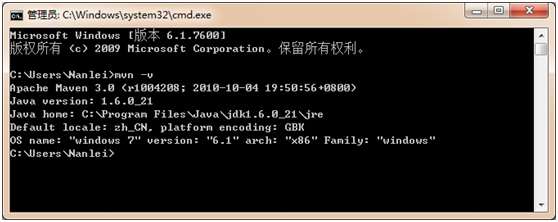
**[用Maven构建Java Web开发环境（Jetty容器）之一](http://sarin.iteye.com/blog/784275)**

**博客分类：**

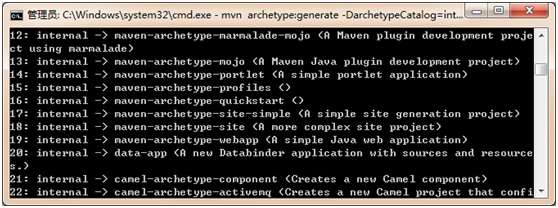
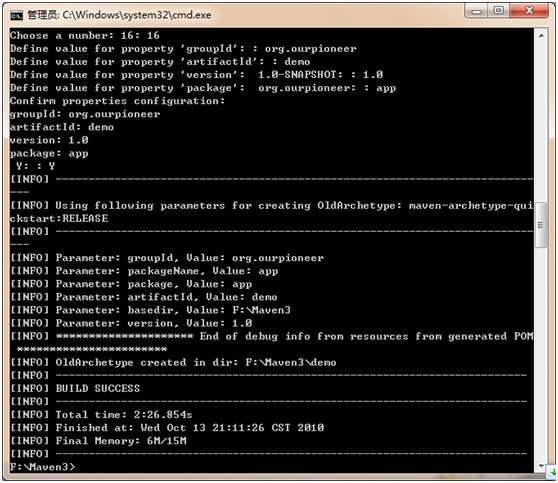
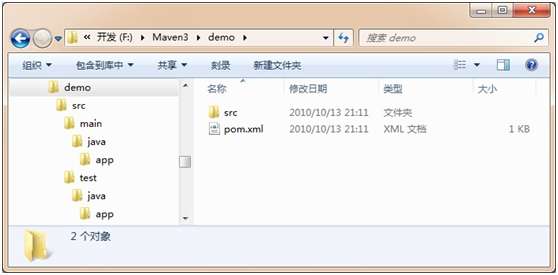
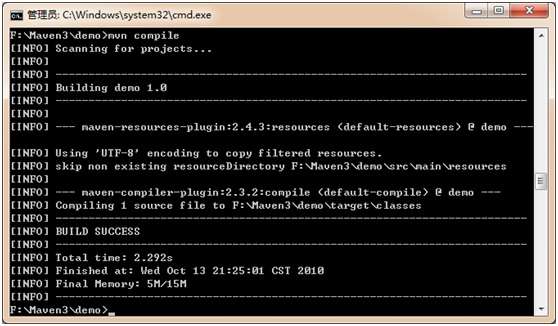
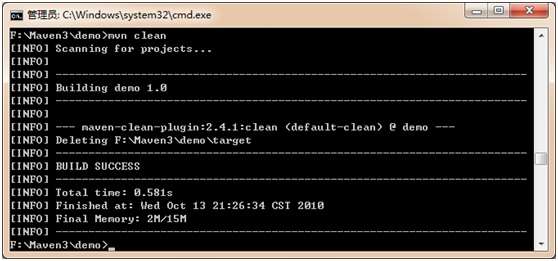
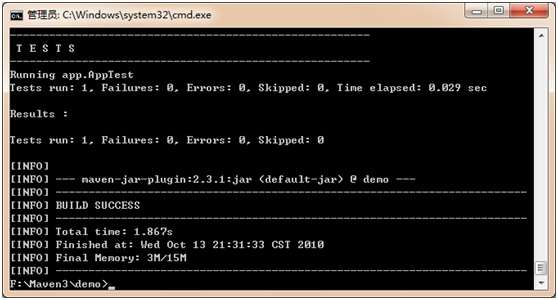
* [积少成多](http://sarin.iteye.com/category/93571)

[maven](http://www.iteye.com/blogs/tag/maven)[Web](http://www.iteye.com/blogs/tag/Web)[Java](http://www.iteye.com/blogs/tag/Java)[项目管理](http://www.iteye.com/blogs/tag/%E9%A1%B9%E7%9B%AE%E7%AE%A1%E7%90%86)[应用服务器](http://www.iteye.com/blogs/tag/%E5%BA%94%E7%94%A8%E6%9C%8D%E5%8A%A1%E5%99%A8)

    Maven是构建Java应用的优秀管理工具，是Apache软件基金会的顶级项目。其具有开发目录构建和管理，统一管理第三方依赖，编译，测试，打包，发布等功能。最大的特点要数第三方依赖管理了，因为其它功能都能从Ant中找到踪影。以往每个项目独立管理依赖，使得每个项目打包后都是巨大无比的，而真正的程序文件却没有多少，Maven接管第三方依赖后，我们的应用就只有程序文件了，没有了jar包轻巧很多，而统一的管理，使得我们可以随意获取jar资源。   
    初学者使用Maven总是感觉很难上手，其中也有一部分人是被Maven下载资源那疯狂的做法吓退的，这就是Maven的精巧之处。从Apache官方网站下载的Maven工具仅仅2M大小，其中是不包含我们要使用的依赖的，那么就要从网络中获取这些依赖，就是那个疯狂下载的过程。下面我们从头开始构建Maven环境，直到运行Jetty容器执行Web应用，这一切都很简单。   
    下载Maven的开发包之后，解压到一个固定的文件夹，建议不要使用汉字并在磁盘的根目录下，这是比较好的实践。那么本文以Maven 3为例来说明，Maven的位置是F:\Maven 3目录。解压后就要添加系统环境变量，这里要说明，因为Maven也是Java应用，所以JDK就必须要提前安装并设置环境遍历。将JAVA\_HOME和M3\_HOME单独设置，防止其使用时不能用Path变量下发现所需路径。下面是JAVA\_HOME和M3\_HOME的配置位置，这个因人而异，但是最好固定不要随意修改。   
   
   
    在Path中，这样来添加即可，非常简单。   
   
    至此Maven已经安装完成了，下面我们来验证安装，在CMD中输入mvn –v   
   
    可以看到，已经识别mvn命令了。做点简单的设置，就是第三方依赖下载的时候的存放位置，默认是在C盘用户目录的当前用户文件夹下（如C:\用户\Admin，这里是Windows 7的命名方式），在这里Maven会创建.m2文件夹，这里面就会存放下载的类库。当然我不想这些类库放在C盘下，不是特别安全，我想让它们存放在Maven安装目录下，那么就可以这样进行：在Maven安装目录的conf下打开settings.xml，这是对Maven进行设置的XML文件，这里我们简单设置第三方依赖的位置，如下设置：

**Xml代码  [收藏代码](javascript:void())**

1. **<localRepository>**F:/Maven3/repository**</localRepository>**

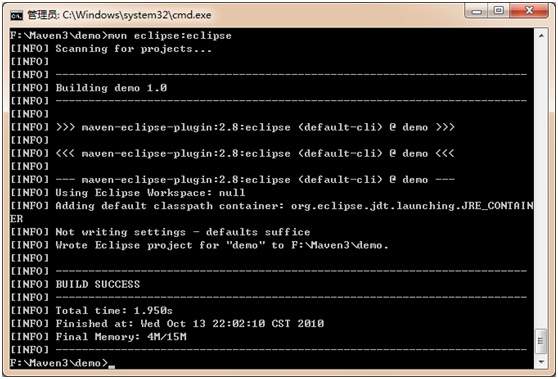
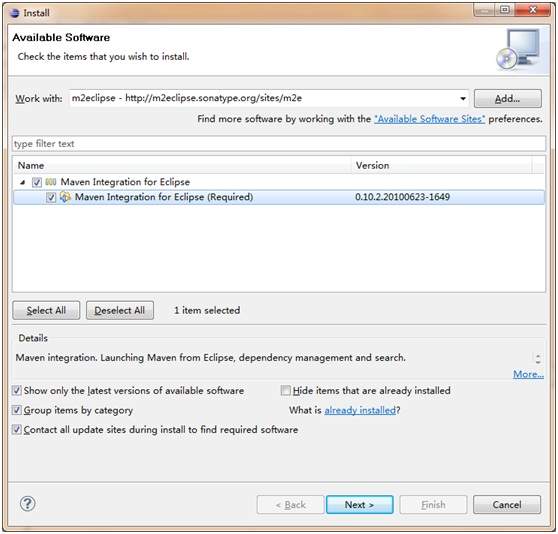
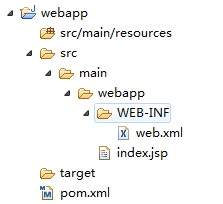
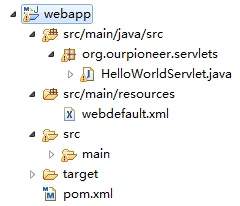
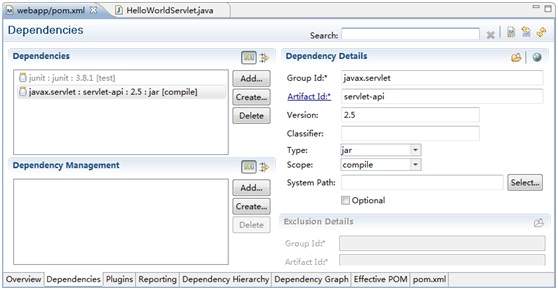
    这样，那些疯狂下载的依赖就在这个位置下了，要留给足够大的空间，因为依赖下载很可能占用很大的一块空间。为什么？下载后去看看它的真实面目就知道了。   
    下面我们就来创建应用目录，使用mvn archetype:generate命令一步一步根据提示，选择项目类型，添加groupId和artifactId就可以创建项目了，我们来看一下，这里执行mvn archetype:generate -DarchetypeCatalog=internal命令，要手工设置一下项目类型分类属性，可能因为类型太多导致屏幕显示不全，那么我们就简单来看几个常用的就行，就是internal类型。执行该命令后，在控制台我们看到有如下的内容。（切记这个操作位置就是生成项目目录的位置）   
   
    这里我们先选择16是maven-archetype-quickstart，就是基本的JavaSE项目结构的构建，19是maven-archetype-webapp，用来创建简单的Java Web应用，我们先选择16来看看。   
   
    选择16之后，Maven提示输出groupId，就是我们的项目命名组成之一，没什么过多解释的东西，artifactId用来标记项目名称，它们两个是联合使用的。而package定义的是包名，我们在文件夹内就发现了demo命名的项目文件夹。看看这个目录的结构，这是Maven为我们创建的，就是一个标准的格式：   
   
    根目录下有一个pom.xml，这就是项目的Maven配置文件，暂时不介绍其强大的功能，仅仅简单看一下实际应用，这是基础的目录结构，Maven为我们创建了一个Hello World的小程序，我们来运行它。想要运行这个程序，那么就要编译，mvn的编译命令是mvn compile，这样就为我们编译整个项目，这个操作要在demo目录下完成。   
   
    回头看看demo文件夹，多出一个target目录，这就是我们的类路径了，class文件就在下面。这个操作的逆过程是mvn clean，顾名思义，就是清理，清理生成的可执行部分，仅留下源码部分，如下：   
   
    Maven执行了Deleting操作，删除了target目录。如果想执行程序，是需要class文件或jar文件的，前面的mvn compile命令仅仅是编译，而没有对应用进行打包发布，而mvn package命令可以将应用打包（jar或war等），那么我们就来为项目打包发布：   
   
    这个过程，Maven执行了很多操作，有日志可以细看，其中还有测试过程，这是JUnit的测试。其实我们创建项目时，Maven为我们引入的第一个依赖就是JUnit。下面我们来执行jar文件，运行程序。（当然执行class是一样的效果）   
   
    这样就得到执行效果了，Hello World!   
（未完待续，下一篇将结合Eclipse以图片方式来说明Web项目的配置和运行）

[**用Maven构建Java Web开发环境（Jetty容器）之二**](http://sarin.iteye.com/blog/784697)

**博客分类：**

* [积少成多](http://sarin.iteye.com/category/93571)

[maven](http://www.iteye.com/blogs/tag/maven)[Web](http://www.iteye.com/blogs/tag/Web)[Java](http://www.iteye.com/blogs/tag/Java)[Eclipse](http://www.iteye.com/blogs/tag/Eclipse)[Servlet](http://www.iteye.com/blogs/tag/Servlet)

    本文接上一篇[第一部分](http://sarin.iteye.com/blog/784275)继续来介绍。   
    目前为止我们还是手工命令行方式执行程序的，没有和IDE结合，其实Maven天生就对Eclipse做了集成，我们使用mvn eclipse:eclipse就得到了一个Eclipse的项目结构，在Eclipse中使用import功能就能直接导入到IDE中了。我们来看一下这个过程：   
   
    此时的demo就是Eclipse项目格式的了，出现了.project和.classpath文件。我们在Eclipse中引入这个项目，此时的Eclipse没有安装Maven插件，不能自动运行Maven命令，我们来安装Maven的Eclipse插件M2E。   
   
    在Eclipse的Install New Software中直接选择安装即可，非常简单。下面我们来创建Web项目并导入Eclipse中，在Jetty容器中运行程序。首先执行mvn archetype:generate命令创建。   
   
    可以看到，刚创建的web项目结构包含了resources目录，而没有java代码目录，我们需要手工创建，在Eclipse中创建source folder，路径为src/main/java/src，现在我们得到如下一个项目结构，新建一个Servlet用于测试。   
   
    此时，项目中没有Servlet的依赖，需要添加，我们使用m2eclipse插件来直接添加依赖，如下所示：   
   
    相应的XML为：

**Xml代码  [收藏代码](javascript:void())**

1. **<dependency>**
2. **<groupId>**javax.servlet**</groupId>**
3. **<artifactId>**servlet-api**</artifactId>**
4. **<version>**2.5**</version>**
5. **<type>**jar**</type>**
6. **<scope>**compile**</scope>**
7. **</dependency>**

    下面就可以编写Servlet了，很简单，就输出HelloWorld吧。

**Java代码  [收藏代码](javascript:void())**

1. **package** org.ourpioneer.servlets;
3. **import** java.io.IOException;
4. **import** java.io.PrintWriter;
6. **import** javax.servlet.ServletException;
7. **import** javax.servlet.http.HttpServlet;
8. **import** javax.servlet.http.HttpServletRequest;
9. **import** javax.servlet.http.HttpServletResponse;
11. **public** **class** HelloWorldServlet **extends** HttpServlet {
13. **protected** **void** doGet(HttpServletRequest request,
14. HttpServletResponse response) **throws** ServletException, IOException {
15. **this**.process(request, response);
16. }
18. **protected** **void** doPost(HttpServletRequest request,
19. HttpServletResponse response) **throws** ServletException, IOException {
20. **this**.process(request, response);
21. }
23. **private** **void** process(HttpServletRequest request,
24. HttpServletResponse response) **throws** ServletException, IOException {
25. response.setContentType("text/html;charset=utf-8");
26. response.setCharacterEncoding("utf-8");
27. PrintWriter out = response.getWriter();
28. String title="Webapp Demo";
29. out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">");
30. out.println("<html xmlns=\"http://www.w3.org/1999/xhtml\">");
31. out.println("<head>");
32. out.println("<meta http-equiv=\"Content-Type\" content=\"text/html;charset=utf-8\" />");
33. out.println("<title>" + title + "</title>");
34. out.println("<body>");
35. out.println("<h1>Hello World!</h1>");
36. out.println("</body>");
37. out.println("</html>");
38. }
39. }

    然后不能忘了在web.xml中配置这个Servlet，这里是Servlet 2.5的规范，不是Servlet 3，不能用注解。这也很简单。

**Xml代码  [收藏代码](javascript:void())**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"**>**
7. **<display-name>**Archetype Created Web Application**</display-name>**
9. **<servlet>**
10. **<servlet-name>**helloworld**</servlet-name>**
11. **<servlet-class>**org.ourpioneer.servlets.HelloWorldServlet**</servlet-class>**
12. **</servlet>**
14. **<servlet-mapping>**
15. **<servlet-name>**helloworld**</servlet-name>**
16. **<url-pattern>**/helloworld**</url-pattern>**
17. **</servlet-mapping>**
18. **</web-app>**

    程序都有了，剩下就是运行了，Maven既然天生和Jetty是一对儿，这里我们就使用Jetty吧，在Maven中配置Jetty，首先是webdefault.xml要准备好，它是配置Jetty的，这个可以从Jetty的包中找到，并复制到resources下，这里多说一点，默认Jetty运行时是锁定JS/CSS等静态文件的，如果想在Jetty运行时也能修改它们，要在webdefault.xml中修改如下设置：

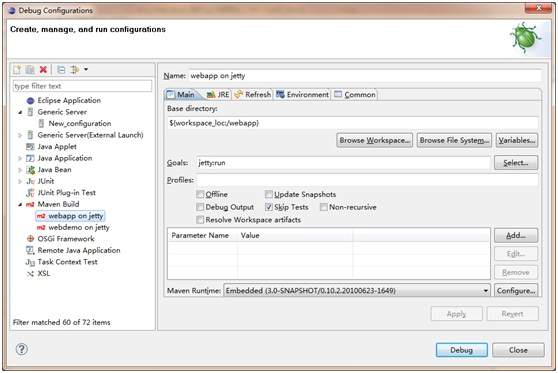
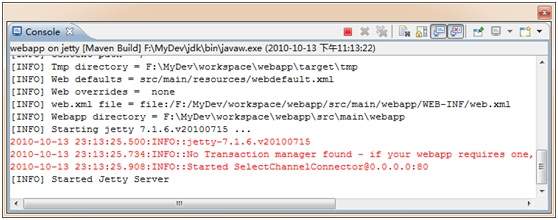
**Xml代码  [收藏代码](javascript:void())**

1. **<init-param>**
2. **<param-name>**useFileMappedBuffer**</param-name>**
3. **<param-value>**false**</param-value>**
4. **</init-param>**

    Jetty也准备了，运行命令是jetty:run，这要在Maven中设置，那么需要在pom.xml中加入Jetty的插件的设置信息。这里直接贴出其整体构建信息。

**Xml代码  [收藏代码](javascript:void())**

1. **<build>**
2. **<finalName>**webapp**</finalName>**
3. **<sourceDirectory>**src/main/java/src**</sourceDirectory>**
4. **<testSourceDirectory>**src/test**</testSourceDirectory>**
5. **<plugins>**
6. **<plugin>**
7. **<groupId>**org.apache.maven.plugins**</groupId>**
8. **<artifactId>**maven-compiler-plugin**</artifactId>**
9. **<version>**2.0.2**</version>**
10. **<configuration>**
11. **<source>**1.6**</source>**
12. **<target>**1.6**</target>**
13. **<encoding>**utf-8**</encoding>**
14. **</configuration>**
15. **</plugin>**
16. **<plugin>**
17. **<groupId>**org.apache.maven.plugins**</groupId>**
18. **<artifactId>**maven-resources-plugin**</artifactId>**
19. **<configuration>**
20. **<encoding>**UTF-8**</encoding>**
21. **</configuration>**
22. **</plugin>**
23. **<plugin>**
24. **<groupId>**org.mortbay.jetty**</groupId>**
25. **<artifactId>**jetty-maven-plugin**</artifactId>**
26. **<version>**7.1.6.v20100715**</version>**
27. **<configuration>**
28. **<stopKey>**stop**</stopKey>**
29. **<stopPort>**5599**</stopPort>**
30. **<webAppConfig>**
31. **<contextPath>**/**</contextPath>**
32. **<defaultsDescriptor>**src/main/resources/webdefault.xml**</defaultsDescriptor>**
33. **</webAppConfig>**
34. **<scanIntervalSeconds>**0**</scanIntervalSeconds>**
35. **<connectors>**
36. **<connector** implementation="org.eclipse.jetty.server.nio.SelectChannelConnector"**>**
37. **<port>**80**</port>**
38. **<maxIdleTime>**60000**</maxIdleTime>**
39. **</connector>**
40. **</connectors>**
41. **</configuration>**
42. **</plugin>**
43. **<plugin>**
44. **<groupId>**org.apache.maven.plugins**</groupId>**
45. **<artifactId>**maven-eclipse-plugin**</artifactId>**
46. **<version>**2.7**</version>**
47. **<configuration>**
48. **<addVersionToProjectName>**false**</addVersionToProjectName>**
49. **<useProjectReferences>**false**</useProjectReferences>**
50. **<encoding>**UTF-8**</encoding>**
51. **<wtpmanifest>**false**</wtpmanifest>**
52. **<wtpapplicationxml>**true**</wtpapplicationxml>**
53. **<wtpversion>**1.5**</wtpversion>**
54. **<additionalBuildcommands>**
55. **<buildcommand>**org.eclipse.jdt.core.javabuilder**</buildcommand>**
56. **<buildcommand>**org.eclipse.wst.common.project.facet.core.builder**</buildcommand>**
57. **<buildcommand>**org.eclipse.wst.validation.validationbuilder**</buildcommand>**
58. **</additionalBuildcommands>**
60. **<additionalProjectnatures>**
61. **<nature>**org.springframework.ide.eclipse.core.springnature**</nature>**
62. **<nature>**org.maven.ide.eclipse.maven2Nature**</nature>**
63. **<nature>**org.eclipse.wst.common.project.facet.core.nature**</nature>**
64. **<nature>**org.eclipse.jdt.core.javanature**</nature>**
65. **<nature>**org.eclipse.wst.common.modulecore.ModuleCoreNature**</nature>**
66. **</additionalProjectnatures>**
67. **<classpathContainers>**
68. **<classpathContainer>**org.eclipse.jdt.launching.JRE\_CONTAINER**</classpathContainer>**
69. **</classpathContainers>**
70. **</configuration>**
71. **</plugin>**
72. **<plugin>**
73. **<groupId>**org.apache.maven.plugins**</groupId>**
74. **<artifactId>**maven-war-plugin**</artifactId>**
75. **<version>**2.1-beta-1**</version>**
76. **<configuration>**
77. **<warName>**webapp**</warName>**
78. **</configuration>**
79. **</plugin>**
80. **</plugins>**
81. **</build>**

    此时，更新一下Maven依赖，它们就都自动下载到本地了，到这个过程结束，我们就可以在Eclipse中配置Debug运行了。配置很简单，如下。   
   
    这是Debug模式运行，Run模式下是一样的，用Debug模式可以在Eclipse中断点运行程序，非常便于调试。下面我们就让它跑起来吧。运行命令是jetty:run，Base directory配置是：${workspace\_loc:/应用名}，启动调试，看到如下信息，Jetty就成功启动了。   
   
    这里我们使用了80端口，配置方式在pom.xml中，上面的代码已经体现了。在浏览器中访问地址如下：http://localhost/helloworld，之后，我们就看到了效果。   
   
    本文系作者本人的实践和探索，希望对使用者有用，欢迎交流。

[**maven集成tomcat进行web应用测试**](http://vivus.iteye.com/blog/1561664)

**博客分类：**

* [maven](http://vivus.iteye.com/category/222743)

[maven](http://www.iteye.com/blogs/tag/maven)[tomcat](http://www.iteye.com/blogs/tag/tomcat)[pom](http://www.iteye.com/blogs/tag/pom)[plugin](http://www.iteye.com/blogs/tag/plugin)

1、环境介绍   
maven：apache-maven-3.0.3   
tomcat：apache-tomcat-6.0.32   
jdk：jdk1.5.0\_11   
eclipse：3.7.2   
  
2、配置tomcat-maven-plugin   
在pom.xml中加入如下配置

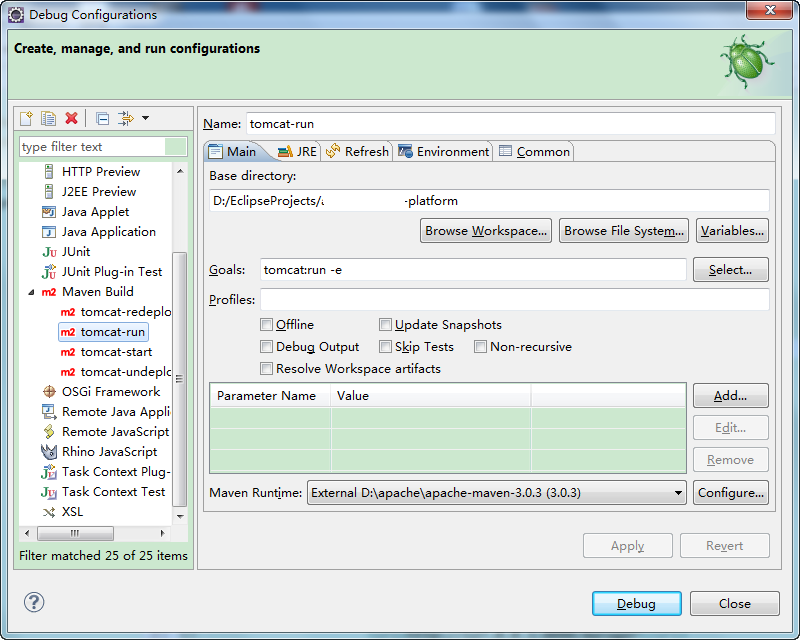
**Xml代码  [收藏代码](javascript:void())**

1. **<plugin>**
2. **<groupId>**org.codehaus.mojo**</groupId>**
3. **<artifactId>**tomcat-maven-plugin**</artifactId>**
4. **<version>**1.1**</version>**
5. **<configuration>**
6. **<server>**tomcat-6.0.32**</server>**
7. **<url>**http://127.0.0.1:8080/manager**</url>**
8. **<uriEncoding>**${encoding}**</uriEncoding>**
9. **<warSourceDirectory>**WebContent**</warSourceDirectory>**
10. **</configuration>**
11. **</plugin>**

说明：

* tomcat-maven-plugin现在已拆分成tomcat7-maven-plugin和tomcat6-maven-plugin了，而groupId也由org.codehaus.mojo改为org.apache.tomcat.maven。

详情可参考：<http://tomcat.apache.org/maven-plugin-2/index.html>

3、集成tomcat测试的方法   
方法一（推荐）：   
运行mvn tomcat:run命令，启动内嵌的tomcat进行测试   
1） 可在命令行直接运行mvn tomcat:run命令   
2） 如果使用eclipse，可通过maven的eclipse插件m2e新增一个maven build运行tomcat:run命令，如图：   
  
  
  
  
方法二：   
运行mvn tomcat:redeploy命令，将web应用发布到外部已启动的tomcat进行测试   
1） 在maven的文件里加入如下设置：

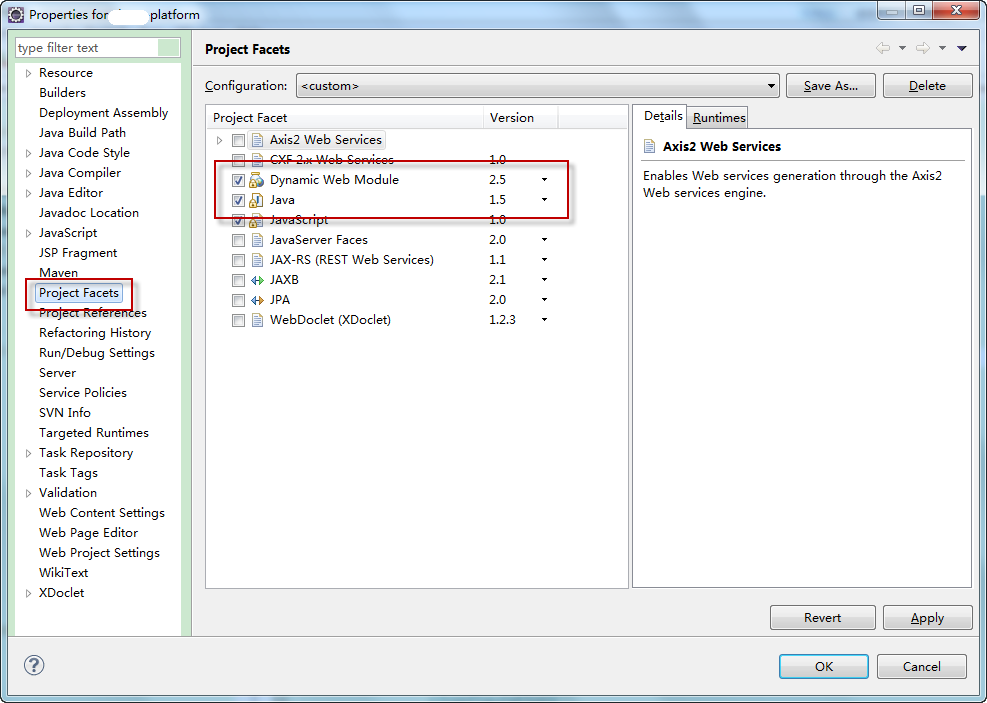
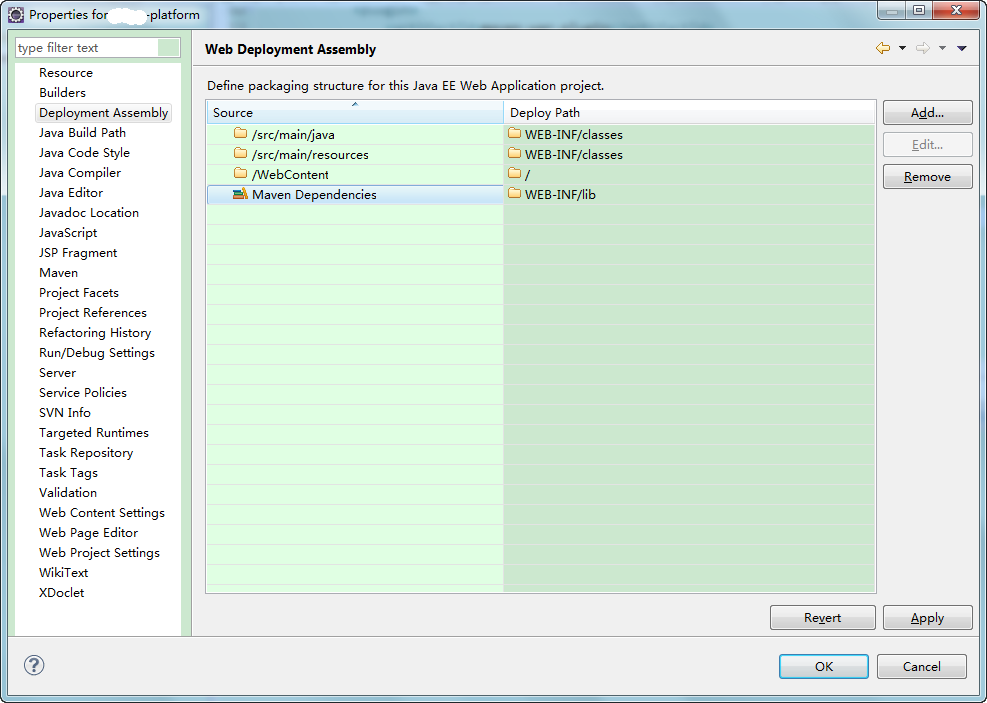
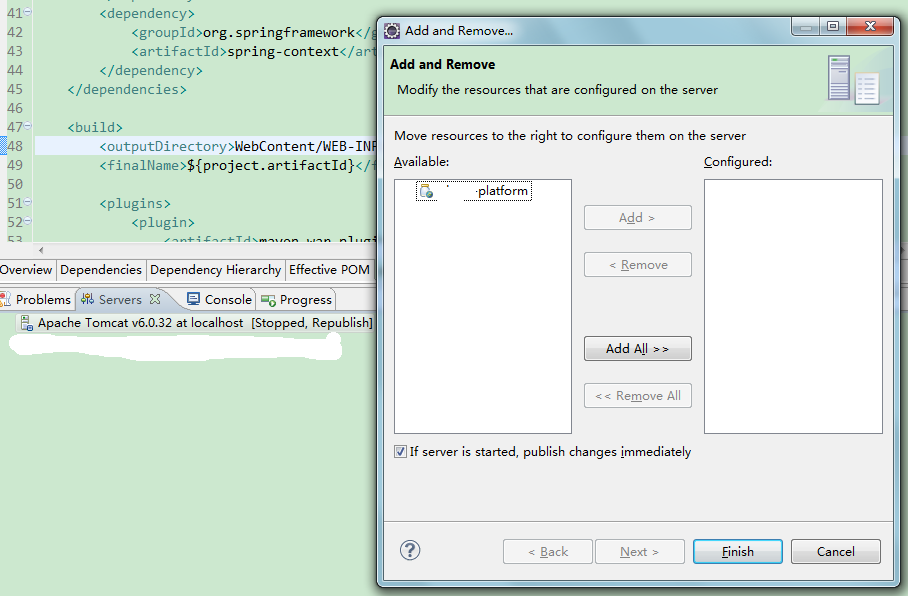
**Xml代码  [收藏代码](javascript:void())**

1. **<server>**
2. **<id>**tomcat-6.0.32**</id>**
3. **<username>**tomcat**</username>**
4. **<password>**tomcat**</password>**
5. **</server>**

2） 在tomcat/conf的tomcat-users.xml加入如下配置：

**Xml代码  [收藏代码](javascript:void())**

1. **<role** rolename="manager-gui"**/>**
3. **<role** rolename="manager-script"**/>**
5. **<user** username="tomcat" password="tomcat" roles="manager-gui, manager-script"**/>**

3） 启动tomcat   
4） 类似方法一那样执行tomcat:redeploy命令   
  
方法三（不推荐）：   
增加eclipse web工程支持，利用传统的eclipse集成tomcat测试的方法进行测试。此方法其实跟maven和tomcat-maven-plugin无关，发布到tomcat下测试的war包不一定跟maven构建的包一致，因此不推荐。   
1） 在工程属性里的Project Facets里勾上Dynamic Web Module，如下图：   
  
  
  
2） 在Deployment Assembly增加将maven依赖发布到“WEB-INF/lib”下，并将“/src/test/java”和“/src/test/resources”两项删除（因为maven打包时是不会打包这两部分内容的）   
  
  
  
3） 这样便可以利用传统的eclipse集成tomcat测试的方法进行测试了，如下图：   
  
  
  
补充：   
如果是多模块工程，只需在父工程下运行tomcat-maven-plugin的goal即可。如：D:\<父工程目录>>tomcat:run。tomcat-maven-plugin会自动检测和替换相关的依赖。

**Text代码  [收藏代码](javascript:void())**

1. NOTE If you have a multi module Maven projects and use Maven3, you don't need to install all modules before use the run goal, just use tomcat6/7:run from the root module and the plugin will auto detect build output directory from various modules and replace dependencies with those directories in the webapp classloader.

更多关于tomcat-maven-plugin的使用，请参考：   
<http://mojo.codehaus.org/tomcat-maven-plugin/index.html>   
<http://tomcat.apache.org/maven-plugin-2/index.html>

# [I am myself!](http://www.cnblogs.com/MyFavorite/)

* [博客园](http://www.cnblogs.com/)
* [首页](http://www.cnblogs.com/MyFavorite/)
* [新闻](http://news.cnblogs.com/)
* [新随笔](http://www.cnblogs.com/MyFavorite/admin/EditPosts.aspx?opt=1)
* [联系](http://space.cnblogs.com/msg/send/MyFavorite)
* [管理](http://www.cnblogs.com/MyFavorite/admin/EditPosts.aspx)
* [订阅](http://www.cnblogs.com/MyFavorite/rss)

随笔- 55  文章- 0  评论- 30

# [5个必须掌握的maven命令](http://www.cnblogs.com/MyFavorite/archive/2012/03/18/2404330.html)

1. mvn help:describe 你是否因为记不清某个插件有哪些goal而痛苦过,你是否因为想不起某个goal有哪些参数而苦恼,那就试试这个命令吧,它会告诉你一切的. 参数: 1. -Dplugin=pluginName 2. -Dgoal(或-Dmojo)=goalName:与-Dplugin一起使用,它会列出某个插件的goal信息,如果嫌不够详细,同样可以加-Ddetail.(注:一个插件goal也被认为是一个 “Mojo”) 下面大家就运行mvn help:describe -Dplugin=help -Dmojo=describe感受一下吧!

2. mvn archetype:generate 你是怎么创建你的maven项目的?是不是像这样:mvn archetype:create -DarchetypeArtifactId=maven-archetype-quickstart -DgroupId=com.ryanote -Dartifact=common,如果你还再用的话,那你就out了,现代人都用mvn archetype:generate了,它将创建项目这件枯燥的事更加人性化,你再也不需要记那么多的archetypeArtifactId,你只需输入archetype:generate,剩下的就是做”选择题”了.

3. mvn tomcat:run 用了maven后,你再也不需要用eclipse里的tomcat来运行web项目(实际工作中经常会发现用它会出现不同步更新的情况),只需在对应目录(如/ryanote)里运行 mvn tomat:run命令,然后就可在浏览器里运行http://localhost:8080/ryanote查看了.如果你想要更多的定制,可以在pom.xml文件里加下面配置: 01 02 03 04 org.codehaus.mojo 05 tomcat-maven-plugin 06 07 /web 08 9090 09 10 11 12 当然你也可以在命令里加参数来实现特定的功能,下面几个比较常用: 1. 跳过测试:-Dmaven.test.skip(=true) 2. 指定端口:-Dmaven.tomcat.port=9090 3. 忽略测试失败:-Dmaven.test.failure.ignore=true 当然,如果你的其它关联项目有过更新的话,一定要在项目根目录下运行mvn clean install来执行更新,再运行mvn tomcat:run使改动生效.

4. mvnDebug tomcat:run 这条命令主要用来远程测试,它会监听远程测试用的8000端口,在eclipse里打开远程测试后,它就会跑起来了,设断点,调试,一切都是这么简单.上面提到的那几个参数在这里同样适用.

5. mvn dependency:sources 故名思义,有了它,你就不用到处找源码了,运行一下,你项目里所依赖的jar包的源码就都有了

mvn archetype:generate -DarchetypeCatalog=internal

### [maven笔记](http://celestialwolf.iteye.com/blog/720303)

[maven](http://www.iteye.com/blogs/tag/maven)[Eclipse](http://www.iteye.com/blogs/tag/Eclipse)[Web](http://www.iteye.com/blogs/tag/Web)[JDK](http://www.iteye.com/blogs/tag/JDK)[单元测试](http://www.iteye.com/blogs/tag/%E5%8D%95%E5%85%83%E6%B5%8B%E8%AF%95)

    初学maven即感受到其强大的便捷性，大体记录下maven的基础知识以备不时之需。

    常用命令为 ：

            mvn archetype:create ：创建 Maven 项目

            mvn compile ：编译源代码

            mvn test-compile ：编译测试代码

            mvn test ： 运行应用程序中的单元测试

            mvn site ： 生成项目相关信息的网站

            mvn clean ：清除目标目录中的生成结果

            mvn package ： 依据项目生成 jar 文件

            mvn install ：在本地 Repository 中安装 jar

            mvn deploy：将jar包发布到远程仓库

            mvn eclipse:eclipse ：生成 Eclipse 项目文件

    生成项目

           建一个 JAVA 项目 ： mvn archetype:create -DgroupId=com.test -DartifactId=App

           建一个 web 项目 ： mvn archetype:create -DgroupId=com.test -DartifactId=web-app -DarchetypeArtifactId=maven-archetype-webapp

     解释：

           archetype 是一个内建插件，它的create任务将建立项目骨架

           archetypeArtifactId 项目骨架的类型

           groupId 项目的java包结构，可修改 artifactId 项目的名称，生成的项目目录也是这个名字，可修改

     可用项目骨架有：

           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

      web项目目录结构的布局：

           详见<http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

      Eclipse导入已存在的maven项目 ：

            在导入之前需要在项目根目录下面执行如下命令：

                  普通 Eclipse 项目执行 ： mvn eclipse:eclipse Eclipse

                  web 项目执行 ： mvn eclipse:eclipse –Dwtpversion=1.0

            然后通过Eclipse的maven插件中选择导入已存在的maven项目即可将其导入。

       学习过程中的问题：

             安装好m2eclipse插件后一般会报错误：Eclipse is running in a JRE, but a JDK is required Some Maven plugins may not work when importing projects or updating source folders.

            解决方式为：

                1.创建一个Eclipse的启动的快捷方式，在其启动路径后添加-vm 你的JDK安装路径\bin\javaw.exe

                2.在eclipse的安装目录下打开eclipse.ini中的-vmargs上面添加-vm 你的JDK安装路径\bin\javaw.exe

            eclipse导入创建的web项目问题

                在用eclipse导入web项目时 如果eclipse安装了wtp插件在会自动将其转化为eclipse认得到的web项目，在项目上面点击右键run as 的run on server可以用，但是当eclipse没有安装wtp插件时run on server则不用，此时的解决方式为用命令行切换到项目的根目录然后执行 mvn eclipse:eclipse –Dwtpversion=1.0即可解决。

Maven配置2－使用Maven创建WEB项目支持Tomcat目录部署

博客分类： Maven2

mavenWebTomcat项目管理配置管理

一、创建Maven　Web项目操作步骤：

　１、创建一个新的Dynamic Web Project。

　２、设置JAVA源代码目录为scr/main/java，保持和MAVEN的源代码目录一至。

　３、设置Web内容的目录为src/main/webapp。也是为了保持和Maven目录一至。

　４、创建完成后，右击项目选择Maven->Enable Dependency Management,使用Maven管理JAR依赖。

　５、设置POM，将打包类型设为WAR.

　６、将Maven Dependencies添加到Java EE Module Dependencies中。

　７、设置JAVA代码的编译路径。编译路径设置到webapp/WEB-INF/classes就是为了支持目录部署。

８、设置编译器版本。

二、新建一个Tomcat运行Server。在新建的过程中不要选择部署的项目，等建完后，双击这个Server在配置中再添加就能进行目录部署。但是到目前lib下面的jar包还不会到webapp/WEB-INF/lib下。默认是到/target/ECOP\_M2-1.0.1-SNAPSHOT/WEB-INF/lib目录下。

三、让Maven打包时，将jar拷贝一份到webapp/WEB-INF/lib下。

　１、双击打开pom.xml文件，切换到pom.xml卡片下。增加maven依赖插件，设置在打包时将jar拷贝一份到web的lib下去，这样tomcat启动时就能找到项目的jar包了。

　　代码如下：

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-dependency-plugin</artifactId>

<version>2.1</version>

<executions>

<execution>

<id>copy-dependencies</id>

<phase>package</phase>

<goals>

<goal>copy-dependencies</goal>

</goals>

<configuration>

<outputDirectory>${basedir}/src/main/webapp/WEB-INF/lib</outputDirectory>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

maven标准目录架构

博客分类： maven

在使用jenkins时编译同学们的maven项目时，读不到配置文件。

晕菜了。

原因：

配置文件都放在config里面，所以。。。

约定优于配置是maven的一个最重要设计理念。

标准目录结构：

src

-main

--bin 脚本库

--java java源代码文件

--resources 资源库，会自动复制到classes目录里

--filters 资源过滤文件

--assembly 组件的描述配置（如何打包）

--config 配置文件

--webapp web应用的目录。WEB-INF、css、js等

-test

--java 单元测试java源代码文件

--resources 测试需要用的资源库

--filters 测试资源过滤库

-site Site（一些文档）

target

LICENSE.txt Project’s license

README.txt Project’s readme

工程根目录下就只有src和target两个目录

target是有存放项目构建后的文件和目录，jar包、war包、编译的class文件等。

target里的所有内容都是maven构建的时候生成的

maven eclipse 集成

博客分类： 开发工具

在Eclipse中使用Maven插件

简介

本文介绍如何在Eclipse中通过maven插件编写java项目和web项目。

安装Maven

下载Maven最新版本，见：maven.apache.org/download.html

当前版本2.0.7。

解压缩下载的文件, 将其中的bin目录设置到windows Path环境变量中.

测试安装是否成功：在命令行中输入

mvn -version

安装Maven插件

安装Eclipse的Maven插件M2Eclipse。

本文使用：

JDK：SUN JDK1.6.0\_03

Eclipse：3.3.1.1

M2Eclipse插件网址：m2eclipse.codehaus.org

通过Eclipse在线更新网址：m2eclipse.codehaus.org/update/

具体安装步骤见插件网址。

为Maven设置Classpath变量，设置java>build path>classpath variables

通过new...按钮增加一个变量条目：

name：M2\_REPO

path：C:/Documents and Settings/zhangsan/.m2/repository，zhangsan是你的xp中的用户名

创建Maven的Java项目并导入到Eclipse中

在命令行中，进入eclipse的workspace目录下，输入：

mvn archetype:create -DgroupId=org.marshal -DartifactId=helloworld

其中：

org.marshal是组织名称，另外maven自动生成了源代码的包org/marshal

helloworld是项目名称

运行命令后，workspace目录下生成helloworld目录，并生成：

项目构建文件: pom.xml

代码框架：src\main\java\org\marshal\helloworld\App.java

测试代码：src\test\java\org\marshal\helloworld\AppTest.java

这时可以通过文本编辑器修改App.java源代码，并通过如下命令编译打包：

maven pacage

编写AppTest.java后，可通过下面命令运行JUnit测试：

mvn test

通过如下命令运行main方法：

java -cp target\helloworld-1.0-SNAPSHOT.jar org.marshal.App

不过，还是最好通过IDE工具编写和调试代码，这需要将maven项目导入为Eclipse项目。

在helloworld目录下，运行下面命令，使项目支持eclipse：

mvn eclipse:eclipse

然后，打开Eclipse，通过File>import...，general>existing projects into workspace，将helloworld导入。

导入后，如下图：

创建maven Web项目并导入到Eclipse中

如果需要创建maven的web项目，在eclipse的workspace目录下，运行：

mvn archetype:create -DgroupId=org.marshal -DartifactId=myweb -DarchetypeArtifactId=maven-archetype-webapp

将创建名为myweb的web项目。

为该项目增加eclipse WTP的支持，到myweb目录下，运行：

mvn -Dwtpversion=1.0 eclipse:eclipse

将该项目导入Eclipse，和上面java项目类似。

使用M2Eclipse插件

安装M2Eclipse插件成功后，在Eclipse点击刚才创建的web项目或者java项目。

鼠标右键，Maven>Enable dependency management，使m2eclipse插件对该项目生效。

再次鼠标右键，选择Maven>add dependency，就可以通过网络增加需要的类库了。

见下图：

选中后，maven将自动下载该类库版本到本地，并且自动加入到eclipse类库中。

**maven生成web project,后转换为eclipse**

2012-06-13 16:55:00|  分类： [java](http://blog.163.com/liuweiyoung@126/blog/#m=0&t=1&c=fks_084071086094084070087083087095085081082066087087086071081)|字号 订阅

网上查找资料 ，自己 看看

Maven3.0.3 入门

创建工程存放目录： E:\maven3 ，开始->cmd, cd E:\maven3

1、创建一个工程的命令：

mvn archetype:create -DgroupId=cn.cdm -DartifactId=app  
该工程的groupId是cn.cdm,那么该工程的源代码文件将放在java包cn.cdm中。  
artifactId是app，那么该工程的根目录名称将为app，即该项目为app ；

   进入app，即(cd E:\maven3\app) ,并执行mvn compile，然后会在该目录下发现新生成的目录target\classes，即编译后的class文件(包括它的包目录)就放在了这里。  
   执行命令 mvn clean，会清除本工程的所有输出，在本例子中，就是删除了target目录。  
   执行命令 mvn install，会把本例子输出的jar发布到maven库里。如果没有调整设置的话，localRepository在~/.m2/repository，比如C:\Documents and Settings\Administrator\.m2\repository，我的机器是调整过了，所以在E:\Maven3\app\m3\repository。  
     
     
2、创建一个web工程的命令：  
mvn archetype:create -DgroupId=cn.cdm -DartifactId= webApp -DarchetypeArtifactId=maven-archetype-webapp  
web工程与普通工程的差别是多一个属性archetypeArtifacttId，该属性的值为maven-archetype-webapp。即告诉Maven，将要创建的工程是一个Web应用工程。创建app工程时没有使用该属性值，是由于archetype默认创建的是应用程序工程。

需要到创建的webApp目录中的src\下创建

src/main/java

src/test/java

src/test/resources

在java build path 中修改四个文件class文件输出路径   
            src/main/java src/java/resources 选择target/classes   
            src/test/java src/test/resources 选择target/test-classes   
        修改项目jdk以及compiler   
        把项目转换成dynamic web--右键项目，选择project facets--配置project facets   
        设置部署程序集--右键项目，选择deployment assembly，并进行相关配置

3、将maven3.0.3项目转换为eclipse项目

  进入maven的项目目录，用 mvn eclipse:eclipse生成eclipse项目；  
  执行完后，生成eclipse的工程文件；  
  调整eclipse的classpath变量(用于指向maven的本地库的目录)，选择eclipse Window-->Preferences -->Java-->Build Path-->ClassPath Variables,增加一个M2\_REPO变量，选择maven的本地库的目录(E:\Maven3\app\m3\repository).  
  工程导入：File-->Import-->Existing Projects into Workspace-->Select root directory,完成导入  
  或者 在eclipse的 Project Expor中 右键 Import-->Maven-->Existing Maven Projects-->Select root Directory,完成导入（我都是用这个） ；  
    
    
    
    
maven3实战之maven使用入门(打包和运行)

----------

将项目进行编译，测试之后，下一个重要步骤就是打包。pom.xml中如果没有指定打包类型，默认打包类型为：jar。我们可以简单地执行命令打包： mvn clean pachage。

类似地，maven会在打包之前执行编译，测试等操作。如以打jar包为例，jar:jar任务负责打包，实际上就是jar插件的jar目标将项目主代码打包成一个名为xxxxxx-1.0-SNAPSHOT.jar的文件。该文件也位于target/输出目录中，它是根据artifact-version.jar的规则进行命名的，如有需要，还可以使用finalName来自定义该文件的名称，这里暂且不展开，后面会详细解释。

在打包之后，又执行了安装任务install:install。从输出可以看到任务将项目输出的jar安装到了maven本地仓库中，可以打开相应的文件夹看到项目的pom和jar。之前我们说只有构件被下载到本地仓库后，才能由所有Maven项目使用，这里同样的道理，只有将自己编写的项目安装到本地仓库之后，其他maven项目才能使用它。

我们已经体验了maven最主要的命令：mvn clean compile，mvn clean test，mvn clean package，mvn clean install。执行test之前是会先执行compile的，执行package之前是会先执行test的，而类似地，install之前会执行package。可以在任何一个maven项目中执行这些命令。

默认打包是生成jar的，如果我的项目还有main方法的，我想打包成可执行的jar包，为了生成可执行的jar文件，需要借助maven-shade-plugin,配置该插件如下：

Xml代码    
1.<plugin>    
2.    <groupId>org.apache.maven.plugins</groupId>    
3.    <artifactId>maven-shade-plugin</artifactId>    
4.    <version>1.2.1</version>    
5.    <executions>    
6.        <execution>    
7.            <phase>package</phase>    
8.            <goals>    
9.                <goal>shade</goal>    
10.            </goals>    
11.            <configuration>    
12.                <transformers>    
13.                    <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">   
14.                        <mainClass>com.juvenxu.mvnbook.helloworld.HelloWorld</mainClass>   
15.                    </transformer>    
16.                </transformers>    
17.            </configuration>    
18.        </execution>    
19.    </executions>    
20.</plugin>    
<plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-shade-plugin</artifactId>  
    <version>1.2.1</version>  
    <executions>  
        <execution>  
            <phase>package</phase>  
            <goals>  
                <goal>shade</goal>  
            </goals>  
            <configuration>  
                <transformers>  
                    <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">  
                        <mainClass>com.juvenxu.mvnbook.helloworld.HelloWorld</mainClass>  
                    </transformer>  
                </transformers>  
            </configuration>  
        </execution>  
    </executions>  
</plugin>

默认打包生成jar是不能够直接运行的，因为带有main方法的类信息不会添加到manifest中(打开jar文件中的META-INF/MANIFEST.MF文件，将无法看到Main-Class一行)。

plugin元素在POM中的相对位置应该在<project><build><plugins>下面。我们配置了mainClass为com.juvenxu.mvnbook.helloworld.HelloWorld，项目在打包时会将该信息放到MANIFEST中。现在执行mvn clean install，待构建完成之后打开target/目录，可以看到hello-world-1.0-SNAPSHOT.jar和original-hello-world-1.0-SNAPSHOT.jar，前者是带有Main-Class信息的可执行jar，后者是原始的jar，打开hello-world-1.0-SNAPSHOT.jar的META-INF/MANIFEST.MF，可以看到它包含这样一行信息：

Main-Class:com.juvenxu.mvnbook.helloworld.HelloWorld

现在，在项目根目录中执行该jar文件：

D:\code\hello-world > java-jar target\hello-world-1.0-SNAPSHOT.jar，可以得到正确的输出了。

mvn compiler:compile 将java文件编译至target/class目录

mvn resources:resources 拷贝properties文件至target/classes相应的目录下

mvn clean 清除target目录

mvn clean compile 完成mvn clean/mvn resources:resources/mvn compiler:compile三个命令

mvn package 打包class和properties并输出到target文件夹

mvn install:install 打包并安装到本地仓库

mvn clean:clean 表示的意思是clean插件的clean目标其他同理

mvn archetype:generate 创建项目架构

mvn dependency:list/mvn dependency:tree 查看项目中所有以来的具体信息

scope为test表示该包只在测试范围内有效，测试代码中import是没问题，如果是主代码中应用则不行，默认scope的值为compile，表示该以来对测试代码和主代码都有效

我想使用到jdbc的oracle驱动，但是这个在apache项目中并不开源，所以只能先将本地的jar安装到本地仓库

mvn install:install-file -Dfile=E:\workspace\seasky\WebRoot\WEB-INF\lib\ojdbc-14.jar -DgroupId=com.oracle -DartifactId=ojdbc14 -Dversion=10.2.0.4.0 -Dpackaging=jar

-Dfile 当前jar目录

-DgroupId/-DartifactId 安装到本地仓库的groupId和artifactId

安装完以后可以在pom.xml中如下配置

<dependency>  
     <groupId>com.oracle</groupId>  
     <artifactId>ojdbc14</artifactId>  
     <version>10.2.0.4.0</version>

</dependency>

maven常用命令   
常用命令为 ：   
    mvn archetype:create ：创建 Maven 项目   
    mvn compile ：编译源代码   
    mvn test-compile ：编译测试代码   
    mvn test ： 运行应用程序中的单元测试   
    mvn site ： 生成项目相关信息的网站   
    mvn clean ：清除目标目录中的生成结果   
    mvn package ： 依据项目生成 jar 文件   
    mvn install ：在本地 Repository 中安装 jar（mvn install -D maven.test.skip=true 跳过TestCase检验，否则在install时会运行TestCase测试）  
    mvn deploy：将jar包发布到远程仓库   
    mvn eclipse:eclipse ：生成 Eclipse 项目文件

生成项目   
   建一个 JAVA 项目 ： mvn archetype:create -DgroupId=com.test -DartifactId=App   
   建一个 web 项目 ： mvn archetype:create -DgroupId=com.test -DartifactId=web-app -DarchetypeArtifactId=maven-archetype-webapp

将已有的项目转变为maven项目

创建如下新的Maven目录结构：

·       将所有已经存在的java源文件移到“\src\main\java”

·       将“web.xml”文件移到“\src\main\webapp\WEB-INF”

·       创建一个新的文件，命名为“pom.xml”，放到项目的根目录下

项目目录结构\--projectname  
   \--src  
      \--main  
         \--java (all java source here)  
      \--test  
         \--- (all unit test here)  
      \--webapp  
         \--WEB-INF  
            \--web.xml

在“pom.xml”文件中填写已经存在的项目详细信息，添加远程资源库、war插件和编译插件。

填写项目详细信息<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>com.test</groupId>  
  <artifactId>webdemo</artifactId>  
  <packaging>jar</packaging>  
  <version>1.0</version>  
  <name>serlvetdemo</name>  
  <url>http://maven.apache.org</url>

  <repositories>  
    <repository>  
      <id>java.net</id>  
      <url>http://download.java.net/maven/2</url>  
    </repository>  
  </repositories>

  <build>  
    <plugins>  
      <plugin>  
        <groupId>org.apache.maven.plugins</groupId>  
        <artifactId>maven-war-plugin</artifactId>  
        <configuration>  
          <webResources>  
            <resource>  
              <directory>${basedir}/src/main/java</directory>  
              <targetPath>WEB-INF/classes</targetPath>  
              <includes>  
                <include>\*\*/\*.properties</include>  
                <include>\*\*/\*.xml</include>  
                <include>\*\*/\*.css</include>  
                <include>\*\*/\*.html</include>  
              </includes>  
            </resource>  
          </webResources>  
        </configuration>  
      </plugin>

      <plugin>  
        <artifactId>maven-compiler-plugin</artifactId>  
        <configuration>  
            <source>1.6</source>  
            <target>1.6</target>  
        </configuration>  
      </plugin>  
    </plugins>  
  </build>  
</project>

编译项目 C:\Users\lxh\workspace>\webdemo>mvn compile

接着，将基于Maven的项目转换为支持Eclipse IDE。

mvn eclipse:eclipse –Dwtpversion=2.0

然后，我们可以将该项目导入到Eclipse IDE了。

使用“mvn war:war”生成项目的WAR文件，新生成的WAR文件将位于“/rootproject/target”目录下，名为“servletdemo-1.0.war”，Maven将会自动地把所有的依赖库、类和部署结构打包在一起。

4.构建项目

要构建基于Maven的项目，打开控制台，使用cd命令进入到项目目录下（pom.xml文件所在目录），使用如下的命令： mvn package。

这将执行Maven的“package”(打包)阶段。

Maven构建生命周期：Maven的运行是分阶段逐步进行的，因此，当“package”阶段被执行时，它之前的所有阶段-“validate”、“compile”、“test”，包括当前的阶段“package”，都将被顺序地执行。

Pom.xml文件中

<packaging>jar</packaging>

jar 生成jar文件 ，war 生成war包

5.maven 的war部署到Tomcat服务器中

使用管理员访问权限为Tomcat添加一个用户。要添加Tomcat用户，编辑这个文件Tomcat安装目录/conf/tomcat-users.xml”。

文件：tomcat-users.xml<?xml version='1.0' encoding='utf-8'?>  
<tomcat-users>  
    <role rolename="manager"/>  
    <role rolename="admin"/>  
    <user username="admin" password="password" roles="admin,manager"/>  
</tomcat-users>

在Maven站点中，我们需要在“MAVEN安装目录/conf/settings.xml”中添加相同的用户验证信息。

<server>  
    <id>TomcatServer</id>  
    <username>admin</username>  
    <password>password</password>  
</server>

pom.xml文件 添加tomcat 的plugin配置文件

<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>com.test</groupId>  
  <artifactId>webtest</artifactId>  
  <packaging>war</packaging>  
  <version>1.0</version>  
  <name> xuejavaweb-core Maven Webapp</name>  
  <url>http://maven.apache.org</url>  
  <build>  
    <plugins>

      <!-- Maven Tomcat Plugin -->  
      <plugin>  
          <groupId>org.codehaus.mojo</groupId>  
          <artifactId>tomcat-maven-plugin</artifactId>  
          <configuration>  
              <url>http://127.0.0.1:8080/manager</url>  
              <server>TomcatServer</server>  
              <path>/webtest</path>  
          </configuration>  
      </plugin>

      <!-- Maven compiler plugin -->  
      <plugin>  
        <artifactId>maven-compiler-plugin</artifactId>  
        <configuration>  
            <source>1.6</source>  
            <target>1.6</target>  
        </configuration>  
      </plugin>

    </plugins>

  </build>  
</project>

命令窗口中切换到 C:\Users\lxh\workspace>webtest

使用“mvn tomcat:deploy”命令来将项目打包成一个WAR文件，并将其部署到Tomcat服务器

**Web开发教程4－掌握Maven**   [编程](http://bluedash.net/categories/%E7%BC%96%E7%A8%8B/spaces) / [阿男](http://bluedash.net/users/%E9%98%BF%E7%94%B7) / 浏览(391) / 评论(0) / 喜欢(7)

有关本系列教程的目录，源代码以及相关说明，请参考 [附录](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B%EF%BC%8D%E9%99%84%E5%BD%95)

本系列文章主要介绍Spring框架的使用方法，特别针对Spring框架2.x版本中的功能特征进行讲解。本篇文章讲解Java项目构建工具－Maven。

您是否在开发中使用过一些第三方的程序库？比如Struts，比如Spring，或者是Hibernate？相信您一定不会只用JDK提供的那些基础库函数来编写您的项目，从轮子造起太可怕了。在编写代码的时候，肯定会使用到第三方的项目，而在管理这些项目时，您是使用什么样的方式？可能最常见的办法就是从相关的网站上下载所需要的Jar文件，然后把jar添加到项目的路径中进行编译。这样的方法对于小项目还可以，但是对于依赖几十个库的大项目就是恶梦了：不仅需要从几十个不同的网站上下载所需要的程序库，而这些程序库还可能又依赖别的库，这时就得一层一层地手工解决这些依赖并进行下载。最要命的是这些库之前还可能有版本冲突，程序运行起来以后，抛出莫名其妙的错误，还找不到错在哪里。如果这里提到的问题正好是每天都让您非常头痛的问题，那么通过将学习强大的项目管理工具：Maven，您将彻底摆脱这些烦恼。

## 什么是Maven

Maven是由Apache基金会资助的开源项目，它在项目装配流程中起到了非常重要的作用。因此有必要学会使用Maven自动化装配流程，并应用其对项目进行打包。Maven这个词原有的含义是"知识的累积"，而Maven项目的官方解释是"为Java项目提供的编译及项目管理工具"。对于Maven的用途，官方解释如下：

* 使编译过程更简单
* 提供统一的项目构建系统
* 提供项目的质量控制信息
* 提供最棒的开发流程
* 允许透明地添加新功能

## Maven的安装

要学习Maven，首先当然要学会如何在您的计算机上安装这个项目。实际上安装它非常简单，下载－解压后几乎就可以使用了。安装Maven的步骤概括起来就是：下载、配置、运行。此外，Maven在不同的操作系统平台上，最简便的安装方式有所不同：在MacOS Leopard操作系统下，Maven是默认已安装的。在Linux及Windows下，需要手工安装。对于安装了yum的Linux，使用 yum install maven 即可安装。下面重点介绍Windows下的安装方法[1](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B4%EF%BC%8D%E6%8E%8C%E6%8F%A1Maven#fn1)。

1 实际上这个手工方法适用于所有操作系统。

在安装前，需要确保系统中已经安装了JDK环境。为了检查我们的机器上是否已经正确地安装了JDK，请您在命令行中输入：

|  |  |
| --- | --- |
| 1 | java -version |

应该有类似这样的返回：

|  |  |
| --- | --- |
| 1 | java version "1.5.0\_13" |
| 2 | Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0\_13-b05-237) | |

|  |  |
| --- | --- |
| 3 | Java HotSpot(TM) Client VM (build 1.5.0\_13-119, mixed mode, sharing) |

Maven对JDK的版本有一定的要求，它需要JDK的版本在1.4以上。如果您的机器上没有正确地安装JDK，请从网站上下载安装完JDK环境后，再进行Maven的安装。如果JDK没什么问题，下面就可以进行Maven的安装了。安装Maven的第一步是从Maven的官方网站上下载程序包。Maven网站的网址为： <http://maven.apache.org/> 。您在这个网站个可以很容易地找到Download链接，下接您所需的版本即可。一般情况下，下载最新版就行了，Maven的下载文件是一个压缩包。下载完成后，您需要做的就是将这个压缩包解压到一个方便好记的目录当中。这个目录就是Maven的程序目录了，Maven不需要额外的安装过程。接下来需要做的就是把Maven的可执行程序目录加到系统环境中，假设Maven的安装包解压到了 c:\maven 中，那就在命令行下设置Windows系统路径如下：

|  |  |
| --- | --- |
| 1 | set PATH="c:\maven\bin";%PATH% |

如果使用的是Linux及MacOS操作系统，并使用了这样的手工安装方式，那么就需要设定~/.profiles中 的PATH路径。配置完路径后，我们要检查一下配置是否正确。请您在命令行下输入Maven的命令，检查一下Maven是否已经可以使用：

|  |  |
| --- | --- |
| 1 | mvn --version |

如果系统正确地返回了Maven的版本号，类似如下：

|  |  |
| --- | --- |
| 1 | Maven version: 2.0.6 |

就说明Maven已经正确地安装到系统当中了。安装完成后，我们接下来学习如何使用它。

## Maven的基本使用方法

Maven在整个的工程开发周期中都有自己应用的范围，在这一节从创建Java项目开始，为大家介绍Maven的使用方法。

**生成一个Java工程**

用Maven生成一个最一般的Java工程，需要做的是在终端下输入一条Maven命令：

|  |  |
| --- | --- |
| 1 | mvn archetype:create -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=javaProj -DartifactId=javaProj |

这个命令用于生成一般Java项目。我们把这个命令分解并进行说明：

* mvn是Maven的主命令，几乎所有的任务都由mvn这个命令完成。
* 其中 archetype:create 表示我们要使用Maven执行项目的生成的任务。  
  -DarchetypeGroupId=org.apache.maven.archetypes 用于引入此项目在创建时所使用的项目模型组，它用于指定要生成的Java项目类型。实际上Maven可以生成各种各样的Java项目，如Web应用或带Hibernate的Web应用等，随指定的项目类型不同，这些应用需要的函数库各自不同，程序目录架构也不尽相同。我们稍后会看到用maven创建web项目的例子，在这里，我们使用 -DarchetypeGroupId=org.apache.maven.archetypes 生成默认的普通Java工程。
* 接下来我们看 -DgroupId=javaProj及-DartifactId=javaProj ，这两个参数给我们要创建的项目起个名字，并且成为唯一标识我们这个项目的信息。

命令敲入后，Maven为了生成这个项目，会根据您的定制要求，从网上主动下载程序所依赖的程序库，并下载编译这个项目所需的这些配置文件及工具。一些的工作由Maven完成，您只需要等待。由于Maven的使用大量依赖于网络(它其实帮我们自动解决了下载程序库的问题)，下载的过程取决于您的网络传输速度，在网速很慢的情况下，您需要耐心的等待。此时系统会产生很多的日志输出，如果您最后看到SUCCESSFUL的字样，会有类似下面这样的日志输出：

|  |  |  |
| --- | --- | --- |
| 1 | [INFO] ----------------------------------------------------- | |
| 2 | [INFO] BUILD SUCCESSFUL |

|  |  |  |
| --- | --- | --- |
| 3 | [INFO] ----------------------------------------------------- | |
| 4 | [INFO] Total time: 57 seconds |

|  |  |  |
| --- | --- | --- |
| 5 | [INFO] Finished at: Tue Aug 19 14:59:28 CST 2008 | |
| 6 | [INFO] Final Memory: 8M/14M |

|  |  |
| --- | --- |
| 7 | [INFO] ----------------------------------------------------- |

就说明命令成功执行了。如果最后系统显示FAILED，那么就说明在执行过程中产生了错误，这时就需要您仔细查看输出的日志，看看具体的错误原因，并在网上查找相关信息进行解决。如果您按照本书的指导去做，并且机器的网络连接没有什么问题，那么这一步应该会正确通过。命令执行完成后，Maven应该在您的执行命令所在目录中生成了一个名为javaProj的项目，其目录结构如下所示：

|  |  |
| --- | --- |
| 01 | javaProj |
| 02 | |-- pom.xml | |

|  |  |
| --- | --- |
| 03 | `-- src |
| 04 | |-- main | |

|  |  |
| --- | --- |
| 05 | |   `-- java |
| 06 | |       `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 07 | |           `-- App.java | |
| 08 | `-- test |

|  |  |
| --- | --- |
| 09 | `-- java |
| 10 | `-- javaProj | |

|  |  |
| --- | --- |
| 11 | `-- AppTest.java |

对这个项目，我们来对Maven为我们生成的文件做一个简单说明：

* pom.xml 是Maven的配置文件，可以说整个项目的配置都围绕这个文件展开。如果您用过Ant，那么可以将这个文件类比为 build.xml ，但 pom.xml 不仅是控制编译过程，它还控制很多别的东西。我们稍后会看到对这个文件的详细说明。
* src目录是存放源代码的地方。请您注意：Maven对项目的目录结构有比较严格的要求，如果您希望使用Maven编译打包您的项目，就需要按照Maven的目录结构存放相关的文件，不能轻易变更Maven的工程目录结构。比如src是存放代码的地方，您不能将这个目录改名，否则Maven在执行编译过程时，将找不到您的源代码。
* src/main/java 是存放项目Java源代码的地方。在这个目录里，Maven帮您生成了一个文件App.java。这是一个自动生成的主程序，您可以打开看看这个代码。它没有什么实际意义，主要目的是告诉您应该在这个目录里放置所有的Java源代码。
* src/test/java 是存放单元测试文件的地方，Maven帮您生成了一个测试文件AppTest.java 。是一个永远返回成功的JUnit单元测试。告诉用户这里面放JUnit单元测试用例，Maven在执行测试任务时，会执行这个目录里的JUnit单元测试任务。

**使用Maven编译项目**

在上节，我们使用Maven生成了一个Java项目，项目生成后，我们需要做的就是编写实际的项目代码，这一点是需要我们自己完成的，我们不能要求Maven自动帮我们编写代码。但代码编写完成后，Maven可以帮助我们编译代码。现在我们编译这个项目试试。在项目根目录下运行：

|  |  |
| --- | --- |
| 1 | mvn compile |

这时Maven会自动从网上下载编译这个项目所需要的程序库及其它文件，并调用javac编译器对项目进行编译。Maven在运行编译任务时，日志输出类似如下：

|  |  |
| --- | --- |
| 01 | [INFO] Scanning for projects... |
| 02 | [INFO] -------------------------------------------------- | |

|  |  |
| --- | --- |
| 03 | [INFO] Building javaProj |
| 04 | [INFO]    task-segment: [compile] | |

|  |  |  |
| --- | --- | --- |
| 05 | [INFO] -------------------------------------------------- | |
| 06 | [INFO] [resources:resources] |

|  |  |  |
| --- | --- | --- |
| 07 | [INFO] Using default encoding to copy filtered resources. | |
| 08 | [INFO] [compiler:compile] |

|  |  |  |
| --- | --- | --- |
| 09 | [INFO] Compiling 1 source file to /Users/liweinan/javaProj/target/classes | |
| 10 | [INFO] -------------------------------------------------- |

|  |  |
| --- | --- |
| 11 | [INFO] BUILD SUCCESSFUL |
| 12 | [INFO] -------------------------------------------------- | |

|  |  |
| --- | --- |
| 13 | [INFO] Total time: 1 second |
| 14 | [INFO] Finished at: Tue Aug 19 16:14:48 CST 2008 | |

|  |  |
| --- | --- |
| 15 | [INFO] Final Memory: 3M/10M |
| 16 | [INFO] -------------------------------------------------- | |

编译完成后，您会发现自己的工程目录中多了一些文件，如下所示：

|  |  |
| --- | --- |
| 01 | javaProj |
| 02 | |-- pom.xml | |

|  |  |
| --- | --- |
| 03 | |-- src |
| 04 | |   |-- main | |

|  |  |
| --- | --- |
| 05 | |   |   `-- java |
| 06 | |   |       `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 07 | |   |           `-- App.java | |
| 08 | |   `-- test |

|  |  |
| --- | --- |
| 09 | |       `-- java |
| 10 | |           `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 11 | |               `-- AppTest.java | |
| 12 | `-- target |

|  |  |
| --- | --- |
| 13 | `-- classes |
| 14 | `-- javaProj | |

|  |  |
| --- | --- |
| 15 | `-- App.class |

首先，项目中多出了一个target目录，里面包含一个classes目录，这里面放置编译好的类文件。

**使用Maven进行单元测试**

Maven除了创建项目，编译项目和还可以对项目执行单元测试。当然具体的单元测试代码还是需要您自己编写。并且按Maven的命名约定，您还需要把单元测试的代码放置在src/test/java 目录中[2](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B4%EF%BC%8D%E6%8E%8C%E6%8F%A1Maven#fn2)。执行单元测试的命令如下：

|  |  |
| --- | --- |
| 1 | mvn test |

您需要在项目的工程目录根目录下执行上述命令。

2 有关JUnit单元测试方面的知识，不在本书的讲解范围内，对这方面感兴趣的读者可以查看 [JUnit的网站](http://www.junit.org/)

此时Maven会花一些时间从网上下载运行测试所需要的文件，并返回测试结果。Maven的具体的日志输出类似如下：

|  |  |
| --- | --- |
| 01 | [INFO] Scanning for projects... |
| 02 | [INFO] ------------------------------------------------- | |

|  |  |
| --- | --- |
| 03 | [INFO] Building javaProj |
| 04 | [INFO]    task-segment: [test] | |

|  |  |  |
| --- | --- | --- |
| 05 | [INFO] ------------------------------------------------- | |
| 06 | [INFO] [resources:resources] |

|  |  |  |
| --- | --- | --- |
| 07 | [INFO] Using default encoding to copy filtered resources. | |
| 08 | [INFO] [compiler:compile] |

|  |  |  |
| --- | --- | --- |
| 09 | [INFO] Nothing to compile - all classes are up to date | |
| 10 | [INFO] [resources:testResources] |

|  |  |  |
| --- | --- | --- |
| 11 | [INFO] Using default encoding to copy filtered resources. | |
| 12 | [INFO] [compiler:testCompile] |

|  |  |  |
| --- | --- | --- |
| 13 | [INFO] Compiling 1 source file to /Users/liweinan/javaProj/target/test-classes | |
| 14 | [INFO] [surefire:test] |

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | Downloading:<http://repo1.maven.org/maven2/org/apache/maven/surefire/surefire-junit/2.4.2/surefire-junit-2.4.2.pom> | | |
| 16 | | 1K downloaded |

|  |  |  |  |
| --- | --- | --- | --- |
| 17 | Downloading:<http://repo1.maven.org/maven2/org/apache/maven/surefire/surefire-junit/2.4.2/surefire-junit-2.4.2.jar> | | |
| 18 | | 14K downloaded |

|  |  |  |
| --- | --- | --- |
| 19 | [INFO] Surefire report directory: /Users/liweinan/javaProj/target/surefire-reports | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | ------------------------------------------------------- | |
| 22 | T E S T S |

|  |  |  |
| --- | --- | --- |
| 23 | ------------------------------------------------------- | |
| 24 | Running javaProj.AppTest |

|  |  |  |
| --- | --- | --- |
| 25 | Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.059 sec | |
| 26 |  |

|  |  |  |
| --- | --- | --- |
| 27 | Results : | |
| 28 |  |

|  |  |  |
| --- | --- | --- |
| 29 | Tests run: 1, Failures: 0, Errors: 0, Skipped: 0 | |
| 30 |  |

|  |  |  |
| --- | --- | --- |
| 31 | [INFO] ------------------------------------------------- | |
| 32 | [INFO] BUILD SUCCESSFUL |

|  |  |  |
| --- | --- | --- |
| 33 | [INFO] ------------------------------------------------- | |
| 34 | [INFO] Total time: 22 seconds |

|  |  |  |
| --- | --- | --- |
| 35 | [INFO] Finished at: Tue Aug 19 16:20:26 CST 2008 | |
| 36 | [INFO] Final Memory: 4M/10M |

|  |  |
| --- | --- |
| 37 | [INFO] ------------------------------------------------- |

因为样例项目中只有一个由Maven生成的单元测试并且永远返回成功。因此报告中说明运行了一个测试，并且全部通过。此时项目结构如下所示：

|  |  |
| --- | --- |
| 01 | javaProj |
| 02 | |-- pom.xml | |

|  |  |
| --- | --- |
| 03 | |-- src |
| 04 | |   |-- main | |

|  |  |
| --- | --- |
| 05 | |   |   `-- java |
| 06 | |   |       `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 07 | |   |           `-- App.java | |
| 08 | |   `-- test |

|  |  |
| --- | --- |
| 09 | |       `-- java |
| 10 | |           `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 11 | |               `-- AppTest.java | |
| 12 | `-- target |

|  |  |
| --- | --- |
| 13 | |-- classes |
| 14 | |   `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 15 | |       `-- App.class | |
| 16 | |-- surefire-reports |

|  |  |  |
| --- | --- | --- |
| 17 | |   |-- TEST-javaProj.AppTest.xml | |
| 18 | |   `-- javaProj.AppTest.txt |

|  |  |
| --- | --- |
| 19 | `-- test-classes |
| 20 | `-- javaProj | |

|  |  |
| --- | --- |
| 21 | `-- AppTest.class |

**使用Maven进行项目打包**

编译完项目，执行完测试，接下来要做的就是对项目打包了。使用Maven，我们可以很方便地将项目打包成jar。我们所需要做的就是在工程目录根目录下执行下面这条命令：

|  |  |
| --- | --- |
| 1 | mvn package |

日志输出类似如下：

|  |  |
| --- | --- |
| 01 | [INFO] Scanning for projects... |
| 02 | [INFO] ---------------------------------------------- | |

|  |  |
| --- | --- |
| 03 | [INFO] Building javaProj |
| 04 | [INFO]    task-segment: [package] | |

|  |  |  |
| --- | --- | --- |
| 05 | [INFO] ---------------------------------------------- | |
| 06 | [INFO] [resources:resources] |

|  |  |  |
| --- | --- | --- |
| 07 | [INFO] Using default encoding to copy filtered resources. | |
| 08 | [INFO] [compiler:compile] |

|  |  |  |
| --- | --- | --- |
| 09 | [INFO] Nothing to compile - all classes are up to date | |
| 10 | [INFO] [resources:testResources] |

|  |  |  |
| --- | --- | --- |
| 11 | [INFO] Using default encoding to copy filtered resources. | |
| 12 | [INFO] [compiler:testCompile] |

|  |  |  |
| --- | --- | --- |
| 13 | [INFO] Nothing to compile - all classes are up to date | |
| 14 | [INFO] [surefire:test] |

|  |  |  |
| --- | --- | --- |
| 15 | [INFO] Surefire report directory: /Users/liweinan/javaProj/target/surefire-reports | |
| 16 |  |

|  |  |  |
| --- | --- | --- |
| 17 | ------------------------------------------------------- | |
| 18 | T E S T S |

|  |  |  |
| --- | --- | --- |
| 19 | ------------------------------------------------------- | |
| 20 | Running javaProj.AppTest |

|  |  |  |
| --- | --- | --- |
| 21 | Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.035 sec | |
| 22 |  |

|  |  |  |
| --- | --- | --- |
| 23 | Results : | |
| 24 |  |

|  |  |  |
| --- | --- | --- |
| 25 | Tests run: 1, Failures: 0, Errors: 0, Skipped: 0 | |
| 26 |  |

|  |  |
| --- | --- |
| 27 | [INFO] [jar:jar] |
| 28 | [INFO] Building jar: /Users/liweinan/javaProj/target/javaProj-1.0-SNAPSHOT.jar | |

|  |  |  |
| --- | --- | --- |
| 29 | [INFO] ------------------------------------------------- | |
| 30 | [INFO] BUILD SUCCESSFUL |

|  |  |  |
| --- | --- | --- |
| 31 | [INFO] ------------------------------------------------- | |
| 32 | [INFO] Total time: 3 seconds |

|  |  |  |
| --- | --- | --- |
| 33 | [INFO] Finished at: Tue Aug 19 16:26:31 CST 2008 | |
| 34 | [INFO] Final Memory: 6M/13M |

|  |  |
| --- | --- |
| 35 | [INFO] ------------------------------------------------- |

从日志从我们可以看到，Maven在打包前，会先执行compile编译任务，然后执行test测试任务。前两步通过后，才开始对项目进行打包。最后，我们将在target目录里找到打包好的jar： javaProj-1.0-SNAPSHOT.jar 。此时目录中的内容如下所示：

|  |  |
| --- | --- |
| 01 | javaProj |
| 02 | |-- pom.xml | |

|  |  |
| --- | --- |
| 03 | |-- src |
| 04 | |   |-- main | |

|  |  |
| --- | --- |
| 05 | |   |   `-- java |
| 06 | |   |       `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 07 | |   |           `-- App.java | |
| 08 | |   `-- test |

|  |  |
| --- | --- |
| 09 | |       `-- java |
| 10 | |           `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 11 | |               `-- AppTest.java | |
| 12 | `-- target |

|  |  |
| --- | --- |
| 13 | |-- classes |
| 14 | |   `-- javaProj | |

|  |  |
| --- | --- |
| 15 | |       `-- App.class |
| 16 | |-- javaProj-1.0-SNAPSHOT.jar | |

|  |  |
| --- | --- |
| 17 | |-- maven-archiver |
| 18 | |   `-- pom.properties | |

|  |  |
| --- | --- |
| 19 | |-- surefire-reports |
| 20 | |   |-- TEST-javaProj.AppTest.xml | |

|  |  |  |
| --- | --- | --- |
| 21 | |   `-- javaProj.AppTest.txt | |
| 22 | `-- test-classes |

|  |  |
| --- | --- |
| 23 | `-- javaProj |
| 24 | `-- AppTest.class | |

通过这几节的学习，我们学会了使用Maven进行项目的创建、编译、测试、打包。但我们还没有讲到一开始的话题，Maven如何帮我们解决复杂的项目库依赖关系？别着急，从下节开始，我们就要学习Maven进行更深入的项目管理工作。

**理解POM文件**

Maven如此神奇，可以做这么多事情，到底是什么东西在驱动着它完成这些任务？答案就在Maven的pom.xml配置文件中，我们在这一节来仔细研究一下这个文件[3](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B4%EF%BC%8D%E6%8E%8C%E6%8F%A1Maven#fn3) 。下面是javaProj中的pom.xml文件的全部内容：

|  |  |  |
| --- | --- | --- |
| 01 | <project xmlns="<http://maven.apache.org/POM/4.0.0>"xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" | |
| 02 | | xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0><http://maven.apache.org/maven-v4_0_0.xsd>"> |

|  |  |  |
| --- | --- | --- |
| 03 | <modelVersion>4.0.0</modelVersion> | |
| 04 | <groupId>javaProj</groupId> |

|  |  |  |
| --- | --- | --- |
| 05 | <artifactId>javaProj</artifactId> | |
| 06 | <packaging>jar</packaging> |

|  |  |  |
| --- | --- | --- |
| 07 | <version>1.0-SNAPSHOT</version> | |
| 08 | <name>javaProj</name> |

|  |  |  |
| --- | --- | --- |
| 09 | <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 | </project> |

3 在Maven的官方网站中，针对pom.xml有超详细的讲解，详情请参考：<http://maven.apache.org/guides/getting-started/index.html>

这个文件定义了项目的方方面面。其中包含这个项目的识别信息，所依赖的Jave包等。下面我们来逐一看一下这些元素的含义：

* project – pom.xml的根，指定所需的一些xml命名空间。这个元素不必深究。
* modelVersion – 本项目所使用的生成模型版本，咱们生成的是一般java项目，使用的是Maven定义的4.0.0的模型。这个元素不必深究。
* groupId – 组别id，用于唯一标识本项目的创造者或所属组织单位。咱们在创建这个项目时，起的组名是javaProj。
* artifactId – 项目id，用于唯一标识本项目。咱们在创建这个项目时，起的项目名是javaProj。
* packaging – 本项目的打包方式。对于咱们这个最一般的java工程，Maven默认使用jar方式打包。对于web项目，Maven可能使用到的打包方式还有war, ear等。
* version – 本项目的版本名称。
* name – 本项目的项目名。
* url – 本项目的对应网址。如果您的项目做得很大很棒，拥有了自己的网站，那么可以写在这里。这个信息与上面的name元素，主要是给使用您的项目的人看的。它不像groupId、artifactId、version、packageing等元素，会影响到maven的实际执行动作。
* dependencies – 这个元素就是解答我们在本章一开始时提出的问题，它指明这个项目都依赖什么别的项目。Maven会根据依赖关系，自动从网上下载这些项目。我们可以看到，maven是通过groupId，artifactId及version三个元素唯一定义一个所需要的项目的[4](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B4%EF%BC%8D%E6%8E%8C%E6%8F%A1Maven#fn4)。另外有一个scope元素，是指明在项目的何种阶段需要依赖这个项目。这里对Junit的scope指明为test，这就说明仅在执行test时，Maven才会把JUnit的程序包调用进项目中，这样在执行package打包后，JUnit相关的jar就不会被加入到您的项目中。如果不指明scope，Maven默认认为所有的阶段都需要，因此在编译、测试、打包，各种阶段执行前，都会下载这个项目相关的jar。当然Maven会把下载过的文件储存在一个本地目录中，这样如果您在一台机器上有多个项目依赖同一个库，Maven只需下载一次即可。以后再需要这个库时，都会直接从本地的程序库目录中获取。这个本地的程序库目录叫做本地Maven仓库，即 M2\_REPO 。如果您使用的是Windows操作系统，它的位置一般在Document and Setting中您的用户目录根目录下，叫做.m2的目录中。这个我们随后会做详细说明。

4 因此您自己的项目中，groupId、artifactId及version三个元素的用处也就明白了。

通过指定dependency，所有项目依赖的包都无需手工从网上下载，并且那种层层依赖的链条式依赖关系，Maven会通过 pom.xml 中的线索自动解决。比如：我们的项目依赖JUnit，而JUnit又依赖其它的项目，在Maven仓库中，所有的项目都有 pom.xml ，JUnit自己也有一个pom.xml，写明了它的依赖关系，假设它依赖另外几个项目，那么Maven则会把相关的依赖项目全部自动下载，这样一条完整的线被串起来，不用再手工解决恐怖的包依赖性了。

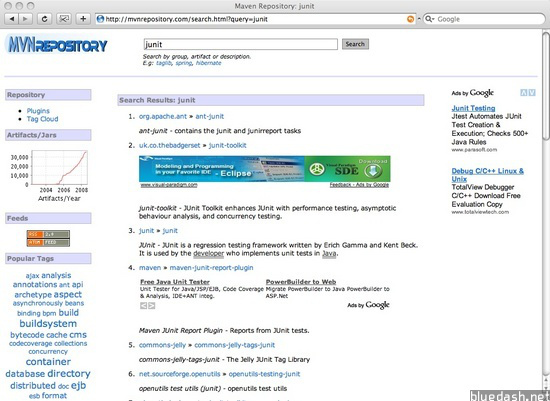
现在有一个问题：您知道自己的项目需要哪些项目，但是去什么地方寻找这些项目的groupId、artifactId及version的标识信息呢？我们来进入下一节。

**在网上查找依赖关系**

查找pom描述信息的最好地点是Maven的官方查询点： <http://mvnrepository.com/> 。在这里，我可以找到几乎全部所需项目的pom描述信息。网站主页如下图所示：

http://bluedash.b0.upaiyun.com/system/photos/869/medium/mvnrepo.jpg!medium

这个网站是一个自助式的查询注册站点。您既可以在这里查询所需项目的信息，也可以在这里把自己的项目信息及项目发布上去，供别人下载使用。我们在这里使用它的搜索功能，查找需要的项目描述信息。比如您需要JUnit，那么就在搜索栏中输入junit，系统将会返回查询结果，如下图所示：



在结果中，可以看到与junit相关的项目有不少，并且还是由不同的组织发布的，它们的groupId五花八门，有 org.apache.ant ，也有 net.sourceforge.openutils ，这些都不是我们需要的。我们需要的是由junit发行的junit项目，即groupId为junit、artifactId也为junit的那个。点进去看一下[5](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B4%EF%BC%8D%E6%8E%8C%E6%8F%A1Maven#fn5)，结果如下图所示：

http://bluedash.b0.upaiyun.com/system/photos/870/medium/mvnrepo-junit-pom.jpg!medium

5 注意应该点击artifactId，而不是groupId，即 junit >> junit 中的后一个。点击artifactId后，才能看到junit项目的描述信息。如果点击groupId，您将看到junit组织发行的所有项目的列表，其中包含junit项目。

可以看到，这里面包含了junit所有版本的列表，每个版本都有其pom标识，并且在下方的POM Dependency一节，很方便地给出了最新版本的pom标识。我们将这个dependency描述信息粘贴到我们的javaProj项目的pom文件的dependencies一节中：

|  |  |
| --- | --- |
| 1 | <dependencies> |
| 2 | <dependency> | |

|  |  |
| --- | --- |
| 3 | <groupId>junit</groupId> |
| 4 | <artifactId>junit</artifactId> | |

|  |  |  |
| --- | --- | --- |
| 5 | <version>4.4</version> | |
| 6 | </dependency> |

|  |  |
| --- | --- |
| 7 | <dependencies> |

这样，我们的项目便完成了对junit的引用。当您在对javaProj执行测试时，Maven便会自动从Maven的代码仓库中下载所需的JUnit库文件。理解了pom文件后，我们接下来再深入地学习一些Maven的配置技巧。

## Maven配置讲解

在这一节中，我们将对项目进行更细致的配置，并学习一些新的使用技巧，如创建Eclipse工程，使用Maven创建web项目等。

**使用Maven生成Eclipse工程文件**

使用记事本写程序并不能证明您就是编程高手。如果有好用的开发环境，还是要使用它来提高您的工作效率。Maven就提供了这种功能，帮助您在项目中生成Eclipse[6](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B4%EF%BC%8D%E6%8E%8C%E6%8F%A1Maven#fn6)。这样，我们就可以在Java世界里非常流行的Eclipse平台中进行项目开发了。为了实现这一目的，我们需要做的就是在程序根目录输入以下命令：

|  |  |
| --- | --- |
| 1 | mvn eclipse:eclipse |

6 除了生成Eclipse工程以外，Maven还支持生成IntelliJ IDEA工程，使用这个IDE环境的读者请查看Maven的文档进行学习。

这个命令的日志输出类似如下：

|  |  |
| --- | --- |
| 01 | [INFO] Scanning for projects... |
| 02 | [INFO] Searching repository for plugin with prefix:'eclipse'. | |

|  |  |  |
| --- | --- | --- |
| 03 | [INFO] ------------------------------------------------------- | |
| 04 | [INFO] Building javaProj |

|  |  |
| --- | --- |
| 05 | [INFO]    task-segment: [eclipse:eclipse] |
| 06 | [INFO] ------------------------------------------------------- | |

|  |  |
| --- | --- |
| 07 | [INFO] Preparing eclipse:eclipse |
| 08 | [INFO] No goals needed for project - skipping | |

|  |  |
| --- | --- |
| 09 | [INFO] [eclipse:eclipse] |
| 10 | [INFO] Using as WTP server : null | |

|  |  |
| --- | --- |
| 11 | [INFO] Adding default classpath contaigner: org.eclipse.jdt.launching.JRE\_CONTAINER |
| 12 | [INFO] Using source status cache: /Users/liweinan/javaProj/target/mvn-eclipse-cache.properties |

|  |  |
| --- | --- |
| 13 | [INFO] Not writing settings - defaults suffice |
| 14 | [INFO] Wrote Eclipse project for "javaProj" to /Users/liweinan/javaProj. | |

|  |  |
| --- | --- |
| 15 | [INFO] |
| 16 | Sources for some artifacts are not available. | |

|  |  |  |
| --- | --- | --- |
| 17 | Please run the same goal with the -DdownloadSources=true parameter in order to check remote repositories for sources. | |
| 18 | List of artifacts without a source archive: |

|  |  |  |
| --- | --- | --- |
| 19 | o junit:junit:3.8.1 | |
| 20 |  |

|  |  |
| --- | --- |
| 21 | Javadoc for some artifacts is not available. |
| 22 | Please run the same goal with the -DdownloadJavadocs=true parameter in order to check remote repositories for javadoc. | |

|  |  |  |
| --- | --- | --- |
| 23 | List of artifacts without a javadoc archive: | |
| 24 | o junit:junit:3.8.1 |

|  |  |
| --- | --- |
| 25 |  |
| 26 | [INFO] ----------------------------------------------------- | |

|  |  |
| --- | --- |
| 27 | [INFO] BUILD SUCCESSFUL |
| 28 | [INFO] ----------------------------------------------------- | |

|  |  |
| --- | --- |
| 29 | [INFO] Total time: 2 seconds |
| 30 | [INFO] Finished at: Tue Aug 19 18:21:06 CST 2008 | |

|  |  |
| --- | --- |
| 31 | [INFO] Final Memory: 5M/10M |
| 32 | [INFO] ----------------------------------------------------- | |

如果您是第一次使用这个命令，Maven还会从网上下载一些必须的文件来帮助生成Eclipse工程文件。程序执行完毕后，您的项目中应该已经具有了生成好的Eclipse工程文件。这时我们查看项目的根目录，可以看到目录中多了 .classpath 及 .project 这两个Eclipse的工程文件：

|  |  |
| --- | --- |
| 01 | javaProj |
| 02 | |-- .classpath | |

|  |  |  |
| --- | --- | --- |
| 03 | |-- .project | |
| 04 | |-- pom.xml |

|  |  |
| --- | --- |
| 05 | |-- src |
| 06 | |   |-- main | |

|  |  |
| --- | --- |
| 07 | |   |   `-- java |
| 08 | |   |       `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 09 | |   |           `-- App.java | |
| 10 | |   `-- test |

|  |  |
| --- | --- |
| 11 | |       `-- java |
| 12 | |           `-- javaProj | |

|  |  |  |
| --- | --- | --- |
| 13 | |               `-- AppTest.java | |
| 14 | `-- target |

|  |  |
| --- | --- |
| 15 | |-- classes |
| 16 | |   `-- javaProj | |

|  |  |
| --- | --- |
| 17 | |       `-- App.class |
| 18 | |-- javaProj-1.0-SNAPSHOT.jar | |

|  |  |
| --- | --- |
| 19 | |-- maven-archiver |
| 20 | |   `-- pom.properties | |

|  |  |  |
| --- | --- | --- |
| 21 | |-- mvn-eclipse-cache.properties | |
| 22 | |-- surefire-reports |

|  |  |  |
| --- | --- | --- |
| 23 | |   |-- TEST-javaProj.AppTest.xml | |
| 24 | |   `-- javaProj.AppTest.txt |

|  |  |
| --- | --- |
| 25 | `-- test-classes |
| 26 | `-- javaProj | |

|  |  |
| --- | --- |
| 27 | `-- AppTest.class |

接下来便可以使用Eclipse很方便地导入这个工程了。 我们打开Eclipse，并在功能菜单中选择 File -> Import -> Existing Projects into Workspace 。然后选取javaProj的工程目录，点确定。这样导入工作便完成了：

http://bluedash.b0.upaiyun.com/system/photos/871/medium/javaProj-eclipse-import.jpg!medium

但是现在还有一个问题，在Eclipse中查看一下项目属性的"Java Build Path"一节：

http://bluedash.b0.upaiyun.com/system/photos/872/medium/javaProj-buildpath.jpg!medium

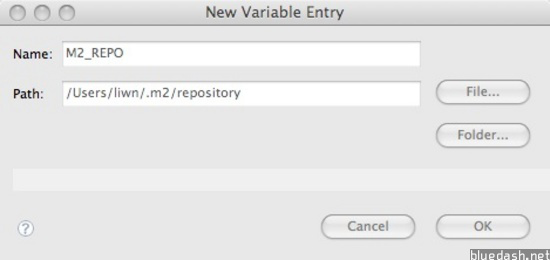
可以看到，pom中定义的依赖项目junit，相关的文件 junit-3.8.1.jar 已经从网上下载并被引入了。但是文件的指向是相对路径 M2\_REPO ，实际上Maven把所有下载下来的文件都会放在它指定的目录中。对于Windows用户来说，这个目录一般在当前用户的文档目录下，名字为 .m2 。比如某Windows用户系统登录名为anan，那么Maven会把所有的下载文件默认放在 C:\Documents and Settings\anan\.m2\repository 中。但Eclipse并不知道这个路径具体指向，需要配置一下Eclipse，把 M2\_REPO 这个路径加进去。您需要做的是点击Eclipse功能菜单上的：

|  |  |
| --- | --- |
| 1 | Window -> Preferences -> Java -> Build Path -> Classpath Variables |

如下图所示：

http://bluedash.b0.upaiyun.com/system/photos/873/medium/eclipse-buildpath.jpg!medium

这里面新建一个环境变量： M2\_REPO 。并将其指向正确的位置。由于我在写这本书时，使用的是MacOS操作系统，Maven会把 .m2 目录放在当前用户的根目录下，如下图所示：



这样，Eclipse就可以找到Maven下载的库文件，并用其编译您的项目。您现在便可以使用Eclipse来开发您的项目了。

**使用Maven创建WEB项目**

我们在前面学习了使用Maven创建一般的Java工程。现在，我来学习使用Maven创建一个Web项目。我们在前面学到过，Maven支持创建很多种不同的Java工程，那么它到底具体支持哪些种不同的项目生成模型呢？我们可以使用下面这个命令来看一下：

|  |  |
| --- | --- |
| 1 | mvn archetype:generate |

这个是Maven用来替代 @archetype:create|的命令，它的好处是可以列出一个项目模型菜单供您选择。下面是系统的输出：

|  |  |
| --- | --- |
| 01 | 1: internal -> appfuse-basic-jsf (AppFuse archetype forcreating a web application with Hibernate, Spring and JSF) |
| 02 | 2: internal -> appfuse-basic-spring (AppFuse archetype forcreating a web application with Hibernate, Spring and Spring MVC) |

|  |  |
| --- | --- |
| 03 | 3: internal -> appfuse-basic-struts (AppFuse archetype forcreating a web application with Hibernate, Spring and Struts 2) |
| 04 | 4: internal -> appfuse-basic-tapestry (AppFuse archetype forcreating a web application with Hibernate, Spring and Tapestry 4) |

|  |  |
| --- | --- |
| 05 | 5: internal -> appfuse-core (AppFuse archetype for creating a jar application with Hibernate and Spring and XFire) |
| 06 | 6: internal -> appfuse-modular-jsf (AppFuse archetype forcreating a modular application with Hibernate, Spring and JSF) |

|  |  |
| --- | --- |
| 07 | 7: internal -> appfuse-modular-spring (AppFuse archetype forcreating a modular application with Hibernate, Spring and Spring MVC) |
| 08 | 8: internal -> appfuse-modular-struts (AppFuse archetype forcreating a modular application with Hibernate, Spring and Struts 2) |

|  |  |
| --- | --- |
| 09 | 9: internal -> appfuse-modular-tapestry (AppFuse archetypefor creating a modular application with Hibernate, Spring and Tapestry 4) |
| 10 | 10: internal -> maven-archetype-j2ee-simple (A simple J2EE Java application) |

|  |  |
| --- | --- |
| 11 | 11: internal -> maven-archetype-marmalade-mojo (A Maven plugin development project using marmalade) |
| 12 | 12: internal -> maven-archetype-mojo (A Maven Java plugin development project) |

|  |  |  |
| --- | --- | --- |
| 13 | 13: internal -> maven-archetype-portlet (A simple portlet application) | |
| 14 | 14: internal -> maven-archetype-profiles () |

|  |  |
| --- | --- |
| 15 | 15: internal -> maven-archetype-quickstart () |
| 16 | 16: internal -> maven-archetype-site-simple (A simple site generation project) | |

|  |  |
| --- | --- |
| 17 | 17: internal -> maven-archetype-site (A more complex site project) |
| 18 | 18: internal -> maven-archetype-webapp (A simple Java web application) |

|  |  |
| --- | --- |
| 19 | 19: internal -> struts2-archetype-starter (A starter Struts 2 application with Sitemesh, DWR, and Spring) |
| 20 | 20: internal -> struts2-archetype-blank (A minimal Struts 2 application) |

|  |  |
| --- | --- |
| 21 | 21: internal -> struts2-archetype-portlet (A minimal Struts 2 application that can be deployed as a portlet) |
| 22 | 22: internal -> struts2-archetype-dbportlet (A starter Struts 2 portlet that demonstrates a simple CRUD interface with db backing) |

|  |  |
| --- | --- |
| 23 | 23: internal -> struts2-archetype-plugin (A Struts 2 plugin) |
| 24 | 24: internal -> shale-archetype-blank (A blank Shale web application with JSF) | |

|  |  |
| --- | --- |
| 25 | 25: internal -> maven-adf-archetype (Archetype to ease the burden of creating a new application based with ADF) |
| 26 | 26: internal -> data-app (A new Databinder application with sources and resources.) |

|  |  |
| --- | --- |
| 27 | 27: internal -> jini-service-archetype (Archetype for Jini service project creation) |
| 28 | 28: internal -> softeu-archetype-seam (JSF+Facelets+Seam Archetype) |

|  |  |  |
| --- | --- | --- |
| 29 | 29: internal -> softeu-archetype-seam-simple (JSF+Facelets+Seam (no persistence) Archetype) | |
| 30 | 30: internal -> softeu-archetype-jsf (JSF+Facelets Archetype) |

|  |  |
| --- | --- |
| 31 | 31: internal -> jpa-maven-archetype (JPA application) |
| 32 | 32: internal -> spring-osgi-bundle-archetype (Spring-OSGi archetype) | |

|  |  |
| --- | --- |
| 33 | 33: internal -> confluence-plugin-archetype (Atlassian Confluence plugin archetype) |
| 34 | 34: internal -> jira-plugin-archetype (Atlassian JIRA plugin archetype) |

|  |  |
| --- | --- |
| 35 | 35: internal -> maven-archetype-har (Hibernate Archive) |
| 36 | 36: internal -> maven-archetype-sar (JBoss Service Archive) | |

|  |  |  |
| --- | --- | --- |
| 37 | 37: internal -> wicket-archetype-quickstart (A simple Apache Wicket project) | |
| 38 | 38: internal -> quickstart (A simple Apache Tapestry 5 Project) |

|  |  |
| --- | --- |
| 39 | 39: internal -> scala-archetype-simple (A simple scala project) |
| 40 | 40: internal -> lift-archetype-blank (A blank/empty liftweb project) | |

|  |  |
| --- | --- |
| 41 | 41: internal -> lift-archetype-basic (The basic (liftweb) project) |
| 42 | 42: internal -> cocoon-22-archetype-block-plain ([<http://cocoon.apache.org/2.2/maven-plugins/>]) | |

|  |  |
| --- | --- |
| 43 | 43: internal -> cocoon-22-archetype-block ([<http://cocoon.apache.org/2.2/maven-plugins/>]) |
| 44 | 44: internal -> cocoon-22-archetype-webapp ([<http://cocoon.apache.org/2.2/maven-plugins/>]) |

可以看到，Maven为我们提供了相当多的模型选择，其中有最简单的第15号quickstart，这个就是我们刚刚生成的最普通的Java项目。也有常用的16号，生成一个简单的Web工程。还有比较经典的8号，创建一个Web工程，并包含Struts框架、Spring框架、及Hibernate框架，这样我们就不用自己去查找这些框架的pom描述信息了。

在这一节里，我们要生成一个最普通的Web项目。我们需要做的就是在终端输入Maven建立Web项目的命令，由于新的 archetype:generate 命令使用上比较简单，无需过多讲解了。这里还是使用老的 archetype:create 命令来创建项目：

|  |  |
| --- | --- |
| 1 | mvn archetype:create |
| 2 | -DarchetypeGroupId=org.apache.maven.archetypes | |

|  |  |  |
| --- | --- | --- |
| 3 | -DarchetypeArtifactId=maven-archetype-webapp | |
| 4 | -DgroupId=j2eeProj -DartifactId=j2eeProj |

与生成Java项目不同的是，这里指定了创建的模型为maven-archetype-webapp，这样Maven便会生成Web项目。这个项目的目录结构如下图所示：

|  |  |
| --- | --- |
| 1 | j2eeProj |
| 2 | |-- pom.xml | |

|  |  |
| --- | --- |
| 3 | `-- src |
| 4 | `-- main | |

|  |  |  |
| --- | --- | --- |
| 5 | |-- resources | |
| 6 | `-- webapp |

|  |  |
| --- | --- |
| 7 | |-- WEB-INF |
| 8 | |   `-- web.xml | |

|  |  |
| --- | --- |
| 9 | `-- index.jsp |

与前几节所讲的普通Java工程项目不同，这个项目中多了resources及webapp两个目录，其中resources用于放置项目的资源文件。webapp目录则是标准的WEB项目的目录结构。此外，Maven还生成了 web.xml 配置文件及 index.jsp 欢迎页面。但这个工程里似乎少了放置源代码的目录： src/main/java 。这个没有关系，您自己创建一个就行了，Maven会按它的目录结构标准，把您创建的代码目录识别出来，并在编译打包时做正确的处理。

下面我们来看看Maven自动生成的web项目的pom.xml文件的具体内容：

|  |  |  |
| --- | --- | --- |
| 01 | <project xmlns="<http://maven.apache.org/POM/4.0.0>"xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" | |
| 02 | | xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0><http://maven.apache.org/maven-v4_0_0.xsd>"> |

|  |  |  |
| --- | --- | --- |
| 03 | <modelVersion>4.0.0</modelVersion> | |
| 04 | <groupId>j2eeProj</groupId> |

|  |  |
| --- | --- |
| 05 | <artifactId>j2eeProj</artifactId> |
| 06 | <packaging>war</packaging>(!\* \label{pom.xml:packaging} \*!) | |

|  |  |
| --- | --- |
| 07 | <version>1.0-SNAPSHOT</version> |
| 08 | <name>j2eeProj Maven Webapp</name> | |

|  |  |  |
| --- | --- | --- |
| 09 | <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>j2eeProj</finalName> | |
| 20 | </build> |

|  |  |
| --- | --- |
| 21 | </project> |

在第6行，请注意打包方式是标准的Web应用格式war，而不是jar。现在完成了这个项目的创建，与一般的Java项目不同，您如果想调试网络应用项目，往往需要将整个项目部署到Web服务器当中。而Eclipse在Web应用调试方面提供了一个开源插件Calistro，但这个插件使用起来不是十分方便。在Web调试方面，还是Eclipse的商业插件MyEclipse比较好用。做为Eclipse的一个商业插件，为部署及调试WEB项目提供了非常方便的工具，使用者也非常多。为了方便使用MyEclipse的读者使用Maven，特别在下一节中介绍如何让Maven与MyEclipse协同工作，共同管理您的Web项目。如果您对MyEclipse不是很感兴趣，可以跳过下一节。

**使用Maven与MyEclipse协同工作**

我们在前一节中，学习了如何使用Maven和Eclipse来共同开发我们的项目。接下来，为了让j2eeProj项目能够使用MyEclipse的web部署及调试能力，要将这个项目声明为MyEclipse的Web Project，请保证您已经在Eclipse中正确地安装了MyEclipse插件，然后点击功能菜单上的：

|  |  |
| --- | --- |
| 1 | MyEclipse -> Add Web Project Capabilities |

此时系统弹出一个对话框，在里面要做两个修改：首先要将MyEclipse的 Web Root Directory 要指向Maven创建的webapp目录；然后，要把 Add J2EE libraries to buildpath 的选项去掉，因为我们已经用Maven来管理项目的代码库依赖关系了，就不再需要用到MyEclipse提供的库。 Create web.xml 项可去可不去，因为MyEclipse发现已经存在 web.xml 后，就不会再创建了。效果如下图所示：

http://bluedash.b0.upaiyun.com/system/photos/875/medium/myeclipse-add-web.jpg!medium

完成这一步后，还需要在项目中添加构建Web项目所必需的JSTL[7](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B4%EF%BC%8D%E6%8E%8C%E6%8F%A1Maven#fn7)及Servlet程序库。我们需要做的是在 pom.xml 中添加这两个依赖关系：

|  |  |
| --- | --- |
| 01 | <project xmlns="<http://maven.apache.org/POM/4.0.0>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" | |

|  |  |  |  |
| --- | --- | --- | --- |
| 03 | xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0><http://maven.apache.org/maven-v4_0_0.xsd>"> | | |
| 04 | | <modelVersion>4.0.0</modelVersion> |

|  |  |
| --- | --- |
| 05 | <groupId>j2eeProj</groupId> |
| 06 | <artifactId>j2eeProj</artifactId> | |

|  |  |
| --- | --- |
| 07 | <packaging>war</packaging> |
| 08 | <version>1.0-SNAPSHOT</version> | |

|  |  |
| --- | --- |
| 09 | <name>j2eeProj Maven Webapp</name> |
| 10 | <url>http://maven.apache.org</url> |

|  |  |
| --- | --- |
| 11 | <dependencies> |
| 12 | <dependency> | |

|  |  |
| --- | --- |
| 13 | <groupId>junit</groupId> |
| 14 | <artifactId>junit</artifactId> | |

|  |  |  |
| --- | --- | --- |
| 15 | <version>3.8.1</version> | |
| 16 | <scope>test</scope> |

|  |  |  |
| --- | --- | --- |
| 17 | </dependency> | |
| 18 | <dependency> |

|  |  |
| --- | --- |
| 19 | <groupId>jstl</groupId> |
| 20 | <artifactId>jstl</artifactId> | |

|  |  |  |
| --- | --- | --- |
| 21 | <version>1.1.2</version> | |
| 22 | </dependency> |

|  |  |
| --- | --- |
| 23 | <dependency> |
| 24 | <groupId>servletapi</groupId> | |

|  |  |
| --- | --- |
| 25 | <artifactId>servletapi</artifactId> |
| 26 | <version>2.4.public\_draft</version> |

|  |  |  |
| --- | --- | --- |
| 27 | </dependency> | |
| 28 | </dependencies> |

|  |  |
| --- | --- |
| 29 | <build> |
| 30 | <finalName>j2eeProj</finalName> | |

|  |  |  |
| --- | --- | --- |
| 31 | </build> | |
| 32 | </project> |

7 Java Server Pages Standard Tag Library

并在项目的根目录下重新执行：

|  |  |
| --- | --- |
| 1 | mvn eclipse:eclipse |

此时Maven会从网上自动下载所需的项目，并更新Eclipse工程文件。这时再回到Eclipse，刷新一下项目，右键点击本项目，选择

|  |  |
| --- | --- |
| 1 | Build Path -> Configure Build Path -> Java Build Path |

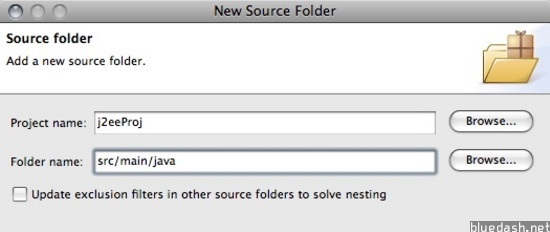
可以发现这时相关的项目已被引入了：

http://bluedash.b0.upaiyun.com/system/photos/876/medium/myeclipse-add-dep.jpg!medium

这时还存在一个问题，这个项目里没有放置Java代码的地方，您的项目不可能是只由JSP页面文件组成的，因此您还需要添加一个目录，用于放置代码。在Eclipse中右键点击本工程，选择

|  |  |
| --- | --- |
| 1 | new -> Source Folder |

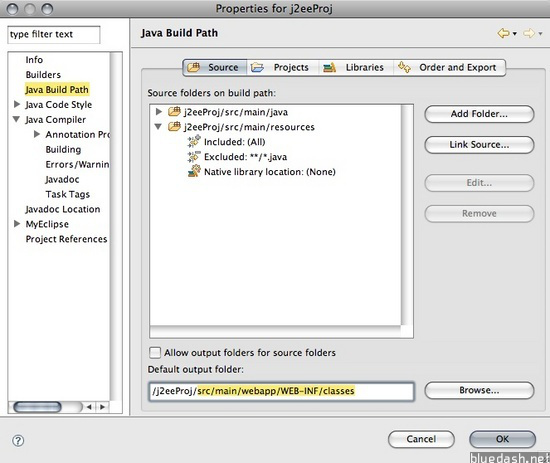
按Maven的目录结构标准添加一个 main/java：



完成这一步后，您还需要让MyEclipse在部署项目时，把编译过的程序放置在正确的位置，即 WEB-INF/classes/ 目录中，这与Maven生成的Eclipse工程的默认位置是相冲突的，因此您需要手工指定一下：在Eclipse中右键点击本项目，选

|  |  |
| --- | --- |
| 1 | Properties -> Java Build Path -> Source |

将 Default output folder 改为 src/main/webapp/WEB-INF/classes ，这样MyEclipse就可以正确部署此项目了[8](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B4%EF%BC%8D%E6%8E%8C%E6%8F%A1Maven#fn8)：



8 请注意，每执行完mvn eclipse:eclipse后，都需要重复此操作。因为Maven会把Java Build Path恢复成Maven的默认位置：target目录中，这样的话我们的Web应用就不能正确地执行了。

**添加新的Maven仓库**

细心的读者可能会发现一个问题。既然Maven从网上自动下载所需的项目包，那么这些文件是从哪里下载来的？答案很简单，它从Maven的官方项目仓库下载，那么官方的仓库能否包含了所有您想要的程序库呢？答案是不能！因此，当Maven从官方的代码仓库找不到您所需的项目时，它便会在编译或打包过程中报错，就像下面这样：

|  |  |
| --- | --- |
| 01 | Missing: |
| 02 | ---------- | |

|  |  |  |
| --- | --- | --- |
| 03 | 1) javaee:javaee-api:jar:5 | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | Try downloading the file manually from the project website. | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | Then, install it using the command: |
| 08 | mvn install:install-file -DgroupId=javaee -DartifactId=javaee-api \ | |

|  |  |  |
| --- | --- | --- |
| 09 | -Dversion=5 -Dpackaging=jar -Dfile=/path/to/file | |
| 10 |  |

|  |  |
| --- | --- |
| 11 | Path to dependency: |
| 12 | 1) user-tool:user-tool:jar:1.0-SNAPSHOT | |

|  |  |  |
| --- | --- | --- |
| 13 | 2) javaee:javaee-api:jar:5 | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | ---------- |
| 16 | 1 required artifact is missing. | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | for artifact: | |

|  |  |  |
| --- | --- | --- |
| 19 | user-tool:user-tool:jar:1.0-SNAPSHOT | |
| 20 |  |

|  |  |
| --- | --- |
| 21 | from the specified remote repositories: |
| 22 | central (<http://repo1.maven.org/maven2>) | |

Maven抱怨它找不到 javaee-api 这个包。因为这个包由Sun发行，在Maven官网上不存在。因此它给出建议让用户用自行下载。并给出了安装命令，告诉您在下载完成后，使用mvn install:install 命令把这个包手工添加到本地的 .m2 仓库中。当然按照Maven的建议来做是可以解决这个问题的。但这不是最好的办法，网上的Maven仓库有千千万万，为什么一定要在Maven的官方仓库中下载呢？比如 javaee-api 这个包在别的maven仓库里会有也说不定。因此我们可以"javaee-api maven"为关键字，在Google上进行搜索，果然找到了在 http://download.java.net/maven/1 这个Maven仓库里面有这个包。因此我们在自己项目的pom.xml中，加上这个Maven仓库，Maven可以在执行命令的时候从这个站台下载我们所需要的文件了。我们在 pom.xml 文件中添加这个Maven仓库：

|  |  |
| --- | --- |
| 1 | <repositories> |
| 2 | <repository> | |

|  |  |
| --- | --- |
| 3 | <id>java.net</id> |
| 4 | <url>[http://download.java.net/maven/1</](http://download.java.net/maven/1%3C/)url> | |

|  |  |  |
| --- | --- | --- |
| 5 | <layout>legacy</layout> | |
| 6 | </repository> |

|  |  |
| --- | --- |
| 7 | </repositories> |

在这里面有几个元素：其中id随便起一个名字即可；url是仓库位置必须写对；layout是说明仓库的结构。由于找到的仓库是Maven 1.x版本时使用的结构，因此指定layout为legacy。对于新版本的仓库结构就不用指定了[9](http://bluedash.net/spaces/Web%E5%BC%80%E5%8F%91%E6%95%99%E7%A8%8B4%EF%BC%8D%E6%8E%8C%E6%8F%A1Maven#fn9)。添加完成后，再次运行Maven任务：

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Downloading:<http://download.java.net/maven/1/javaee/poms/javaee-api-5.pom> | | |
| 2 | | 1K downloaded |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | Downloading:<http://download.java.net/maven/1/javaee/jars/javaee-api-5.jar> | | |
| 4 | | 610K downloaded |

9 在网上，Maven 1.x的仓库一般用 maven/1/ 来命名，Maven 2.x的一般是 maven/2/

可以看到这个包从添加的Maven仓库中成功下载了。

## 小结

我们在这章学习了Maven的基本使用技巧。这个工具十分强大，使用起来非常方便。要想熟练使用它，您还需要亲自动手，在您自己的项目中不断地使用它。

[Eclipse下使用maven开发web项目](http://blog.csdn.net/moviethief/article/details/5653109)

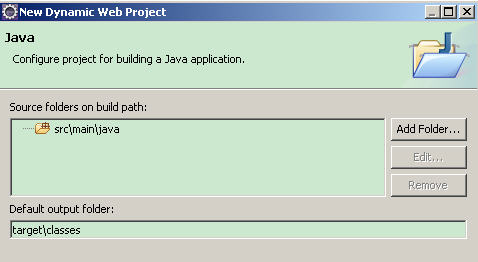
分类： [java](http://blog.csdn.net/moviethief/article/category/667788)2010-06-07 15:22 807人阅读 [评论](http://blog.csdn.net/moviethief/article/details/5653109#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/moviethief/article/details/5653109#report)

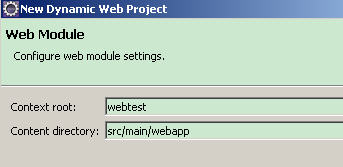
这两天一直在折腾Eclipse下maven开发web项目，maven构建的web项目的文件目录比较奇特，而且依赖又不是用lib下面的，之前不知道哪里出现的问题，maven依赖的jar包始终不能部署到tomcat里面，后来重新新建了项目，一步一步重新弄成功了，记录一下

1、创建web项目

（1）使用maven命令：  
mvn archetype:create -DgroupId=xxxxx -DartifactId=webtest -DarchetypeArtifactId=maven-archetype-webapp  
然后执行mvn eclipse:eclipse 使项目兼容Eclipse

（2）是用Eclipse创建，创建过程中Content Directory 填写src/main/webapp, Java Source Directory 填写src/main/java





这样就跟 maven 的产生的项目布局一样了

2、选中项目，右点击属性，更改项目属性设置

设置J2EE Module Dependencies ,把 Maven Dependenecies 选中（目的是为builder时输出依赖的jar到指定目录 )

这样就可以了，如此之简单，我折腾了好久。。。。

## mvn创建web项目步骤

1安装mvn（包括path）  
  
2命令：mvn archetype:create -DgroupId=cn.edu.sdau.neat -DartifactId=Project -DarchetypeArtifactId=maven-archetype-webapp  
其中，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}  
我只使用过最后一个骨架，每一个骨架都会建相应的目录结构和一些通用文件。至于mvn的完整目录结构定义，可参考：  
http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html  
   如果在项目中有servlet/jsp，推荐jetty容器，它可做为一个插件嵌入到mvn中，使用时，只需要在pom.xml中加入：  
    chenheng  
            org.mortbay.jetty  
            maven-jetty-plugin  
使用：mvn jetty:run  
就可以在浏览器中测试了，非常方便。  
  
3结合eclipse  
mvn eclipse:clean  
mvn eclipse:eclipse  
普通 Eclipse 项目执行 ： mvn eclipse:eclipse Eclipse  
web 项目执行 ： mvn eclipse:eclipse –Dwtpversion=1.0   
  
  
  
4.eclipse导入创建的web项目问题  
  
                在用eclipse导入web项目时 如果eclipse安装了wtp插件在会自动将其转化为eclipse认得到的web项目，在项目上面点击右键run as 的run on server可以用，但是当eclipse没有安装wtp插件时run on server则不用，此时的解决方式为用命令行切换到项目的根目录然后执行 mvn eclipse:eclipse –Dwtpversion=1.0即可解决。  
  
src/test/java  
src/main/java  
src/main/resources

maven3 部署webApp to tomcat

首先保证tomcat启动

然后在pom.xml中加入

<build>

       <plugins>

               <plugin>

               <groupId>org.codehaus.mojo</groupId>

               <artifactId>tomcat-maven-plugin</artifactId>

               <configuration>

               <server>tomcat</server>

               </configuration>

               </plugin>

       </plugins>

       <finalName>topic</finalName>

</build>

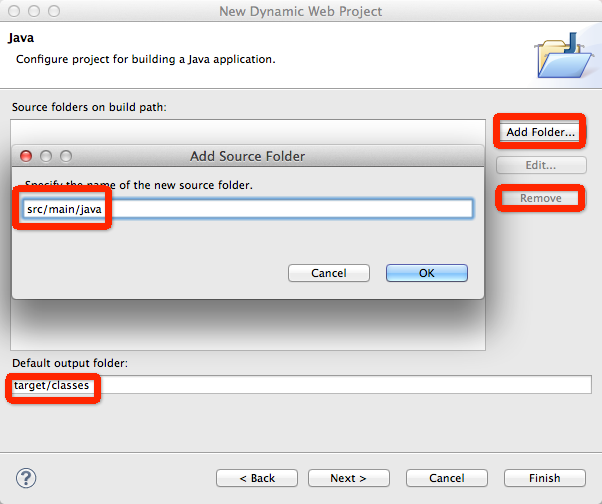
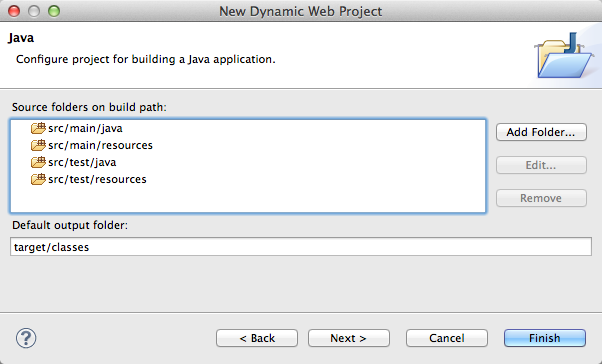
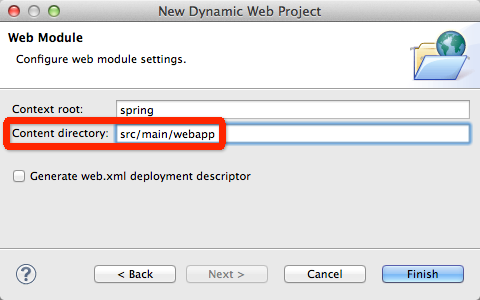
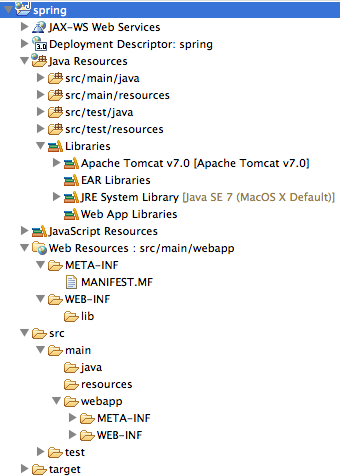
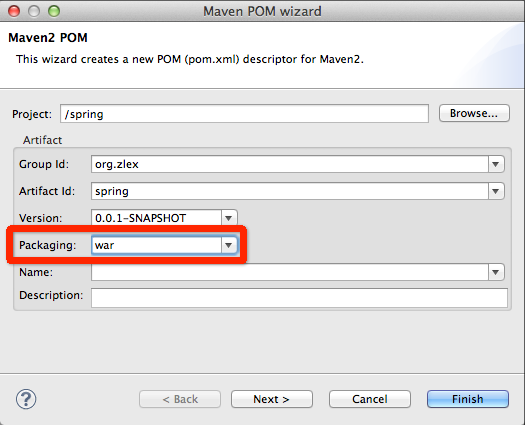
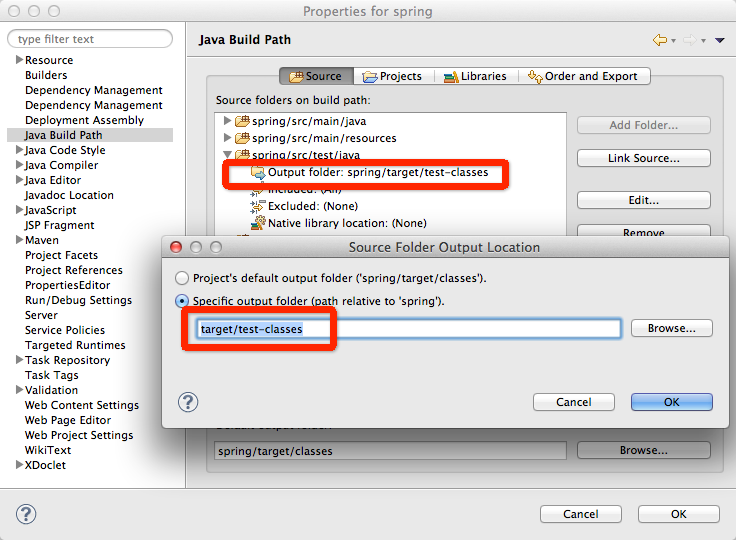
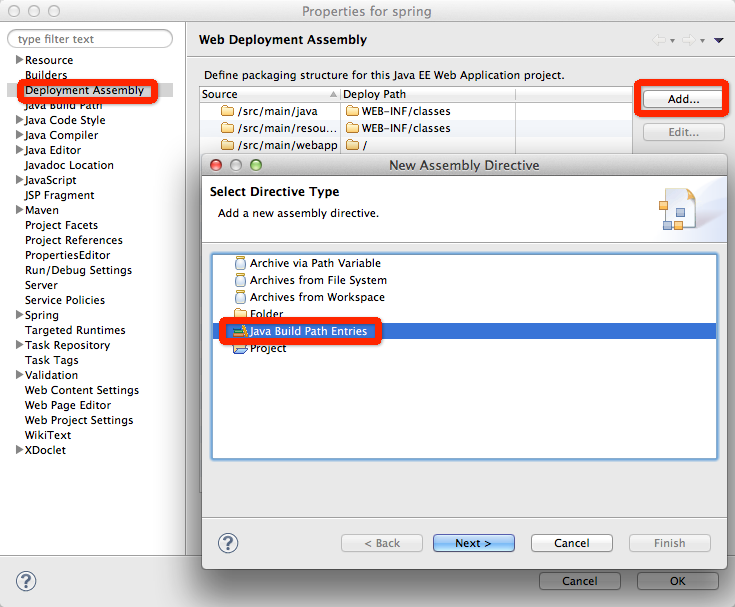
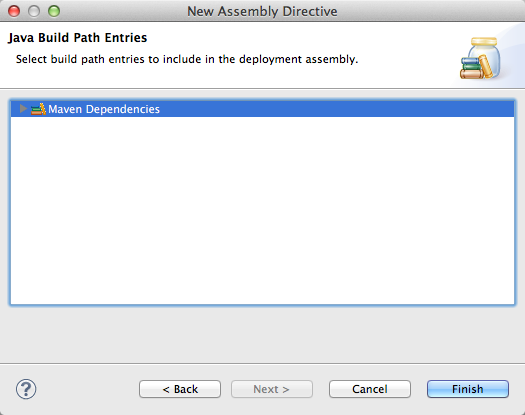
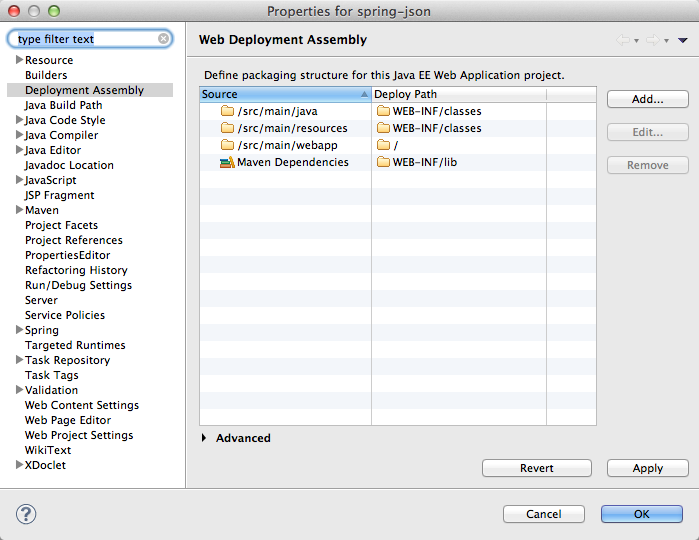
运行mvn tomcat：redeploy

### [Eclipse+Maven快速生成Web项目，解决部署时Maven lib依赖问题](http://snowolf.iteye.com/blog/1627343)

**博客分类：**

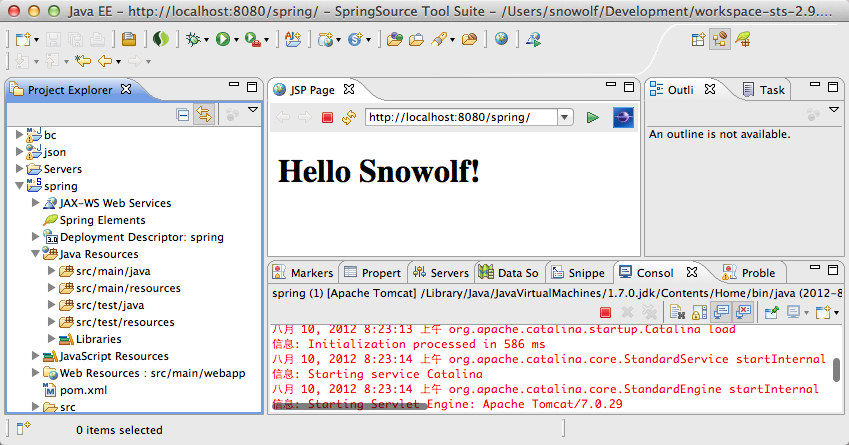
* [Eclipse](http://snowolf.iteye.com/category/38039)
* [Java／Maven](http://snowolf.iteye.com/category/147023)

[eclipse](http://www.iteye.com/blogs/tag/eclipse)[maven](http://www.iteye.com/blogs/tag/maven)[web](http://www.iteye.com/blogs/tag/web)[wtp](http://www.iteye.com/blogs/tag/wtp)[reference](http://www.iteye.com/blogs/tag/reference)

用Maven开发做了很久了，用Eclipse做开发就更久了。自打用Eclipse开发Web项目，就一直用着Eclipse的WTP，离开了WTP的Server，就超级不习惯。虽然Maven已经3.0时代了，虽然Maven For Eclipse的插件也做了很久了，但还是不够好用，兼容做的不够好。http://snowolf.iteye.com/images/smiles/icon_sad.gif   
  
接下来，我们通过简单配置，解决wtp发布工程时，对于Maven的Lib依赖！http://snowolf.iteye.com/images/smiles/icon_biggrin.gif   
  
通过mvn命令的方式产生wtp的eclipse项目的方法，我已经尝试过了，还不如新建**Dynamic Web Project**，然后将其转化为**Maven Project**。   
  
新建**Dynamic Web Project**，选择源代码路径时，与Maven Project保持一致，删除原有**src**目录，直接建立**src/main/java**、**src/main/resources**、**src/test/java**、**src/test/resources**目录。同时，指定输出目录为**target/class**   
   
全部目录：   
   
  
构建**Web Module**时，调整下**Content directory**，与Maven Webapp项目保持一致。   
   
  
这时目录结构如下：   
   
  
转化**Maven Project**时，要注意**Packaging**选为**war**：   
   
  
调整测试源代码输出路径：   
  
  
如果你跳过了上面的步骤已经建立的web项目，但想引用Maven的Lib怎么办？   
  
选中**Maven Dependencies**   
   
  
最后得到这个结果：   
  
  
或者直接修改**.classpath**文件，找到**<classpathentry kind="con" path="org.maven.ide.eclipse.MAVEN2\_CLASSPATH\_CONTAINER"/>**   
替换为：

**Xml代码  [收藏代码](javascript:void())**

1. **<classpathentry** kind="con" path="org.maven.ide.eclipse.MAVEN2\_CLASSPATH\_CONTAINER"**>**
2. **<attributes>**
3. **<attribute** name="org.eclipse.jst.component.dependency" value="/WEB-INF/lib"**/>**
4. **</attributes>**
5. **</classpathentry>**

也是一样的效果！http://snowolf.iteye.com/images/smiles/icon_biggrin.gif   
随便写个页面，在eclipse运行server：   
  
  
搞定！   
这基本上不费什么力气，比用**mvn eclipse:eclipse –Dwtpversion=2.0**的种种方便多了。http://snowolf.iteye.com/images/smiles/icon_biggrin.gif