Maven使用教程  
一、Maven介绍

我们在开发项目的过程中，会使用一些开源框架、第三方的工具等等，这些都是以jar包的方式被项目所引用，并且有些jar包还会依赖其他的jar包，我们同样需要添加到项目中，所有这些相关的jar包都会作为项目的依赖。通常，一个Java EE项目所依赖的jar包会有很多。然而，这还并非是主要问题，在管理这些jar包过程中，jar包的版本往往是最令人头疼的问题。选择一个jar包的版本，需要考虑它所依赖的jar包是否支持这个版本，并且还需要确认依赖它的jar包能不能对这个版本兼容。所以，在过去的构建项目过程中，整理出这么一堆能让项目正常运行的jar包之后，这个lib目录就会变为禁区。jar包版本更新了，我们也很少会自找麻烦去触碰它。至于是不是存在冗余的jar包？能正常运行就好了嘛。。。。  
Maven的出现，解决了开发过程中的这一难题。它可以对项目依赖的jar包进行管理，可以让你的项目保持基本的依赖，排除冗余jar包，并且可以让你非常轻松的对依赖的jar包进行版本升级。而这些仅仅是Maven最基本的功能，它可以在这基础上对项目进行清理、编译、测试、打包、发布等等构建项目的工作。  
   
可以说，Maven是现在Java社区中最强大的项目管理和项目构建工具，而更加值得庆幸的是，这样一个强大的工具，它的使用也是非常简单的。  
   
现在，JavaEE项目使用的开源软件都可以通过Maven来获取，并且，越来越多的公司也开始使用Maven来管理构建项目了。

二、Maven下载安装

1、配置Java运行环境  
Maven依赖Java运行环境，因此使用Maven之前需要配置Java的运行环境。下载并安装JDK，配置JDK的环境变量：  
    JAVA\_HOME=D:DevToolJavajdk1.6.0\_38  
    在path中加入%JAVA\_HOME%in;  
2、安装Maven  
Maven下载地址：http:*maven.apache.org/download.cgi  
下载Maven最新版本的二进制zip压缩包就可以，如:apache-maven-3.0.5-bin.zip  
下载完成后，解压，例如我们把解压后的文件夹放在D:DevTool目录下。  
然后，将Maven的bin目录添加到path环境变量中，我们这里就是这个目录：D:DevToolapache-maven-3.0.5in  
在命令提示符下，输入mvn -v测试一下，*

三、Maven依赖管理

前面讲了，Maven最核心的就是对依赖jar包的管理，那么它是通过什么方式来进行管理的呢？  
Maven要求每一个jar包都必须明确定义自己的坐标，Maven就是通过这个坐标来查找管理这些jar包的。  
在Maven中，一个jar包的坐标是由它的groupId、artifactId、version这些元素来定义的。例如：  
    <groupId>org.apache.struts</groupId>  
    <artifactId>struts2-core</artifactId>  
    <version>2.3.8</version>  
    <packaging>jar</packaging>  
groupId：表明其所属组织或公司及其所属项目，命名规则为组织或公司域名反转加项目名称。  
artifactId：项目的模块名，通常与实际项目名称一致。模块的命名通常为项目名前缀加模块名。  
version：当前项目的版本号。  
packaging：定义项目的打包方式，可选值有jar、war、pom。默认为jar  
注：：一个组织或公司都会有很多的项目，而每个项目下都会划分多个模块，在开发中我们可以灵活选择依赖某个模块。而Maven管理的jar包基本都是模块性质的项目构建出的jar包。所以，artifactId通常都是模块名，而不是项目名称。项目名称是和组织名称组合作为groupId来使用的。

Maven的配置文件中dependencies元素包含了所有依赖的jar包，每一个jar包依赖使用dependency元素定义。  
在声明一个jar包依赖时，除了指定groupId、artifactId、version这三项基本坐标外，还可以使用使用以下元素进行配置：  
scope元素：指定依赖的范围  
exclusions元素：排除传递性依赖  
   
Maven有以下几种依赖范围：  
compile：编译依赖范围(默认值），依赖在编译、测试、运行期间都有效。  
test：测试依赖范围，只对测试的classpath有效，在编译或运行时无法使用指定为test的依赖包。  
provided：已提供的依赖范围，只对编译和测试的classpath有效，运行期间不会使用这个依赖。例如servlet-api，在运行时容器已经提供，不需要再重复引入。  
runtime：运行时依赖范围，编译时无效，只在测试和运行时使用这个依赖。  
system：系统依赖范围，和provided范围一致，但是provided是使用容器提供依赖，system是使用系统提供依赖，需要指定依赖文件路径。  
   
传递性依赖，是指依赖包对其他包的依赖，比如，我们依赖struts2-core，而strtus2-core需要依赖xwork-core、ognl等，Maven会将这些传递性依赖同时引入项目之中。这也是Maven的一大优点，简化了我们对jar包依赖的管理。而有时我们希望替换某个传递性依赖时，就需要使用exclusions排除掉这个传递性依赖，然后再添加我们自己要替换的依赖包。

四：mvn常用命令

Maven常用命令：

1. 创建Maven的普通java项目：   
      mvn archetype:create   
      -DgroupId=packageName   
      -DartifactId=projectName    
      如：mvn archetype:create  -DgroupId=com.test.mvn -DartifactId=testmvn   
   2. 创建Maven的Web项目：     
       mvn archetype:create   
       -DgroupId=packageName      
       -DartifactId=webappName   
       -DarchetypeArtifactId=maven-archetype-webapp      
         
       如：   mvn archetype:create -DgroupId=com.test.mvn    -DartifactId=webappName -DarchetypeArtifactId=maven-archetype-webapp  
   3. 编译源代码： mvn compile   
   4. 编译测试代码：mvn test-compile      
   5. 运行测试：mvn test     
   6. 产生site：mvn site     
   7. 打包：mvn package     
   8. 在本地Repository中安装jar：mvn install   
   9. 清除产生的项目：mvn clean     
   10. 生成eclipse项目：mvn eclipse:eclipse
   1. 生成idea项目：mvn idea:idea    
      12. 组合使用goal命令，如只打包不测试：mvn -Dtest package     
      13. 编译测试的内容：mvn test-compile    
      14. 只打jar包: mvn jar:jar    
      15. 只测试而不编译，也不测试编译：mvn test -skipping compile -skipping test-compile   
            ( -skipping 的灵活运用，当然也可以用于其他组合命令)    
      16. 清除eclipse的一些系统设置:mvn eclipse:clean

ps：

一般使用情况是这样，首先通过cvs或svn下载代码到本机，然后执行mvn eclipse:eclipse生成ecllipse项目文件，然后导入到eclipse就行了；修改代码后执行mvn compile或mvn test检验，也可以下载eclipse的maven插件。

mvn -version/-v               显示版本信息   
mvn archetype:generate        创建mvn项目   
mvn archetype:create -DgroupId=com.oreilly -DartifactId=my-app   创建mvn项目

mvn package              生成target目录，编译、测试代码，生成测试报告，生成jar/war文件   
mvn jetty:run            运行项目于jetty上,   
mvn compile              编译   
mvn test                 编译并测试   
mvn clean                清空生成的文件   
mvn site                 生成项目相关信息的网站   
mvn -Dwtpversion=1.0 eclipse:eclipse        生成Wtp插件的Web项目   
mvn -Dwtpversion=1.0 eclipse:clean          清除Eclipse项目的配置信息(Web项目)   
mvn eclipse:eclipse                         将项目转化为Eclipse项目

mvn deploy:deploy-file -DgroupId=com -DartifactId=client -Dversion=0.1.0 -Dpackaging=jar -Dfile=d:client-0.1.0.jar -DrepositoryId=maven-repository-inner -Durl=<ftp://xxxxxxx/opt/maven/repository/>

发布第三方Jar到本地库中：

mvn install:install-file -DgroupId=com -DartifactId=client -Dversion=0.1.0 -Dpackaging=jar -Dfile=d:client-0.1.0.jar

-DdownloadSources=true

-DdownloadJavadocs=true

mvn -e              显示详细错误 信息.   
mvn validate        验证工程是否正确，所有需要的资源是否可用。   
mvn test-compile    编译项目测试代码。 。   
mvn integration-test     在集成测试可以运行的环境中处理和发布包。   
mvn verify               运行任何检查，验证包是否有效且达到质量标准。       
mvn generate-sources     产生应用需要的任何额外的源代码，如xdoclet。

常用命令：   
mvn -v 显示版本   
mvn help:describe -Dplugin=help 使用 help 插件的  describe 目标来输出 Maven Help 插件的信息。   
mvn help:describe -Dplugin=help -Dfull 使用Help 插件输出完整的带有参数的目标列   
mvn help:describe -Dplugin=compiler -Dmojo=compile -Dfull 获取单个目标的信息,设置  mojo 参数和  plugin 参数。此命令列出了Compiler 插件的compile 目标的所有信息   
mvn help:describe -Dplugin=exec -Dfull 列出所有 Maven Exec 插件可用的目标   
mvn help:effective-pom 看这个“有效的 (effective)”POM，它暴露了 Maven的默认设置

mvn archetype:create -DgroupId=org.sonatype.mavenbook.ch03 -DartifactId=simple -DpackageName=org.sonatype.mavenbook 创建Maven的普通java项目，在命令行使用Maven Archetype 插件   
mvn exec:java -Dexec.mainClass=org.sonatype.mavenbook.weather.Main Exec 插件让我们能够在不往 classpath 载入适当的依赖的情况下，运行这个程序   
mvn dependency:resolve 打印出已解决依赖的列表   
mvn dependency:tree 打印整个依赖树

mvn install -X 想要查看完整的依赖踪迹，包含那些因为冲突或者其它原因而被拒绝引入的构件，打开 Maven 的调试标记运行   
mvn install -Dmaven.test.skip=true 给任何目标添加maven.test.skip 属性就能跳过测试  
mvn install assembly:assembly 构建装配Maven Assembly 插件是一个用来创建你应用程序特有分发包的插件

mvn jetty:run     调用 Jetty 插件的 Run 目标在 Jetty Servlet 容器中启动 web 应用   
mvn compile       编译你的项目   
mvn clean install 删除再编译

mvn hibernate3:hbm2ddl 使用 Hibernate3 插件构造数据库