# 精讲Maven

# 1 什么是Maven

* Maven是基于项目对象模型(POM)，可以通过一小段描述信息来管理项目的构建，报告和文档的软件项目管理工具。
* Maven是跨平台的项目管理工具。主要服务于基于Java平台的项目构建，依赖管理和项目信息管理。
* [相当于iOS cocoapods]
* Maven的好处【 lib】

1. 方便jar的管理
2. 不需要上传jar包到svn

# 2如何使用Maven

## 2.1第一：安装本地Maven

* 先打开浏览器访问maven官网【rz】，下载下图**红色的zip**文件

|  |
| --- |
|  |

* 接着解压文件夹到用户目录下C:\Users\guoyongfeng\中，【这里解压的路径随机，个人喜欢放在**用户目录**而已】
* 然后需要配置path**环境变量**
  + 系统变量：MAVEN\_HOME = C:\Users\guoyongfeng\apache-maven-3.5.0
  + 系统变量：path = %MAVEN\_HOME%\bin
* 最后打开**命令行**，输入mvn -version,出现下面内容表示本地maven安装和配置成功了

|  |
| --- |
|  |

## 2.2第二：了解下Maven的仓库

* 仓库是用来干啥的呢？其实就是用来**存放网上下载到本地的jar包**
* 仓库路径默认情况下在用户目录下C:\Users\guoyongfeng\.m2\repository
* 配置仓库路径 在bin目录下的config/settings.xml

|  |
| --- |
|  |

* 在 cmd 中敲并回车执行：mvn help:system



maven默认会从上面的服务器(中央仓库) 下载 jar包到本地

## 2.3第三 修改setting.xml配置文件

在mirrors中添加下面的内容，使用阿里云服务器下载jar包，因为国外的下载太慢了，把国外的注释不用

|  |
| --- |
| <!-- 阿里云仓库 -->  <mirror>  <id>alimaven</id>  <mirrorOf>central</mirrorOf>  <name>aliyun maven</name>  <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>  </mirror>    <!-- 中央仓库1 -->  <mirror>  <id>repo1</id>  <mirrorOf>central</mirrorOf>  <name>Human Readable Name for this Mirror.</name>  <url>http://repo1.maven.org/maven2/</url>  </mirror>    <!-- 中央仓库2 -->  <mirror>  <id>repo2</id>  <mirrorOf>central</mirrorOf>  <name>Human Readable Name for this Mirror.</name>  <url>http://repo2.maven.org/maven2/</url>  </mirror> |

## 2.4第四：安装Eclipse的Maven插件 【工作中遇到再学习】

* 为什么要安装maven插件？
  + 需要一些命令，需要maven插件
  + 因为以前创建动态的**javaweb项目**，以后我们创建的是一个maven项目，maven也是一个web项目,java项目
* Maven项目是通过pom.xml文件来管理jar包的
* Eclipse安装Maven插件有三种方式
* **Eclipse neon的版本默认已经安装了maven的插件**
* **如果是旧版本的Eclipse，就需要手动安装maven插件，**
  + **一种离线安装，比较快**
  + **一种在线安装，比较慢**

### 第一种方式：把插件放在eclipse的dropins中【练】

* 【离线安装】，把插件放在eclipse的dropins
* 打开preferences点击Maven后，会出现下面的警告框，需要配置下eclipse.ini文件

|  |  |
| --- | --- |
|  |  |
| 解决上面的问题，只需要在eclipse.ini文件添加下面两行代码  -vm  C:\Program Files\Java\jdk1.8.0\_131\bin\javaw | |

* 接着需要点击maven的**User Setting**选项，配置路径为本地maven的settings.xml路径，如图

|  |  |
| --- | --- |
|  |  |

### 第二种方式：点击Eclipse的Help -> Install New Software

* 在弹出的界面中，添入下面红色框内容点击OK安装【http://m2eclipse.sonatype.org/sites/m2e】
* 装完后重启Eclipse

|  |
| --- |
|  |
|  |

* 安装完后，会在Eclipse的Preference偏好设置中出现maven选项

### 第三种方式：点击Eclipse的Help ->Eclipse Marketplace

* 然后在find中搜索maven安装即可

|  |
| --- |
|  |

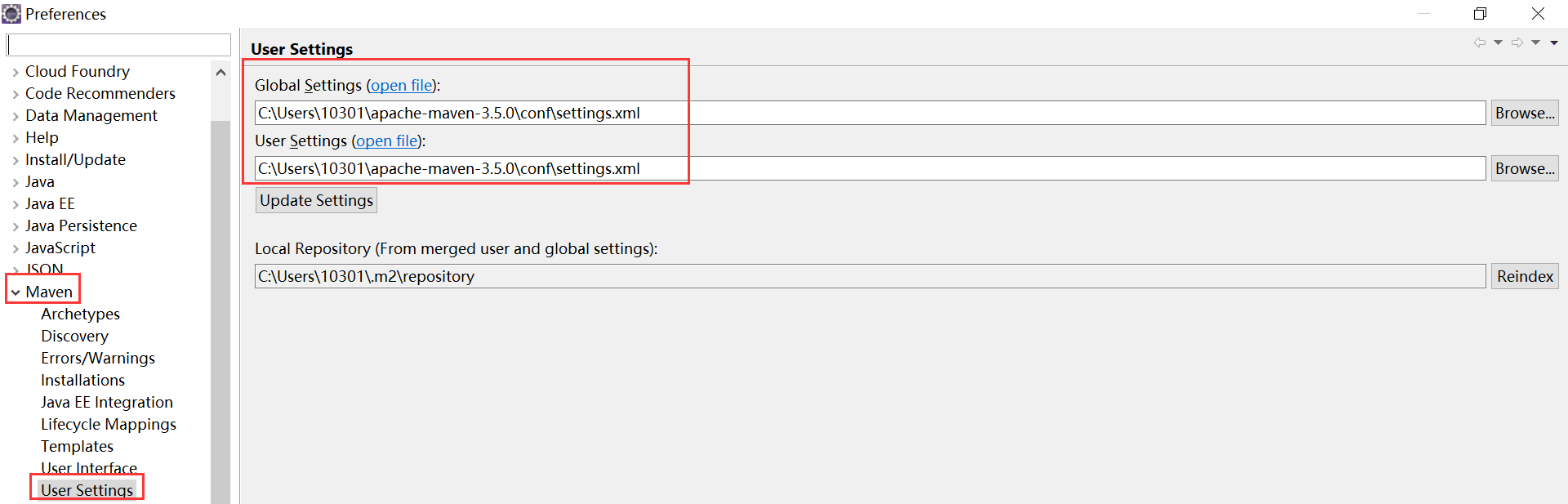
# 3 Maven常用命令

## 3.1 Maven项目结构

|  |
| --- |
| ProjectName  |-src  | |-main  | | |-java —— 存放项目的.java文件  | | |-resources —— 存放项目资源文件，如spring, hibernate配置文件  | |-test  | |-java ——存放所有测试.java文件，如JUnit测试类  | |-resources —— 测试资源文件  |-target —— **目标文件输出位置**例如.class、.jar、.war文件  |-pom.xml ——maven项目核心配置文件 |

## 3.2 eclipse创建Maven简单项目

创建项目前，把eclipse中maven配置文件路径改下



|  |
| --- |
| * **通过Eclipse在项目浏览器右键创建一个简单的maven项目，项目名为【test1】步骤如图** |
|  |
|  |
|  |
| * **Pom.xml，这里使用【junit 4.10进行单行测试】** |
| <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/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.gyf</groupId>  <artifactId>test1</artifactId>  <version>0.0.1-SNAPSHOT</version>  <!-- 添加依赖 -->  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.10</version>  <scope>test</scope>  </dependency>  </dependencies>  </project> |

## 在学习下面的命令前，打开终端切换到项目路径

|  |
| --- |
|  |

## 3.3 mvn compile

* 完成编译操作
* 执行完毕后，会生成target目录，该目录中存放了编译后的字节码文件。

## 3.4 mvn clean

* 执行完毕后，会将target目录删除。

## 3.4 mvn test

* 完成单元测试操作
* 执行完毕后，会在target目录中生成三个文件夹：surefire、surefire-reports（测试报告）、test-classes（测试的字节码文件）

## 3.4 mvn package

* 完成打包操作
* 执行完毕后，会在target目录中生成一个文件，该文件可能是jar、war

## 3.4 mvn install

* 执行 mvn install命令，完成将打好的jar包安装到本地仓库的操作
* 执行完毕后，会在本地仓库中出现安装后的jar包，方便其他工程引用

## 3.4 mvn 组合命令

* mvn clean compile
* mvn clean test
* mvn clean package
* mvn clean install

## 3.5 maven插件集成命令

* 当我们安装完maven的插件到eclipse后，插件提供了一些菜单可用
* 演示下clean install test 的效果

|  |
| --- |
|  |

**如果再使用过程中，遇到下面问题,解决办法**



# 4 项目引用另一个项目

* 现在我们来学习一个maven项目引用另一个maven项目
* 首先创建一个maven**简单项目**，名为test2
* 然后test2项目里写个单元测试代码，调用test1的service代码
* 实现的关键只需要在test2中添加test1项目的依赖即可，注意下图红色框的内容

|  |
| --- |
|  |

# 5 Maven的核心概念

## 5.1 坐标

* 在平面几何中坐标（x,y）可以标识平面中唯一的一点。在maven中坐标就是为了定位一个唯一确定的jar包。
* Maven世界拥有大量构建，我们需要找一个用来唯一标识一个构建的统一规范，拥有了统一规范，就可以把查找工作交给机器
* Maven坐标主要组成(GAV) -确定一个jar在互联网位置

|  |
| --- |
| **groupId**：定义当前Maven组织名称  **artifactId**：定义实际项目名称  **version**：定义当前项目的当前版本 |

## 5.2 坐标的查找

* 访问http://www.mvnrepository.com或者http://search.maven.org/网站
* 假设搜索所spring core,如图然后点击sping,接点选择所需要的版本，就能看到所需要的jar包了

|  |  |
| --- | --- |
|  |  |
|  |  |

## 5.3 依赖管理

### 5.3.1 scope 依赖范围

|  |
| --- |
| E:\工作\java\课件\17-Maven\讲义\依赖范围.JPG |

* 其中依赖范围**scope** 用来控制依赖和编译，测试，运行的classpath的关系. 主要的是三种依赖关系如下：
* compile： 默认编译依赖范围。对于编译，测试，运行三种classpath都有效
* test：测试依赖范围。只对于测试classpath有效
* provided：已提供依赖范围。对于编译，测试的classpath都有效，但对于运行无效。因为由容器已经提供，例如servlet-api
* runtime:运行时提供。例如:jdbc驱动

### 5.3.2依赖传递

* 直接依赖和间接依赖
* test2 依赖 test1，test3依赖test2
* test2 直接依赖 test1，test3间接依赖test1

|  |
| --- |
|  |
|  |

* 当第二依赖的范围是compile的时候，依赖可以传递
* 当第二直接依赖的范围是test的时候，依赖不会得以传递
* provided和runtime后面有遇到再讲，一般很少用，可以不用掌握。

### 5.3.3依赖冲突

* 假如test1使用junit4.10依赖,并且scope是compile,那test2,test3都可以使用test1的junit4.10，因为传递下来了
* 假如test2使用junit4.9依赖，那test3会使用【就近的一个依赖】，也就是使用junit4.9

### 5.3.4可选依赖

* <optional> true/false<optional> 是否可选，也可以理解为是否向下传递。
* 在依赖中添加optional选项决定此依赖是否向下传递，如果是true则不传递，如果是false就传递，默认为false

|  |
| --- |
|  |

### 5.3.5 排除依赖

* exclusions可用于排除依赖,**注意exclusions是写在dependency中**

|  |
| --- |
|  |

## 5.4 生命周期【了解】

* Maven生命周期就是为了**对所有的构建过程进行抽象和统一**。
* 包括项目清理、初始化、编译、打包、测试、部署等几乎所有构建步骤。
* **生命周期可以理解为构建工程的步骤。**
* 在Maven中有三套相互独立的生命周期，请注意这里说的是“三套”，而且“相互独立”，这三套生命周期分别是：
* Clean Lifecycle： 在进行真正的构建之前进行一些清理工作。
* Default Lifecycle： 构建的核心部分，编译，测试，打包，部署等等。
* Site Lifecycle： 生成项目报告，站点，发布站点。

### 5.4.1 Clean生命周期：清理项目

* Clean生命周期一共包含了三个阶段：

|  |
| --- |
| pre-clean 执行一些需要在clean之前完成的工作  clean 移除所有上一次构建生成的文件  post-clean 执行一些需要在clean之后立刻完成的工作 |

* 也就是说，mvn clean 等同于 mvn pre-clean clean
* 如果我们运行 mvn post-clean ，那么 pre-clean，clean 都会被运行。
* 这是Maven很重要的一个规则，可以大大简化命令行的输入。

### 5.4.2 Default生命周期：构造项目

* Default生命周期是Maven生命周期中最重要的一个，绝大部分工作都发生在这个生命周期中。这里，只解释一些比较重要和常用的阶段

|  |
| --- |
| validate  generate-sources  process-sources  generate-resources  process-resources 复制并处理资源文件，至目标目录，准备打包。  **compile** 编译项目的源代码。  process-classes  generate-test-sources  process-test-sources  generate-test-resources  process-test-resources 复制并处理资源文件，至目标测试目录。  test-compile 编译测试源代码。  process-test-classes  **test** 使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。  prepare-package  **package** 接受编译好的代码，打包成可发布的格式，如 JAR 。  pre-integration-test  integration-test  post-integration-test  verify  **install** 将包安装至本地仓库，以让其它项目依赖。  deploy 将最终的包复制到远程的仓库，以让其它开发人员与项目共享 |

* **运行任何一个阶段的时候，它前面的所有阶段都会被运行**
* 这也就是为什么我们**运行mvn install 的时候，代码会被编译，测试，打包，安装到本地仓库**
* 此外，Maven的插件机制是完全依赖Maven的生命周期的，因此理解生命周期至关重要。

### 5.4.3 Sit生命周期：生成项目站点

* 这里经常用到的是site阶段和site-deploy阶段，用以生成和发布Maven站点，这可是Maven相当强大的功能
* Manager比较喜欢，文档及统计数据自动生成，很好看。我们就不用学，以后自己研究

|  |
| --- |
| pre-site 执行一些需要在生成站点文档之前完成的工作  site 生成项目的站点文档  post-site 执行一些需要在生成站点文档之后完成的工作，并且为部署做准备  site-deploy 将生成的站点文档部署到特定的服务器上 |

# 6 Maven插件

* 可以通过pom.xml配置插件来更改项目编译compile的jdk版本

## 6.1 compile插件

* 如图：

|  |
| --- |
|  |
| **代码** |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  </build> |

# 7 创建web的maven项目

## 7.1创建步骤如图

|  |
| --- |
| * 跟前面创建maven简单项目一样，只不过到了下面截图步骤时，要在packaging改成war |
|  |
| * maven的web项目目录结构比简单项目多了个webapp目录 * 我们需要在webapp项目中添加WEB-INF目录和web.xml文件 |
|  |

## 7.2配置tomcat插件跑web项目

### tomcat:run

* 直接在运行配置中配置一个tomcat:run,默认跑的是tomcat 6.0,如图
* Tomcat6与jdk1.8不兼容，在jre改成1.7

|  |
| --- |
|  |

### tomcat7:run

* 需要先在pom.xml配置tomcat7的插件，然后运行配置中配置tomcat7:run

|  |
| --- |
| <build>  <plugins>  <plugin>  <!-- 配置插件tomcat7 -->  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8080</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build> |
|  |

### 本地tomcat运行

|  |
| --- |
| pom.xml |
| <build>  <plugins>  <plugin>  <groupId>org.codehaus.cargo</groupId>  <artifactId>cargo-maven2-plugin</artifactId>  <version>1.2.4</version>  <configuration>  <container>  <containerId>tomcat7x</containerId>  <home>C:\Users\guoyongfeng\apache-tomcat-7.0.77</home>  </container>  <configuration>  <type>existing</type>  <home>C:\Users\guoyongfeng\apache-tomcat-7.0.77</home>  <properties>  <cargo.servlet.port>9234</cargo.servlet.port>  </properties>  </configuration>  </configuration>  </plugin>  </plugins>  </build> |
| run config 运行配置,start前需要install |
|  |

## 7.3配置jetty插件跑web项目

|  |
| --- |
| Pom.xml |
| <build>  <finalName>test4</finalName>  <plugins>  <plugin>  <groupId>org.mortbay.jetty</groupId>  <artifactId>maven-jetty-plugin</artifactId>  <version>6.1.26</version>  <configuration>  <scanIntervalSeconds>10</scanIntervalSeconds>  <webApp>${basedir}/target/test4</webApp>  <webAppConfig>  <!-- 访问路径 根路径访问，演示/test4又是什么情况-->  <contextPath>/</contextPath>  </webAppConfig>  <connectors>  <connector implementation=*"org.mortbay.jetty.nio.SelectChannelConnector"*>  <port>8888</port>  <headerBufferSize>16192</headerBufferSize>  </connector>  </connectors>  </configuration>  </plugin>  </plugins>  </build> |
| Run configrations |
|  |

## 7.4 也可以直接run as 到tomcat中

# 8 继承

* 继承是为了消除重复，可以把很多相同的配置提取出来。例如：grouptId，version等

## 创建父工程

|  |
| --- |
|  |

## 创建子工程

* 配置Parent的GAV
* 然后在父工程中添加的依赖，子工程可以直接继承使用

|  |
| --- |
|  |

## 父工程统一管理依赖

|  |
| --- |
| * **在父工程中通过配置dependencies依赖，子工程可以直接继承使用，如下图** |
|  |
| * **在父工程中通过配置dependencyManagement管理依赖的版本号** * 注意：此处只是定义依赖jar包的版本号，并不实际依赖。如果子工程中需要依赖jar包还需要添加dependency节点。 |
|  |

## 父工程中版本号提取

* 当父工程中定义的jar包越来越多，找起来越来越麻烦，所以可以把版本号提取成一个属性集中管理。
* 在父工程的pom.xml中配置一个**properties标签**，里面的**标签自己自定义即可**

|  |
| --- |
|  |

# 聚合

聚合一般是一个工程拆分成多个模块开发，每个模块是一个独立的工程，但是要是运行时必须把所有模块聚合到一起才是一个完整的工程，此时可以使用maven的聚合工程。

例如电商项目中，包括商品模块、订单模块、用户模块等。就可以对不同的模块单独创建工程，最终在打包时，将不同的模块聚合到一起。

例如同一个项目中的表现层、业务层、持久层，也可以分层创建不同的工程，最后打包运行时，再聚合到一起。

## 9.1创建聚合项目

* 聚合模块的创建的packagin也是使用pom，GroupId和ArtifaceID根据项目来写，**假如我有个淘宝电商项目**

|  |
| --- |
| **创建时，聚合项目的packaging一定要用pom,配置如图** |
|  |
| 在pom.xml配置指定版本的编译器 |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  </build> |

## 9.2创建Dao模块

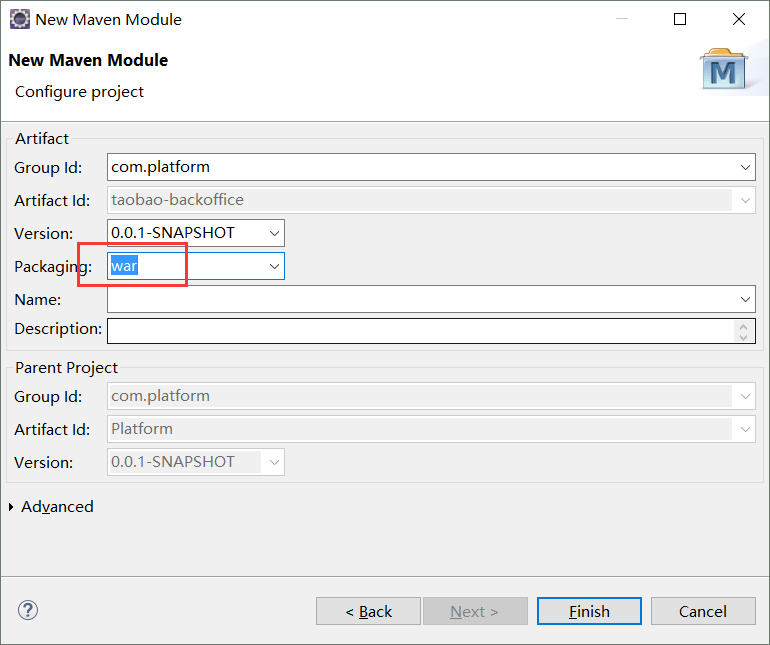
|  |
| --- |
| 创建模块时，要使用Maven Module |
|  |
| **注意下面红色框框下的Parent Project要选我们写的聚合项目Platform** |
|  |

## 9.3创建Service模块

* 与dao创建方式相同

## 9.4创建web模块

* 这是要注意packaging要先war,代表是一个web项目
* 这里取名为taobao-**backoffice,意为后台管理**



## 9.5聚合项目结构如图

|  |
| --- |
|  |

## 9.6 运行backoffic web工程

* 在backoffice的pom.xml中配置tomcat插件即可,然后配置run configurations的tomcat7:run

|  |
| --- |
| <build>  <plugins>  <plugin>  <!-- 配置插件tomcat7 -->  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8080</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build> |
|  |

## 9.7 模块引用

如果一个模块要引用另一个模块，如backoffice要引用service,只需要添加依赖即可，如图

|  |
| --- |
|  |

## 9.8创建表实体模块

其实在开发中，还会把表实体也抽取一个模块出来,如图

|  |
| --- |
|  |
| * 一般会在dao中引用entity模块 * service中引用dao模块 |

## 9.9如何发布聚合项目

* 只需要把backoffice 项目install下，在他的targit目录下，把.war包丢进服务器即可,如图

|  |
| --- |
|  |
| 看下模块结构的结果，使用360打开war包，可见其它的jar所导入到了web的lib中了,这是  Maven自动导入的 |
|  |

# Maven的仓库

## 10.1什么是仓库

* 用来统一存储所有Maven共享构建的位置就是仓库。
* 根据Maven坐标GAV定义每个构建在仓库中唯一存储路径大致为

|  |
| --- |
| groupId/artifactId/version/artifactId-version.packaging |
|  |
|  |

## 10.2仓库的分类

1、本地仓库

~/.m2/repository每个用户只有一个本地仓库

2、远程仓库

* 中央仓库：Maven默认的远程仓库，不包含版权资源 <http://repo1.maven.org/maven2>
* 但是中央仓库地址一般都下载比较慢时，可以换成**阿里云的中央仓库**
* <http://maven.aliyun.com/nexus/content/repositories/central/>
* 也可以自己抢建一个仓库服务器来下载依赖
* 私服：是一种特殊的远程仓库，它是架设在局域网内的仓库

|  |
| --- |
| 私服 |

* nexus就是用来创建maven的私服的，也就是提高依赖的下载速度，其实我们用阿里云的私服就好了，就不要折腾了。我是一条咸鱼

# 11.Maven的配置文件

* maven的用户配置文件有两种方式，一种是全局配置文件，一种是用户配置文件
* 本人很少用，直接用全局

## 案例：使用用户配置文件

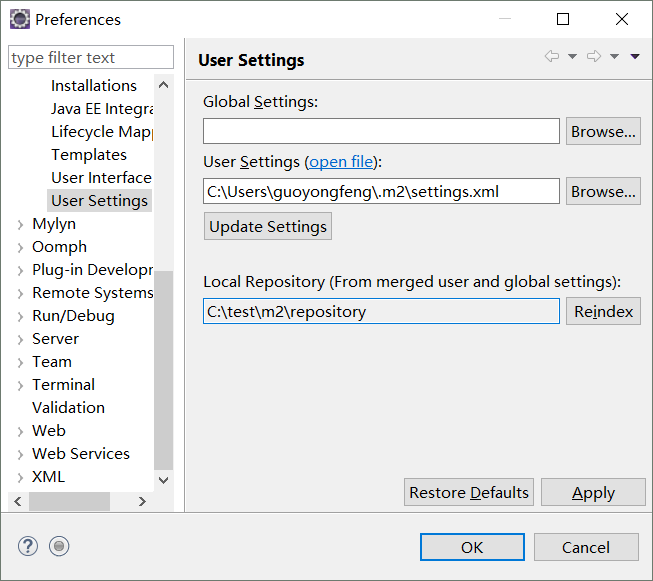
* 根据下面的截图创建目录，用于放置依赖的存储路径

|  |
| --- |
|  |

* 然后从全局中复制一个settings.xml到用户的.m2目录中
* 然后编辑setting.xml文件，配置localRepository为新创建的仓库路径

|  |
| --- |
|  |
|  |

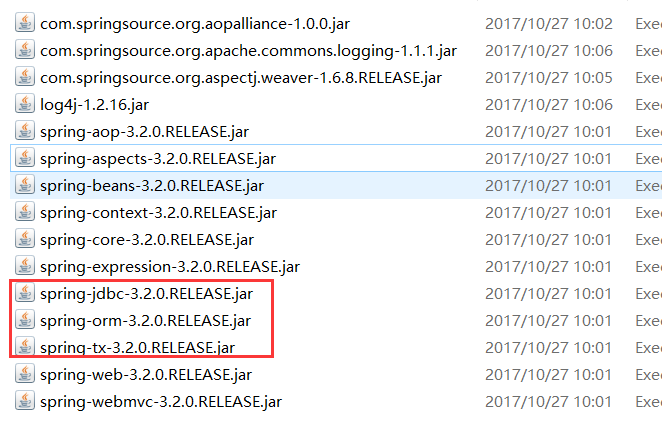
* 在偏好设置中，配置user settings选向为用户目录的settings.xml
* 以后所有的maven项目的**依赖都会下载到c:\test\m2\repository**中



**注意：**

**用户级别的仓库在全局配置中一旦设置，全局配置将不再生效，转用用户所设置的仓库**

# 12. 整合springmvc



|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-expression</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aop</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>aopalliance</groupId>  <artifactId>aopalliance</artifactId>  <version>1.0</version>  </dependency> |