# Maven

## 一、基本概念

maven：行家、专家

Maven是一个基于项目对象模型(POM)的**项目管理工具**。

Maven采用了**POM(Project Object Model项目对象模型)**来管理项目，所有的配置信息都被定义在了一个pom.xml文件中，通过该文件Maven可以管理项目的整个生命周期，包括**清除(clean)、编译(compile)、测试(test)、报告、打包(package)、部署**等。

maven的作用：可以构建工厂、管理jar包、编译代码、自动进行单元测试、打包项目、部署项目等。

①当项目中jar包过多时，可能会产生jar包冲突、版本冲突等，而Maven则可以解决这些问题。

②Maven同其它集成开发工具一样，也可以编译java文件

③可以将项目中指定目录下的单元测试一次性完成。

④可以将项目打包。

### 1.Maven的安装

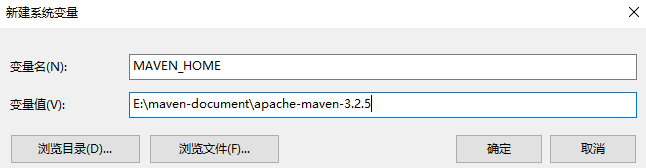
注意Maven是一个基于 Java 的工具，所以系统中首先要有JDK。

**步骤：**

**①**从网站上下载Maven安装包，解压并安装。自己的安装到E:\maven-document中了。

**②**配置环境变量：

右键电脑图标-->属性-->高级系统设置-->环境变量。在系统变量中，点击新建一个系统变量，变量名为MAVEN\_HOME，变量值为：自己的maven安装目录，然后点击确定。



在系统变量中找到Path，然后点击编辑，进入编辑环境变量窗口，点击新建，输入%MAVEN\_HOME%\bin，点击确定即可。

**③**win+R进入cmd窗口，输入**mvn –version**，出现maven和jdk的版本信息则表示安装成功。

在cmd窗口输入**mvn –help:sytem**命令，则会下载help插件并运行它，且打印出Java系统属性和环境变量。

注意：原本配置了maven的环境变量，在cmd直接运行mvn –v有效果，过段时间再直接在cmd中运行mvn –v显示mvn不是内部或外部命令，也不是可运行的程序或批处理文件。但是以管理员身份打开cmd，然后运行mvn –v则可以正常查看，若这样觉得麻烦，可以采用如下方法解决：



**Maven安装之后的目录结构：**

bin：存放Maven的运行脚本

boot：Maven自己的类装载器

conf：配置文件。包含了一个全局行为定制文件settings.xml。

lib：Maven运行时所需要的类库。

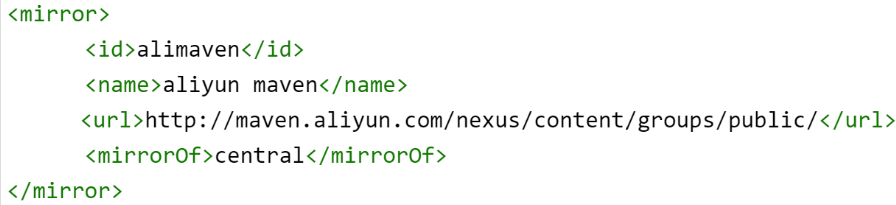
### 2.配置settings.xml

在安装的maven目录中的conf目录中找到settings.xml文件。

**更改本地仓库的位置**：**<localRepository>**中修改成指定位置，默认地址是系统盘的用户文件夹下的一个.m2/Repository。



**配置镜像**：使用阿里云的镜像。[配置mave镜像地址](#_七、配置maven中央仓库的镜像地址)



### 3.Maven常用的命令

以mvn开头的常用命令

**mvn –version**：查看版本信息

**mvn clean**：清理

**mvn compile**：编译

**mvn test**：测试

**mvn install**：安装

**mvn package**：打包

**mvn deploy**：发布

**Maven的其它命令**

mvn site：用于创建项目的文档。

mvn archetype:generate 用于创建项目

**mvn dependency:list** 打印出依赖列表

**mvn dependency:tree**  打印出依赖树

**mvn dependency:analyze** 分析当前依赖

## 二、POM文件

POM是工程对象模型，是一个xml文件，它被放在工程的根目录下，文件命名为pom.xml，**它是maven的核心配置文件**。

 POM 中包含了关于工程和各种配置细节的信息，Maven 使用这些信息构建工程。当执行一个任务或者目标时，Maven 会查找当前目录下的 POM，从其中读取所需要的配置信息，然后执行目标。

### 1.<project>根节点下的子节点

注意每个工程都应该只有一个POM文件。pom文件的根节点是**<project>**。<project>下的子节点有如下：

**①<groupId>**：这是工程组的标识。它在一个组织或项目中通常是唯一的。如一个银行组织拥有所有与银行相关的项目。命名方式一般是公司网址的反写，com+公司名称+项目名称的方式。

**②<artifactId>**：这是工程的标识。通常是工程的名称。groupId和artifactId一起定义了artifact在仓库中的位置。命名方式一般是项目名+模块名。这里的artifact意为项目、产品。

**③<version>**：这是工程的版本号。在artifact的仓库中用来区分不同的版本。该命名方式是版本号+版本类型。如“1.0-SNAPSHOT”，1.0为版本号，SNAPSHOT则标识版本类型为快照型。

**④<modelVersion>**：表示pom.xml使用的是哪个版本的项目对象模型。

**⑤<packaging>**：打包类型，一般是jar、war、ear等。

**⑥<name>**：项目的显示名。常用于maven生成的文档。

**⑦<url>**：组织的站点。常用于maven生成的文档。

**⑧<description>**：项目描述。常用于maven生成的文档。

**⑨<dependencies>**：用来添加依赖的。其下有<dependency>子节点。

**⑩<parent>**：模型的继承。

**11.<dependencyManagement>**：依赖管理。

**12.<reporting>**：创建报告。

**13.<build>**：构建

**14.<repositories>**：引入第三方库。

**15.<licenses>**：许可。

### 2.依赖的配置

当一个maven项目需要使用到的外部的jar包，比如jdbc加载驱动的jar包等，不能直接在该项目的src目录下建一个lib文件夹然后放入jar包，这样是不起作用的。解决办法，在这个maven项目中的pom.xml文件中进行配置依赖。在<dependencies>标签内配置如下：

**一个java构建(artifact)的五大坐标元素**：**groupId、artifactId、version、packaging**(**包类型，如jar、pom、war，就是工程发布时的类型**)**、classifier**(分类)。其中groupId、artifactId、version三个是必不可少的，是依赖的基本坐标。其它两个坐标元素在不指定时采用默认值。

**<dependency>**

**<groudId>指定的jar包名</groudId>**

**<artifactId>指定的jar包名</artifactId>**

**<version>对应的版本号</version>**

<scope>作用域</scope>

<classifier></classifier>

<type></type>

<systemPath></systemPath>

<optional></optional>

<exclusions></exclusions>

**</dependency>**

type则是声明引入的依赖的类型，默认为jar，一般是jar、war等类型。

<optional>：标记依赖是否可选。A->B->C，当A依赖于C时则是可设置为可选。

<exclusions>：排除传递依赖。

**<scope>**：依赖的范围。compile、test、runtime、provided、system、import。

<systemPath>：该标签和<scope>system</scope>一起配合使用，<systemPath>标签中一般写本地jar包的路径。

要导入相应的依赖或插件，直接在百度上搜索Maven，进入Maven的官网然后查找相应的依赖或插件，然后复制<dependency>标签中的内容即可。

### 3.依赖配置的scope

scope定义了类包在项目的使用阶段。项目阶段包括： 编译、运行、测试和发布。下面的依赖就是所需要的jar包。

**①compile**：**默认的作用域scope的范围是compile**。表示为当前依赖参与项目的编译、测试和运行阶段，属于强依赖。采用package命令打包之时，也会将依赖打包到包中。

**②test**：表示该依赖仅仅参与测试相关的内容，包括测试代码的编译和执行。如**Junit**。

**③runtime**：表示该依赖无需参与项目的编译，而参与项目的测试和运行阶段，与compile相比只是跳过编译阶段。一般这种类库都是接口与实现相分离的类库，比如**JDBC类库**。JDBC类库在编译之时仅依赖相关的接口，在具体的运行之时，才需要具体的mysql、oracle等等数据的驱动程序。runtime的依赖通常和optional搭配使用。

**④provided**：表示该依赖可以参与编译、测试、运行阶段，与compile等同，但是区别是在打包阶段进行了exclude操作，该依赖不会被打包进去。**缺少的依赖一般由其它工具提供如web容器**。

即把这个项目打成war包时，scope=provided的jar包不会出现在WEB-INFO/lib目录下，而scope=compile的jar包会放到WEB-INFO/lib目录。

**⑤system**：使用上与provided相同，区别在于该依赖不从maven仓库中提取，而是从本地文件系统中提取，它会参照systemPath的属性进行提取依赖，**和systemPath一起配合使用**。

当maven的依赖是来自本地文件系统中，而不是从仓库(本地仓库、私服仓库、中央仓库)中获取时，需要在配置的<systemPath>中指定本地jar包的路径。

**⑥import**：这个是maven2.0.9版本后出的属性，import只能在dependencyManagement的中使用，能解决maven单继承问题，import依赖关系实际上并不参与限制依赖关系的传递性。

**作用域的使用**：**默认情况就使用compile，测试工具Junit则用test，JDBC则用runtime，能够由外部容器提供jar包则用provided，如tomcat，大部分用默认就行**。

maven中有三种classpath，即**编译、测试、运行**。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 依赖范围 | 编译有效 | 测试有效 | 运行有效 | 例子 |
| **compile** | **Y** | **Y** | **Y** | spring-core |
| **test** | N | **Y** | N | Junit |
| **provided** | **Y** | **Y** | N | servlet.api |
| **system** | **Y** | **Y** | N | 本地文件，不从仓库中获取 |
| **runtime** | N | **Y** | **Y** | JDBC驱动 |
| **import** | import导入依赖不会对上面三种起作用 | | | |

上面的**spring-core是在项目的编译、测试、运行阶段都需要被使用的依赖。而servlet.api则是在项目的编译、测试阶段需要被使用的依赖，但是在项目的运行阶段该依赖已经由tomcat提供了，所以不需要再重复的引入了。而Junit只需要再测试阶段使用**。

注意在导入servlet的依赖时，servlet-api.jar包是3.0之前的版本，javax.servlet-api.jar包则是3.0之后的版本。



### 4.依赖的传递

A->B->C->D-X(1.6版本)：A依赖于B，B依赖于C，C依赖于D，D依赖于1.6版本的X。

A->D-X(2.0版本)：A依赖于D，D依赖于2.0版本的X。

则X是A的传递依赖。

Maven 提供一些功能来控制可传递的依赖的程度。

注意：**只有compile范围的依赖可以传递**。

#### ⑴依赖调节

决定当多个手动创建的依赖版本同时出现时，哪个依赖版本将会被使用。

**调节原则：**

**①第一原则**：**路径近者优先原则，即谁离的最近就使用谁的jar包**。

上面2.0版本的X离A更近，就使用2.0版本的X的jar包。

**②第二原则**：**第一声明者优先原则**。

即当路径是一样长时，则由POM声明的依赖顺序决定。

如A->X(2.0)，A->X(1.6)，则谁先声明则就使用哪个的jar包。

#### ⑵依赖管理

直接的指定手动创建的某个依赖版本被使用。

#### ⑶依赖范围

通过为依赖指定范围，这样依赖就只能在指定的阶段有效。

#### ⑷依赖排除

任何可传递的依赖都可以通过 "exclusion" 元素被排除在外。举例说明，A 依赖 B， B 依赖 C，因此 A 可以标记 C 为 “被排除的”。

#### ⑸依赖可选

任何可传递的依赖可以被标记为可选的，通过使用 "optional" 元素。例如：A 依赖 B， B 依赖 C。因此，B 可以标记 C 为可选的， 这样 A 就可以不再使用 C。

**jar包冲突问题**：若项目同时依赖两个子项目的某个包，路径一样但版本号不一致。编译打包后target目录生成了2个不同版本的jar包，运行时候会报错。解决办法是将其中一个版本去掉。

**Maven中依赖的继承**：**Maven中，子项目是可以继承父项目的依赖**。如一个父maven项目中依赖了一个Junit，若子项目中没有依赖Junit，则子项目依然能够使用Junit，因为子项目继承了父项目的Junit依赖。

**常用的与依赖有关的命令：**

**mvn dependency:list** 打印出依赖列表

**mvn dependency:tree**  打印出依赖树

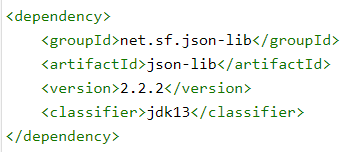
**mvn dependency:analyze** 分析当前依赖

### 5.classifier的使用

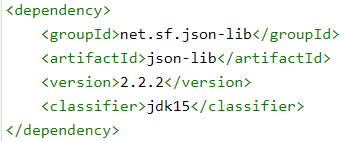
<classifier>是用来分类的。

**使用情形一**：**用于区分基于不同jdk版本的jar包**。

若项目的依赖是json-lib-2.2.2-jdk13.jar包，则在pom.xml中配置内容如下：



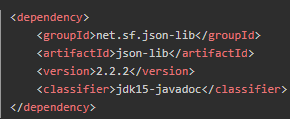
若项目的依赖是json-lib-2.2.2-jdk15.jar包，则在pom.xml中配置内容如下：



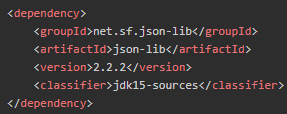
注意，若 json-lib没有提供json-lib-2.2.2.jar。那么，设置依赖的时候，必须使用 classifier ，否则会报错，因为找不到指定的jar包。

**使用情形二**：**区分项目不同的组成部分。如源代码、javadoc、类文件等**。

如需要json-lib-2.2.2-jdk15-javadoc.jar，则在pom.xml中配置内容如下：



如需要json-lib-2.2.2-jdk15-sources.jar，则在pom.xml中配置内容如下：



### 6.父POM

注意，所有得POM都继承自一个父POM(无论是否显示的定义了这个父POM)。父POM也称为Super POM，它包含了一些可以被继承的默认设置。

Maven使用effective pom(Super POM加上工程自己的配置)来执行相关的目标，可帮助开发者在pom.xml中做尽可能少的配置。在有pom.xml文件的目录下，打开cmd，输入**mvn help:effective-pom**即可查看Super POM的默认配置。

### 7.POM包含的三种信息

**POM(项目对象模型)中一般包含三种信息**：

**①**项目自身信息

**②**项目运行所依赖的jar包信息

**③**项目运行环境信息，比如jdk、tomcat信息。

其中项目运行所依赖的jar包信息独立变成一个**依赖管理模型**，依赖管理模型中放置的是jar包的坐标。jar包的坐标信息包括groupId对应的公司组织的名称、artifactId对应的项目名称以及version对应的版本号信息。

依赖管理模型根据jar包的坐标从本地仓库local、远程仓库b2b、中央仓库central中搜索对应的jar包。

maven的默认生命周期(compile、test、package、install、deploy)中每一个构建项目的命令都对应了一个maven底层插件。

maven的两大核心功能是**依赖管理**和**一键构建**。

**maven的依赖管理**：就是maven工程对jar包的管理过程。传统的Web工程是将jar包放到项目中，而maven则是将jar包专门放到仓库中，然后通过工程中的jar包坐标从仓库中找到对应的jar包。

**项目的构建**是指项目从编译、测试、运行、打包、安装、部署整个过程都交给maven进行管理。

**一键构建**是指整个构建过程使用maven的一个命令就可以完成整个的工作。

pom.xml文件：







### 8.DependencyManagement和Dependencies的区别

#### ⑴DependencyManagement

**DependencyManagement是用来统一管理依赖的版本号，只声明依赖，并不实现依赖的引入，因此在子项目中需显示的声明需要使用的依赖而不用显示的列出依赖的版本号**。

**如：**当父项目中用DependencyManagement声明依赖时，其声明的依赖中定义了依赖的版本号，父项目中并没有对依赖进行引入，因此子项目需要继承父项目的依赖时，需要自己显示的声明需要使用的依赖，同时可以不用指定依赖的版本号，而是使用父项目中指定的依赖的版本号。若子项指定了依赖的版本号，则子项目使用自己指定的版本号的依赖。若子项目没有显示的声明需要使用的依赖，则子项目不会从父项目中继承依赖。

**优点：方便对于项目中依赖的版本号的管理**

**①**若多个子项目都使用同一版本的依赖，则可以避免在多个子项目中都声明依赖的版本号。

**②**当想使用依赖的另一个版本时，则无需修改每个子项目中依赖的版本号，只需修改顶层父项目中依赖的版本号。

**③**若单独某个子项目需要使用依赖的别版本号，则只需要在该子项目中显示的声明依赖的版本号即可。

**④**统一管理项目的版本号，确保应用的各个项目的依赖和版本一致，才能保证测试的和发布的是相同的成果。

**例子**：





#### ⑵dependencies

相对于DependencyManagement，**所有声明在dependencies里的依赖都会自动引入，并默认被所有子项目继承。dependencies相当于管理的是依赖的关系，父项目中声明的依赖，子项目只能被动接受**。

当父项目中的依赖直接用dependencies声明，则所有的子项目都会从父项目中继承所有的依赖，即使子项目中并没有声明该依赖。

若依赖只在某个子项目中使用，则直接在子项目中声明所需要的依赖，不要在父项目中声明，否则该依赖也会被其它子项目继承使用。

**例子：**



**父项目中的依赖最好使用DependencyManagement来管理依赖的关系，子项目中需要使用什么依赖显示的声明即可。**

## 三、仓库

**仓库分类**：**本地仓库、远程仓库**。

其中远程仓库又分为：**中央仓库(远程公用仓库)**、**私服仓库**。

**①本地仓库(local)**

在安装maven后并不会创建，它是在第一次执行maven命令的时候才被创建。maven本地仓库的默认位置：在系统盘下的用户目录下的一个.m2/repository/目录中。本地仓库中保存了工程的依赖(就是jar包)。

下面是本地仓库的默认地址：C:\Users\15566\.m2\repository



若要修改本地仓库的位置，则要在maven安装目录中的conf文件夹下的setting.xml中设置：

修改成指定路径：



**②私服仓库(remote)**

是一种特殊的远程仓库，它是架设在局域网内的仓库。一般由公司自己设立，只在本公司内部共享使用。也可以是其它主体设立的，如aliyun的maven仓库。

私服仓库又可分为：**全局应用的私服仓库、应用到项目自身的私服仓库**。

**③中央仓库(central)**

中央仓库是由Maven社区提供的仓库，里面包含了大量的jar包。

这个仓库是由Maven社区管理的，不需要配置，而且需要通过网络才能访问。

**情形一：没有私服仓库的情况**

在默认情况下，当启动一个Web工程时，会先从本地仓库中搜索jar包，若没有找到jar包且在联网的状态下则会从中央仓库搜索jar包，如果找到了则下载到本地仓库中已备将来使用，若没有找到则会报错。

**情形二：有私服仓库的情况**

先从本地仓库中搜索jar包，若没找到则从私服仓库中搜索，若找到了，则下载到本地仓库已备将来继续使用。若没找到，则可以从本地上传，或者从中央仓库中搜索，若找到则下载到本地仓库已备将来继续使用，若没找到则报错。

大致顺序可理解为下面这样：

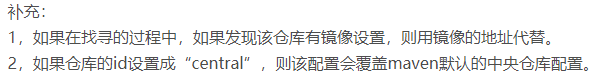
①在本地仓库中寻找，如果没有则进入下一步。

②在全局应用的私服仓库中寻找，如果没有则进入下一步。

③在项目自身的私服仓库中寻找，如果没有则进入下一步。

④在中央仓库中寻找，如果没有找的则报错。

对于下面的有疑问



## 四、maven项目标准目录结构

src/main/java目录：存放核心代码。

src/main/resources目录：存放配置文件。

src/test/java目录：存放测试代码。

src/test/resources目录：存放测试文件

src/main/webapp目录：存放页面资源，如js、css、html、图片等

一个Maven项目，它的根目录下必定存在src文件夹和pom.xml文件。

**mvn clean** 用来清除项目编译信息。

**mvn compile**会将src/main/java目录下的代码进行编译。

**mvn test**会将src/test/java目录下的代码进行编译，同时也会编译src/main/java目录下的代码。

**mvn package** 会将src/main/java目录下的代码和src/test/java目录下的代码进行编译，同时也会将项目打包成指定类型的文件如打包成war包、jar包。

**mvn install**会将src/main/java目录下的代码和src/test/java目录下的代码进行编译，同时也会将项目打包成指定类型的文件如打包成war包、jar包，同时还会将打包好的war包或jar包放到本地仓库中。

**mvn deploy**的执行内容除了和 mvn install操作执行的内容相同之外，还多一个将项目jar包、war包拷贝到私服仓库中，以共享给其它开发人员和工程。

## 五、maven生命周期

maven的生命周期：

**①清理生命周期：**对应clean操作。分为pre-clean、clean、post-clean三阶段。

**②默认生命周期：**compile、test、package、install、deploy等操作构成默认生命周期。

**③**站点生命周期：一般分为pre-site、site、post-site、site-deploy几个阶段。

## 六、maven中的插件

注意**Maven的一切构建操作都是基于插件执行的**。

maven一般有下面两种类型的插件。

**①**Build plugins：在构建时执行，并在pom.xml文件中配置。

**②**Reporting plugins：在网站生成过程中执行，并在pom.xml文件中配置。

**常用插件**：

①**clean**：构建之后清理目标文件。删除目标目录。

②**compiler**：编译java源文件

③surefile：运行JUnit单元测试，创建测试报告。

④**jar**：从当前工程中构建jar文件

⑤**war**：从当前工程中构建war文件

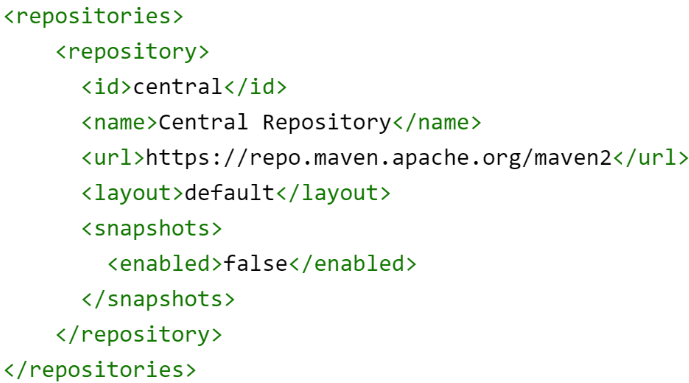
⑥**javadoc**：为工程生成javadoc

⑦antrun：从构建过程的任意一个阶段中运行一个 ant 任务的集合。

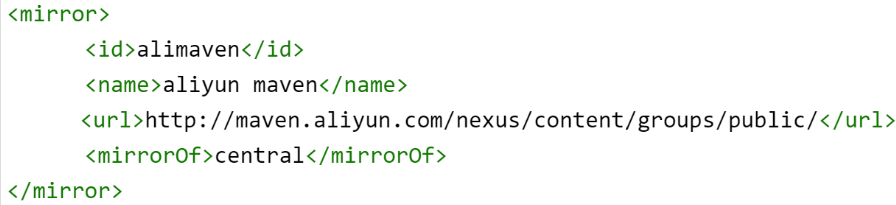
<http://cmsblogs.com/?p=6187>这里的构建配置文件没学

## 七、配置maven中央仓库的镜像地址

在默认的情况下，maven存在一个默认的中央仓库地址，该中央仓库地址如下：



由于在国内直接访问中央仓库的地址网速很慢，因此我们一般使用的是maven的镜像仓库地址，在maven的镜像仓库地址中，网速比较快的。我们使用阿里云的镜像仓库地址，在maven安装目录下的conf目录下的settings.xml文件中的 mirrors标签内配置如下内容：



## 八、maven创建工程

采用maven方式创建项目，每个项目(不管是普通java项目还是web项目)都包含**一个src目录和一个pom.xml文件**。

### 1.在cmd一步一步创建项目

**①**打开cmd进入窗口，采用**cd /d 指定目录**的方式将窗口的目录切换到需要创建项目的文件夹中。输入**mvn archetype:generate**回车，则开始进行项目的创建。

**②**当maven插件下载完之后，进入archetype选择界面中。根据不同的数字选择不同的archetype，默认情况下选择maven-archetype-quickstart，直接输入7回车，接着继续下载相关得依赖。

**③**然后依次根据groupId、artifactId、version、package等输入相应得内容。

**④**根据输入的信息是否有误输入y或者no。当控制台出现Build Success则说明maven项目已经创建完成。则在对应的项目文件夹中可以看到内容。

### 2.cmd中快速创建java项目

**mvn archetype:generate**

**–DgroupId=com.aowin**

**–DartifactId=test\_maven\_project**

**–DarchetypeArtifactId=maven-archetype-quickstart**

**–Dversion=1.0**

–DinteractiveMode=false(这个好像能省略，不知到是干啥的)

进入创建好的项目的目录中，然后在该目录中打开cmd窗口，输入相应的mvn 命令，如mvn package则可以进行项目的打包，然后在该目录中就可以看到生成了一个target文件夹。

普通java项目打包之后好像就是jar包类型。

**注意**以cmd方式采用quickstart骨架生成的java项目，src目录下只有main/java....和test/java...目录，没有对应的resources目录，需要自己手动创建。

### 3.cmd中快速创建web项目

**mvn archetype:generate**

**-DgroupId=com.aowin**

**-DartifactId=web\_maven\_project**

**-DarchetypeArtifactId=maven-archetype-webapp**

**-Dversion=1.0**

–DinteractiveMode=false(这个好像能省略，不知到是干啥的)

web项目打包之后好像就是war包类型的。

**注意**以cmd方式采用webapp骨架生成web项目，src目录下只有main文件夹，没有test有关的文件夹，需要手动创建。

## 九、在maven项目中执行mvn命令

若想对maven项目执行相应的mvn compile、mvn test、mvn package等命令，有多种方式：

**法一**：在cmd窗口执行。

打开项目所在的文件夹，进入cmd窗口，然后输入相应的指令即可。如mvn package，则项目文件夹中会多一个target文件夹，打包好的项目就在这个文件夹中，测试报告则放在target文件下的surefire-reports文件夹中，执行mvn clean则会将这个文件夹清除。

若要运行代码，则要进入target目录下的classes目录中，在这代开cmd窗口，输入**java 类的包名+类型**回车即可运行对应的类的代码。



如果需要将该maven项目转换成Eclipse中的maven项目，可以进入该项目在cmd中使用命令：**mvn eclipse:eclipse**

执行**mvn eclipse:clean**命令则可以清除Eclipse的设置信息。

**法二**：在Eclipse中的执行：[2.Eclipse中创建Maven](#_2.Eclipse中创建Maven)

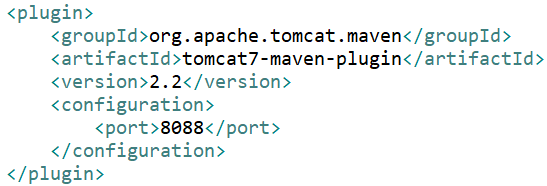
**法三**：在IDEA中执行：

## 十、maven web项目的运行

**法一**：在cmd窗口进入该项目目录中，执行mvn package将该web项目打包成war包，然后将该war包放到tomcat安装目录的webapps目录下，然后在tomcat目录下的bin目录中点击start.bat 运行tomcat，然后tomcat会自动将该war包解压成文件夹。最后在网页上输入地址即可访问该项目。

**法二**：引入tomcat插件运行

在该web项目中的pom.xml中添加maven-tomcat插件，然后执行**mvn tomcat:run**。



注意运行web项目时，使用tomcat:run运行项目时，使用的是maven中默认的tomcat插件，而不是自己的安装的tomcat，若要使用指定的tomcat插件，需要使用田添加特定的maven-tomcat插件。

**法三**：直接在Eclipse中运行。**若是web类型的项目，一定要先执行mvn clean清理项目，然后mvn package将项目打包，然后一定要刷新一下项目，然后run as server选择tomcat运行即可**。**若不刷新项目，可能会出现项目下没有打包好的war包，然直接运行的话，在路径没错的情况下也会报404错误**。即404错误也有可能不是路径出错，也有可能是maven中web项目下对应的lib包下没有jar包(没打包项目，或者打包了但没刷新项目所以lib下没有jar包)。

## 十一、快照(snapshot)

快照是一个特殊的版本，它表示当前开发的一个副本。

对于版本，Maven 一旦下载了指定的版本(例如 data-service:1.0)，它将不会尝试从仓库里再次下载一个新的 1.0 版本。想要下载新的代码，数据服务版本需要被升级到 1.1。 对于快照，每次用户接口团队构建他们的项目时，Maven 将自动获取最新的快照(data-service:1.0-SNAPSHOT)。

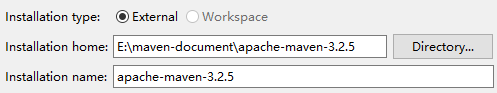
maven会每次自动更新快照，但自己也可以在任何 maven 命令中使用 **-u参数**强制 maven 下载最新的快照。如mvn package –u。

## 十二、在Eclipse中创建Maven项目

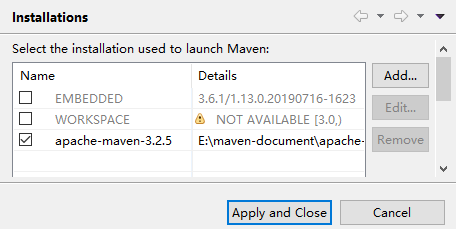
### 1.Eclipse中配置Maven

建议在Eclipse中使用安装的Maven，而不是Eclipse自带的。

**步骤一**：window-->preference-->Maven-->Installs-->Add



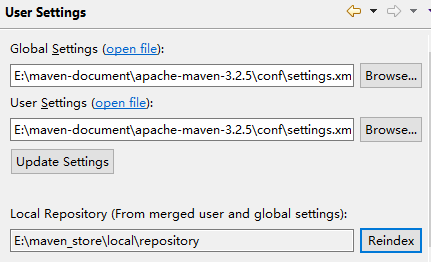
Install home为Maven的安装路径，name为给这个maven工具取得名字，可自定义。之后点finish，然后勾选添加的这个maven，点aplly即可。



**步骤二**：window-->preference-->Maven-->Uer settings

然后在Global settings中选择自己安装的maven工具的conf目录下的settings.xml。下面的User settings也可以同Global settings的一样，或者使用本地仓库文件夹下的settings.xml。

注意，还要点击一下下面的Reindex，使其定位到Global settings中设置的本地仓库中。



**注意**：maven安装包下的conf目录下的settings.xml是为全局范围的配置文件，修改后将影响本机所有用户的配置。而本地仓库目录下的conf目录下的settings.xml则是用户范围的配置文件。一般将maven安装包的settings.xml复制一份放到本地仓库中和repository文件夹同级的目录下，然后只修改本地仓库中的settings.xml，这样只会修改用户级别的配置，既不影响其它用户，也不影响后期升级。

### 2.Eclipse中创建Maven

**创建Maven项目有两种方式**：**不使用骨架创建和使用骨架创建**。**建议普通java项目和web项目都不使用骨架创建，**若要使用骨架创建，则普通java项目选择maven-archetype-quickstart，web项目选择maven-archetype-webapp骨架创建。

一般普通java项目的packaging类型为jar，web项目的packaging则为war，pom项目的packaging类型为pom，一般父工程用pom。

web项目不使用骨架创建，一般会报错，报web.xml缺失的错误，解决办法自己新建web.xml或者右击项目，选择Java EE Tools-->Generate Deployment...即可解决。



#### ⑴创建单个工程

File->new->other->Maven->Maven Project->Next->可勾上create a simple project，这样可以跳过骨架选择，也可不勾上->next->输入对应的groupId、artifactId和选择version、packaging(这是工程发布类型)->点击finish即可。

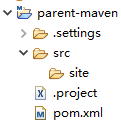
#### ⑵创建父工程、子工程

**Maven Module是Maven Project的子模块**。

##### 法一：先创建父工程，在由父工程创建子工程

**①创建父工程**：创建一个Maven Project类型的工程，**注意父工程的packaging的类型必须为pom类型**。

创建好之后的父工程的目录结构如下：

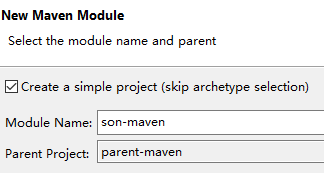


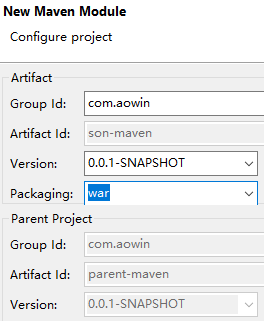
**②创建子工程**：右键父工程-->maven-->New Maven Module。在Module Name中输入子工程的名称，建议勾选跳过使用骨架，然后点击next—>选择packaging包类型jar或者war -->finish。

若选择的类型是jar，则finish之后还要进行下面**③配置web项目**操作。

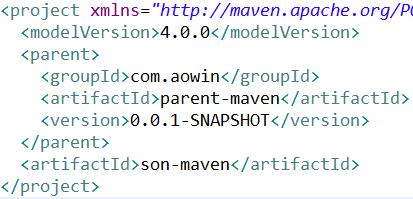
若选择的类型是war，则finish之后不用进行下面的配置web项目操作。但是直接finish之后，若使用的是跳过使用骨架创建的web工程，一般来说会报错，原因是因为没有web.xml文件。解决办法是法一：自己在src/main目录下新建一个WEB-INF文件夹，且建一个web.xml文件，并将一个web.xml的头拷贝进去。法二：右键点击war类型的工程-->Java EE Tools-->Generate Deployment...即可。







创建的子工程的pom.xml文件的内容如下所示：

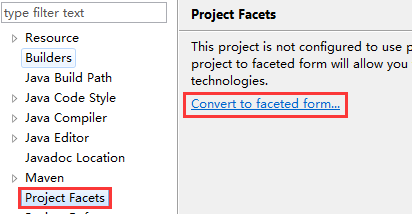


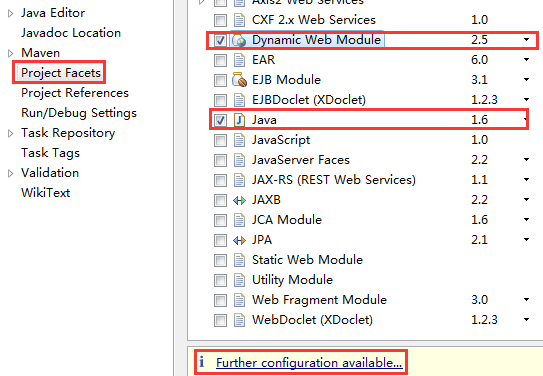
**③配置web项目**：jar类型的子工程才要进行这步。

若需要将项目放置到tomcat等容器中运行，还需要配置web参数。

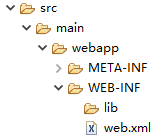
**步骤一**：右击子工程-->Properties-->Project Facets-->点击convert to faceted form-->然后勾选Dynamic web Module，这里可以选择对应的版本，以及选择勾选的java对应的版本-->勾选了Dynamic web Module之后，下方出现Further Configuration available，点击进入-->修改Content Directory的内容，修改为src/main/webapp，并勾选生成web.xml—>点击apply和apply and close。

注意下面这个页面只有当创建子工程的packaging类型为jar时才有下面如图的设置。



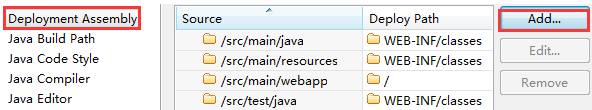


配置完上面的内容之后，在子工程的src目录下会生成相应的web.xml文件，如图：



这时在点击tomcat的add and remove就可以看到该子工程了。

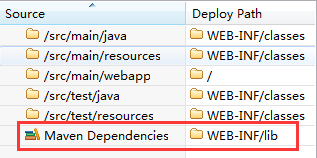
**步骤二**：右击子项目-->Properties-->Deployment Assembly, 配置webapp为项目的根目录(好像不用配置这个，默认就是)，继续点击右边的add按钮，进行添加依赖。-->选中Java Build Path Entries-->点击next-->选中Maven Dependencies，点击finish即可。







配置之后的效果如下：



##### 法二：创建多个独立的工程，然后整合成父子工程

**①分别创建多个独立的工程**。

如4个，parent-maven是表示父工程，daughter-maven和son-maven是表示子工程，other-maven表示子工程的依赖。

**②分别修改这些工程的pom.xml文件**。

**步骤一**：修改父工程的pom文件

父工程的<packaging>的类型类型必须为pom类型，若不是则修改，若没有这个标签则添加。

采用<modules>和<module>标签将所有子工程和子工程的依赖集合起来。

**parent-maven的POM文件：**



**步骤二**：修改子工程的pom文件

在子工程中添加<parent>标签，将父工程的内容引入，注意还要在<relativePath>标签中填写父工程的pom.xml文件的相对路径。

疑问：在路径中是否要加父项目名称?下面哪种写法是正确的。

<relativePath>../pom.xml</relativePath>

<relativePath>../parent-maven/pom.xml</relativePath>

**son-maven的POM文件**：



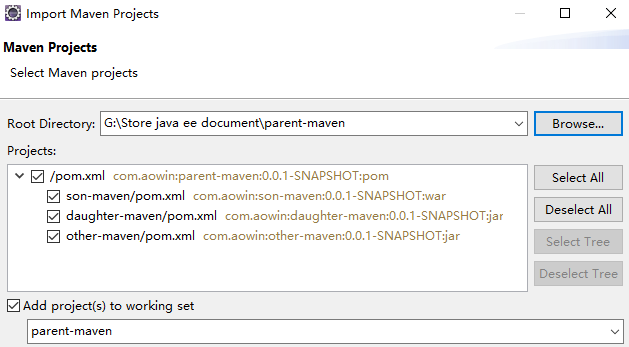
**daughter-maven的POM文件**：



**other-maven的POM文件**：



**步骤三**：将这四个工程从Eclipse中删除(不是真的删除，不能勾选从磁盘中删除，只是从Eclipse工作台上删除)，然后找到这几个工程在磁盘中的位置，将daughter-maven、son-maven、other-maven添加到parent-maven的目录中。然后再Eclipse中点击File-->import-->Maven-->Existing Maven Projects—>导入父工程，当出现如下界面，然后点击finish就完成了。



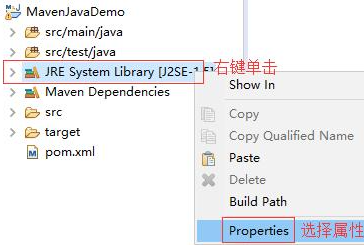
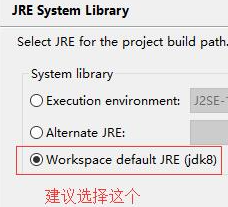
#### ⑶Eclipse中创建web类型的maven工程

不使用骨架创建：创建一个Maven工程，选择不使用骨架，打包类型选择war类型。创建完之后，右键项目🡪properties属性—>Project Facets🡪选择java版本为1.8，将Dynamic Web Module去掉，点击Apply,然后选择相应的Dynamic Web Module，然后点击下面的这个内容 ，进入界面，添加src/main/webapp目录，同时勾选下面的web.xml内容，点击完成。然后右键项目更新项目就不会报错了。

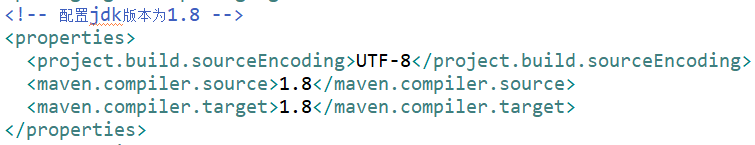
### 3.修改maven的JRE环境

创建完maven项目之后，更改maven项目中使用的JRE环境，默认使用的是jdk1.5。

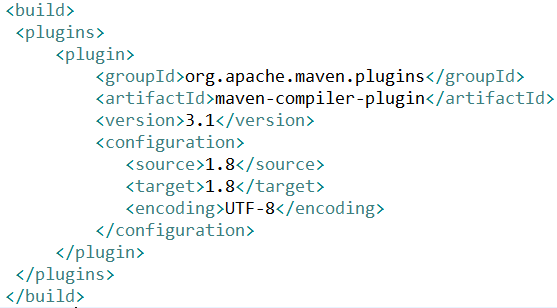
**法一：这种方式更新一下项目则又变回原来的jdk版本**

**法二：**在pom文件中添加如下。

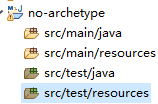


**法三**：在项目的pom.xml文件中添加插件，如下所示：

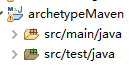


### 4.使用骨架和不使用骨架的区别

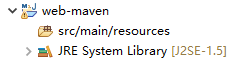
不使用骨架创建普通的java项目，项目的目录是标准的maven项目的目录。如图：



使用骨架maven-archetype-quickstart创建普通的java项目，项目的目录是不标准的，需要添加resources目录。即手动添加src/main/resources和src/test/resources。



使用骨架maven-archetype-webapp创建web项目的目录也是不标准的，如下所示，需要自己手动添加其它目录。



## 十三、依赖版本号的管理

依赖的版本号管理有两种方式：

**法一**：父工程中采用<dependencyManagement>标签统一来管理依赖的版本号。[DependencyManagement管理依赖版本号](#_⑴DependencyManagement)

**法二**：在工程中采用<properties>标签，然后自定义标签统一声明版本号，然后在依赖中使用**${}**进行占位。



## 十四、maven搜索jar包的顺序

Maven允许在多个地方配置仓库的位置，如在pom.xml文件中或在settings.xml文件中。当需要加载一个新的Jar包时，Maven会从配置文件中读取仓库位置，并按照优先级逐一从仓库中判断是否有指定的依赖文件，如果有，则加载，如果没有，则继续搜索。

**搜索jar包的顺序如下**：(未经检验，网上直接搜的)

**①**先从本地仓库中搜索，若没有则往下进行。

**②**从Maven安装包的settings.xml文件中<profile>标签下的<repository>标签中配置的仓库位置中搜索，若没有则往下进行。

**③**从pom.xml中<profile>标签中定义的仓库中查找，若没有则往下进行。

**④**从pom.xml中<repositorys>中的<repository>标签中定义的仓库中查找，若定义了多个则按顺序依次查找。若没有则往下进行。

**⑤**从<mirrors>下的<mirror>中配置的仓库中搜索。注意若<mirror>中配置了，则其它仓库配置的地址会失效，以这个为准。

## 十五、其他补充

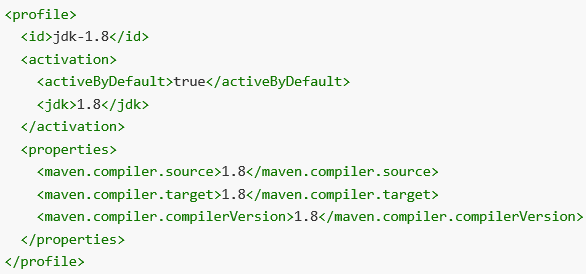
### 1. 解决Maven中存在的资源过滤问题

下面<include>标签中的\*\*表示任意级目录，\*表示某个文件的名。\*\*/\*.xml表示在src/main/java或src/main/resources目录下的任意包中的任意.xml文件。



### 2.maven中设置指定编译的java版本

在maven的settings中设置如下：



### 3.将应用打包成jar包的插件

添加如下插件，可以将应用打成jar包，直接使用java -jar的命令进行执行；

