# 1.简介

maven是JAVA的依赖构建管理工具，这里的依赖和Spring中的依赖概念相同，但指的并不是同一个东西，一个项目中常常要用到第三方包，这样可以大幅加快我们的开发进度，但是第三方包往往会产生烦人的依赖链，比如包A是依赖于包B的，想要使用A的功能，必须先配置B，这就组成了依赖链，一旦依赖链过长，那就是一场灾难，而maven有效的帮助我们解决了这一问题，maven实际上是在**项目对象模型(pom.xml)**中声明项目所需要的依赖包以及版本，maven会自动帮我们配置依赖链，只需要告诉它你需要什么即可，不要关心更多细节

# 2.maven和cvs

maven是包依赖管理工具，他和git、svn等cvs版本控制工具不同，它是我们管理第三方代码的工具，而cvs软件是为我们管理我们自己写的代码的工具

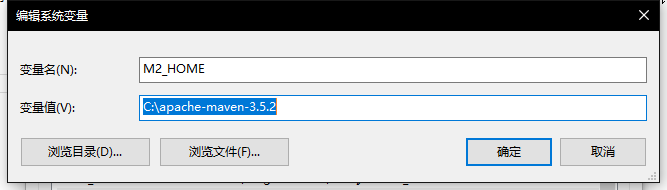
maven的诸多依赖包一般不使用git或svn等cvs工具管理，只需要将maven的**项目对象模型(pom.xml)**管理起来即可，就好像你到外地看病，不需要把药买好带在身上，只需要把药方记着即可，javaEE中几乎每一个项目都会使用maven进行包依赖管理

# 3.maven安装和配置

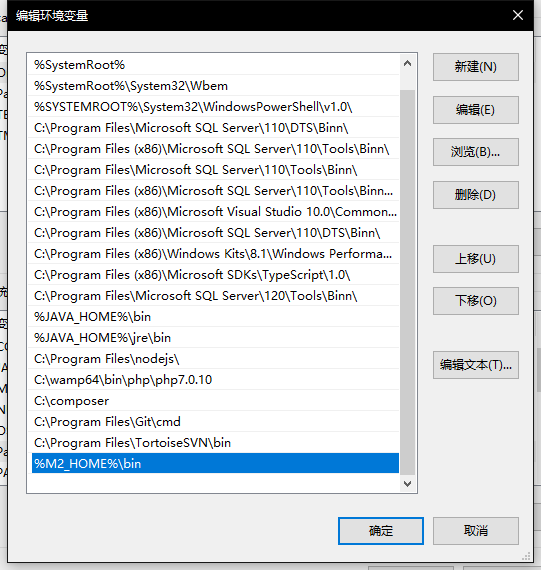
## 3.1安装

打开<https://maven.apache.org/download.cgi> ，推荐选择 Binary zip和link下载，解压后需要配置相应的环境变量

打开windows环境变量界面，新增系统变量M2\_HOME，变量值就是解压目录



然后修改系统变量path，在path原来的值后面新增一行**%M2\_HOME%\bin**



然后测试一下是否安装成功，在cmd命令行下输入mvn –version，可以看到当前maven版本



至此maven安装成功

## 3.2配置

为了防止以后maven升级还要重新配置，我们将maven配置文件**(conf/settings.xml)**放置到%HOME%目录下的.m2目录下(如果maven刚下载完还未第一次运行时没有.m2目录的，可以手动创建，也可以运行maven创建)，这样以后升级maven时也不需要修改配置了

PS: %HOME%是用户根目录，就是桌面上的{用户名}文件夹

settings.xml有以下常用的xml元素节点

**1.localRepository**:它是maven本地仓库路径设置，在maven中主要有两种包仓库：

**(1)remote repository：**相当于公共的仓库，任何人都能通过网络访问到，一般可以用URL的形式访问,一般默认的地址：[http://search.maven.org/](http://search.maven.org/" \o "http://search.maven.org/" \t "_blank)

**(2)local repository：**存放在本地磁盘的一个文件夹，是maven的中央仓库，也是你依赖包存储的位置，例如，windows上默认是C:\Users\｛用户名｝\.m2\repository目录，也可以通过**localRepository**自行修改.

eg：<localRepository>E:\Study\mavenRepository</localRepository>

**2. mirror:**它是maven镜像，在maven中，即使是本地的中央仓库，也需要从web上下载依赖包，仓库中才会有依赖，而maven官方的中心仓库网站因为访问的人太多，不得不配置镜像站点进行分流，mirror节点就是给我们的maven配置镜像使用的

下面推荐一个阿里巴巴的maven镜像站点

eg：

<mirror>

<id>nexus-aliyun</id>

<mirrorOf>\*</mirrorOf>

<name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>

**集成maven到MyEclipse**

为了方便使用，我们将安装好的maven集成到编译器中，依次点击windows->preference->Maven4MyEclipse->Installations->add->选择maven路径->勾选新增的刚安装的maven->apply-> preference左侧边栏User settings->查看settings文件是否为自己存放的位置->确定

配置完成后，我们就能通过cmd命令行使用maven命令创建springMVC了

# 4.maven创建web项目

## 4.1创建maven项目

打开cmd命令行，cd进入项目根目录的将要存放的位置，输入：

mvn archetype:generate -DgroupId=com.mvc -DartifactId=mySpringMVC -DarchetypeArtifactId=maven-archetype-webapp

**mvn调用maven运行**

**archetype:generate调用**archetype Plugin**插件**

**DgroupId是项目的唯一识别码，相当于数据库的主键id，一般使用域名反写命名，如项目是baidu.com，那么一般就写com.baidu**

**DartifactId简单地当做是项目根目录的目录名就行了**

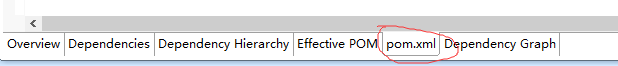
**DarchetypeArtifactId指明了我们创建的项目类型**

通过上述maven命令配置好maven项目后，需要通过import maven project导入编译器

## 4.2项目对象模型pom.xml

pom本质上就是一个xml配置文件，它是我们整个项目所需依赖包的”清单”， 一个完整的pom.xml文件，放置在项目的根目录下。

此时，jsp文件会报错，没关系打开pom.xml文件，将package类型由war改为jar并保存，然后切换显示方式



在<dependencies>下添加节点：

<dependency>

<groupId>javax</groupId>

<artifactId>javaee-api</artifactId>

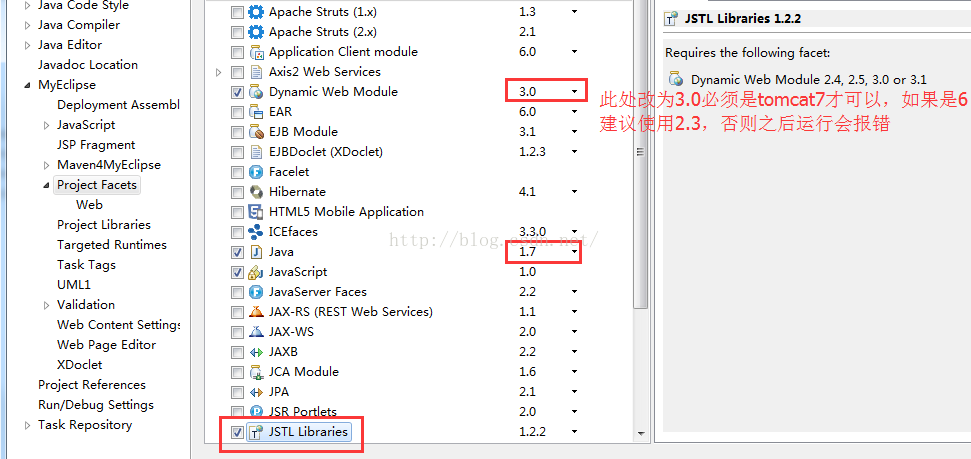
<version>7.0</version>

<scope>provided</scope>

</dependency>

jsp文件不再报错

然后右键项目->properties->搜索project facets



## 4.2添加Source Folder

Maven规定，必须在项目根目录下创建以下几个Source Folder

**src/main/resources**

**src/main/java**

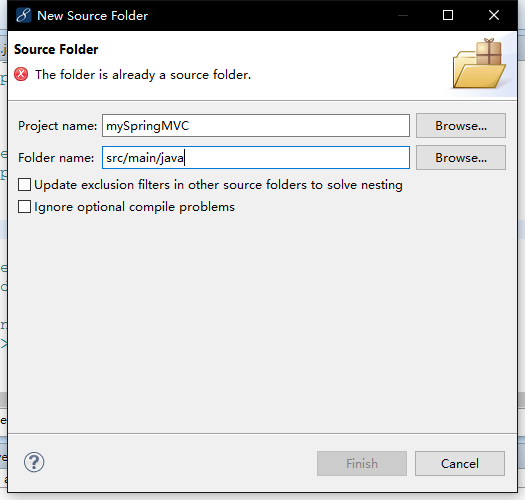
**src/test/resources**

**src/test/java**

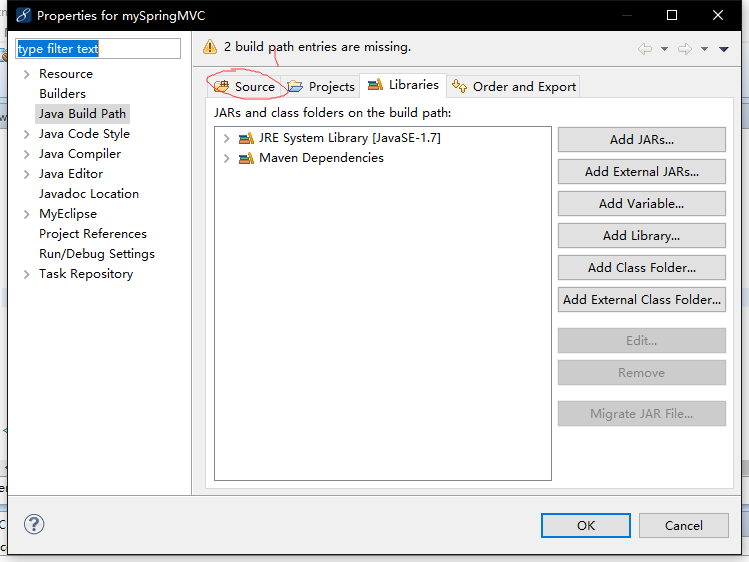
Source folder是存放源文件的文件夹的意思。

所谓源文件就是你所有的以.java为后缀的文件，就是java程序最原始未编译的形态，需要和.class文件分开存放，一般存放在src文件夹里（source 的缩写）。

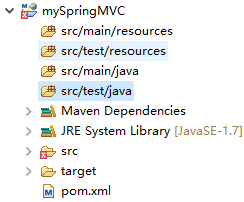
如果创建上述的两个java资源文件**src/main/java、src/test/java**出错：



则右击改工程的Properties属性选中Java Build Path项，找到source，发现有两个丢失(missing)的资源包，将其选中remove掉，就可以继续创建了(别忘了remove后要点确定)；

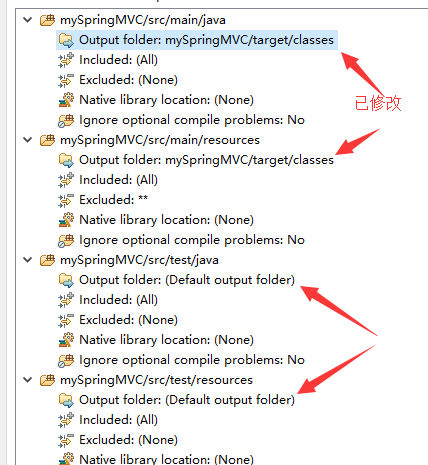


最终创建结果为：



## 4.3修改配置

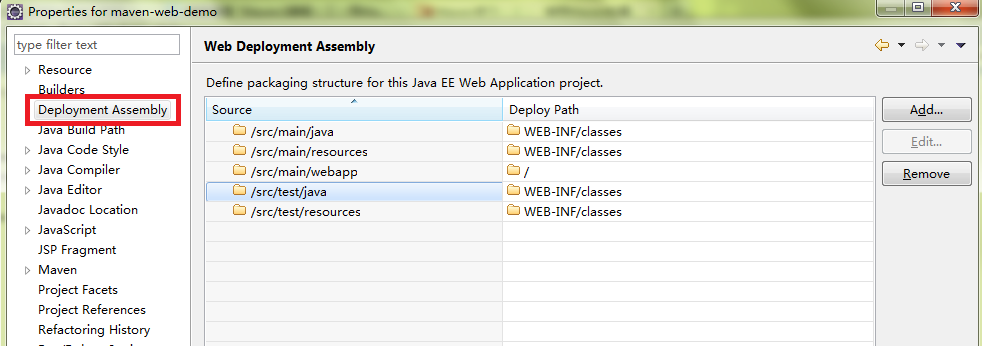
设定4个文件夹的输出Output folder，双击修改，分别改为target/classes和target/test-classes



别忘了点确定

然后设置部署程序集，在项目上右键单击，选择Properties，在左侧选择Deployment Assembly

我们删除test的两项，因为test是测试使用，并不需要部署。



最后，在src文件夹下

至此，一个空的maven项目构建完成：

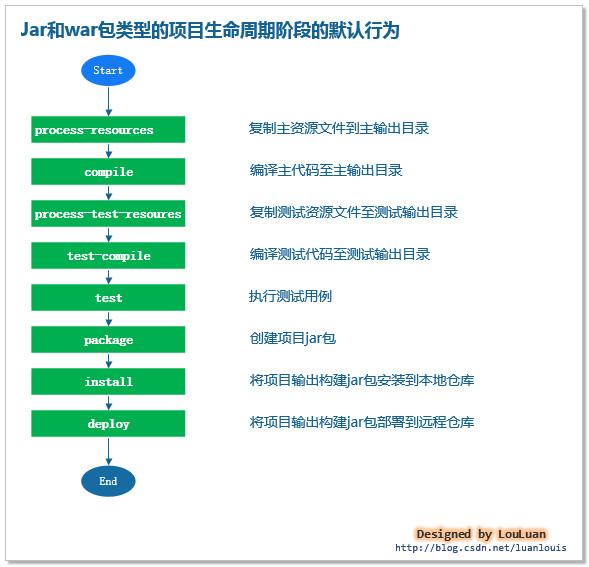
# 5.maven的项目周期

学习一样新东西，永远都是先了解其生命周期，maven项目也不例外



在实际的项目开发过程中， 并不是上述所有的生命周期阶段都是必须的。

然而，在实际的开发过程中，往往我们的项目的一些生命周期的阶段不需要相应的行为，我们只需要关心其中某些重要的生命周期阶段而已。下面，请看一下日常开发中，我们需要关注的生命周期阶段，即广大开发人员对项目周期阶段处理的约定：  
**1).应该将resource资源文件准备好，放到指定的target目录下----process-resources 阶段；  
2).将java源文件编译成.class文件，然后将class 文件放置到对应的target目录下----compile阶段；  
3).将test类型的resource移动到指定的 target目录下------process-test-resource阶段；  
4).将test类型的java 源文件编译成class文件，然后放置到指定的target目录下------test-compile阶段；  
5).运行test测试用例-------test阶段；  
6).将compile阶段编译的class文件和resource资源打包成jar包或war包--------package阶段；  
7).将生成的包安装到本地仓库中------install阶段  
8).将生成的包部署到远程仓库中-----deploy阶段**



maven项目的每一个目录都有相应的作用：



# 6.pom.xml

pom.xml相当于我们的弹药库，当我们需要某个包时，只需要”告诉”pom.xml，我需要某某某版本的某某某包，可以使用maven根据pom.xml”清单”上声明的依赖进行依赖构建，非常方便

## 6.1 基础节点

这些节点在maven项目创建后基本为我们自动建好，多数情况下不需要额外修改，但需要了解相关含义：

<modelVersion>4.0.0</modelVersion>

<groupId>com.mvc</groupId>

<artifactId>mySpringMVC</artifactId>

<version>1.0-SNAPSHOT</version>

<name>mySpringMVC Maven Webapp</name>

<url>http://maven.apache.org</url>

**groupId**:项目或者组织的唯一标志，并且配置时生成路径也是由此生成，如org.myproject.mojo生成的相对路径为：/org/myproject/mojo

**artifactId**:项目的通用名称

**version**:项目的版本,注意：这是我们自己要写的项目的版本，具体多少自己定

**packaging**:打包机制，如pom,jar,maven-plugin,ejb,war,ear,rar,par

**name**:用户描述项目的名称，无关紧要的东西，可选

**url**:应该是只是写明开发团队的网站，无关紧要，可选

**classifer**:分类

**其中groupId,artifactId,version,packaging这四项组成了项目的唯一坐标。一般情况下，前面三项就可以组成项目的唯一坐标了。也就是说，不管是自己的项目还是别人的项目，只要通过groupId,artifactId,version，就能找得到，就像地图上的坐标一样**

## 6.2 dependencies节点

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<type>jar</type>

<scope>test</scope>

</dependency>

<dependencies>节点下面有诸多的<dependency>节点，这是实际声明我们需要哪些依赖的地方，每一个<dependency>节点都是一个依赖包，其中groupId, artifactId, version这三个组合标示依赖的具体工程(我们导入的依赖包其实也是别人做好的项目，也有自己的唯一坐标)，一般来说，声明的依赖只要是在maven中央仓库(我们上面配置的是阿里的镜像仓库)存在的开源项目都可以被导入，如果是非开源包也有处理方式，这里不再赘述

<type>声明要导入的第三方依赖包的类型，一般都是jar包

<scope>是用来限制Dependency的作用范围的, 影响maven项目在各个生命周期时导入的package的状态。现在有了6种**scope：**

**1.compile**

**默认的scope，表示 dependency可以在项目的各个生命周期中使用。**在编译和打包时都会将依赖存储进去

**2.provided**

在编译和测试的过程有效，最后生打包时不会加入，诸如：servlet-api，因为servlet-api，tomcat等web服务器已经存在了，如果再打包会冲突**使用这个时，不会将包打入本项目中，只是依赖过来。也就是项目的实际目录中并不会有这个包，只是引了”链接”过来，线上也不会有，因为你打包时都没加，但是编译和测试时是可以用这个包的，**

**3.test**

test范围指的是测试范围有效，在编译和打包时都不会使用这个依赖，也就是说最终上限的项目不会有这个包，Junit测试框架就是一个典型的例子，在开发时，我们需要Junit框架来帮助我们编写测试用例，而项目上线时，我们则不再需要Junit。

**4.runtime**

在运行和测试的时候依赖，在编译的时候不依赖

**5.system**

类似provided，**在系统中**需**要以外部JAR包的形式**提供依赖，Maven不会在Repository中查找它，eg：

<scope>system</scope>

<systemPath>${java.home}/lib/rt.jar</systemPath>

**最后一个import初级阶段不常用**

## 6.3 properties节点

该节点一般用于帮助我们进行依赖版本控制的，如：

<properties>

<spring.version>3.2.5.RELEASE</spring.version>

</properties>

上述<properties>节点是帮助我们管理依赖版本的，试想一下，spring有诸多的依赖包，如果我们像这样引入spring中的spring-core和spring-aop：

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>1.2.6</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>1.2.6</version>

</dependency>

那么如果我们需要更换spring版本时，需要手动地将这这两个包的<version>1.2.6</version>改掉，既不方便，又容易出错，可如果我们事先定义了一个属性<properties>通过占位符(${spring.version})来帮助我们管理spring版本，那么就方便多了，如：

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${spring.version}</version>

</dependency>

这样如果更换版本只需要更改<properties>节点即可

## 6.4 安装所需依赖

# 7.maven多模块项目

## 6.1 为什么要划分多模块

在实际开发中，为了降低项目耦合度和复杂度，常常需要将一个项目划分多个模块，每个模块对应一个pom.xml，彼此之间通过继承和聚合相互关联

我们假设有这样一个商城项目，包括以下几个模块：

商城前台（shop）

管理后台（admin）

数据库交互模块（dao）

通用业务模块（service）

接口模块（api）

通用工具（util）

其中shop和admin需要单独部署，dao、service、util你可能想要一些经验丰富的人来维护，如果使用一个应用来管理的话，所有的功能和模块都会耦合在一起，所有人都可以随意修改代码，这显然不是我们所期望的。

而且使用一个应用来管理的话，任何一个点的代码有变更，整个项目就需要重新build打包，使用模块化开发的另一个好处是如果dao的代码被修改，只需要重新build dao模块就可以了。web模块可以build成war，dao、service、util等可以build成jar，只需要配置好依赖关系，就可以实现模块间的解耦合。这样的设计才是遵循“高内聚，低耦合”设计原则的。

## 6.2 一种划分模块的方式

Maven模块划分有诸多不同的划分方式，但所有划分方式都需要有一个共有的父项目，该项目用于管理依赖，所有子项目的依赖都是继承自父项目，版本都与父项目保持一致(子项目pom.xml中不得声明版本)

并且，我们希望把DAO和service作为一个通用的底层工具使用

**---------- mall**  //顶级项目

|------------ pom.xml //packaging = pom

**|------------ mall-util** //通用工具类

| |--------------- pom.xml //packaging = jar

**|------------ mall-core**  //包含业务逻辑和数据持久层模块

| |--------------- pom.xml //packaging = jar

**|------------ mall-web-api**  //接口模块

| |--------------- pom.xml //packaging = war

**|------------ mall-web-admin** //商城管理后台表示层

| |--------------- pom.xml //packaging = war

**|------------- mall-web-front** //商城前台表示层

| |--------------- pom.xml //packaging = war

这些模块中api、admin、shop均是可以单独部署的web应用，相互之间没有依赖关系，但都依赖于core模块(service层和dao层)，而core模块依赖于util模块。

参考<https://blog.csdn.net/u011404265/article/details/54891229>

PS：

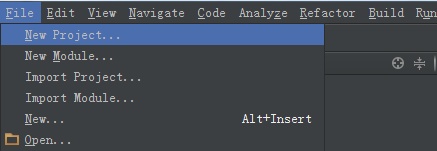
通常一个jar包就是一个java程序或者一个java库。

War包是Java的Web应用程序的包，tomcat这种Servlet容器会认出war包并自动部署。

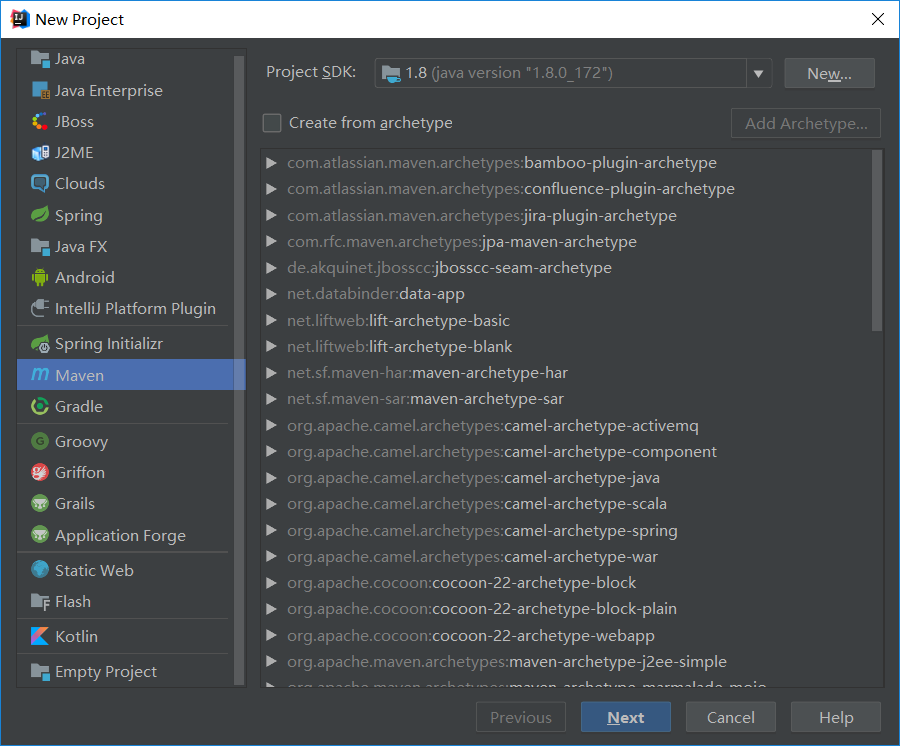
## 具体划分

### 6.3.1创建一个普通maven项目

1.New Project

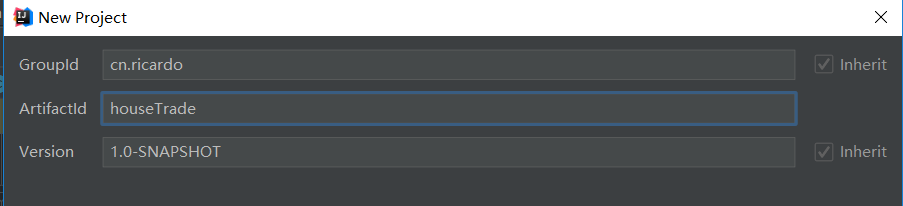


2.仅选择sdk即可



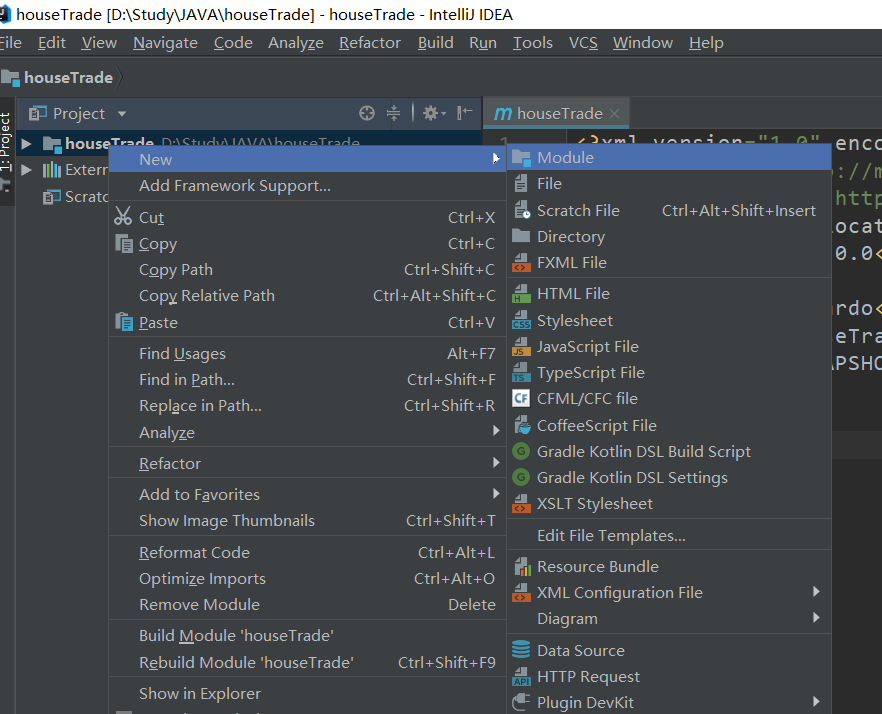
3. groupId一般分为多个段，这里只说两段，第一段为域，第二段为公司名称。域又分为org、com、cn等等许多，其中org为非营利组织，com为商业组织。

Eg：比如apache公司的tomcat项目：这个项目的groupId是org.apache，它的域是org（因为tomcat是非营利项目），公司名称是apache，artigactId是tomcat。

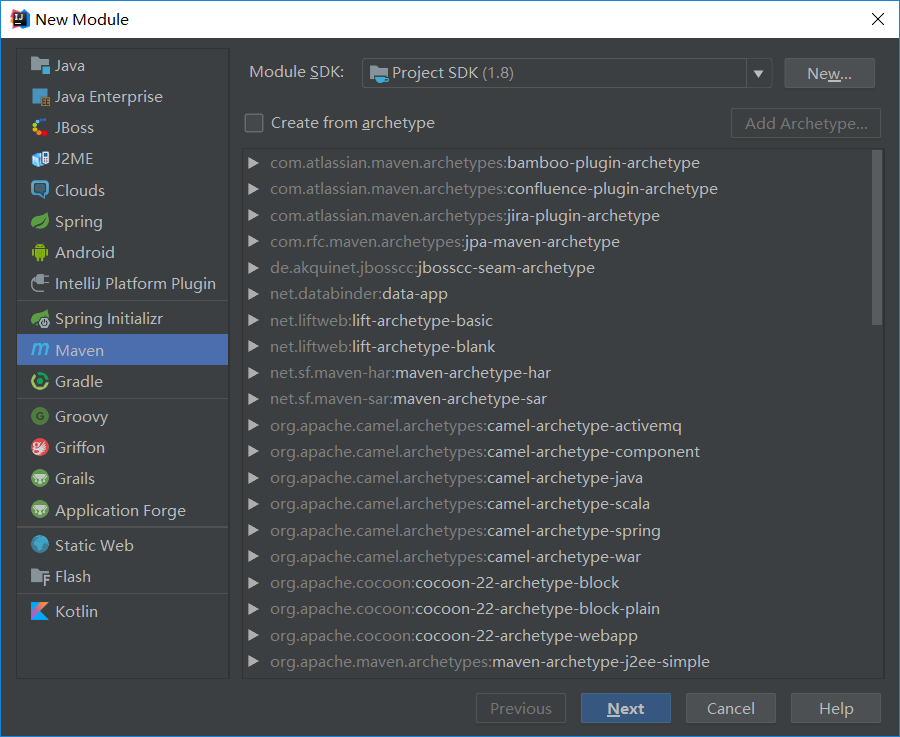


### 6.3.2添加模块

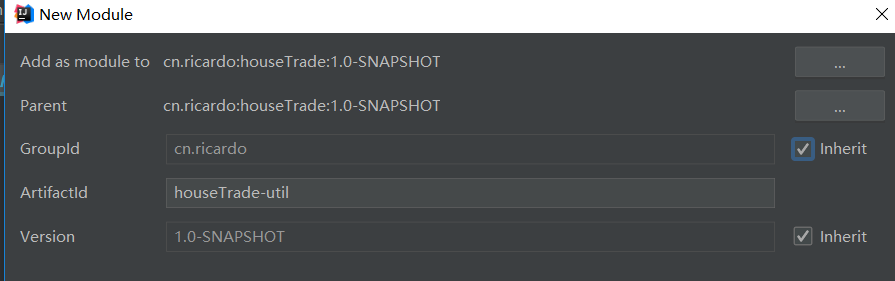
1.New Module

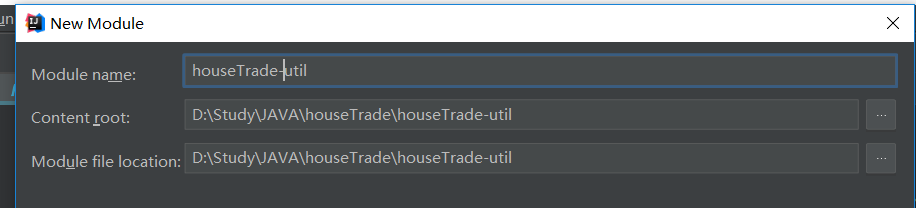


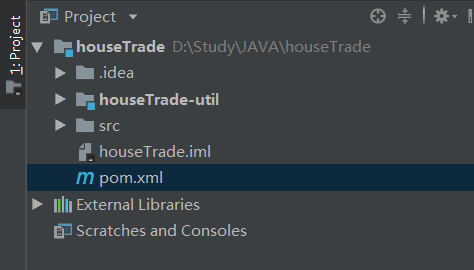
2.填写基本信息，同样不需要使用Maven模板(archetype)搭建



3.groupid保持一致，只改变aftifactid





4.finish之后，打开父项目的pom.xml，可以看到module已经被引入到父项目的pom文件里了  


<groupId>cn.ricardo</groupId>  
<artifactId>houseTrade</artifactId>  
<packaging>pom</packaging>  
<version>1.0-SNAPSHOT</version>  
<modules>  
 <module>houseTrade-util</module>  
</modules>

4.打开子模块的pom.xml，设置打包方式

<parent>

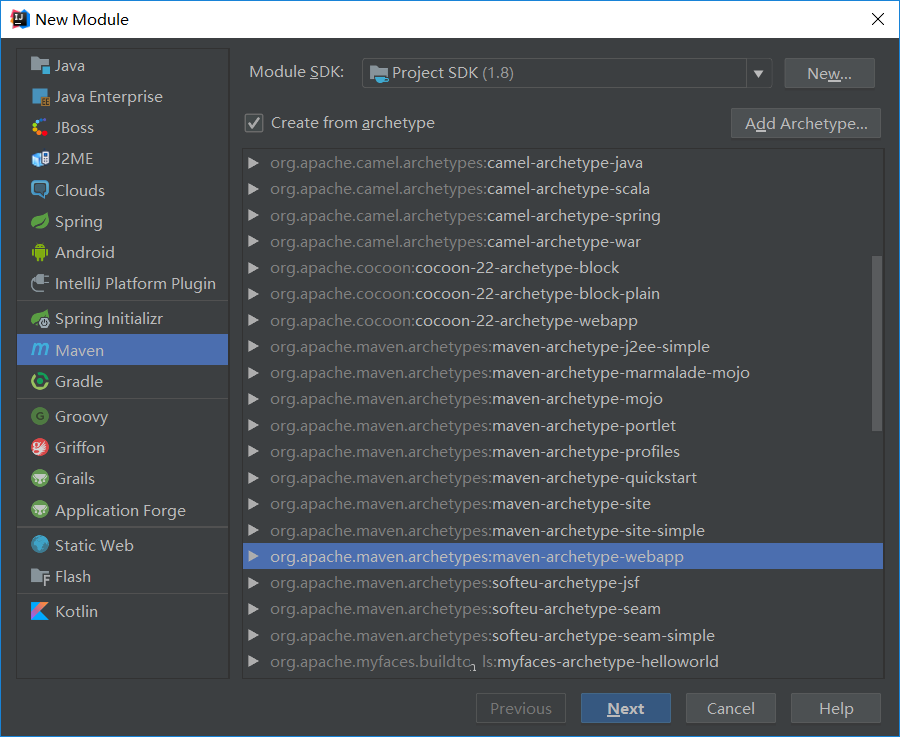
<groupId>cn.ricardo</groupId>  
 <artifactId>houseTrade</artifactId>  
 <version>1.0-SNAPSHOT</version>  
</parent>  
<modelVersion>4.0.0</modelVersion>  
  
<artifactId>houseTrade-util</artifactId>  
<packaging>jar</packaging>

5.修改url标签为项目主页的url

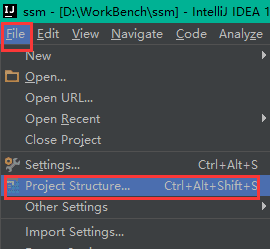
6.重复上述方式添加除了web模块外的其他模块(即core模块)

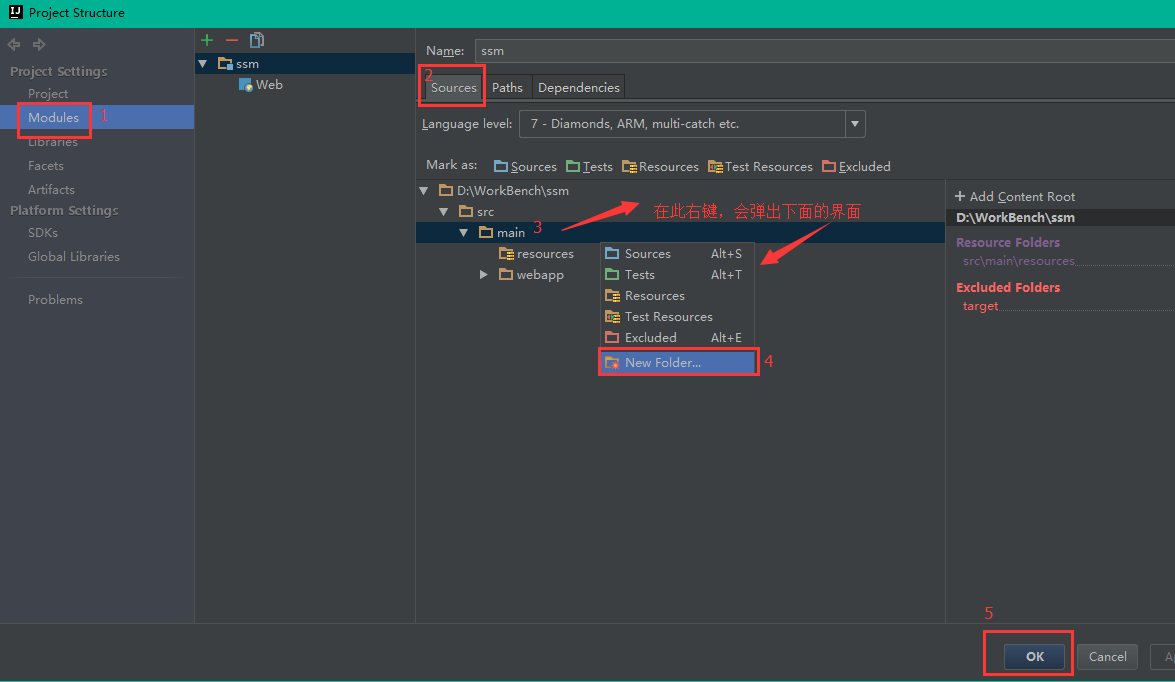
### 6.3.3添加web模块

1.new module，选中create from archetype，选择webapp



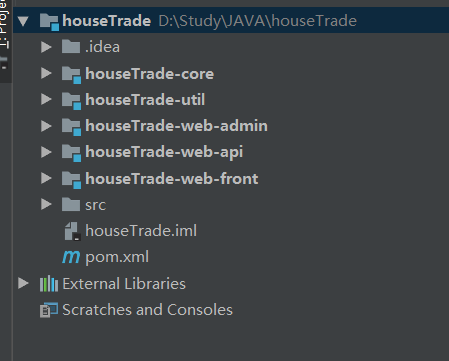
2.创建java和resource文件夹





2.重复6.3.2接下来的操作

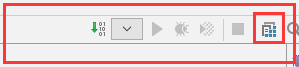
最终结果如图



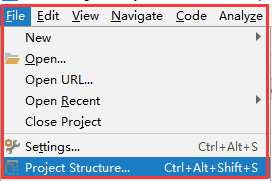
### 6.3.4配置项目

　1）打开“Project Structure”

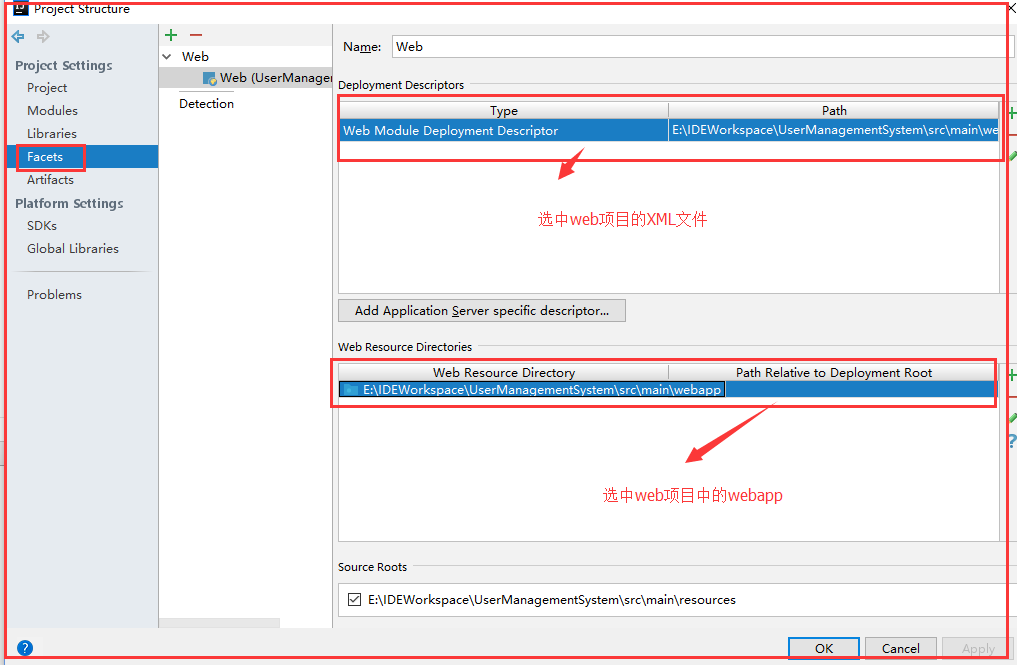
　　　　方式一：



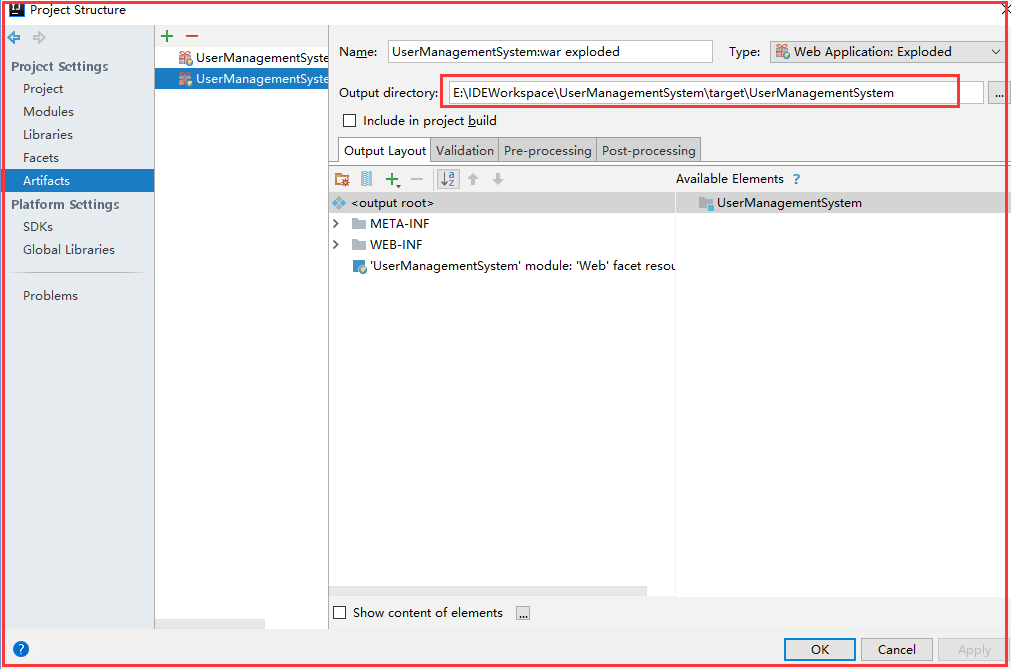
　　　　方式二：



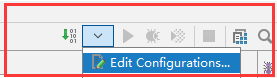
　　2）配置“Facets”



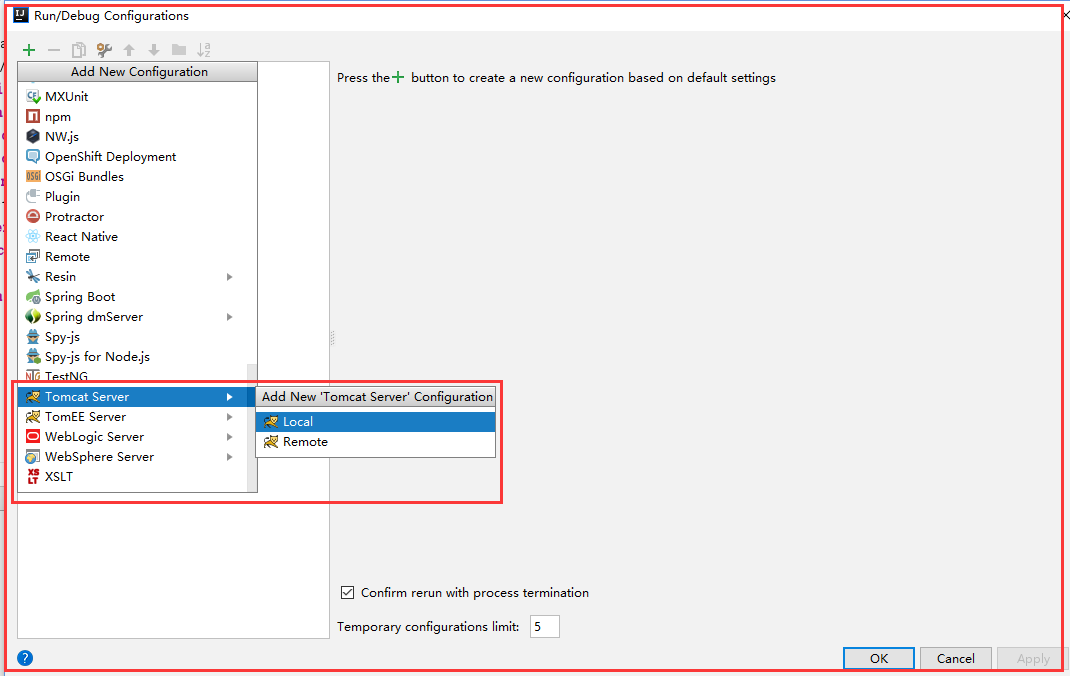
　　3）配置“Artifacts”



　　4）启动“Edit Configurations”

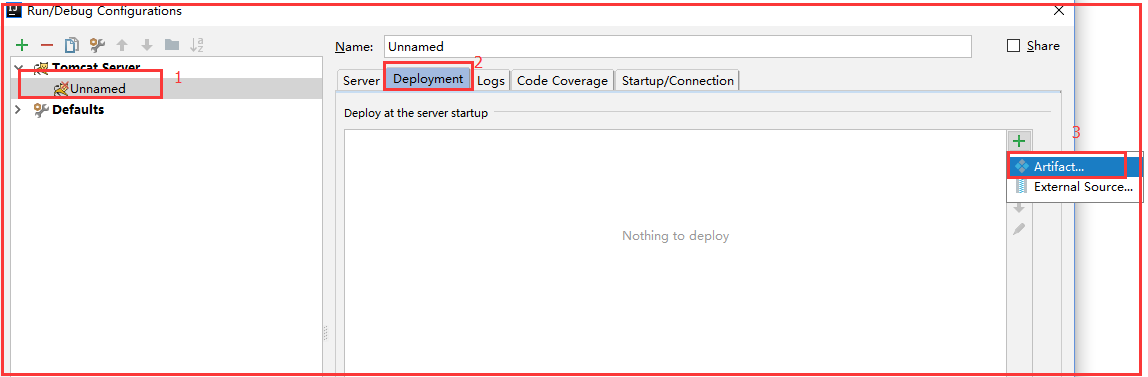


　　5）Add New 'Tomcat Server' 配置

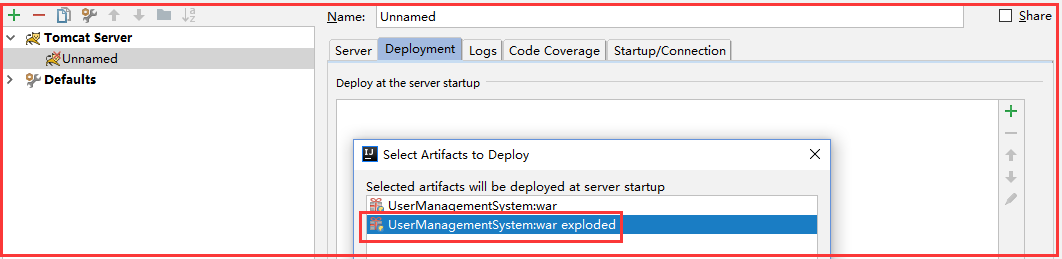


　　6）配置Deplyment

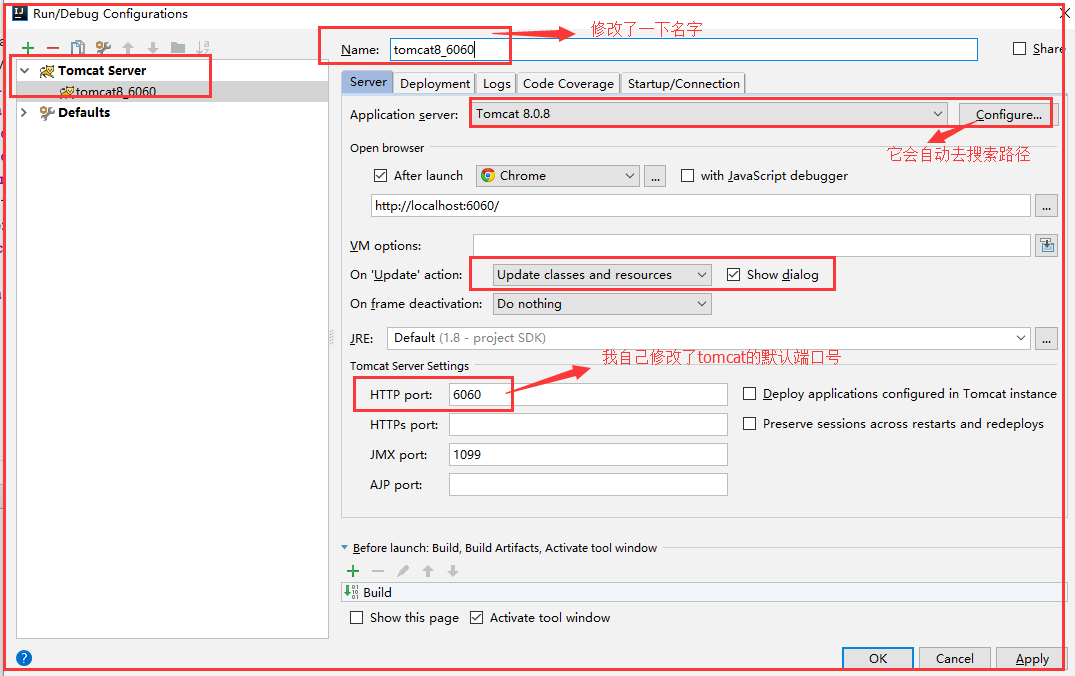
　　　　第一步：



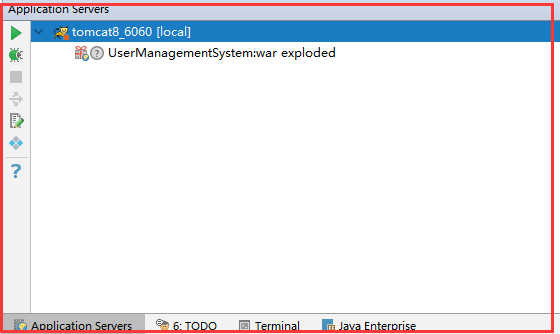
　　　　第二步：



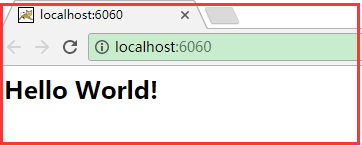
　　7）配置Server



　　8）配置结果



　　9）启动Web Server



### 6.3.5添加模块间的依赖关系

Web模块间没有依赖关系，但web模块依赖于core模块(service层和dao层)，而core模块依赖于util模块。

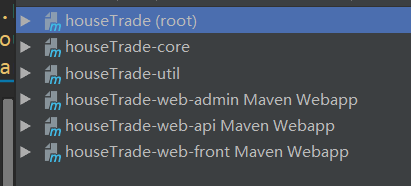
Core模块的pom.xml

<dependencies>  
 <dependency>  
 <artifactId>houseTrade-util</artifactId>  
 <groupId>cn.ricardo</groupId>  
 <version>1.0-SNAPSHOT</version>  
 </dependency>  
</dependencies>

三个web模块的pom.xml

<dependency>  
 <artifactId>houseTrade-core</artifactId>  
 <groupId>cn.ricardo</groupId>  
 <version>1.0-SNAPSHOT</version>  
</dependency>

### 6.3.6打包和发布



所有在root项目中进行的构建都会传递到模块中，例如root中的package会打包整个项目，当文件有变动时会进行重新聚合，其他命令同理。模块中的package只会打包当前模块。

使用source:jar命令会将源码打包。

web模块可以单独部署也可聚合部署。

### 6.3.7多模块依赖管理

上面有说到，我们所有的子模块都有一个父项目，这个父项目用来管理子项目的依赖，让所有的子项目使用依赖项的统一版本

在父项目的pom.xml文件中，有一个dependencyManagement元素，我们通过它来管理jar包的版本，让子项目中引用一个依赖而不用显示的列出版本号。Maven会沿着父子层次从下向上走，直到找到一个拥有dependencyManagement元素的项目，然后它就会使用在这个dependencyManagement元素中指定的版本号。

强烈建议不要在父项目的dependencyManagement的依赖声明中指定scope，scope应放在子项目中指定，子项目中不要再声明版本