Maven 基础课程第一天

第1章 Maven 介绍

1.1 什么是 Maven

1.1.1 什么是 Maven

Maven 的正确发音是[ˈmevən]，而不是“马瘟”以及其他什么瘟。Maven 在美国是一个口语化的词语，代表专家、内行的意思。
一个对 Maven 比较正式的定义是这么说的：Maven 是一个项目管理工具，它包含了一个**项目对象模型 (POM：Project Object Model)**，一组标准集合，一个项目生命周期(Project Lifecycle)，一个依赖管理系统(Dependency Management System)，和用来运行定义在生命周期阶段(phase)中插件(plugin)目标(goal)的逻辑。

1.1.2 Maven 能解决什么问题

可以用更通俗的方式来说明。我们知道，项目开发不仅仅是写写代码而已，期间会伴随着各种必不可少的事情要做，下面列举几个感受一下：

1、我们需要引用各种 jar 包，尤其是比较大的工程，引用的 jar 包往往有几十个乃至上百个， 每用到一种 jar 包，都需要手动引入工程目录，而且经常遇到各种让人抓狂的 jar 包冲突，版本冲突。

2、我们辛辛苦苦写好了 Java 文件，可是只懂 0 和 1 的白痴电脑却完全读不懂，需要将它编译成二进制字节码。好歹现在这项工作可以由各种集成开发工具帮我们完成，Eclipse、IDEA 等都可以将代码即时编译。当然，如果你嫌生命漫长，何不铺张，也可以用记事本来敲代码，然后用 javac 命令一个个地去编译，逗电脑玩。

3、世界上没有不存在 bug 的代码，计算机喜欢 bug 就和人们总是喜欢美女帅哥一样。为了追求美为了减少 bug，因此写完了代码，我们还要写一些单元测试，然后一个个的运行来检验代码质量。

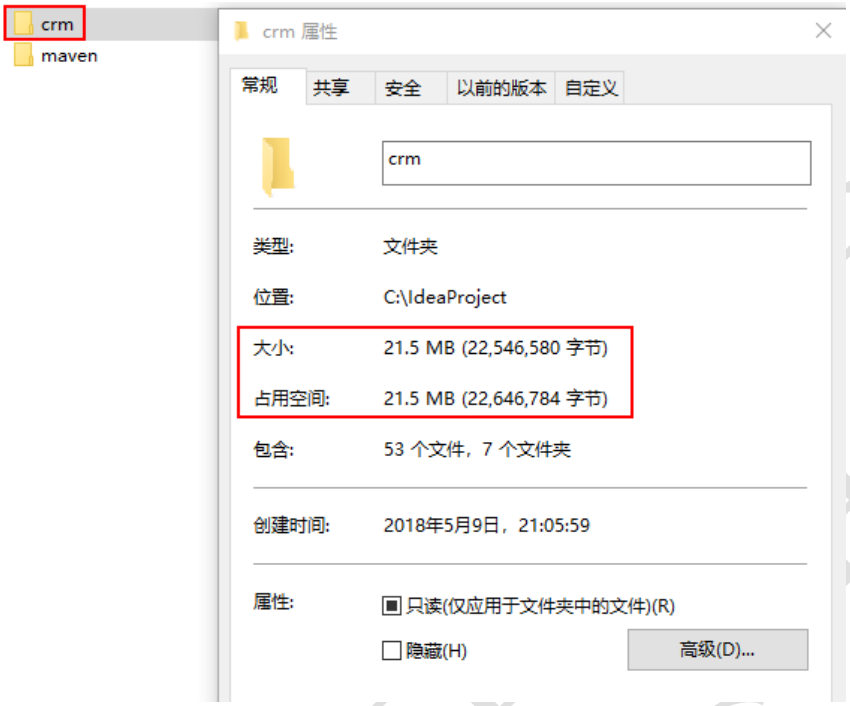
4、再优雅的代码也是要出来卖的。我们后面还需要把代码与各种配置文件、资源整合到一起，定型打包，如果是 web 项目，还需要将之发布到服务器，供人蹂躏。

试想，如果现在有一种工具，可以把你从上面的繁琐工作中解放出来，能帮你构建工程，管理 jar 包，编译代码，还能帮你自动运行单元测试，打包，生成报表，甚至能帮你部署项目，生成 Web 站点，你会心动吗？Maven 就可以解决上面所提到的这些问题。

1.1.3 Maven 的优势举例

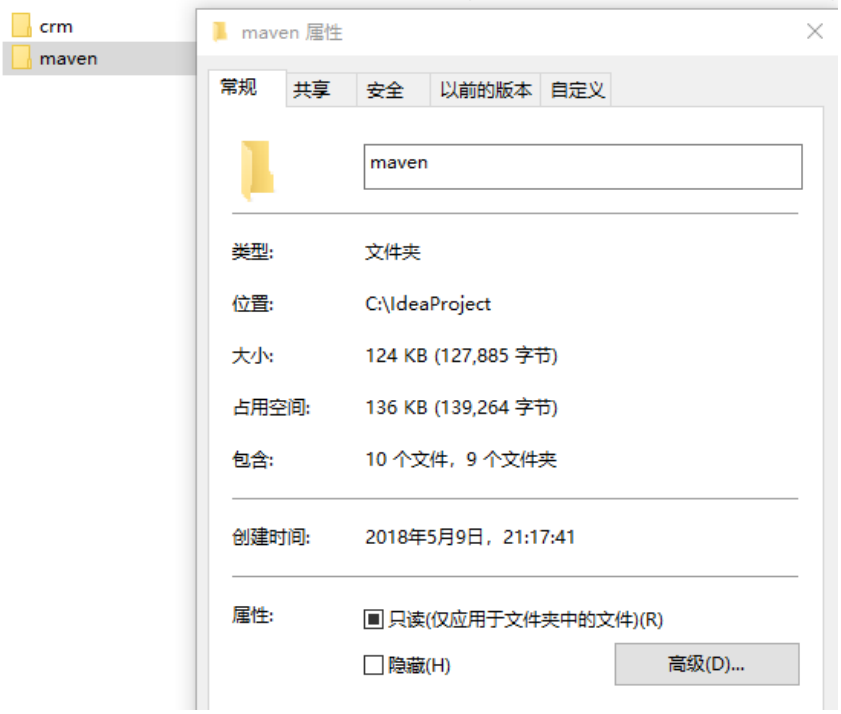
前面我们通过 Web 阶段项目，要能够将项目运行起来，就必须将该项目所依赖的一些 jar 包添加到工程中，否则项目就不能运行。试想如果具有相同架构的项目有十个，那么我们就需要将这一份 jar 包复制到十个不同的工程中。我们一起来看一个 CRM 项目的工程大小。

使用传统 Web 项目构建的 CRM 项目如下：



原因主要是因为上面的 WEB 程序要运行，我们必须将项目运行所需的 Jar 包复制到工程目录中，从而导致了工程很大。

同样的项目，如果我们使用 Maven 工程来构建，会发现总体上工程的大小会少很多。如下图:



小结：可以初步推断它里面一定没有 jar 包，继续思考，没有 jar 包的项目怎么可能运行呢？

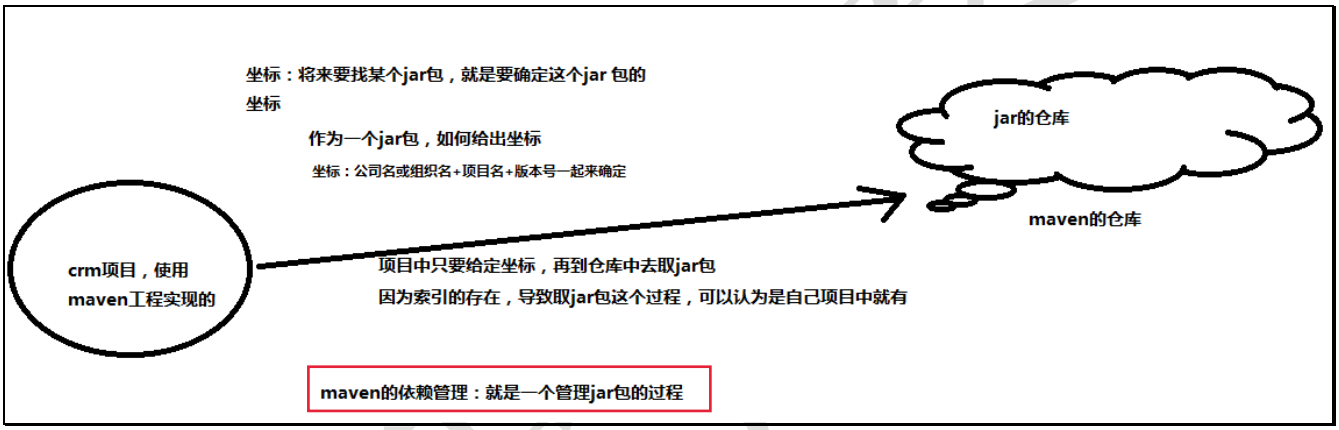
1.2 Maven 的两个精典作用

1.2.1 Maven 的依赖管理

Maven 的一个核心特性就是依赖管理。当我们涉及到多模块的项目（包含成百个模块或者子项目），管理依赖就变成一项困难的任务。Maven 展示出了它对处理这种情形的高度控制。

传统的 WEB 项目中，我们必须将工程所依赖的 jar 包复制到工程中，导致了工程的变得很大。那么 maven 工程是如何使得工程变得很少呢？

分析如下：



通过分析发现：maven 工程中不直接将 jar 包导入到工程中，而是通过在 pom.xml 文件中添加所需 jar 包的坐标，这样就很好的避免了 jar 直接引入进来，在需要用到 jar 包的时候，只要查找 pom.xml 文件，再通过 pom.xml 文件中的坐标，到一个专门用于”存放 jar 包的仓库”(maven 仓库)中根据坐标从而找到这些 jar 包，再把这些 jar 包拿去运行。

那么问题来了

第一：”存放 jar 包的仓库”长什么样？

第二：通过读取 pom.xml 文件中的坐标，再到仓库中找到 jar 包，会不会很慢？从而导致这种方式不可行！

第一个问题：存放 jar 包的仓库长什么样，这一点我们后期会分析仓库的分类，也会带大家去看我们的本地的仓库长什么样。

第二个问题：通过 pom.xml 文件配置要引入的 jar 包的坐标，再读取坐标并到仓库中加载 jar 包，这样我们就可以直接使用 jar 包了，为了解决这个过程中速度慢的问题，maven 中也有索引的概念，通过建立索引，可以大大提高加载 jar 包的速度，使得我们认为 jar 包基本跟放在本地的工程文件中再读取出来的速度是一样的。这个过程就好比我们查阅字典时，为了能够加快查找到内容，书前面的目录就好比是索引，有了这个目录我们就可以方便找到内容了，一样的在 maven 仓库中有了索引我们就可以认为可以快速找到 jar 包。

1.2.2 项目的一键构建

我们的项目，往往都要经历编译、测试、运行、打包、安装 ，部署等一系列过程。

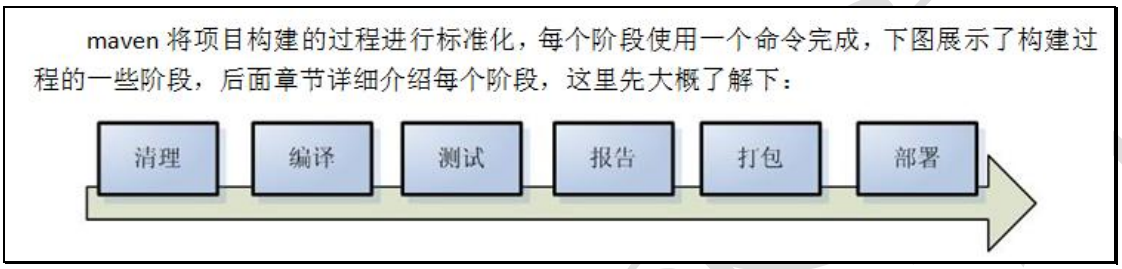
什么是构建？

指的是项目从编译、测试、运行、打包、安装 ，部署整个过程都交给 maven 进行管理，这个过程称为构建。

一键构建

指的是整个构建过程，使用 maven 一个命令可以轻松完成整个工作。

Maven 规范化构建流程如下：



我们一起来看 Hello-Maven 工程的一键运行的过程。通过 tomcat:run 的这个命令，我们发现现在的工程编译，测试，运行都变得非常简单。

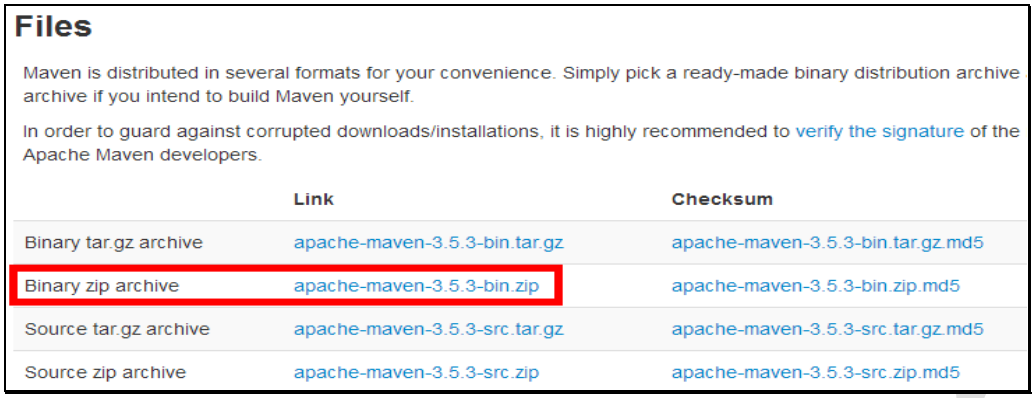
第2章 Maven 的使用

2.1 Maven 的安装

2.1.1 Maven 软件的下载

为了使用 Maven 管理工具，我们首先要到官网去下载它的安装软件。通过百度搜索“Maven“如下：

点击 Download 链接，就可以直接进入到 Maven 软件的下载页面：



Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the Apache Maven developers.

	Link	Checksum
Binary tar.gz archive	apache-maven-3.5.3-bin.tar.gz	apache-maven-3.5.3-bin.tar.gz.md5
Binary zip archive	apache-maven-3.5.3-bin.zip	apache-maven-3.5.3-bin.zip.md5
Source tar.gz archive	apache-maven-3.5.3-src.tar.gz	apache-maven-3.5.3-src.tar.gz.md5
Source zip archive	apache-maven-3.5.3-src.zip	apache-maven-3.5.3-src.zip.md5

目前最新版是 apache-maven-3.5.3 版本，我们当时使用的是 apache-maven-3.5.2 版本，大家也可以下载最新版本。

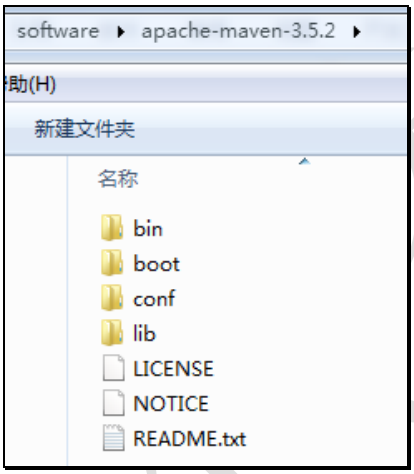
Apache-maven-3.5.2 下载地址：http://archive.apache.org/dist/maven/maven-3/

下载后的版本如下:

2.1.2 Maven 软件的安装

Maven 下载后，将 Maven 解压到一个没有中文没有空格的路径下，比如 D:\software\maven 下面。

解压后目录结构如下：



bin:存放了 maven 的命令，比如我们前面用到的 mvn tomcat:run

boot:存放了一些 maven 本身的引导程序，如类加载器等

conf:存放了 maven 的一些配置文件，如 setting.xml 文件

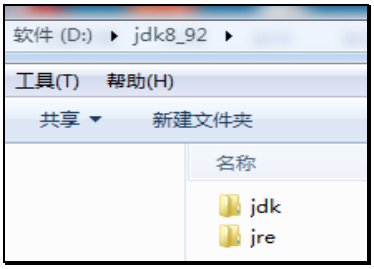
lib:存放了 maven 本身运行所需的一些 jar 包

至此我们的 maven 软件就可以使用了，前提是你的电脑上之前已经安装并配置好了 JDK。

2.1.3 JDK 的准备及统一

本次课程我们所使用工具软件的统一，JDK 使用 JDK8 版本

1. JDK 环境:

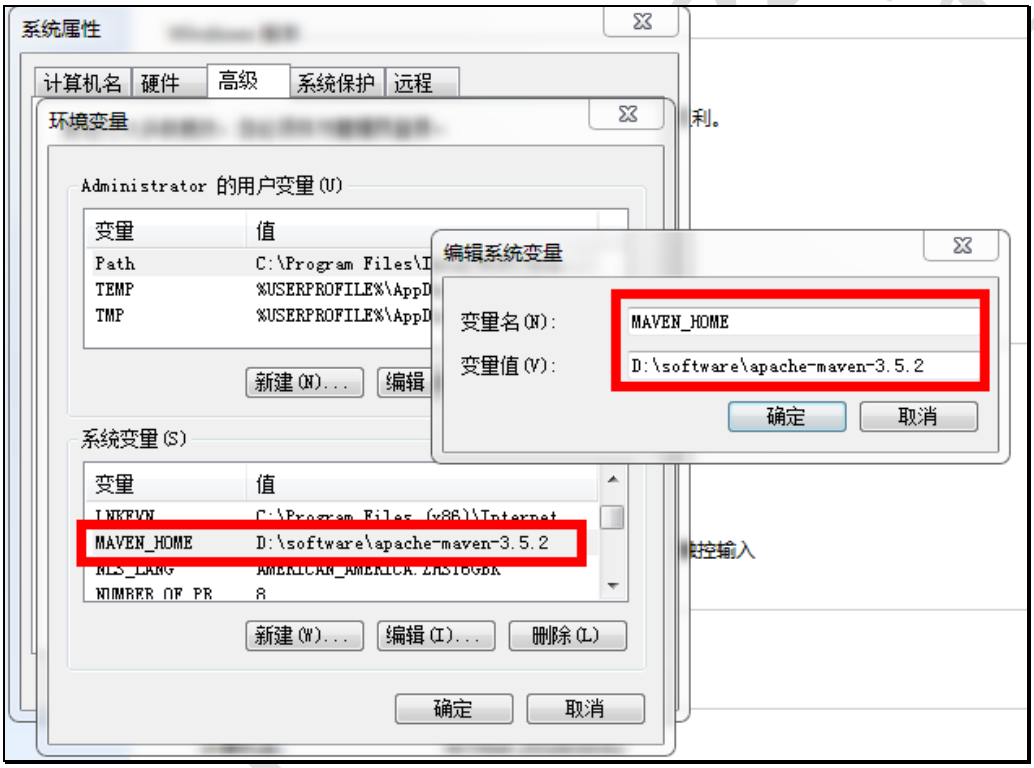


2.1.4 Maven 及 JDK 配置

电脑上需安装 java 环境，安装 JDK1.7 + 版本 （将 JAVA_HOME/bin 配置环境变量 path ），我们使用的是 JDK8 相关版本

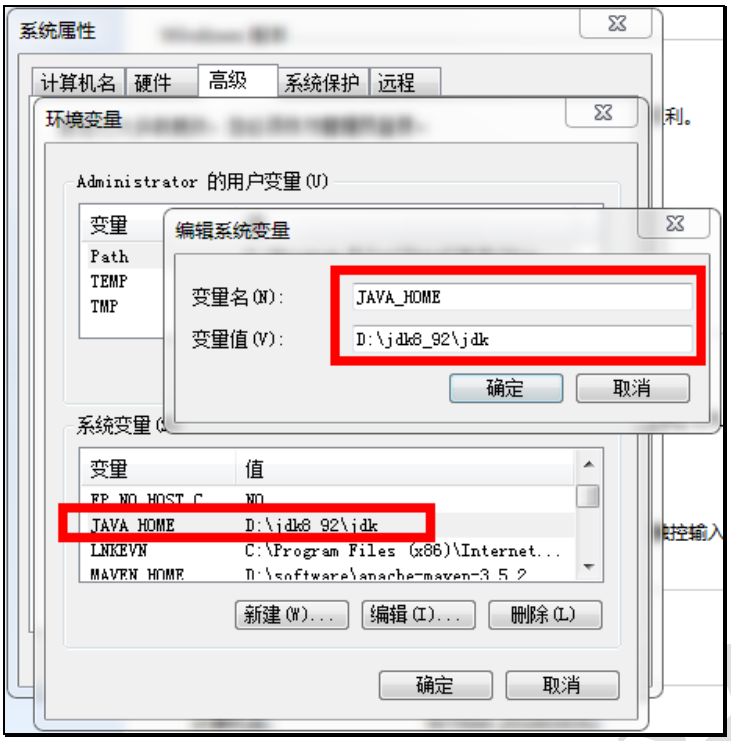
还要设置path变量bin目录

配置 MAVEN_HOME ，变量值就是你的 maven 安装 的路径（bin 目录之前一级目录）



上面配置了我们的 Maven 软件，注意这个目录就是之前你解压 maven 的压缩文件包在的的目录，最好不要有中文和空格。

再次检查 JDK 的安装目录，如下图：



2.1.5 Maven 软件版本测试

通过 mvn -v 命令检查 maven 是否安装成功，看到 maven 的版本为 3.5.2 及 java 版本为 1.8 即为安装成功。

找开 cmd 命令，输入 mvn –v 命令，如下图：

```
C:\Users\Administrator>mvn -v
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d; 2017-10-18T15:58:13+08:00)
Maven home: D:\software\apache-maven-3.5.2\bin\..
Java version: 1.8.0_92, vendor: Oracle Corporation
Java home: D:\jdk8_92\jdk\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

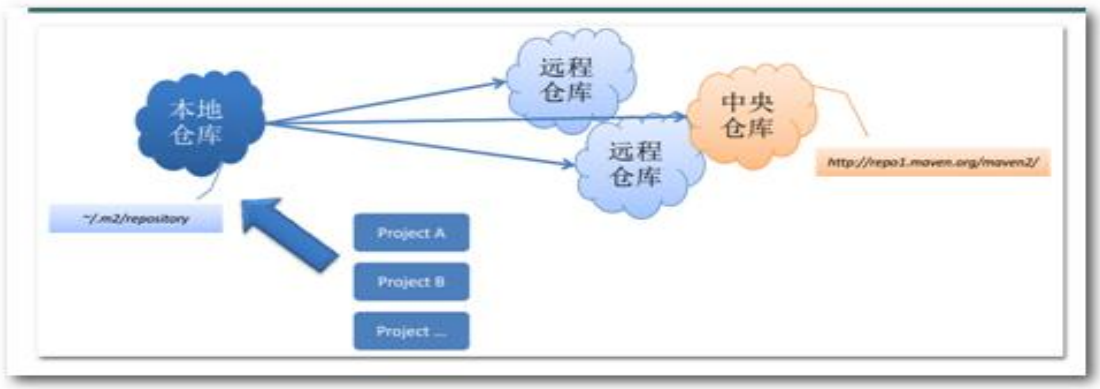
我们发现 maven 的版本，及 jdk 的版本符合要求，这样我们的 maven 软件安装就成功了。

2.2 Maven 仓库

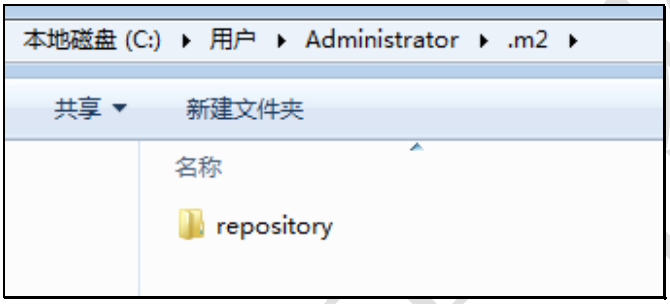
2.2.1 Maven 仓库的分类

maven 的工作需要从仓库下载一些 jar 包，如下图所示，本地的项目 A、项目 B 等都会通过 maven 软件从远程仓库（可以理解为互联网上的仓库）下载 jar 包并存在本地仓库，本地仓库 就是本地文件夹，当第二次需要此 jar 包时则不再从远程仓库下载，因为本地仓库已经存在了，可以将本地仓库理解为缓存，有了本地仓库就不用每次从远程仓库下载了。

下图描述了 maven 中仓库的类型：



- 本地仓库 ：用来存储从远程仓库或中央仓库下载的插件和 jar 包，项目使用一些插件或 jar 包，优先从本地仓库查找
  默认本地仓库位置在 ${user.dir}/.m2/repository，${user.dir}表示 windows 用户目录。

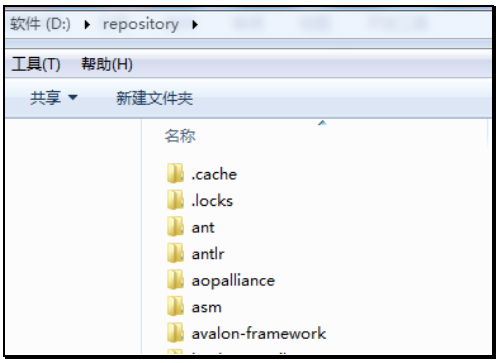


- 远程仓库：如果本地需要插件或者 jar 包，本地仓库没有，默认去远程仓库下载。
  远程仓库可以在互联网内也可以在局域网内。

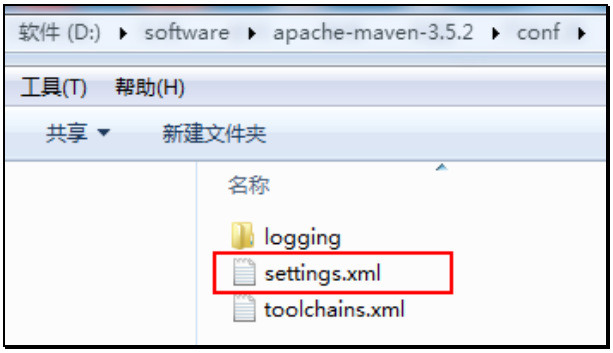
- 中央仓库 ：在 maven 软件中内置一个远程仓库地址 http://repo1.maven.org/maven2 ，它是中央仓库，服务于整个互联网，它是由 Maven 团队自己维护，里面存储了非常全的 jar 包，它包含了世界上大部分流行的开源项目构件。

2.2.2 Maven 本地仓库的配置

本课程是在无网的状态下学习，需要配置老师提供的本地仓库，将 “repository.rar”解压至自己的电脑上，我们解压在 D:\repository 目录下（可以放在没有中文及空格的目录下）。



在 MAVE_HOME/conf/settings.xml 文件中配置本地仓库位置（maven 的安装目录下）：



打开 settings.xml 文件，配置如下：

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- localRepository
   | The path to the local repository maven will use to store artifacts.
   |
   | Default: ${user.home}/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  -->
  <localRepository>D:/repository</localRepository>
```

D:\repository目录是maven本地仓库所在的目录

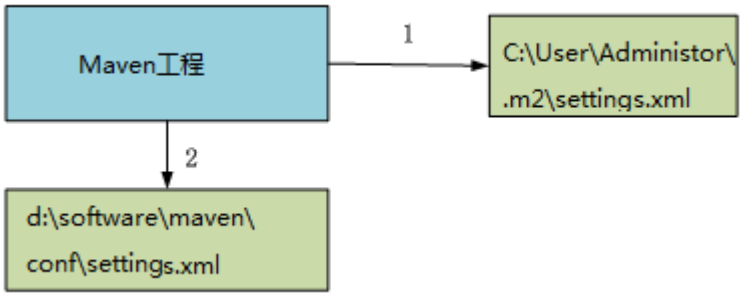
2.2.3 全局 setting 与用户 setting

maven 仓库地址、私服等配置信息需要在 setting.xml 文件中配置，分为全局配置和用户配置。

在 maven 安装目录下的有 conf/setting.xml 文件，此 setting.xml 文件用于 maven 的所有 project 项目，它作为 maven 的全局配置。

如需要个性配置则需要在用户配置中设置，用户配置的 setting.xml 文件默认的位置在：${user.dir}/.m2/settings.xml 目录中，${user.dir} 指 windows 中的用户目录。

maven 会先找用户配置，如果找到则以用户配置文件为准，否则使用全局配置文件。



2.3 Maven 工程的认识

2.3.1 Maven 工程的目录结构

作为一个 maven 工程，它的 src 目录和 pom.xml 是必备的。
进入 src 目录后，我们发现它里面的目录结构如下：

src/main/java —— 存放项目的.java 文件
src/main/resources —— 存放项目资源文件，如 spring, hibernate 配置文件
src/test/java —— 存放所有单元测试.java 文件，如 JUnit 测试类
src/test/resources —— 测试资源文件
target —— 项目输出位置，编译后的 class 文件会输出到此目录
pom.xml——maven 项目核心配置文件

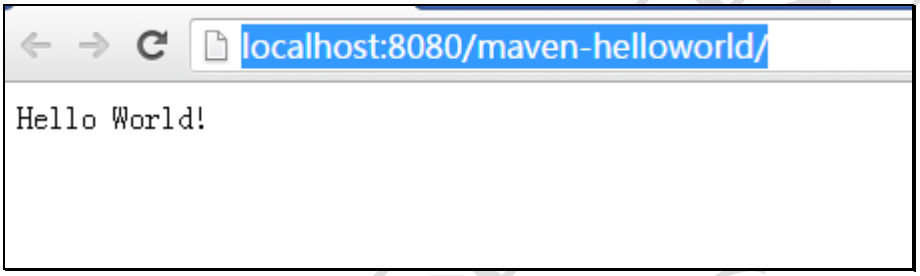
注意：如果是普通的 java 项目，那么就没有 webapp 目录。

2.3.2 Maven 工程的运行

进入 maven 工程目录（当前目录有 pom.xml 文件），运行 tomcat:run 命令。

```
F:\develop\maven\project\maven-helloworld>mvn tomcat:run
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for cn.itcast.maven:maven-helloworld:war:0.0.1-SNAPSHOT
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 28, column 12
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building 第一个maven工程 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> tomcat-maven-plugin:1.1:run (default-cli) > compile @ maven-helloworld >>>
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ maven-helloworld ---
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ maven-helloworld ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] <<< tomcat-maven-plugin:1.1:run (default-cli) < compile @ maven-helloworld <<<
[INFO]
[INFO] --- tomcat-maven-plugin:1.1:run (default-cli) @ maven-helloworld ---
[INFO] Running war on http://localhost:8080/maven-helloworld
[INFO] Using existing Tomcat server configuration at F:\develop\maven\project\maven-helloworld\target\tomcat
七月 23, 2016 4:27:43 下午 org.apache.catalina.startup.Embedded start
信息: Starting tomcat server
七月 23, 2016 4:27:43 下午 org.apache.catalina.core.StandardEngine start
信息: Starting Servlet Engine: Apache Tomcat/6.0.29
七月 23, 2016 4:27:43 下午 org.apache.coyote.http11.Http11Protocol init
信息: Initializing Coyote HTTP/1.1 on http-8080
七月 23, 2016 4:27:43 下午 org.apache.coyote.http11.Http11Protocol start
信息: Starting Coyote HTTP/1.1 on http-8080
```

根据上边的提示信息，通过浏览器访问：http://localhost:8080/maven-helloworld/



2.3.3 问题处理

如果本地仓库配置错误会报下边的错误

```
F:\develop\maven\project\maven-helloworld>mvn tomcat:run
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for cn.itcast.maven:maven-helloworld:war:0.0.1-SNAPSHOT
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 28, column 12
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/3.1/maven-compiler-plugin-3.1.pom
```

分析：

maven 工程运行先从本地仓库找 jar 包，本地仓库没有再从中央仓库找，上边提示 downloading...表示 从中央仓库下载 jar，由于本地没有联网，报错。

解决：

在 maven 安装目录的 conf/setting.xml 文件中配置本地仓库，参考：“maven 仓库/配置本地仓库章节”。

第3章 Maven 常用命令

我们可以在 cmd 中通过一系列的 maven 命令来对我们的 maven-helloworld 工程进行编译、测试、运行、打包、安装、部署。

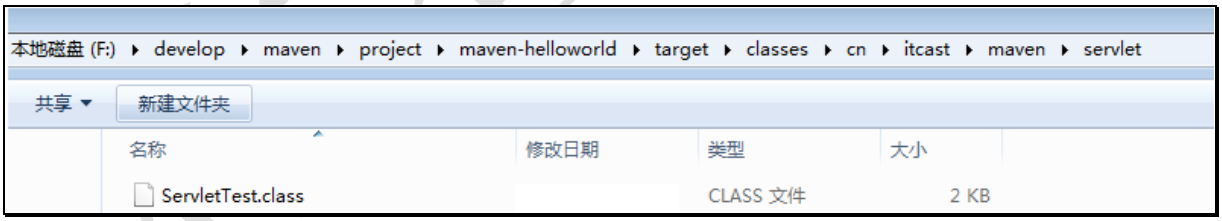
3.1.1 compile

compile 是 maven 工程的编译命令，作用是将 src/main/java 下的文件编译为 class 文件输出到 target 目录下。

cmd 进入命令状态，执行 mvn compile，如下图提示成功：

```
F:\develop\maven\project\maven-helloworld>mvn compile
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for cn.itcast.maven:maven-helloworld:war:0.0.1-SNAPSHOT
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 28, column 12
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building 第一个maven工程 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ maven-helloworld ---
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ maven-helloworld ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to F:\develop\maven\project\maven-helloworld\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.985 s
[INFO] Finished at: 2016-07-23T16:57:49+08:00
[INFO] Final Memory: 9M/22M
[INFO] ------------------------------------------------------------------------
```

查看 target 目录，class 文件已生成，编译完成。



3.1.2 test

test 是 maven 工程的测试命令　mvn test，会执行 src/test/java 下的单元测试类。

什么是单元测试类？

cmd 执行 mvn test 执行 src/test/java 下单元测试类，下图为测试结果，运行 1 个测试用例，全部成功。

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running cn.itcast.maven.test.HelloTest
hello test....
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.042 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.992 s
[INFO] Finished at: 2016-07-23T17:14:07+08:00
[INFO] Final Memory: 11M/27M
[INFO] ------------------------------------------------------------------------
```

3.1.3 clean

clean 是 maven 工程的清理命令，执行 clean 会删除 target 目录及内容。

3.1.4 package

package 是 maven 工程的打包命令，对于 java 工程执行 package 打成 jar 包，对于 web 工程打成 war 包。

3.1.5 install

install 是 maven 工程的安装命令，执行 install 将 maven 打成 jar 包或 war 包发布到本地仓库。
从运行结果中，可以看出：
当后面的命令执行时，前面的操作过程也都会自动执行，

3.1.6 Maven 指令的生命周期

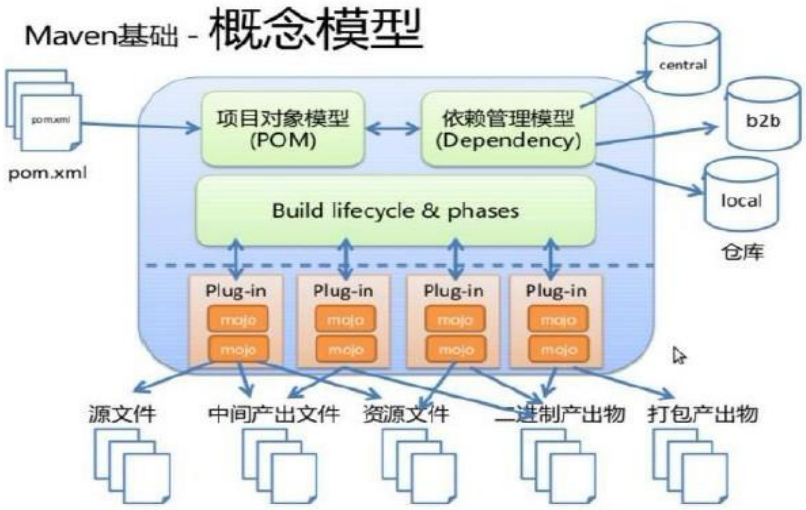
Maven 构建生命周期就是 Maven 将一个整体任务划分为一个个的阶段，类似于流程图，按顺序依次执行。也可以指定该任务执行到中间的某个阶段结束。

maven 对项目构建过程分为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。

3.1.7 maven 的概念模型

Maven 包含了一个项目对象模型 (Project Object Model)，一组标准集合，一个项目生命周期(Project Lifecycle)，一个依赖管理系统(Dependency Management System)，和用来运行定义在生命周期阶段(phase)中插件(plugin)目标(goal)的逻辑。



- 项目对象模型 (Project Object Model) 项目本身配置

一个 maven 工程都有一个 pom.xml 文件，通过 pom.xml 文件定义项目的坐标、项目依赖、项目信息、插件目标等。

- 依赖管理系统(Dependency Management System)

通过 maven 的依赖管理对项目所依赖的 jar 包进行统一管理。

比如：项目依赖 junit4.9，通过在 pom.xml 中定义 junit4.9 的依赖即使用 junit4.9，如下所示是 junit4.9 的依赖定义：

```
<!-- 依赖关系 -->
    <dependencies>
        <!-- 此项目运行使用 junit，所以此项目依赖 junit -->
        <dependency>
            <!-- junit 的项目名称 -->
            <groupId>junit</groupId>
            <!-- junit 的模块名称 -->
            <artifactId>junit</artifactId>
            <!-- junit 版本 -->
            <version>4.9</version>
            <!-- 依赖范围：单元测试时使用 junit -->
            <scope>test</scope>
        </dependency>
```

- 一个项目生命周期(Project Lifecycle)

使用 maven 完成项目的构建，项目构建包括：清理、编译、测试、部署等过程，maven 将这些过程规范为一个生命周期，如下所示是生命周期的各各阶段：

maven 通过执行一些简单命令即可实现上边生命周期的各各过程，比如执行 mvn compile 执行编译、执行 mvn clean 执行清理。

- 一组标准集合

  maven 将整个项目管理过程定义一组标准，比如：通过 maven 构建工程有标准的目录结构，有标准的生命周期阶段、依赖管理有标准的坐标定义等。

- 插件(plugin)目标(goal)

maven 管理项目生命周期过程都是基于插件完成的。

3.2 idea 开发 maven 项目

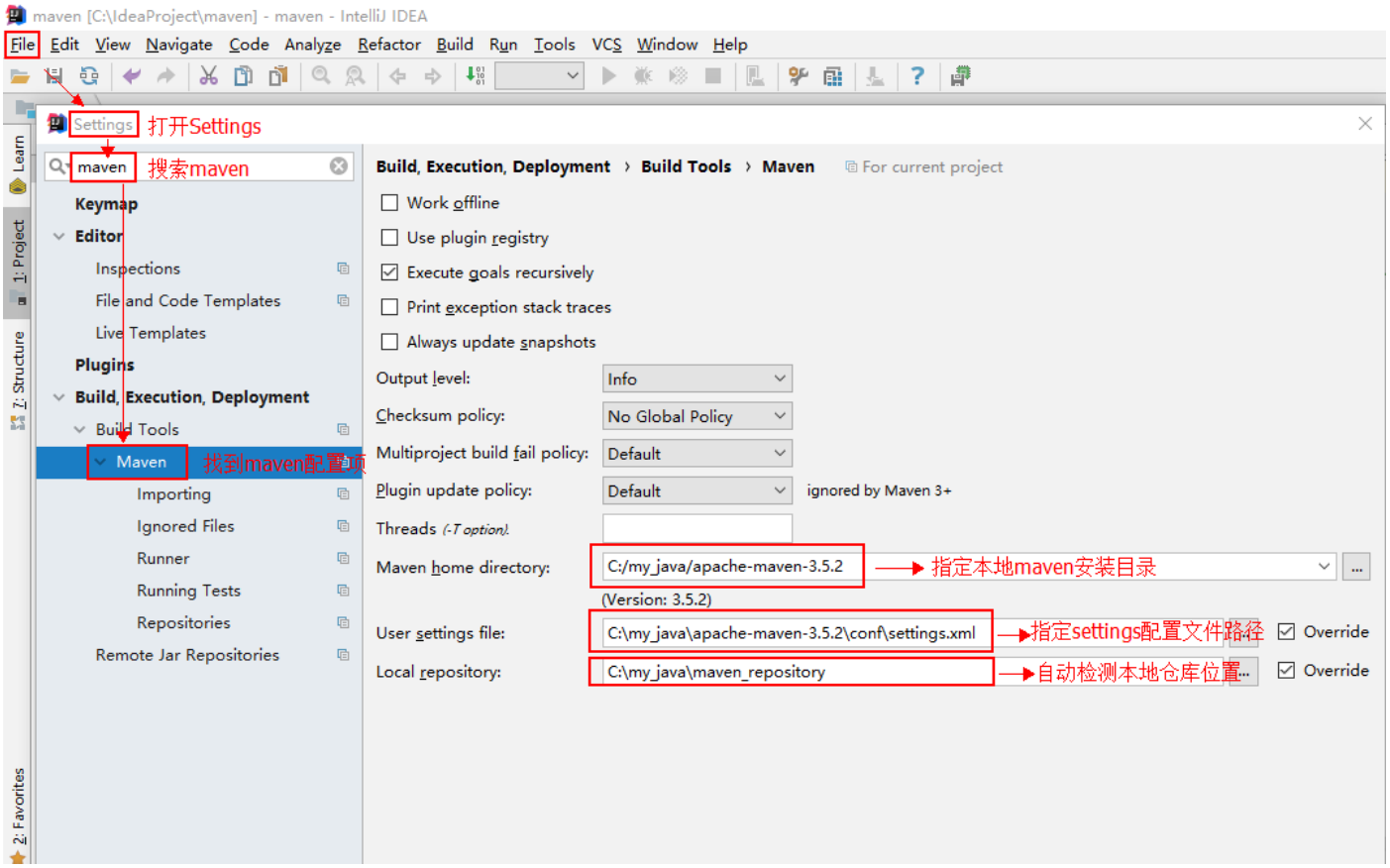
在实战的环境中，我们都会使用流行的工具来开发项目。

3.2.1 idea 的 maven 配置

3.2.1.1 打开→File→Settings 配置 maven

vm option填写-DarchetypeCatalog=internal

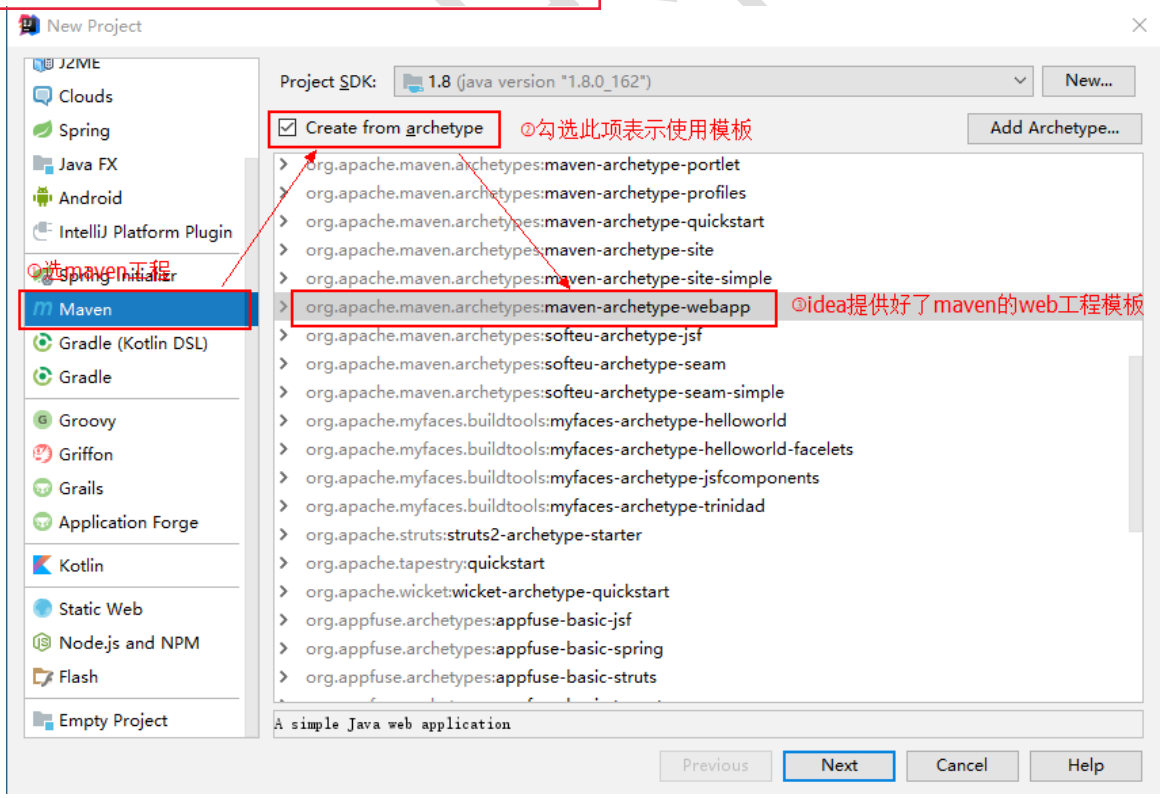
依据图片指示，选择本地 maven 安装目录，指定 maven 安装目录下 conf 文件夹中 settings 配置文件。



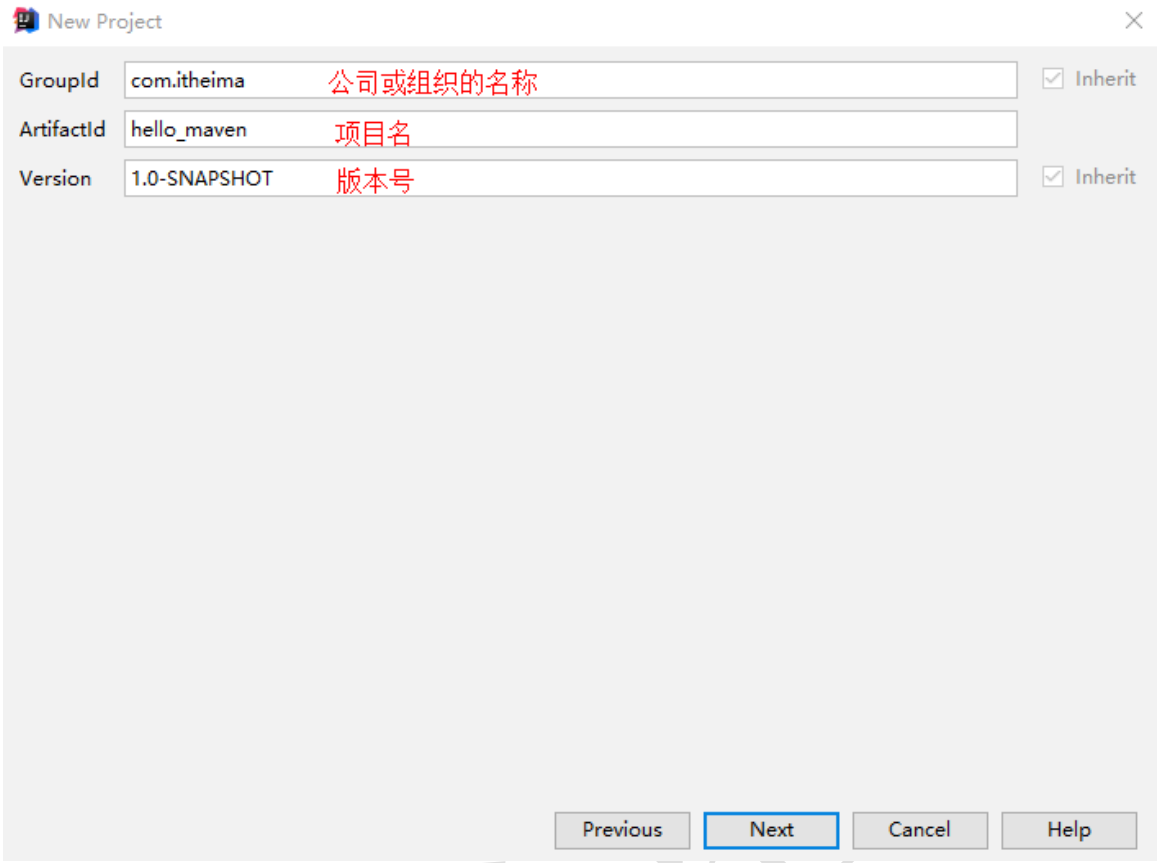
3.2.2 idea 中创建一个 maven 的 web 工程

打开 idea，选择创建一个新工程

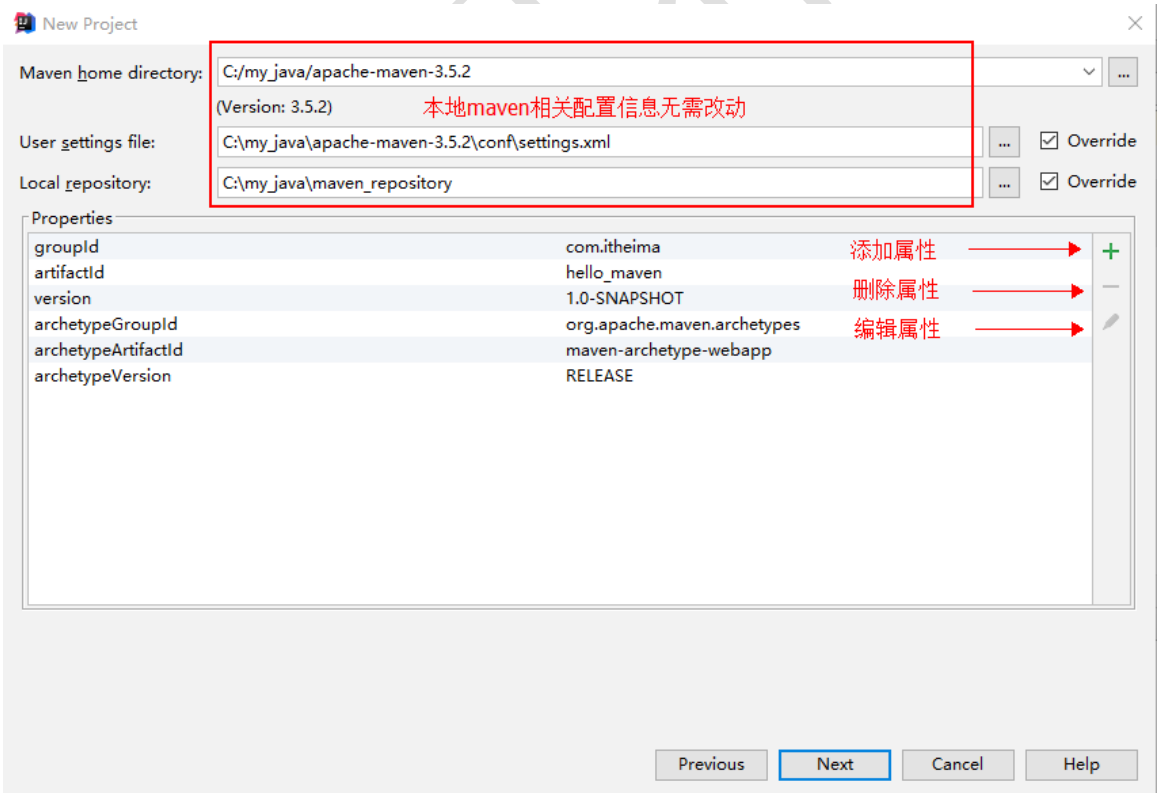
选择 idea 提供好的 maven 的 web 工程模板



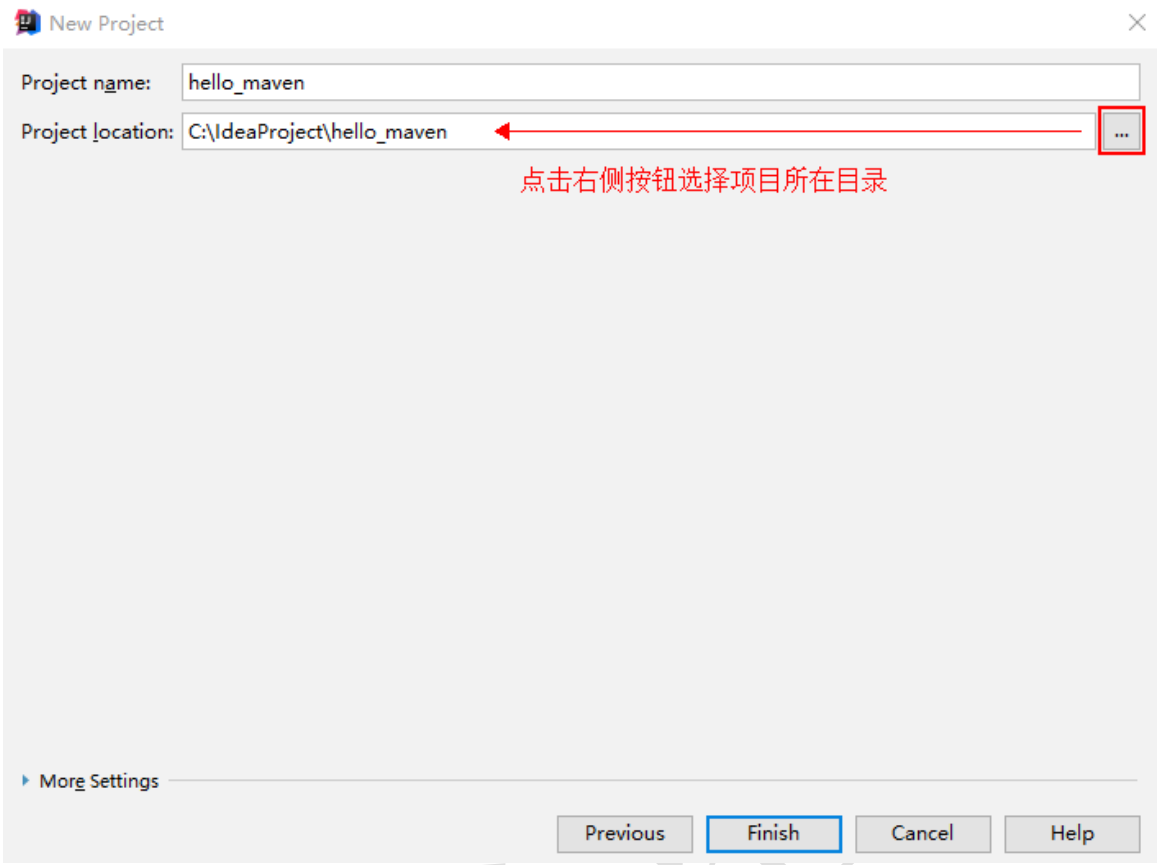
点击 Next 填写项目信息



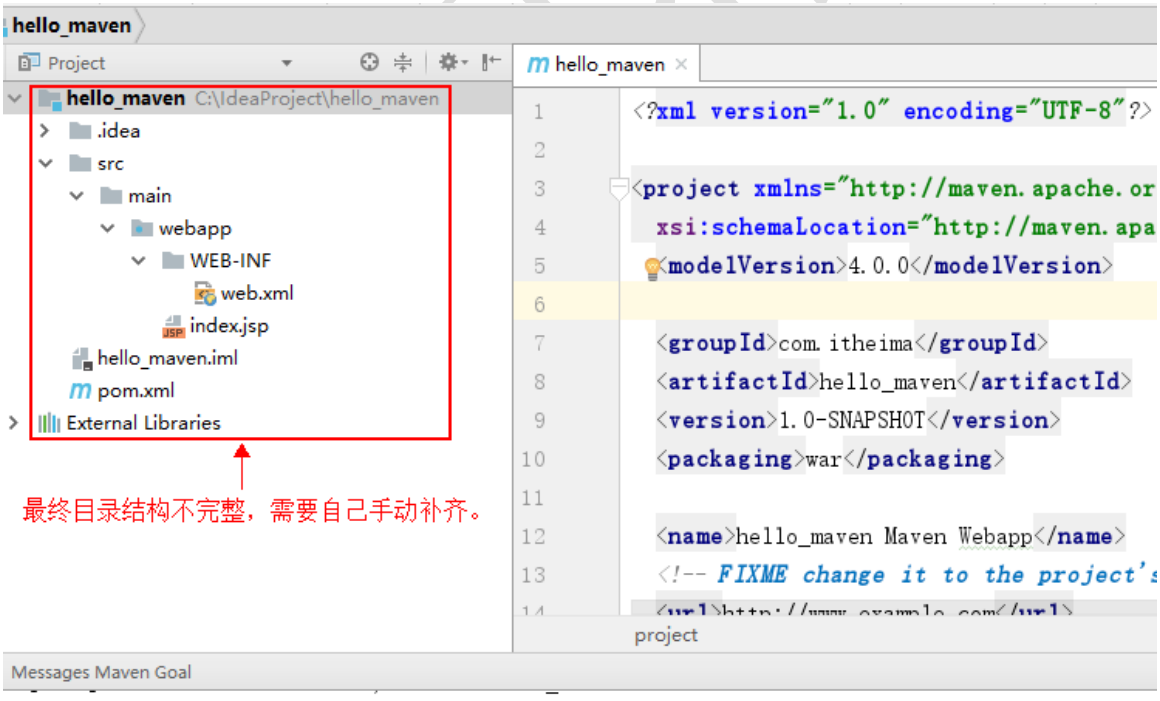
点击 Next，此处不做改动。

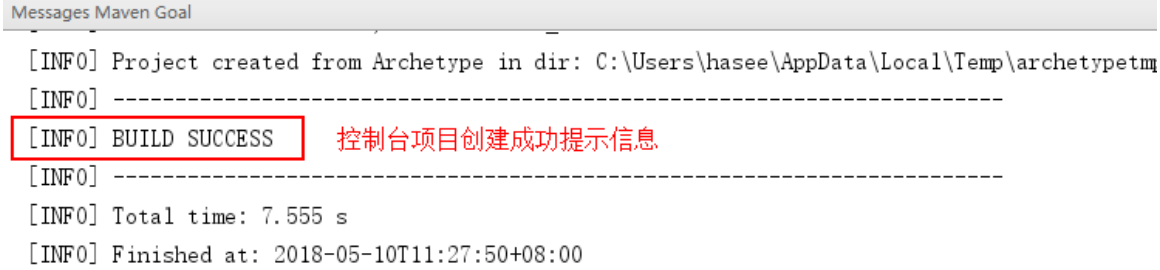


点击 Next 选择项目所在目录

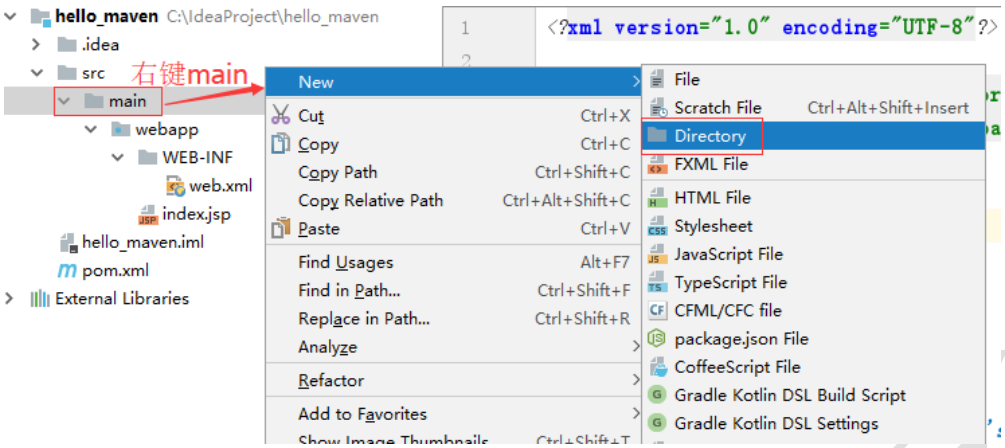


点击 Finish 后开始创建工程，耐心等待，直到出现如下界面。

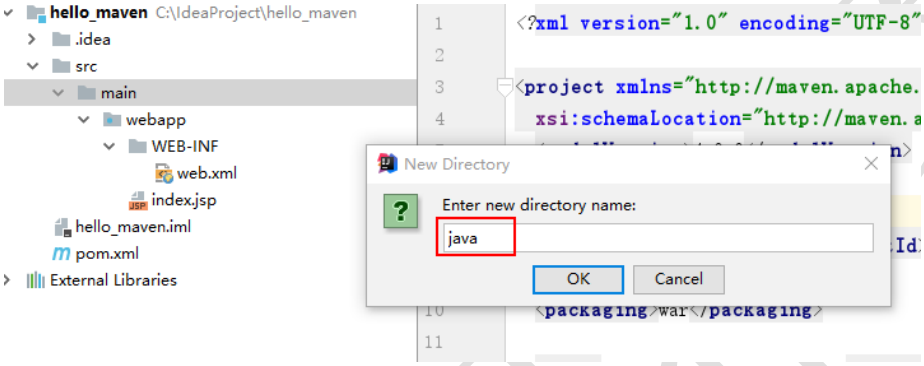




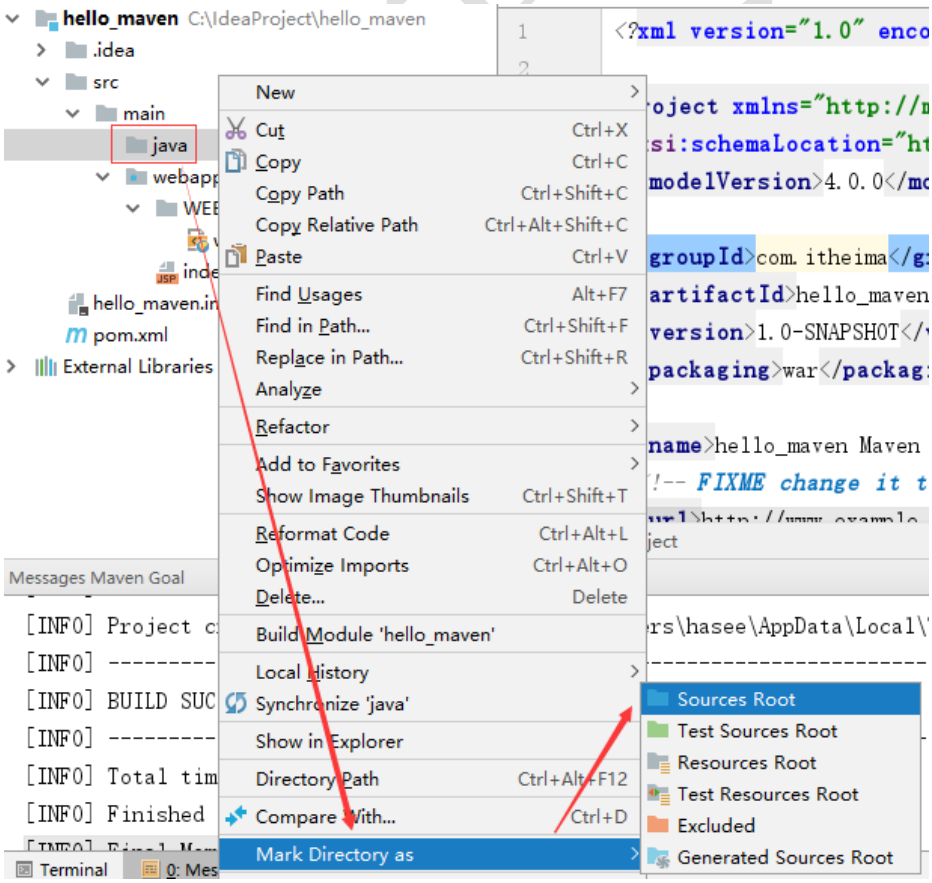
手动添加 src/main/java 目录，如下图右键 main 文件夹→New→Directory



创建一个新的文件夹命名为 java



点击 OK 后，在新的文件夹 java 上右键→Make Directory as→Sources Root



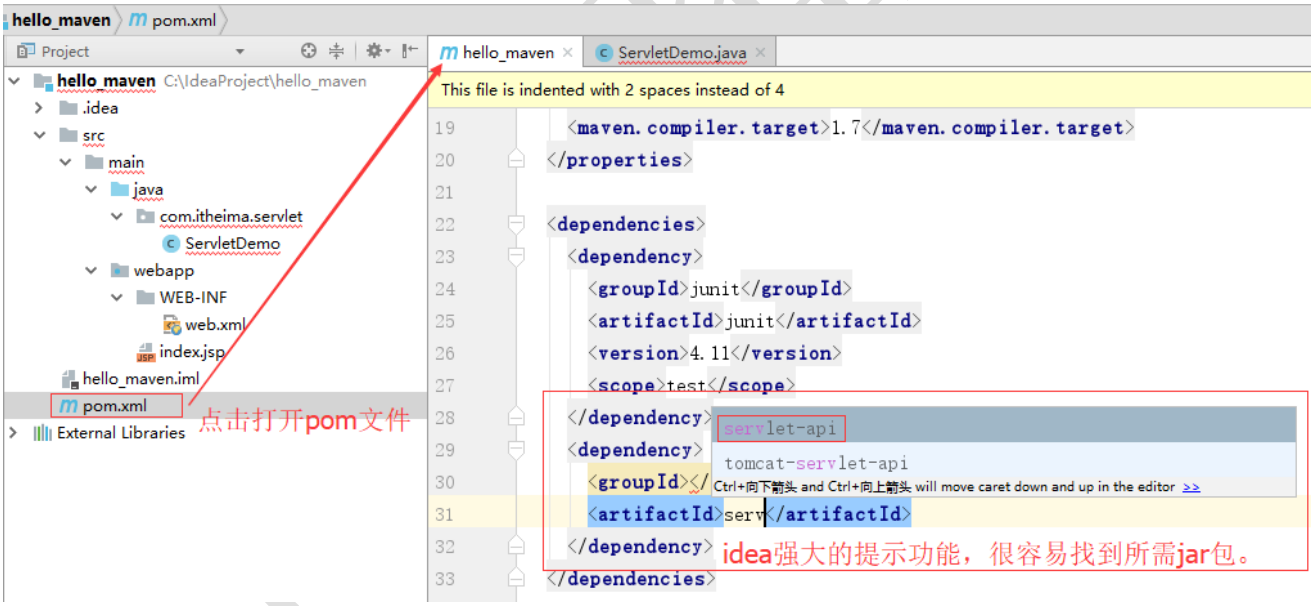
3.2.2.1 创建一个 Servlet

src/java/main 创建了一个 Servlet，但报错

要解决问题，就是要将 servlet-api-xxx.jar 包放进来，作为 maven 工程应当添加 servlet 的坐标，从而导入它的 jar

3.2.2.2 在 pom.xml 文件添加坐标

直接打开 hello_maven 工程的 pom.xml 文件，再添加坐标



添加 jar 包的坐标时，还可以指定这个 jar 包将来的作用范围。

每个 maven 工程都需要定义本工程的坐标，坐标是 maven 对 jar 包的身份定义，比如：入门程序的坐标定义如下：

```
<!--项目名称，定义为组织名+项目名，类似包名-->
<groupId>com.itheima</groupId>
<!-- 模块名称 -->
<artifactId>hello_maven</artifactId>
<!-- 当前项目版本号，snapshot 为快照版本即非正式版本，release 为正式发布版本 -->
```

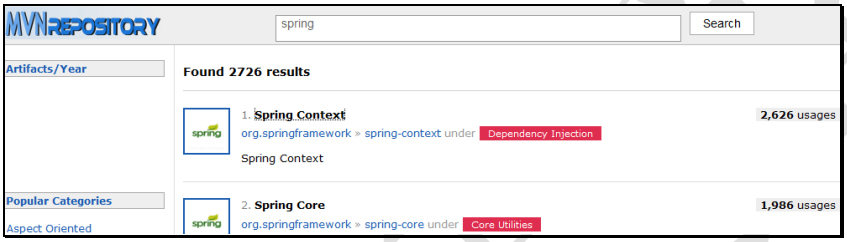
<version>0.0.1-SNAPSHOT</version>
<packaging> ：打包类型
jar：执行 package 会打成 jar 包
war：执行 package 会打成 war 包
pom ：用于 maven 工程的继承，通常父工程设置为 pom

3.2.2.3 坐标的来源方式

添加依赖需要指定依赖 jar 包的坐标，但是很多情况我们是不知道 jar 包的的坐标，可以通过如下方式查询：

3.2.2.3.1 从互联网搜索

http://search.maven.org/
http://mvnrepository.com/
网站搜索示例：



3.2.3 依赖范围 6种

A 依赖 B，需要在 A 的 pom.xml 文件中添加 B 的坐标，添加坐标时需要指定依赖范围，依赖范围包括：

- ✓ compile：编译范围，指 A 在编译时依赖 B，此范围为默认依赖范围。编译范围的依赖会用在编译、测试、运行，由于运行时需要所以编译范围的依赖会被打包。 最大

- ✓ provided：provided 依赖只有在当 JDK 或者一个容器已提供该依赖之后才使用， provided 依赖在编译和测试时需要，在运行时不需要，比如：servlet api 被 tomcat 容器提供。

- ✓ runtime：runtime 依赖在运行和测试系统的时候需要，但在编译的时候不需要。比如：jdbc 的驱动包。由于运行时需要所以 runtime 范围的依赖会被打包。

- ✓ test：test 范围依赖 在编译和运行时都不需要，它们只有在测试编译和测试运行阶段可用，比如：junit。由于运行时不需要所以 test 范围依赖不会被打包。 在main包中的代码是引用不到的，因此会报错

- ✓ system：system 范围依赖与 provided 类似，但是你必须显式的提供一个对于本地系统中 JAR 文件的路径，需要指定 systemPath 磁盘路径，system 依赖不推荐使用。

- import

This scope is only supported on a dependency of type `pom` in the `<dependencyManagement>` section. It indicates the dependency to be replaced with the effective list of dependencies in the specified POM's `<dependencyManagement>` section. Since they are replaced, dependencies with a scope of `import` do not actually participate in limiting the transitivity of a dependency.

依赖范围	对于编译classpath有效	对于测试classpath有效	对于运行时classpath有效	例子
compile	Y	Y	Y	spring-core
test	-	Y	-	Junit
provided	Y	Y	-	servlet-api
runtime	-	Y	Y	JDBC驱动
system	Y	Y	-	本地的，Maven仓库之外的类库

Runtime
只是反射，编译时不需要import类．
由于还在测试别的功能的阶段，还没有编写mysql，所以编译的时候不会用到这个包。但是生成Context上下文的时候会用到。 也就是说这个包mysql包编译的时候是用不到的，但是在运行的时候会用到。所以如果我们在package的时候没有将其打包到target中，运行时就会出错。

在 maven-web 工程中测试各各 scop。
测试总结：
- ✓ 默认引入 的 jar 包 ------- compile 【默认范围 可以不写】（编译、测试、运行 都有效 ）
- ✓ servlet-api 、jsp-api ------- provided （编译、测试 有效， 运行时无效 防止和 tomcat 下 jar 冲突）
- ✓ jdbc 驱动 jar 包 ---- runtime （测试、运行 有效 ）
- ✓ junit ----- test （测试有效）

依赖范围由强到弱的顺序是：compile>provided>runtime>test

3.2.4 项目中添加的坐标

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

3.2.5 设置 jdk 编译版本

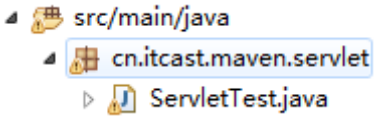
本教程使用 jdk1.8，需要设置编译版本为 1.8，这里需要使用 maven 的插件来设置：
在 pom.xml 中加入：

```
<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>
```

3.2.6 编写 servlet

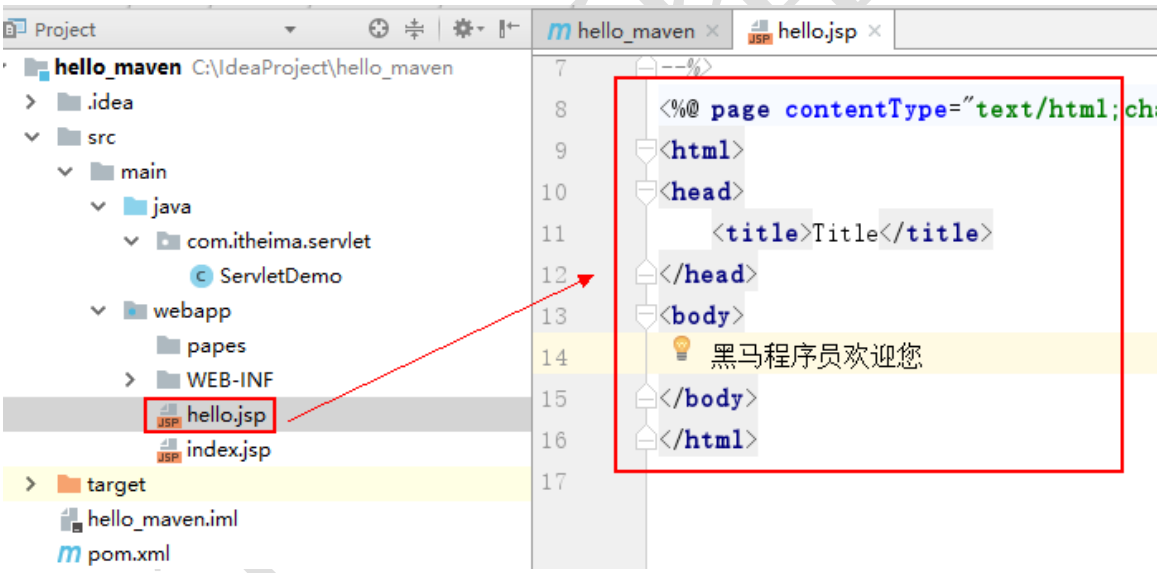
在 src/main/java 中创建 ServletTest



内容如下

```
public class ServletDemo extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
        req.getRequestDispatcher( s: "/hello.jsp").forward(req, resp);
    }
}
```

3.2.7 编写 jsp



3.2.8 在 web.xml 中配置 servlet 访问路径

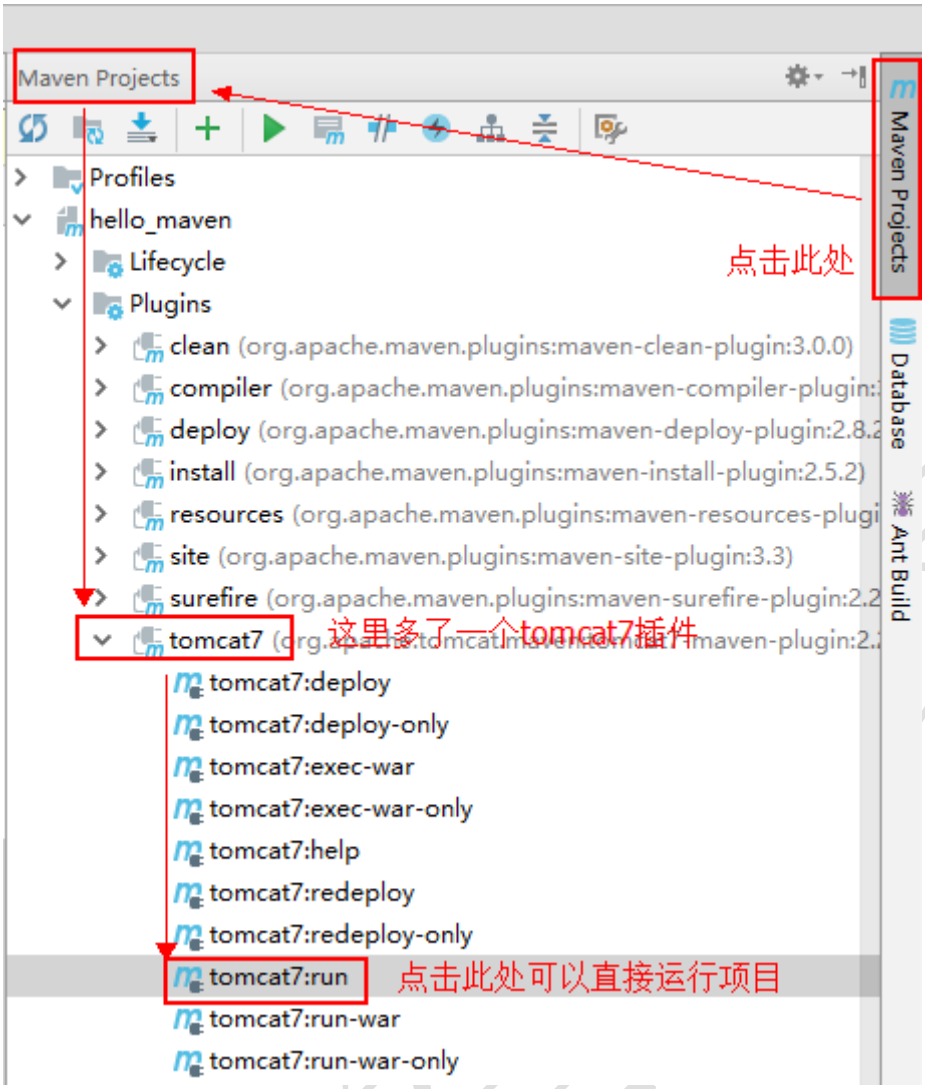
```
<servlet>
  <servlet-name>servletDemo</servlet-name>
  <servlet-class>com.itheima.servlet.ServletDemo</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>servletDemo</servlet-name>
  <url-pattern>/maven</url-pattern>
</servlet-mapping>
```

3.2.9 添加 tomcat7 插件

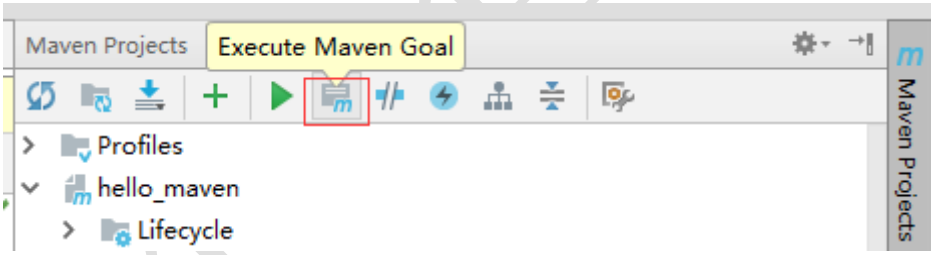
在 pom 文件中添加如下内容

```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <configuration>
    <port>8080</port>   端口号
    <path>/</path>      访问路径
  </configuration>
</plugin>
```

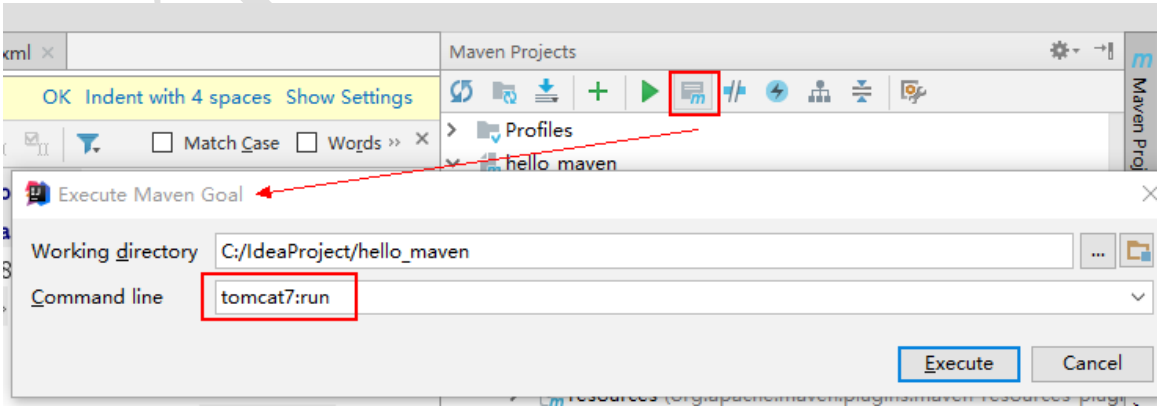
此时点击 idea 最右侧 Maven Projects，
就可以看到我们新添加的 tomcat7 插件
双击 tomcat7 插件下 tomcat7:run 命令直接运行项目



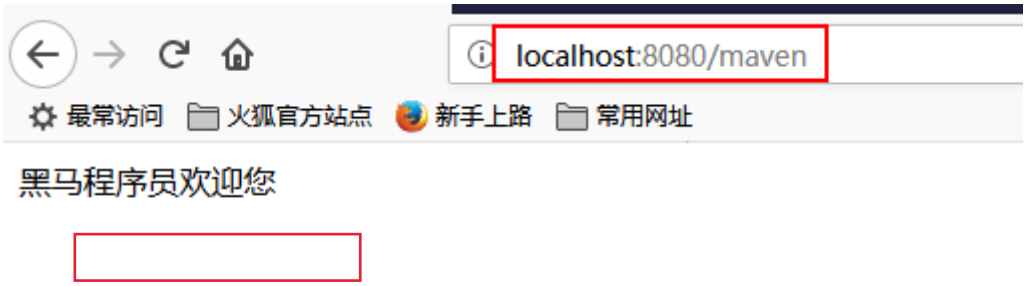
也可以直接点击如图按钮，手动输入 tomc7:run 命令运行项目



点击后弹出如下图窗口



3.2.10运行结果



第4章 maven 工程运行调试

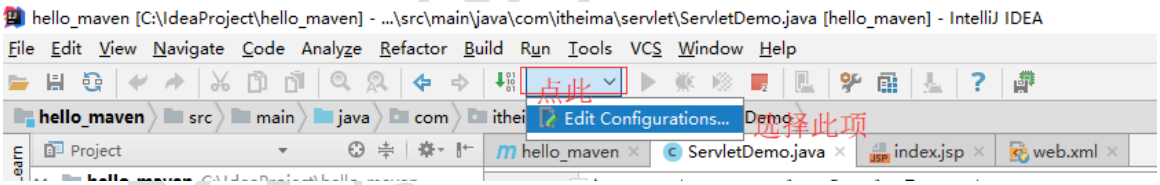
4.1 端口占用处理

重新执行 tomcat:run 命令重启工程，重启之前需手动停止 tomcat，否则报下边的错误：

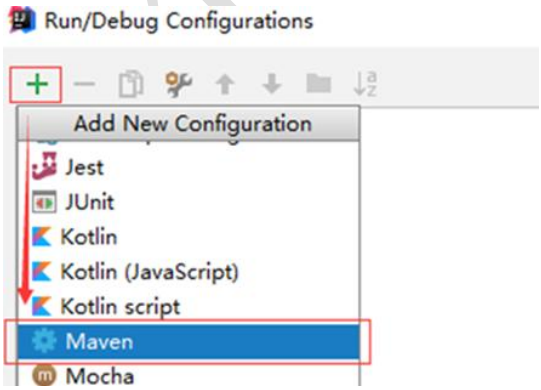
```
严重: Failed to initialize end point associated with ProtocolHandler ["http-bio-8080"]
java.net.BindException: Address already in use: JVM_Bind <null>:8080
    at org.apache.tomcat.util.net.JIoEndpoint.bind(JIoEndpoint.java:407)
```

4.2 断点调试

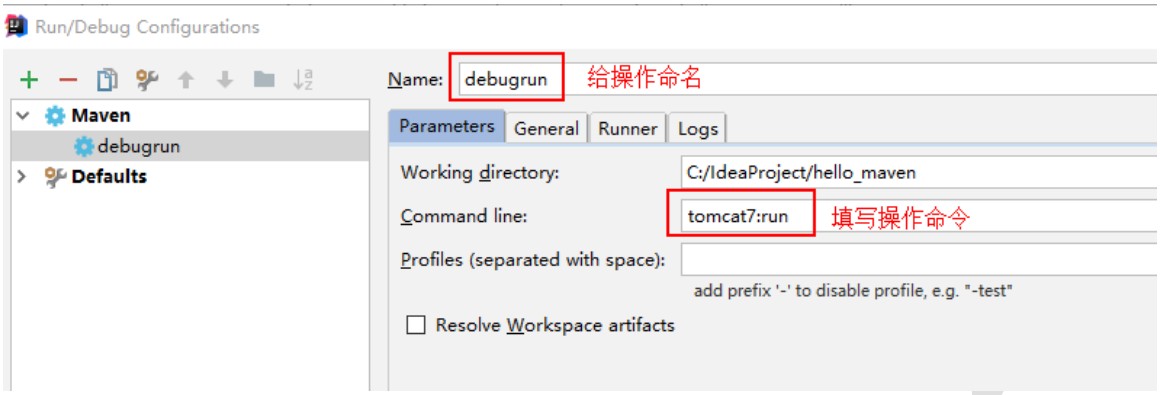
点击如图所示选项



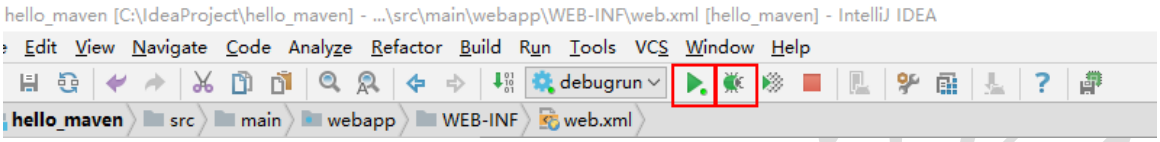
在弹出框中点击如图加号按钮找到 maven 选项



在弹出窗口中填写如下信息



完成后先 Apply 再 OK 结束配置后，可以在主界面找到我们刚才配置的操作名称。



如上图红框选中的两个按钮，左侧是正常启动，右侧是 debug 启动。

第5章 总结

5.1 maven 仓库

1、maven 仓库的类型有哪些？
2、maven 工程查找仓库的流程是什么？
3、本地仓库如何配置？

5.2 常用的 maven 命令

常用 的 maven 命令包括：
compile：编译
clean：清理
test：测试
package：打包
install：安装

5.3 坐标定义

在 pom.xml 中定义坐标，内容包括：groupId、artifactId、version，详细内容如下：

```
<!--项目名称，定义为组织名+项目名，类似包名-->
<groupId>cn.itcast.maven</groupId>
<!-- 模块名称 -->
```

<artifactId>maven-first</artifactId>
<!-- 当前项目版本号，snapshot 为快照版本即非正式版本，release 为正式发布版本 -->
<version>0.0.1-SNAPSHOT</version>
<packaging > ： 打包类型
jar：执行 package 会打成 jar 包
war：执行 package 会打成 war 包
pom ：用于 maven 工程的继承，通常父工程设置为 pom

5.4 pom 基本配置

pom.xml 是 Maven 项目的核心配置文件，位于每个工程的根目录，基本配置如下：

<project > ： 文件的根节点 .
<modelversion > ： pom.xml 使用的对象模型版本
<groupId > ： 项目名称，一般写项目的域名
<artifactId > ： 模块名称，子项目名或模块名称
<version > ： 产品的版本号 .
<packaging > ： 打包类型，一般有 jar、war、pom 等
<name > ： 项目的显示名，常用于 Maven 生成的文档。
<description > ： 项目描述，常用于 Maven 生成的文档
<dependencies> ： 项目依赖构件配置，配置项目依赖构件的坐标
<build> ： 项目构建配置，配置编译、运行插件等。

二、 maven 构建 SSM 工程[应用]

1. 需求

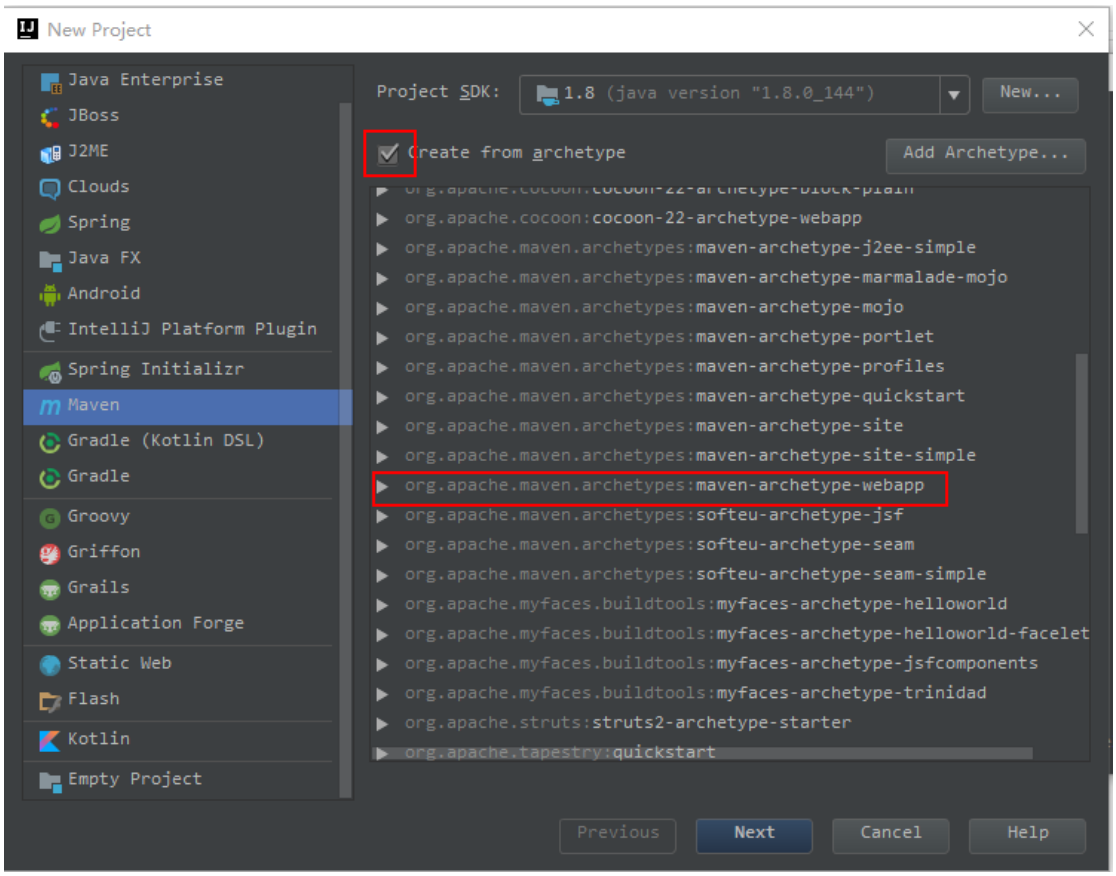
实现 SSM 工程构建，规范依赖管理。场景：根据 id 展示商品信息

2. 准备数据库

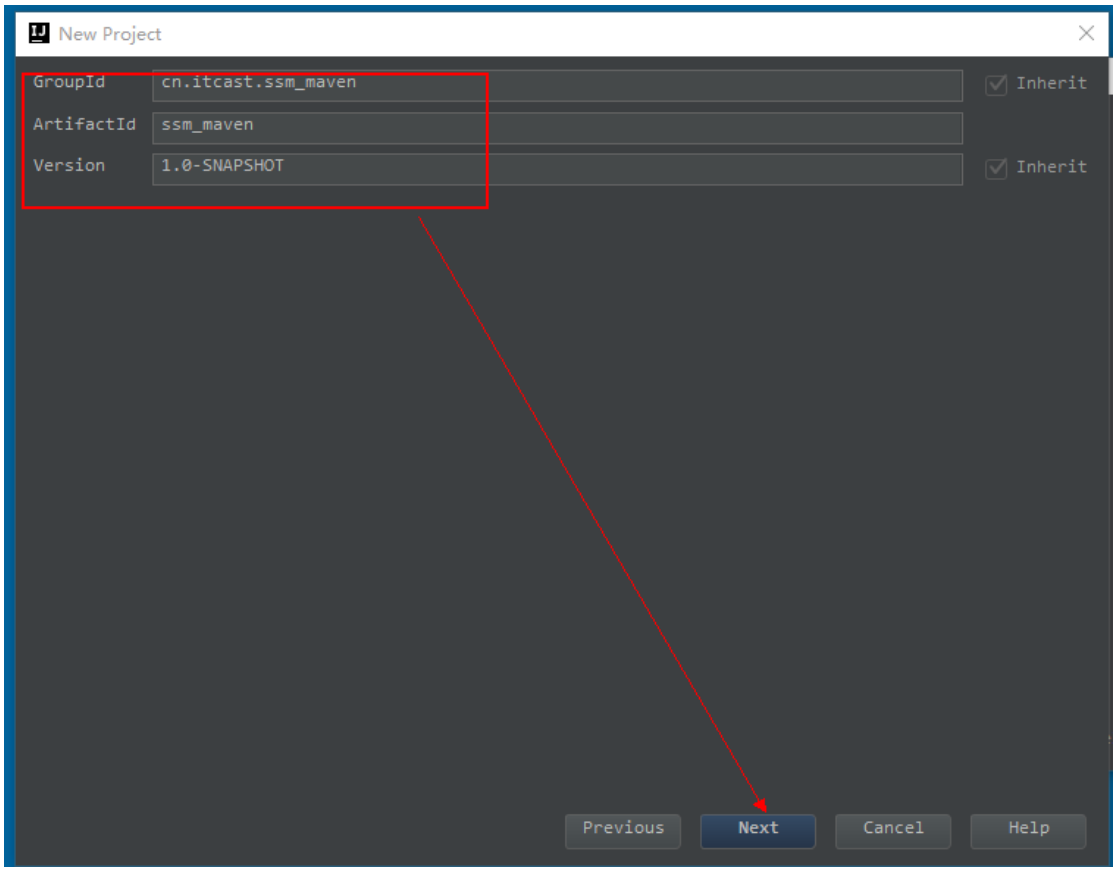
导入以下语句

3. 创建一个 maven 工程

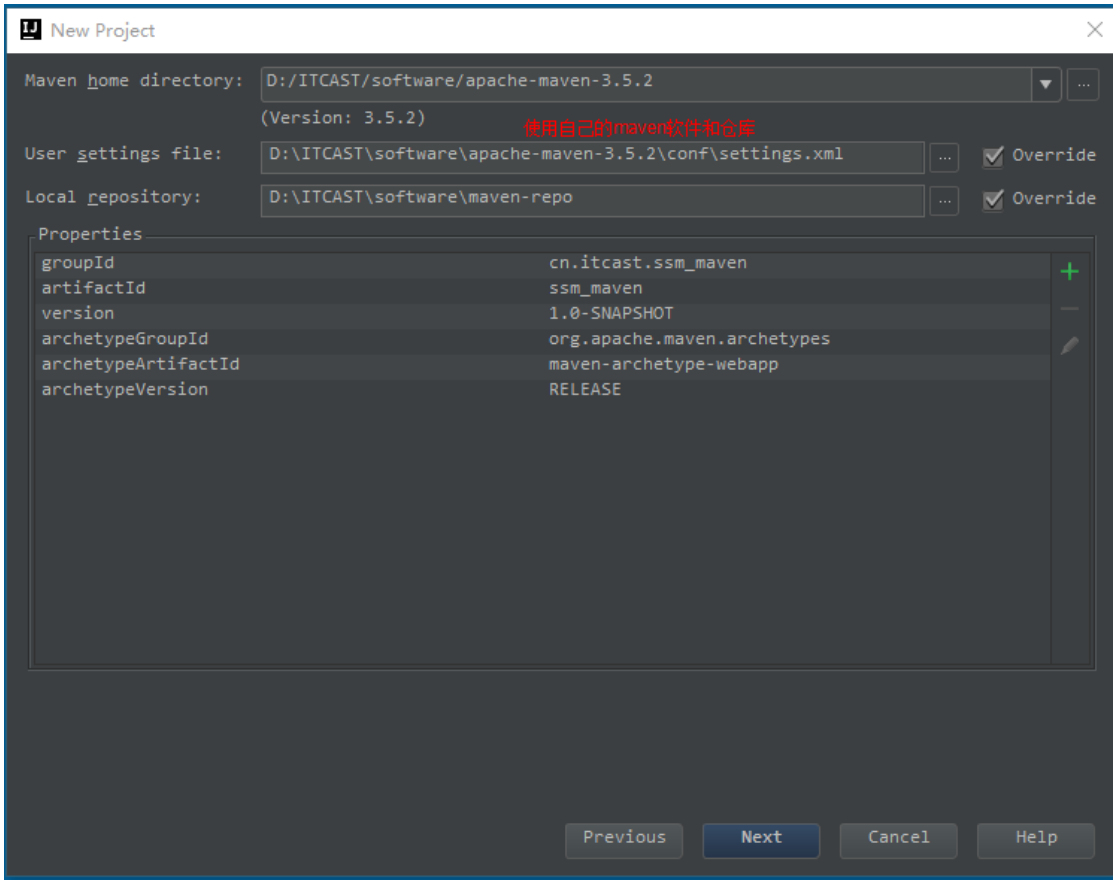
1、新建一个 ssm_maven 项目,使用下图选中的骨架



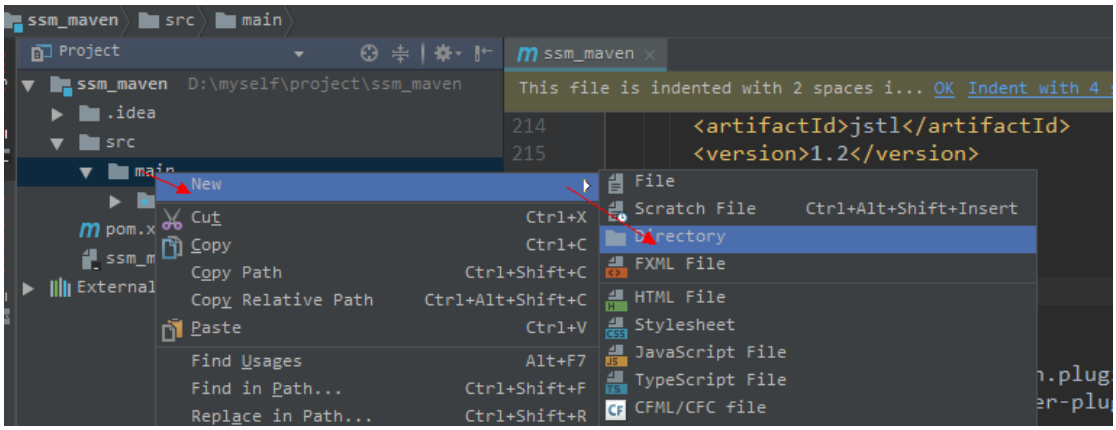
2、填写坐标



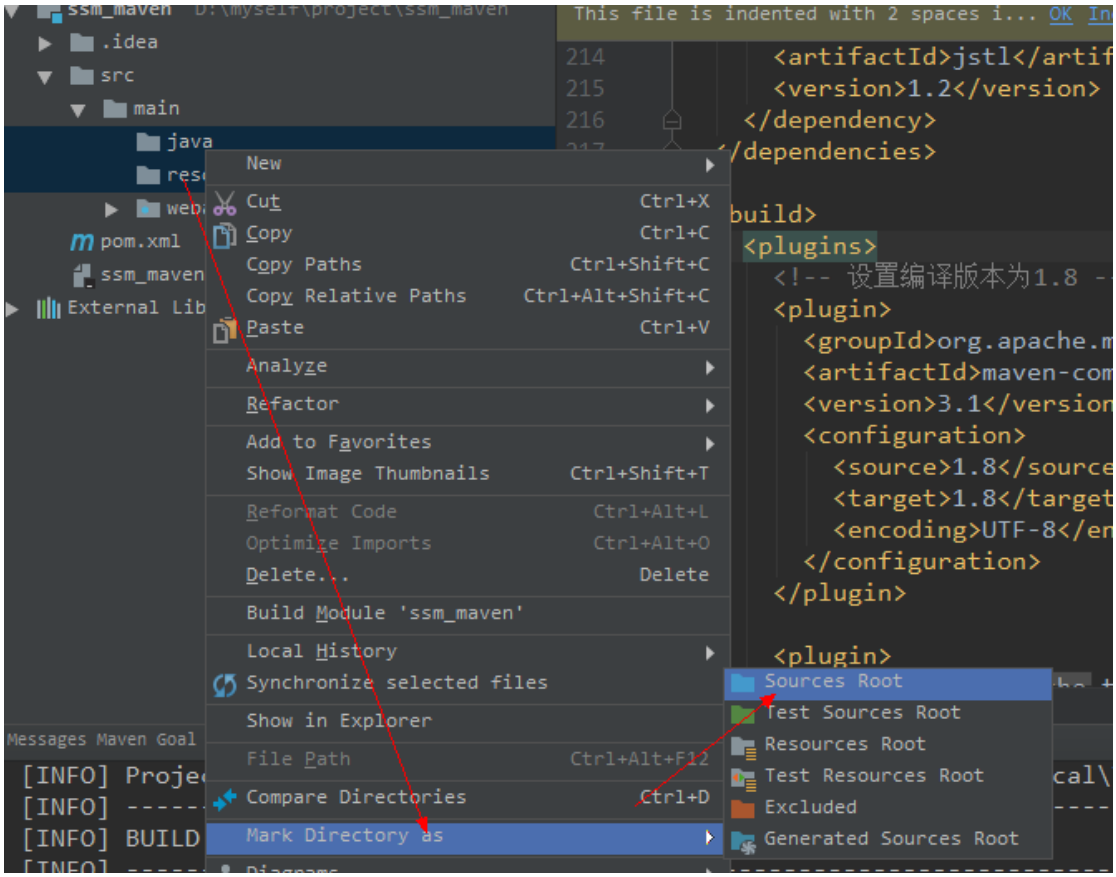
3、查看是否使用的自己的私服



5、在 main 目录下新建 java 和 resources 文件夹



6、把 java 和 resources 文件夹转成 source root



7、修改编译版本，在 pom.xml 文件中添加

```
<build>
  <plugins>
    <!-- 设置编译版本为 1.8 -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version >
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>
    </plugins>
</build>
```

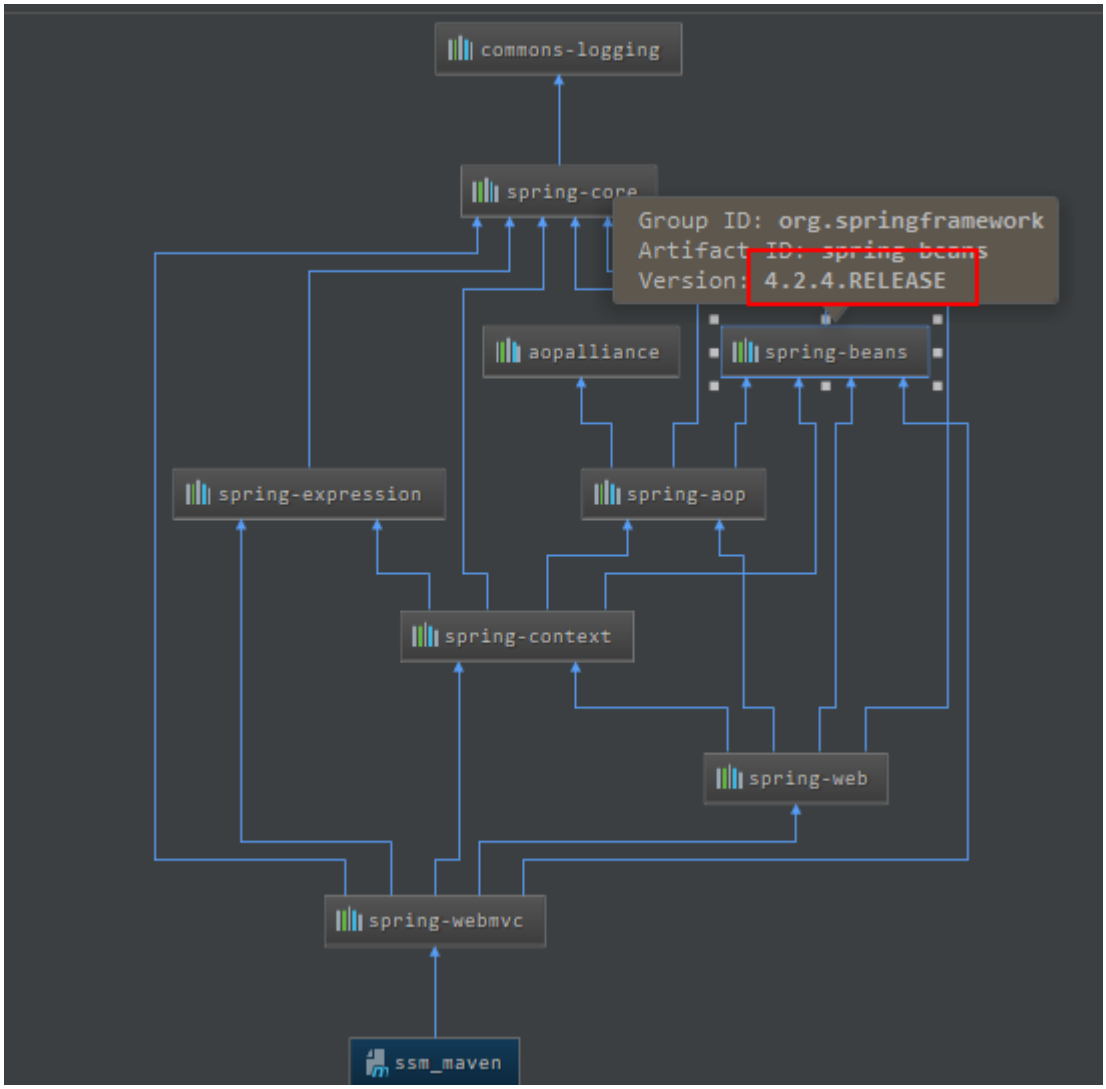
4. 知识点准备

4.1 什么是依赖传递

先添加 springmvc 的核心依赖的坐标

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.2.4.RELEASE</version>
  </dependency>
</dependencies>
```

会发现出现除了 spring-webmvc 以外的其他 jar。因为我们的项目依赖 spring-webmv.jar，而 spring-webmv.jar 会依赖 spring-beans.jar 等等，所以 spring-beans.jar 这些 jar 包也出现在了我们的 maven 工程中，这种现象我们称为依赖传递。从下图中可看到他们的关系：（请注意 spring-beans 的版本）

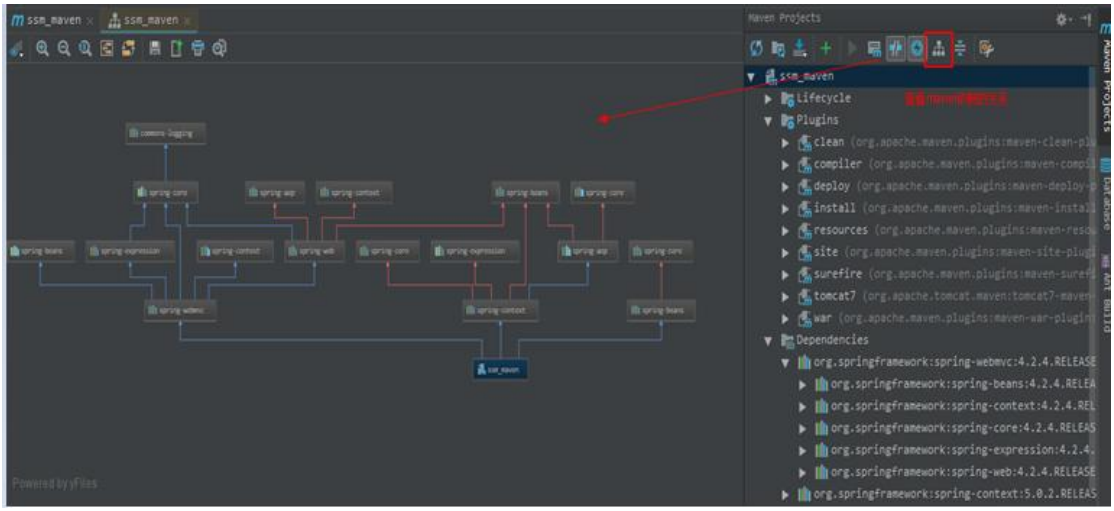


4.2 依赖冲突的解决

接着添加一个依赖

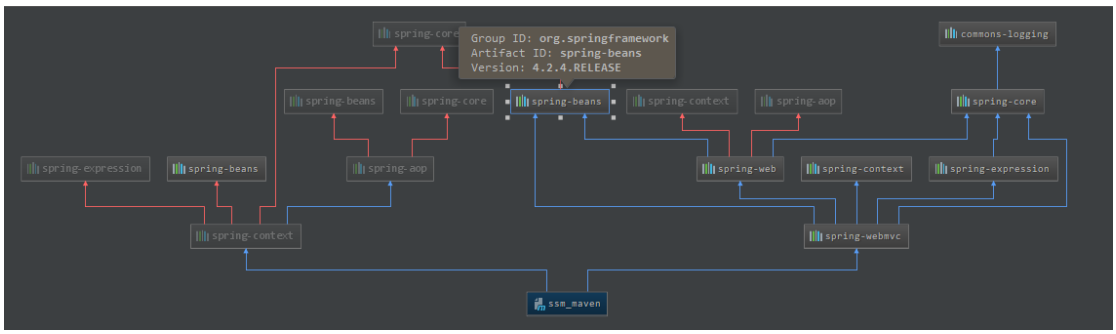
```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.2.4.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>

</dependencies>
```

我们会发现这两个 jar 包同时都依赖了 spring-beans



但是

spring-webmvc 依赖 spirng-beans-4.2.4，spring-context 依赖 spring-beans-5.0.2，但是发现 spirng-beans-4.2.4 加入到工程中



而我们希望 spring-beans-5.0.2 加入工程。这就造成了依赖冲突。解决依赖冲突有以下原则：

4.2.1 依赖调解原则

maven 自动按照下边的原则调解：

✓ **1、第一声明者优先原则**

在 pom 文件定义依赖，先声明的依赖为准。

测试：

如果将上边 spring-webmvc 和 spring-context 顺序颠倒，系统将导入 spring-beans-5.0.2。

分析：

由于 spring-webmvc 在前边以 spring-webmvc 依赖的 spring-beans-5.0.2 为准，所以最终

spring-beans-5.0.2 添加到了工程中。

✓ **2、路径近者优先原则** **==直接依赖原则**

例如：还是上述情况，spring-contex 和 spring-webmvc 都会传递过来 spirng-beans，那如果直接把 spring-beans 的依赖直接写到 pom 文件中，那么项目就不会再使用其他依赖传递来的 spring-beans，因为自己直接在 pom 中定义 spring-beans 要比其他依赖传递过来的路径要近。

在本工程中的 pom 中加入 spirng-beans-5.0.2 的依赖，根据路径近者优先原则，系统将导入 spirng-beans-5.0.2：

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.2.4.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>
</dependencies>
```

4.2.2 排除依赖

上边的问题也可以通过排除依赖方法辅助依赖调解，如下：

比如在依赖 spring-webmvc 的设置中添加排除依赖，排除 spring-beans，

下边的配置表示：依赖 spring-webmvc，但排除 spring-webmvc 所依赖的 spring-beans。

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.2.4.RELEASE</version>
    <exclusions>
      <exclusion>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>
</dependencies>
```

4.2.3 锁定版本

一个项目依赖另一个项目，

面对众多的依赖，有一种方法不用考虑依赖路径、声明优化等因素可以采用直接锁定版本的方法确定依赖构件的版本，版本锁定后则不考虑依赖的声明顺序或依赖的路径，以锁定的版本的为准添加到工程中，此方法在企业开发中常用。

如下的配置是锁定了 spring-beans 和 spring-context 的版本：

```
<dependencyManagement>
  <dependencies>
    <!--这里锁定版本为5.0.2.RELEASE -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>5.0.2.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.0.2.RELEASE</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

还可以把版本号提取出来，使用<properties>标签设置成变量。

```
<properties>
  <spring.version>5.0.2.RELEASE</spring.version>
</properties>

<dependencyManagement>
  <dependencies>
    <!--这里锁定版本为5.0.2.RELEASE -->
    <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>
  </dependencies>
</dependencyManagement>
```

注意：在工程中锁定依赖的版本并不代表在工程中添加了依赖，如果工程需要添加锁定版本的依赖则需要单独添加<dependencies></dependencies>标签，如下：

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
  </dependency>
</dependencies>
```

上边添加的依赖并没有指定版本，原因是已在<dependencyManagement>中锁定了版本，所以在<dependency>下不需要再指定版本。

5. 定义 pom.xml

maven 工程首先要识别依赖，web 工程实现 SSM 整合，需要依赖 spring-webmvc5.0.2、spring5.0.2、mybatis3.4.5 等，在 pom.xml 添加工程如下依赖：

（在实际企业开发中会有架构师专门来编写 pom.xml）

分两步：

1）锁定依赖版本

2）添加依赖

```
<?xml version="1.0" encoding="UTF-8"?>

<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>cn.itcast.ssm_maven</groupId>
  <artifactId>ssm_maven</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <properties>
    <spring.version>5.0.2.RELEASE</spring.version>
    <springmvc.version>5.0.2.RELEASE</springmvc.version>
    <mybatis.version>3.4.5</mybatis.version>
  </properties>

<!--锁定依赖版本-->
  <dependencyManagement>
    <dependencies>
      <!-- Mybatis -->
      <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>${mybatis.version}</version>
      </dependency>

      <!-- springMVC -->
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${springmvc.version}</version>
      </dependency>

      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
      </dependency>

      <!-- spring -->
      <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>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</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-beans</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-context-support</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
```

```
        <version>${spring.version}</version>
      </dependency>
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>${spring.version}</version>
      </dependency>

    </dependencies>

  </dependencyManagement>
<!--添加依赖-->
  <dependencies>
    <!-- Mybatis 和 mybatis 与 spring 的整合 -->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
    </dependency>
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis-spring</artifactId>
      <version>1.3.1</version>
    </dependency>

    <!-- MySql 驱动 -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.32</version>
    </dependency>

    <!-- druid 数据库连接池 -->
    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>druid</artifactId>
      <version>1.0.9</version>
    </dependency>

    <!-- springMVC 核心-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
    </dependency>

    <!-- spring 相关 -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
    </dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-expression</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
</dependency>


<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aspects</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-tx</artifactId>
</dependency>
<!-- junit 测试 -->
<dependency>
```

```
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
  </dependency>

  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.0</version>
    <scope>provided</scope>
  </dependency>

  <!-- jstl -->
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <!-- 设置编译版本为 1.8 -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version >
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>

    <plugin>
      <groupId>org.apache.tomcat.maven</groupId>
      <artifactId>tomcat7-maven-plugin</artifactId>
      <version>2.2</version>
      <configuration>
        <path>/</path>
        <port>8080</port>
      </configuration>
```

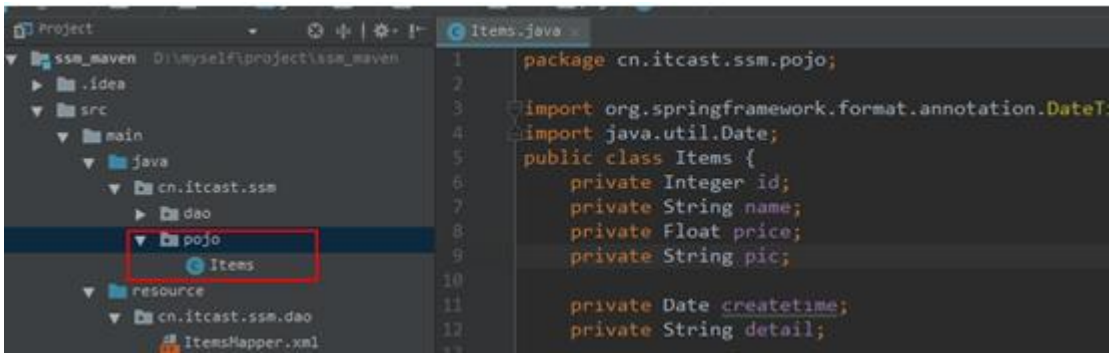
```
      </plugin>
    </plugins>
  </build>
</project>
```

6. Dao 层

在 src/main/java 中定义 dao 接口，实现根据 id 查询商品信息：

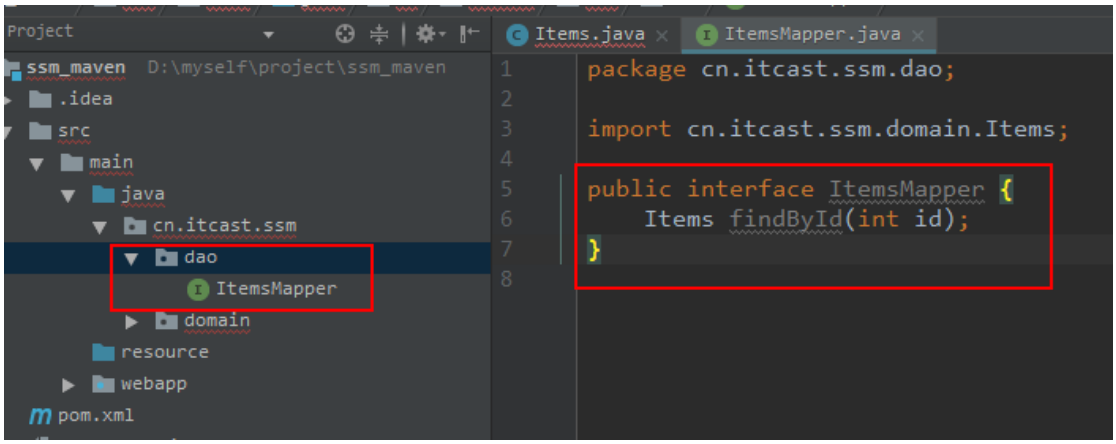
6.1 pojo 模型类

在 src/main/java 创建模型类



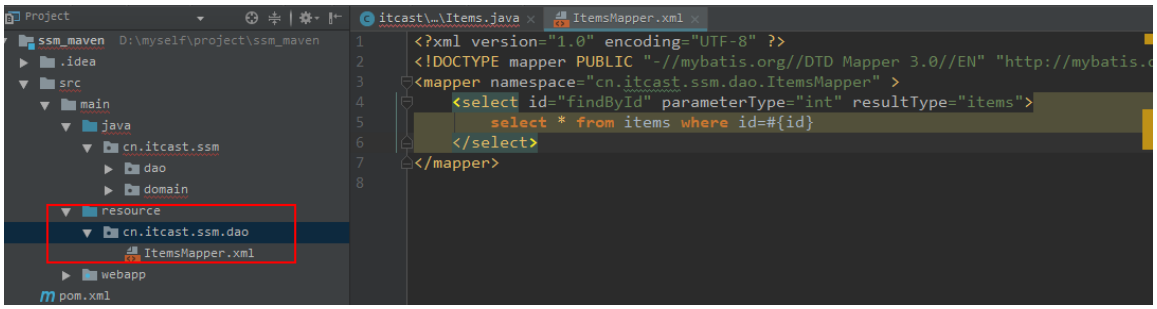

```
public class Items {
    private Integer id;
    private String name;
    private Float price;
    private String pic;

    private Date createtime;
    private String detail;
      ……
}
```

6.2dao 层代码



6.3配置文件

注意配置文件的位置



- 内容如下

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="cn.itcast.ssm.dao.ItemsMapper" >
    <select id="findById" parameterType="int" resultType="items">
        select * from items where id=#{id}
    </select>
</mapper>
```

- 在 src/main/resources 创建 applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
   http://www.springframework.org/schema/context
```

```
http://www.springframework.org/schema/context/spring-context-4.0.xsd
   http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
   http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">
    <!-- 数据库连接池 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <!-- 驱动 -->
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <!-- url -->
        <property name="url" value="jdbc:mysql://localhost:3306/maven" />
        <!-- 用户名 -->
        <property name="username" value="root" />
        <!-- 密码 -->
        <property name="password" value="root" />
    </bean>
    <!-- mapper 配置 -->
    <!-- 让 spring 管理 sqlsessionfactory 使用 mybatis 和 spring 整合包中的 -->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <!-- 数据库连接池 -->
        <property name="dataSource" ref="dataSource" />
        <property name="typeAliasesPackage"
value="cn.itcast.ssm.pojo"></property>
    </bean>

    <!-- mapper 扫描器 : 用来产生代理对象-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="cn.itcast.ssm.dao"></property>
    </bean>


</beans>
```

- 在 src/main/resources 配置 log4j.properties

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5
p %c{1}:%L - %m%n
### set log levels - for more verbose logging change 'info' to
'debug' ###
#在开发阶段日志级别使用 debug
log4j.rootLogger=debug, stdout
### 在日志中输出 sql 的输入参数 ###
log4j.logger.org.hibernate.type=TRACE
```

6.4 单元测试

在 src/test/java 创建单元测试类

```
public class ItemsMapperTest {
   @Test
   public void testFindItemsById() {
      //获取 spring 容器
      ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("classpath:applicationContext.xml");
      //获取 Mapper
      ItemsMapper itemsMapper =
applicationContext.getBean(ItemsMapper.class);
      //调用 Mapper 方法
      Items items = itemsMapper.findById(1);
      System.out.println(items);
   }
}
```

7. Service 层

7.1代码

```
@Service
@Transactional
public class ItemsServiceImpl implements ItemsService {

    @Autowired
    private ItemsMapper itemsMapper;
    @Override
    public Items findById(int itemId) {
        return itemsMapper.findById(itemId);
    }
}
```

7.2配置文件

在 applicationContext.xml 中配置 service

```
 <context:component-scan base-package="cn.itcast.ssm.service"/>
```

事务其他呢

8. Web 层

8.1代码

```
@Controller
@RequestMapping("/items/")
public class ItemsController {

   @Autowired
   private ItemsService itemsService ;
// 展示商品信息页面
   @RequestMapping("/showItem")
    public String showItem(int id,Model model){
      Items items = itemsService.findById(id);
      model.addAttribute("item", items);
      return "viewItem";
   }
}
```

8.2配置文件

- 在 springmvc.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.2.xsd
        http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.2.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.2.xsd">

    <context:component-scan
base-package="cn.itcast.ssm.controller"></context:component-scan>

    <!--  配置视图解析器的前缀和后缀 -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolve
```

```
r">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

Web.xml

加载 spring 容器，配置 springmvc 前端控制器

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
version="2.5">
  <!--   前端控制器 加载 springmvc 容器 -->
  <servlet>
    <servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servl
et-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>*.action</url-pattern>
  </servlet-mapping>
  <!--     监听器 加载 spring 容器 -->
  <listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</
listener-class>
  </listener>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:applicationContext*.xml</param-value>
  </context-param>
</web-app>
```

9. Jsp

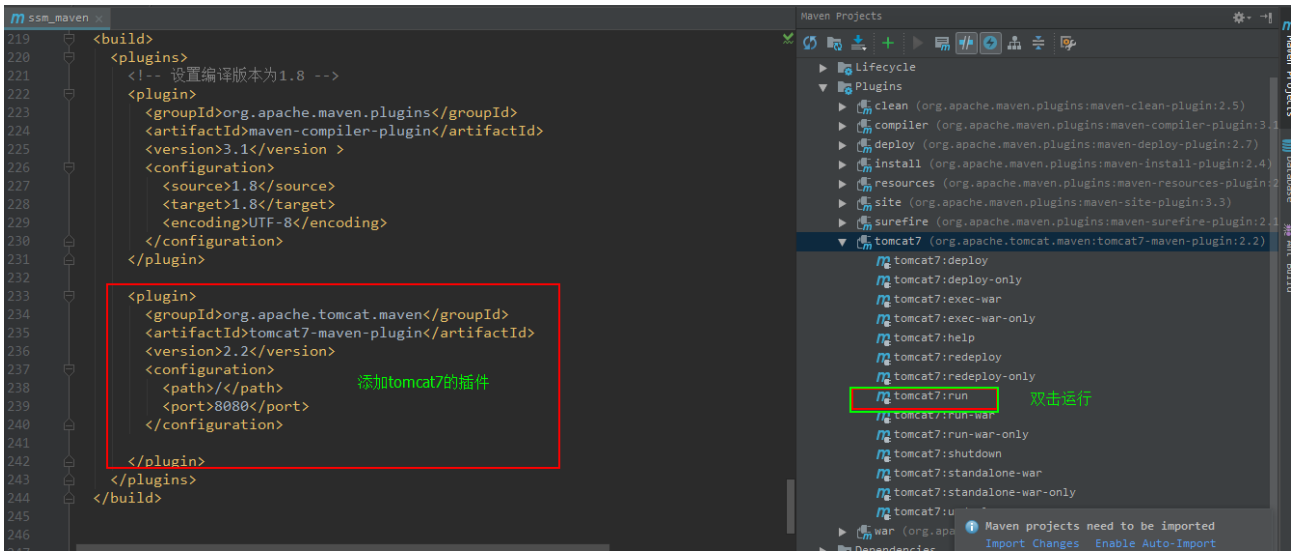
/WEB-INF/jsp/viewItem.jsp 如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

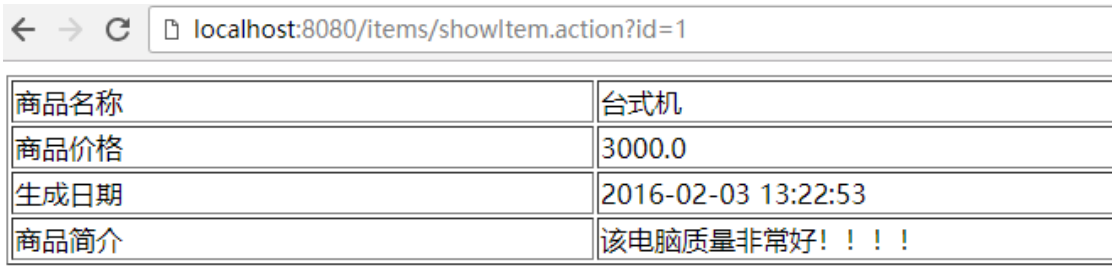
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt"  prefix="fmt"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>商品信息</title>
</head>
<body>
<form>
    <table width="100%" border=1>
        <tr>
            <td>商品名称</td>
            <td> ${item.name } </td>
        </tr>
        <tr>
            <td>商品价格</td>
            <td> ${item.price } </td>
        </tr>
        <tr>
            <td>生成日期</td>
            <td> <fmt:formatDate value="${item.createtime}"
pattern="yyyy-MM-dd HH:mm:ss"/> </td>
        </tr>
        <tr>
            <td>商品简介</td>
            <td>${item.detail} </textarea>
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

10. 运行与调试

添加 tomcat7 插件，双击右侧 tomcat7 运行



运行结果如下：



三、 分模块构建工程[应用]

基于上边的三个工程分析

继承：创建一个 parent 工程将所需的依赖都配置在 pom 中

聚合：聚合多个模块运行。

jar包和代码是没有区别的，最后都是.class文件，让计算机识别的，因此没有关系。导入坐标就是导入jar包，有些东西是不需要打包的，因此jar没有。

1. 需求

难道resource包不会冲突吗？

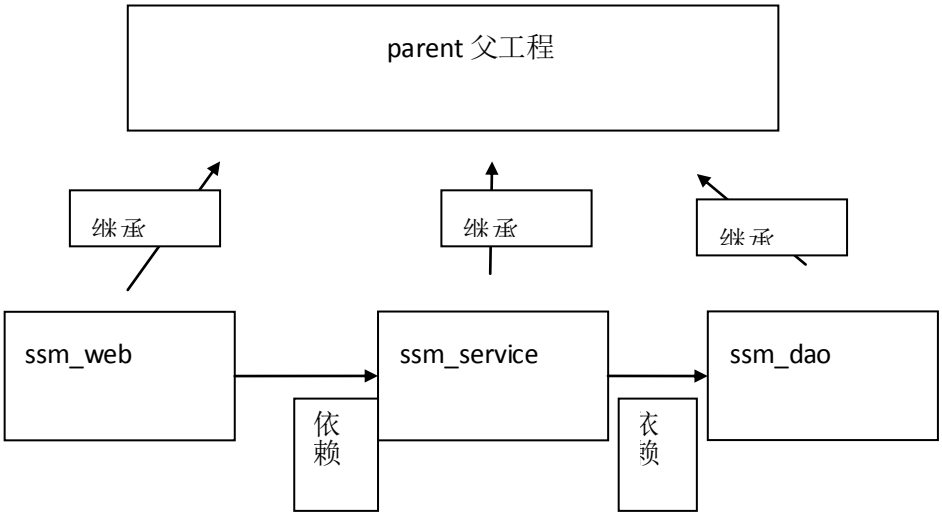
需求描述

将 SSM 工程拆分为多个模块开发：

ssm_dao

ssm_service

ssm_web



理解继承和聚合

通常继承和聚合同时使用。

■ 何为继承?

继承是为了消除重复，如果将 dao、service、web 分开创建独立的工程则每个工程的 pom.xml 文件中的内容存在重复，比如：设置编译版本、锁定 spring 的版本的等，可以将这些重复的配置提取出来在父工程的 pom.xml 中定义。

■ 何为聚合?

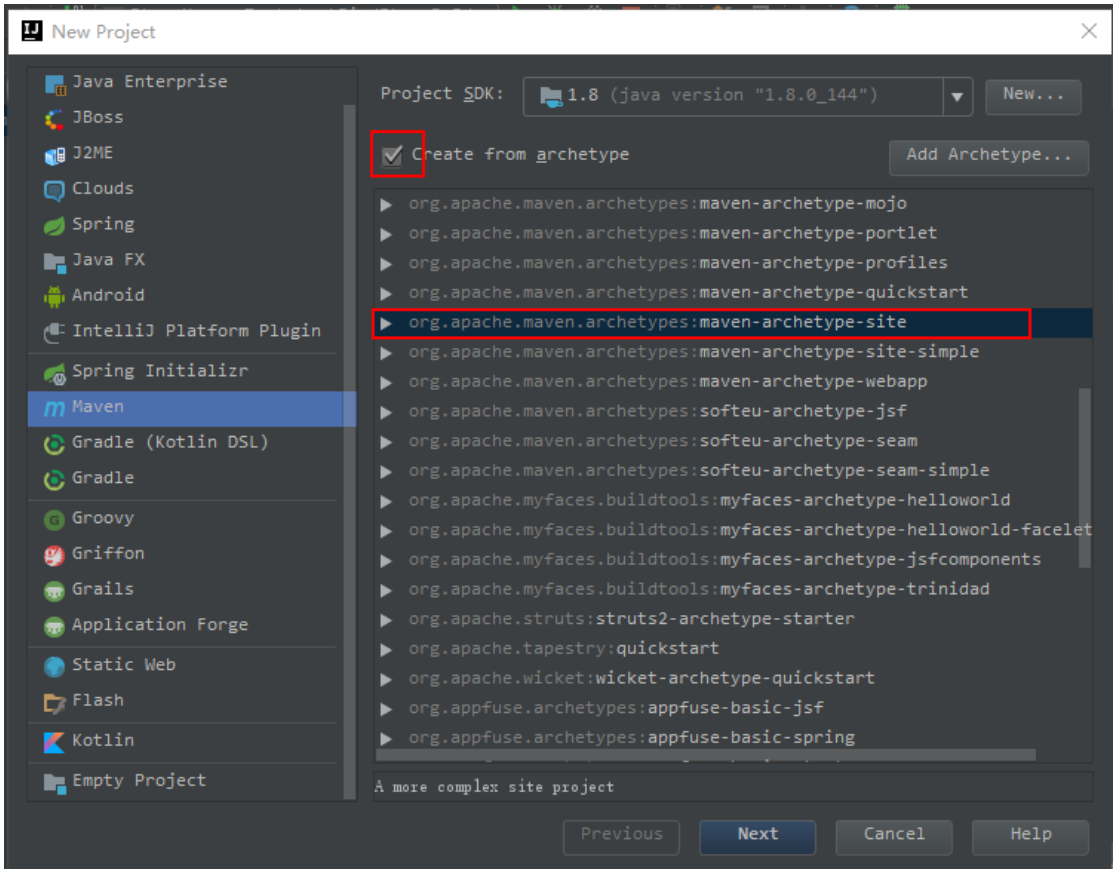
项目开发通常是分组分模块开发，每个模块开发完成要运行整个工程需要将每个模块聚合在一起运行，比如：dao、service、web 三个工程最终会打一个独立的 war 运行。

2. 案例实现

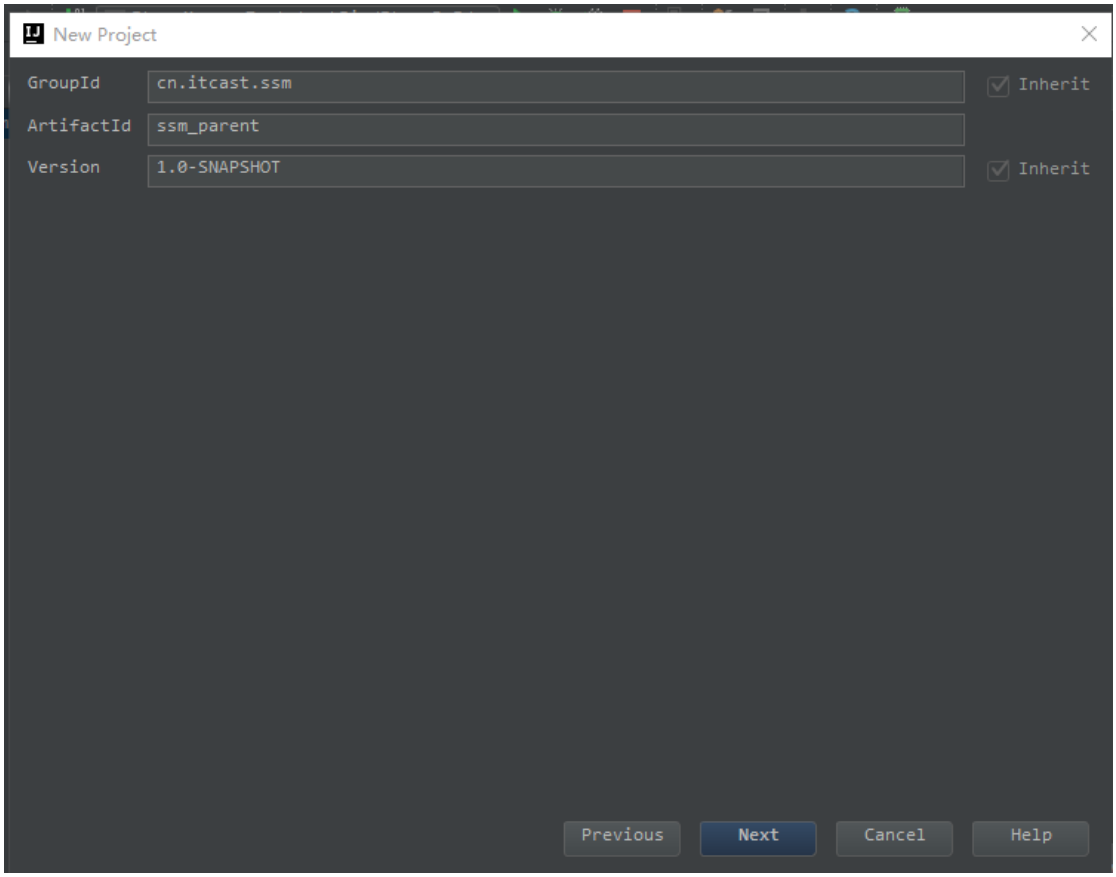
2.1 maven-parent 父模块

2.1.1 创建父工程

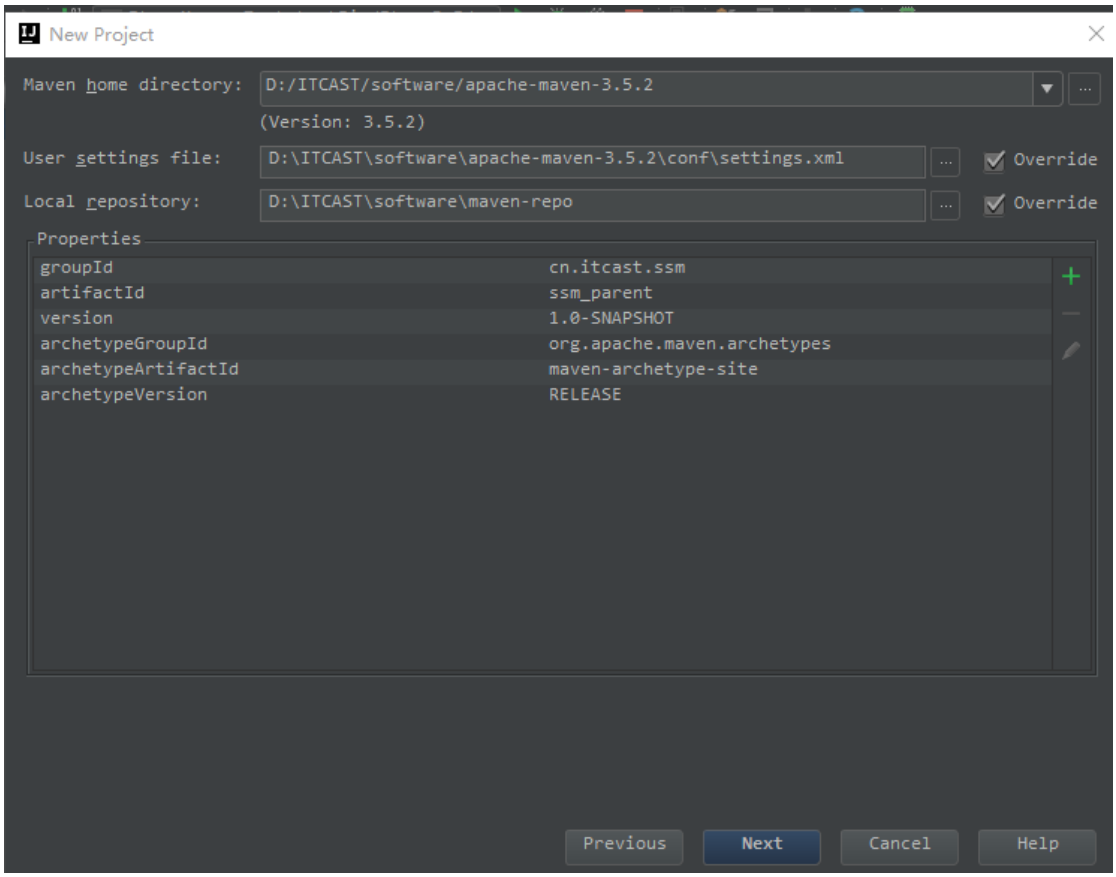
1、选择骨架创建父工程



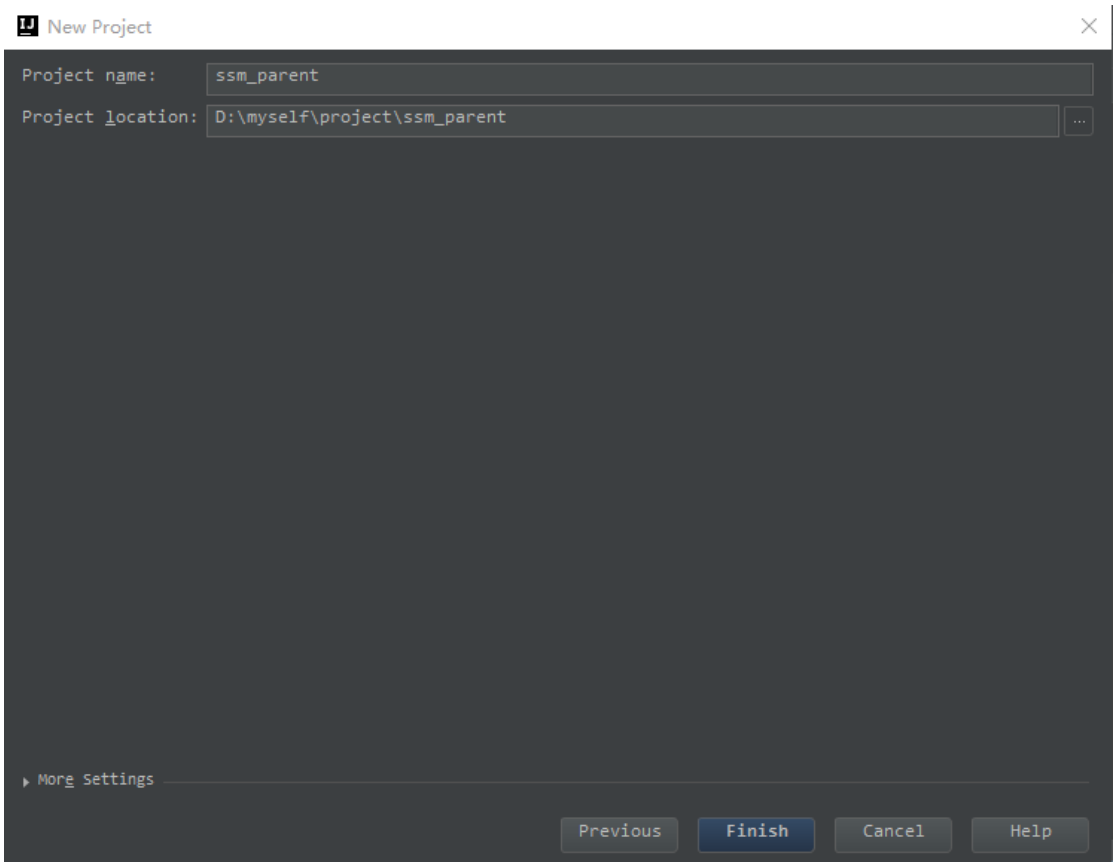
2、填写坐标



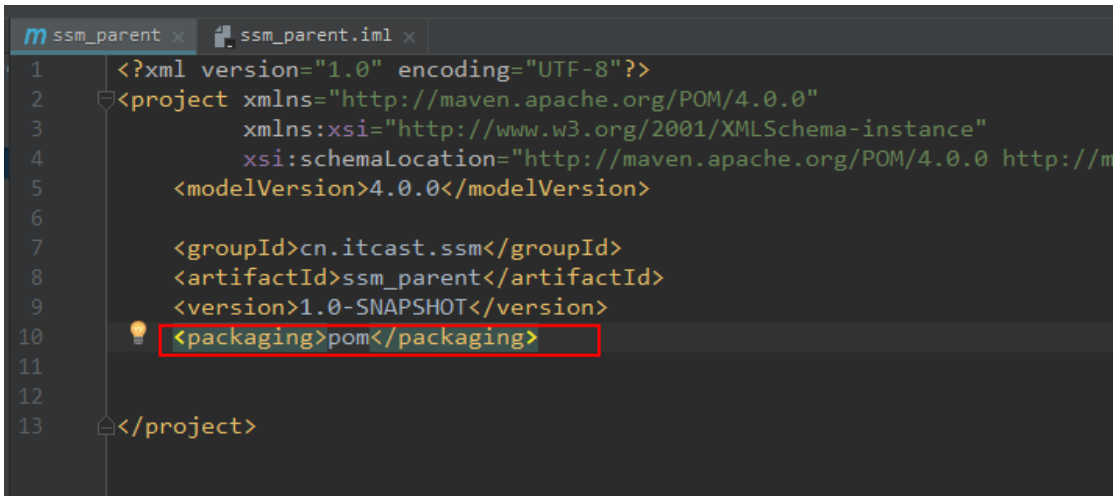
3、确认使用的是本地仓库



5、注意代码所在的路径（默认）



6、设置项目的打包方式



2.1.2 定义 pom.xml

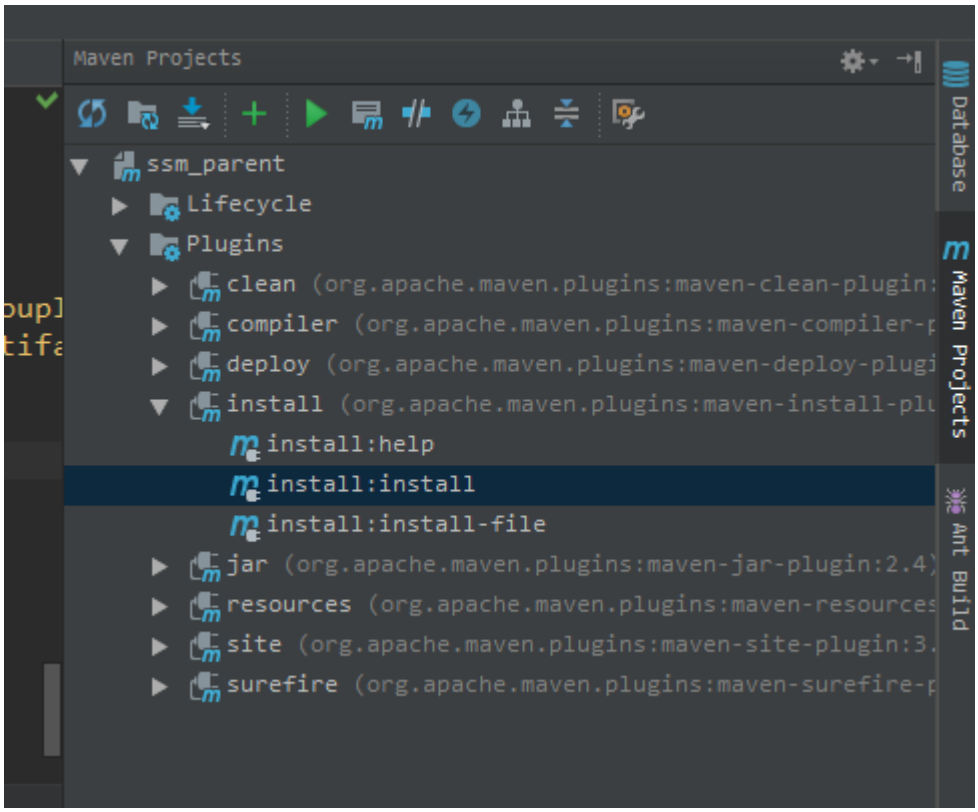
在父工程的 pom.xml 中抽取一些重复的配置的，比如：锁定 jar 包的版本、设置编译版本等。

```
<properties>
    <spring.version>5.0.2.RELEASE</spring.version>
    <springmvc.version>5.0.4.RELEASE</springmvc.version>
    <mybatis.version>3.4.5</mybatis.version>
</properties>


<dependencyManagement>
    <dependencies>
        <!-- Mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>${mybatis.version}</version>
        </dependency>

        <!-- springMVC -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${springmvc.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- spring -->
        <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>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</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-beans</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-context-support</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${spring.version}</version>
        </dependency>

    </dependencies>

</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.1</version>
            <configuration>
                <target>1.8</target>
                <source>1.8</source>
            </configuration>
        </plugin>
    </plugins>
</build>
```

2.1.3 将父工程发布至仓库

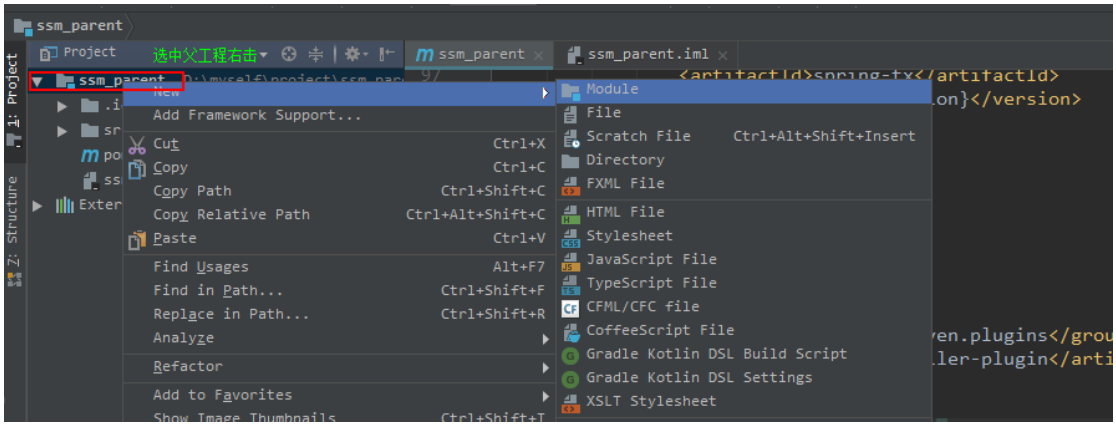
父工程创建完成执行 install 将父工程发布到仓库方便子工程继承：



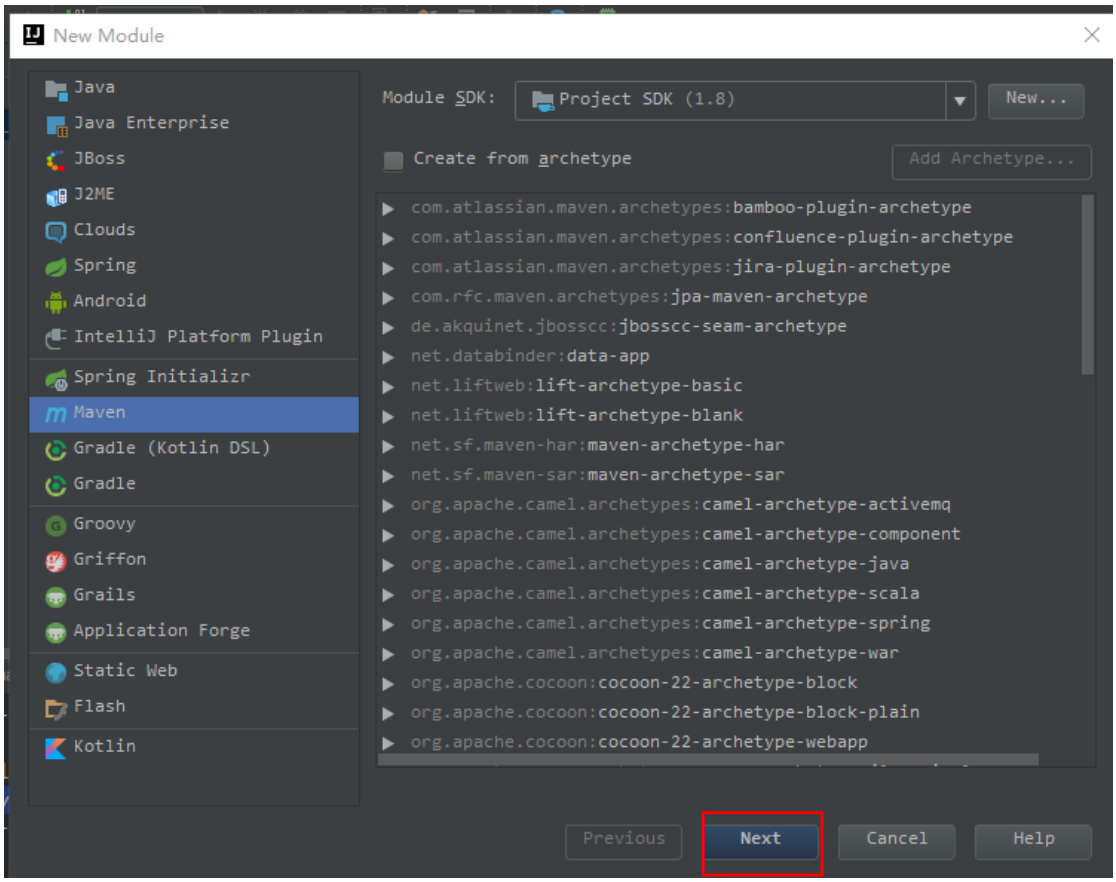
2.2 ssm_dao 子模块

2.2.1 创建 dao 子模块

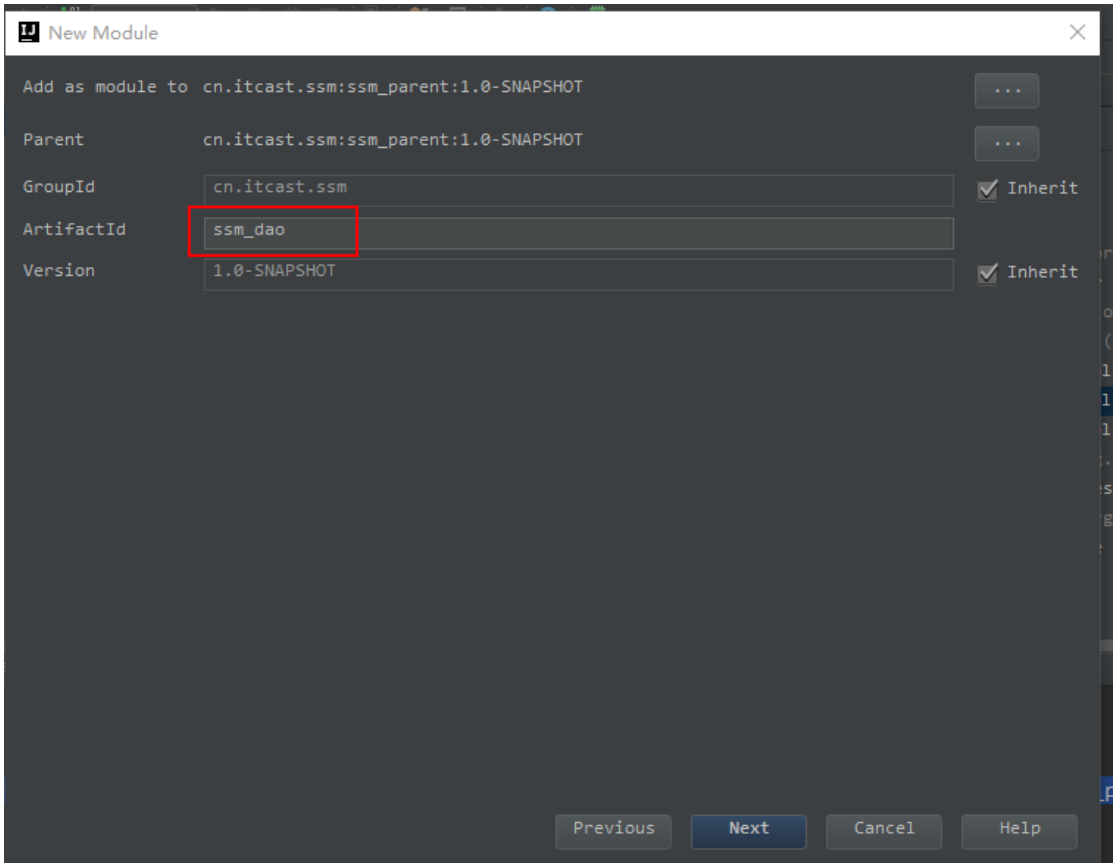
1、在父工程上右击创建 maven 模块：



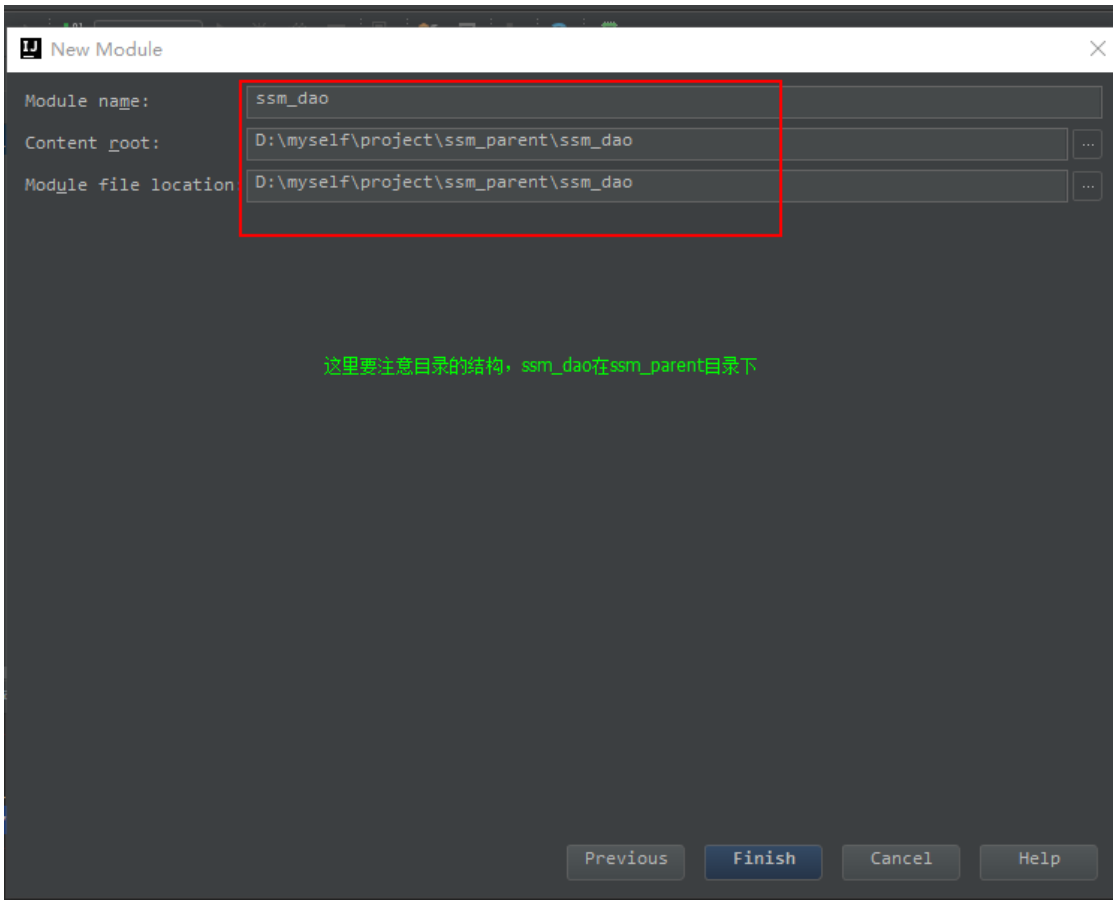
2、选择“跳过骨架选择”：



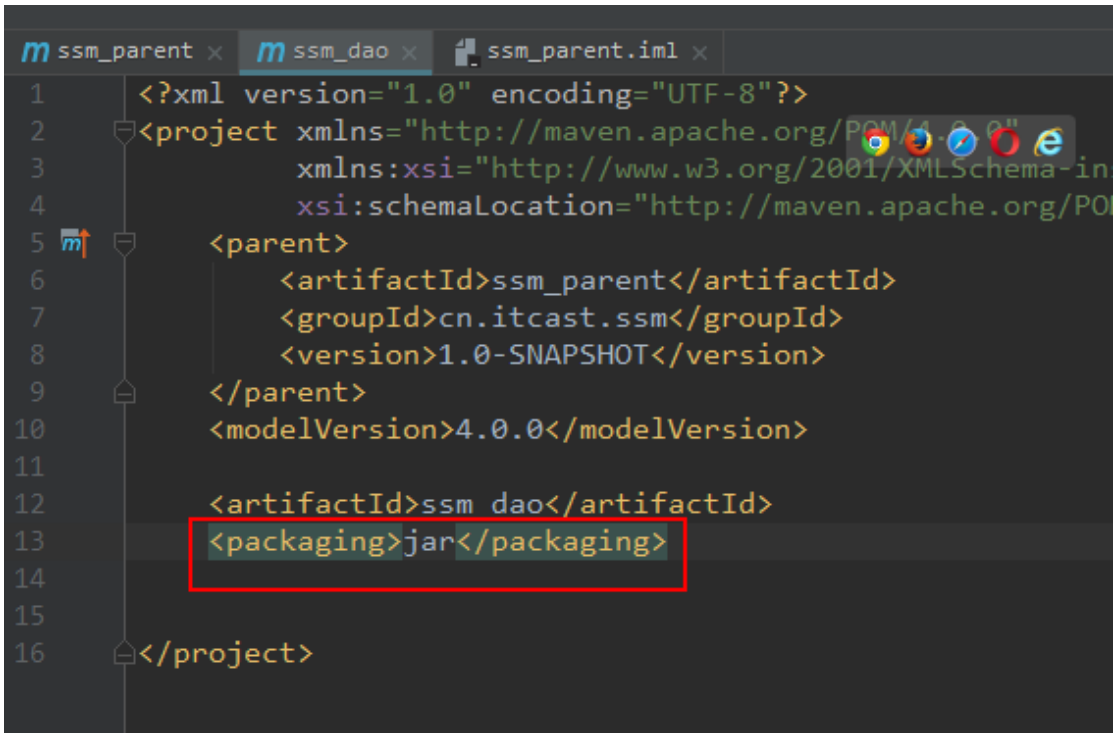
3、填写模块名称



4、下一步，确定项目的目录



5、打包方式是 jar



2.2.2 定义 pom.xml

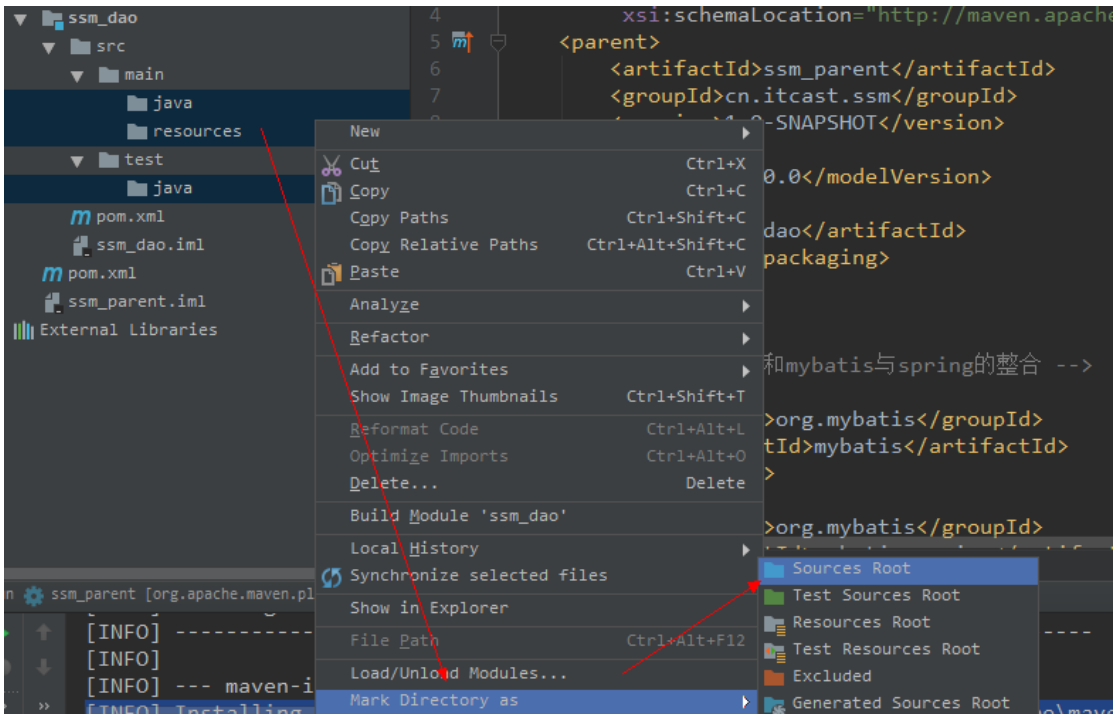
只添加到层的 pom，mybatis 和 spring 的整合相关依赖

```
<dependencies>
    <!-- Mybatis 和 mybatis 与 spring 的整合 -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
    </dependency>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>1.3.1</version>
    </dependency>

    <!-- MySql 驱动 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.32</version>
    </dependency>

    <!-- druid 数据库连接池 -->
    <dependency>
        <groupId>com.alibaba</groupId>
```

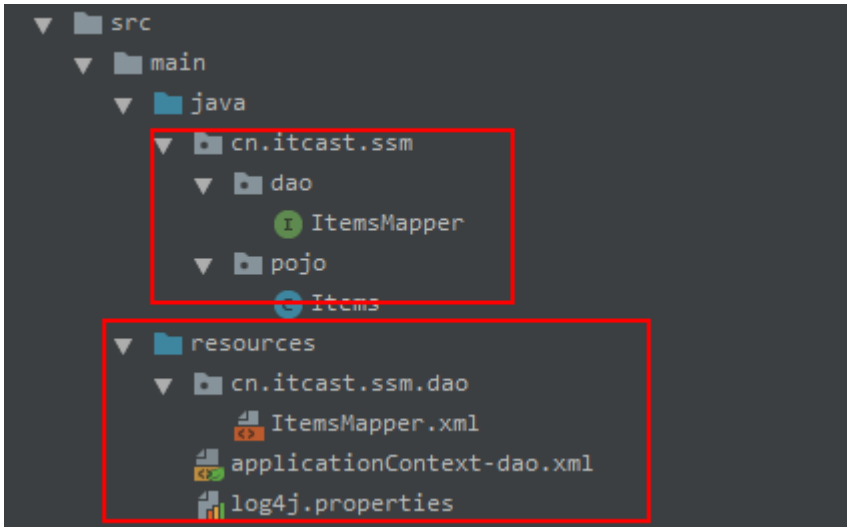
```
        <artifactId>druid</artifactId>
        <version>1.0.9</version>
    </dependency>

    <!-- spring 相关 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-expression</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aspects</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
    </dependency>
</dependencies>
```

2.2.3 dao 代码

把文件夹转成 sources root



将 ssm_maven 工程中的 dao 接口、映射文件及 pojo 类拷贝到 src/main/java 中：



2.2.4 配置文件

将 applicationContext.xml 拆分出一个 applicationContext-dao.xml，此文件中只配置 dao 相关

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
   http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
   http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
   http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-4.0.xsd">



    <!-- 数据库连接池 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
        <!-- 驱动 -->
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <!-- url -->
        <property name="url" value="jdbc:mysql://localhost:3306/maven" />
        <!-- 用户名 -->
        <property name="username" value="root" />
        <!-- 密码 -->
        <property name="password" value="root" />
    </bean>
    <!-- mapper 配置 -->
    <!-- 让 spring 管理 sqlsessionfactory 使用 mybatis 和 spring 整合包中的 -->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <!-- 数据库连接池 -->
        <property name="dataSource" ref="dataSource" />
        <property name="typeAliasesPackage"
value="cn.itcast.ssm.pojo"></property>
    </bean>

    <!-- mapper 扫描器 : 用来产生代理对象-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="cn.itcast.ssm.dao"></property>
    </bean>


</beans>
```

2.2.5 单元测试

https://blog.csdn.net/chaijunkun/article/details/35796335

测试只能使用本模块添加的jar包，不能使用jar包依赖，继承也是一种依赖关系

1、首先在 dao 模块的 pom.xml 添加 junit 的依赖，添加时 Scope 选择 test

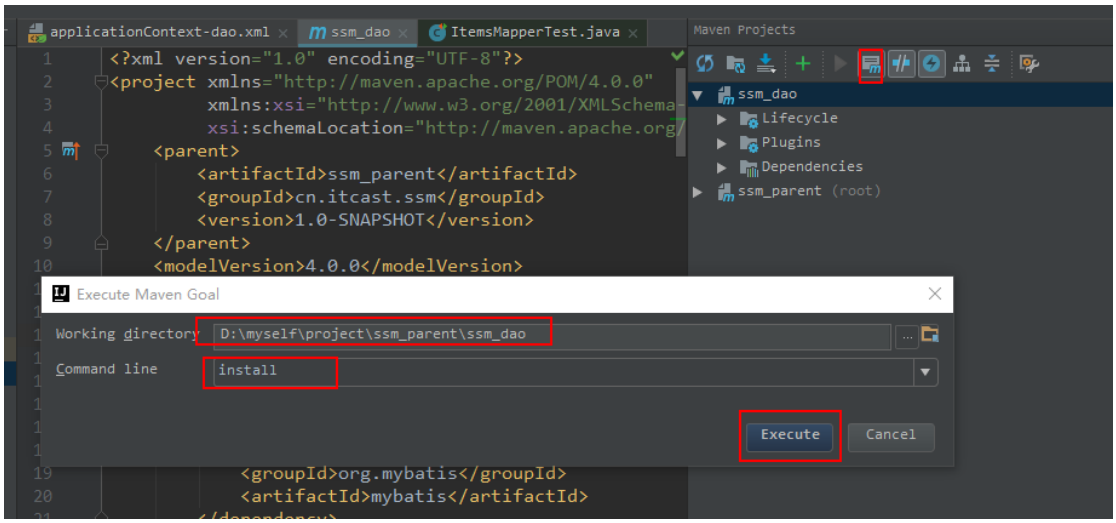
```
<!-- junit测试 -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

2、编写 junit 测试代码

```
public class ItemsMapperTest {
    @Test
    public void testFindItemsById() {
        //获取 spring 容器
        ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("classpath:applicationContext-dao.xml");
        //获取 Mapper
        ItemsMapper itemsMapper =
applicationContext.getBean(ItemsMapper.class);
        //调用 Mapper 方法
        Items items = itemsMapper.findById(1);
        System.out.println(items);
    }
}
```

2.2.6 把 dao 模块 install 到本地仓库

调过测试，install 到本地仓库



2.3 ssm_service 子模块

2.3.1 创建 service 子模块

方法同 ssm_dao 模块创建方法，模块名称为 ssm_service。

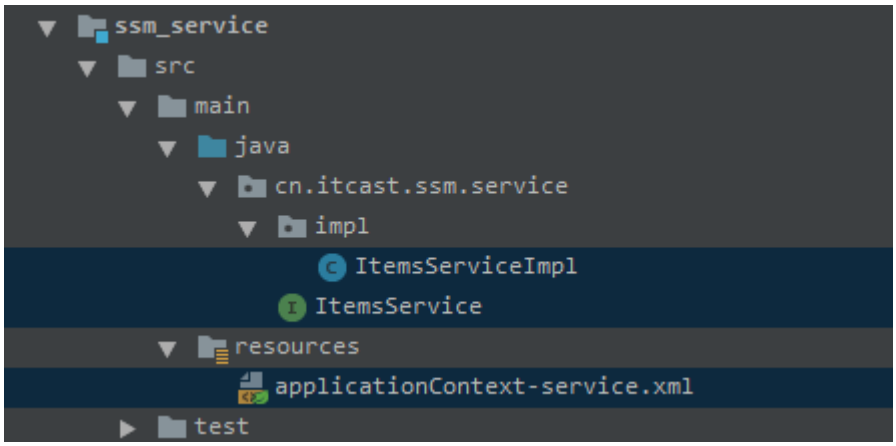
2.3.2 定义 pom.xml

ssm_service 模块的 pom.xml 文件中需要继承父模块，ssm_service 依赖 ssm_dao 模块，添加 spring 相关的依赖：

```
<dependencies>
    <!-- spring 相关 -->

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
    </dependency>
    <!--     dao 层的依赖 -->
    <dependency>
        <groupId>cn.itcast.ssm</groupId>
        <artifactId>ssm_dao</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>


</dependencies>
```

2.3.3 service 接口

将 ssm_maven 工程中的 service 接口拷贝到 src/main/java 中：



2.3.4 配置文件

创建 applicationContext-service.xml，此文件中定义的 service。

```
<context:component-scan base-package="cn.itcast.ssm"/>
```

2.3.5 依赖范围对传递依赖的影响（了解）

2.3.5.1 问题描述

当在写 junit 测试时发现，代码报出没有找不到类的错误信息：

```
import cn.itcast.ssm.pojo.Items;
import cn.itcast.ssm.service.ItemsService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ItemsServiceTest {
    @Test
    public void testFindItemsById() {
        //获取spring容器
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext( configLocation: "classpath*:applicationContext-*.xml");
        //获取Mapper
        ItemsService itemsService =  applicationContext.getBean(ItemsService.class);
        //调用Mapper方法
        Items items = itemsService.findById( itemId: 1);
        System.out.println(items);
    }
}
```

是因为没有 junit.jar 引起的!为什么会这样呢？我们 ssm_dao 模块中有 junit 依赖而 ssm_service 模块依赖了 ssm_dao，难道 junit 不应该传递过来吗？

2.3.5.3 依赖范围对传递依赖的影响

是因为依赖会有依赖范围，依赖范围对传递依赖也有影响，例如有 A、B、C，A 依赖 B、B 依赖 C，C 可能是 A 的传递依赖，如下图：

——的地方的jar不能使用，因为没有

直接依赖 \ 传递依赖	compile	provided	runtime	test
compile	compile	-	runtime	-
provided	provided	provided	provided	-
runtime	runtime	-	runtime	-
test	test	-	test	-

最左边一列为直接依赖，理解为 A 依赖 B 的范围，最顶层一行为传递依赖，理解为 B 依赖 C 的范围，行与列的交叉即为 A 传递依赖 C 的范围。

举例 1：

比如 A 对 B 有 compile 依赖，B 对 C 有 runtime 依赖，那么根据表格所示 A 对 C 有 runtime 依赖。

ssm_dao 依赖 junit，scop 为 test

ssm_service 依赖 ssm_dao.

查看下图红色框内所示传递依赖范围：

直接依赖 \ 传递依赖	compile	provided	runtime	test
compile	compile	-	runtime	-
provided	provided	provided	provided	-
runtime	runtime	-	runtime	-
test	test	-	test	-

所以 ssm_dao 工程所依赖的 junit 的 jar 没有加入到 ssm_service 工程。

举例 2：如果修改 ssm_dao 工程依赖 junit 的 scop 为 compile，ssm_dao 工程所依赖的 junit 的 jar 包会加入到 ssm_service 工程中，符合上边表格所示，查看下图红色框内所示：

直接依赖 \ 传递依赖	compile	provided	runtime	test
compile	compile	-	runtime	-
provided	provided	provided	provided	-
runtime	runtime	-	runtime	-
test	test	-	test	-

2.3.6 单元测试

遇到依赖没有传递过来的问题我们通常的解决方案是在本工程中直接添加依赖：

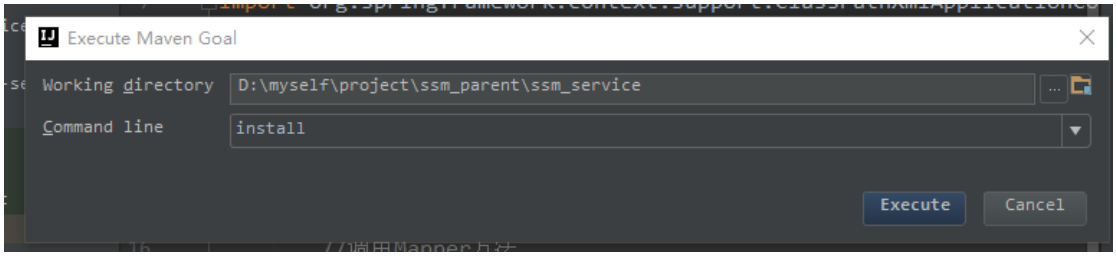
把如下依赖添加到 ssm_service 的工程中：

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

再看测试代码也不报错了

```
import cn.itcast.ssm.pojo.Items;
import cn.itcast.ssm.service.ItemsService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ItemsServiceTest {
    @Test
    public void testFindItemsById() {
        //获取spring容器
        ApplicationContext applicationContext = new ClassPathXmlApplicationContext( configLocation: "classpath*:applicationContext-*.xml");
        //获取Mapper
        ItemsService itemsService =  applicationContext.getBean(ItemsService.class);
        //调用Mapper方法
        Items items = itemsService.findById( itemId: 1);
        System.out.println(items);
    }
}
```

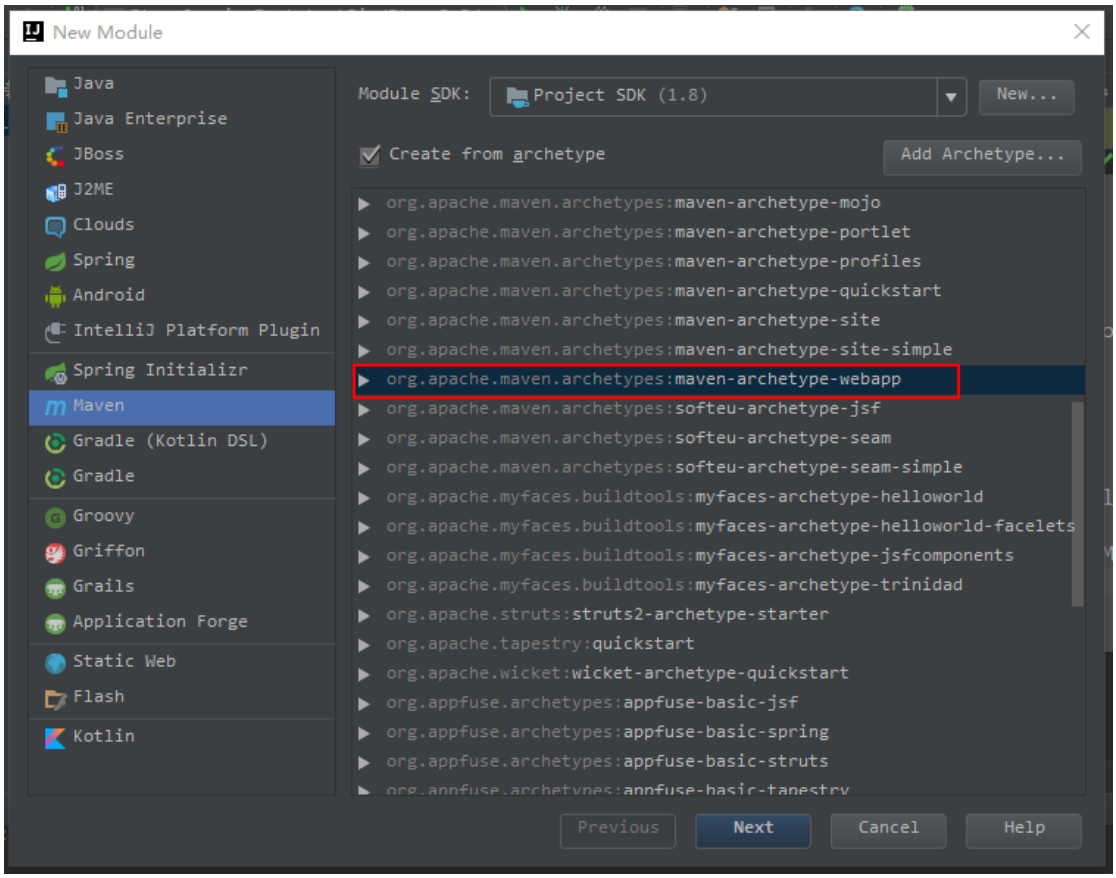
2.3.7 Install 到本地仓库



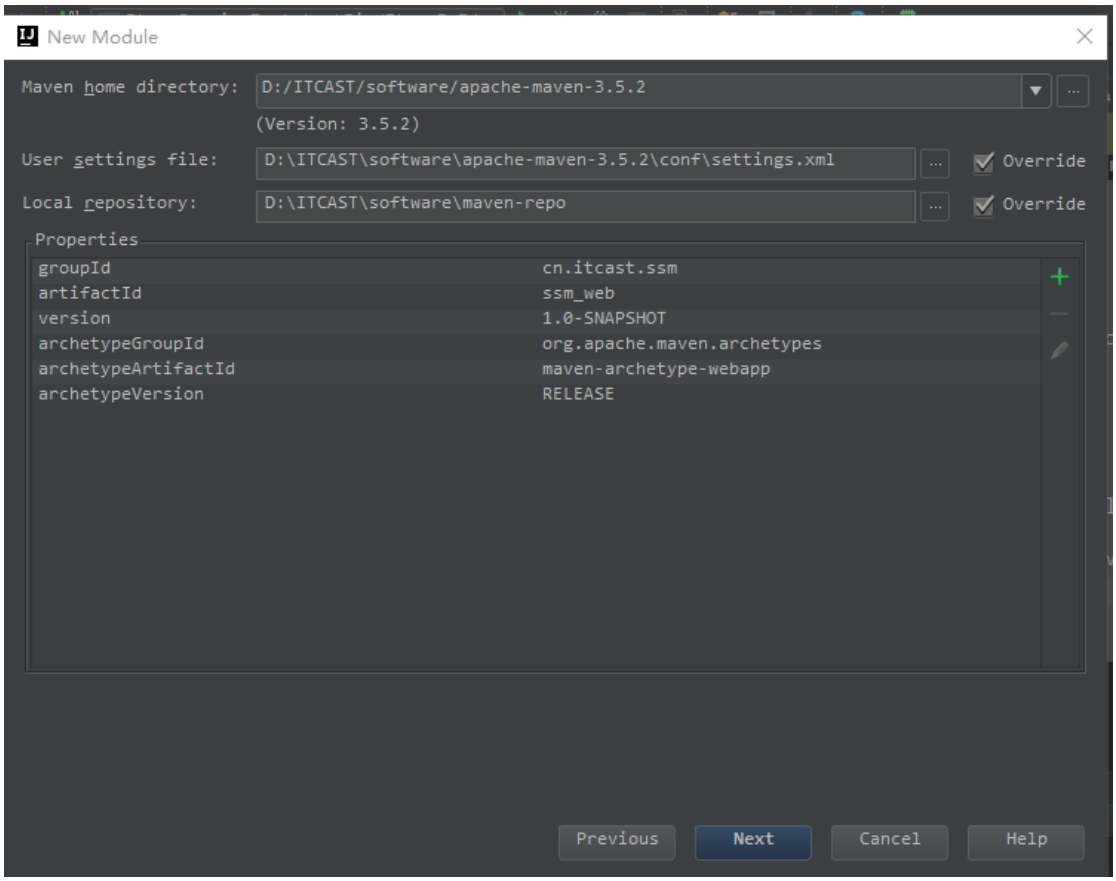
2.4 ssm_web 子模块

2.4.1 创建 web 子模块

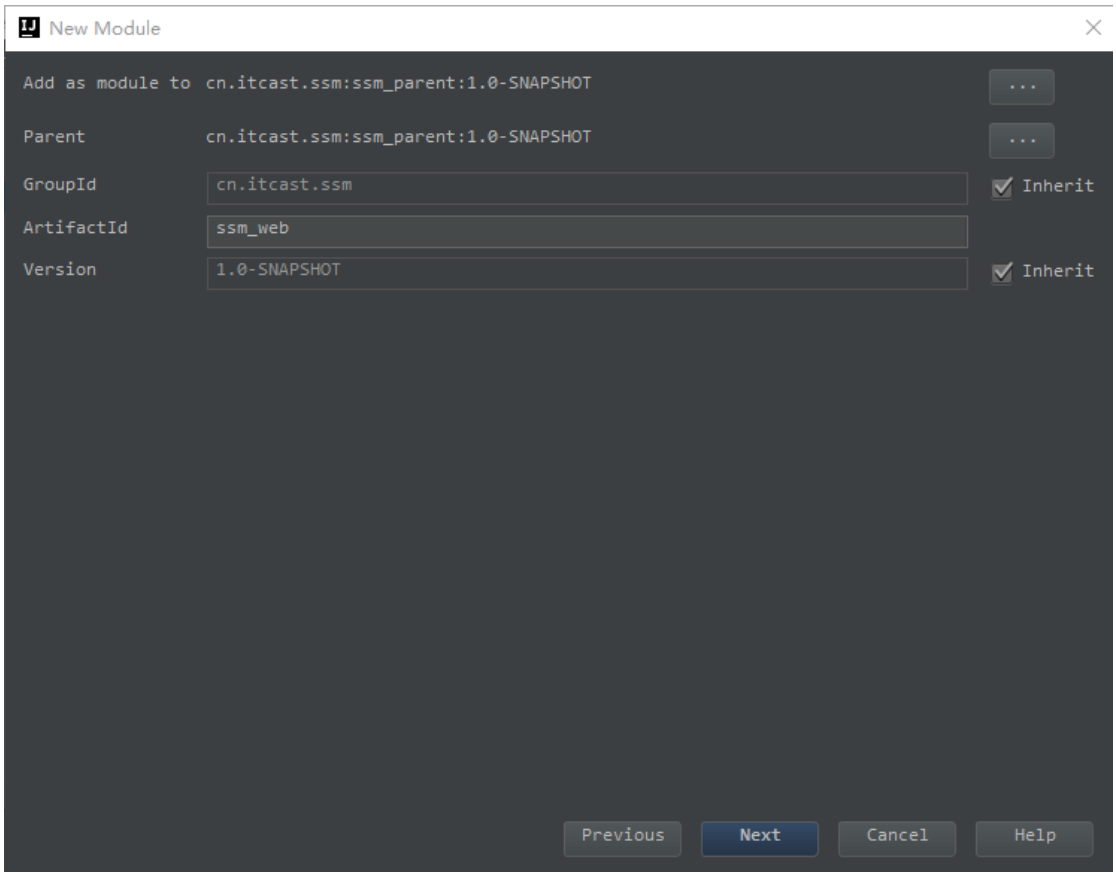
1、选择骨架创建 web 子模块



2、确认使用自己的本地仓库

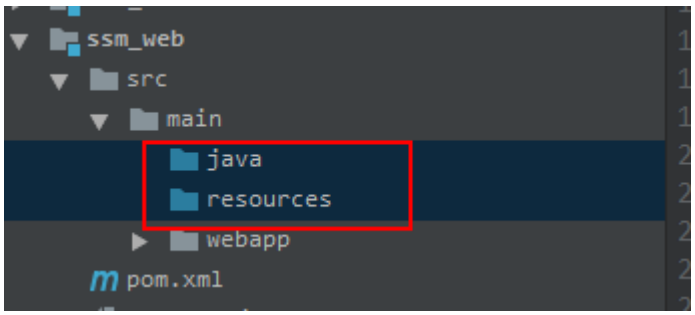

New Module
Maven home directory:
D:/ITCAST/software/apache-maven-3.5.2
(Version: 3.5.2)
User settings file:
D:\ITCAST\software\apache-maven-3.5.2\conf\settings.xml
Override
Local repository:
D:\ITCAST\software\maven-repo
Override
Properties
groupId
cn.itcast.ssm
artifactId
ssm_web
version
1.0-SNAPSHOT
archetypeGroupId
org.apache.maven.archetypes
archetypeArtifactId
maven-archetype-webapp
archetypeVersion
RELEASE
Previous
Next
Cancel
Help

3、填写模块名称



使用骨架创建 web 项目会花费些时间，请耐心等待

4、创建 java 和 resources 文件夹，转成 source root



5、添加打包方式 war

```
        xsi:schemaLocation="http://maven.apache.org/P
    <parent>
        <artifactId>ssm_parent</artifactId>
        <groupId>cn.itcast.ssm</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>ssm_web</artifactId>
    <packaging>war</packaging>

</project>
```

2.4.2 定义 pom.xml

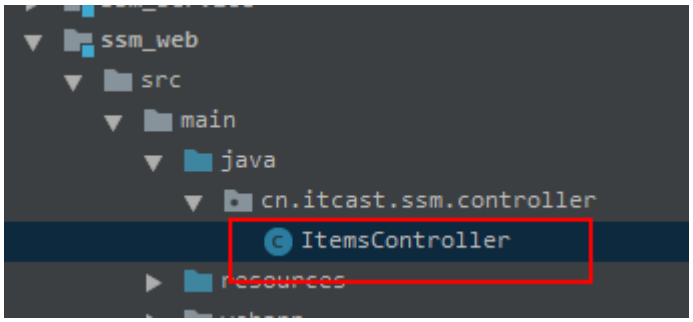
ssm_web 模块的 pom.xml 文件中需要继承父模块，ssm_web 依赖 ssm_service 模块,和 springmvc 的依赖 **注意打包方式，父层和web比较特殊，pom，war**

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
    <parent>
        <artifactId>ssm_parent</artifactId>
        <groupId>cn.itcast.ssm</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>ssm_web</artifactId>
    <packaging>war</packaging>
    <dependencies>
        <!-- 依赖service -->
        <dependency>
            <groupId>cn.itcast.ssm</groupId>
            <artifactId>ssm_service</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <!-- springMVC -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${springmvc.version}</version>
        </dependency>
    </dependencies>
```

```
</project>
```

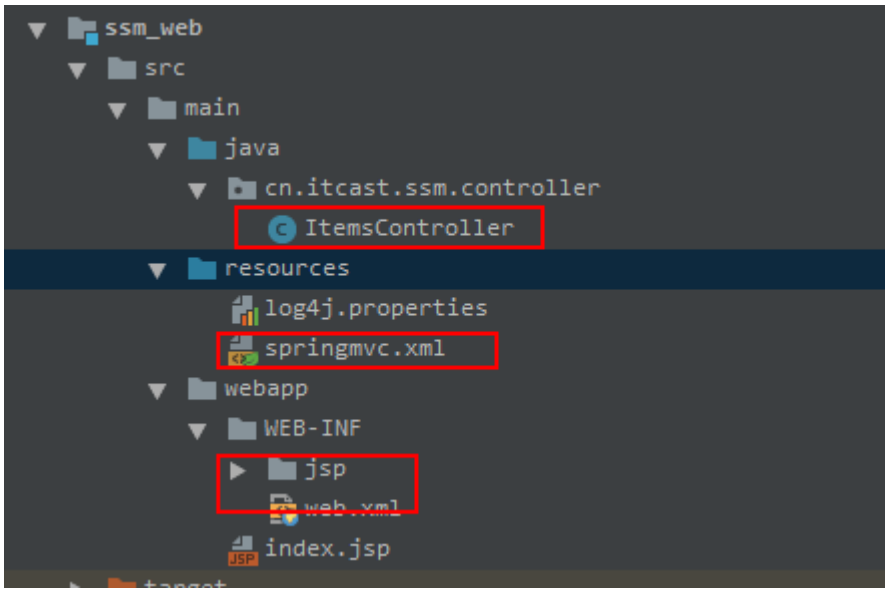
2.4.3 controller

将 ssm_web 工程中的 controller 代码拷贝到 src/main/java 中：



2.4.4 配置文件

拷贝 ssm_web 工程中如下配置文件：



2.5 运行调试

方法 1：在 ssm_web 工程的 pom.xml 中配置 tomcat 插件运行

运行 ssm_web 工程它会从本地仓库下载依赖的 jar 包，所以当 ssm_web 依赖的 jar 包内容修改了必须及时发布到本地仓库，比如：ssm_web 依赖的 ssm_service 修改了，需要及时将

ssm_service 发布到本地仓库。

方法 2：在父工程的 pom.xml 中配置 tomcat 插件运行，自动聚合并执行

推荐方法 2，如果子工程都在本地，采用方法 2 则不需要子工程修改就立即发布到本地仓库，父工程会自动聚合并使用最新代码执行。

注意：如果子工程和父工程中都配置了 tomcat 插件，运行的端口和路径以子工程为准。

方法3：使用外置tomcat

3. 分模块构建工程-依赖整合

每个模块都需要 spring 或者 junit 的 jar，况且最终 package 打完包最后生成的项目中的 jar 就是各个模块依赖的整合，所以我们可以把项目中所需的依赖都可以放到父工程中,模块中只留模块和模块之间的依赖，那父工程的 pom.xml 可以如下配置：

```
<properties>
    <spring.version>5.0.2.RELEASE</spring.version>
    <springmvc.version>5.0.2.RELEASE</springmvc.version>
    <mybatis.version>3.4.5</mybatis.version>
</properties>


<dependencyManagement>
    <dependencies>
        <!-- Mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>${mybatis.version}</version>
        </dependency>

        <!-- springMVC -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${springmvc.version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>

        <!-- spring -->
        <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>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</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-beans</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-context-support</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${spring.version}</version>
        </dependency>
```

```
    </dependencies>

</dependencyManagement>

<dependencies>
    <!-- Mybatis 和 mybatis 与 spring 的整合 -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
    </dependency>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>1.3.1</version>
    </dependency>

    <!-- MySql 驱动 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.32</version>
    </dependency>

    <!-- druid 数据库连接池 -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.0.9</version>
    </dependency>

    <!-- springMVC -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${springmvc.version}</version>
    </dependency>



    <!-- spring 相关 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
```

```
        <artifactId>spring-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-expression</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
    </dependency>


    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aspects</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
    </dependency>

    <!-- spring 相关 事务相关 -->

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
    </dependency>

    <!-- junit 测试 -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
```

```
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.0</version>
        <scope>provided</scope>
    </dependency>

    <!-- jstl -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

</dependencies>


<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.1</version>
            <configuration>
                <target>1.8</target>
                <source>1.8</source>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.2</version>
        </plugin>
    </plugins>
</build>
```

四、 maven 私服[了解]

1. 需求

正式开发，不同的项目组开发不同的工程。

ssm_dao 工程开发完毕，发布到私服。

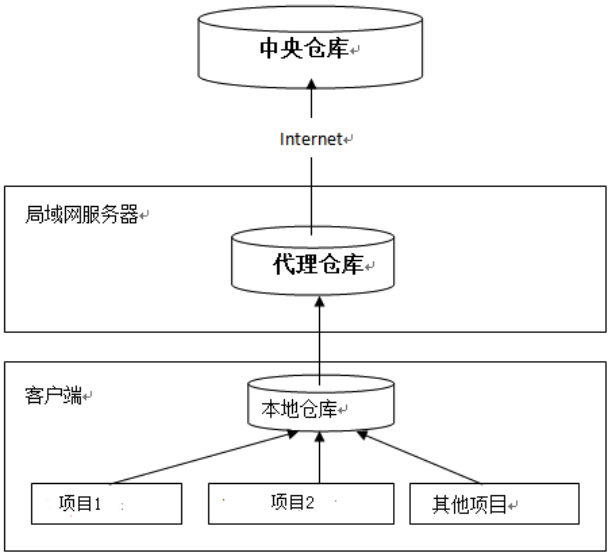
ssm_service 从私服下载 dao

2. 分析

公司在自己的局域网内搭建自己的远程仓库服务器，称为私服，私服服务器即是公司内部的 maven 远程仓库，每个员工的电脑上安装 maven 软件并且连接私服服务器，员工将自己开发的项目打成 jar 并发布到私服服务器，其它项目组从私服服务器下载所依赖的构件（jar）。

私服还充当一个代理服务器，当私服上没有 jar 包会从互联网中央仓库自动下载，如下图：

经过 私服吗



3. 搭建私服环境

3.1 下载 nexus

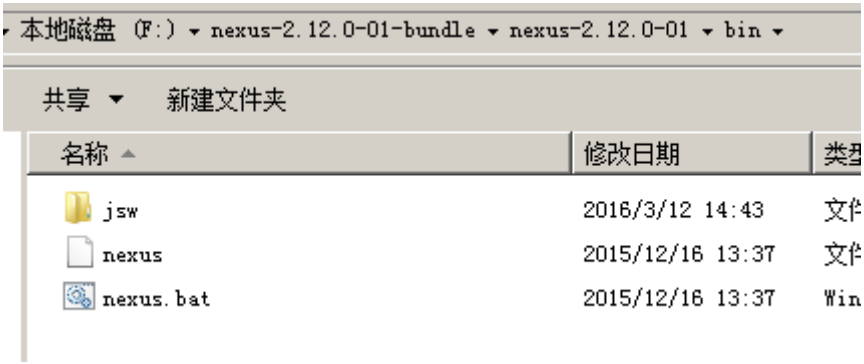
Nexus 是 Maven 仓库管理器，通过 nexus 可以搭建 maven 仓库，同时 nexus 还提供强大的仓库管理功能，构件搜索功能等。

下载 Nexus， 下载地址：http://www.sonatype.org/nexus/archived/

下载：nexus-2.12.0-01-bundle.zip

3.2 安装 nexus

解压 nexus-2.12.0-01-bundle.zip，本教程将它解压在 F 盘，进入 bin 目录：



cmd 进入 bin 目录，执行 nexus.bat install

```
F:\nexus-2.12.0-01-bundle\nexus-2.12.0-01\bin>nexus.bat install
wrapper  | nexus installed.
```

安装成功在服务中查看有 nexus 服务：

3.3 卸载 nexus

cmd 进入 nexus 的 bin 目录，执行：nexus.bat uninstall

```
F:\nexus-2.12.0-01-bundle\nexus-2.12.0-01\bin>nexus.bat uninstall
```

查看 window 服务列表 nexus 已被删除。

3.4 启动 nexus

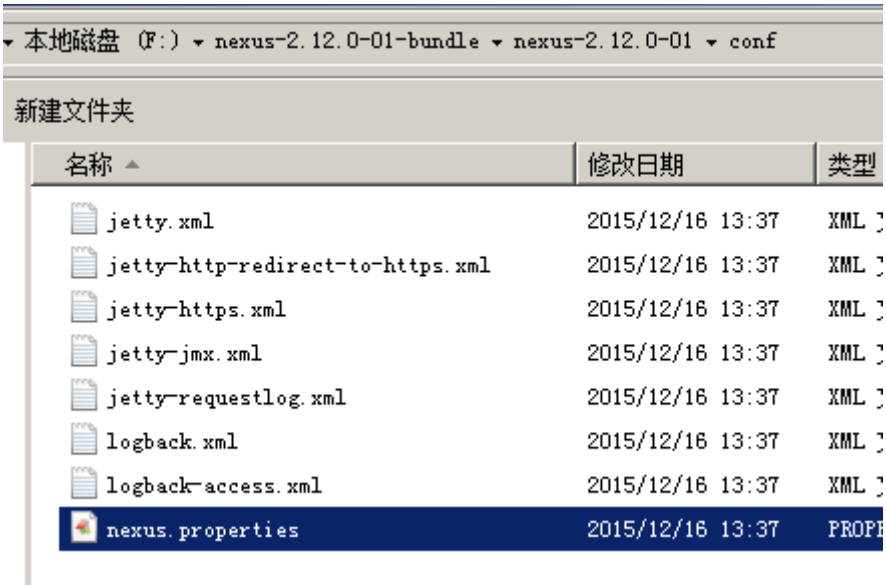
方法 1：

cmd 进入 bin 目录，执行 nexus.bat start

方法 2：

直接启动 nexus 服务

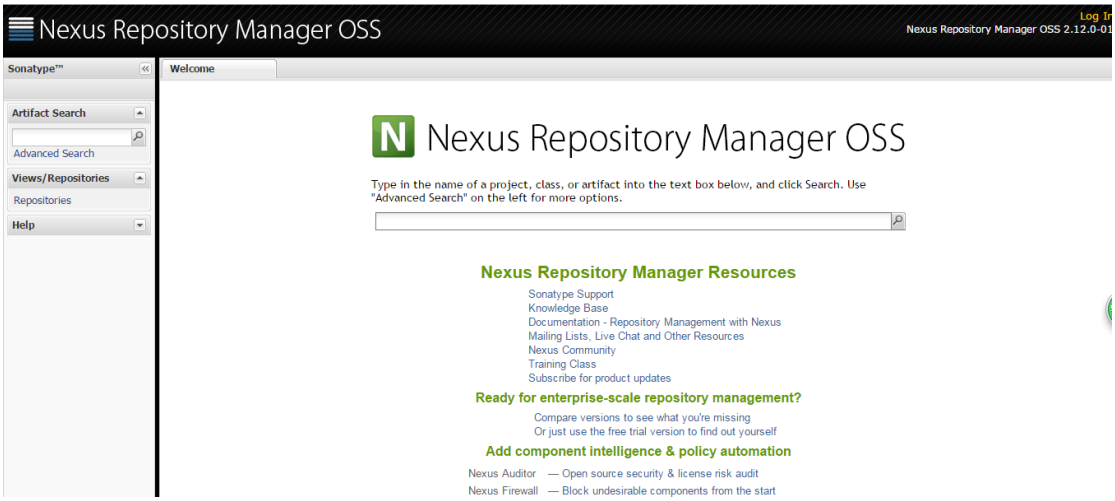
查看 nexus 的配置文件 conf/nexus.properties




```
# Jetty section
application-port=8081          # nexus 的访问端口配置
application-host=0.0.0.0       # nexus 主机监听配置(不用修改)
nexus-webapp=${bundleBasedir}/nexus      # nexus 工程目录
nexus-webapp-context-path=/nexus      # nexus 的 web 访问路径
# Nexus section
nexus-work=${bundleBasedir}/../sonatype-work/nexus     # nexus 仓库目录
runtime=${bundleBasedir}/nexus/WEB-INF    # nexus 运行程序目录
```

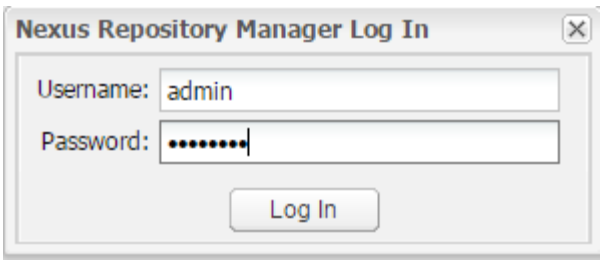
访问：

http://localhost:8081/nexus/



使用 Nexus 内置账户 admin/admin123 登陆：

点击右上角的 Log in，输入账号和密码 登陆

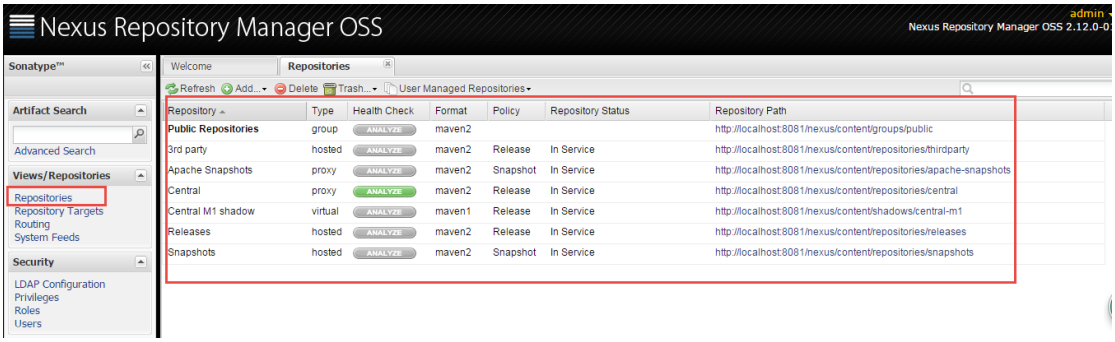


登陆成功：

3.5 仓库类型

nexus

查看 nexus 的仓库：



nexus 的仓库有 4 种类型：

Repository	Type
Public Repositories	group
3rd party	hosted
Apache Snapshots	proxy
Central	proxy
Central M1 shadow	virtual
Releases	hosted
Snapshots	hosted

1. hosted，宿主仓库，部署自己的 jar 到这个类型的仓库，包括 releases 和 snapshot 两部分，Releases 公司内部发布版本仓库、 Snapshots 公司内部测试版本仓库
2. proxy，代理仓库，用于代理远程的公共仓库，如 maven 中央仓库，用户连接私服，私服自动去中央仓库下载 jar 包或者插件。
3. group，仓库组，用来合并多个 hosted/proxy 仓库，通常我们配置自己的 maven 连接仓库组。
4. virtual(虚拟)：兼容 Maven1 版本的 jar 或者插件

nexus 仓库默认在 sonatype-work 目录中：

✓ **central：代理仓库，代理中央仓库**

✓ **apache-snapshots：代理仓库**

存储 snapshots 构件，代理地址 https://repository.apache.org/snapshots/

✓ **central-m1：**virtual 类型仓库，兼容 Maven1 版本的 jar 或者插件

✓ **releases：本地仓库，存储 releases 构件。**

✓ **snapshots：本地仓库，存储 snapshots 构件。**

✓ **thirdparty：第三方仓库**

✓ **public：仓库组**

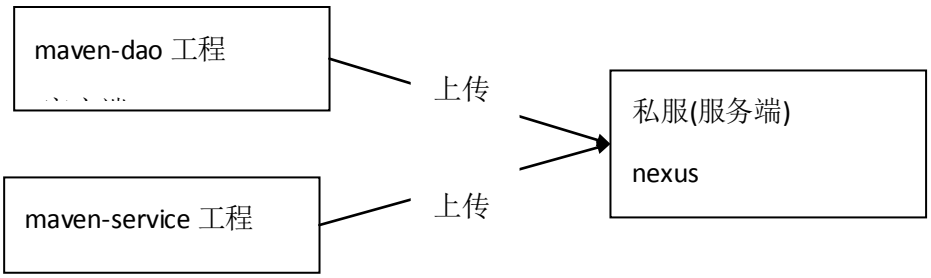
4. 将项目发布到私服

通过maven

4.1 需求

企业中多个团队协作开发通常会将一些公用的组件、开发模块等发布到私服供其它团队或模块开发人员使用。

本例子假设多团队分别开发 ssm_dao、ssm_service、ssm_web，某个团队开发完在 ssm_dao 会将 ssm_dao 发布到私服供 ssm_service 团队使用，本例子会将 ssm_dao 工程打成 jar 包发布到私服。



4.2 配置

第一步：需要在客户端即部署 ssm_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>
```

releases 连接发布版本项目仓库

snapshots 连接测试版本项目仓库

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>
```

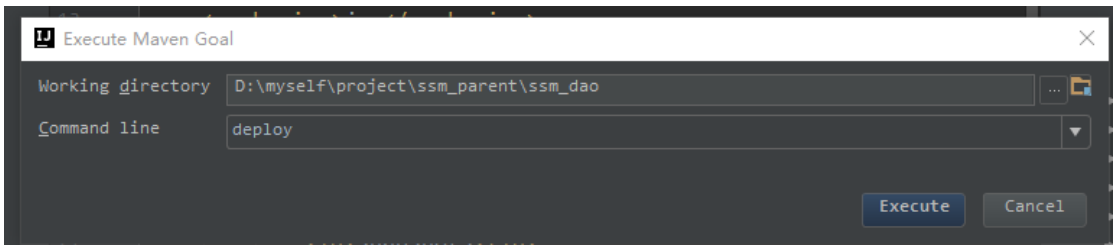
注意：pom.xml 这里<id> 和 settings.xml 配置 <id> 对应！

4.3 测试

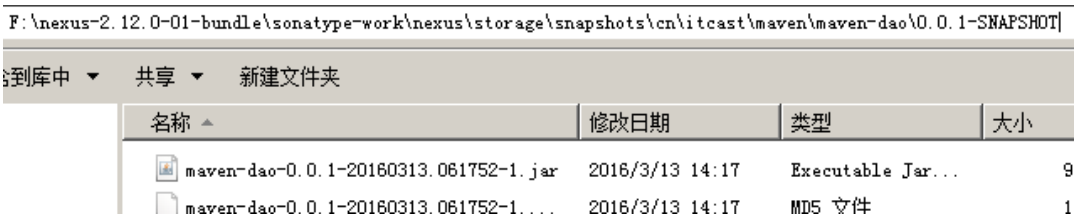
将项目 dao 工程打成 jar 包发布到私服：

1、首先启动 nexus

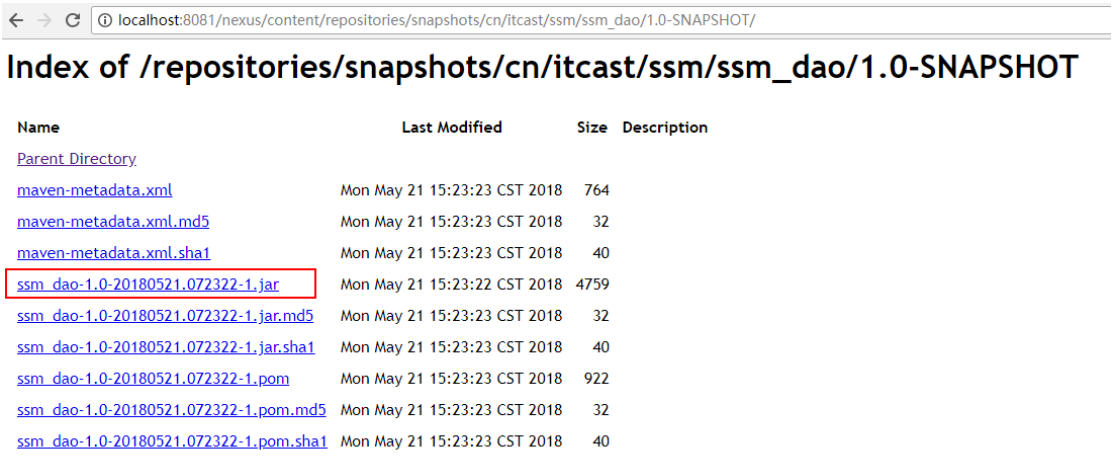
2、对 ssm_dao 工程执行 deploy 命令



根据本项目 pom.xml 中 version 定义决定发布到哪个仓库，如果 version 定义为 snapshot，执行 deploy 后查看 nexus 的 snapshot 仓库，如果 version 定义为 release 则项目将发布到 nexus 的 release 仓库，本项目将发布到 snapshot 仓库：



也可以通过 http 方式查看：



5. 从私服下载 jar 包

5.1 需求

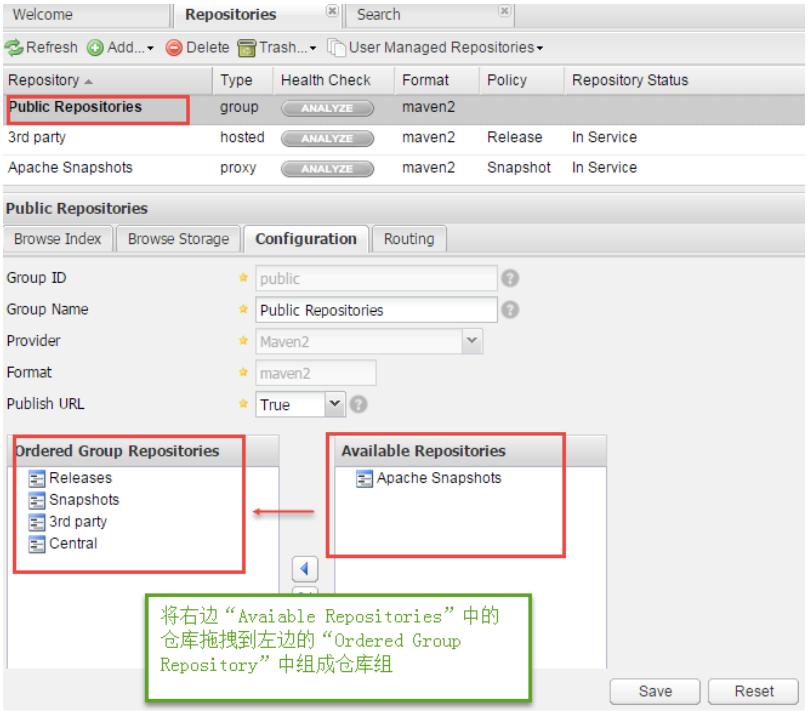
没有配置 nexus 之前，如果本地仓库没有，去中央仓库下载，通常在企业中会在局域网内部署一台私服服务器，有了私服本地项目首先去本地仓库找 jar，如果没有找到则连接私服从私服下载 jar 包，如果私服没有 jar 包私服同时作为代理服务器从中央仓库下载 jar 包，这样做的好处是一方面由私服对公司项目的依赖 jar 包统一管理，一方面提高下载速度，项目连接私服下载 jar 包的速度要比项目连接中央仓库的速度快的多。

本例子测试从私服下载 ssm_dao 工程 jar 包。

5.2 管理仓库组

nexus 中包括很多仓库，hosted 中存放的是企业自己发布的 jar 包及第三方公司的 jar 包，proxy 中存放的是中央仓库的 jar，为了方便从私服下载 jar 包可以将多个仓库组成一个仓库组，每个工程需要连接私服的仓库组下载 jar 包。

打开 nexus 配置仓库组，如下图：



上图中仓库组包括了本地仓库、代理仓库等。

5.3 在 setting.xml 中配置仓库

在客户端的 setting.xml 中配置私服的仓库，由于 setting.xml 中没有 repositories 的配置标签需要使用 profile 定义仓库。

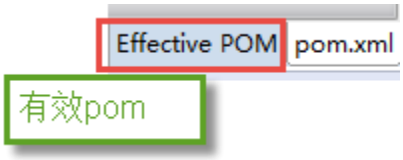
```
<profile>
    <!--profile 的 id-->
  <id>dev</id>
   <repositories>
     <repository>
        <!--仓库 id，repositories 可以配置多个仓库，保证 id 不重复-->
       <id>nexus</id>
```

```
        <!--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>
```

使用 profile 定义仓库需要激活才可生效。

```
<activeProfiles>
   <activeProfile>dev</activeProfile>
</activeProfiles>
```

配置成功后通过 eclipse 查看有效 pom，有效 pom 是 maven 软件最终使用的 pom 内容，程序员不直接编辑有效 pom，打开有效 pom



有效 pom 内容如下：

下边的 pom 内容中有两个仓库地址，maven 会先从前边的仓库的找，如果找不到 jar 包再从下边的找，从而就实现了从私服下载 jar 包。

```
<repositories>
   <repository>
     <releases>
       <enabled>true</enabled>
     </releases>
     <snapshots>
       <enabled>true</enabled>
     </snapshots>
     <id>public</id>
     <name>Public Repositories</name>
     <url>http://localhost:8081/nexus/content/groups/public/</url>
   </repository>
   <repository>
     <snapshots>
       <enabled>false</enabled>
     </snapshots>
     <id>central</id>
     <name>Central Repository</name>
     <url>https://repo.maven.apache.org/maven2</url>
   </repository>
 </repositories>
 <pluginRepositories>
   <pluginRepository>
     <id>public</id>
```

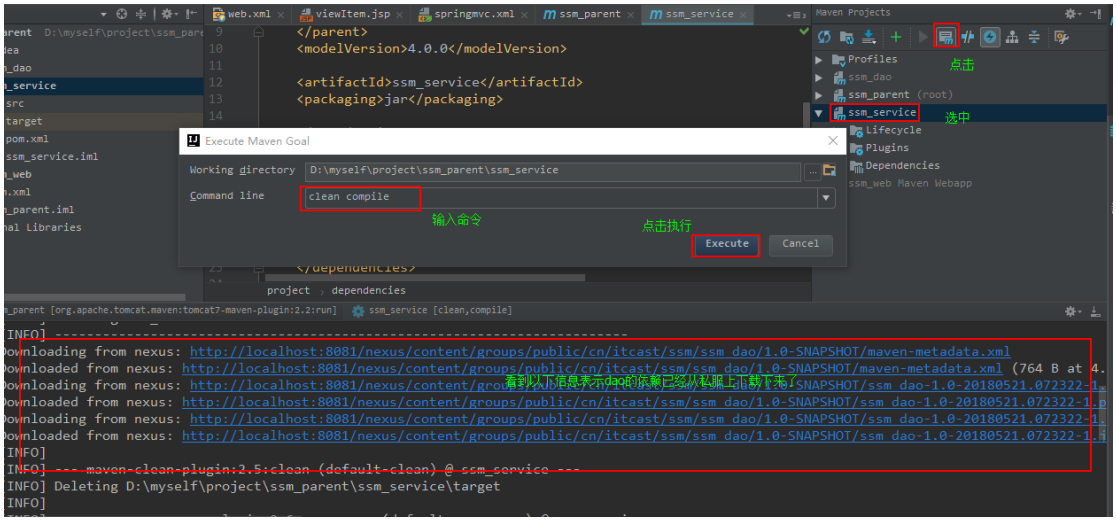
```
    <name>Public Repositories</name>
    <url>http://localhost:8081/nexus/content/groups/public/</url>
  </pluginRepository>
  <pluginRepository>
    <releases>
      <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <id>central</id>
    <name>Central Repository</name>
    <url>https://repo.maven.apache.org/maven2</url>
  </pluginRepository>
</pluginRepositories>
```

5.4 测试从私服下载 jar 包

测试 1：局域网环境或本地网络即可

在 ssm_service 工程中添加以上配置后，添加 ssm_dao 工程的依赖，删除本地仓库中 ssm_dao 工程，同时在 eclipse 中关闭 ssm_dao 工程。

观察控制台：



项目先从本地仓库找 ssm_dao，找不到从私服找，由于之前执行 deploy 将 ssm_dao 部署到私服中，所以成功从私服下载 ssm_dao 并在本地仓库保存一份。

如果此时删除私服中的 ssm_dao，执行 update project 之后是否正常？

如果将本地仓库的 ssm_dao 和私服的 ssm_dao 全部删除是否正常？

测试 2：需要互联网环境

在项目的 pom.xml 添加一个依赖，此依赖在本地仓库和私服都不存在，maven 会先从本地仓库找，本地仓库没有再从私服找，私服没有再去中央仓库下载，jar 包下载成功在私服、本地仓库分别存储一份。

五、 把第三方 jar 包放入本地仓库或私服

1. 导入本地库

随便找一个 jar 包测试，可以先 CMD 进入到 jar 包所在位置，运行

mvn install:install-file -DgroupId=com.alibaba -DartifactId=fastjson -Dversion=1.1.37

-Dfile= fastjson-1.1.37.jar -Dpackaging=jar

```
D:\class314>mvn install:install-file -DgroupId=com.alibaba -DartifactId=fastjson -Dversion=1.1.37 -Dfile=fastjson-1.1.37
.jar -Dpackaging=jar
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
[INFO] Installing D:\class314\fastjson-1.1.37.jar to D:\ITCAST\software\maven-repo\com\alibaba\fastjson\1.1.37\fastjson-
1.1.37.jar
[INFO] Installing C:\Users\sy1\AppData\Local\Temp\mvninstall1708850793970657641 2.pom to D:\ITCAST\software\maven-repo\com
\alibaba\fastjson\1.1.37\fastjson-1.1.37.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.846 s
[INFO] Finished at: 2018-03-09T22:13:29+08:00
[INFO] Final Memory: 6M/15M
```

2. 导入私服

需要在 maven 软件的核心配置文件 settings.xml 中配置第三方仓库的 server 信息

```
<server>
  <id>thirdparty</id>
  <username>admin</username>
  <password>admin123</password>
</server>
```

才能执行一下命令

```
mvn deploy:deploy-file -DgroupId=com.alibaba -DartifactId=fastjson -Dversion=1.1.37
-Dpackaging=jar -Dfile=fastjson-1.1.37.jar
-Durl=http://localhost:8081/nexus/content/repositories/thirdparty/
-DrepositoryId=thirdparty
```

```
D:\class314>mvn deploy:deploy-file -DgroupId=com.alibaba -DartifactId=fastjson -Dversion=1.1.37 -Dpackaging=jar -Dfile=f
astjson-1.1.37.jar -Durl=http://localhost:8081/nexus/content/repositories/thirdparty/ -DrepositoryId=thirdparty
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-deploy-plugin:2.7:deploy-file (default-cli) @ standalone-pom ---
Uploading to thirdparty: http://localhost:8081/nexus/content/repositories/thirdparty/com/alibaba/fastjson/1.1.37/fastjso
n-1.1.37.jar
Uploaded to thirdparty: http://localhost:8081/nexus/content/repositories/thirdparty/com/alibaba/fastjson/1.1.37/fastjson
-1.1.37.jar (357 kB at 1.1 MB/s)
Uploading to thirdparty: http://localhost:8081/nexus/content/repositories/thirdparty/com/alibaba/fastjson/1.1.37/fastjso
n-1.1.37.pom
Uploaded to thirdparty: http://localhost:8081/nexus/content/repositories/thirdparty/com/alibaba/fastjson/1.1.37/fastjson
-1.1.37.pom (393 B at 3.1 kB/s)
Downloading from thirdparty: http://localhost:8081/nexus/content/repositories/thirdparty/com/alibaba/fastjson/maven-meta
data.xml
Uploading to thirdparty: http://localhost:8081/nexus/content/repositories/thirdparty/com/alibaba/fastjson/maven-metadata
.xml
Uploaded to thirdparty: http://localhost:8081/nexus/content/repositories/thirdparty/com/alibaba/fastjson/maven-metadata.
xml (301 B at 2.1 kB/s)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.476 s
[INFO] Finished at: 2018-03-09T22:20:59+08:00
[INFO] Final Memory: 7M/19M
[INFO] ------------------------------------------------------------------------

D:\class314>
```

localhost:8081/nexus/content/repositories/thirdparty/com/alibaba/fastjson/1.1.37/

Index of /repositories/thirdparty/com/alibaba/fastjson/1.1.37

Name	Last Modified	Size	Description
Parent Directory			
fastjson-1.1.37.jar	Fri Mar 09 22:20:58 CST 2018	356654	
fastjson-1.1.37.jar.md5	Fri Mar 09 22:20:59 CST 2018	32	
fastjson-1.1.37.jar.sha1	Fri Mar 09 22:20:58 CST 2018	40	
fastjson-1.1.37.pom	Fri Mar 09 22:20:59 CST 2018	393	
fastjson-1.1.37.pom.md5	Fri Mar 09 22:20:59 CST 2018	32	
fastjson-1.1.37.pom.sha1	Fri Mar 09 22:20:59 CST 2018	40	

3. 参数说明

DgroupId 和 DartifactId 构成了该 jar 包在 pom.xml 的坐标，项目就是依靠这两个属性定位。自己起名字也行。

Dfile 表示需要上传的 jar 包的绝对路径。

Durl 私服上仓库的位置，打开 nexus——>repositories 菜单，可以看到该路径。

DrepositoryId 服务器的表示 id，在 nexus 的 configuration 可以看到。

Dversion 表示版本信息，

关于 jar 包准确的版本：

包的名字上一般会带版本号，如果没有那可以解压该包，会发现一个叫 MANIFEST.MF 的文件，

这个文件就有描述该包的版本信息。

比如 Specification-Version: 2.2 可以知道该包的版本了。

上传成功后，在 nexus 界面点击 3rd party 仓库可以看到这包。

工程和模块的区别：
工程不等于完整的项目，模块也不等于完整的项目，一个完整的项目看的是代码，代码完整，就可以说这是一个完整的项目
和此项目是工程和模块没有关系。

工程天生只能使用自己内部资源，工程天生是独立的。后天可以和其他工程或模块建立关联关系。
模块天生不是独立的，模块天生是属于父工程的，模块一旦创建，所有父工程的资源都可以使用。

父子工程直接，子模块天生集成父工程，可以使用父工程所有资源。
子模块之间天生是没有任何关系的。

父子工程直接不用建立关系，继承关系是先天的，不需要手动建立。

平级直接的引用叫依赖，依赖不是先天的，依赖是需要后天建立的。

maven内置属性详细说明

2017年02月03日 18:09:33 bitcarmanlee 阅读数 3308 更多

Maven共有6类属性：

1.内置属性(Maven预定义,用户可以直接使用)

${basedir}表示项目根目录,即包含pom.xml文件的目录;
${version}表示项目版本;
${project.basedir}同${basedir};
${project.baseUri}表示项目文件地址;
${maven.build.timestamp}表示项目构件开始时间;
${maven.build.timestamp.format}表示属性${maven.build.timestamp}的展示格式,默认值为yyyyMMdd-HHmm,可自定义其格式,其类型可参考Java.text.SimpleDateFormat。用法如下：

```
<properties>
<maven.build.timestamp.format>yyyy-MM-dd HH:mm:ss</maven.build.timestamp.format>
</properties>
```

2.POM属性(使用pom属性可以引用到pom.xml文件对应元素的值)

${project.build.directory}表示主源码路径，缺省为target;
${project.build.outputDirectory} 构建过程输出目录，缺省为target/classes
${project.build.sourceEncoding}表示主源码的编码格式;
${project.build.sourceDirectory}表示主源码路径;
${project.build.finalName}表示输出文件名称，缺省为${project.artifactId}-${project.version};
${project.packaging} 打包类型，缺省为jar;
${project.version}表示项目版本,与${version}相同;

3.自定义属性

在pom.xml文件的< properties >properties>标签下定义的Maven属性

```
<project>
  <properties>
    <my.pro>abc</my.pro>
  </properties>
</project>
```

在其他地方使用${my.pro}使用该属性值。

4.settings.xml文件属性

与pom属性同理,用户使用以settings.开头的属性引用settings.xml文件中的XML元素值
${settings.localRepository}表示本地仓库的地址;

5.Java系统属性

所有的Java系统属性都可以使用Maven属性引用
使用mvn help:system命令可查看所有的Java系统属性;
System.getProperties()可得到所有的Java属性;
${user.home}表示用户目录;

6.环境变量属性

所有的环境变量都可以用以env.开头的Maven属性引用
使用mvn help:system命令可查看所有环境变量;
${env.JAVA_HOME}表示JAVA_HOME环境变量的值;