# 传智播客Maven视频讲解：

## Maven基础：

### 1、Maven的好处如何实现：

这依靠Maven的两大核心：

1. 依赖管理：对jar包管理过程
2. 项目构建：项目在编码完成后，对项目进行编译、测试、打包、部署等操作

### 仓库类型：

1. 本地仓库：在程序员的笔记本上
2. 私服：存在局域网的一台服务器（私服需要安装配置）
3. 中央仓库：在互联网上，几乎存放所有的jar包，由Maven团队维护

### 3、配置本地仓库：

|  |
| --- |
|  |

在安装目录 F:\Maven\apache-maven-3.5.0 下的conf目录下找到 setting.xml文件，添加上面的<localRepository>节点

### 4、Maven项目标准目录结构：

|— helloworld（项目名）

|— .setting

|— src（项目源码）

|— main（程序主要代码）

|— java（程序代码）

|— resources（存放配置文件，如struts.xml）

|— webapp（相当于webContent，存放的是jsp文件、web.xml文件）  
|— test（单元测试代码）

|— java

|— resources

|— target（编译后的文件，即 .class 文件）

|— .classpath

|— .project

|— pom.xml（Maven项目核心配置文件）

### 5、Maven常用命令：

1）清理（mvn：clean）

将项目根目录下target目录清理（即将编译好的 .class 文件清理）。

执行该命令后会将项目根目录的target文件夹删除

1. 编译（mvn：compile）

对java源码进行编译，生成的 .class 文件会生成在target目录下。

问题1：如果想要将 .class 文件按照bao结构生成，应该怎么做？即java源文件在com/chenming包下，如果在target目录下calsses目录也生成com/chenming结构？

1. 单元测试后（mvn：test）

执行src/test/java目录下的单元测试类

注意：单元测试类名有要求： XxxTest.java

1. 打包（mvn：package）

web project就打成war包

java project就打成jar包

1. 安装（mvn：install）

多个项目都需要使用某一个jar包，可以将这个jar包安装本地仓库中，多个项目引用这个jar包即可

## 二、Maven项目的生命周期：

### 1、Maven中的三套生命周期：

1）在Maven中存在三套生命周期，分别是CleanLifeCycle、DefaultLifeCycle、SiteLifeCycle。

2）这三套生命周期相互独立、互不影响。当你执行DefaultLifeCycle生命周期的install命令时，它只会执行DefaultLifeCycle的complie、test、package等命令，不会跑去执行CleanLifeCycle生命周期的clean命令。

3）你可以仅调用 clean来清理工作目录，仅调用 site来生 成站点。当然你也可以直接运行mvn clean install site运行所有这三套生命周期。

4）Maven核心程序为了更好的实现自动化构建，按照这一特点执行生命周期中的各个阶段：在一套生命周期内，不论现在要执行生命周期中的哪一个阶段，都是从这个生命周期最初的位置开始执行。

【如：当你执行DefaultLifeCycle的test命令时，它会依次执行compile、test；当你执行DefaultLifeCycle的install命令时，它会依次执行compile、test、package、install】

### 2、CleanLifeCycle：清理的生命周期

1）包含命令： clean

2）职责： 在进行真正的项目构建之前做一些清理工作

3）CleanLifeCycle包含的阶段：

【1】pre-clean：执行一些需要在clean之前预处理的工作

【2】clean：移除所有上一次构建生成的文件

【3】post-clean：执行一些需要在clean之后立刻完成的工作

### 3、DefaultLifeCycle：构建的核心部分

1）包含命令： compile、test、package、install、deploy等等

2）职责： 对项目进行编译、测试、打包、安装到本地仓库、部署到服务器等等操作

3）DefaultLifeCycle包含的阶段（这里只介绍部分阶段）：

【1】validate

【2】generate-sources

【3】process-sources

【4】generate-resources

【5】process-resources： 复制并处理资源文件，至目标目录，准备打包。

【6】compile：编译项目的源代码。

【7】process-classes

【8】generate-test-sources

【9】process-test-sources

【10】generate-test-resources

【11】process-test-resources： 复制并处理资源文件，至目标测试目录。

【12】test-compile： 编译测试源代码。

【13】process-test-classes

【14】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。

【15】prepare-package

【16】package： 接受编译好的代码，打包成可发布的格式，如JAR、WAR。

【17】pre-integration-test

【18】integration-test

【19】post-integration-test

【20】verify

【21】install：将包安装至本地仓库，以让其它项目依赖。

【22】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或部署到服务器上运行。

### 4、SiteLifeCycle：站点生命周期

1）包含命令： site

2）职责：生成项目报告、站点发布

3）SiteLifeCycle包含的阶段：

【1】pre-site：执行一些需要在生成站点文档之前的工作。

【2】site：生成项目的站点文档

【3】post-site：执行一些需要在生成站点文档之后完成的工作，并为部署做准备

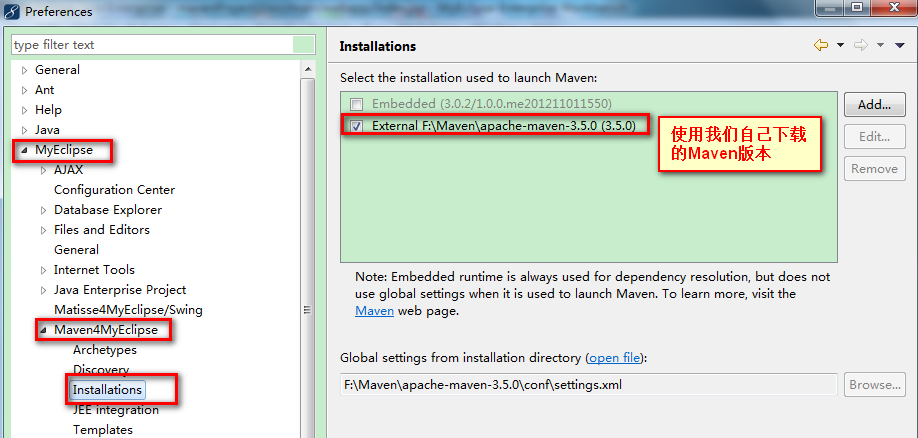
【4】site-deploy：将生成的站点文档部署到特定服务器

## 三、Maven整合web项目：【参考MyEclipse\】

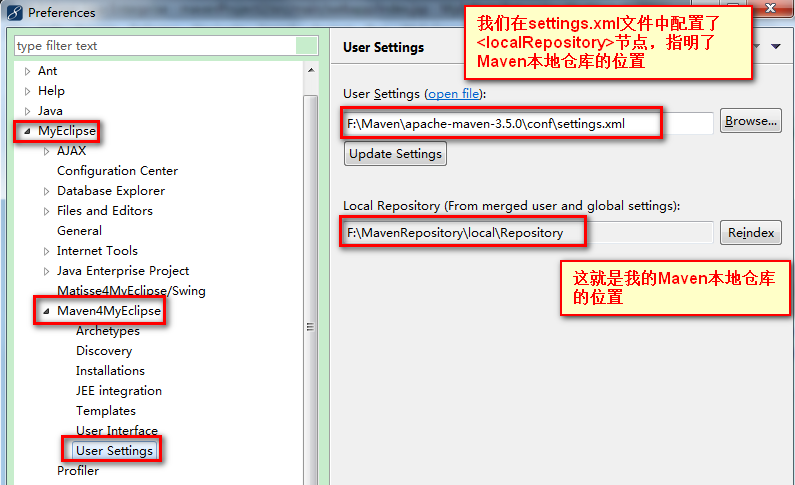
### 1、MyEclipse中安装配置

1）安装Maven插件（m2e）

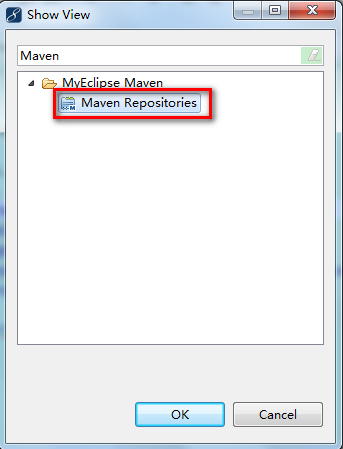
2）配置Maven程序

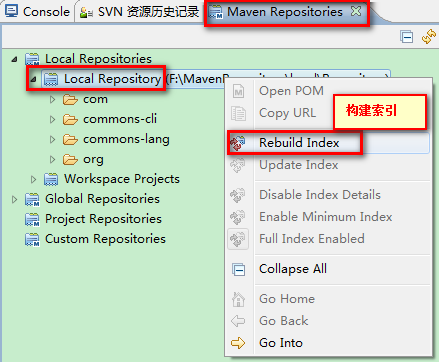


1. 配置userSettings：让MyEclipse知道我们的Maven本地仓库位置



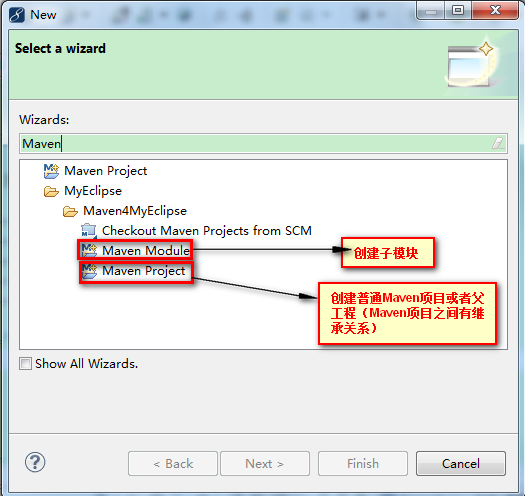
1. 进行了上面的匹配，MyEclipse能够找到Maven本地仓库，然后再找jar包，但是如果本地仓库太大，那么检索jar就比较耗时，为了提高检索效率，可以构建索引

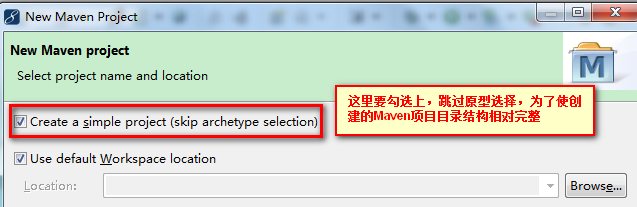


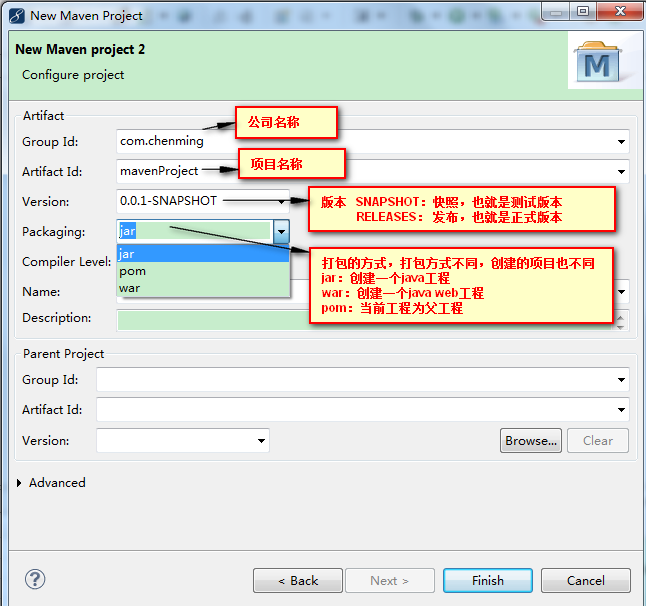


### 2、Maven整合Servlet：【参考mavenProject】

1）创建Maven项目







1. 设置Maven的编译版本

|  |
| --- |
| <build>  <plugins>  <!-- 设置编译版本为1.7 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.7</source>  <target>1.7</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  </build> |

1. 缺少servlet jar包，报错，需要在pom.xml中添加依赖

|  |
| --- |
| <!-- 配置依赖 -->  <dependencies>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  </dependency>  </dependencies> |

### 3、依赖范围：

1）图解：



【注：这里的“对于运行时classpath有效”就是指部署到服务器上是否有效】

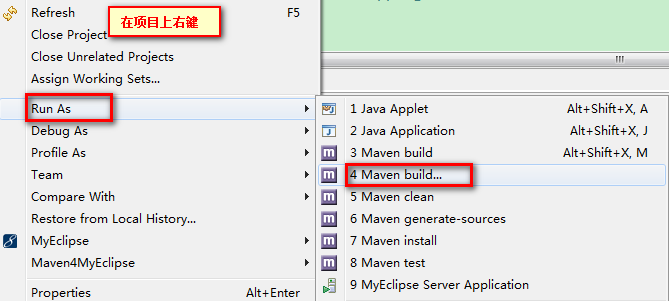
【实际的情况是servlet-api不能部署到服务器上，现在假如servlet-api.2.5.jar 能够部署到服务器上，那么当将项目打war包的时候，war包中会包含一个servlet-api.2.5.jar,但是我们的tomcat服务器自己也包含了一个servlet-api.jar，这时候两个jar包就会冲突，会报错，所以servlet-api.jar的范围是provided，而不是compile】

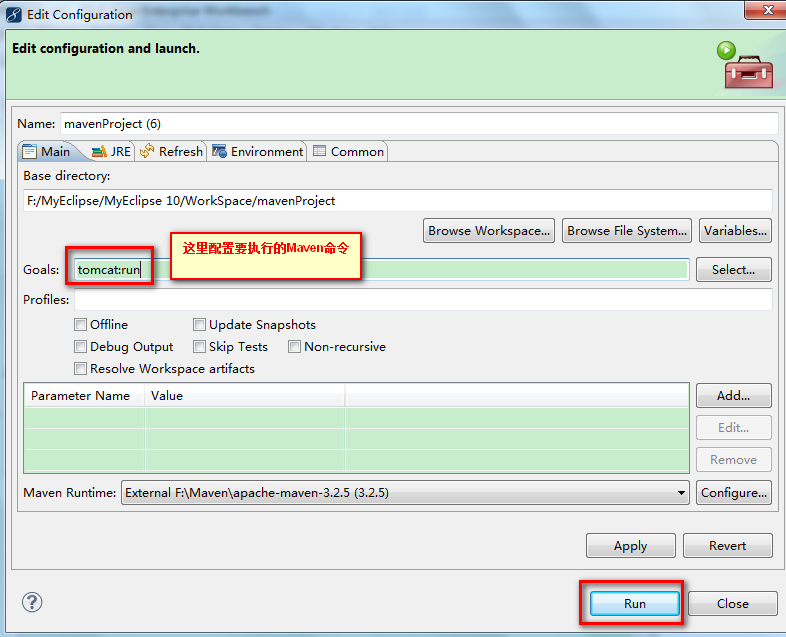
1. 添加的依赖默认的范围是compile
2. 总结：

如果使用到了服务器（Tomcat、JBoss）lib下的jar包，那么依赖的jar包的scope就只能是provided

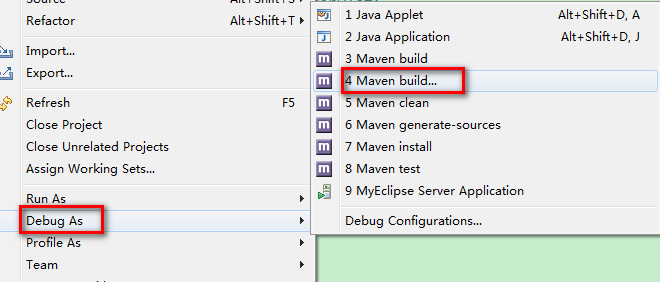
## 四、运行、调试Maven项目：

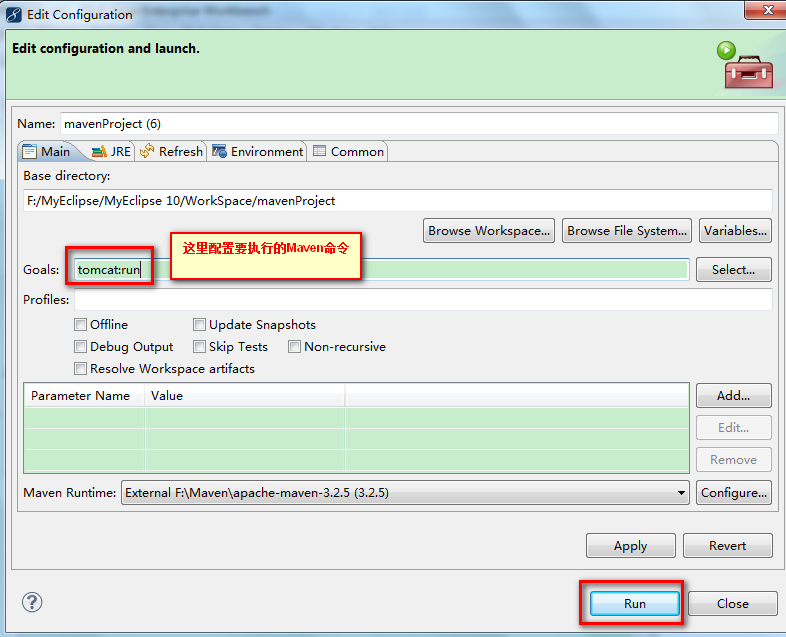
### 1、运行Maven项目：





### 2、调试Maven项目：（debug）





## 五、Maven整合struts2项目：【参考mavenProject2】

### 1、创建Maven项目

### 2、添加struts2依赖

|  |
| --- |
| <!-- 添加依赖 -->  <dependencies>  <!-- 添加struts2依赖 ,而struts2依赖的jar会自动下载并且依赖-->  <dependency>  <groupId>org.apache.struts</groupId>  <artifactId>struts2-core</artifactId>  <version>2.3.24</version>  </dependency>  </dependencies> |

### 3、在src/main/resources下添加struts.xml核心配置文件

### 4、在web.xml中配置struts2框架核心过滤器

|  |
| --- |
| <!-- 配置struts2框架核心过滤器 -->  <filter>  <filter-name>struts</filter-name>  <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>struts</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

### 5、在struts.xml中配置action

## 六、概念模型：

### 1、图解：

|  |
| --- |
|  |

local：本地仓库

b2b：私服

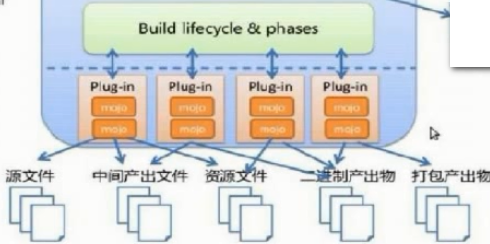
central：远程仓库

### 2、概念模型描述的就是Maven的两大核心

1）依赖管理：



2）项目构建：



## 七、Maven实战：

### 1、传递依赖解决冲突：【参考mavenProject3】

1）传递依赖：A依赖B，B依赖C，那么B就是A的直接依赖，C就是A的传递依赖

2）冲突发生的情况：A工程依赖B jar包，B jar包依赖C jar包（1.1版本）；同时A工程又依赖于D jar包，D jar包又依赖于C jar包（1.2版本），那么最后是导入C jar包的1.1版本还是1.2版本呢？这就依靠与Maven的调节原则：

【1】Maven调节原则：

①第一声明者优先原则：1.1版本和1.2版本谁先定义的就用谁的。（如果A先依赖了B，那么C（1.1）就会被先依赖传递进来，那么C jar包就会使用1.1版本；反之亦然）【总结：就是pom.xml中dependency节点的顺序】

②路径近者原则：直接依赖级别高于传递依赖级别（若是在pom.xml文件中，在<dependency>节点中直接声明了C jar包（1.3），那么不管这个<dependency>节点在<dependencies>节点中的顺序如何，都是使用C jar（1.3）版本的）

【2】排除依赖：

|  |
| --- |
|  |

【3】版本锁定（推荐使用）：指定项目中依赖的版本

|  |
| --- |
|  |
|  |

## 八、Maven整合SSH框架：（重点）【参考mavenProject4】

### 1、搭建Struts2环境：

1）创建struts2的核心创建文件：struts.xml文件

2）在web.xml中配置核心过滤器StrutsPrepareAndExecuteFilter

### 2、搭建spring环境：

1）创建spring核心配置文件：applicationContext.xml

2）在web.xml中配置监听器：ContextLoadListener

### 3、搭建hibernate环境：

1）创建hibernate核心配置文件：hibernate.cfg.xml

### 4、整合struts2和spring

1）整合关键点：action对象的创建交给spring

2）步骤：

【1】创建action类

【2】将action对象配置到spring配置文件中

|  |
| --- |
|  |

【3】在struts.xml中在action节点中class属性配置为spring工厂中action对象bean的id

|  |
| --- |
|  |

### 5、整合hibernate和spring

1）整和关键点：

【1】数据源dataSource交给Spring

【2】SessionFactory对象创建交给Spring

【3】事务管理交给Spring

2）步骤：

【1】配置dataSource：

|  |
| --- |
|  |

【2】配置SessionFactory：

|  |
| --- |
| <!-- 2、配置SessionFactory -->  <bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate4.LocalSessionFactoryBean"* scope=*"prototype"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  <property name=*"configLocations"* value=*"classpath:hibernate.cfg.xml"*></property>  </bean> |

【3】事务管理：

1. 配置事务管理器：PlatFormTransactionManager接口

i）使用jdbc技术：DataSourceTransactionManager

ii）使用Hibernate技术：HibernateTransactionManager

|  |
| --- |
| <!-- 3、配置事务管理器 -->  <bean id=*"hibernateTransactionManager"* class=*"org.springframework.orm.hibernate4.HibernateTransactionManager"*>  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean> |

b）管理事务的两种方式：

i）xml方式管理事务：

①配置通知

|  |
| --- |
| <tx:advice id=*"txAdvice"*>  <tx:attributes>  <!-- 匹配业务类中方法名称,即当业务类中方法名称带 save、delete、update等字符的时候,表明是要事务控制的 -->  <tx:method name=*"save\*"*/>  <tx:method name=*"delete\*"*/>  <tx:method name=*"update\*"*/>  <tx:method name=*"find\*"* read-only=*"true"*/> <!-- 这是查询方法,不需要事务控制,所以设置read-only属性为true -->  <tx:method name=*"\*"*/>  </tx:attributes>  </tx:advice> |

②配置AOP：

|  |
| --- |
| <aop:config>  <!-- 配置切点：具体哪些方法要增强(service层的所有类的所有方法都被配置切点) -->  <aop:pointcut expression=*"execution(\* com.chenming.service.\*.\*(..))"* id=*"cut"*/>  <!-- 配置切面：将增强逻辑作用到切点(通知+切入点) -->  <aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"cut"*/>  </aop:config> |

ii）注解方式管理事务：

①开启注解扫描：

|  |
| --- |
| <tx:annotation-driven transaction-manager=*"transactionManager"*/> |

②在service类上或方法上使用注解 @Transactional

### 6、实际操作：

1）需求：在地址栏输入action请求，action调用service，service调用dao，完成客户查询

2）具体实现：

【1】创建客户实体类、映射文件，将映射文件引入hibernate核心配置文件中：

客户实体类 Customer类：

|  |
| --- |
| **public** **class** Customer  {  **private** String custId;  **private** String custName;  **private** String address;    /\* Setter And Getter \*/  //...  } |

实体类的映射文件 Customer.hbm.xml：

|  |
| --- |
| <hibernate-mapping>  <class name=*"com.chenming.domain.Customer"* table=*"CUSTOMER"*>  <!-- 配置Customer的id属性为CUSTOMER表的主键 -->  <id name=*"custId"* type=*"java.lang.String"*>  <column name=*"cust\_id"* length=*"32"*></column>  <generator class=*"uuid"*></generator>  </id>  <property name=*"custName"* type=*"java.lang.String"*>  <column name=*"cust\_name"*></column>  </property>  <property name=*"address"* type=*"java.lang.String"*>  <column name=*"address"*></column>  </property>  </class>  </hibernate-mapping> |

将映射文件引入hibernate核心配置文件中：

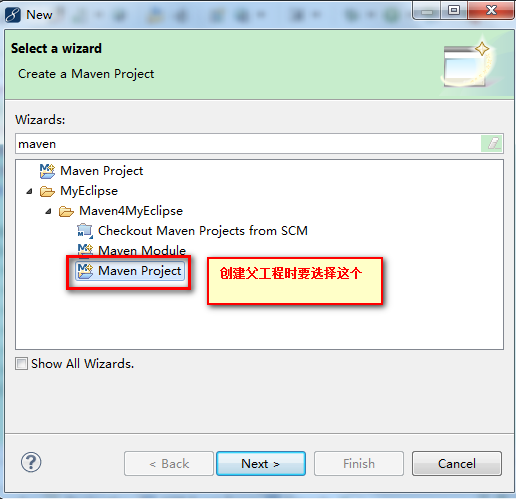
|  |
| --- |
| <!-- 加载映射文件:以包的形式 -->  <mapping resource=*"com/chenming/domain/Customer.hbm.xml"*/> |

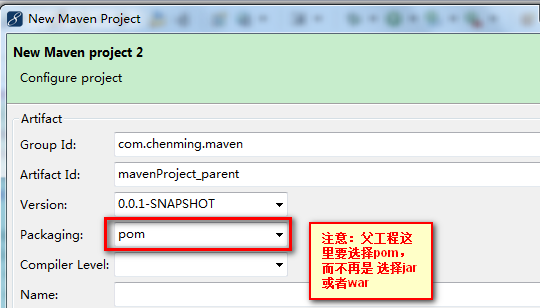
【2】

## 九、Maven项目拆分：

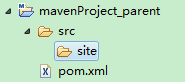
### 1、创建父工程：

1）步骤：





下面是创建好的maven父工程的项目结构，已经与之前创建maven项目不一样了。



创建好的父工程目录结构：只有pom.xml，可以推断父工程不进行编码。

1. 父工程的作用：

【1】项目需要的依赖信息，在父工程中定义，子模块继承父工程也就继承了依赖信息

【2】将各个子模块聚合到一起

### 2、将创建好的父工程发布到本地仓库：

### 3、创建子模块（子工程）

## 传递依赖的作用范围：（了解）

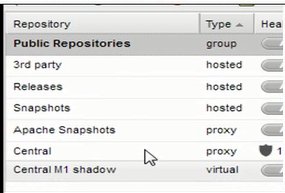
### 1、图解：



## 十一、私服应用：

### 1、私服安装：

### 2、私服类型：



1）hosted：宿主仓库

存放本公司开发的jar包，有正式版本（Releases）、测试版本（Snapshots）、第三方（3rd party）

2）proxy：代理仓库

3）group：组仓库

包含hosted宿主仓库、proxy代理仓库；将来我们连接group组仓库