**通向八段的道路**

**第十九天**

lifetragedy

目录

[一、 前言 3](#_Toc338932651)

[二、 传统构建与使用maven构建 3](#_Toc338932652)

[三、 使用Maven 4](#_Toc338932653)

[3.1 使用前的准备步骤 4](#_Toc338932654)

[3.2 理解maven 4](#_Toc338932655)

[3.3使用maven 5](#_Toc338932656)

[3.3.1设置环境变量 5](#_Toc338932657)

[3.3.2修改settings文件 6](#_Toc338932658)

[3.3.3 使用maven构建工程 7](#_Toc338932659)

[3.3.4 将原有的alpha工程迁移至alpha\_mvn工程 10](#_Toc338932660)

[迁移jar文件 10](#_Toc338932661)

[pom.xml 11](#_Toc338932662)

[迁移原有alpha工程的resource 19](#_Toc338932663)

[迁移原有alpha工程的src 20](#_Toc338932664)

[迁移原有alpha工程的web文件 20](#_Toc338932665)

[3.3.5 使用maven打包 20](#_Toc338932666)

[四、如何让Maven构建的工程在eclipse里跑起来 24](#_Toc338932667)

[4.1 在eclipse里新增一个jdk的run time 25](#_Toc338932668)

[4.2 在eclipse里设置tomcat 26](#_Toc338932669)

[4.3 将alpha\_mvn转变成可在eclipse里运行的工程 27](#_Toc338932670)

[附录、Oracle Sun JDK安全限制 31](#_Toc338932671)

# 前言

上次大家拿了我上传的工程后，有些人自己通过spring3, struts1.3，hibernate3的download的包自行去装配jar包到工程的WEB-INF\lib目录下。有些是通过我上传的alpha\_lib包把里面的jar文件一股脑的copy到了工程的WEB-INF\lib目录下去。

有时经常还会发生少包了，ClassNotFound这样的错误，或者是一些因为缺包还引起的各种莫名奇妙的错误，呵呵，是不是够折腾的啊？

尝过苦头了，才知道幸苦！那么我们有没有一种更好的方式，比如说：

我下了博主的工程，打一条命令或者是在eclipse里build一下，这个工程需要的jar文件自动跑到我的工程里呢？

有的！这就是maven！！！

因此今天就要讲利用maven来构建我们的工程**以及如何在eclipse里跑由maven构建出的web工程**（**网上这方面资料不全**，**因此个人总结了经验一并分享给了大家**）。

# 传统构建与使用maven构建

**传统模式：**

《第十八》天中的这个工程的构建就是一个标准的传统模式的构建，为大多数人所接受。

它把所有的jar文件都摆放在工程的WEB-INF\lib目录下并使用ant来打包和发布. 这样做的好处是简单、直观、明了。坏处是：构建该工程的人除非很有经验，否则就会出现因为少jar文件或者因为jar文件重复而在布署或者是在运行时引起各种各样的稀奇古怪的错误。

有甚者喜欢把一个下载的spring3.1和struts1.3下载包里的的lib目录下所有的jar不管三七二十一全部copy到工程的WEB-INF\lib目录下，有提示需要覆盖他也就选个“ALL”。

我曾看到过一个工程，没几个JSP，CLASS，工程达287mb之大，其中286mb为jar文件，彼彼皆是log4j-1.3.8.jar, log4j-1.4.2.jar这样的重复的jar文件的存在。

有了junit3.8竟然还发觉工程里有junit4.x的包。。。晕啊。

**MAVEN模式：**

而使用maven构建工程时就不太会有这样的事情发生了，大家初学者可以把maven看成是一个“自动取包机”。怎么解释呢，举个例子来说：

你要装载a.jar，但a.jar依赖b.jar，而b.jar又依赖c.jar和d.jar，如果只是2，3级的依赖关系构建者可以记得住，但如果这个依赖关系达6级，7级以上时那么按照传统的构建模式当你发觉一个目录里有abcdefghijk这些个jar文件时，你一般为了避免出错总是一古脑的把所有的jar文件copy进工程，对吧？

但实际这些个jar文件里只有a.jar, b.jar, c.jar才是你需要的。

而maven干这个是它的特长，**当你告诉maven你要下载a.jar文件时，它会发觉a.jar依赖于其它的jar文件，它就会把你指定的jar文件与相关的依赖文件全部拿下来，不会多拿**（90%情况下）。

就好比你要拿org.springframework.web.struts-3.0.0.RELEASE.jar这个文件，但其实它还需要用到asm, collection, bean-util等，那么你只要告诉maven我要拿org.springframework.web.struts-3.0.0.RELEASE.jar这个文件，其它的依赖的相关的jar包会自动“下载”到你的工程中去. 所以当我拿 maven重新去构建那个286mb的工程时，工程所有的功能一点不影响，靠着maven对jar包的自动依赖整个工程从286mb缩成了21mb，哈哈哈哈。。。。。。搞毛啊原来的那帮人在！

# 使用Maven

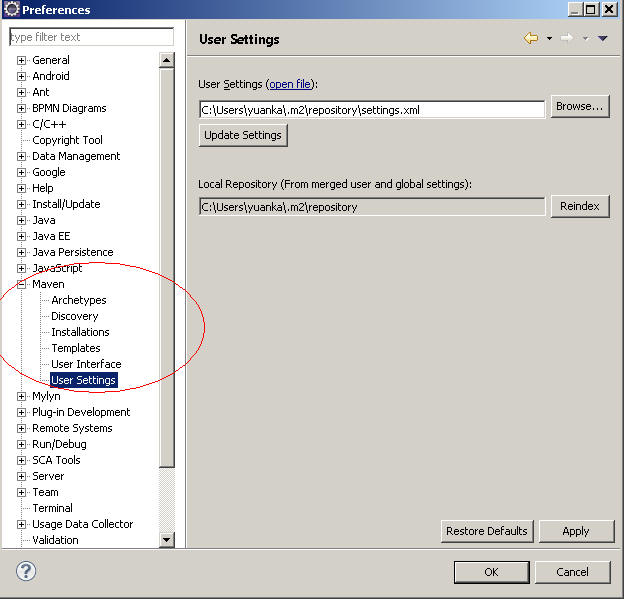
## 3.1 使用前的准备步骤

**准备步骤一：**

请下载最新的maven2，如：apache-maven-3.0.3-bin.tar.gz（我现在使用的就是这个版本），我也放到我的CSDN的“资源”中去了.

**准备步骤二：**

确保你的eclipse为WTP版并且升级了maven for eclipse，即在你的eclipse->window->preference里有maven的选项：



最好的办法就是去eclipse的官网下载一个eclipse wtp版（<http://download.eclipse.org/webtools/downloads/>），然后升级，可能升级过程会比较长，5-6小时也是有可能的（笔者家里的网速是20MB光纤）。但是第一次升完级后，以后每周没事就让你的eclipse去update一下也是有必要的，必经这东西是你以后吃饭的家伙，维持它永远是最新版本是个好习惯。

## 3.2 理解maven

请把以下几个概念记住就行了：

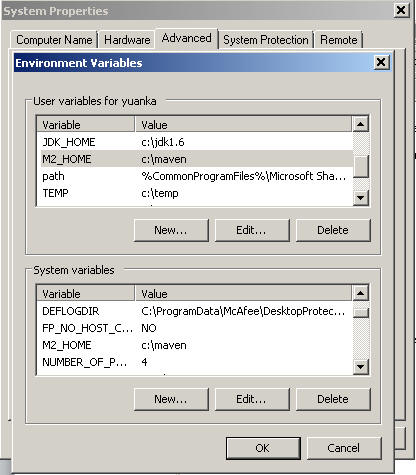
1. maven是**通过布署在*internet上的maven仓库（开源免费）*去拿你需要的jar包**，因此建议你因该是在一个宽带连网的环境下工作；
2. maven不是万能，有时会多拿包，当然这种情况很少但它会自动帮你把有依赖关系的包全拿到本地来的；
3. 由于maven是通过internet去拿 你需要的jar包的，因此你的工程发布可以不用再把几十mb的jar文件连同你的工程一起发布了，而只需要把一份“jar 包使用清单”伴随着你的工程一起发布就可以了。当别人拿着你的maven工程时，它只要也装有maven也能连入internet环境，它就可以在布署时自动把工程需要用到的jar包取下来然后布署入j2ee服务器的。
4. 如果你所在的环境比如说一些公司不让员工上网或者让上网但不让下载的，**那么这时你就需要建立自己的本地maven库，即maven私服**. 把所有的jar包通过正式的手段搞到后在你所在的环境的局域网内建立一台maven服务器，把这些通过下载收集到的jar布署到你的“私服”上去，再把工程的maven连接仓库的url指向你本地的这台私服就行了。

## 3.3使用maven

### 3.3.1设置环境变量

把apache-maven-3.0.3-bin.tar.gz这个文件解压成maven放在c盘或者是d盘的根目录（你要放什么目录，随便，我喜欢用简单直观的名字放在磁盘根目录下）。

然后在系统环境变量中设一个M2\_HOME，使其指向你的maven所在的目录：



### 3.3.2修改settings文件

**我们打开M2\_HOME/conf目录下的settings.xml文件：**

修改repositories这段，使其如下内容：

|  |
| --- |
| <repositories>  <repository>  <id>Ibiblio</id>  <name>Ibiblio</name>  <url>http://www.ibiblio.org/maven/</url>  </repository>  <repository>  <id>PlanetMirror</id>  <name>Planet Mirror</name>  <url>http://public.planetmirror.com/pub/maven/</url>  </repository>  </repositories> |

因为maven默认的internet仓库里的文件不全，所以笔者为大家提供两个目前大家用得最多的maven的internet仓库，由其是这个lbiblio的，基本所有的开源的框架的jar文件都有提供.

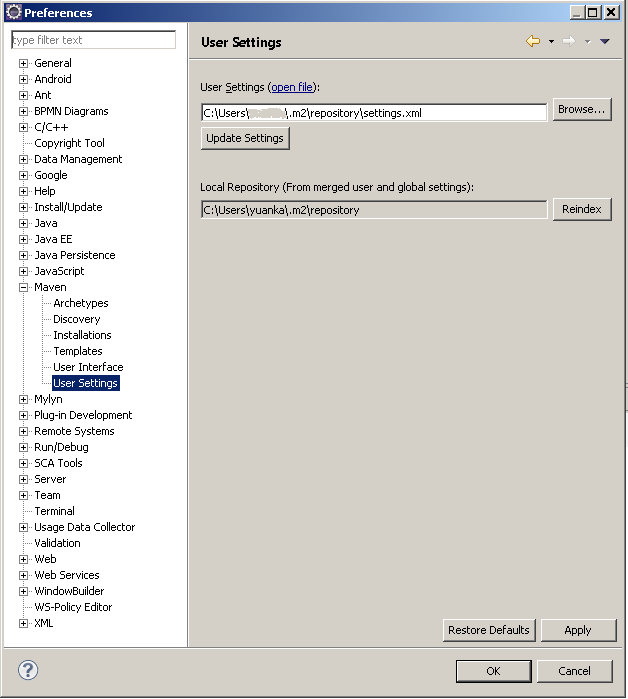
继续看下去，看到proxies这段，把它放开来，**改成如下内容（如果你所在的环境是通过proxy上网的）**，要不然请保证这段proxies为被注释.

|  |
| --- |
| <proxies>  <proxy>  <id>optional</id>  <active>true</active>  <protocol>http</protocol>  <username>username</username>  <password>password</password>  <host>proxy.mycompany.com</host>  <port>8088</port>  <nonProxyHosts>local.net|some.host.com</nonProxyHosts>  </proxy>  </proxies> |

如果你通过的proxy需要用户名密码的话不要忘了把用户名密码也设上，如果没有用户名密码，请保持<username>为username，<password>为password。

### 3.3.3 使用maven构建工程

现在可以打开eclipse了，打开eclipse->window->preference->maven

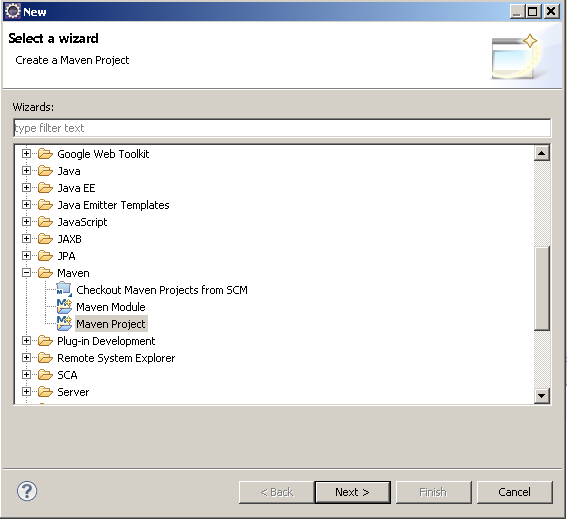


按照上图，点这个Browse…按钮，它会打开c:\users\YourCurrentUserName\.m2\repository\文件夹。

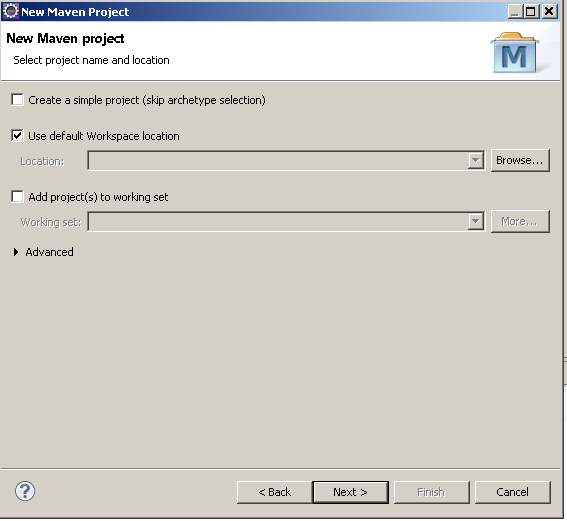
把你的M2\_HOME\conf\目录下的settings.xml文件手动copy（是copy不是move）到

c:\users\YourCurrentUserName\.m2\repository\文件夹中去后，点ok返回该界面，再点“update settings”按钮即可。

在eclipse里新建工程时选”maven project”。

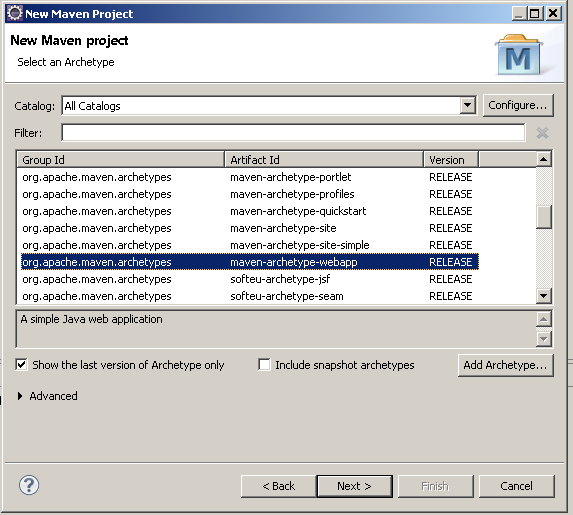


[Next]



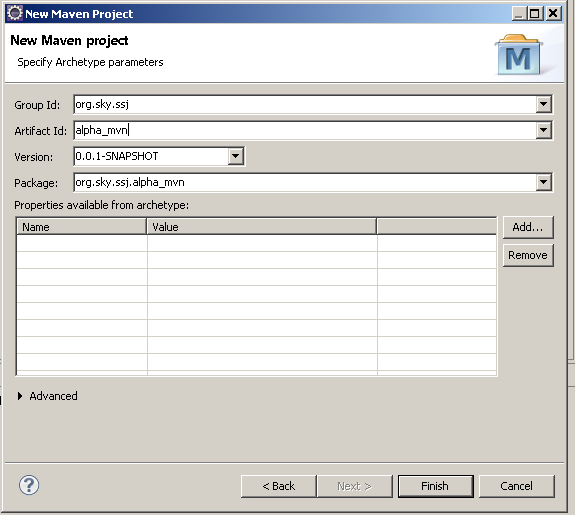
[Next]

这边要选maven-archetype-webapp这个类型，即建立一个标准的基于maven的web工程.



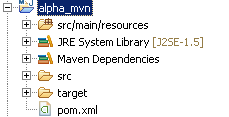
[Next]

* 这边的Groud Id为你的包名我们这边需要填入：org.sky.ssh1.alpha
* Artifact Id为工程名我们就填入alpha\_mvn，代表这个工程是maven建的alpha工程
* 然后version保持不变
* package填入org.sky.ssh1.alpha\_mvn即可



[Next]

Maven将会在eclipse里为你生成这样的一个工程。



请展开src目录，并确保你的工程的目录如下图所示



你可能需要手动在src/main下自己建立一个java的目录，其它的maven在构建工程时应该已经为您建好了。

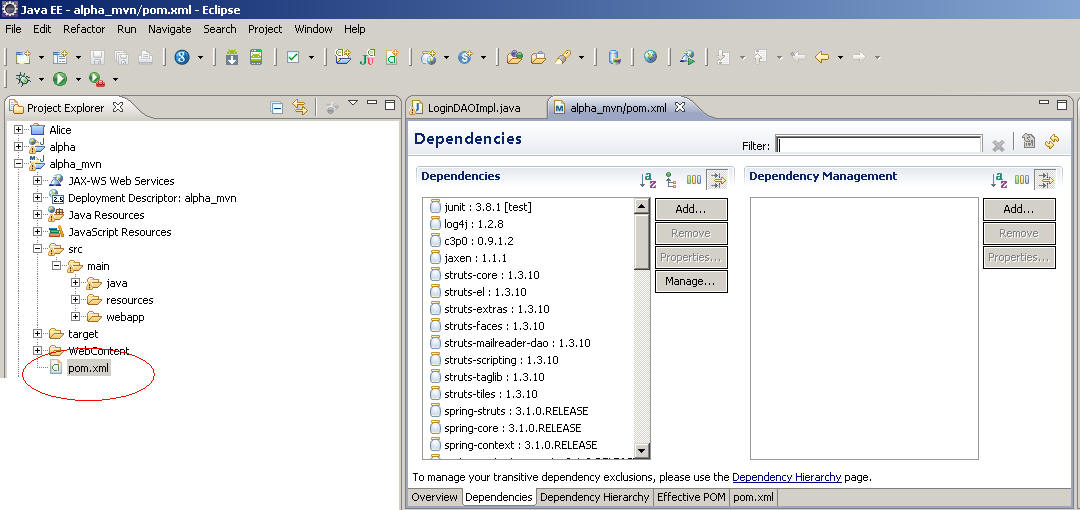
接下来我们就开始迁移原来的alpha工程进我们的maven构建的alpha\_mvn工程中去了.

### 3.3.4 将原有的alpha工程迁移至alpha\_mvn工程

### 迁移jar文件

这边我们就不是手动一个个copy jar文件啦！

Maven是通过一个pom.xml文件来描述你的工程和工程中所用到的jar 文件有哪些的. 在这里我把alpha工程需要用到的pom.xml文件整个在这边提供给了大家，大家可以直接覆盖原有工程的pom.xml文件。



### pom.xml

|  |
| --- |
| <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>org.sky.ssh1.alpha</groupId>  <artifactId>Alpha\_MVN</artifactId>  <packaging>war</packaging>  <version>0.0.1-SNAPSHOT</version>  <name>Alpha\_MVN 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>  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.8</version>  </dependency>  <dependency>  <groupId>c3p0</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.1.2</version>  </dependency>  <dependency>  <groupId>jaxen</groupId>  <artifactId>jaxen</artifactId>  <version>1.1.1</version>  <exclusions>  <exclusion>  <artifactId>xercesImpl</artifactId>  <groupId>xerces</groupId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.apache.struts</groupId>  <artifactId>struts-core</artifactId>  <version>1.3.10</version>  </dependency>  <dependency>  <groupId>org.apache.struts</groupId>  <artifactId>struts-el</artifactId>  <version>1.3.10</version>  </dependency>  <dependency>  <groupId>org.apache.struts</groupId>  <artifactId>struts-extras</artifactId>  <version>1.3.10</version>  </dependency>  <dependency>  <groupId>org.apache.struts</groupId>  <artifactId>struts-faces</artifactId>  <version>1.3.10</version>  </dependency>  <dependency>  <groupId>org.apache.struts</groupId>  <artifactId>struts-mailreader-dao</artifactId>  <version>1.3.10</version>  </dependency>  <dependency>  <groupId>org.apache.struts</groupId>  <artifactId>struts-scripting</artifactId>  <version>1.3.10</version>  </dependency>  <dependency>  <groupId>org.apache.struts</groupId>  <artifactId>struts-taglib</artifactId>  <version>1.3.10</version>  </dependency>  <dependency>  <groupId>org.apache.struts</groupId>  <artifactId>struts-tiles</artifactId>  <version>1.3.10</version>  </dependency>  <!-- springframework 3.1 -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-struts</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-orm</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc-portlet</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jms</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-asm</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>3.1.0.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springmodules</groupId>  <artifactId>spring-modules-jakarta-commons</artifactId>  <version>0.8a</version>  </dependency>  <!-- aspectj -->  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjrt</artifactId>  <version>1.6.12</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>1.6.12</version>  </dependency>  <!-- hibernate 3.3.1 -->  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-core</artifactId>  <version>3.3.1.GA</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-c3p0</artifactId>  <version>3.3.1.GA</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-ehcache</artifactId>  <version>3.3.1.GA</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-entitymanager</artifactId>  <version>3.3.1.ga</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-commons-annotations</artifactId>  <version>3.3.0.ga</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-annotations</artifactId>  <version>3.3.1.GA</version>  </dependency>  <!-- log4j 1.2.14 -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.16</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>1.6.4</version>  </dependency>  <!-- commons utils -->  <dependency>  <groupId>commons-beanutils</groupId>  <artifactId>commons-beanutils</artifactId>  <version>1.8.3</version>  </dependency>  <dependency>  <groupId>commons-chain</groupId>  <artifactId>commons-chain</artifactId>  <version>1.2</version>  </dependency>  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>1.6</version>  </dependency>  <dependency>  <groupId>commons-collections</groupId>  <artifactId>commons-collections</artifactId>  <version>3.2.1</version>  </dependency>  <dependency>  <groupId>commons-configuration</groupId>  <artifactId>commons-configuration</artifactId>  <version>1.7</version>  </dependency>  <dependency>  <groupId>commons-digester</groupId>  <artifactId>commons-digester</artifactId>  <version>2.1</version>  </dependency>  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>1.2.2</version>  </dependency>  <dependency>  <groupId>commons-io</groupId>  <artifactId>commons-io</artifactId>  <version>2.1</version>  </dependency>  <dependency>  <groupId>commons-lang</groupId>  <artifactId>commons-lang</artifactId>  <version>2.6</version>  </dependency>  <dependency>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  <version>1.1.1</version>  </dependency>  <dependency>  <groupId>commons-net</groupId>  <artifactId>commons-net</artifactId>  <version>3.0.1</version>  </dependency>  <dependency>  <groupId>commons-pool</groupId>  <artifactId>commons-pool</artifactId>  <version>1.6</version>  </dependency>  <dependency>  <groupId>commons-validator</groupId>  <artifactId>commons-validator</artifactId>  <version>1.3.1</version>  </dependency>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-compress</artifactId>  <version>1.3</version>  </dependency>  <!-- jsp servlet api -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.4</version>  <scope>compile</scope>  </dependency>  <!-- mail -->  <dependency>  <groupId>org.apache.velocity</groupId>  <artifactId>velocity</artifactId>  <version>1.7</version>  </dependency>  <dependency>  <groupId>javax.mail</groupId>  <artifactId>mail</artifactId>  <version>1.4.4</version>  </dependency>  <!-- jasypt -->  <dependency>  <groupId>org.jasypt</groupId>  <artifactId>jasypt</artifactId>  <version>1.9.0</version>  </dependency>  <dependency>  <groupId>org.jasypt</groupId>  <artifactId>jasypt-spring3</artifactId>  <version>1.9.0</version>  </dependency>  <dependency>  <groupId>org.jasypt</groupId>  <artifactId>jasypt-springsecurity3</artifactId>  <version>1.9.0</version>  </dependency>  <!-- ehCache -->  <dependency>  <groupId>net.sf.ehcache</groupId>  <artifactId>ehcache</artifactId>  <version>1.6.2</version>  </dependency>  <!-- test -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.10</version>  </dependency>  <dependency>  <groupId>org.dbunit</groupId>  <artifactId>dbunit</artifactId>  <version>2.4.8</version>  </dependency>  <dependency>  <groupId>mockit</groupId>  <artifactId>jmockit</artifactId>  <version>0.999.4</version>  </dependency>  </dependencies>  <build>  <finalName>Alpha\_MVN</finalName>  </build>  </project> |

大家可以看到我需要什么包，只要在这个pom.xml文件中加入这样的一段东西

|  |
| --- |
| <dependency>  <groupId>c3p0</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.1.2</version>  </dependency> |

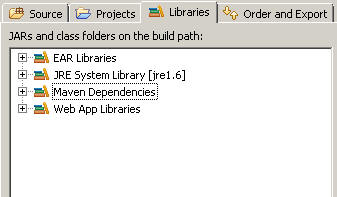
**那么有人要问我就算知道我要下一个c3p0.jar但这个artifactid与version我怎么填？**

**简单，使用这个网址：**[**http://www.ibiblio.org/**](http://www.ibiblio.org/)

**在里面搜你要的jar，搜完后它会出一个列表，然后找到你需要的版本，点进去后看看有没有后缀名为.pom的文件，如果有，直接在IE中打开该文件或者下载下来后使用纯文本编辑器打开,里面就有你要的artifactid与version的正确描述了，然后填入工程的pom.xml文件不就行了.**

当你的pom.xml文件没有**红色的叉叉**（我叉叉PLMM的圈圈，嘿）时，eclipse就开始连上在M2\_HOME/conf/setting.xml文件中描述的相关的maven的repository去拿 jar了，拿下来的jar文件：

1. 存放在本地的C:\Users\YourCurrentUserName\.m2\repository目录下；
2. 在工程中生成一个classpath

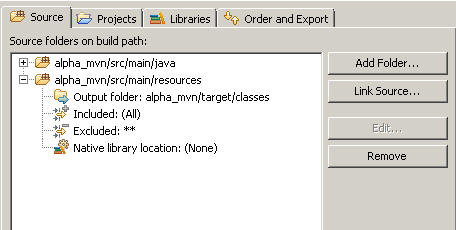


### 迁移原有alpha工程的resource

整个copy到alpha\_mvn的src/main/resources目录下即可.

### 迁移原有alpha工程的src

整个copy到alpha\_mvn的src/main/java目录下，并把src/main/java设成src目录

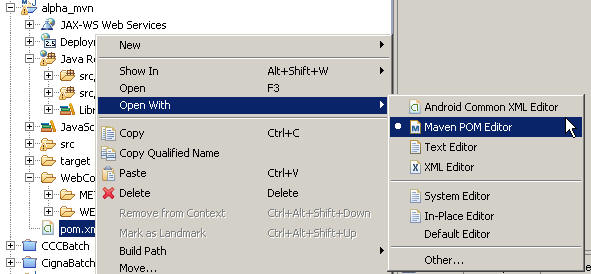


### 迁移原有alpha工程的web文件

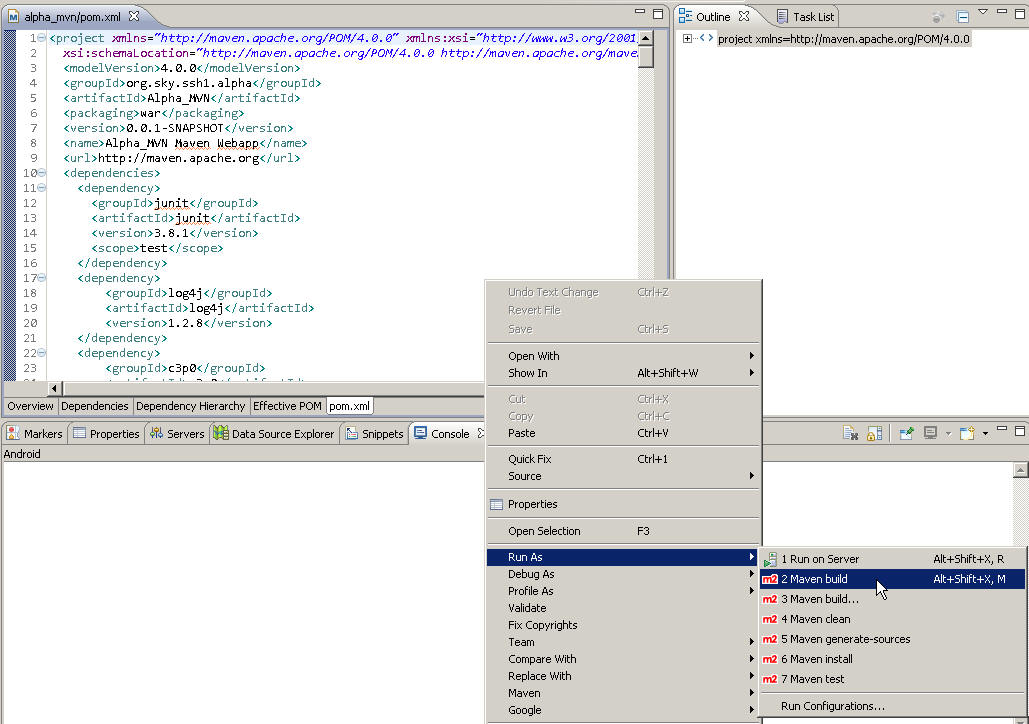
直接把原有alpha工程下的WebContent目录下所有的内容（**除去WEB-INF\lib**）copy到alpha\_mvn的src/main/webapp目录下即可

### 3.3.5 使用maven打包

右键单击pom.xml文件选择Maven POM Editor打开

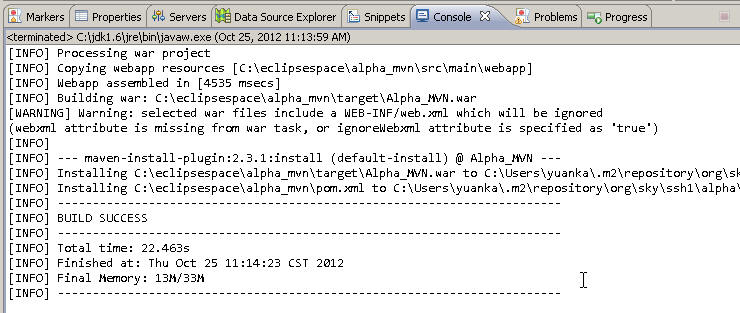


在Maven POM Editor的视图中切换到pom.xml这个tab，然后它会打开这个pom.xml文件的真实内容，右键选Run As->**Maven install**



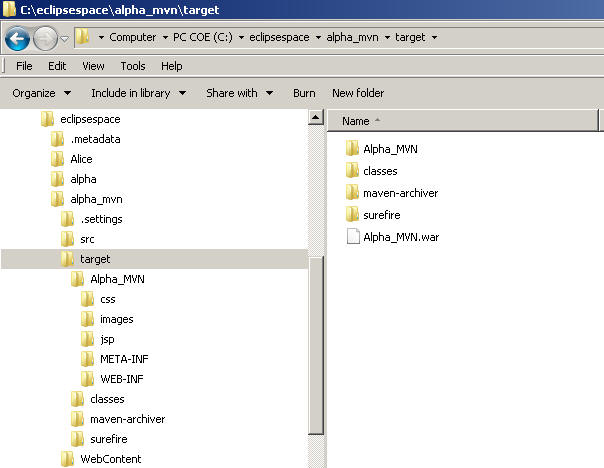
等一会，现在maven开始compile和create标准的J2EE的war工程了.

当我们看到如下输出：



代表工程打包完毕了，如果是fail或者是其它错误，可以先Run As->Maven clean一下，再调整一下你的pom.xml或者是settings.xml文件，然后再Maven install。

Maven会在你的工程所在的目录生成一个target目录，以下是该目录内容：



这个**Alpha\_MVN.war**就是我们的工程布署包，那么有些人不喜欢.war包，**喜欢用打碎掉的war目录结构**，**那么注意Alpha\_MVN这个目录**,就是一个标准的war格式的目录，这两个东西都可以直接扔到tomcat的webapp目录下进行布署，布署完后

**此时我们启动Tomcat。。。！**

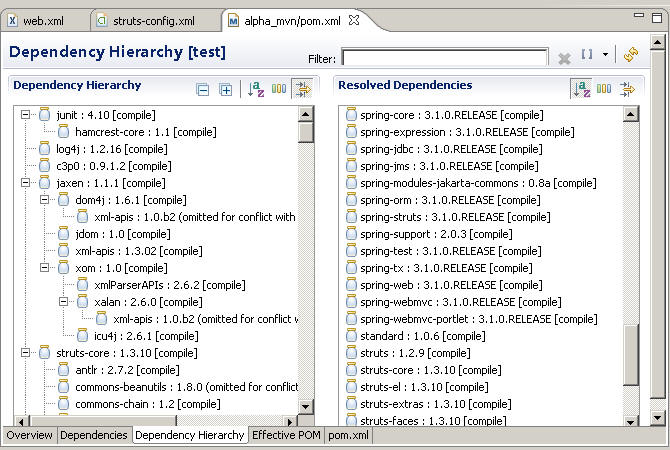
**然后我们等着激动人心的到来。。。！**

**此时我们看到tomcat停了一会，然后过了半天抛了一个java.net.ConnectException Service Unavilable的错误。。。oh…shit!!!!**

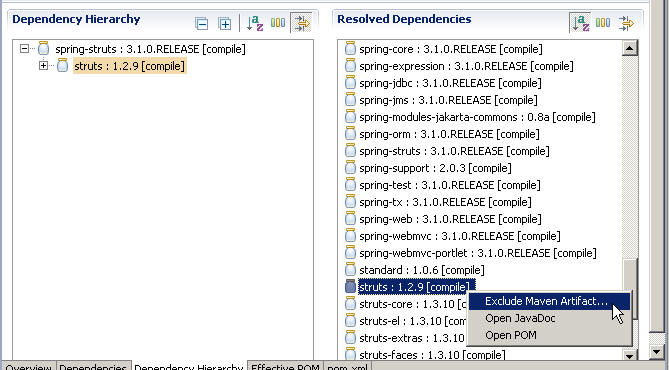
为啥啦。。。别急 ，别急.

解决包冲突，前面说了，Maven在绝大多数情况下会保证你的保依赖关系没有问题，都会帮你下到本地的，但是这次我们就碰到了一个问题。

用Maven POM Editor打开我们的pom.xml文件，切换到：Dependency Hierarchy视图



我们可以看到为了满足spring-struts这个插件的应用（Spring + Struts时用的），Maven多给我们拿 一个struts包，因此导致我们的工程中有两个struts核心包，一个是1.2.9, 一个是1.3.10，由于struts1.3与struts1.2命名空间的不同，因此才产生了刚才那个狗P错误。

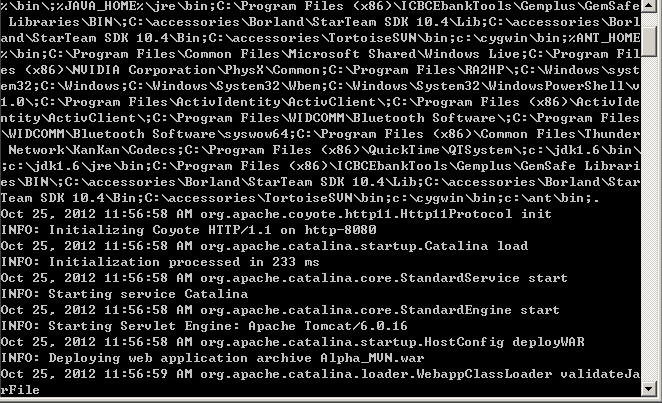


右键单击struts:1.2.9[compile]，选Exclude Maven Artifact后确定并保存pom.xml文件，这步操作相当于我们在原有的pom.xml文件中增加了这样的一段描述：

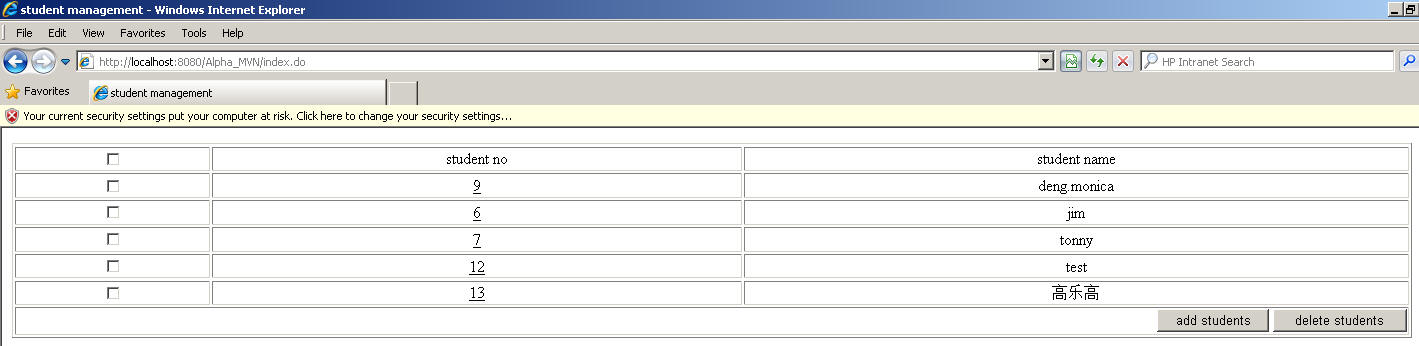
|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-struts</artifactId>  <version>3.1.0.RELEASE</version>  **<exclusions>**  **<exclusion>**  **<artifactId>struts</artifactId>**  **<groupId>struts</groupId>**  **</exclusion>**  **</exclusions>**  </dependency> |

重新Maven clean一下并Maven install一个新的.war文件再布署吧。

Tomcat启动正常



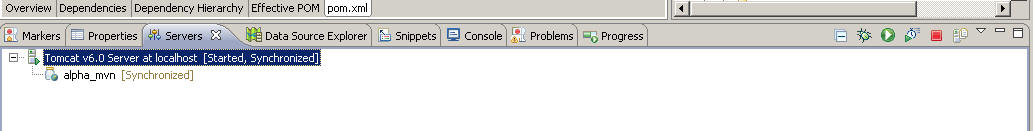
功能测试一切正常(<http://localhost:8080/Alpha_MVN/index.do>)。



# 四、如何让Maven构建的工程在eclipse里跑起来

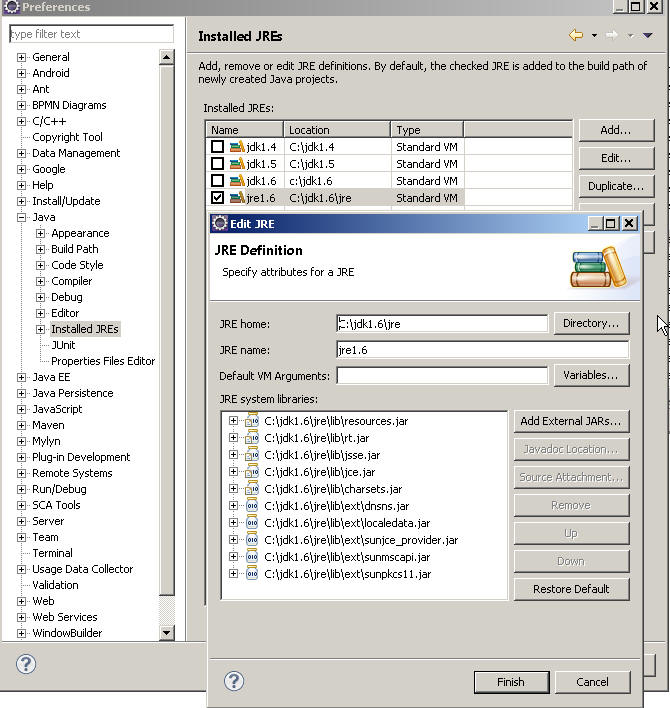
我们刚才利用Maven构建了一个web工程，这个 .war文件才26mb，我们原来的alpha工程加lib库要36-37mb，是不是一下缩水了10多mb啊？因为maven帮我们控制好了所必需的jar，不需要的它不会下载。

但是，这个工程无法在eclipse所内嵌的tomcat里运行起来，这不便于我们的调试。一般我们开发人员都喜欢直接在eclipse里点一下server的运行，然后在eclipse里启动起我们的web工程，这样出了错也便于我们调试，而不用每次改一个jsp或者一个java就重新构建，对吧。

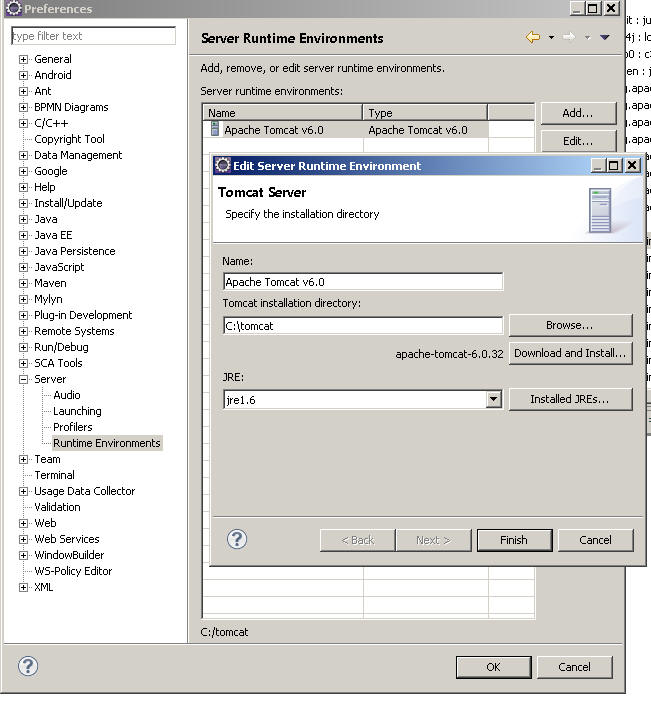


## 4.1 在eclipse里新增一个jdk的run time

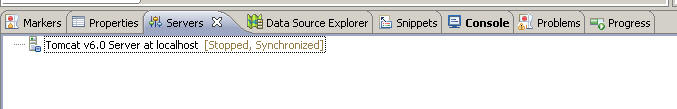
我们在这边设置的jdk把它的JRE home给定位到了JAVA\_HOME\jre目录下了，而不像上面的JDK1.6我们的JRE home是定位到JAVA\_HOME这一层目录的，为什么？**因为在使用jasypt时，eclipse wtp有一个bug就是在eclipse内运行tomcat时，有时会认不出PBEWITHMD5ANDDES这个加密算法**，**它会抛一个secret key not available的Exception，而你如果把你工程的JDK编译环境定位到了JAVA\_HOME\jre目录下，它就能认得出**，这是一个BUG，在eclipse的官方论坛中已经有提。



## 4.2 在eclipse里设置tomcat

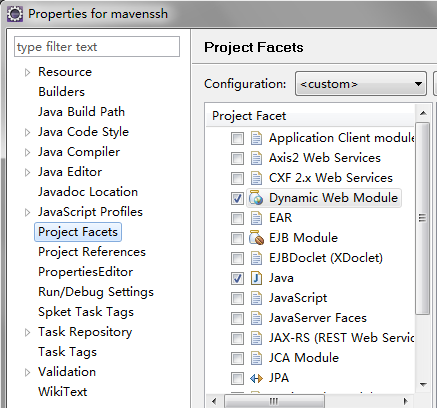


设完后你的工程会在server这个tab下多出一个Tomcat的项。



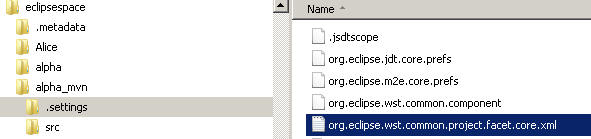
## 4.3 将alpha\_mvn转变成可在eclipse里运行的工程

右键你的工程，选project fact，你会看到一个convert的超链接,点这个超链接。



勾选Dynamic Web Module与Java两项

打开工程所在的eclipse workspace目录下的.settings目录如：C:\eclipsespace\alpha\_mvn\.settings，看到一个叫“org.eclipse.wst.common.project.facet.core.xml”的文件如：

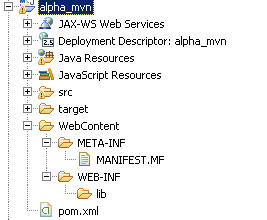


用纯文本编辑工具打开它，把：

installed facet="jst.web" version从3.0或者其它版本，改成 **2.5**如下：

|  |
| --- |
| <installed facet="jst.web" version="**2.5**"/> |

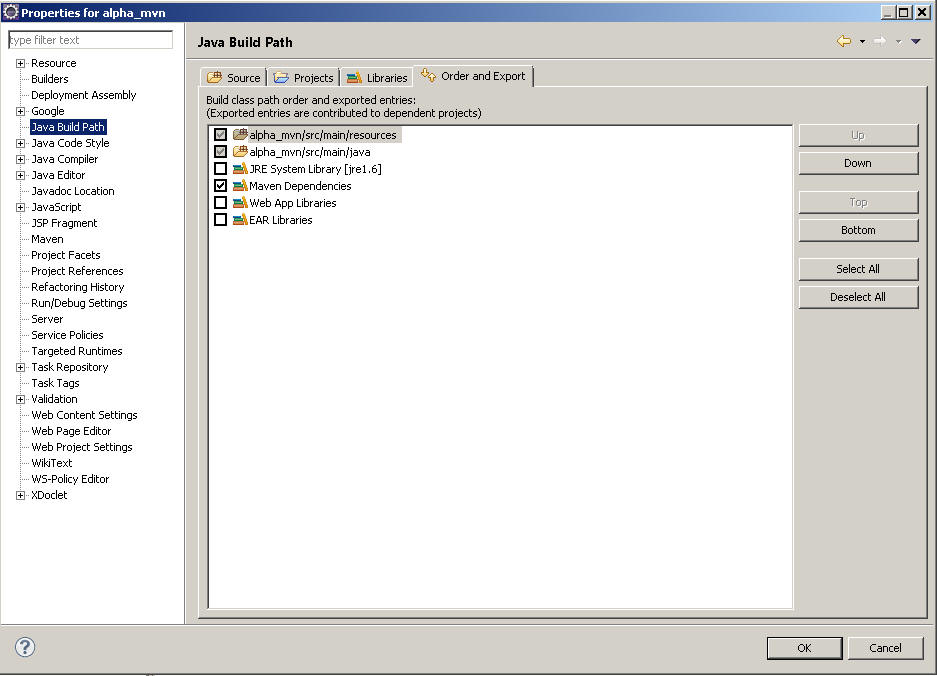
回到eclipse里刷新工程，这时你的工程应该会多出一个目录，WebContent，如下结构：



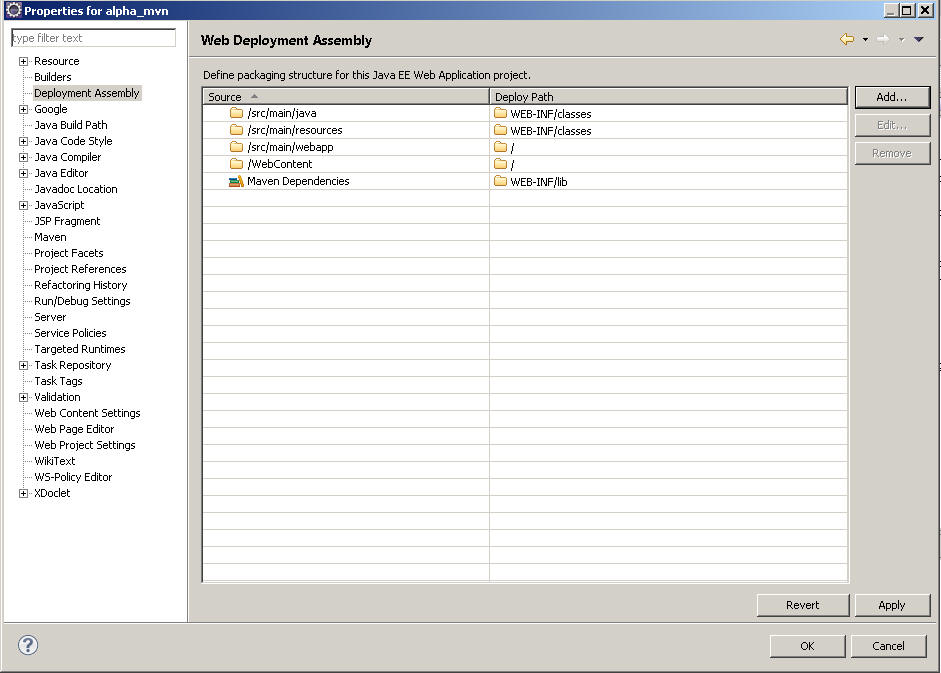
**等等，等等。。。先别急着给我拷东西。**

右键单击工程，选properties，在Java Build Path里选Order and Export这个Tab。

然后把Maven Dependencies给勾选上，按确定。



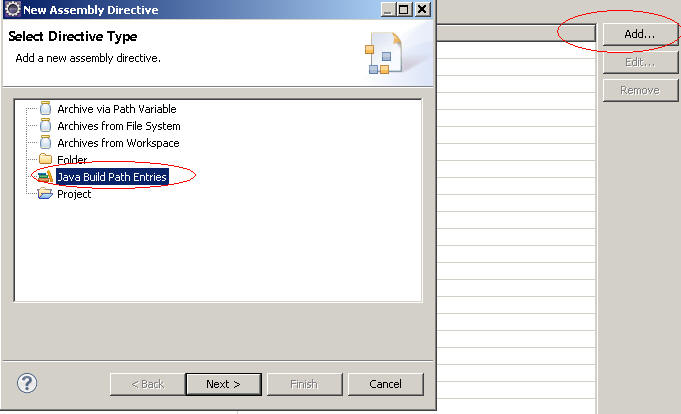
再右键单击工程选



看着这个mapping关系，自己用Add按钮照图来增加吧

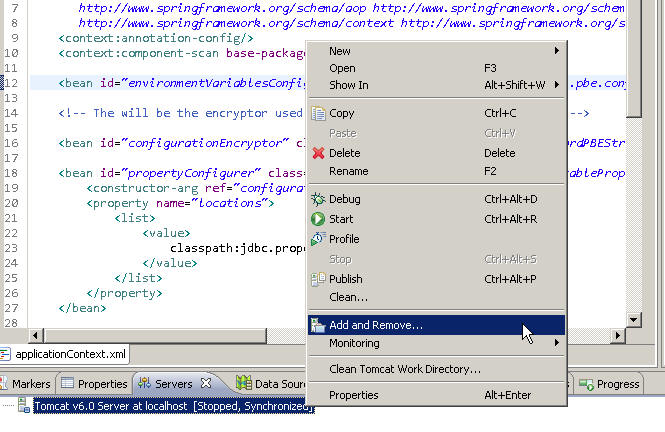
**记住：**

**在Add Maven Dependencies时要选Java Build Path Entries。**

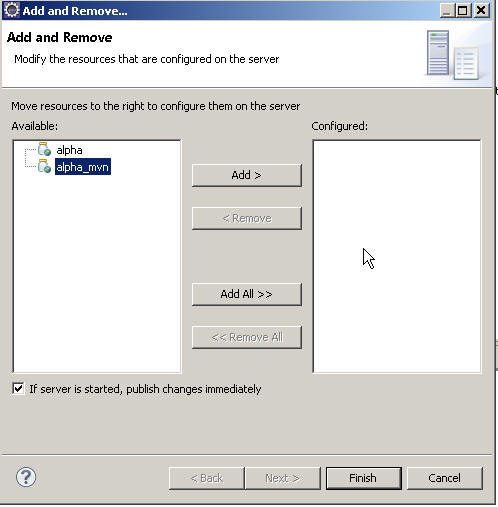


其它的都用Folder来add，完全按照这个mapping 关系来做。

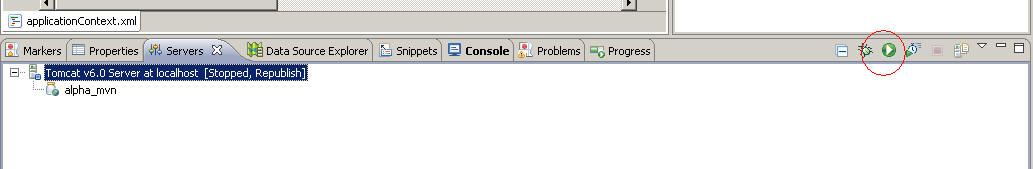
做完后，右键单击server，选”Add and Remove…”



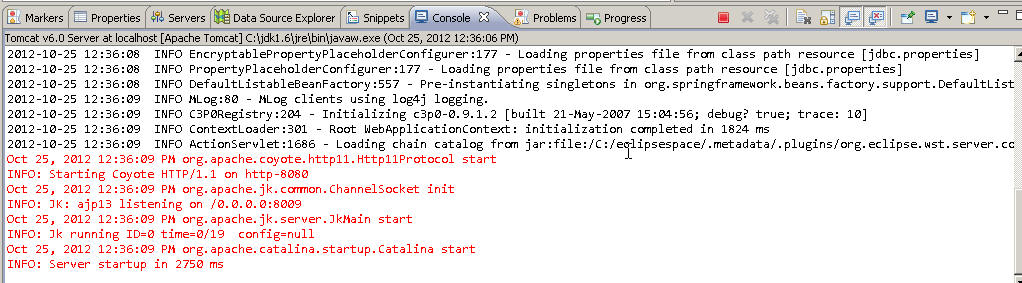
在弹出对话框中把alpha\_mvn通过Add>按钮增加到右边的Configured框中，Finish后



你会发觉原来的servers下的Tomcat已经挂载了我们的工程，点右上角绿色的启动按钮，切换到Console窗口看输出。



输出无误



访问 <http://localhost:8080/alpha_mvn/index.do>测试功能一切正常，结束教程.

今后我们的ssh, ssh2, ssi等工程将基于这个maven来构建，我就不需要为大家再提供jar了，而只是一个pom.xml文件了，真方便（感情着我就是为了图自己发布工程方便啊，呵呵）。

# 附录、Oracle Sun JDK安全限制

经常大家会碰到在使用一些JAVA安全时系统抛出一些not support the algorithm，secret key not available的问题，或者说是DES只支持到512位，不支持1024位的问题。

**其根本原因在于大家在网上下载的jdk都是“安全受限”的，这是美国数字产品出口限制的一个政策。**

因此为了使用更多的算法更长位数的算法，大家需要去Oracle网站下载一个叫**jce\_policy-6.zip**的文件，这就像一个补丁一样，需要把这个补丁打入原有的已安装jdk里。当然，对应于jdk1.5, jdk1.4也有相应的jce\_policy-4, jce\_policy-5的相关文件。

下载后解压后到一个目录中，把这个目录中所有的东西放到你的：

JAVA\_HOME/jre/lib/security目录下，并选择覆盖，重启eclipse或者重新开一个command窗口就可以使用如：PBEWITHMD5ANDDES、Blowfix、rsa1024位以上的一些高层算法了。

下面是jce\_policy-6.zip里的readme的相关安装说明，供参考:

|  |
| --- |
| ----------------------------------------------------------------------  Installation  ----------------------------------------------------------------------  Notes:  o Unix (Solaris/Linux) and Win32 use different pathname separators, so  please use the appropriate one ("\", "/") for your  environment.  o <java-home> refers to the directory where the Java SE Runtime  Environment (JRE) was installed. It is determined based on whether  you are running JCE on a JRE with or without the JDK installed. The  JDK contains the JRE, but at a different level in the file  hierarchy. For example, if the JDK is installed in  /home/user1/jdk1.6.0 on Unix or in C:\jdk1.6.0 on Win32, then  <java-home> is  /home/user1/jdk1.6.0/jre [Unix]  C:\jdk1.6.0\jre [Win32]  If on the other hand the JRE is installed in /home/user1/jre1.6.0  on Unix or in C:\jre1.6.0 on Win32, and the JDK is not  installed, then <java-home> is  /home/user1/jre1.6.0 [Unix]  C:\jre1.6.0 [Win32]  o On Win32, for each JDK installation, there may be an additional  JRE installed under the "Program Files" directory. Please make  sure that you install the unlimited strength policy JAR files  for all JREs that you plan to use.  Here are the installation instruction:  1) Download the unlimited strength JCE policy files.  2) Uncompress and extract the downloaded file.  This will create a subdirectory called jce.  This directory contains the following files:  README.txt This file  COPYRIGHT.html Copyright information  local\_policy.jar Unlimited strength local policy file  US\_export\_policy.jar Unlimited strength US export policy file  3) Install the unlimited strength policy JAR files.  To utilize the encryption/decryption functionalities of  the JCE framework without any limitation, first make a copy of  the original JCE policy files (US\_export\_policy.jar and  local\_policy.jar in the standard place for JCE  jurisdiction policy JAR files) in case you later decide  to revert to these "strong" versions. Then replace the strong  policy files with the unlimited strength versions extracted in the  previous step.  The standard place for JCE jurisdiction policy JAR files is:  <java-home>/lib/security [Unix]  <java-home>\lib\security [Win32] |