
|
|----main
|
|-----java
|
|-----resources
|
|----test
|
|-----java
|
|-----resources
|
--pom.xml
```

如下图所示：



2、编辑项目HelloFriend根目录下的pom.xml，添加如下的代码：



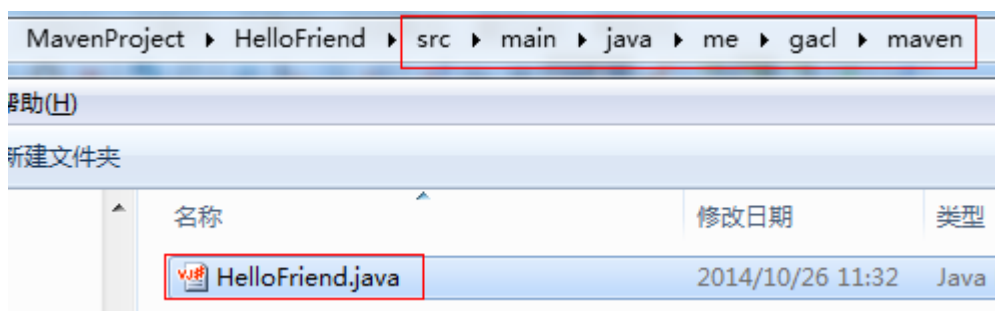
```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
2 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>me.gacl.maven</groupId>
5   <artifactId>HelloFriend</artifactId>
6   <version>0.0.1-SNAPSHOT</version>
7   <name>HelloFriend</name>
8
9   <!--添加依赖的jar包-->
10  <dependencies>
11    <!--项目要使用到junit的jar包，所以在这里添加junit的jar包的依赖-->
12    <dependency>
13      <groupId>junit</groupId>
14      <artifactId>junit</artifactId>
15      <version>4.9</version>
16      <scope>test</scope>
17    </dependency>
18    <!--项目要使用到Hello的jar包，所以在这里添加Hello的jar包的依赖-->
19    <dependency>
20      <groupId>me.gacl.maven</groupId>
21      <artifactId>Hello</artifactId>
22      <version>0.0.1-SNAPSHOT</version>
23      <scope>compile</scope>
24    </dependency>
```

25 </dependencies>

26 </project>



3、在src/main/java/me/gacl/maven目录下新建文件HelloFriend.java，如下图所示：



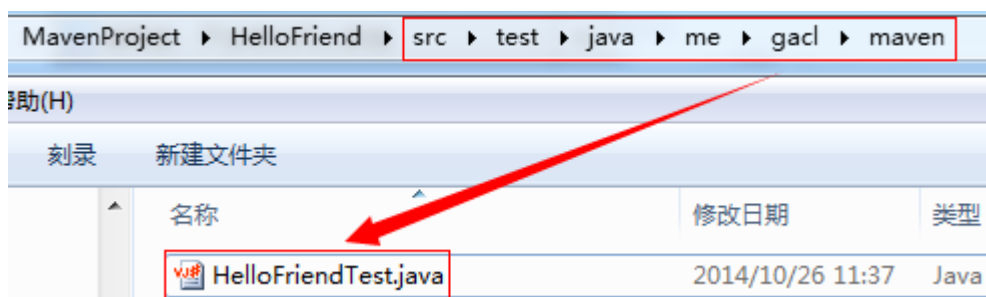
HelloFriend.java的代码如下：



```
1 package me.gacl.maven;
2
3 import me.gacl.maven.Hello;
4
5 public class HelloFriend {
6
7     public String sayHelloToFriend(String name) {
8
9         Hello hello = new Hello();
10        String str = hello.sayHello(name)+" I am "+this.getMyName();
11        System.out.println(str);
12        return str;
13    }
14
15    public String getMyName() {
16        return "John";
17    }
18 }
```



4、在/src/test/java/me/gacl/maven目录下新建测试文件HelloFriendTest.java，如下图所示：



HelloFriendTest.java的代码如下：



```
1 package me.gacl.maven;
2
```

```

3 import static junit.framework.Assert.assertEquals;
4 import org.junit.Test;
5 import me.gacl.maven.Hello;
6
7 public class HelloFriendTest {
8
9     @Test
10     public void tesHelloFriend() {
11
12         HelloFriend helloFriend = new HelloFriend();
13         String results = helloFriend.sayHelloToFriend("gacl");
14         assertEquals("Hello gacl! I am John",results);
15     }
16 }

```



5、在HelloFriend目录下执行命令“mvn package”测试Hello-0.0.1-SNAPSHOT.jar里面的类是否引用成功，如下所示：

```

-----
T E S T S
-----
Running me.gacl.maven.HelloFriendTest
Hello gacl! I am John
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.05 sec

Results :

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

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ HelloFriend ---
[INFO] Building jar: E:\MavenProject\HelloFriend\target\HelloFriend-0.0.1-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.619 s
[INFO] Finished at: 2014-10-26T11:40:51+08:00
[INFO] Final Memory: 11M/29M
[INFO] -----

```

执行Junit测试，测试通过

打包成功

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

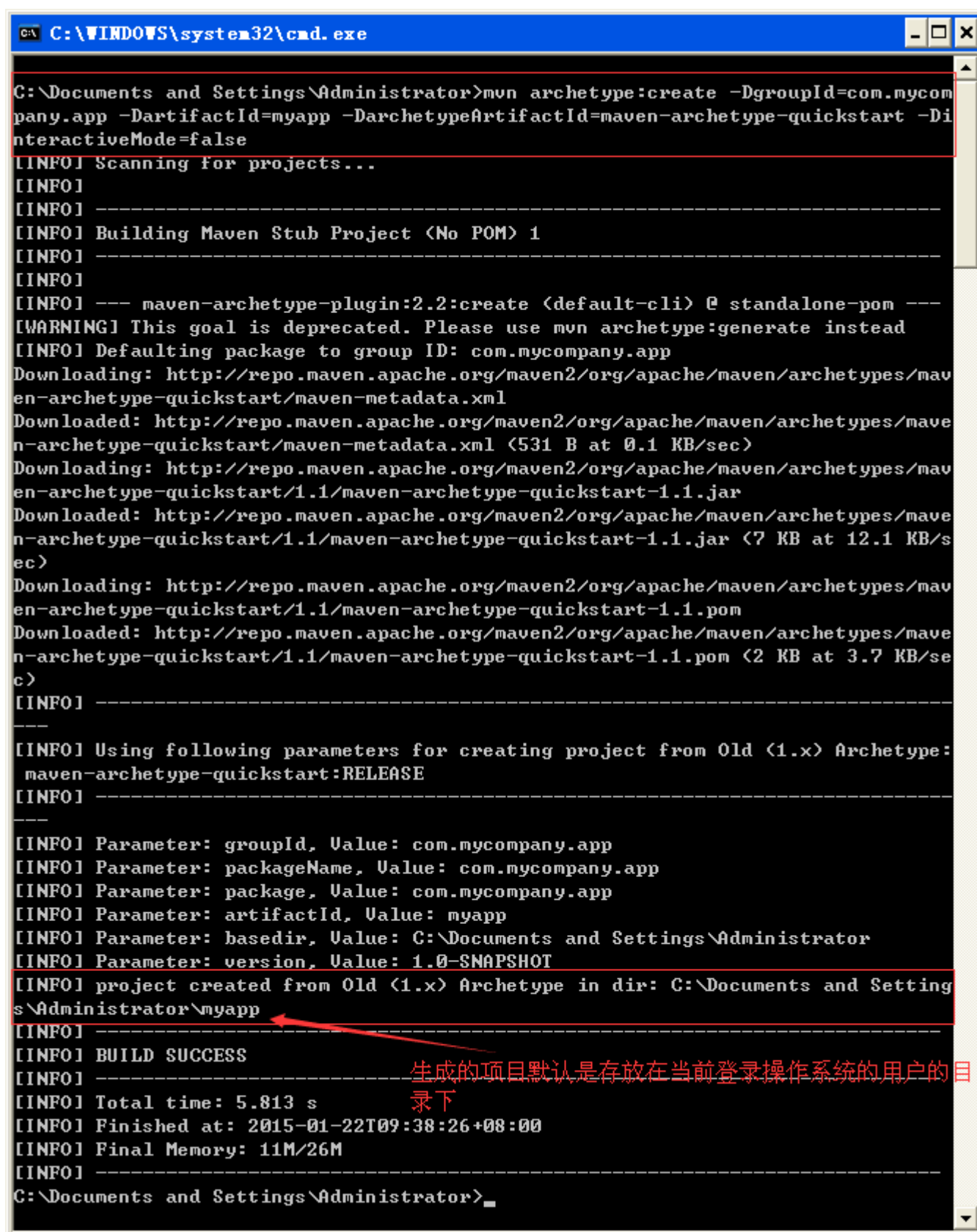
1、使用mvn archetype:generate命令，如下所示：

```
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：



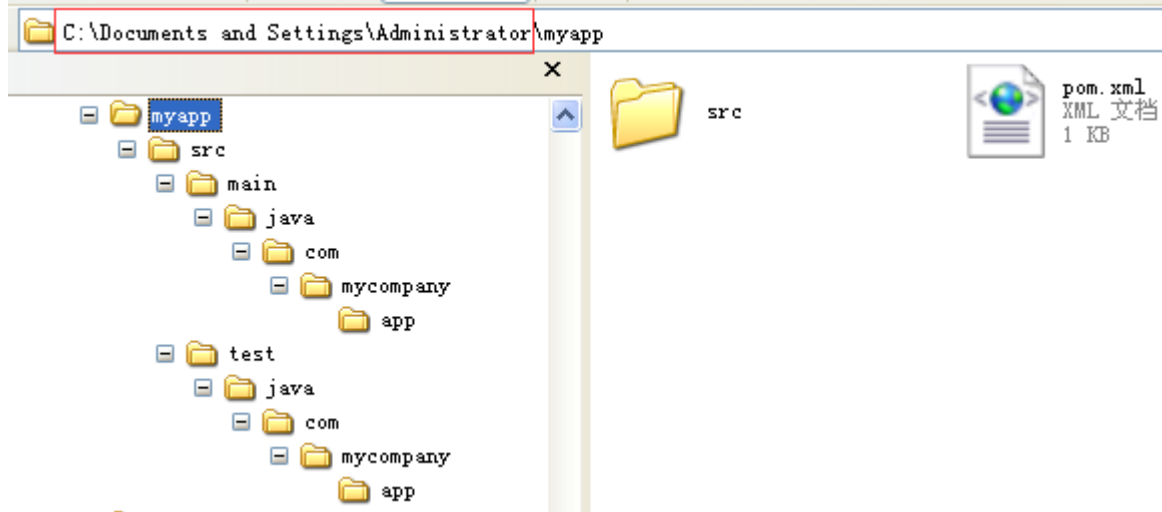
```
C:\WINDOWS\system32\cmd.exe

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]
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-quickstart:RELEASE
[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: 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>
```

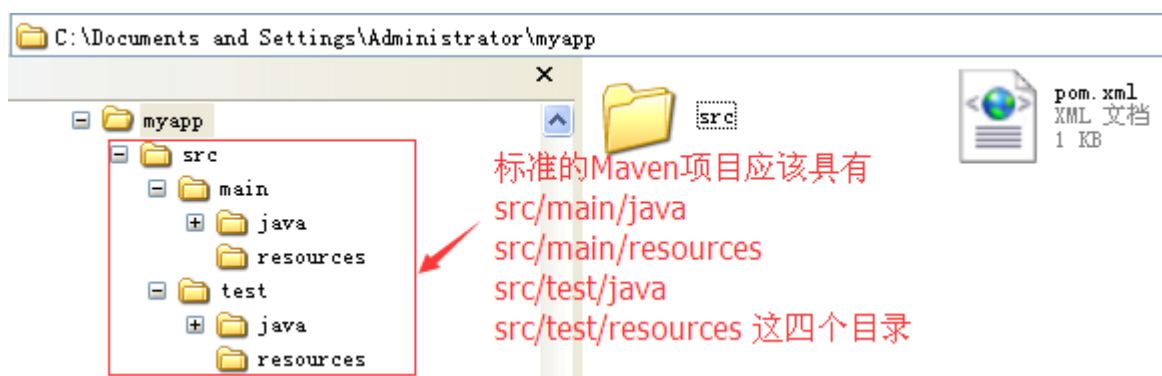
BUILD SUCCESS就表示项目构建成功，当前用户目录下（即C:\Documents and Settings\Administrator）下构建了一个Java Project叫做myapp。

构建好的Java项目的目录结构如下：

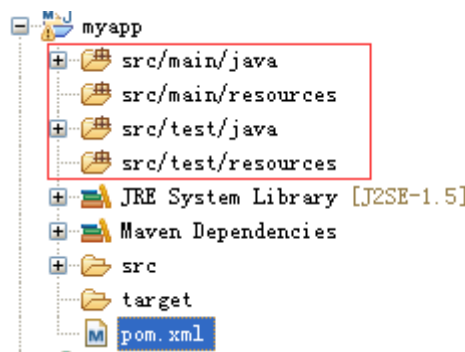




可以看到，Maven帮我们创建的项目是一个标准的Maven项目，不过目前Maven只是帮我们生成了src/main/java(存放项目的源代码)和src/test/java(存放测试源代码)这两个目录，但实际项目开发中我们一般都会有配置文件，例如log4j.properties，所以我们还需要手动创建src/main/resources(存放项目开发中用到的配置文件，如存放log4j.properties等)和src/test/resources(存放测试时用到的配置文件)，如下图所示：

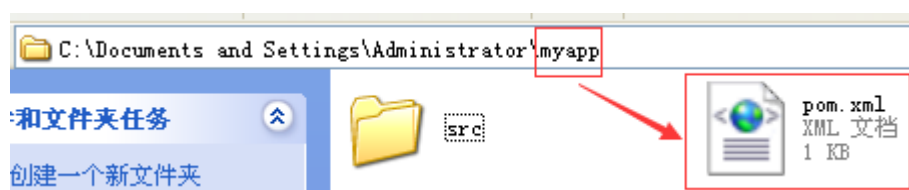


然后我们就可以将创建好的myapp项目导入到Eclipse中进行开发了，如下图所示：



## 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>
```



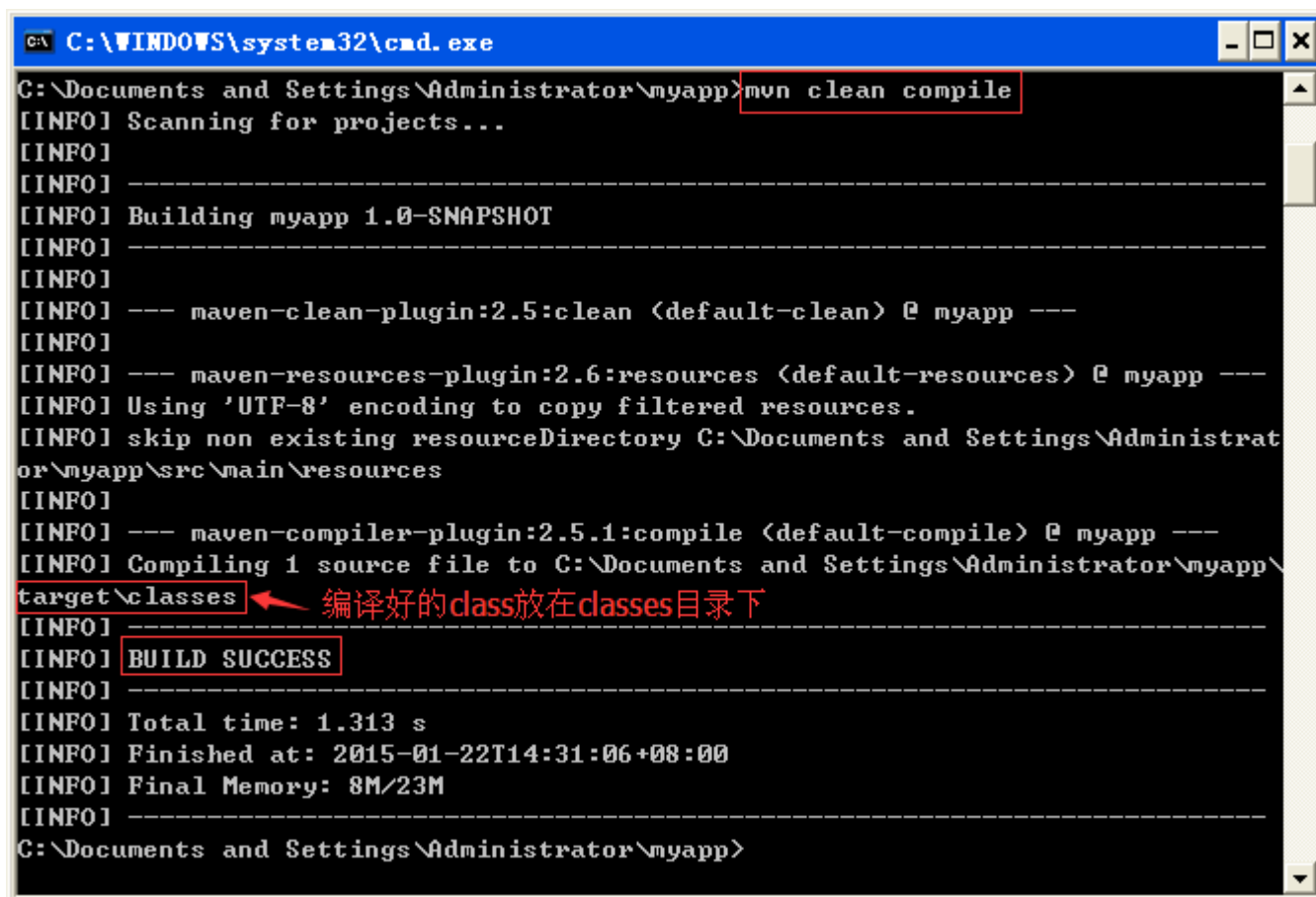
pom.xml文件的节点元素说明:	<project>	pom文件的顶级节点
<modelVersion>	object model版本, 对Maven2和Maven3来说, 只能是4.0.0	
<groupId>	项目创建组织的标识符, 一般是域名的倒写	
<artifactId>	定义了项目在所属组织的标识符下的唯一标识, 一个组织下可以有多个项目	
<version>	当前项目的版本, SNAPSHOT, 表示是快照版本, 在开发	
<packaging>	打包的方式, 有jar、war、ear等	
<name>	项目的名称	
<url>	项目的地址	
<properties>	属性配置,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

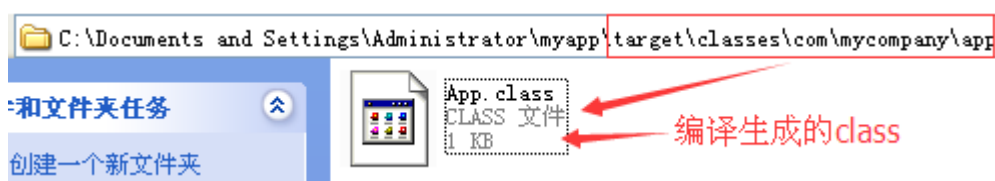
编译源程序，进入命令行，切换到myapp目录，执行命令：mvn clean compile，如下图所示：



```
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\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] BUILD SUCCESS
[INFO]
[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：



### 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

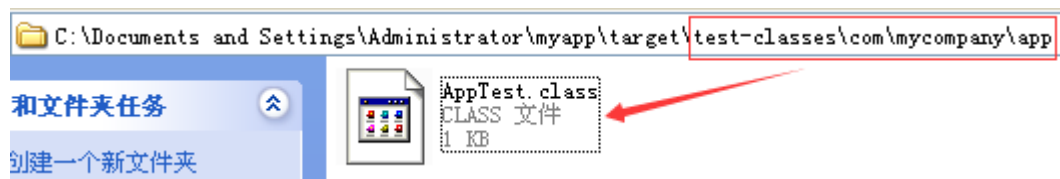
Results :

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

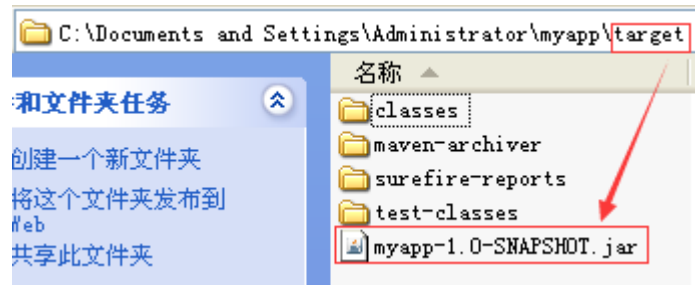
进入命令行，切换到myapp目录，执行命令：mvn clean package，执行打包命令前，会先执行编译和测试命令，如下图所示：

```
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

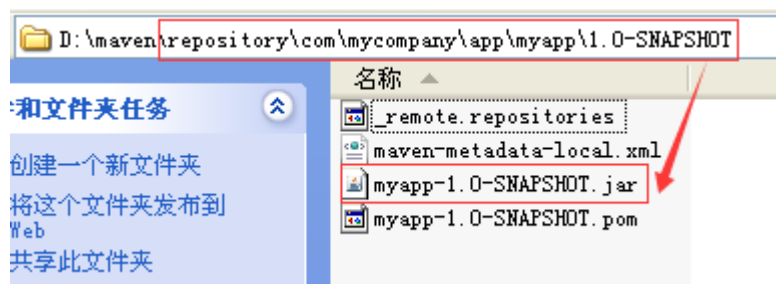
进入命令行，切换到my-app目录，执行命令：mvn clean install，执行安装命令前，会先执行编译、测试、打包命令，如下图所示：

```
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，有时甚至会更长，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：

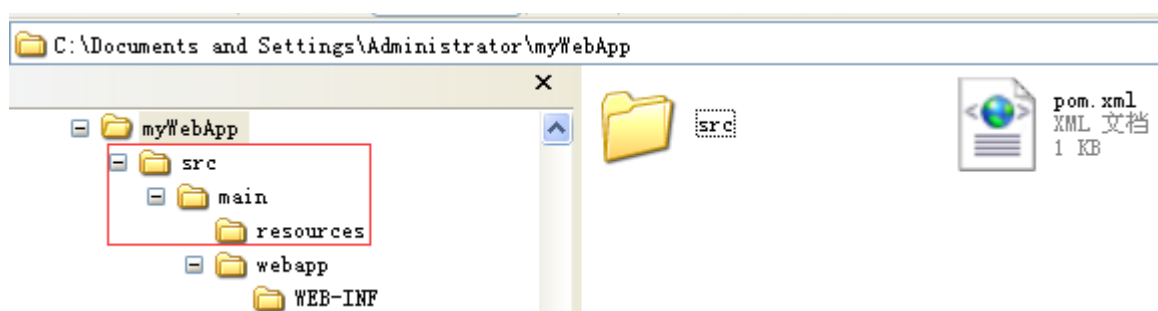
```
C:\WINDOWS\system32\cmd.exe

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。

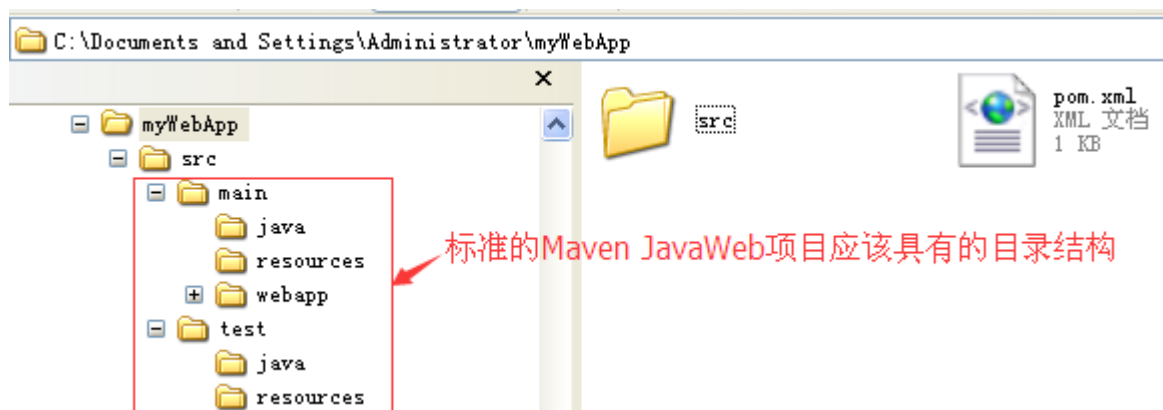
创建好的JavaWeb项目的目录结构如下：



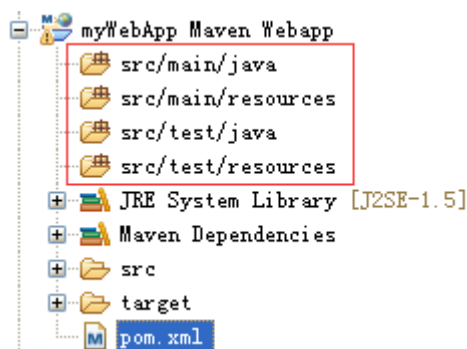
创建好的JavaWeb项目中目前只有src/main/resources目录，因此还需要手动添

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

如下图所示：

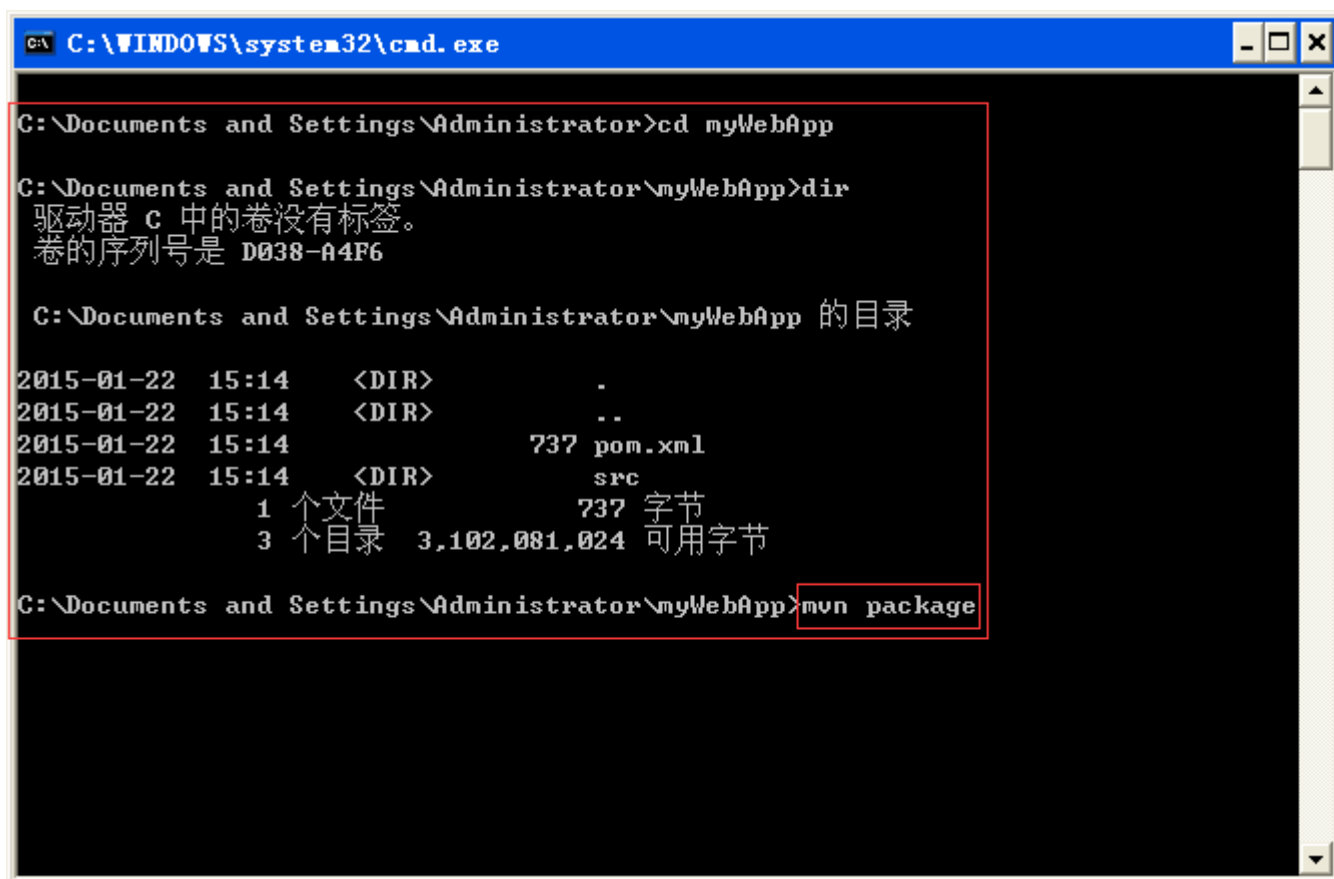


接着我们就可以将创建好的JavaWeb导入Eclipse中进行开发了，如下图所示：



## 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：



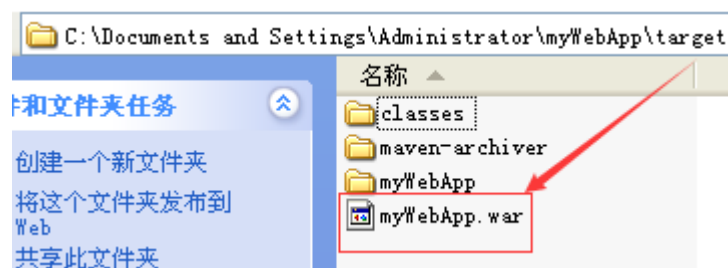


```
C:\WINDOWS\system32\cmd.exe

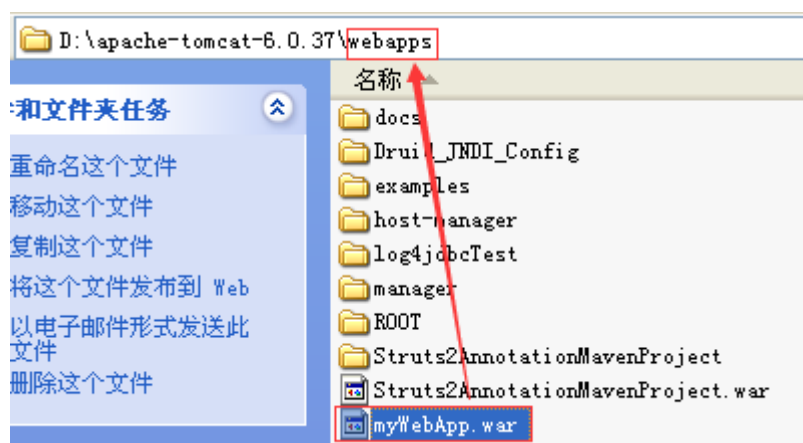
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ myWebApp ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ myWebApp ---
[INFO] Packaging webapp
[INFO] Assembling webapp [myWebApp] in [C:\Documents and Settings\Administrator\myWebApp\target\myWebApp]
[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

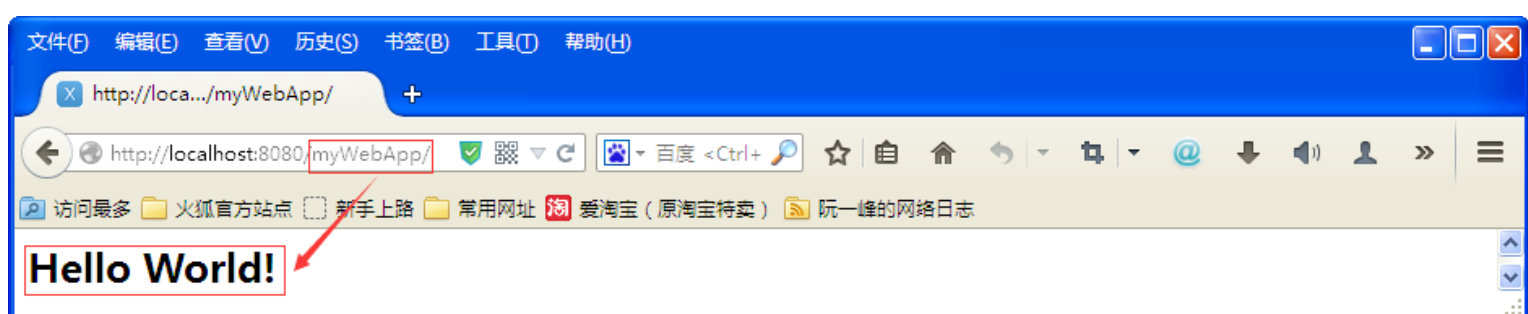
打包成功，在myWebApp\target目录下生成了一个myWebApp.war文件，如下图所示：



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：

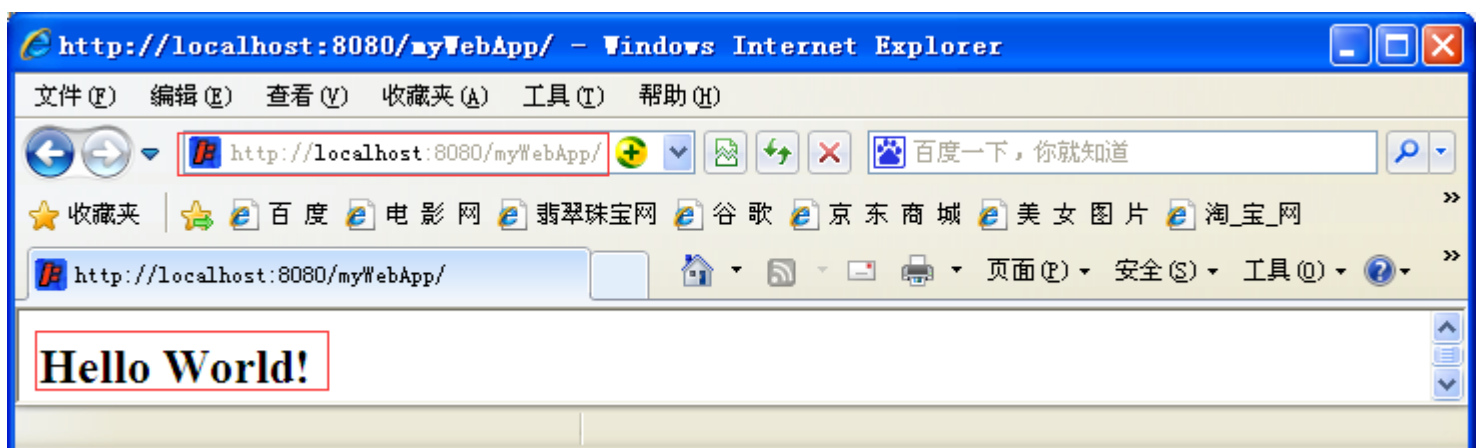
```
C:\Documents and Settings\Administrator\myWebApp>mvn jetty:run
```

```
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。