

day32-Maven&Lombok

学习目标

- ☐ 能够了解Maven的作用
- ☐ 能够理解Maven仓库的作用
- ☐ 能够理解Maven的坐标概念
- ☐ 能够掌握Maven的安装
- ☐ 能够掌握IDEA配置本地Maven
- ☐ 能够使用IDEA创建javase的Maven工程
- ☐ 能够使用IDEA创建javaweb的Maven工程
- ☐ 能够理解依赖范围
- ☐ 了解搭建私服的使用
- ☐ 能够掌握Lombok的使用

第一章-Maven相关的概念

知识点-Maven介绍

1.目标

- 能够了解Maven的作用

2.路径

- 什么是Maven
- Maven的作用
- Maven的好处

3.讲解

3.1什么是Maven

Maven是项目进行模型抽象，充分运用的面向对象的思想，Maven可以通过一小段描述信息来管理项目的构建，报告和文档的软件项目管理工具。Maven 除了以程序构建能力为特色之外，还提供高级项目管理工具。由于 Maven 的缺省构建规则有较高的可重用性，所以常常用两三行 Maven 构建脚本就可以构建简单的项目。

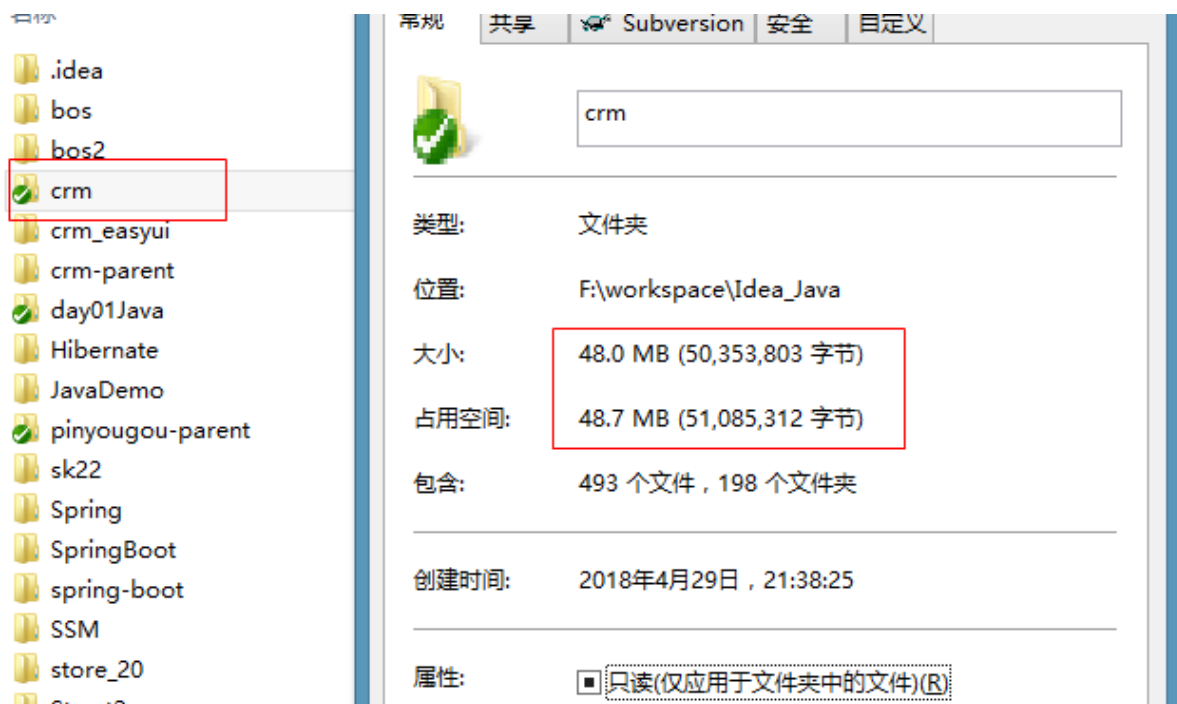
简单来说: ==Maven是由Apache开发的一个工具。==用来管理java项目(依赖(jar)管理, 项目构建, 分模块开发)。

3.2 Maven的作用

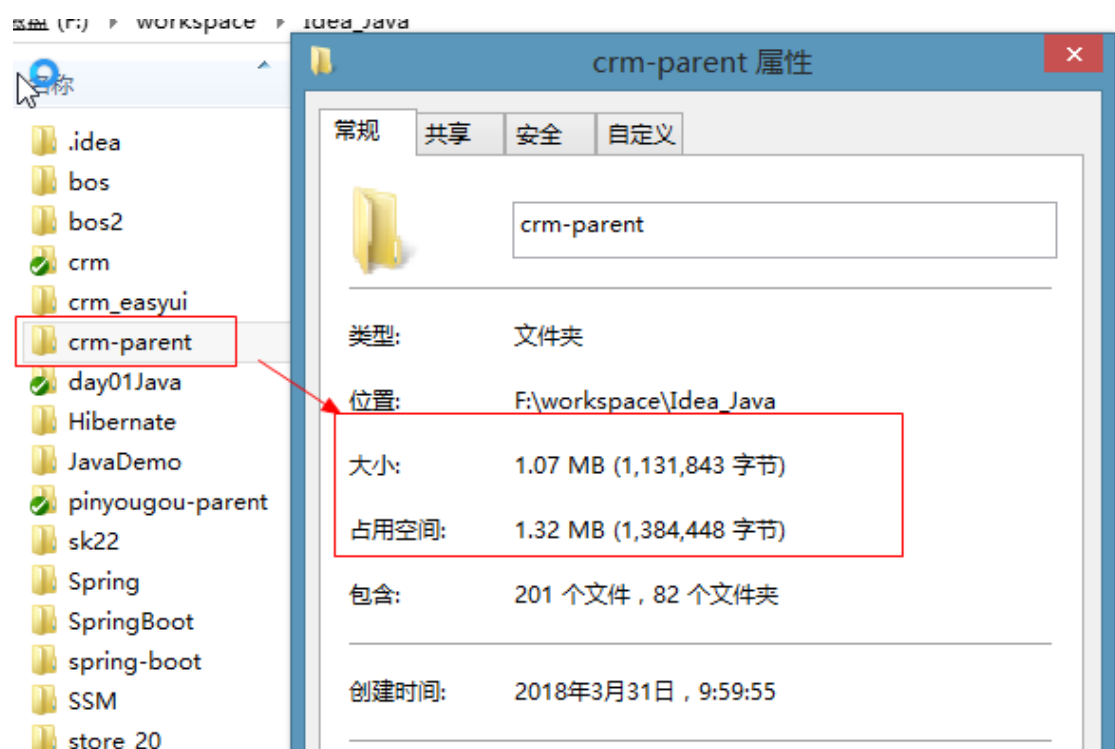
- 依赖管理: maven对项目的第三方构件 (jar包) 进行统一管理。向工程中加入jar包不要手工从其它地方拷贝, 通过**maven定义jar包的坐标**, 自动从**maven仓库**中去下载到工程中。
- 项目构建: maven提供一套对项目生命周期管理的标准, 开发人员、和测试人员统一使用maven进行项目构建。项目生命周期管理: **编译、测试、打包、部署、运行**。
- maven对工程分模块构建, 提高开发效率。(后面Maven高级会涉及)

3.3 Maven的好处

- 使用普通方式构建项目



- 使用Maven构建项目



4.小结

1. Maven是一个项目构建工具，Apache提供，可以进行项目jar包管理。
2. 作用
 1. 依赖管理：管理项目中使用的jar包
 2. 项目构建：编译、测试、打包、部署、运行
 3. 分模块开发：提高开发效率

知识点-Maven仓库和坐标

1.目标

- 能够理解Maven仓库的作用

2.路径

1. Maven的仓库
2. Maven的坐标

3.讲解

3.1Maven的仓库【存放jar包】

仓库名称	作用
本地仓库	本地仓库：本地电脑存放jar包的位置 相当于缓存，工程第一次会从远程仓库（互联网）去下载jar包，将jar包存在本地仓库（在程序员的电脑上）。第二次不需要从远程仓库去下载。先从本地仓库找，如果找不到才会去远程仓库找。
中央仓库	官方提供的jar包存放的位置 仓库中jar由专业团队（maven团队）统一维护。中央仓库的地址： http://repo1.maven.org/maven2/
远程仓库	在公司内部架设一台私服，其它公司架设一台仓库，一般不对外公开。（阿里云 私服 对外公开）

3.2 Maven的坐标

Maven的一个核心的作用就是管理项目的依赖，引入我们所需的各种jar包等。为了能自动化的解析任何一个Java构件，Maven必须将这些jar包或者其他资源进行**唯一标识**，这是管理项目的依赖的基础，也就是我们要说的坐标。**包括我们自己开发的项目，也是要通过坐标进行唯一标识的**，这样才能才其它项目中进行依赖引用。坐标的定义元素如下：

- groupId:项目组织唯一的标识符，实际对应JAVA的包的结构 (一般写公司的组织名称 eg:com.itheima,com.alibaba)
- artifactId: 项目的名称
- version: 定义项目的当前版本

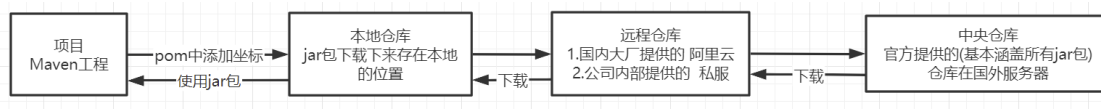
例如：要引入druid，只需要在pom.xml配置文件中配置引入druid的坐标即可：

```
<!--druid连接池-->
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.0.9</version>
</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.39</version>
</dependency>
```

4.小结

1. 仓库(本地仓库,中央仓库,远程仓库(私服))：存储jar包的地方

- 项目获取jar包，会先从本地仓库查找
 - 本地仓库有，直接获取使用
 - 本地仓库没有，则去中央仓库获取，获取之后保存一份在本地仓库，下次需要就可以直接使用



2. 通过坐标(GAV)从仓库里面找到对应的jar使用

坐标：是jar包的唯一标识 通过坐标找到对应的jar包使用

查找jar包坐标：<https://mvnrepository.com/>

```
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.0.9</version>
</dependency>
```

第二章-Maven的安装【掌握】

知识点-Maven的安装

1.目标

- 能够掌握Maven的安装

2.路径

1. 下载Maven
2. 安装Maven
3. Maven目录介绍
4. 配置环境变量
5. 配置本地仓库
6. 测试Maven是否安装成功

3.讲解

3.1下载Maven

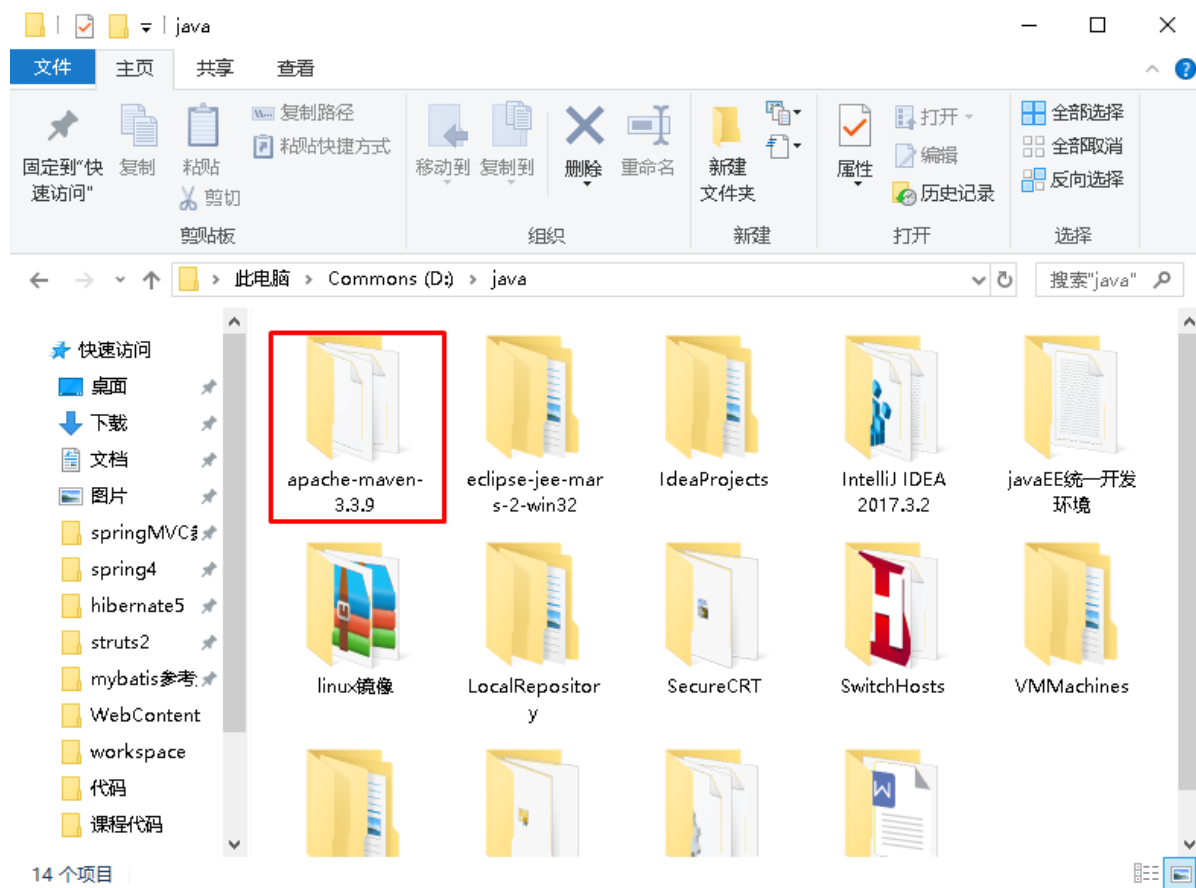


apache-maven-3.3.9-bin.zip

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

3.2 安装Maven

将Maven压缩包解压，即安装完毕



3.3 Maven目录介绍

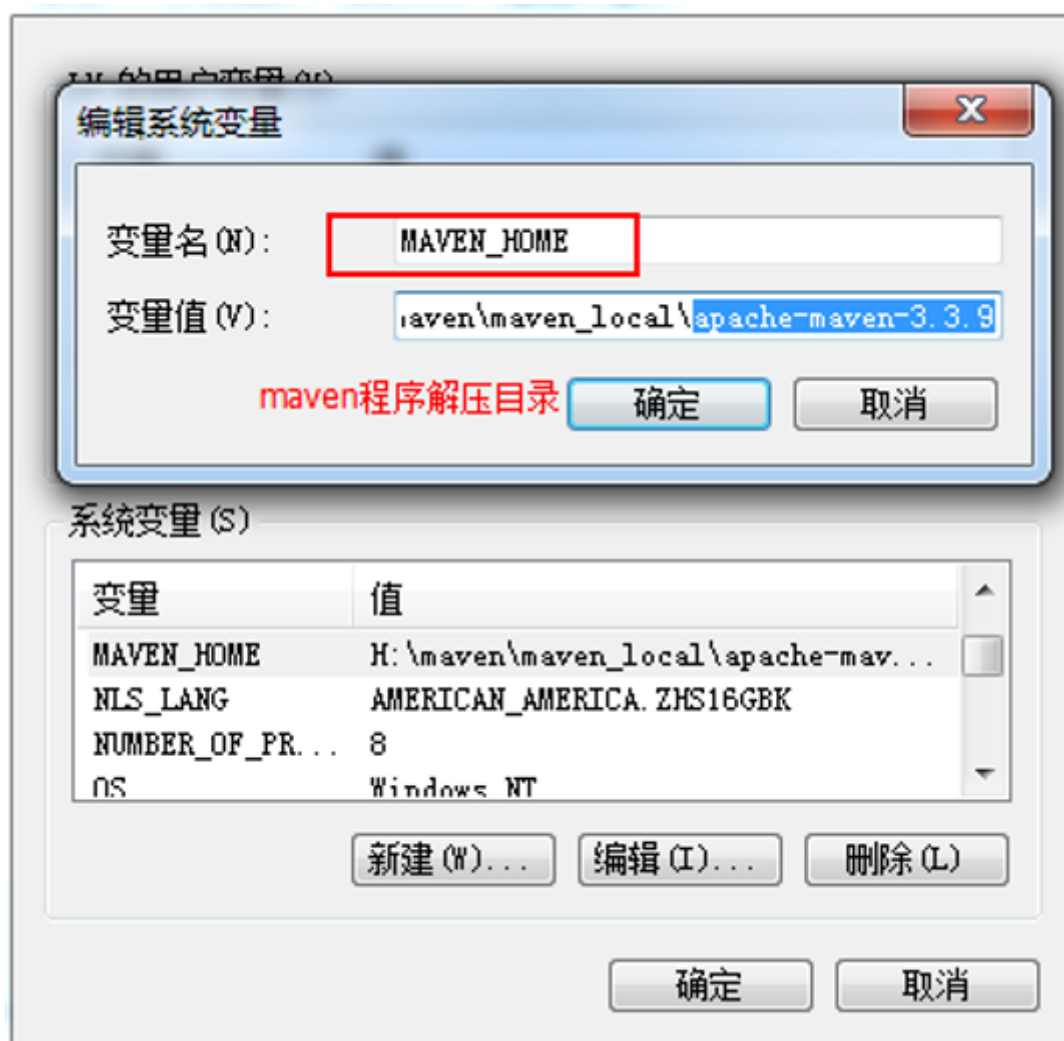
20170721 > apache-maven-3.3.9 >		搜索
名称		修改日期
bin	命令	17/11/21
boot	第三方类加载框架	15/11/10
conf		17/11/21
lib	配置	17/11/21
LICENSE	maven自身jar包	15/11/10
NOTICE		15/11/10
README.txt		15/11/10

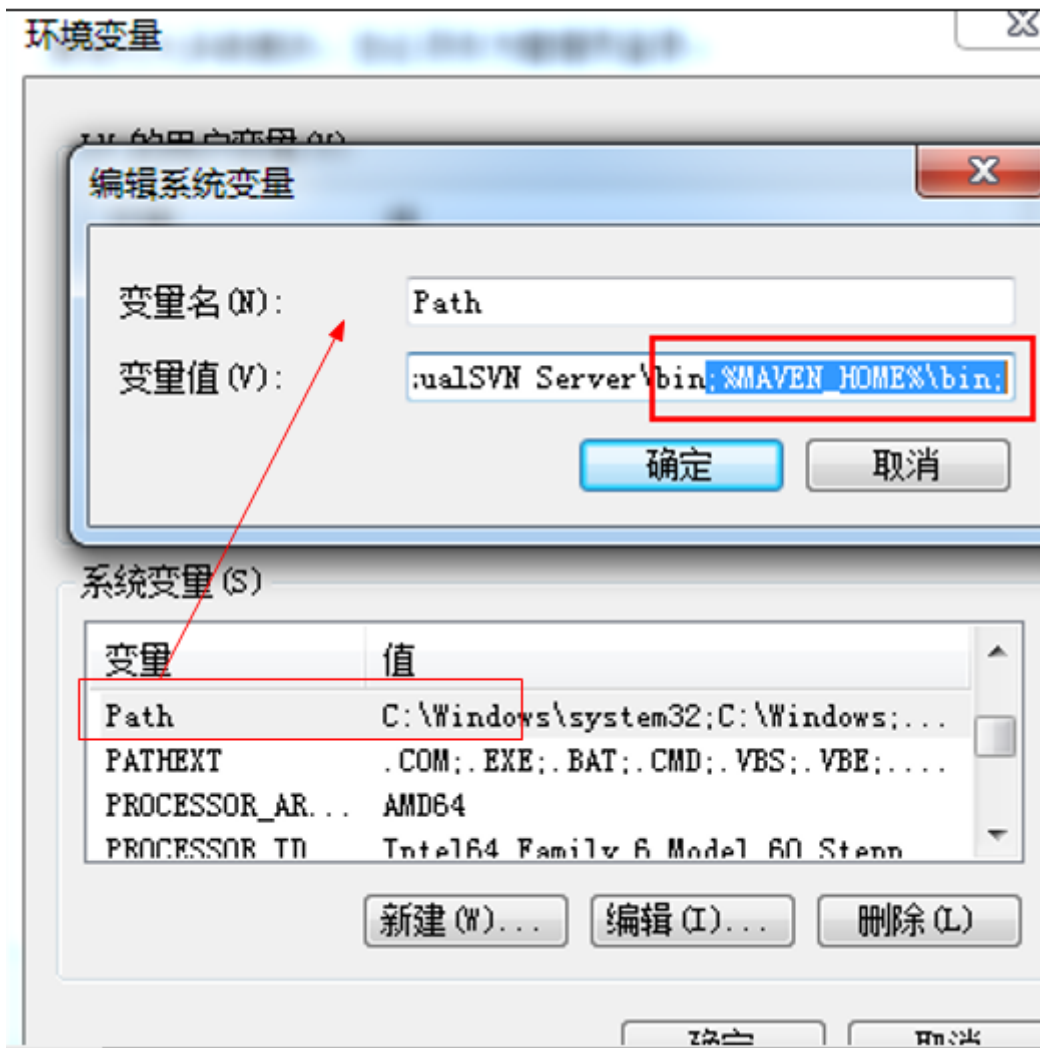
3.4 配置环境变量

- 进入环境变量



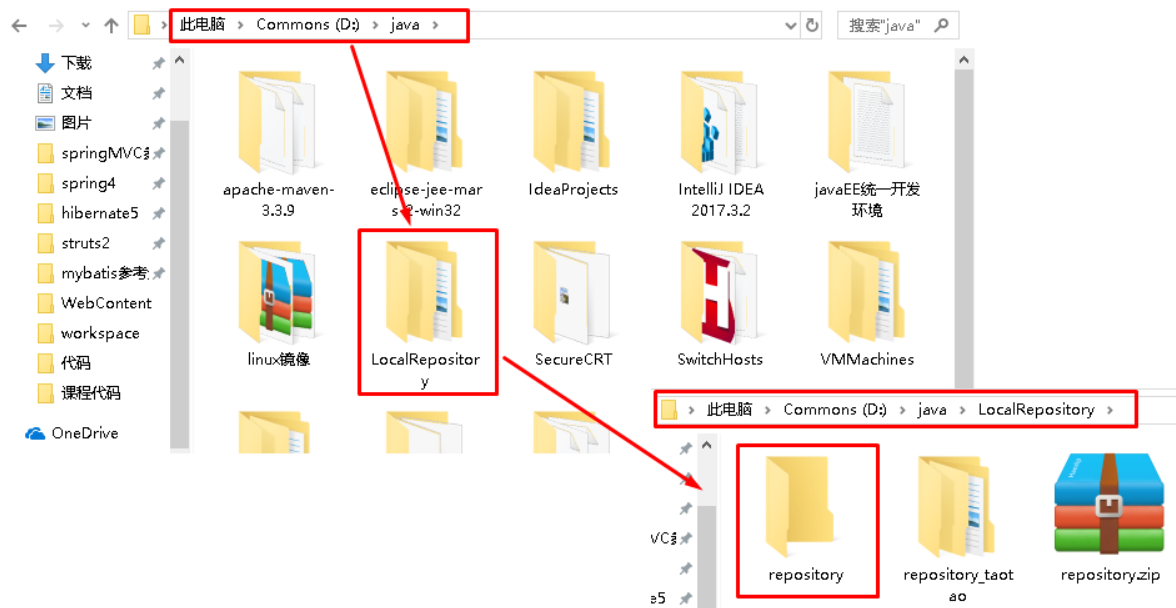
- 配置MAVEN_HOME和Path





3.5 配置本地仓库

3.5.1 将软件文件夹中的Repository解压



3.5.2 配置本地仓库

在maven的安装目录中conf/ settings.xml文件，在这里配置本地仓库


```

53 <localRepository>/path/to/local/repo</localRepository>
54 -->
55 <localRepository>E:/source/04_Maven/repository_pinyougou</localRepository>
56

```

本地仓库的路径

- 示例代码

```

<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>E:/source/04_Maven/repository_pinyougou</localRepository>

```

3.6 测试Maven安装成功

打开cmd本地控制台，输入mvn -version

命令提示符

```

Microsoft Windows [版本 10.0.16299.248]
(c) 2017 Microsoft Corporation。保留所有权利。

C:\Users\muzimoo>mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T00:41:47+08:00)
Maven home: D:\java\apache-maven-3.3.9\bin\..
Java version: 1.7.0_72, vendor: Oracle Corporation
Java home: C:\Program Files (x86)\Java\jdk1.7.0_72\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 8.1", version: "6.3", arch: "x86", family: "windows"

C:\Users\muzimoo>

```

4.小结

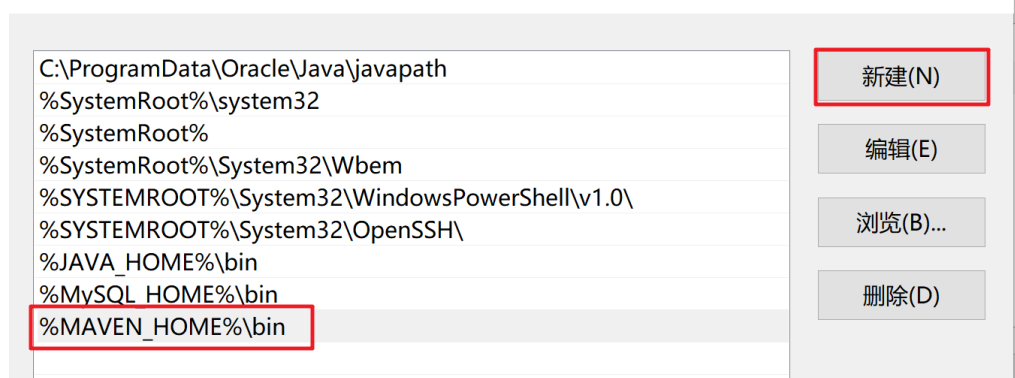
1. 注意事项

- Maven安装包 和 仓库 都需要解压到一个没有中文和空格的目录下(建议解压到不同的目录)
- 环境变量配置
 - MAVEN_HOME 配置到Maven的解压目录 eg:D:\ProgramFiles\apache-maven-3.5.3

系统变量(S)

变量	值
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_161
MAVEN_HOME	D:\ProgramFiles\apache-maven-3.8.1
MYSQL_HOME	D:\ProgramFiles\mysql-5.7.29-winx64
NUMBER_OF_PROCESSORS	16
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\...

- Path 配置到bin目录 eg:%MAVEN_HOME%\bin



- 在 apache-maven-3.5.3\conf\settings.xml 配置本地仓库

```
<localRepository>D:/Maven/repository</localRepository>
```

知识点-IDEA集成Maven

1.目标

- 能够掌握IDEA配置本地Maven

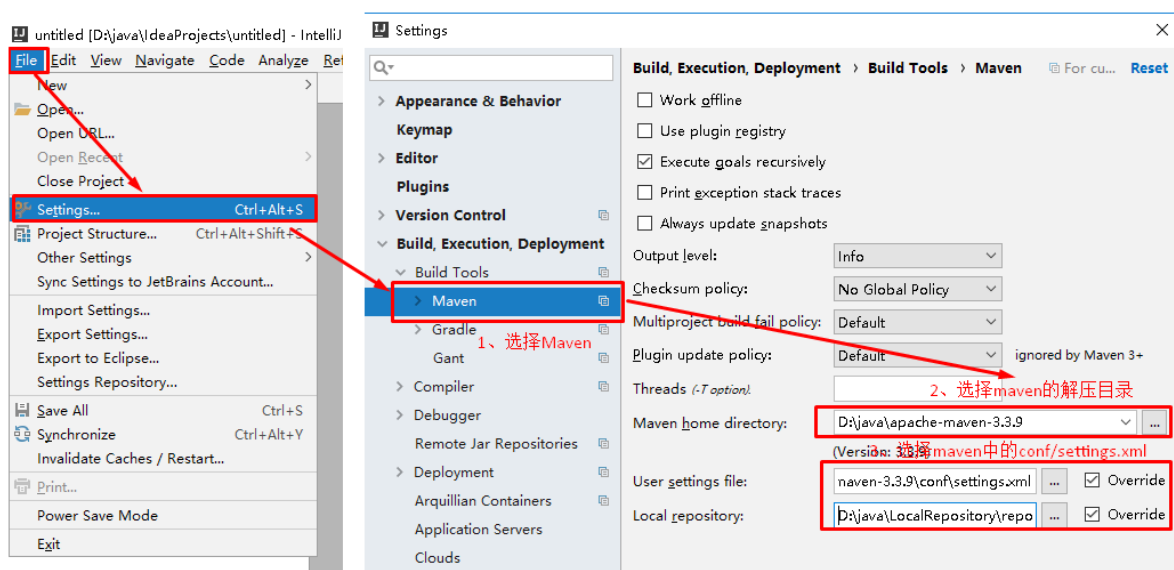
2.路径

- 在IDEA配置Maven
- 配置默认的Maven环境

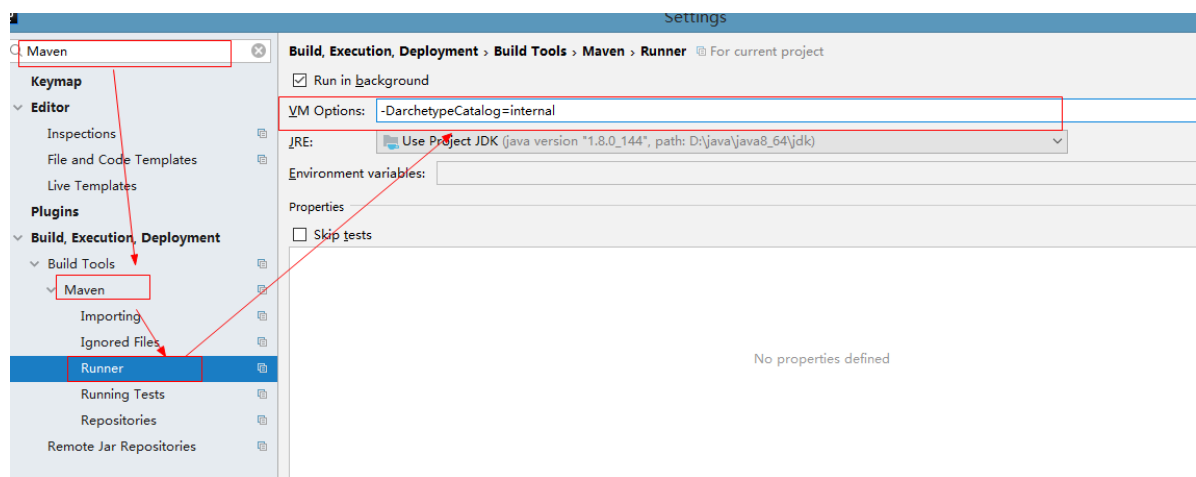
3.讲解

3.1配置Maven

- 配置Maven



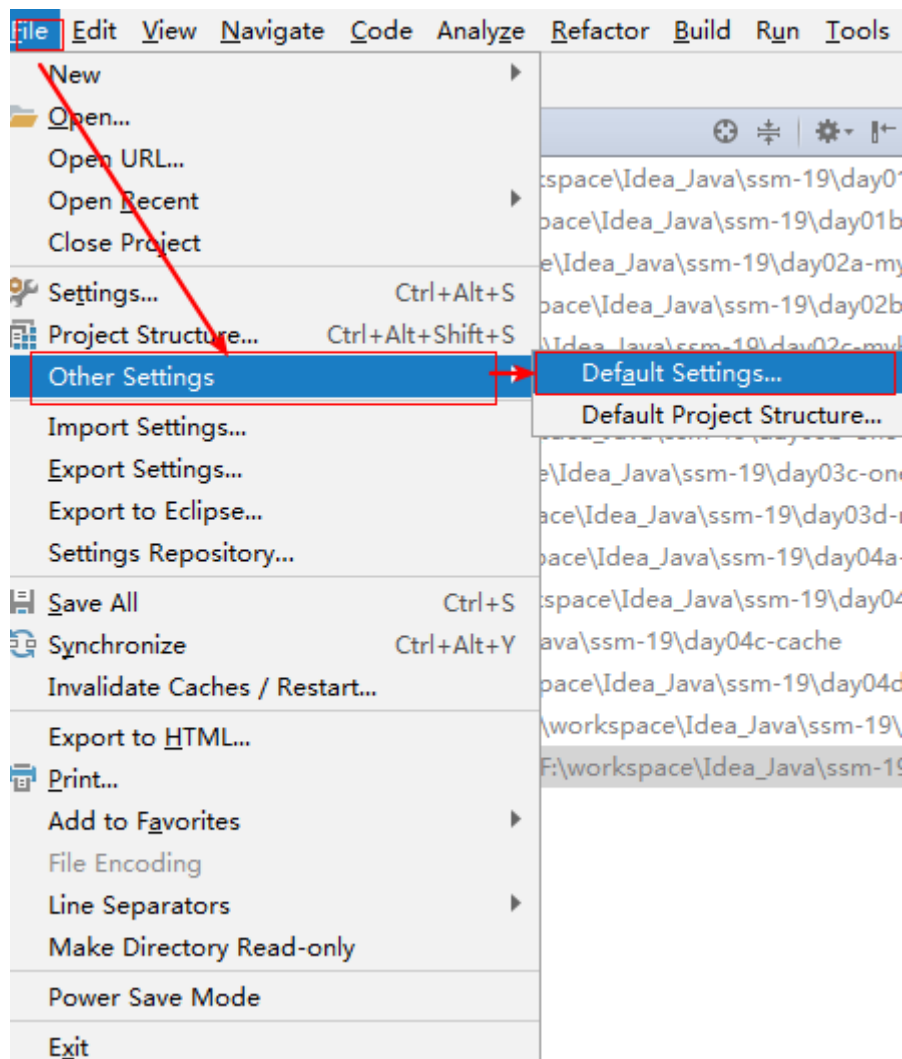
- 配置参数(创建工程不需要联网,解决创建慢的问题) -DarchetypeCatalog=internal



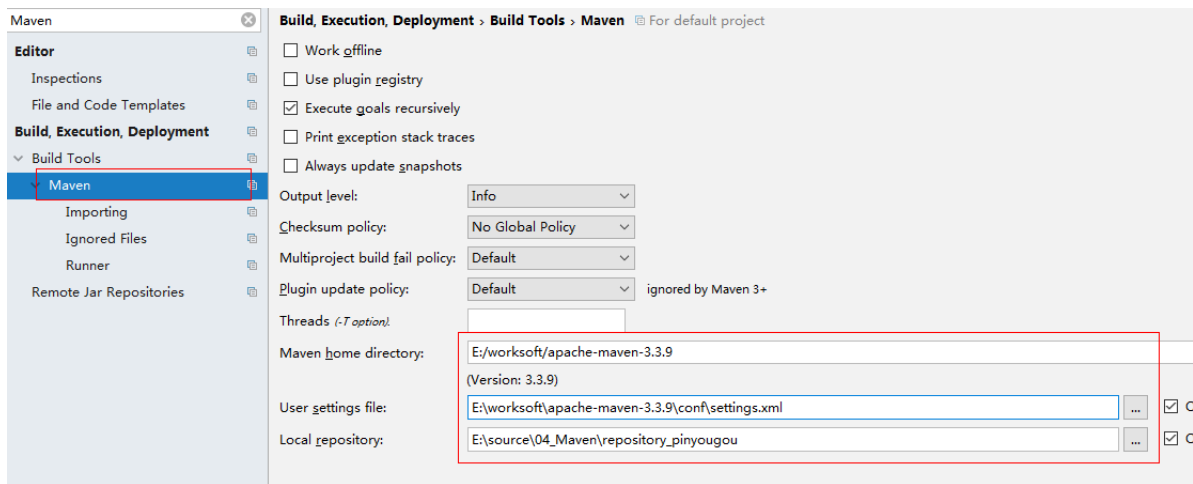
3.2. 配置默认Maven环境

每次创建Maven工程的时候，总是需要重新选择Maven配置信息，那是因为默认的Maven环境不是我们当前的maven环境，所以需要配置。配置流程如下图：

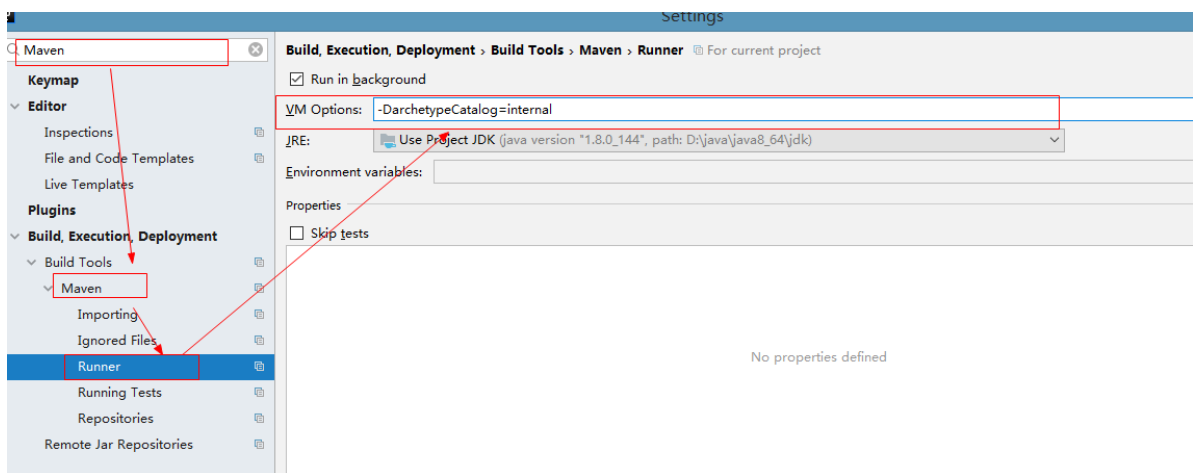
- 选择默认的配置



- 进入配置



- 配置参数(创建工程不需要联网,解决创建慢的问题) -DarchetypeCatalog=internal



- 重启IDEA, 就可以生效了

4.小结

- idea集成Maven

1.配置当前项目Maven环境:

file-settings-搜索 maven 配置三项(Maven Home目录、maven配置文件、maven本地仓库地址)

2.配置新项目Maven环境【默认】: 为了创建新项目时不需要再次配置Maven环境

file->new projects settings -> settings for new projects-->...

- 优化: 为了让创建项目时不联网, 速度比较快添加参数: -DarchetypeCatalog=internal

第三章-使用IDEA创建Maven工程【重点】

实操-创建javase工程

1.目标

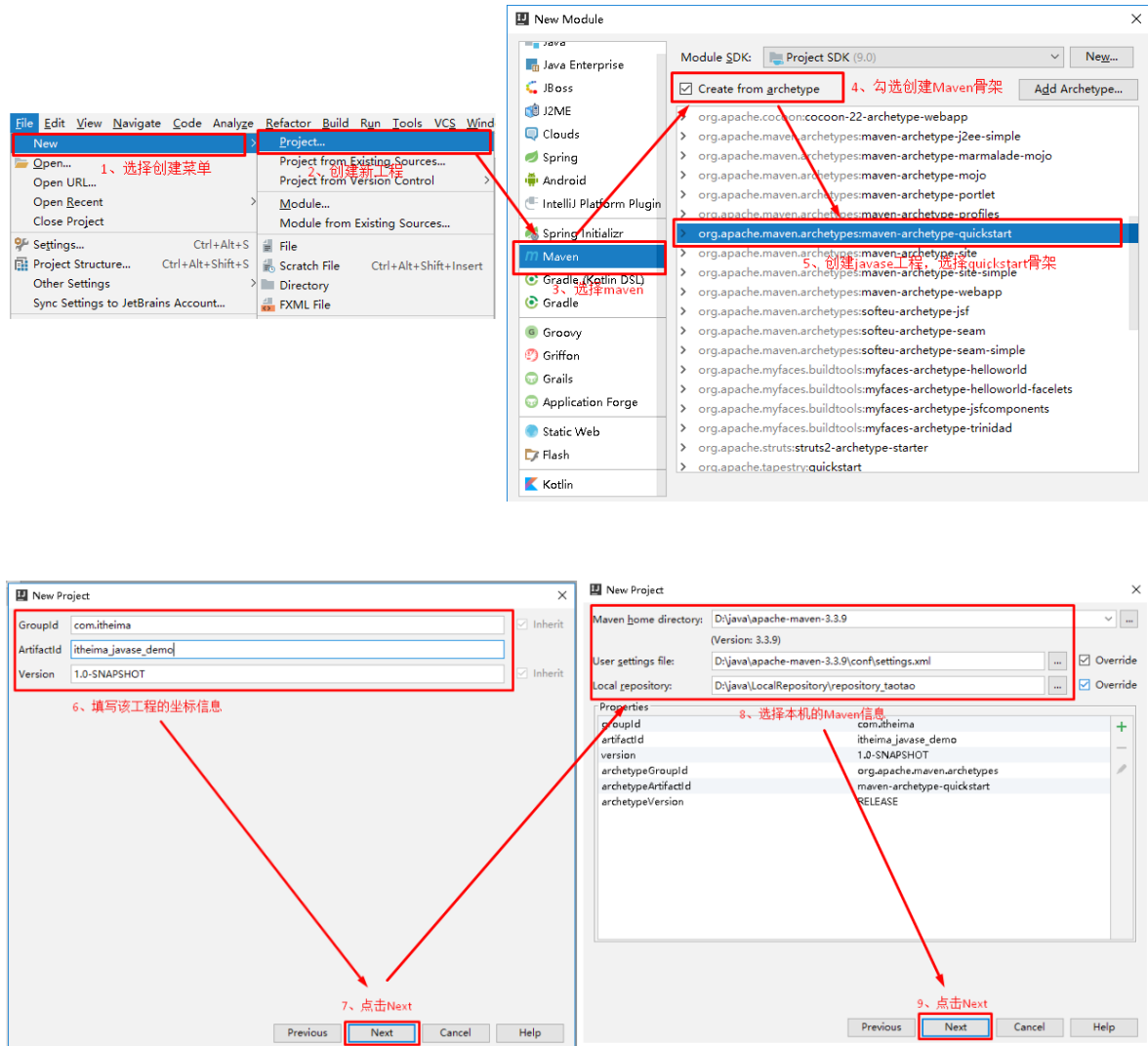
- 能够使用IDEA创建javase的Maven工程

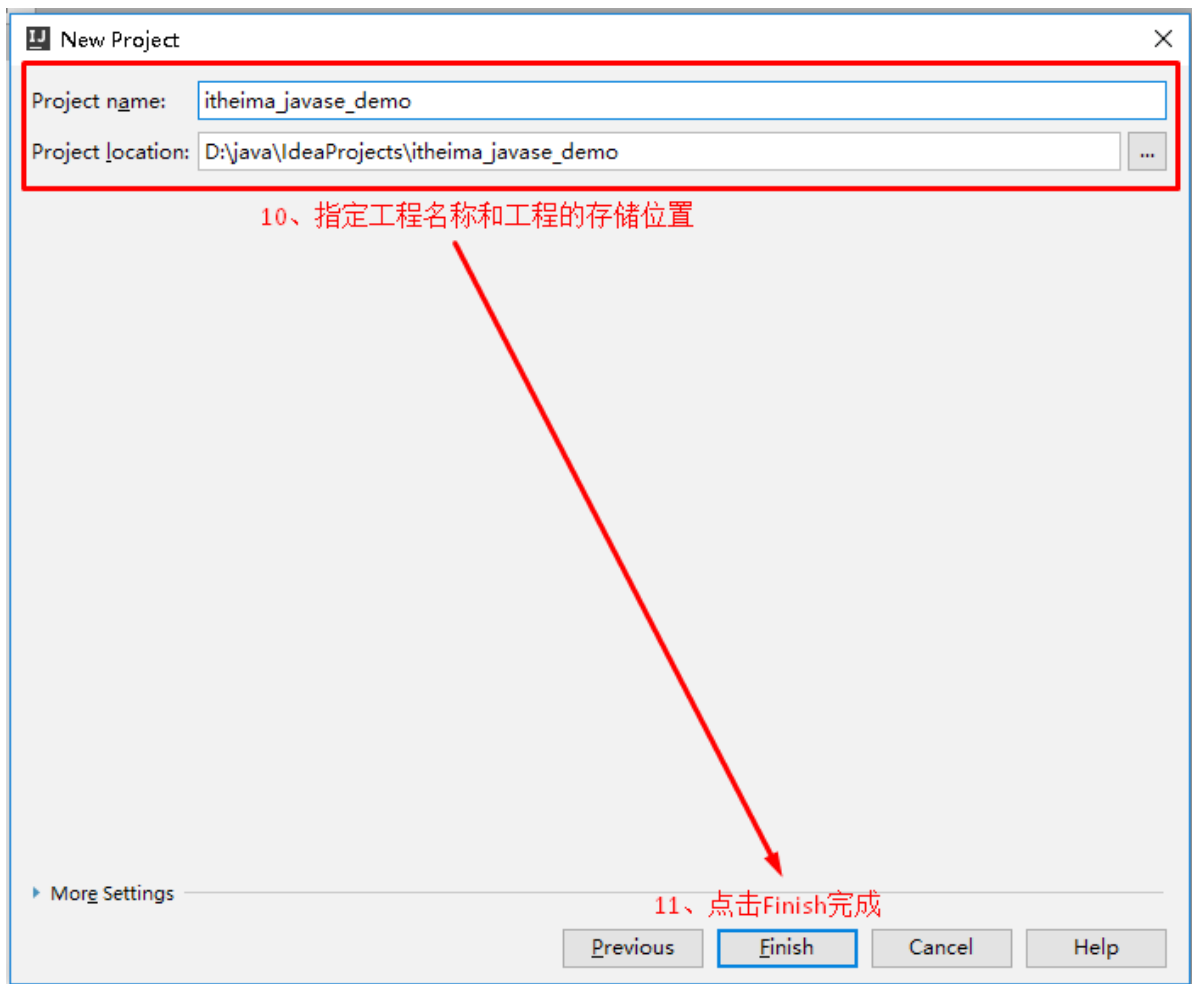
2. 路径

1. 创建java工程
2. java工程目录结构
3. 编写Hello World!

3. 讲解

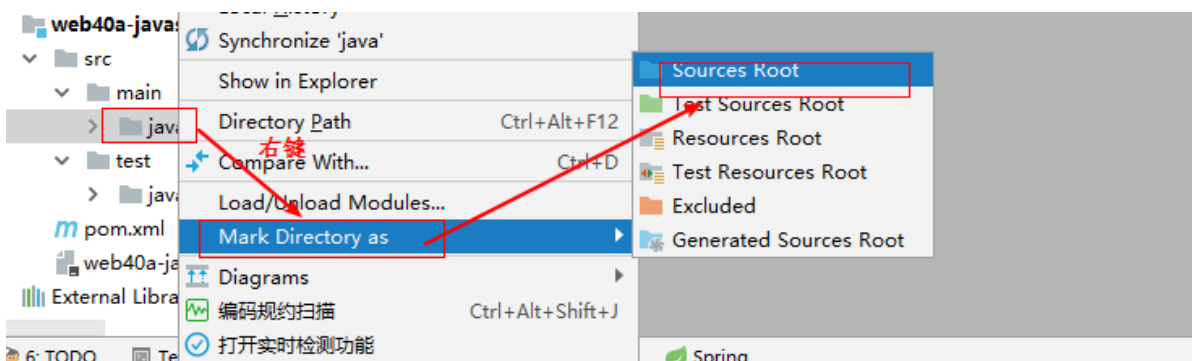
3.1 创建java工程



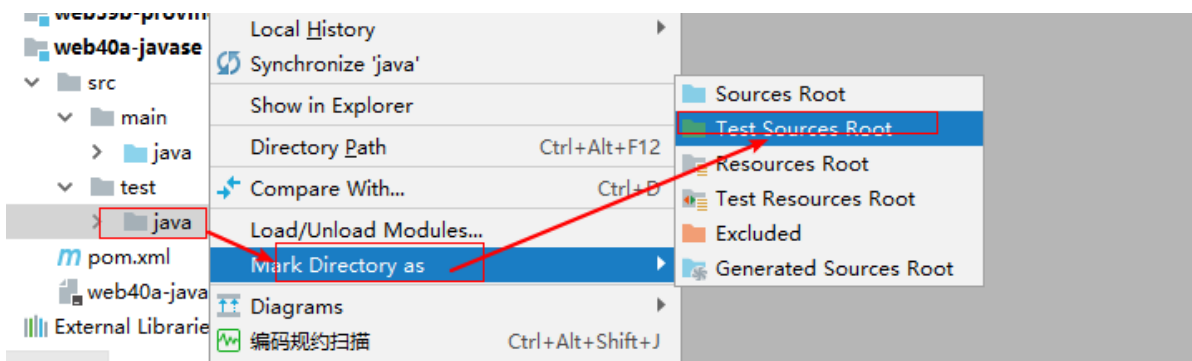


3.2 java工程目录结构

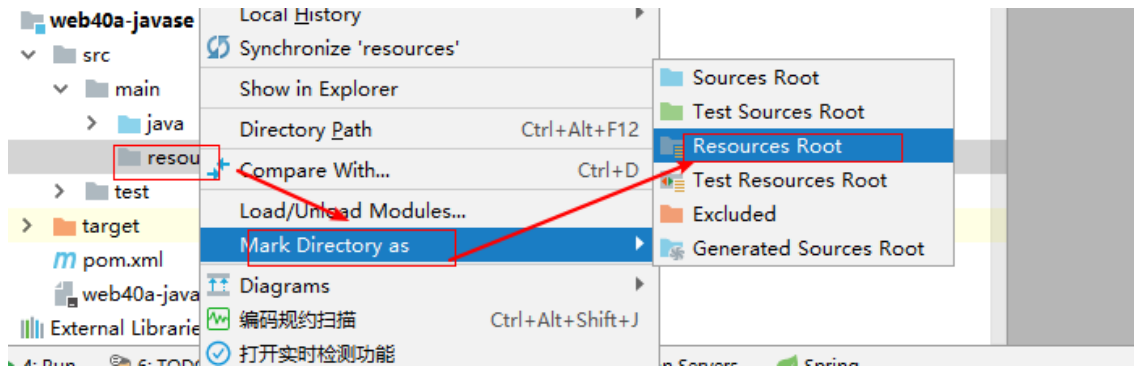
- 需要main/java文件夹变成 源码的目录(存放java源码)



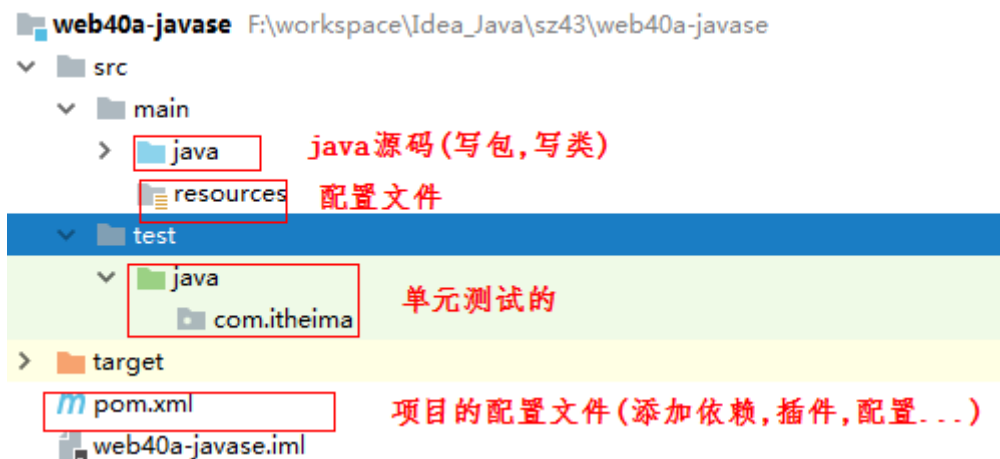
- 需要test/java文件夹变成 测试源码的目录(存放单元测试)



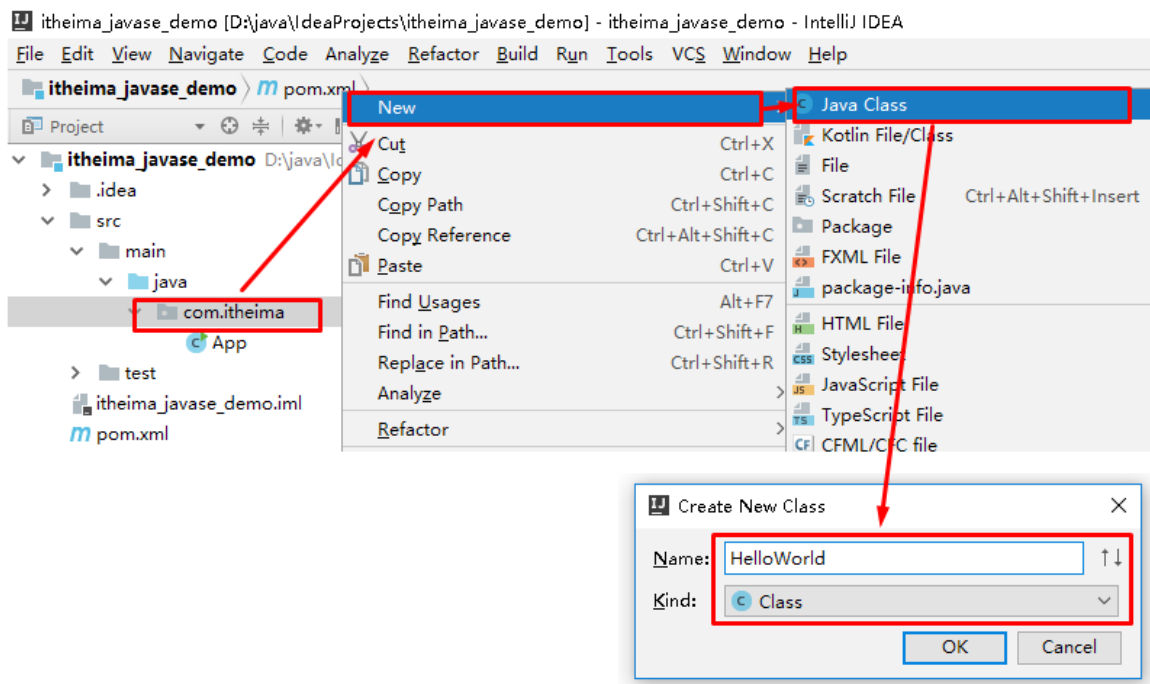
- 创建resources目录, 变成资源的目录



- 整体结构

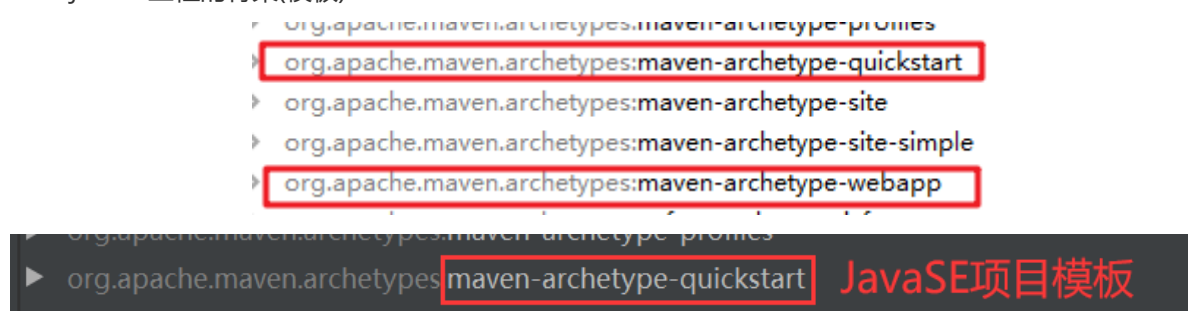


3.3 编写Hello World!

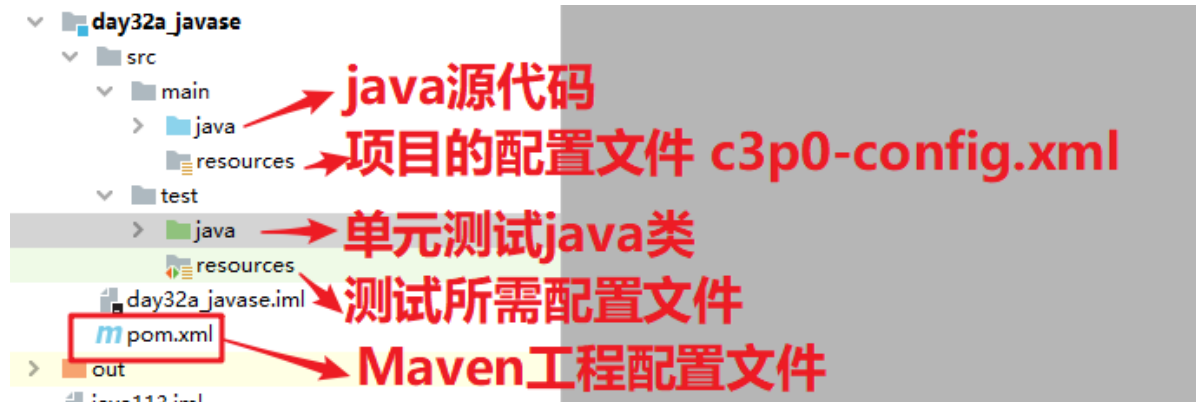


4.小结

1. JavaSe工程的骨架(模板)



2. 项目的结构



实操-创建javaweb工程

1.目标

- 能够使用IDEA创建javaweb的Maven工程

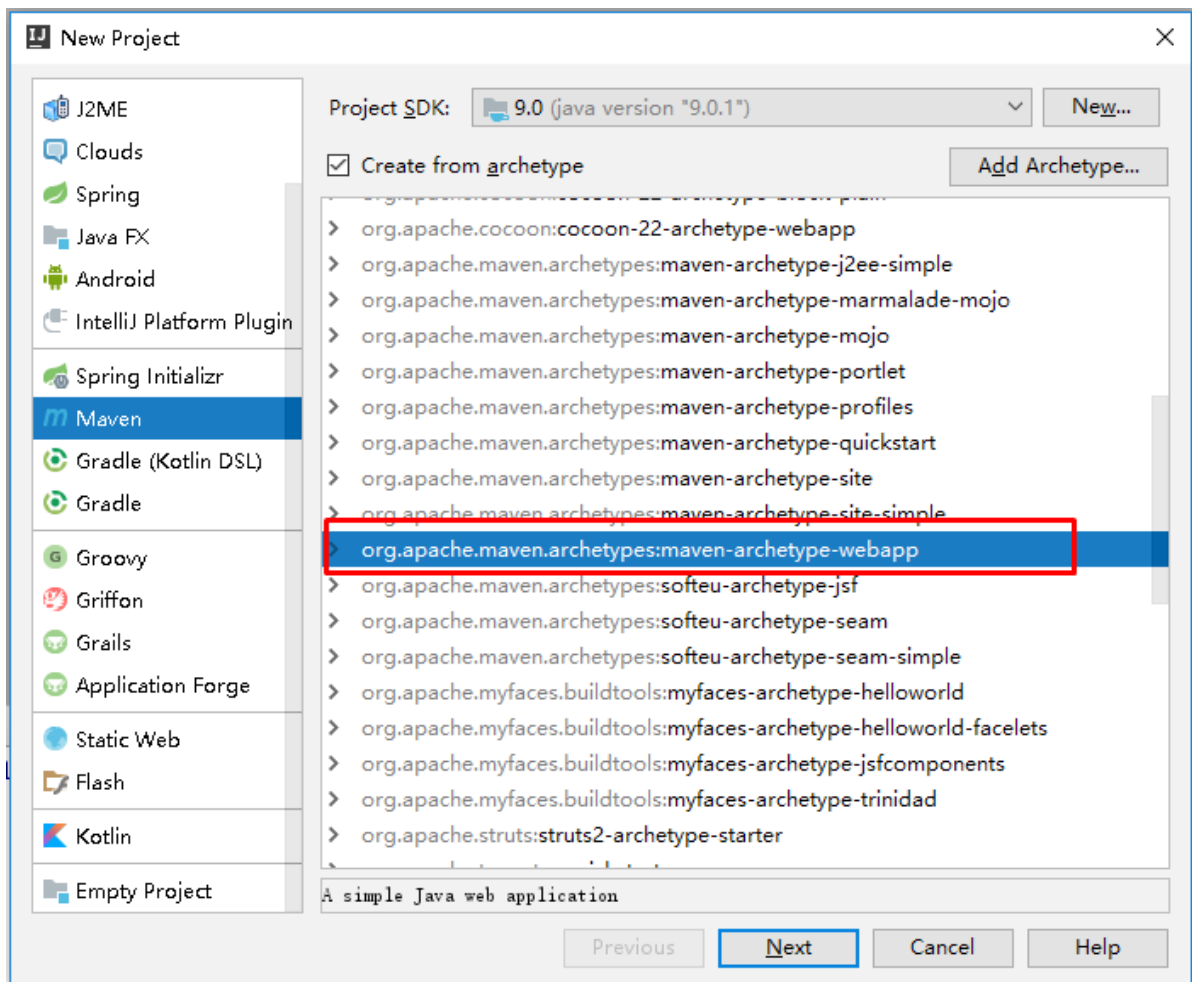
2.路径

1. 创建javaweb工程
2. 发布javaweb工程
3. 浏览器访问效果

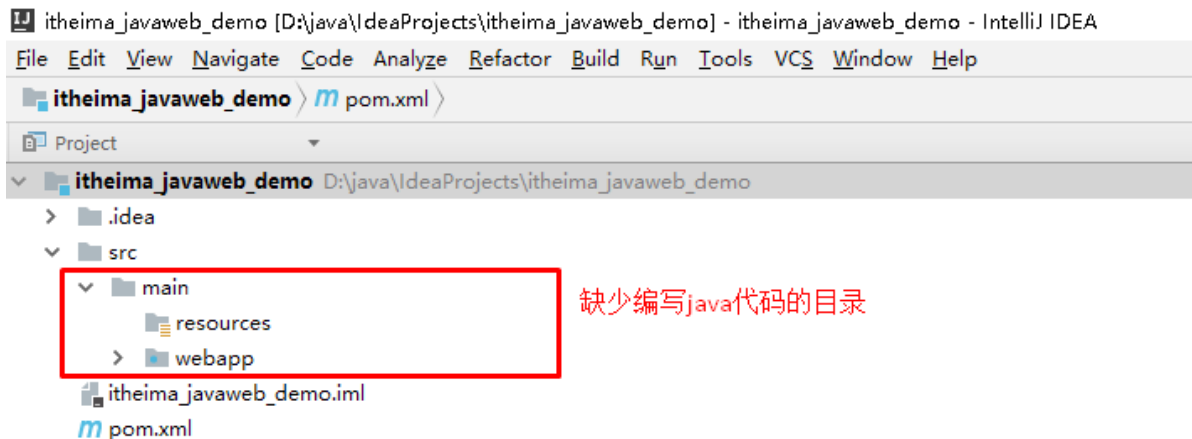
3.讲解

3.1 创建javaweb工程

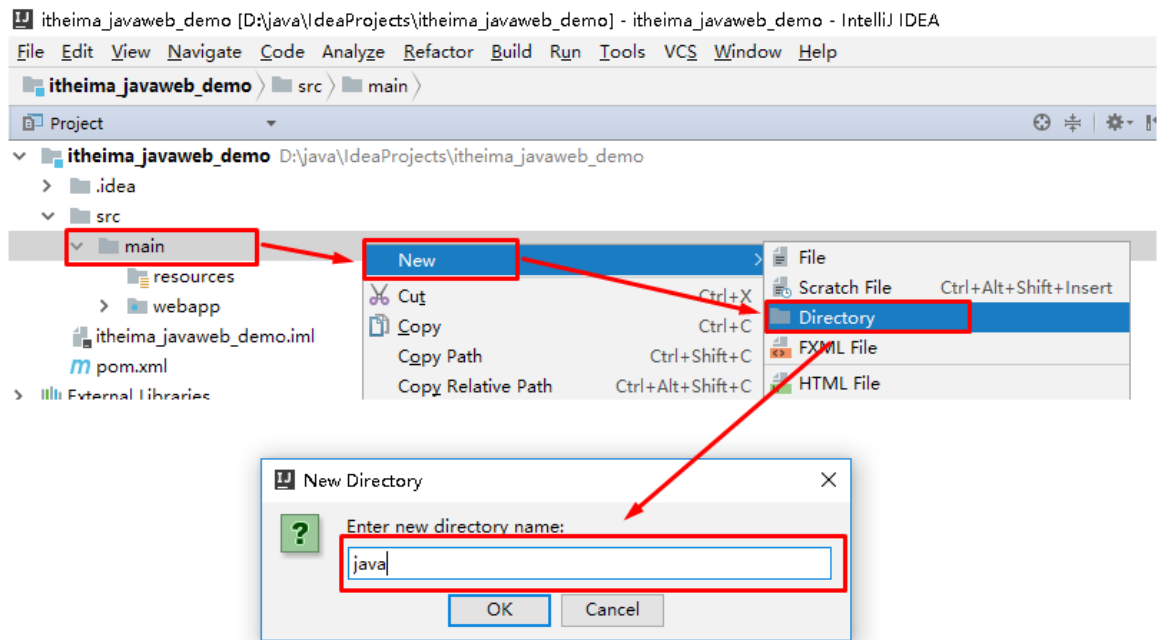
- 创建javaweb工程与创建javase工程类似，但在选择Maven骨架时，选择==maven-archetype-webapp==即可：



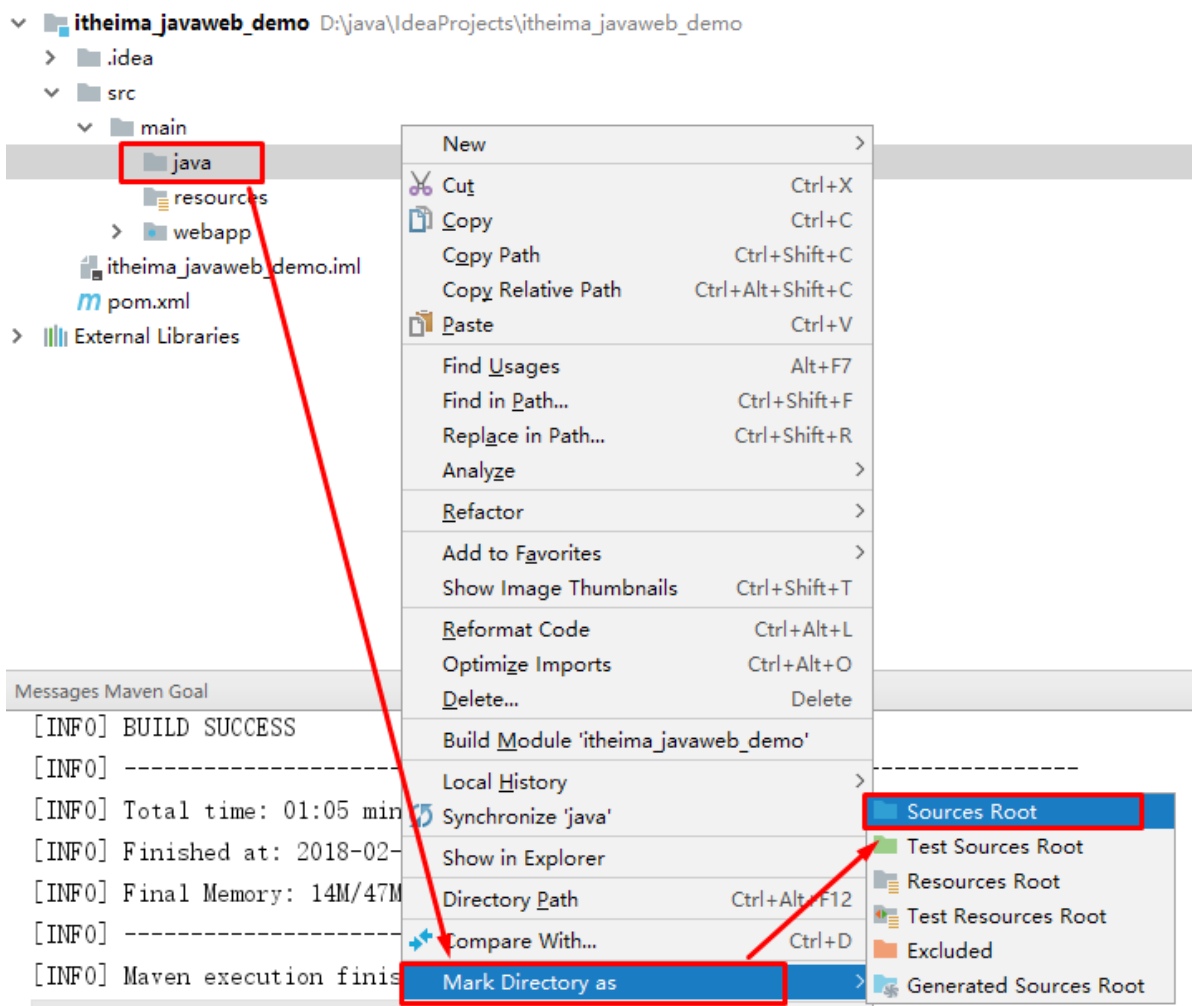
- 创建好的javaweb工程如下：



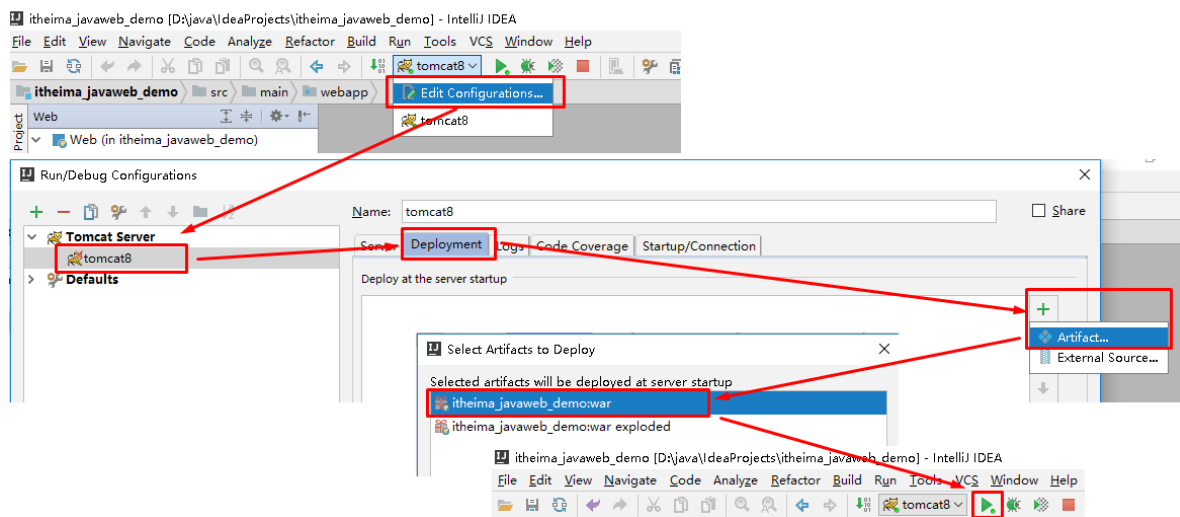
- 所以，要手动创建一个java目录用于编写java代码：



- 还要将java目录添加为Source Root:



3.2 发布javaweb工程



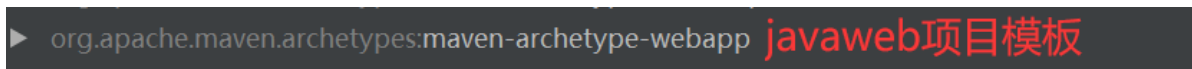
3.3 浏览器访问效果



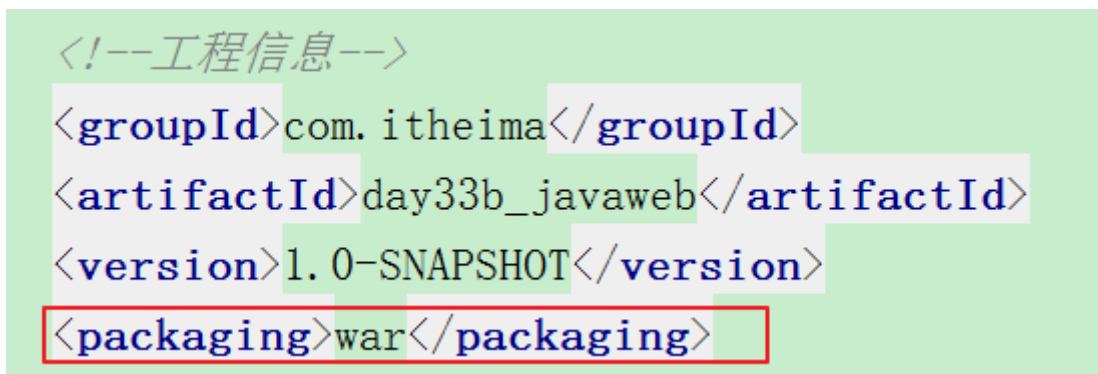
Hello World!

4.小结

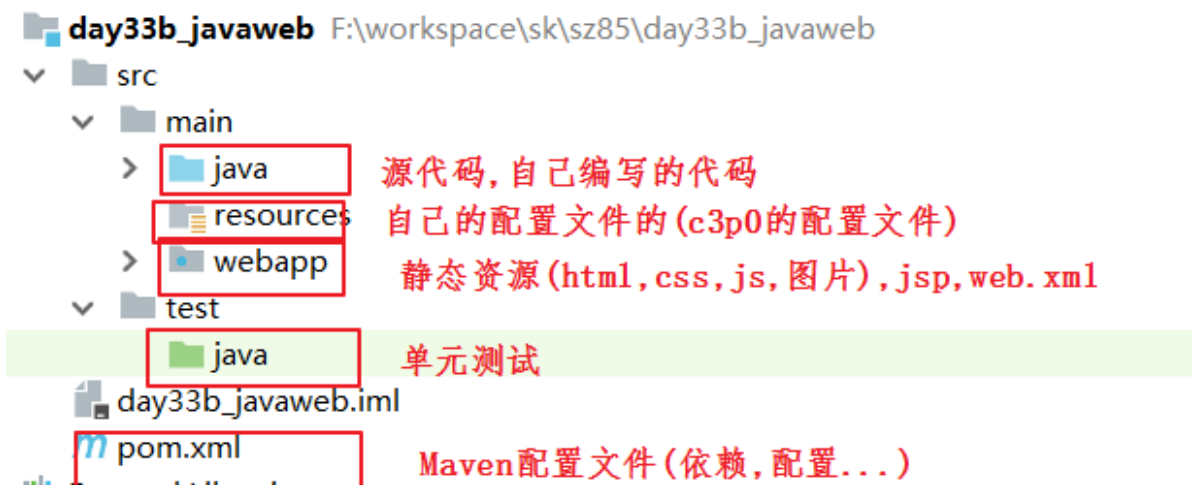
1. 选择骨架选择maven-archetype-webapp



2. pom.xml



3. web工程结构



实操-不使用骨架创建工程

1.目标

上面是使用骨架来创建工程的, 如果不使用骨架, 怎样创建工程呢?

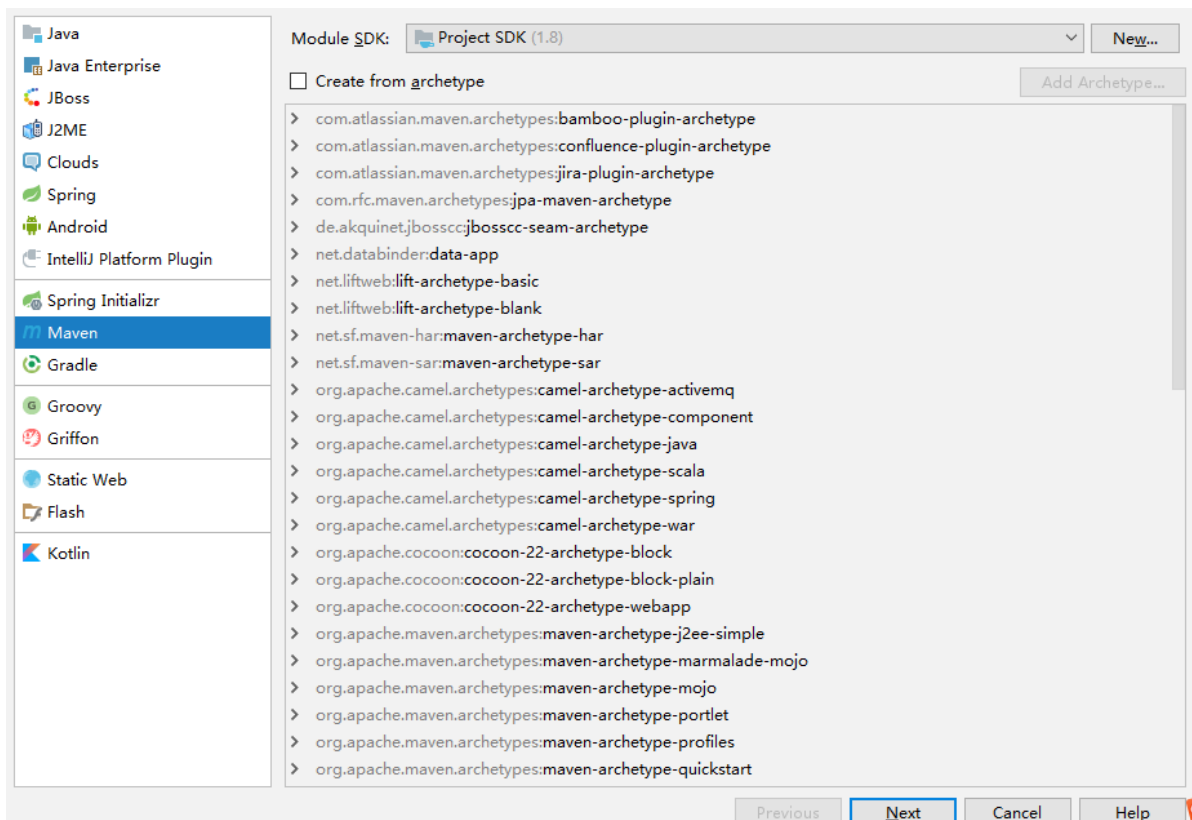
2.路径

1. 不使用骨架创建javase项目
2. 不使用骨架创建javaweb项目

3.讲解

3.1.不使用骨架创建javase项目

- 第一步



- 第二步

Add as module to <none> ...

Parent <none> ...

GroupId com.itheima

ArtifactId javase-demo

Version 1.0-SNAPSHOT

☒ Inherit

☒ Inherit

Previous Next Cancel Help

- 第三步

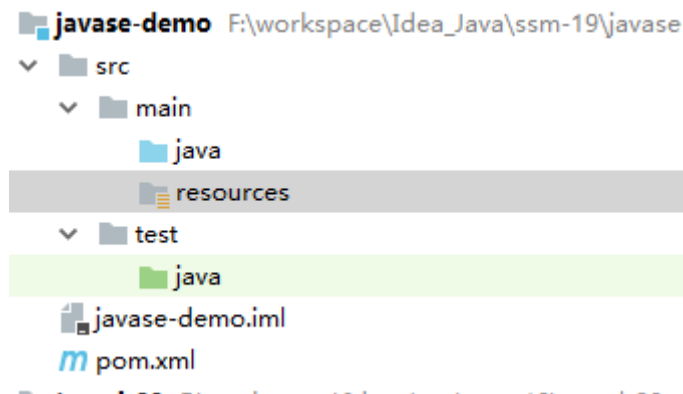
Module name: javase-demo

Content root: F:\workspace\Idea_Java\ssm-19\javase-demo ...

Module file location: F:\workspace\Idea_Java\ssm-19\javase-demo ...

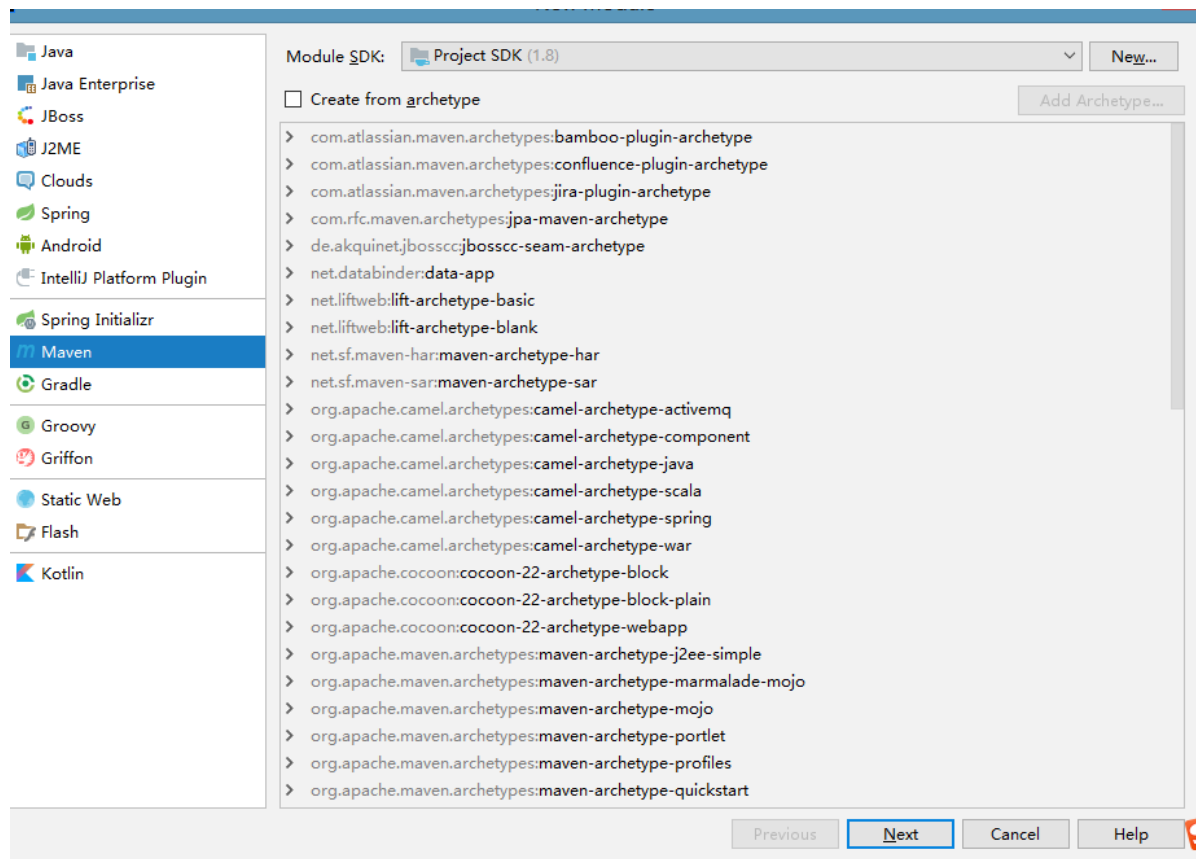
Previous Finish Cancel Help

- 第四步



3.2.不使用骨架创建javaweb项目

- 第一步



- 第二步

Add as module to <none> ...

Parent <none> ...

GroupId com.itheima

ArtifactId javaweb-demo

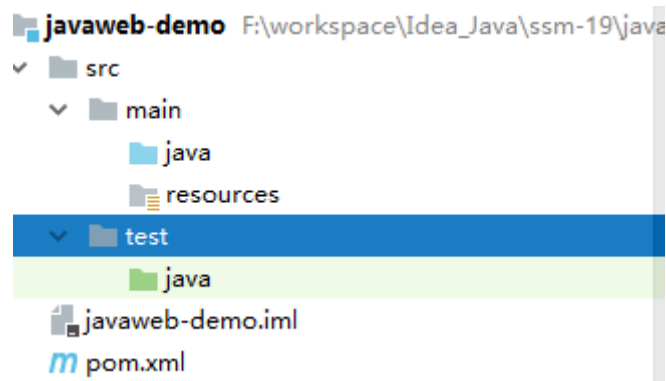
Version 1.0-SNAPSHOT

☒ Inherit

☒ Inherit

Previous Next Cancel Help

- 第三步



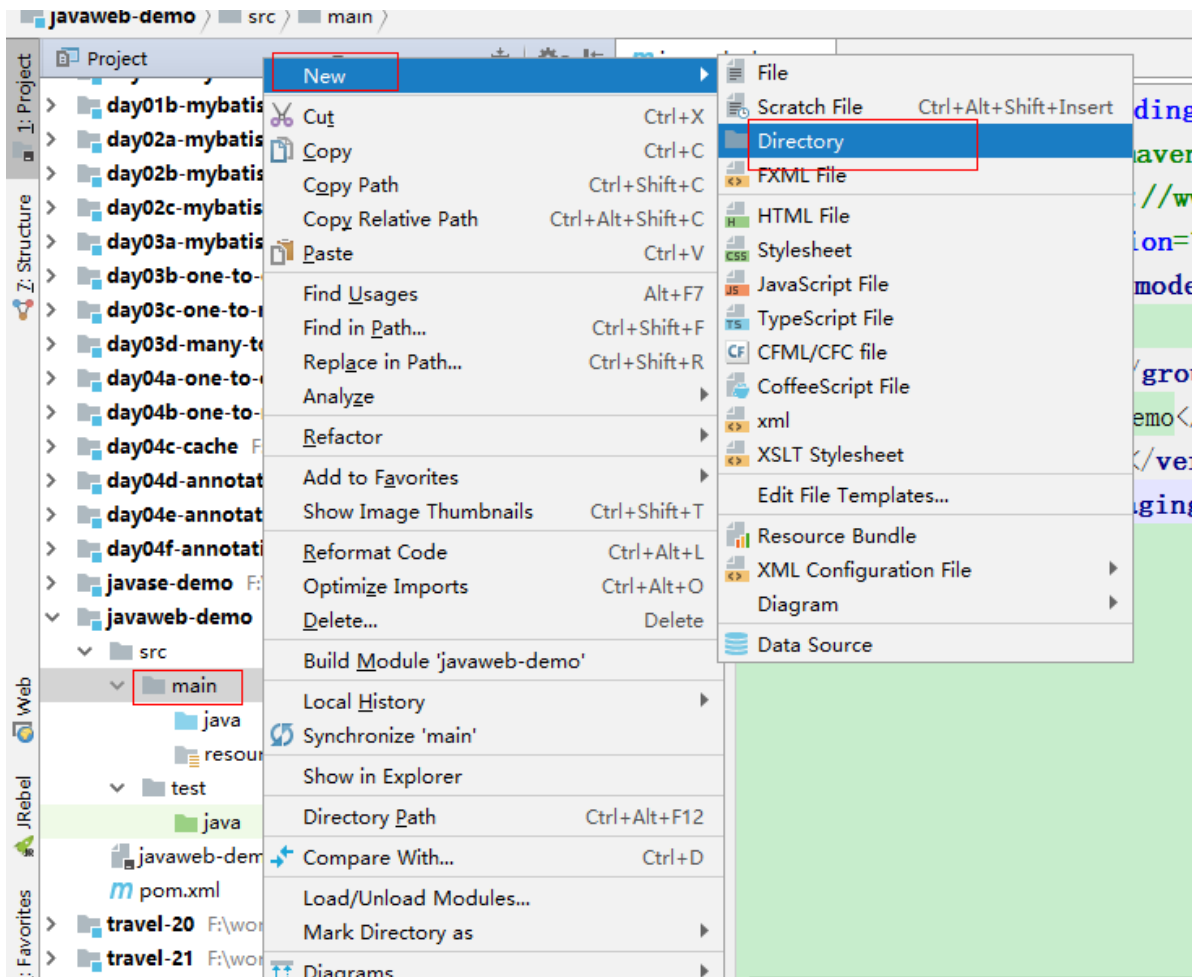
- 第四步, 在pom文件里面添加标签packaging

```
<?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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

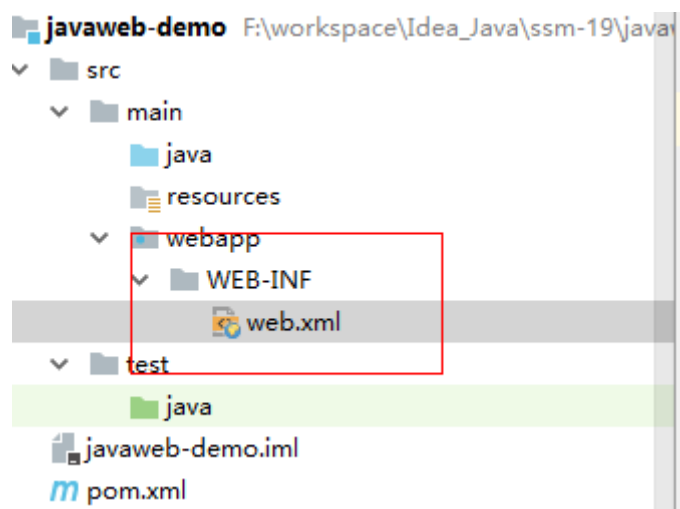
  <groupId>com.itheima</groupId>
  <artifactId>javaweb-demo</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
</project>
```

添加这个标签

- 第五步




- 第六步



4.小结

不使用骨架更快的, 单独有视频和笔记

 不使用骨架创建工程

第四章-Maven的常用命令【重点】

知识点-Maven的常用命令

1.目标

- 掌握Maven的常用命令

2.路径

1. clean命令
2. compile命令
3. test命令
4. package命令
5. install命令

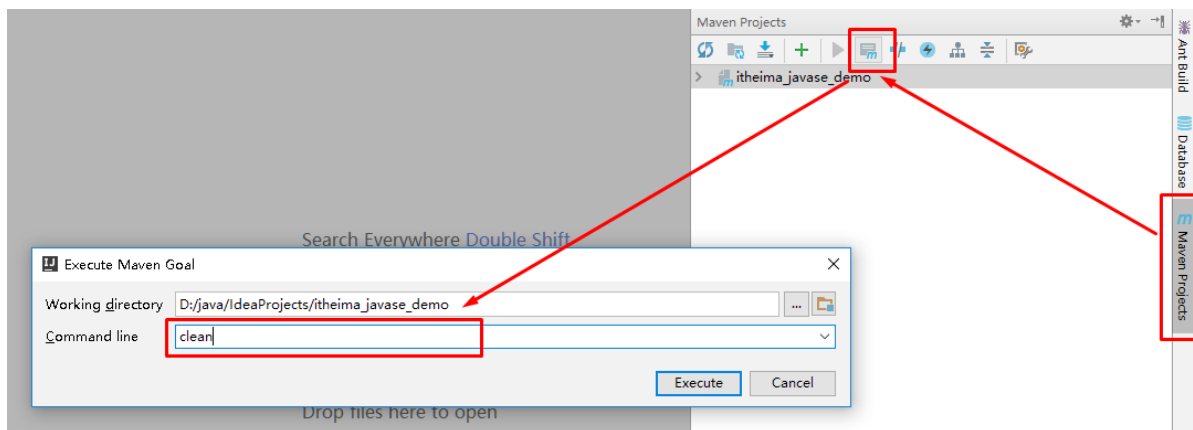
3.讲解

3.1clean命令

清除编译产生的target文件夹内容，作用：防止代码更新之后没有重新编译！

可以配合相应命令一起使用，如mvn clean package， mvn clean test

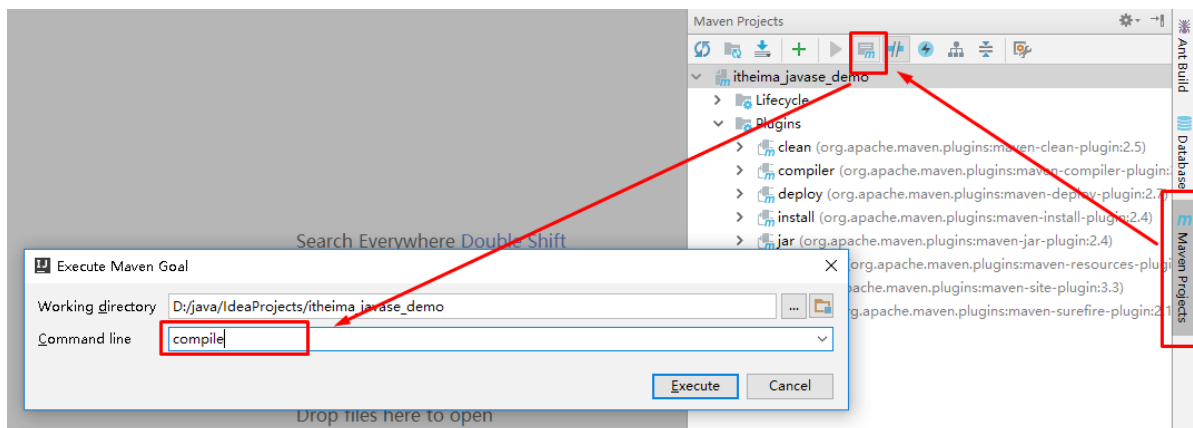
```
F:\workspace\Idea_Java\sz43\web40d-javaweb>mvn clean
```



3.2 compile命令

该命令可以对src/main/java目录的下的代码进行编译

```
F:\workspace\Idea_Java\sz43\web40d-javaweb>mvn compile
```

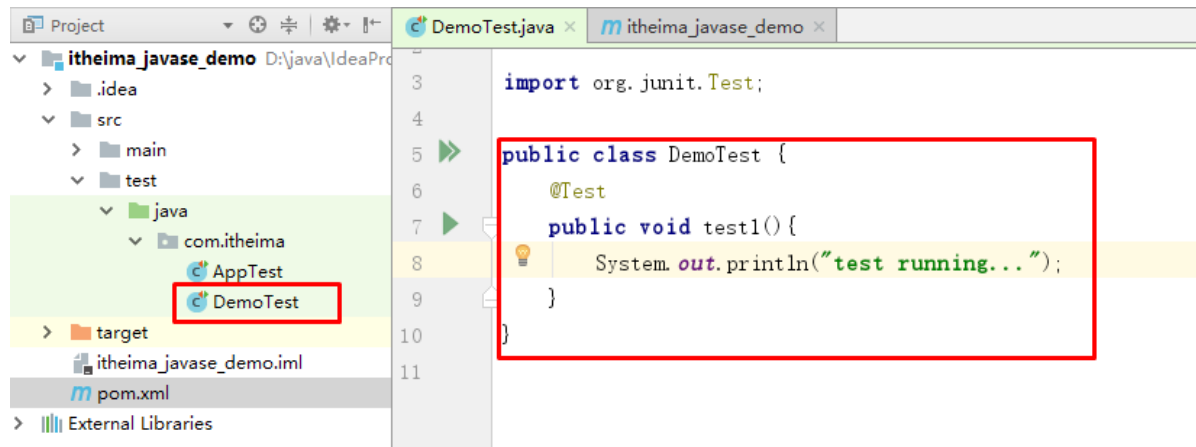


3.3 test命令

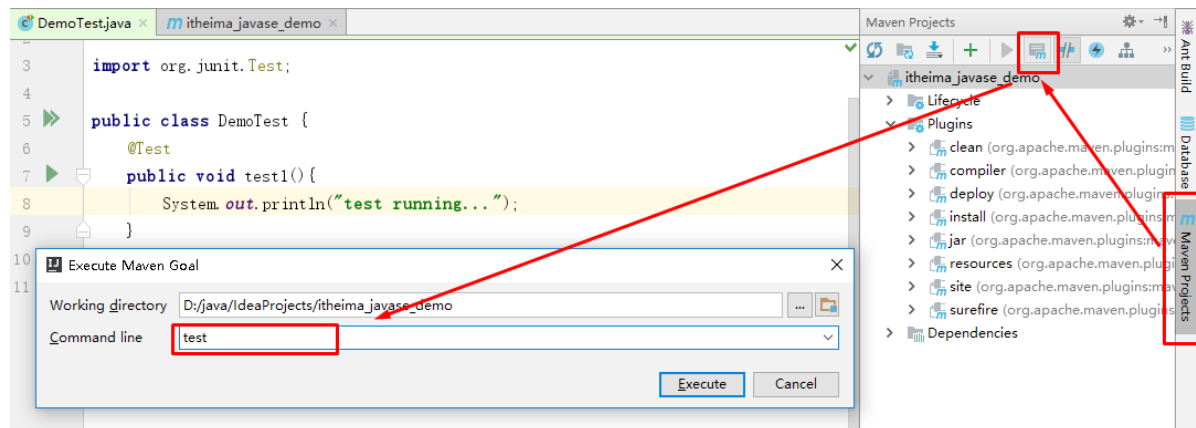
测试命令,或执行src/test/java/下所有junit的测试用例

```
F:\workspace\Idea_Java\sz43\web40d-javaweb>mvn test
```

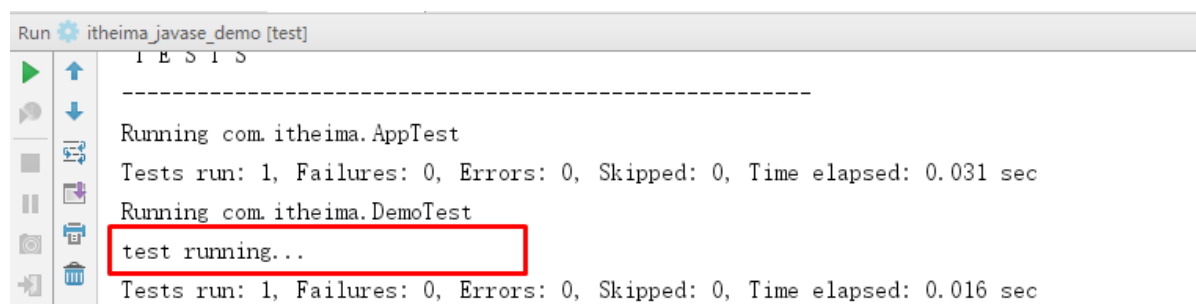
- 在src/test/java下创建测试类DemoTest



- 执行test命令测试



- 控制台显示测试结果

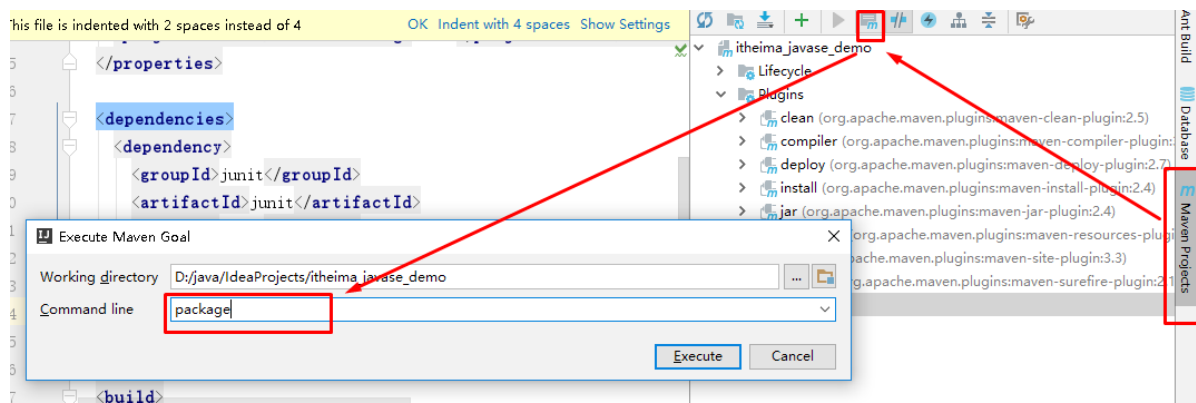


3.4 package命令

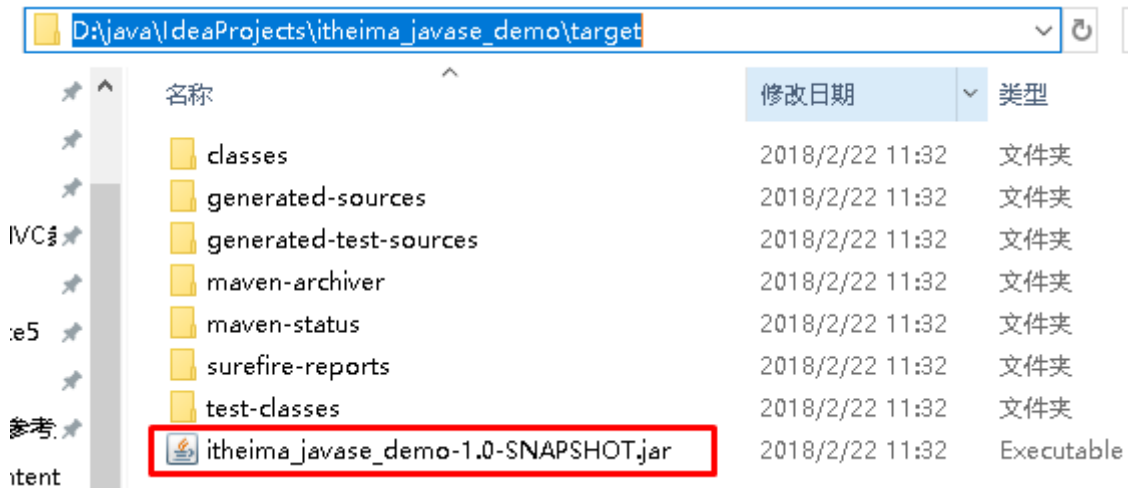
mvn package, 打包项目

- 如果是JavaSe的项目,打包成jar包 方便其他的项目引用
- 如果是JavaWeb的项目,打包成war包 方便部署

```
F:\workspace\Idea_Java\sz43\web40d-javaweb>mvn package
```



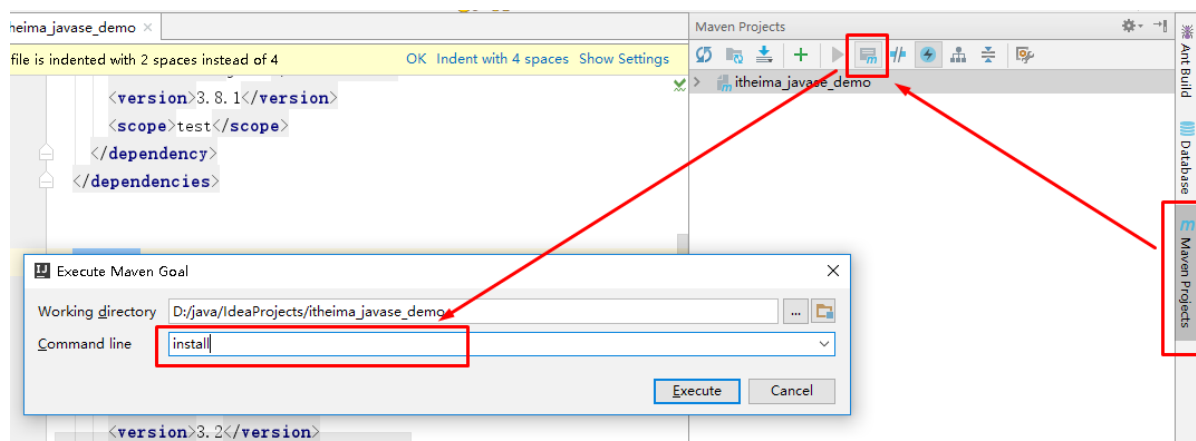
打包后的项目会在target目录下找到



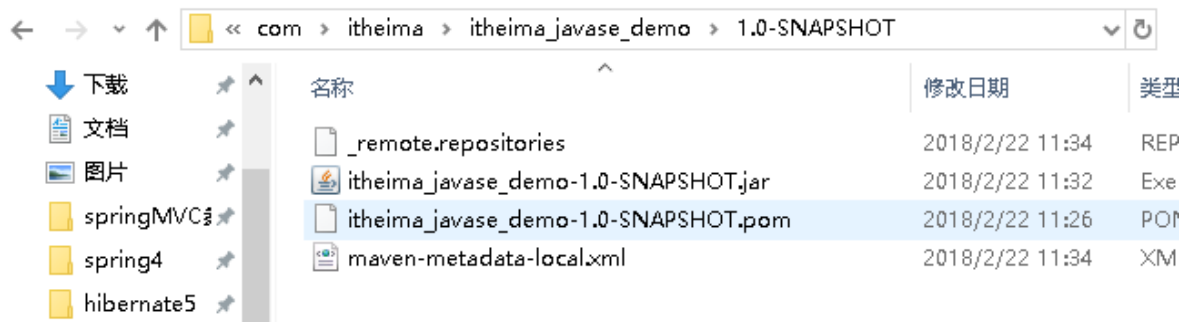
3.5 install命令

mvn install, 打包后将其安装在本地仓库

```
F:\workspace\Idea_Java\sz43\web40d-javaweb>mvn install
```



安装完毕后, 在本地仓库中可以找到itheima_javase_demo的信息



项目构建：

clean：清理缓存(清理编译后的target文件夹下的内容)

compile：编译，对项目重新编译 编译java源代码

test：测试所有junit用例

package：项目打包 javase项目打jar包 javaweb项目打war包

install：将打包后的项目安装到本地仓库 方便maven通过坐标引入到其他项目中使用

clean --> compile --> test --> package --> install

4.小结

1. 命令作用

- clean 用来清除编译后的文件(target文件夹里面的) == 【一般清缓存】 ==
- compile 编译
- test 执行单元测试
- package 打包 (==javaSe-->jar, javaweb-->war==)
- install 把项目打包之后==安装到本地仓库== 方便其他项目通过坐标引用

2. 生命周期

当我们执行了install 也会执行compile test package

第五章-依赖管理和插件

知识点-依赖管理

1.目标

- 能够掌握依赖引入的配置方式

2.路径

1. 导入依赖
2. 导入依赖练习
3. 依赖范围

3.讲解

3.1 导入依赖

导入依赖坐标，无需手动导入jar包就可以引入jar。在pom.xml中使用标签引入依赖。

做项目/工作里面 都有整套的依赖的, 不需要背诵的.

去Maven官网找, 赋值, 粘贴. <http://mvnrepository.com/>

3.1.1 导入junit的依赖

- 导入junit坐标依赖

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

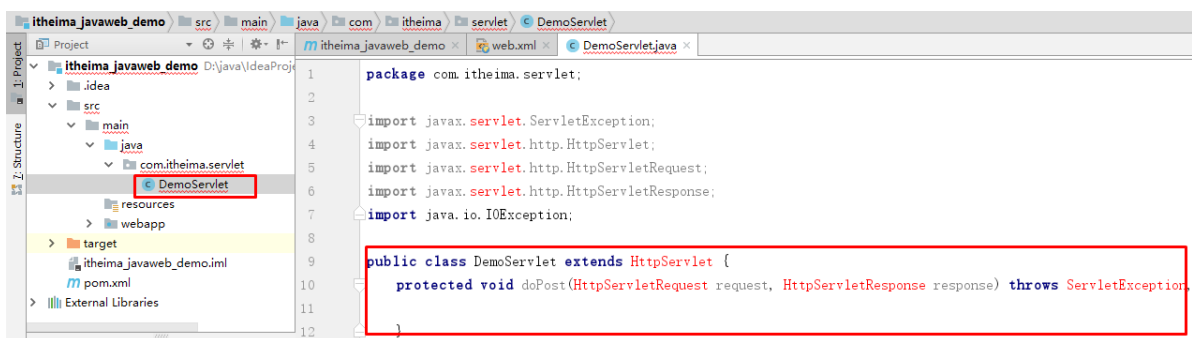
- 进行单元测试

```
import org.junit.Test;

public class DemoTest {
    @Test
    public void test1(){
        System.out.println("test running...");
    }
}
```

3.1.2 导入servlet的依赖

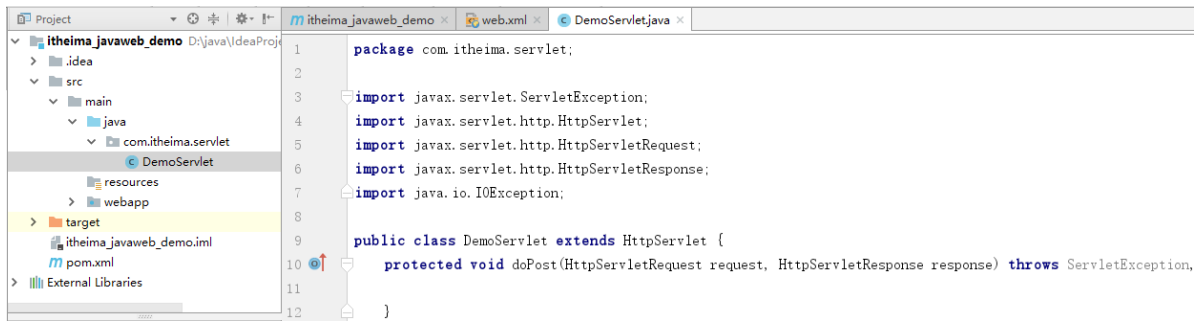
- 创建Servlet，但是发现报错，原因是没有导入Servlet的坐标依赖



- 导入Servlet的坐标依赖

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
</dependency>
```

- 原有报错的Servlet恢复正常



3.2 依赖范围

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

依赖范围指的就是jar包的被需要程度，项目可以简单分为三个阶段(编译时、测试时、运行时)

compile范围最广：默认就是compile，表示该jar包无论在测试时还是运行时都需要，必不可少

test：表示该jar包只需要出现在单元测试中，用于测试，而代码真正运行不需要。eg:jUnit

provided:表示内部提供，一般在编译时和测试时需要，运行时不需要。eg：servlet-api

runtime：测试时和运行时会用到，编译时不会。eg：数据库驱动

- compile 编译、测试、运行，A在编译时依赖B，并且在测试和运行时也依赖
例如：struts-core、spring-beans, C3P0, Druid。打到war包或jar包
- ==provided 编译、和测试有效==，A在编译和测试时需要B
例如：servlet-api就是编译和测试有用，在运行时不用（tomcat容器已提供）
不会打到war
- runtime：测试运行有效
例如：jdbc驱动包，在开发代码中针对java的jdbc接口开发，编译不用
在运行和测试时需要通过jdbc驱动包（mysql驱动）连接数据库，需要的
会打到war
- test：只是测试有效，只在单元测试类中用
例如：junit
不会打到war
- 按照依赖强度，由强到弱来排序：(理解)
compile> provided> runtime> test

4.小结

1. 坐标不需要背, 做项目时候/工作开发 都有整套的坐标. 如果是导入一些特定, 可以查阅网站, 直接拷贝

<https://mvnrepository.com/>

2. jar包作用范围(通过scope指定一个jar包的作用范围)

- compile 编译、测试、打包运行部署 需要 【默认】 druid、fastjson...
- **provided 编译, 测试 需要. 打包运行部署 不需要** servlet-api
- runtime 测试、打包运行部署 需要 编译不需要用到 jdbc驱动
- test 只是测试有效, 只在单元测试类中用 junit

3. Servlet,JSP 这类jar 需要加上provided, 因为部署到Tomcat里面. tomcat里面有, 如果没有加上provided, 可能会导致jar 冲突

单元测试的 建议加上test

知识点-Maven插件

1.目标

Maven是一个核心引擎, 提供了基本的项目处理能力和建设过程的管理, 以及一系列的插件是用来执行实际建设任务。maven插件可以完成一些特定的功能。例如, **集成jdk插件可以方便的修改项目的编译环境; 集成tomcat插件后, 无需安装tomcat服务器就可以运行tomcat进行项目的发布与测试。**在pom.xml中通过plugin标签引入maven的功能插件。

2.路径

1. JDK编译版本的插件
2. Tomcat的插件

3.讲解

3.1 JDK编译版本的插件【了解】

```
<!--项目构建及插件配置-->
<build>
  <plugins>
    <!--jdk编译插件-->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.2</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>utf-8</encoding>
      </configuration>
    </plugin>
  </plugins>
</build>
```


3.2 Tomcat7服务端的插件

- 添加tomcat7插件

```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <configuration>
    <!-- 指定端口 -->
    <port>82</port>
    <!-- 配置项目虚拟目录 -->
    <path></path>
  </configuration>
</plugin>
```

注意: Maven的中央仓库中只有Tomcat7.X版本的插件, 而之前我们使用的是8.X的版本, 如果想使Tomcat8.X的插件可以去其他第三方仓库进行寻找, 或者使用IDEA集成外部Tomcat8及其以上版本, 进行项目的发布。

4.小结

pom.xml:

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

第六章maven 私服【了解】

知识点- 私服搭建

1.目标

- ☐ 了解Maven私服搭建

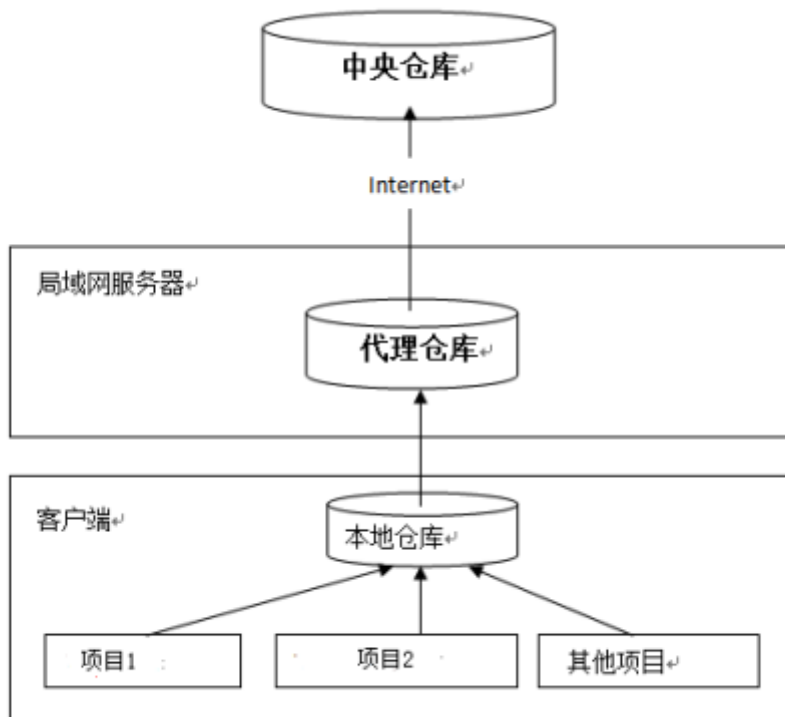
2.路径

1. Maven私服概述
2. 搭建私服环境

3.讲解

3.1Maven私服概述

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：



3.2.搭建私服环境

3.2.1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。

下载地址：<http://www.sonatype.org/nexus/archived/>

下载：nexus-2.12.0-01-bundle.zip

2 Download a Distribution

Once you've selected a version in Step 1, you can download a distribution from the following list.

Download Nexus 2.12.0-01

NEXUS OSS (TGZ)

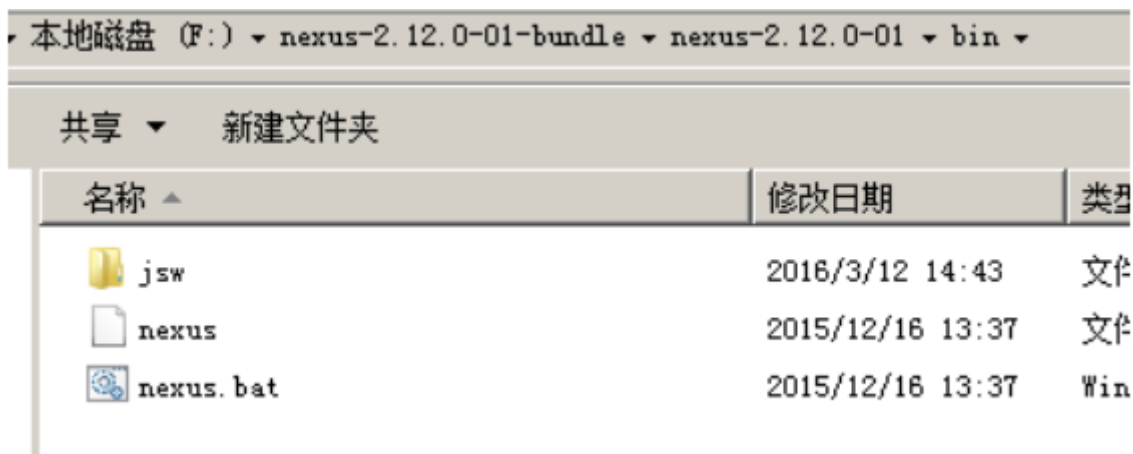
Checksums: MD5 SHA Signature: PGP

NEXUS OSS (ZIP)

Checksums: MD5 SHA Signature: PGP

3.2.2安装 nexus

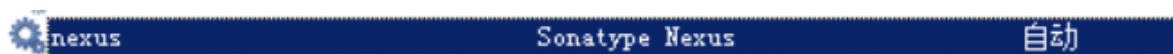
解压 nexus-2.12.0-01-bundle.zip, 进入 bin 目录:



以管理员权限运行命令行,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:



3.2.3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_
```

3.2.4启动 nexus

- 方式一
cmd 进入 bin 目录, 执行 nexus.bat start
- 方式二
直接启动 nexus 服务

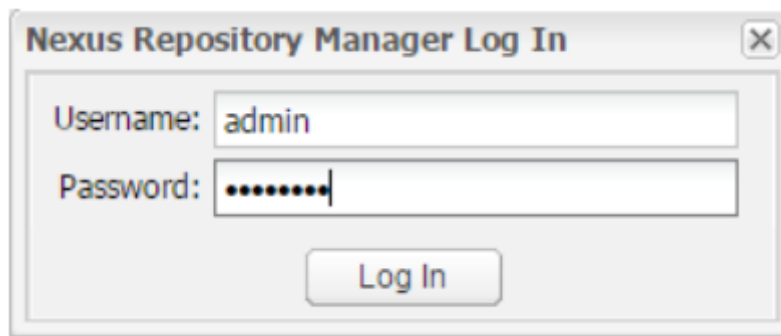


3.2.5登录

- 访问: <http://localhost:8081/nexus/>
查看 nexus 的配置文件 conf/nexus.properties ,里面有端口号



- 点击右上角的 Log in, 输入账号和密码 登陆 (账号admin,密码admin123)



- 登录成功

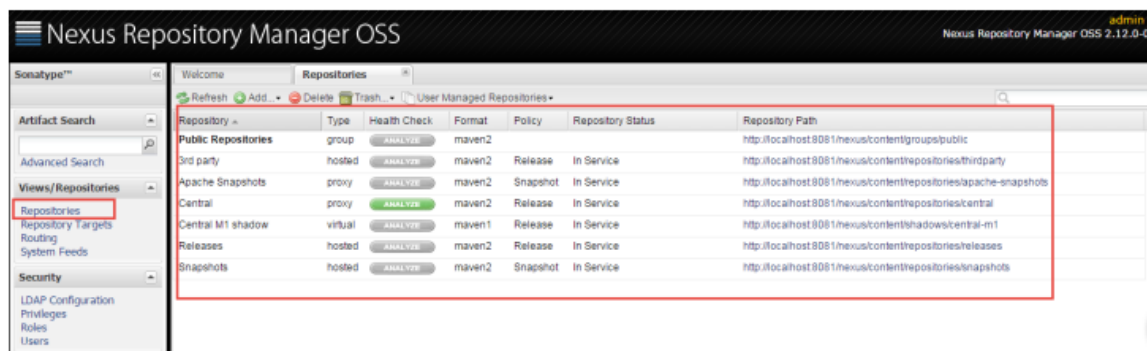


3.2.6仓库类型

自己实际图片1：

自己实际图片2：

老师示例图片：

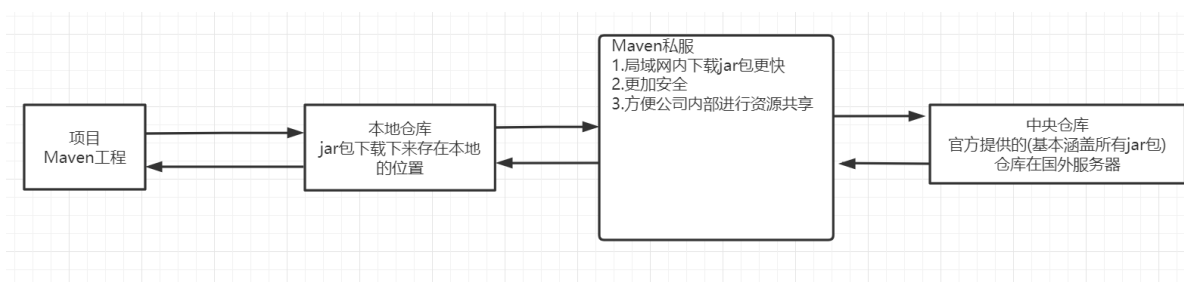


nexus 的仓库有 4 种类型:

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

4.小结



私服: 实际上就是一个远程仓库, 搭建的公司局域网内部

作用: ①保证信息安全 ②方便进行资源共享

知识点-Maven私服的使用

1.目标

☐ 了解Maven私服的使用

2.路径

1. 将项目发布到私服
2. 从私服下载 jar 包

3.讲解

3.1.将项目发布到私服

3.1.1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。某个团队开发完在common_utils, 将 common_utils发布到私服供 其它团队使用。

3.1.2配置

第一步：需要在客户端即部署 common_utils工程的**电脑上配置 maven环境，并修改 settings.xml 文件**(Maven配置文件)，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 (配置到标签下)

```
<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

Releases	hosted	ANALYZE	maven2	Release	In Service
Snapshots	hosted	ANALYZE	maven2	Snapshot	In Service

第二步：**在需要发布配置项目 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 和 settings.xml 配置 对应！

3.1.3测试

- 1、首先启动 nexus
- 2、对 common_utils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：

这台电脑 > 新加卷 (E:) > source > 04_Maven > nexus-2.12.0-01-bundle > sonatype-work > nexus > storage > snapshots > com > itheima > common_utils > 1.0-SNAPSHOT

名称	修改日期	类型	大小
common_utils-1.0-20191222.131828...	12/22 21:18	Executable Jar File	4 KB
common_utils-1.0-20191222.131828...	12/22 21:18	MD5 文件	1 KB

3.2从私服下载 jar 包

3.2.1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 common_utils工程 jar 包。

3.2.2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。（==配置在 <profiles> 标签下==）

```
<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。

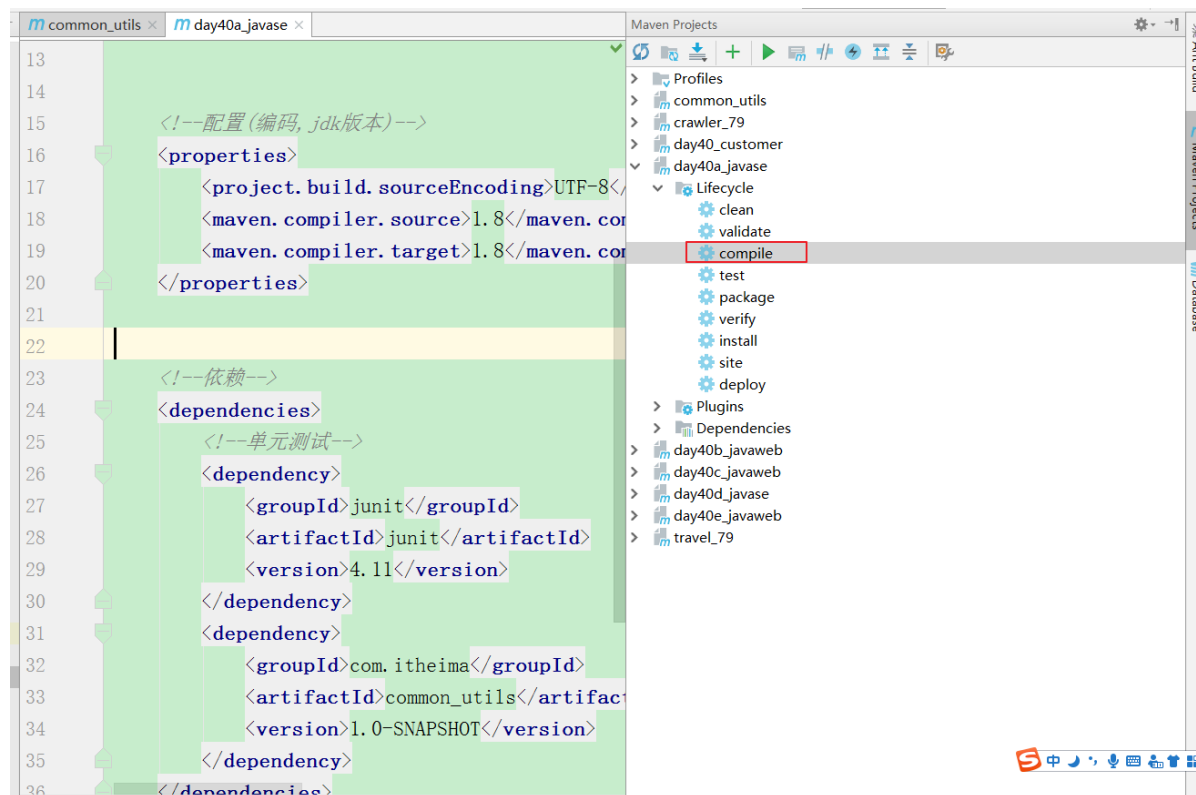
```

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

```

3.2.3测试从私服下载 jar 包

- 删掉本地仓库的common_utils
- 编译依赖common_utils的工程



- 出现如下日志


```
[INFO] -----
Downloading: http://localhost:8081/nexus/content/groups/public/com/itheima/common\_utils/1.0-SNAPSHOT/maven-metadata.xml
Downloaded: http://localhost:8081/nexus/content/groups/public/com/itheima/common\_utils/1.0-SNAPSHOT/maven-metadata.xml (767 B at 3.0 KB/sec)
Downloading: http://localhost:8081/nexus/content/groups/public/com/itheima/common\_utils/1.0-SNAPSHOT/common\_utils-1.0-20191222.131828-1.pom
Downloaded: http://localhost:8081/nexus/content/groups/public/com/itheima/common\_utils/1.0-SNAPSHOT/common\_utils-1.0-20191222.131828-1.pom (2 K
at 38.0 KB/sec)
Downloading: http://localhost:8081/nexus/content/groups/public/com/itheima/common\_utils/1.0-SNAPSHOT/common\_utils-1.0-20191222.131828-1.jar
Downloaded: http://localhost:8081/nexus/content/groups/public/com/itheima/common\_utils/1.0-SNAPSHOT/common\_utils-1.0-20191222.131828-1.jar (4 K
at 127.5 KB/sec)
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ day40a_javase ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ day40a_javase ---
[INFO] Nothing to compile - all classes are up to date
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.725 s
[INFO] Finished at: 2019-12-22T21:22:34+08:00
[INFO] Final Memory: 10M/203M
```



4.小结

1. 对着文档配置就OK
2. 注意: JDK的版本不要是9以上!!!

知识点-把第三方 jar 包放入本地仓库和私服

1.目标

- ☐ 掌握把第三方 jar 包放入本地仓库和私服

2.路径

1. 导入本地库
2. 导入私服
3. 参数说明

3.讲解

3.1.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 -Dfile=D:\czbk\java113\ojdbc14.jar -DgroupId=com.oracle
-DartifactId=ojdbc -Dversion=14 -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\syl\AppData\Local\Temp\mvninstall7088507939706576412.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
```

work (D:) > ITCAST > software > maven-repo > com > alibaba > fastjson > 1.1.37

名称	修改日期	类型	大小
_remote.repositories	2018/3/9 22:13	REPOSITORIES ...	1 KB
fastjson-1.1.37.jar	2016/3/4 13:42	JAR 文件	349 KB
fastjson-1.1.37.pom	2018/3/9 22:13	POM 文件	1 KB

3.2.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=D:\czbk\java113\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] --- 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/fastjso
n-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/fastjso
n-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-metadate
a.xml
Uploaded to thirdparty: http://localhost:8081/nexus/content/repositories/thirdparty/com/alibaba/fastjson/maven-metadate
a.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	

3.3.参数说明

DgroupId 和 ArtifactId 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。

4.小结

1. 有些jar中央仓库没有(eg:oracle驱动), 从官网/网络上下载下来, 安装到本地仓库. 我们的Maven项目就可以使用了
2. **注意：Maven项目有时候很多错误都是由于使用坐标下载jar包不成功或失败导致的，遇到这种情况，删除本地仓库下载的jar包文件夹，重新再将坐标放入pom.xml中重新下载一次**
3. 上传到本地仓库 mvn install
4. 上传到私服 mvn deploy

知识点-配置阿里云远程仓库

1.目标

- ☐ 掌握Maven配置阿里云远程仓库

2.路径

1. 配置阿里云远程仓库

3.讲解

1. 将Maven配置文件settings.xml中原有的关于私服配置的内容全部清除。
2. 将如下内容放入settings.xml中的==mirrors==标签内部即可

```
<!--阿里巴巴远程仓库镜像配置-->
<mirror>
  <id>nexus-aliyun</id>
  <mirrorOf>*</mirrorOf>
  <name>Nexus aliyun</name>
  <url>http://maven.aliyun.com/nexus/content/groups/public</url>
</mirror>
<mirror>
  <id>nexus-aliyun</id>
  <mirrorOf>central</mirrorOf>
  <name>Nexus aliyun</name>
  <url>http://maven.aliyun.com/nexus/content/groups/public</url>
</mirror>
```

第七章-Lombok

知识点-Lombok介绍和配置

1.目标

- ☐ 掌握Lombok的配置

2.路径

1. 什么是Lombok
2. Lombok的作用
3. Lombok的配置

3.讲解

3.1什么是Lombok

Lombok是一个Java库，能自动插入编辑器并构建工具，简化Java开发。

官网: <https://www.projectlombok.org/>

3.2Lombok的作用

通过==添加注解==的方式，Lombok能以简单的注解形式来简化Java代码，提高开发人员的开发效率。

例如开发中经常需要写的JavaBean，都需要花时间去添加相应的getter/setter，也许还要去写构造器、equals等方法，而且需要维护，当属性多时会出现大量的getter/setter方法，这些显得很冗长也没有太多技术含量，一旦修改属性，就容易出现忘记修改对应方法的失误，使代码看起来更简洁些。

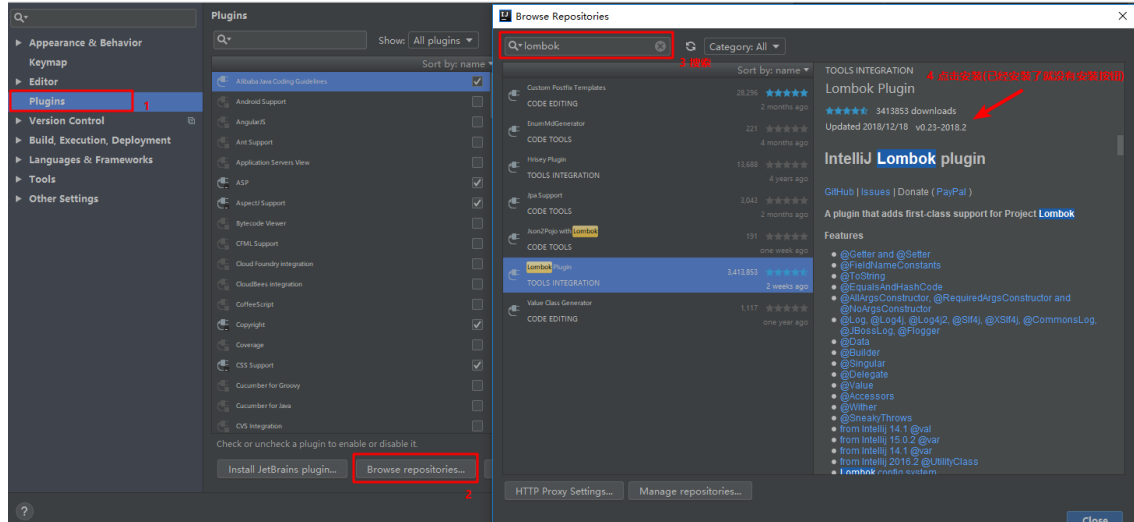
3.3Lombok的配置

- 添加maven依赖

```
<!--lombok-->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.18</version>
</dependency>
```

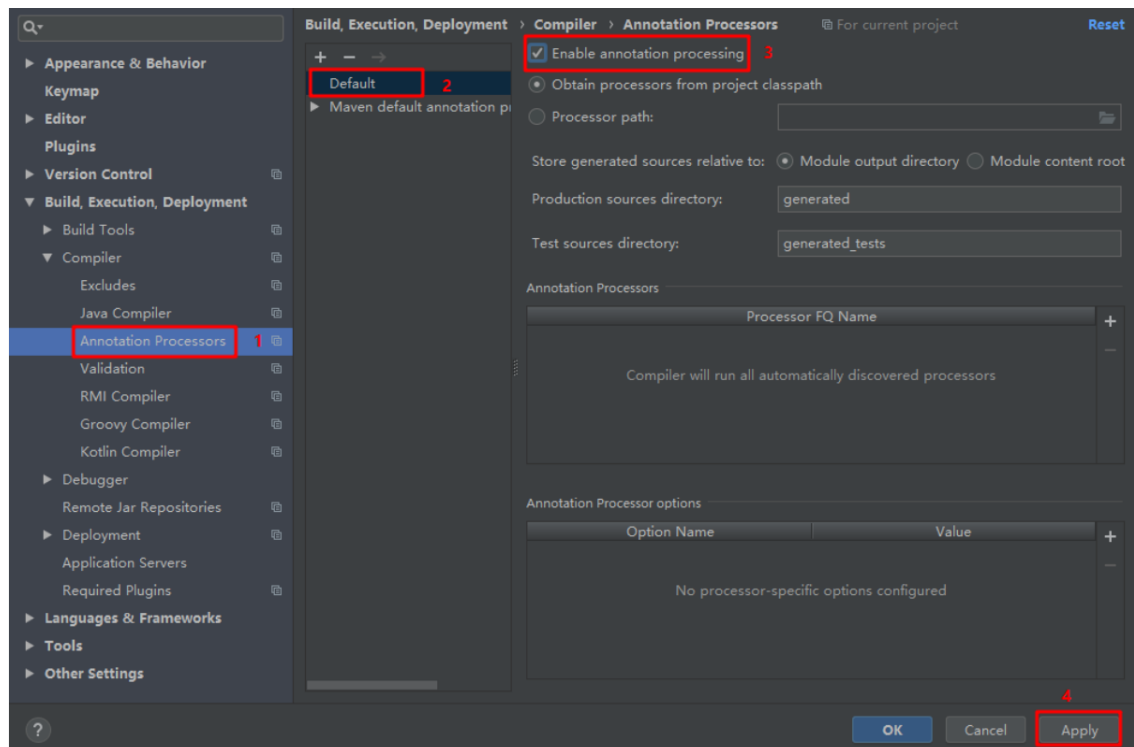
• 安装插件

使用Lombok还需要插件的配合，我使用开发工具为idea. 打开idea的设置，点击Plugins，点击Browse repositories，在弹出的窗口中搜索lombok，然后安装即可



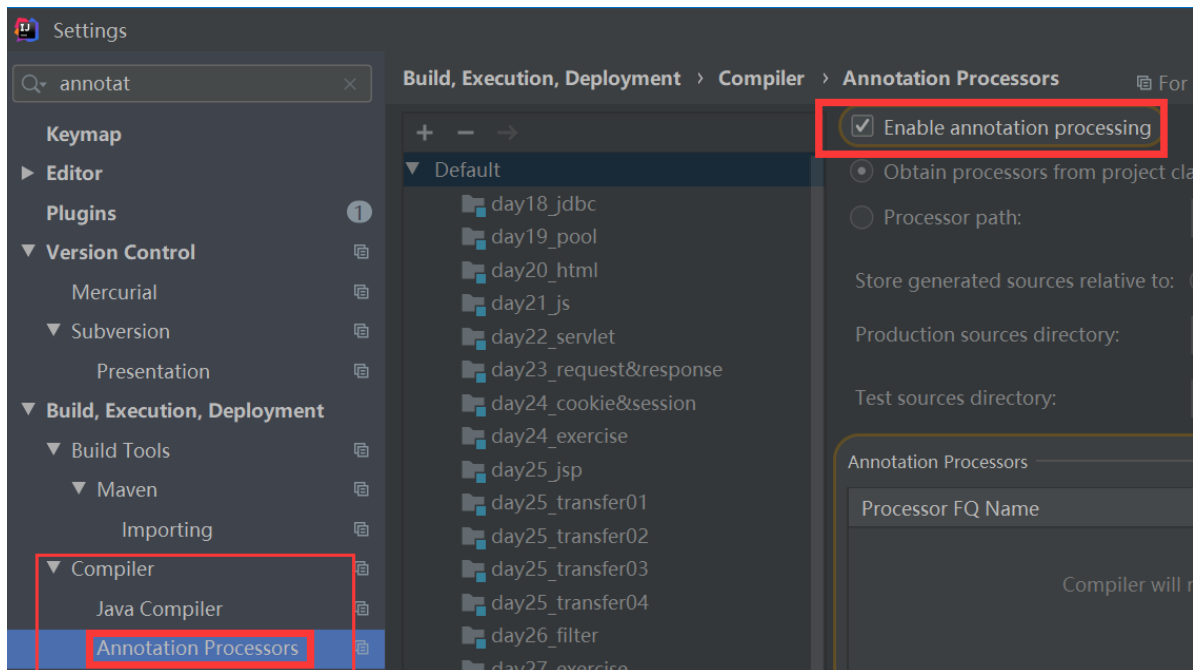
• 解决编译时出错问题

编译时出错，可能是没有enable注解处理器。Annotation Processors > Enable annotation processing。设置完成之后程序正常运行。



4.小结

1. lombok是一个构建工具，可以用来简化java开发
2. lombok配置
 - Maven项目引入lombok
 - idea集成lombok插件(开启注解)



知识点-Lombok的常用注解

1.目标

- ☐ 掌握Lombok的常用注解

2.路径

1. @Data
2. @Getter/@Setter
3. @ToString
4. @NoArgsConstructor, @AllArgsConstructor

3.讲解

3.1@Data

@Data注解在类上，会为类的所有属性自动生成setter/getter、equals、canEqual、hashCode、toString方法，==**如果为final属性，则不会为该属性生成setter方法。**==

```

@Data
public class User implements Serializable{
    private Integer id;
    private String username;
    private String password;
    private String address;
    private String nickname;
    private String gender;
    private String email;
    private String status;
}

```

3.2@Getter/@Setter

如果觉得@Data太过残暴不够精细，可以使用@Getter/@Setter注解，此注解在属性上，可以为相应的属性自动生成Getter/Setter方法。

```

public class User implements Serializable{
    @Setter
    @Getter
    private Integer id;
    private String username;
    private String password;
    private String address;
    private String nickname;
    private String gender;
    private String email;
    private String status;
}

```

3.3@ToString

类使用@ToString注解，Lombok会生成一个toString()方法，默认情况下，会输出类名、所有属性（会按照属性定义顺序），用逗号来分割。通过exclude属性指定忽略字段不输出，

```

@ToString(exclude = {"id"})
public class User implements Serializable{
    private Integer id;
    private String username;
    private String password;
    private String address;
    private String nickname;
    private String gender;
    private String email;
    private String status;
}

```

3.4@xxxConstructor

- @NoArgsConstructor: 无参构造器

```
@NoArgsConstructor
public class User implements Serializable{
    private Integer id;
    private String username;
    private String password;
    private String address;
    private String nickname;
    private String gender;
    private String email;
    private String status;
}
```

- @AllArgsConstructor: 全参构造器

```
@AllArgsConstructor
public class User implements Serializable{
    private Integer id;
    private String username;
    private String password;
    private String address;
    private String nickname;
    private String gender;
    private String email;
    private String status;
}
```

4.小结

4.1注解

- @Data
 - 用在==类上面==的, 生成set,get, toString, hashCode, canEqual、equal方法
- @Getter
 - 用在==字段或类上==, 生成get方法
- @Setter
 - 用在==字段或类==上, 生成set方法
- @ToString
 - 用在==类上面==的 生成toString方法, 可以使用exclude排除不需要输出的属性
- @xxxConstructor @AllArgsConstructor和@NoArgsConstructor一般都会成对出现
 - 用在==类上面==的 生成构造方法 (只能生成无参和全参的构造方法)

4.2优缺点

优点:

1. 能通过注解的形式自动生成构造器、getter/setter、equals、hashCode、toString等方法, 提高了一定的开发效率
2. 让代码变得简洁, 不用过多的去关注相应的方法
3. 属性做修改时, 也简化了维护为这些属性所生成的getter/setter方法等

缺点:

1. 不支持多种参数构造器的重载
2. 虽然省去了手动创建getter/setter方法的麻烦，但大大降低了源代码的可读性和完整性，降低了阅读

本章小结

1. Maven相关概念

- Maven概念：是由apache提供的一个项目构建工具。
- Maven作用：①依赖管理(jar包管理) ②项目构建 ③分模块开发(Maven高级会讲)
- Maven仓库：存储jar包的地方
 - 本地仓库
 - 远程仓库(私服)
 - 中央仓库
- Maven坐标：jar包的唯一标识，通过坐标去Maven仓库中找到jar包

2. ==Maven的安装 【重点】==

- Maven的本地安装
 - 解压到无中文空格的目录下
 - 配置环境变量
 - MAVEN_HOME: D:\ProgramFiles\apache-maven-3.5.3
 - Path: %MAVEN_HOME%\bin
 - 配置本地仓库 (settings.xml) D:/Maven/repository
- idea集成Maven
 - 当前项目Maven环境配置： file->settings->maven->配置MavenHome、Maven配置文件、Maven仓库
 - 新项目Maven环境配置【默认】： new Projects settings->setting for new Projects-->maven-->...

3. ==使用idea创建Maven工程项目【重点】==

- 使用模板创建
- 不使用模板创建
- 注意：一般idea创建出的Maven工程项目目录不完整，需要自己手动补全。

4. ==Maven常用命令【重点】==

- mvn clean：清除编译后的内容 也就是清空target目录
- mvn compile：重新编译项目
- mvn test：执行src/java下的所有单元测试用例
- mvn package：打包 javase-->jar包 javaweb-->war包
- mvn install：将本地项目安装到本地仓库 方便自己在其他项目中使用
- mvn deploy：将本地项目发布到私服【远程仓库】

5. 依赖管理和插件

- 依赖管理--jar包的依赖范围
 - compile：表示该jar包在编译、测试、运行都需要使用 eg: druid fastjson
 - test： 表示该jar包只在测试时需要使用 eg: junit
 - provided：表示该jar包在编译和测试时需要使用 eg: servlet-api

- runtime: 表示该jar包在测试和运行时需要使用 eg: JDBC驱动
- 插件:
 - JDK编译版本插件 【Maven项目默认使用JDK1.5进行编译, 可以设置使用JDK1.8】
 - tomcat插件 【方便javaweb项目直接启动】

6. Maven私服【了解 对着文档配置 配置完就删除! 】

7. Lombok

- 介绍和配置
 - idea集成lombok插件并开启注解
 - Maven项目添加lombok依赖
- 常用注解
 - @Data
 - @Getter/@Setter
 - @ToString
 - @NoArgsConstructor @AllArgsConstructor