# Maven的简介

<https://blog.csdn.net/zhaojianting/article/details/80324533>

## 什么是maven

1. Maven和Svn eclipse是同一量级的开发工具。
2. apache的开源项目，纯java开发，并且**只是用来管理java项目**的。
3. 是软件开发管理工具，主要管理工作是：**依赖管理、项目构建。**

## Maven好处

1. Maven项目比普通的传统项目的占用空间小。因为maven构建的项目不包含jar包。
2. 依赖管理。不需要导入jar包。在pom.xml中定义坐标从仓库中自动获取。
3. 一键构建。一个tomcat:run就能把项目运行起来
4. 跨平台

# Maven的安装和配置

## Maven环境变量配置

**配置环境变量的目的是**：能够在任意位置运行的maven软件中bin目录的mvn命令

|  |
| --- |
| 1. MAVEN\_HOME:D:\installSite\apache-maven-3.3.9(bin文件夹的上一层) 2. 添加MAVEN\_HOME到Path:% MAVEN\_HOME %/bin   验证maven是否配置成功：**mvn–v**  **注意：maven3.3以上的版本所需的jdk版本必须要1.7以上** |

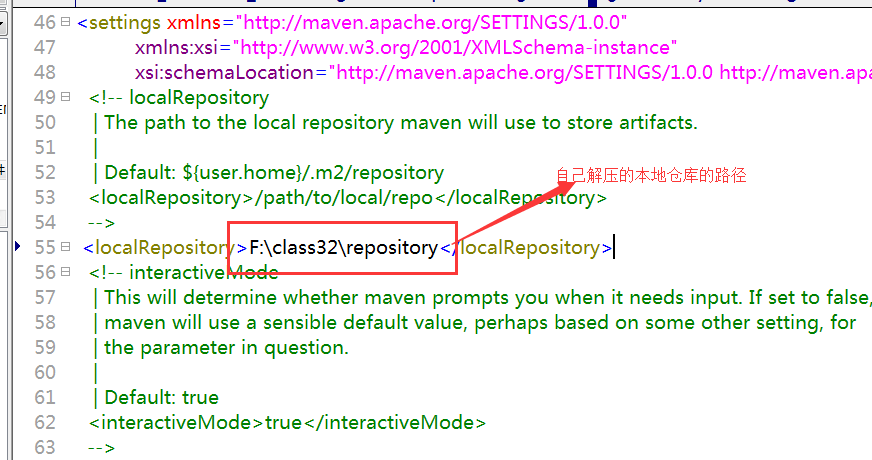
## Maven仓库配置

### 仓库的分类：

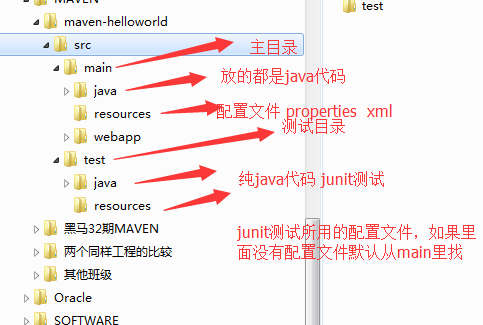
* 本地仓库
* 私服（公司的仓库）
* 中央仓库（maven团队维护，镜像）

#### 配置本地仓库：

本地仓库的配置只需要修改settings.xml文件就可以



# Maven的目录结构



# Maven的常用命令

**首先：进入需要测试的项目的目录。**

## Clean 清理命令

作用：删除target目录。

## Compile 编译命令

作用：将src/**main**/java下的文件编译为class文件，输出到target目录下。

**注意：并没有编译运行test目录下的代码。**

## Test 测试命令

作用：编译并运行了**test目录**的代码

## Package 打包命令

作用：打包成war包或者jar包，打包的位置在pom.xml文件中设置。

## install打包至本地仓库

作用：把自己的maven项目，打包至本地仓库。

一般用于打包成jar包，放入本地仓库中，便于其他项目调用。不会把war包打包到本地仓库，因为war包是不能调用的。

## tomcat：run 一键启动

|  |
| --- |
| 小结：Compile test package install deploy，后面的命令包含前面的命令，比如执行deploy则会执行这5个，执行package则会执行前三个。这个生命周期有关。了解 |

# Maven的生命周期（了解）

1. 三套生命周期：

Clean生命周期、Default生命周期、Site生命周期

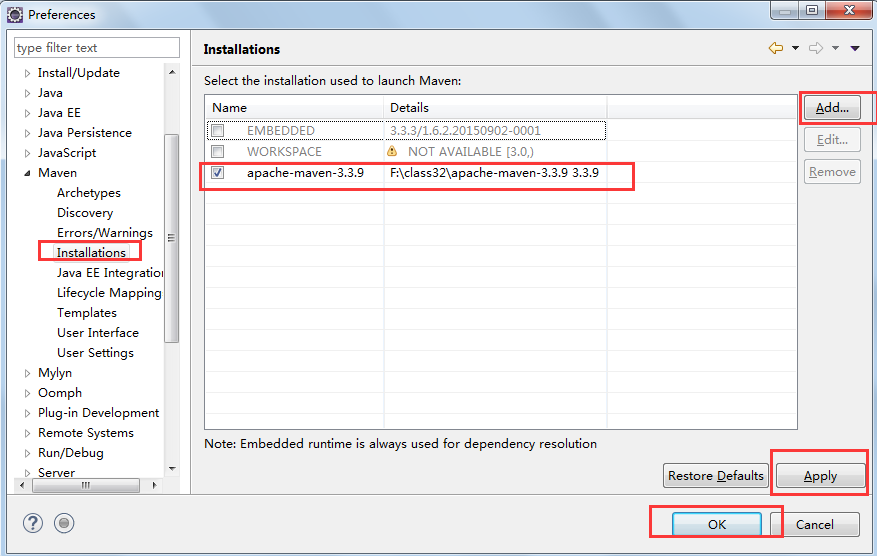
每一套生命周期中又有不同的命令。

1. 不同的生命周期的命令可以同时执行。如：mvn clean package，按顺序执行。（clean是clean生命周期中的命令，package是Default生命周期中的命令）

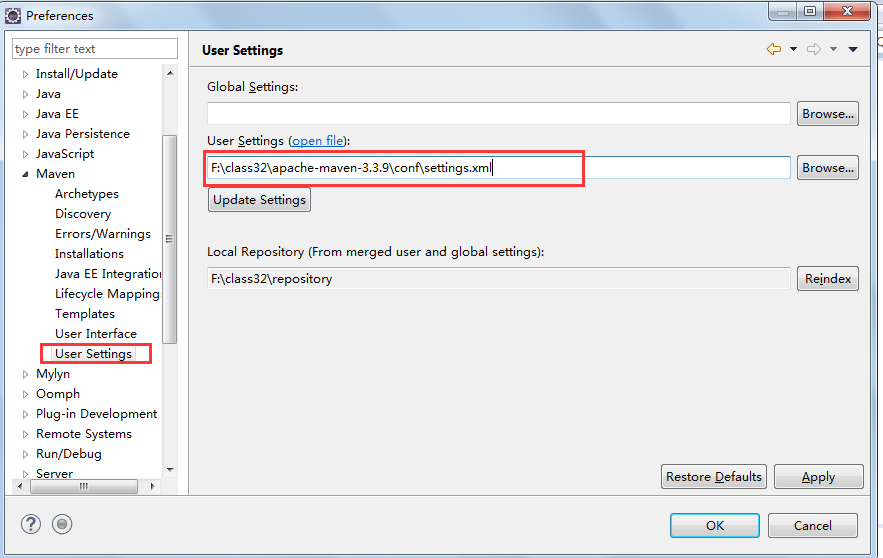
# eclipse开发maven项目

## 准备工作：

* **1.添加自己安装的maven**



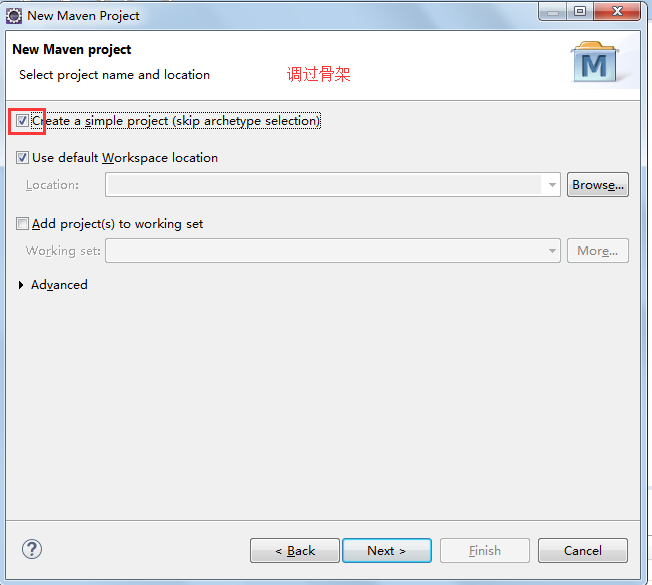
* **2. 指定setting.xml文件的位置**



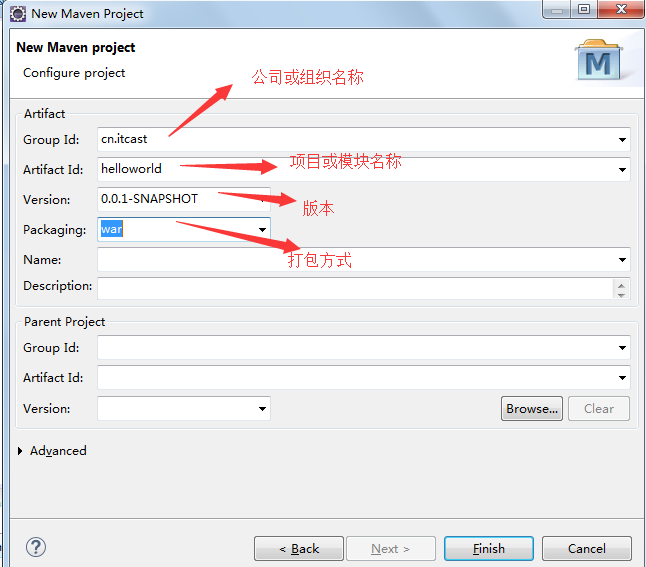
* **3.构建索引**

**打开窗口：maven repositories--->rebuild Index**

## 创建maven项目

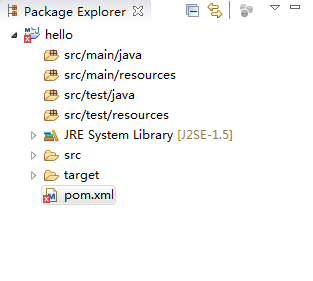


1. 填写坐标



## 处理异常：

1、处理红色叉号



**手动**在webapp文件夹下创建一个WEB-INF文件夹，在里面放一个web.xml文件

2、处理编译版本

在pom.xml中添加如下代码

<build>

<!--配置了很多插件 -->

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

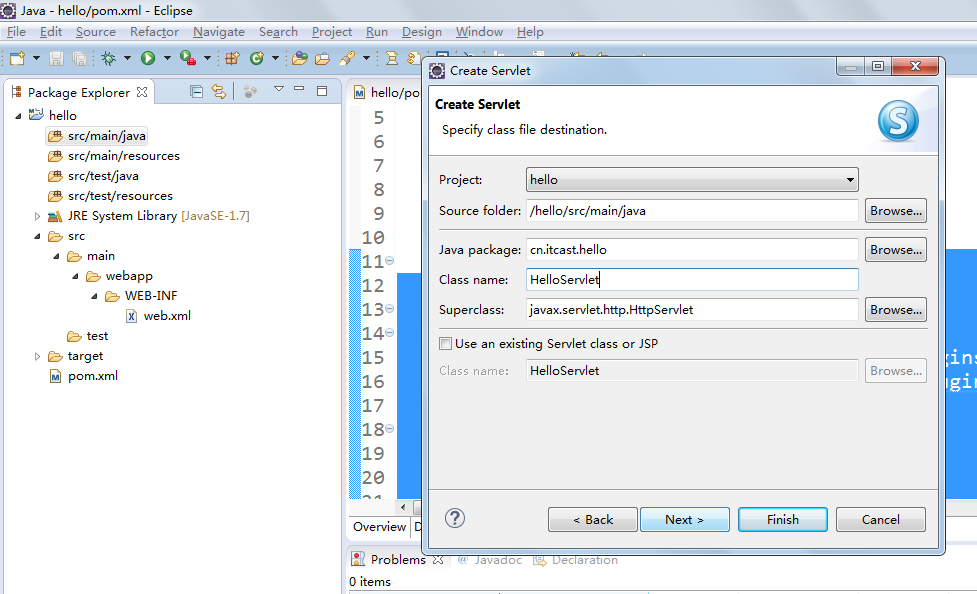
</plugins>

</build>

maven是个项目管理工具，如果我们不告诉它我们的代码要使用什么样的jdk版本编译的话，它就会用maven-compiler-plugin默认的jdk版本来进行处理，这样就容易出现版本不匹配，以至于可能导致编译不通过的问题。

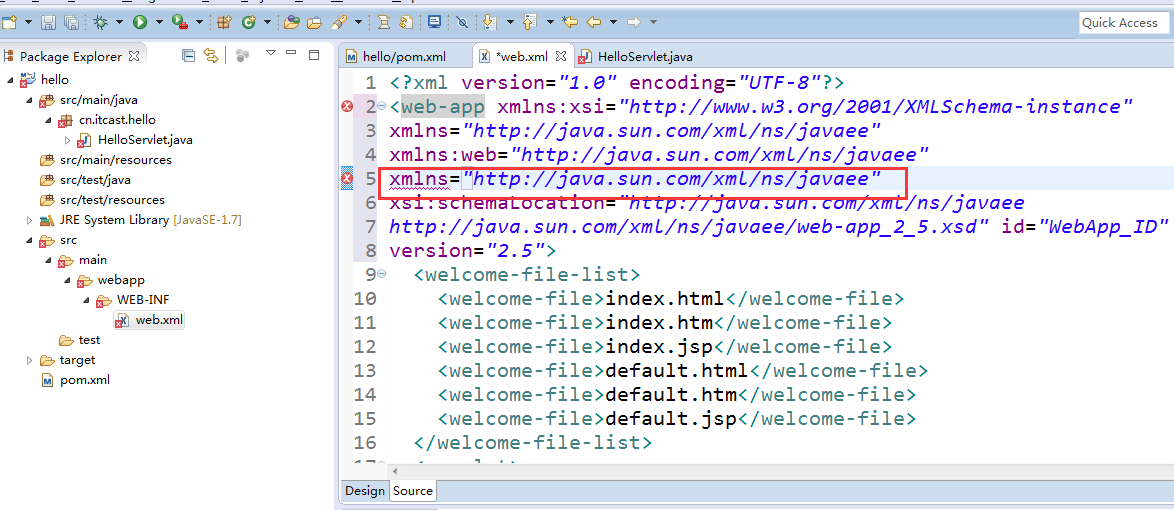
参考网址：<https://blog.csdn.net/liupeifeng3514/article/details/80236077>

3、创建一个servlet



修改web.xml

删除下图红框中的重复的代码(有些版本会出现的情况)



## 添加jar包

在pom中添加如下代码：

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

<scope>test</scope>

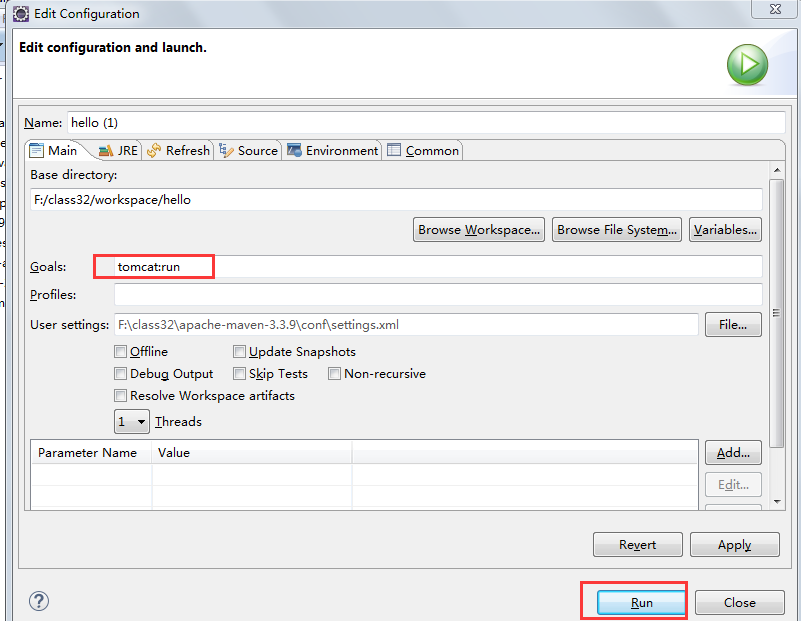
</dependency>

。。。

</dependencies>

## 启动项目

**右击项目---run as –maven build ….**



# 依赖范围

## Compile范围

默认范围。**编译、测试、运行都需要的jar包。**由于运行时需要所以该范围下的jar包会被打包。

例如：spring-core。

## Provided范围（重点关注）

**编译，测试时需要。运行时不需要所以不会被打包。**

例如：jsp-api.jar servlet-api.jar（tomcat容器中有这两个jar包，如果打包了这两个到war包中，会和tomcat冲突。）

## Runtime范围

**编译时不需要，测试时需要，运行时需要所以会被打包**

**例如：数据库驱动包**

**但是这种运行时需要，最终还是需要打包的，建议直接写成compile范围。**

## Test 范围

**编译时不需要，测试时需要，运行时不需要不会被打包**

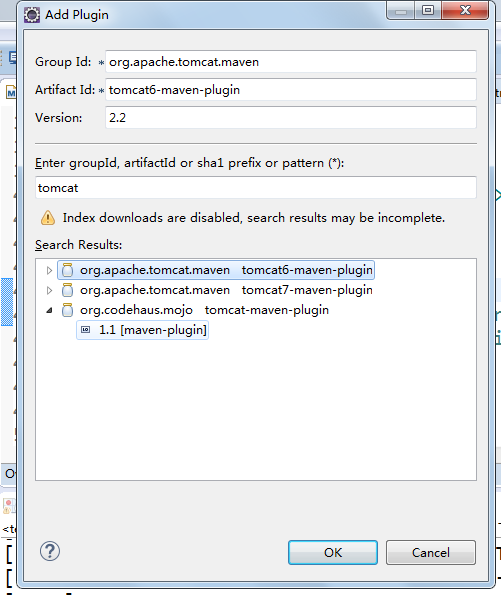
**例如：**junit.jar

**总结：test范围和runtime范围的直接写成compile范围的没啥关系。但是provide范围的不能写compile范围的，会冲突。**

如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Compile** | **Provide** | **Runtime** | **Test** |
| **编译** | **Ｙ** | **Ｙ** |  |  |
| **测试** | **Ｙ** | **Ｙ** | **Ｙ** | **Ｙ** |
| **运行（打包）** | **Ｙ** |  | **Ｙ** |  |

# add plugin（添加插件）



**poject object model**

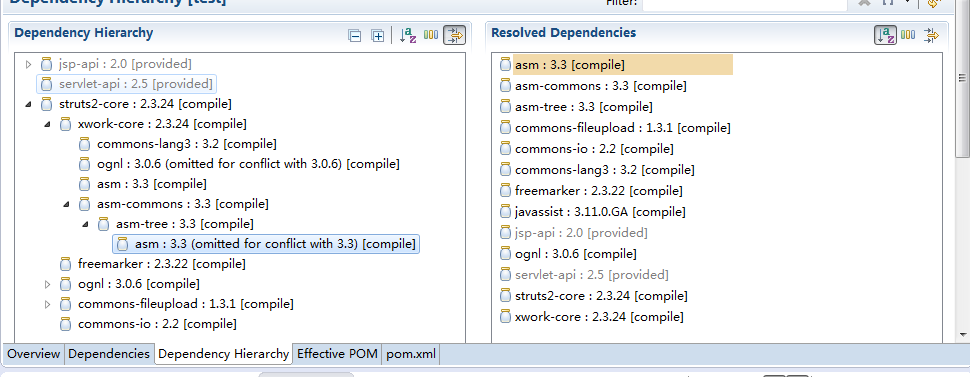


|  |
| --- |
| 添加插件和加入jar包的不同：添加插件一般是修改maven默认的配置，比如：jdk的版本，更改tomcat的配置，访问路径等等这样的操作。 |

# 依赖传递

只添加了一个struts2-core依赖，发现项目中出现了很多jar，

这种情况叫**依赖传递**



依赖传递或者导入版本不同的jar包时会出现“版本冲突”的问题。

解决方式：

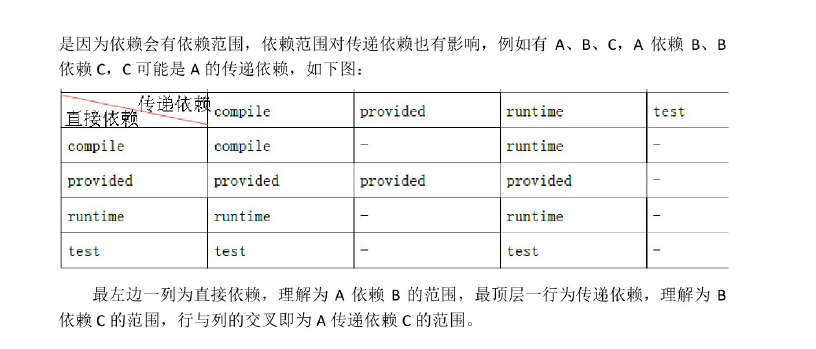
1. 版本锁定

|  |
| --- |
| 一般情况，会在pom文件上面“锁定版本”，这是最常用的解决版本冲突做法。  ”版本锁定”配置如下：  <**dependencyManagement**>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-bean</artifactId>  <version>4.2.4.RELEASE</version>  </dependency>  </dependencies>  </dependencyManagement>  **注意：1.版本锁定只是起到指定依赖版本的作用，并不会导入依赖。**  **2.<dependencyManagement>中的<dependencies>和导入jar包的< dependencies>标签不一样，前者是用于版本锁定，后者用于导入jar包。**  **多数时候还会有properties标签来抽取出常量信息（如版本号），配置如下：**  <properties>  **<spring.version>**4.2.4.RELEASE</spring.version>  　　<hibernate.version>5.0.7.Final</hibernate.version>  　　<struts.version>2.3.24</struts.verson>  </properties>  里面的标签是自定义的**。**  则上面的**version直接引用标签**即可，调用方式类似EL表达式：  <version>${spring.version}</version> |

1. 版本依赖的优先原则：首先，先考虑直接依赖的，再考虑传递依赖的；再者，先考虑排在文件上面的。
2. 排除传递依赖(将不需要的传递依赖的包去除掉)

|  |
| --- |
| https://images2018.cnblogs.com/blog/1026825/201804/1026825-20180419231309428-47365921.png |

依赖范围对依赖传递造成的影响**（了解）**



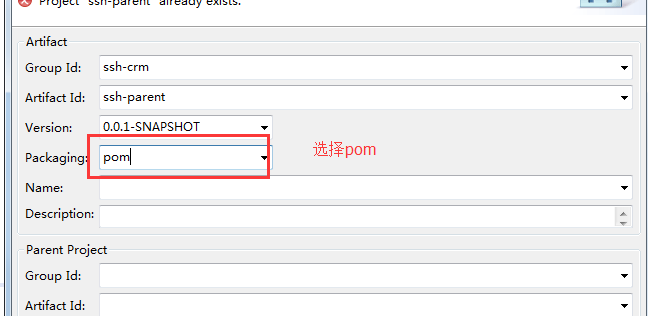
# 分模块开发

## pom、jar、war的区别（重点）

|  |
| --- |
| 1.pom工程：父级工程或聚合工程时使用。主要用来做**jar包的版本控制**。  2.war工程：将会打包成war，**发布在服务器上的工程**。如网站或服务。  3.jar工程：**将会打包成jar用作jar包使用**。 |

## 创建父工程

1、创建父工程需要选择pom



2、创建出的父工程如下



3、在父工程的pom.Xml添加项目需要用到的jar包。

## 创建dao子模块（jar）

1、在ssh-parent项目上右击，**创建子工程时选择 Maven Module,并打包成jar包**

2、填写子模块名称ssh-dao

3、把属于dao的代码拷贝到该模块中：

**4、完成后发布到本地仓库中**

## 创建service子模块（jar）

1、创建方式如上，**打包成jar包**

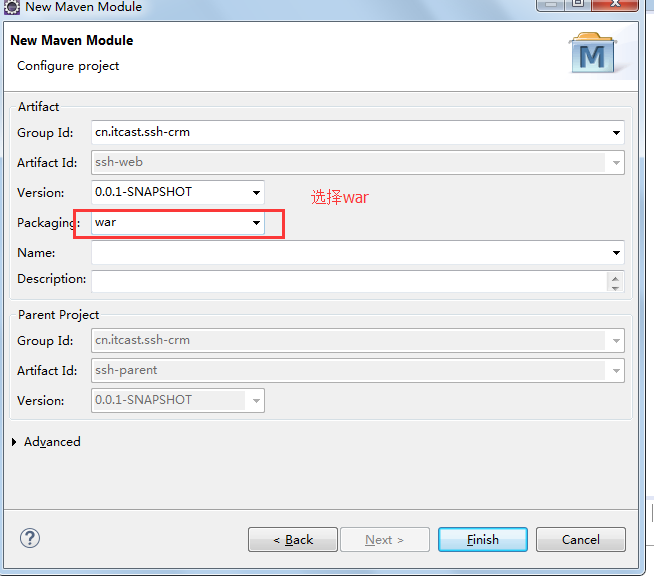
2、把属于service的代码拷贝到该工程中

**3、发布到本地仓库中**

|  |
| --- |
| 注意：   1. Service模块会用到dao模块的配置文件，此时需要把dao模块先关闭，这样service模块调用的就是dao模块的jar包形式，而不是文件夹形式，这样就可以引用dao模块的配置文件。 2. 引用jar包中的配置文件时：   要么直接写“classpath:applicationcontext-dao.xml”全路径的形式，  如果用\*通配符时，需要在classpath后面加\*，即：**classpath\*:applicationcontext-\*.xml** |

## 创建web子模块(war)

1、**选择war的打包方式（dao和service都是打包成jar，因为需要依赖注入，而web层是最后一层，是要发布的，所以打包成war。）**



2.最后运行：运行web项目或parent项目都是可以的。

## 可能出现的问题

1. An error occurred at line: 1 in the generated java file问题处理：

tomcat6启动后，加载jsp页面报错，提示无法将jsp编译为class文件。

原因为：tomcat6不支持jdk1.8版本。

解决办法：

更改tomcat插件为tomcat7。

|  |
| --- |
| 补充：  tomcat:deploy   --部署一个web war包  tomcat:reload   --重新加载web war包  tomcat:start    --启动tomcat  tomcat:stop    --停止tomcat  tomcat:undeploy--停止一个war包  tomcat:run  启动嵌入式tomcat ，并运行当前项目  **不可以随便输入，要与自己的tomcat版本匹配。如tomcat7：run** |

1. Maven内置的tomcat插件版本：只有6和7。

开发service层的团队怎么拿到开发dao层的团队所打包的jar包，以及web怎么拿到前两者的？就要用到私服了。

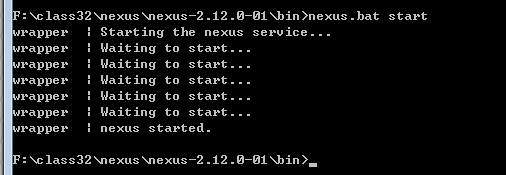
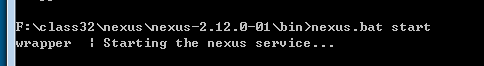
# 私服 nexus（了解）

可以简单“百度云”。在企业中的作用：在局域网内共享。

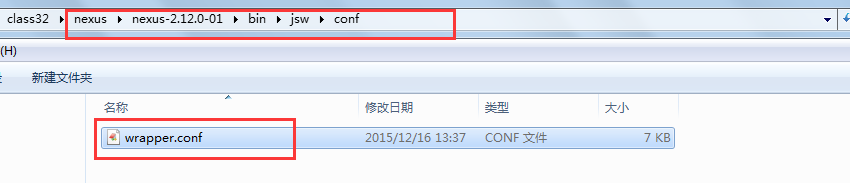
安装nexus

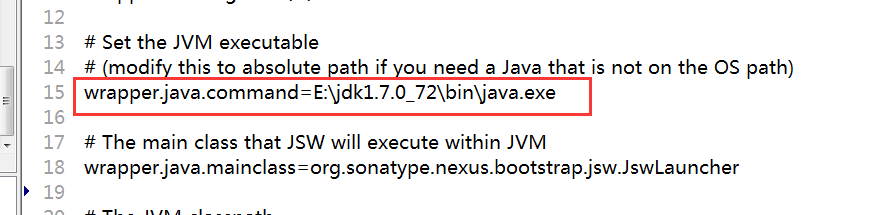


启动服务



启动失败的解决方法：





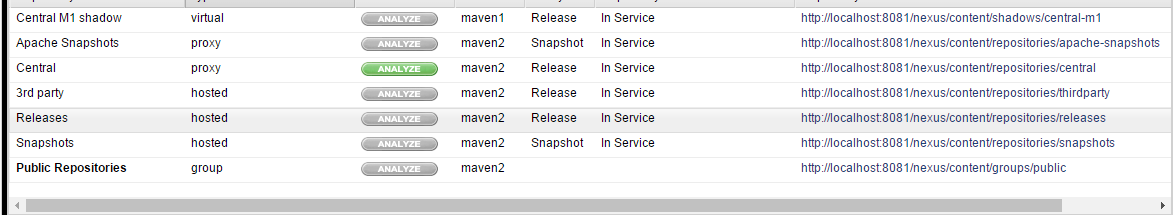
登录nexus

用户名/密码admin/admin123

Nexus Bundle 默认使用的端口号是：8081，如果启动服务可以访问：

http://localhost:8081/nexus

仓库类型



Virtual 虚拟仓库

Proxy 代理仓库

Hosted 宿主仓库本地仓库

Group 组

需求：

把dao放到私服上，然后service从私服上下载

## 上传dao

第一步：需要在客户端即部署dao工程的电脑上配置 maven环境，并修改 settings.xml 文件，配置连接私服的用户和密码。

此用户名和密码用于私服校验，因为私服需要知道上传都的账号和密码是否和私服中的账号和密码一致。

<server>

<id>releases</id>

<username>admin</username>

<password>admin123</password>

</server>

<server>

<id>snapshots</id>

<username>admin</username>

<password>admin123</password>

</server>

第二步：配置项目pom.xml

配置私服仓库的地址，本公司的自己的jar包会上传到私服的宿主仓库，根据工程的版本号决定上传到哪个宿主仓库，如果版本为release则上传到私服的release仓库，如果版本为snapshot则上传到私服的snapshot仓库

<distributionManagement>

<repository>

<id>releases</id>

<url>http://localhost:8081/nexus/content/repositories/releases/</url>

</repository>

<snapshotRepository>

<id>snapshots</id>

<url>http://localhost:8081/nexus/content/repositories/snapshots/</url>

</snapshotRepository>

</distributionManagement>

注意：pom.xml这里<id>和 settings.xml 配置<id>对应！

**第三步：执行deploy命令发布到私服**

## 下载dao

第一步修改settings.xml

<profile>

<!--profile的id-->

<id>dev</id>

<repositories>

<repository>

<!--仓库id，repositories可以配置多个仓库，保证id不重复-->

<id>nexus</id>

<!--仓库地址，即nexus仓库组的地址-->

<url>http://localhost:8081/nexus/content/groups/public/</url>

<!--是否下载releases构件-->

<releases>

<enabled>true</enabled>

</releases>

<!--是否下载snapshots构件-->

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<!--插件仓库，maven的运行依赖插件，也需要从私服下载插件 -->

<pluginRepository>

<!--插件仓库的id不允许重复，如果重复后边配置会覆盖前边 -->

<id>public</id>

<name>Public Repositories</name>

<url>http://localhost:8081/nexus/content/groups/public/</url>

</pluginRepository>

</pluginRepositories>

</profile>

<activeProfiles>

<activeProfile>dev</activeProfile>

</activeProfiles>

第二步删除本地仓库中的dao

第三步 update service工程，出现以下信息说明已经成功

