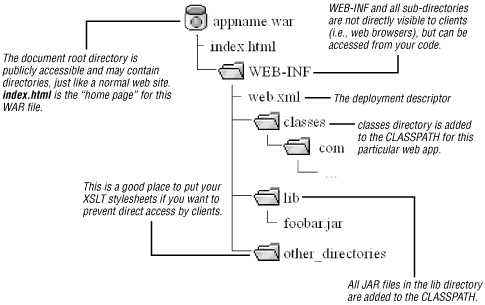
      最近在网上看到一篇介绍maven基础知识的文章，觉得对初学Maven的朋友一定有帮助。水平有限，翻译的不好，请大家见谅。

### 介绍

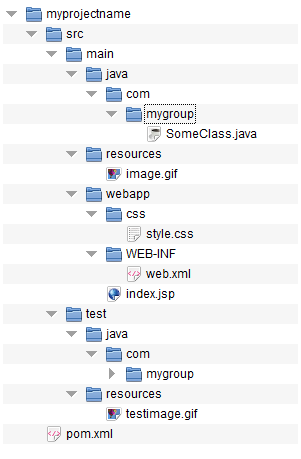
      在处理WEB应用的时候，最终使用的工程文件是以War包的形式交付。Maven编译系统可以轻松的创建War包。接下来就让我们看看Maven是如何把一个源文件的工程转换成War包的。

Maven 版本 Apache Maven 3.0.4



### 工程实例

       让我们来看看这个非常典型的Maven化的WEB工程



对应的POM.xml如下：

<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/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>mygroup.com</groupId>

<artifactId>myprojectname</artifactId>

<packaging>war</packaging>

<version>1.0-SNAPSHOT</version>

<name>myprojectname Maven Webapp</name>

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

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<finalName>myprojectname</finalName>

</build>

</project>

 我们用此命令War包

mvn package

C:\Projects\myprojectname>mvn package

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building myprojectname Maven Webapp 1.0-SNAPSHOT

[INFO] ------------------------------------------------------------------------

[INFO]

...

[INFO] --- maven-war-plugin:2.1.1:war (default-war) @ myprojectname ---

[INFO] Packaging webapp

[INFO] Assembling webapp [myprojectname] in [C:\Projects\myprojectname\target\myprojectname]

[INFO] Processing war project

[INFO] Copying webapp resources [C:\Projects\myprojectname\src\main\webapp]

[INFO] Webapp assembled in [18 msecs]

[INFO] Building war: C:\Projects\target\myprojectname.war

...

 War生成在根目录下

/target/myprojectname.war

如下图，概况Maven生成War包过程



### Maven 默认配置

     我们都知道Maven可以很容易的把源文件工程创建为War包，但是POM文件中什么也没有设置。这是怎么回事啊？实际上Maven有自己默认的设置。这称之为 “**convention over configuration**”，Maven在配置中提供默认值。

 第一，因为有一些Maven插件本身就与Maven 的生命周期绑定在一起。例如，在编译阶段使用

**compiler：compile** 作为默认命令。这就意味着当执行到编译阶段compiler plugin 被调用执行。如果选择生成WAR，那么 war：war 会与这个阶段绑定。

第二，当没有明确设置参数的时候，插件都会有自己默认值。例如 **compiler：compile** 目标有个参数是 compilerId。当默认值是 javac就意味着JDK 会被使用。当需要生成其他形式时可以重写此配置。

第三，一些设置包含在 **Super POM,**此文件是POM文件默认继承的。从Mavne3 起 Super POM 被放在

maven\_dir/lib/maven-model-builder-3.0.3.jar:org/apache/maven/model/pom-4.0.0.xml

 在这里我们可以发现很多默认的配置信息

<build>

<directory>${project.basedir}/target</directory>

<outputDirectory>${project.build.directory}/classes</outputDirectory>

<finalName>${project.artifactId}-${project.version}</finalName>

<testOutputDirectory>${project.build.directory}/test-classes</testOutputDirectory>

<sourceDirectory>${project.basedir}/src/main/java</sourceDirectory>

<scriptSourceDirectory>src/main/scripts</scriptSourceDirectory>

<testSourceDirectory>${project.basedir}/src/test/java</testSourceDirectory>

<resources>

<resource>

<directory>${project.basedir}/src/main/resources</directory>

</resource>

</resources>

<testResources>

<testResource>

<directory>${project.basedir}/src/test/resources</directory>

</testResource>

</testResources>

...

</build>

### Maven 生命周期

     在我们的工程中，当执行 **mvn package** 命令，maven会执行它整个生命周期中的六个阶段

process-resources, compile, process-test-resources, test-compile, test and package

每个阶段会包含一个或多个目标。Maven 插件提供目标：一个插件可以有一个或多个目标。例如

Compiler 插件有两个目标：**compiler：compile  和 compiler：testCompile**

我们可以使用 mvn help：describe -Dcmd=phasename 命令列出如下内容

C:\Project\myprojectname>mvn help:describe -Dcmd=package

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building myprojectname Maven Webapp 1.0-SNAPSHOT

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-help-plugin:2.1.1:describe (default-cli) @ myprojectname ---

[INFO] 'package' is a phase corresponding to this plugin:

org.apache.maven.plugins:maven-war-plugin:2.1.1:war

It is a part of the lifecycle for the POM packaging 'war'. This lifecycle includes the following phases:

\* validate: Not defined

\* initialize: Not defined

\* generate-sources: Not defined

\* process-sources: Not defined

\* generate-resources: Not defined

\* process-resources: org.apache.maven.plugins:maven-resources-plugin:2.5:resources

\* compile: org.apache.maven.plugins:maven-compiler-plugin:2.3.2:compile

\* process-classes: Not defined

\* generate-test-sources: Not defined

\* process-test-sources: Not defined

\* generate-test-resources: Not defined

\* process-test-resources: org.apache.maven.plugins:maven-resources-plugin:2.5:testResources

\* test-compile: org.apache.maven.plugins:maven-compiler-plugin:2.3.2:testCompile

\* process-test-classes: Not defined

\* test: org.apache.maven.plugins:maven-surefire-plugin:2.10:test

\* prepare-package: Not defined

\* package: org.apache.maven.plugins:maven-war-plugin:2.1.1:war

\* pre-integration-test: Not defined

\* integration-test: Not defined

\* post-integration-test: Not defined

\* verify: Not defined

\* install: org.apache.maven.plugins:maven-install-plugin:2.3.1:install

\* deploy: org.apache.maven.plugins:maven-deploy-plugin:2.7:deploy

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 2.496s

[INFO] Finished at: Sat May 12 04:30:35 CEST 2012

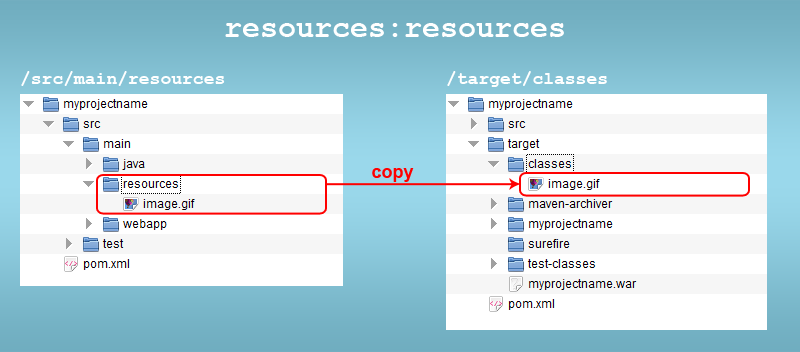
[INFO] Final Memory: 5M/121M

[INFO] ------------------------------------------------------------------------

 下面让我们看看每个目标

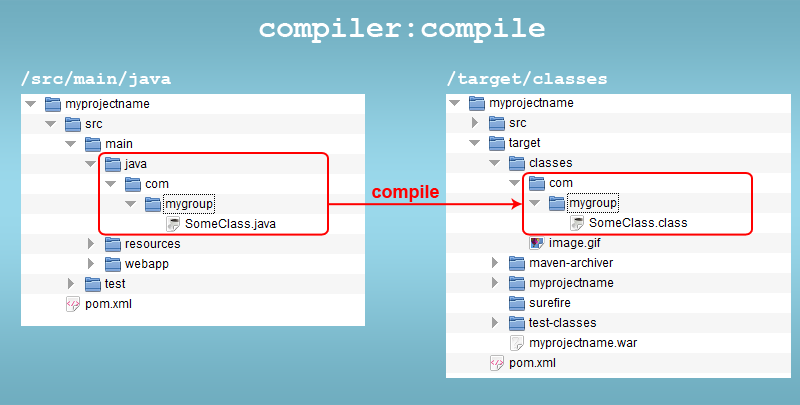
**1.resources:resources**

此目标用来将资源文件夹下的内容拷贝到输出目录



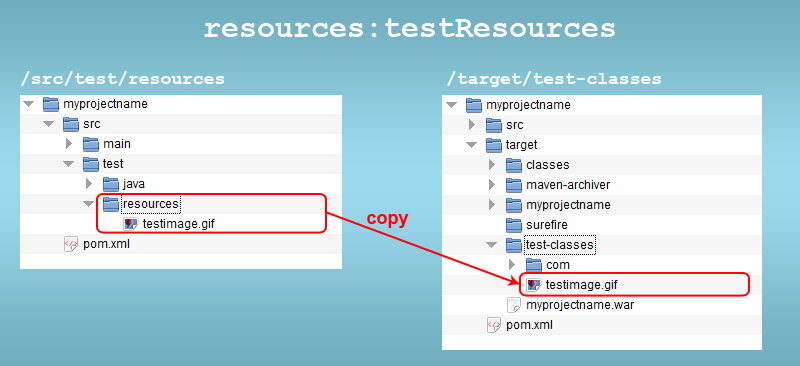
**2.compiler：compile**

此目标编译源项目工程



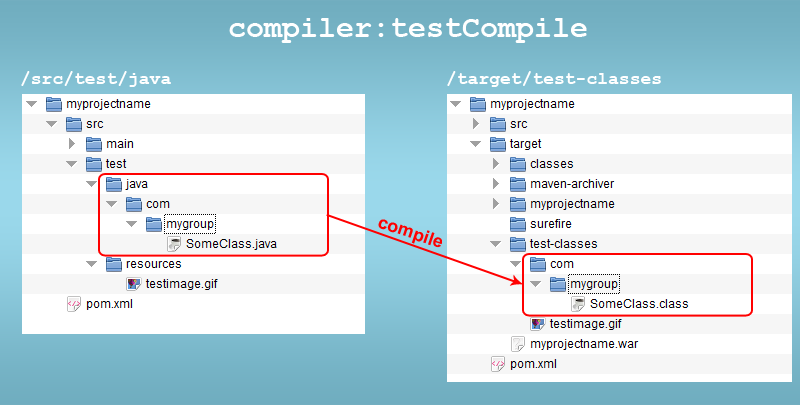
**3.resources:testResources**

此目标拷贝测试资源到测试输出目录



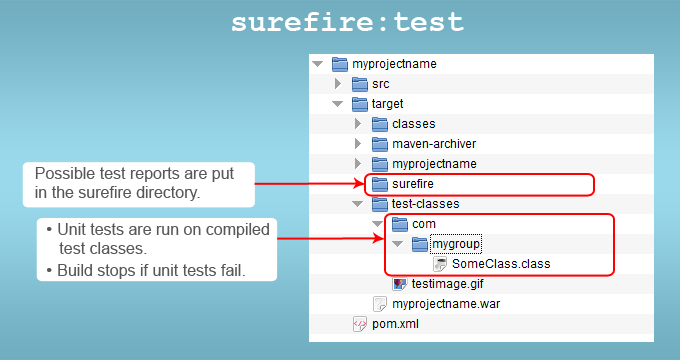
**4.compiler：testCompile**

     此目标编译测试项目



**5.surefire：test**

      此目标执行工程的单元测试，编译的测试类放在 /target/test-classes



**6.war：war**

       此目录创建War包。它会把所有需要的文件放在

/target/myprojectname/

       而后将他们打包生成 \*\*.war。其中一个步骤是将   /src/main/webapp/   拷贝到输出目录。

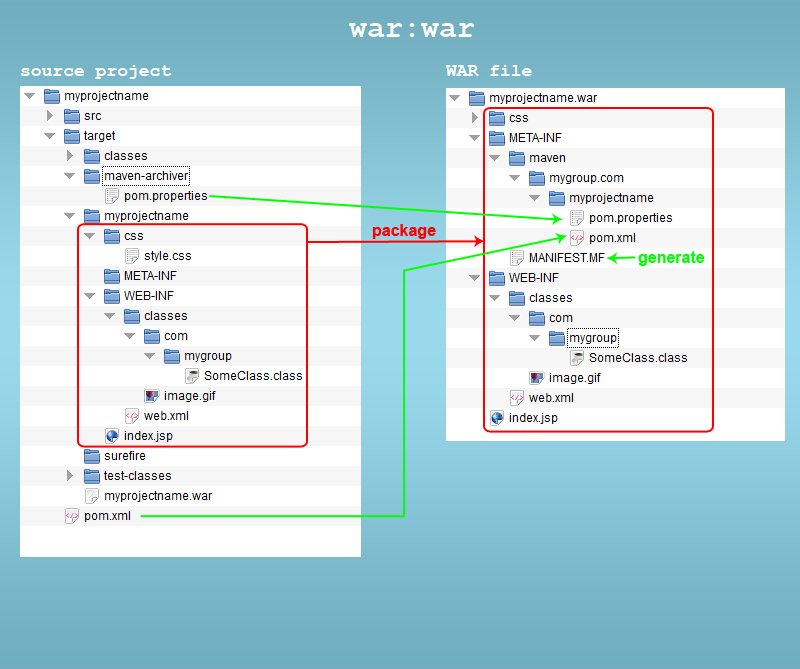


War插件另外一个重要步骤是拷贝Class文件到到 **WEB-INF/classes**目录和项目所依赖的jar包到 **WEB-INF/lib**目录。

默认情况下，插件还包含两个Maven描述文件：

1. META-INF/maven/${groupId}/${artifactId}/pom.xml
2. pom.properties 文件，META-INF/maven/${groupId}/${artifactId}/pom.properties

**#Generated by Maven #Sat May 12 00:50:42 CEST 2012 version=1.0-SNAPSHOT groupId=mygroup.com artifactId=myprojectname**



最终的War包放在**/target/**目录下。

### 项目依赖

      pom.xml文件会有一个默认的（JUnit）依赖。我们可以加另外一个常用的Jar  log4j。

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.16</version>

</dependency>

当没有设置依赖范围 **（** **scope）**，默认为 **compile scope**。这就意味着此依赖在编译、测试、运行阶段都可以得到。

只要是运行中会用到的jar包，都会拷贝到**/WEB-INF/lib**目录

