

day39-maven

学习目标:

1. 能够了解Maven的作用
2. 能够理解Maven仓库的作用
3. 能够理解Maven的坐标概念
4. 能够掌握Maven的安装
5. 能够掌握IDEA配置本地Maven
6. 能够使用IDEA创建javase的Maven工程
7. 能够使用IDEA创建javaweb的Maven工程
8. 能够掌握依赖引入的配置方式
9. 能够了解依赖范围的概念

一 ,Maven概述

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。

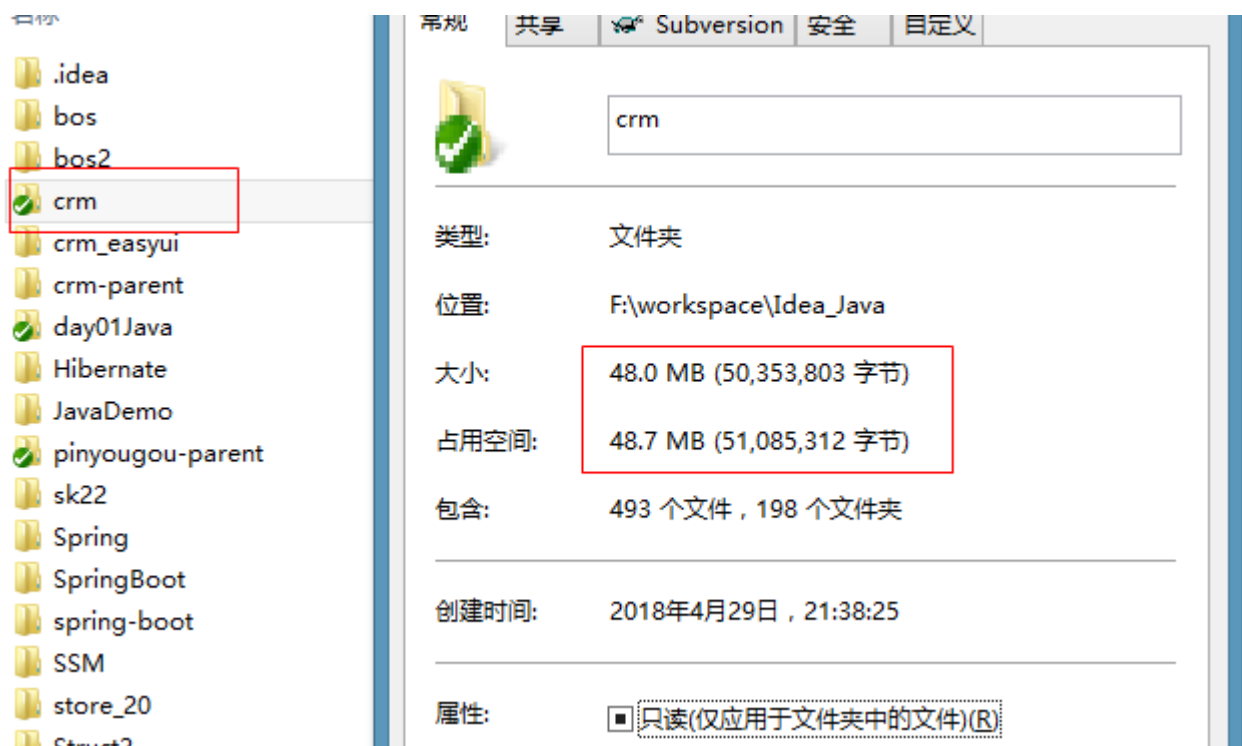
说白了: Maven是由Apache开发的一个工具。用来管理java项目，依赖管理(jar包的管理)，项目构建。

2.Maven的作用

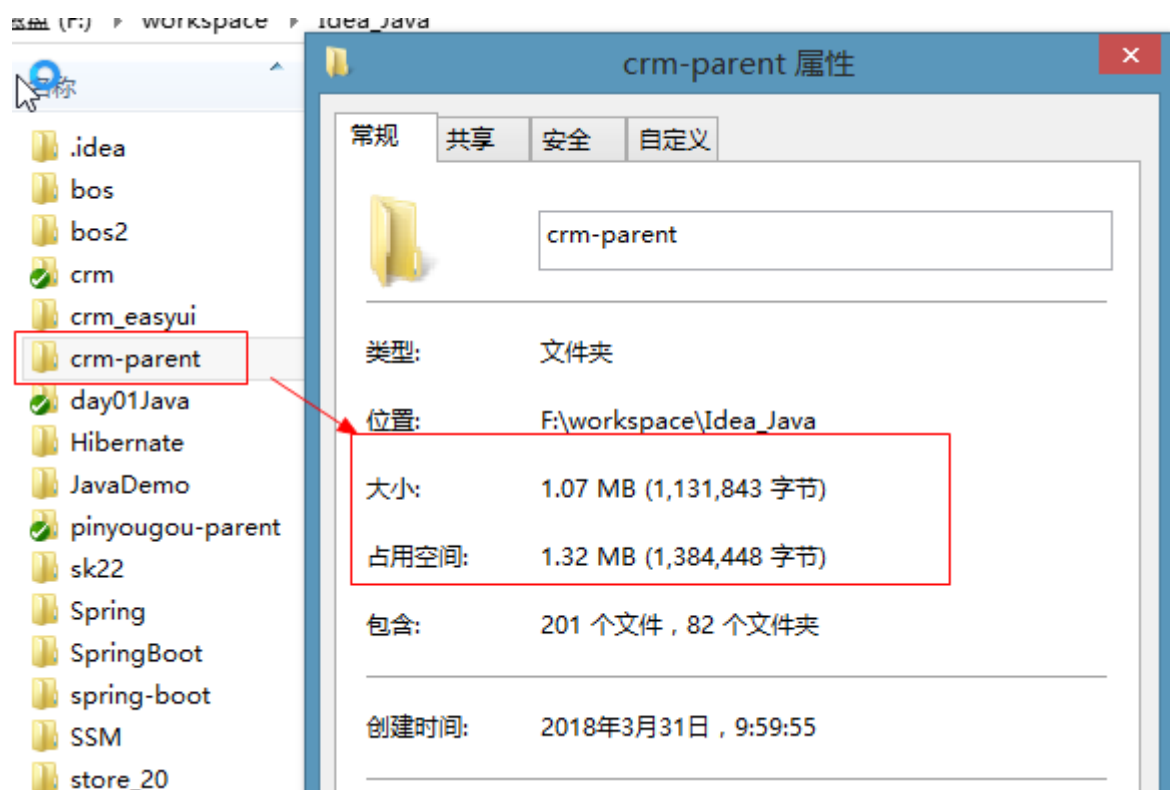
- 依赖管理: maven对项目的第三方构件（jar包）进行统一管理。向工程中加入jar包不要手工从其它地方拷贝，通过maven定义jar包的坐标，自动从maven仓库中去下载到工程中。
- 项目构建: maven提供一套对项目生命周期管理的标准，开发人员、和测试人员统一使用maven进行项目构建。项目生命周期管理：编译、测试、打包、部署、运行。
- maven对工程分模块构建，提高开发效率。

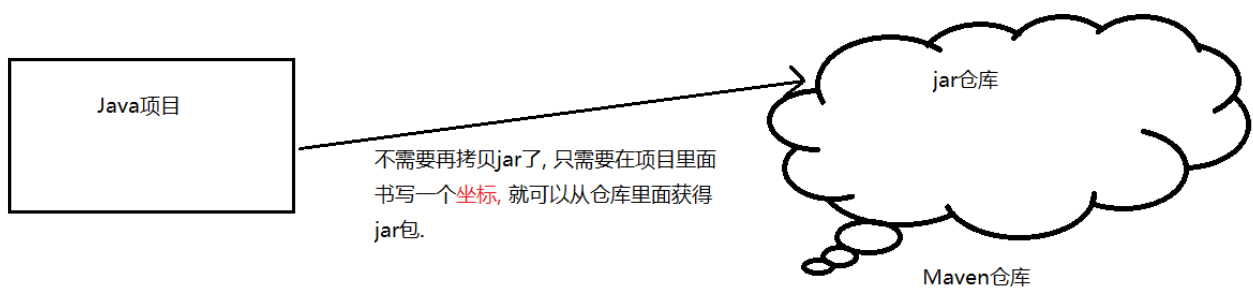
3 Maven的好处

- 使用普通方式构建项目



- 使用Maven构建项目





4. Maven的仓库

仓库名称	作用
本地仓库	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	就是远程仓库, 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, 对外公开。

5. 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junit的坐标即可:

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

二, Maven的安装

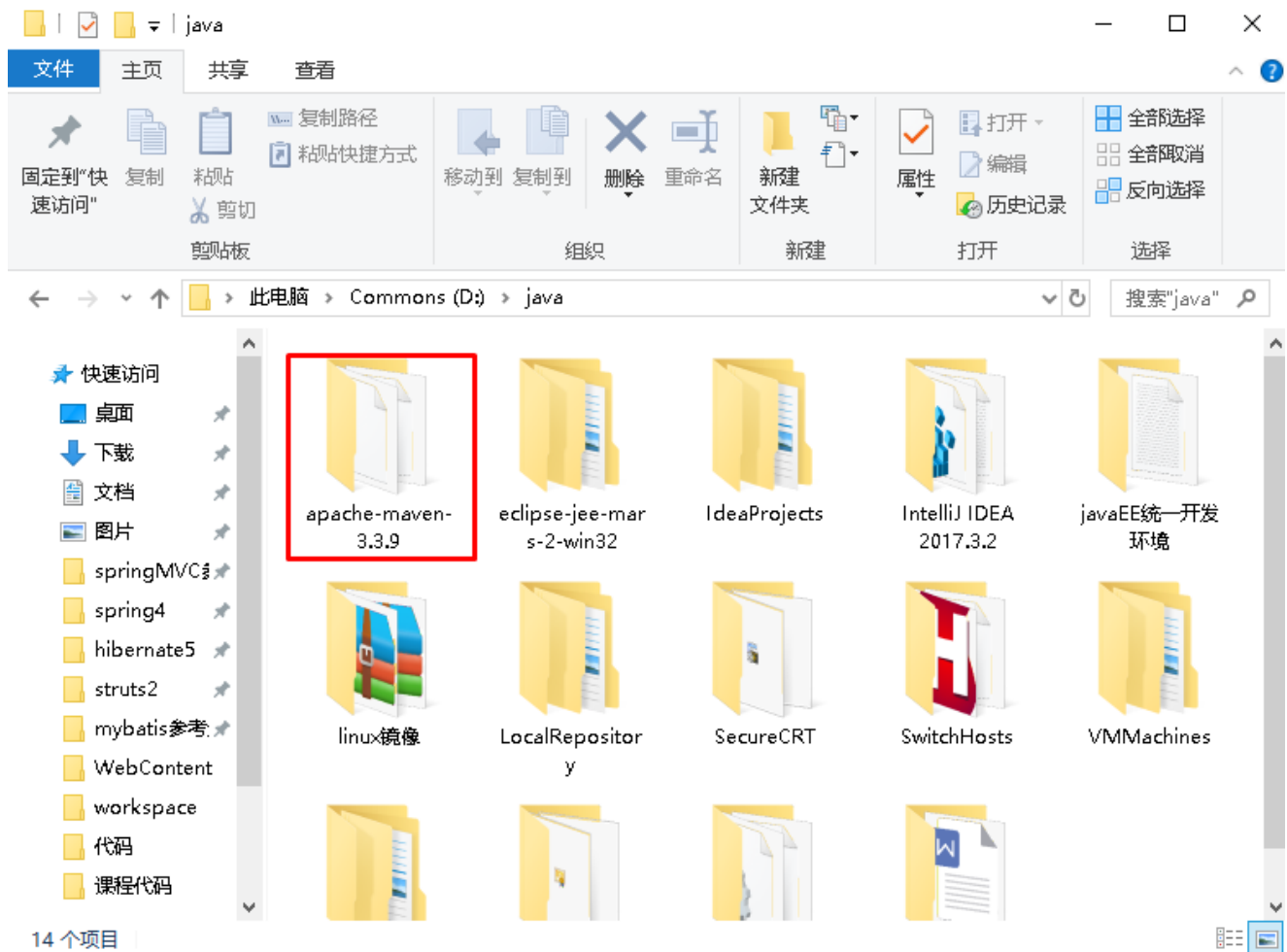
1. 下载Maven



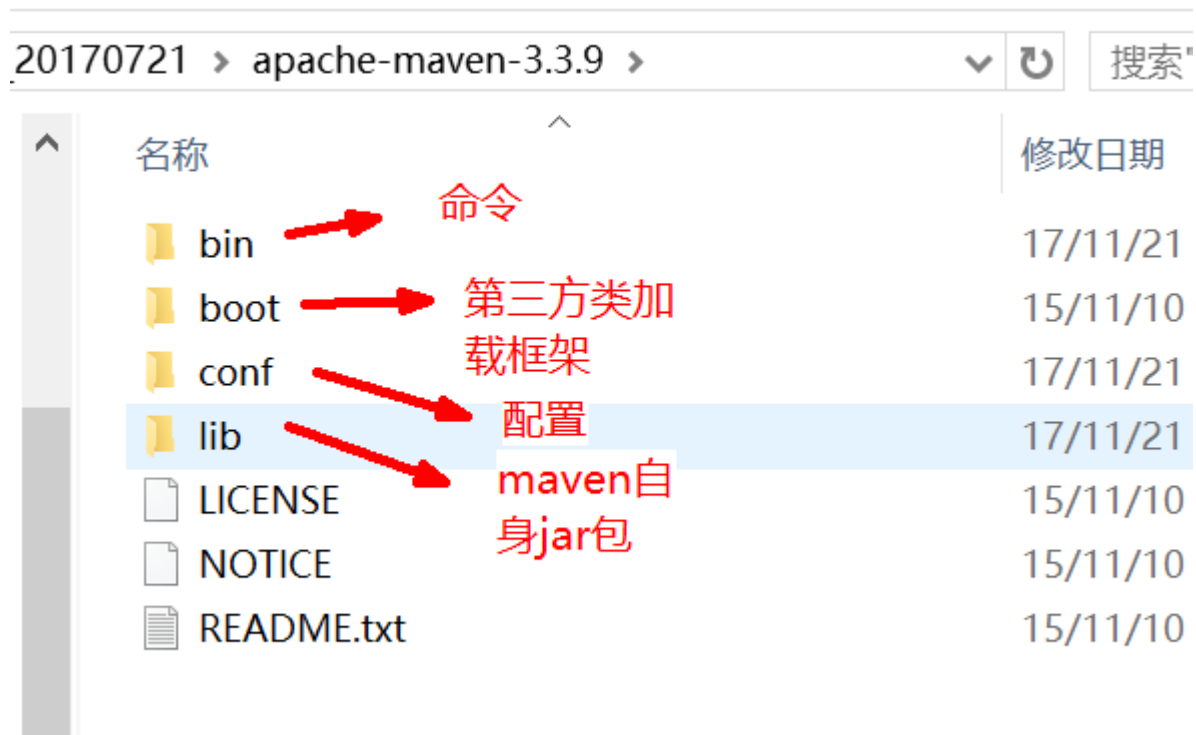
apache-maven-3.3.9-bin.zip

2. 安装Maven

将Maven压缩包解压，即安装完毕



3 Maven目录介绍



4 配置环境变量

- 进入环境变量



- 配置MAVEN_HOME和Path

编辑系统变量

变量名 (N): MAVEN_HOME
变量值 (V): %userprofile%\maven\maven_local\apache-maven-3.3.9

maven程序解压目录

确定

取消

系统变量 (S)

变量	值
MAVEN_HOME	H:\maven\maven_local\apache-mav...
NLS_LANG	AMERICAN_AMERICA.ZHS16GBK
NUMBER_OF_PR...	8
OS	Windows NT

新建 (W)...

编辑 (E)...

删除 (L)

确定

取消

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>
```

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>
```

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。

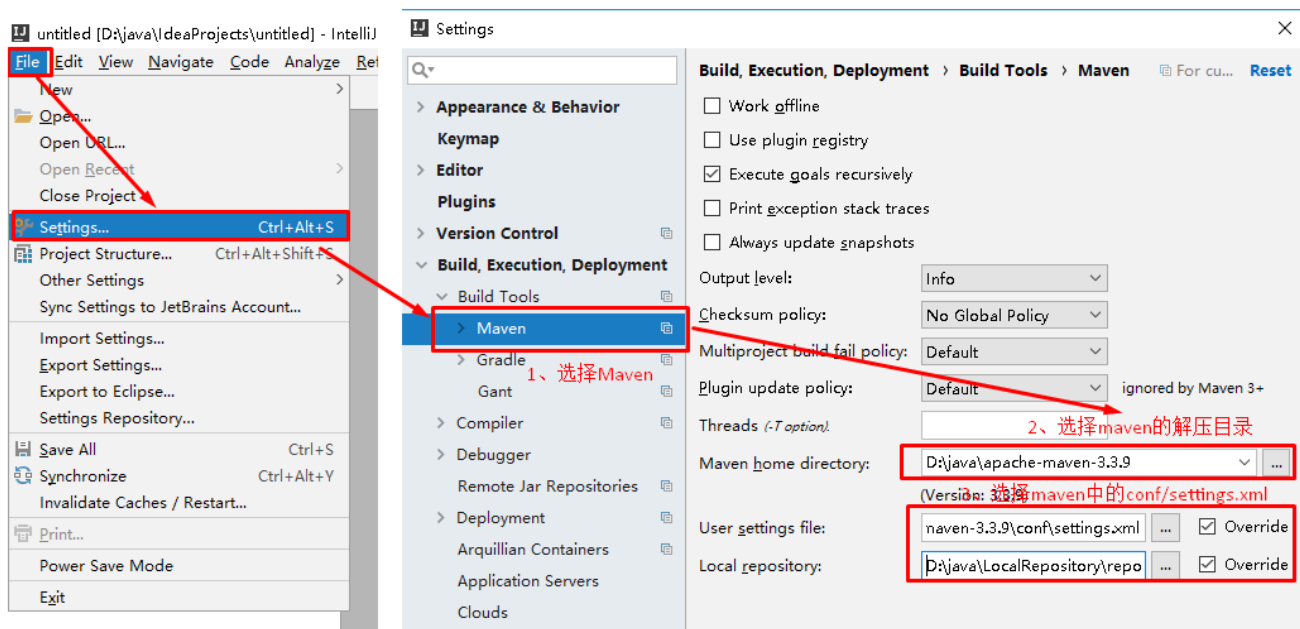
C:\Users\muzimoo>mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T00:41:47+08:00)
Maven home: D:\java\apache-maven-3.3.9\bin\..
Java version: 1.7.0_72, vendor: Oracle Corporation
Java home: C:\Program Files (x86)\Java\jdk1.7.0_72\jre
Default locale: zh_CN, platform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

C:\Users\muzimoo>
```

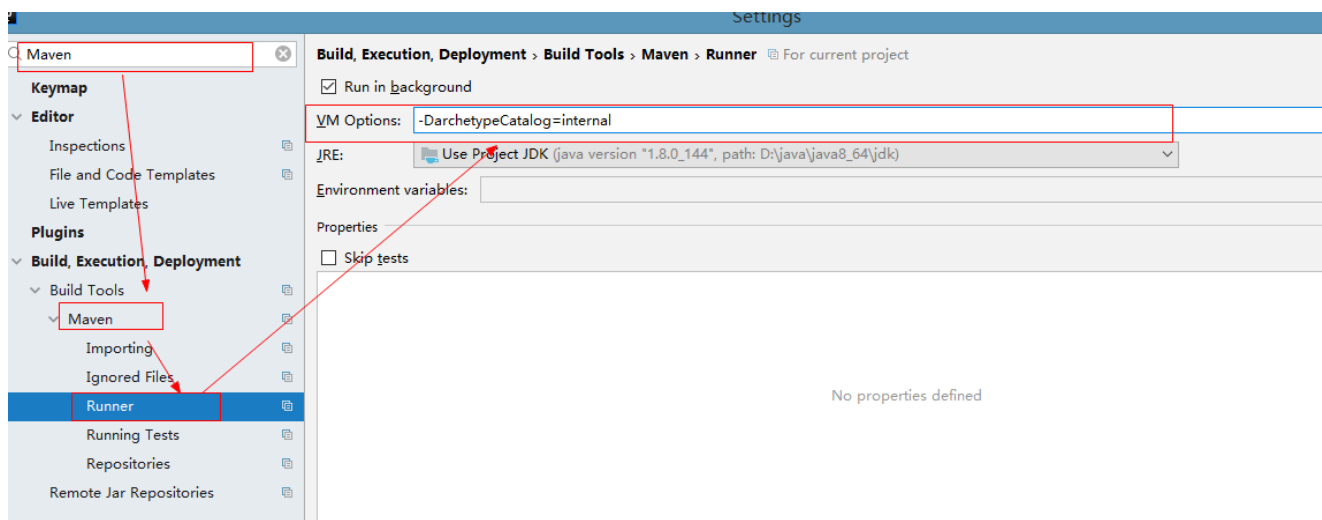
三, IDEA创建Maven工程

1. IDEA指定本地Maven

- 配置Maven

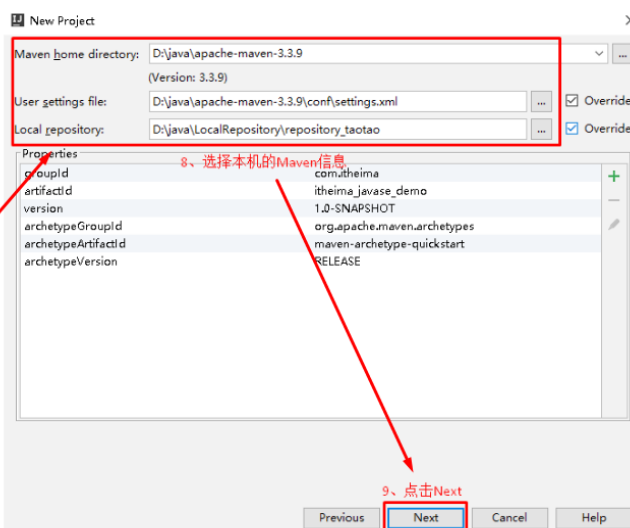
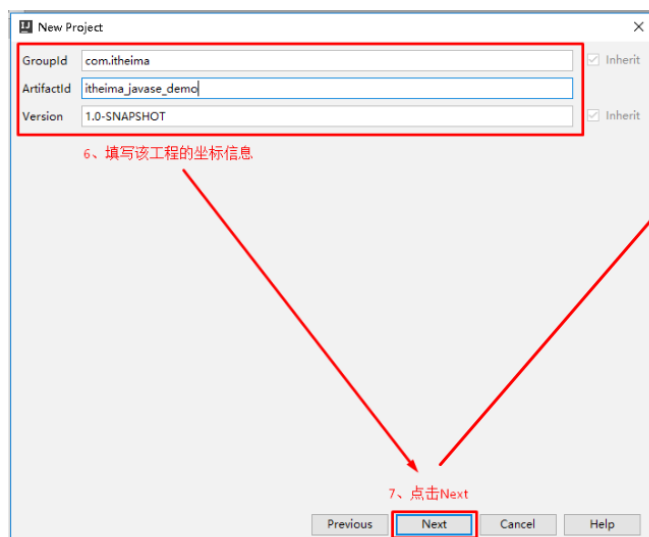
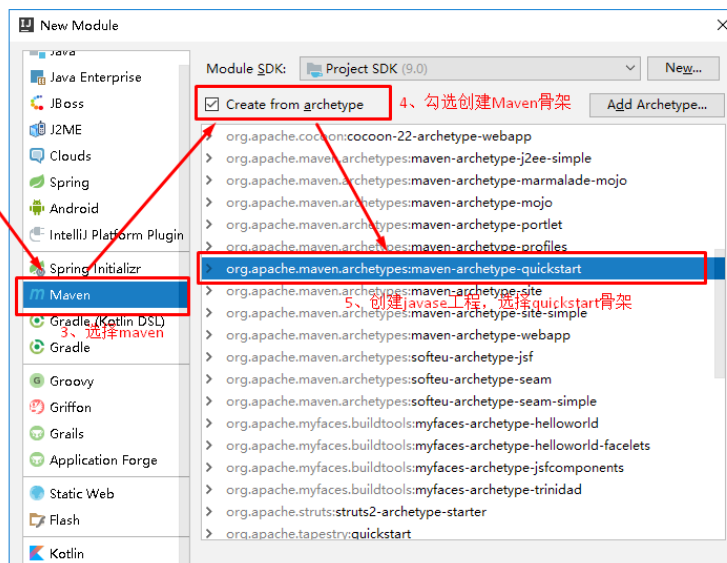
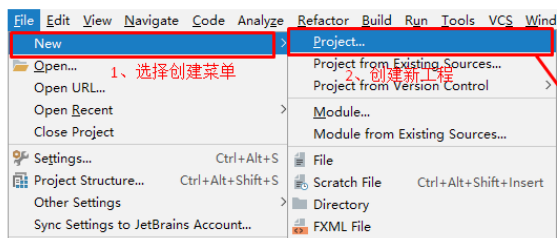


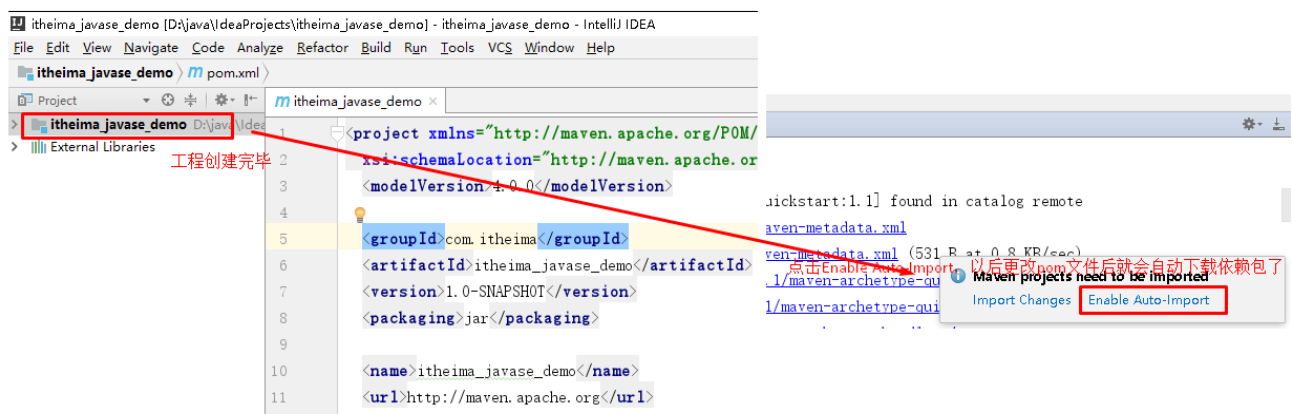
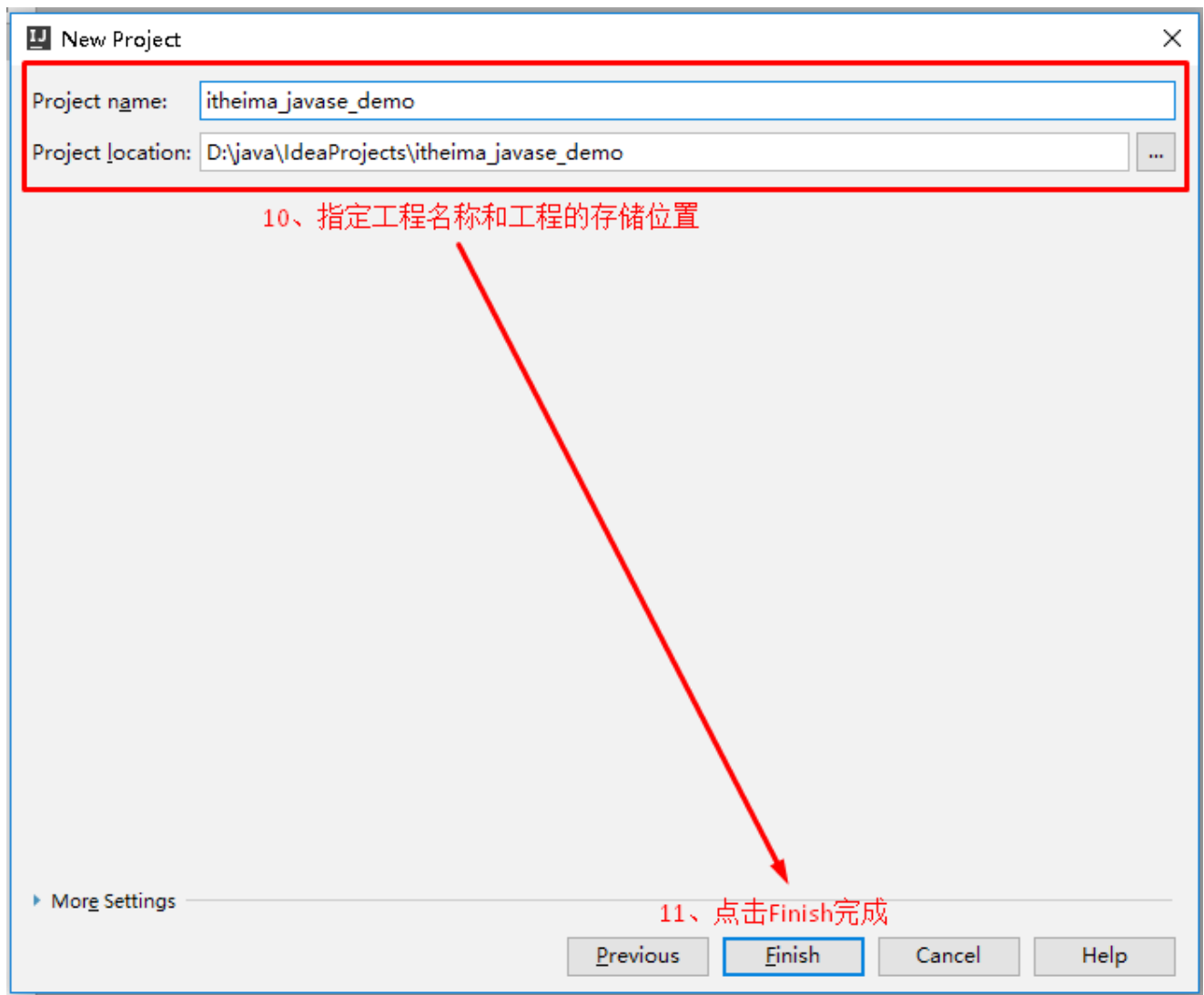
- 配置参数(创建工程不需要联网,解决创建慢的问题) -DarchetypeCatalog=internal



2 创建javase工程

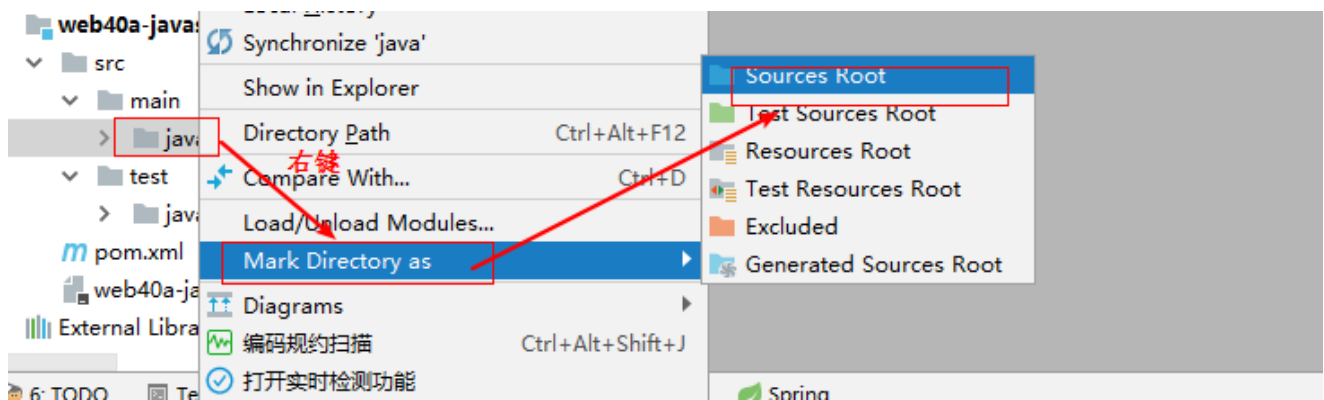
2.1 创建java工程



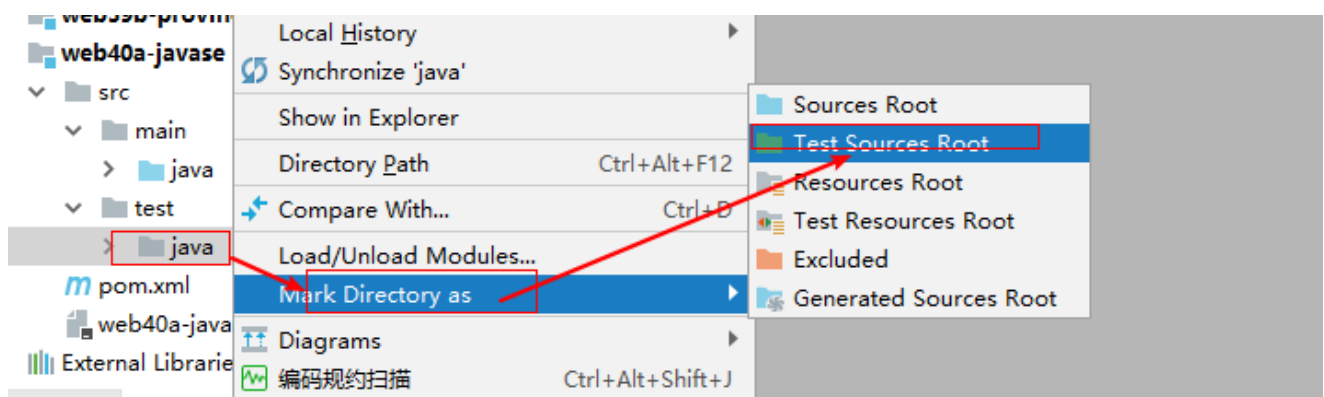


2,2 java工程目录结构

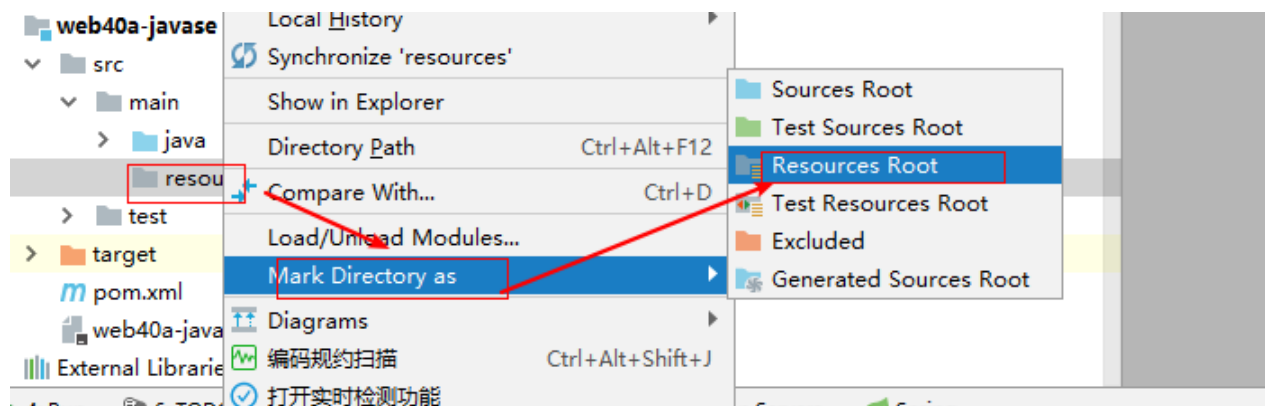
- 需要main/java文件夹变成 源码的目录(存放java源码)



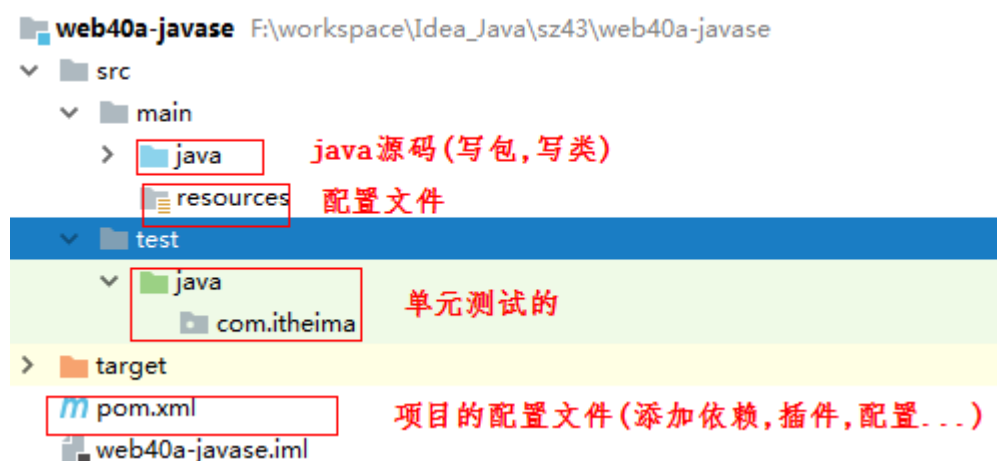
- 需要test/java文件夹变成 测试源码的目录(存放单元测试)



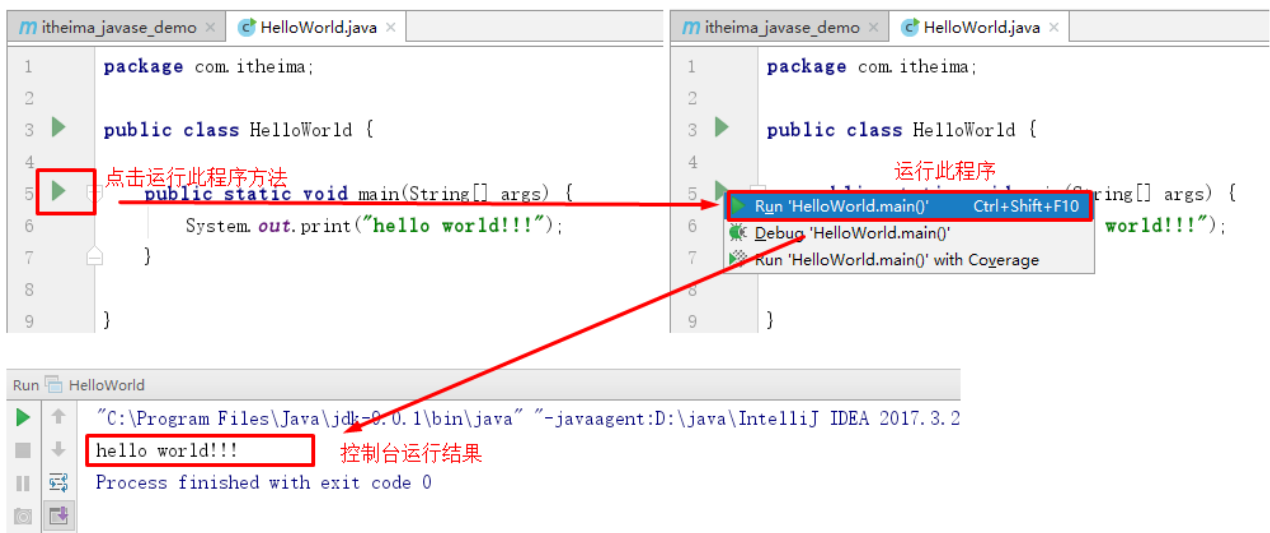
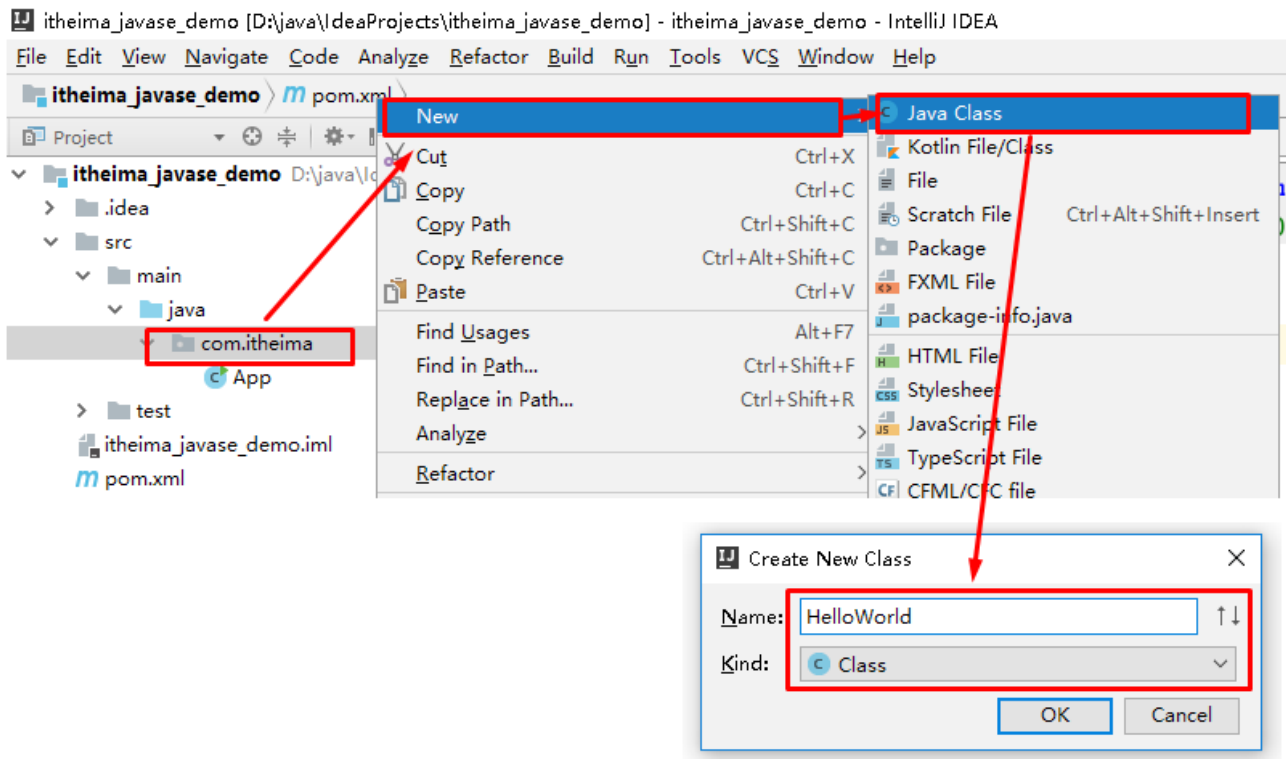
- 创建resources目录, 变成资源的目录



- 整体结构



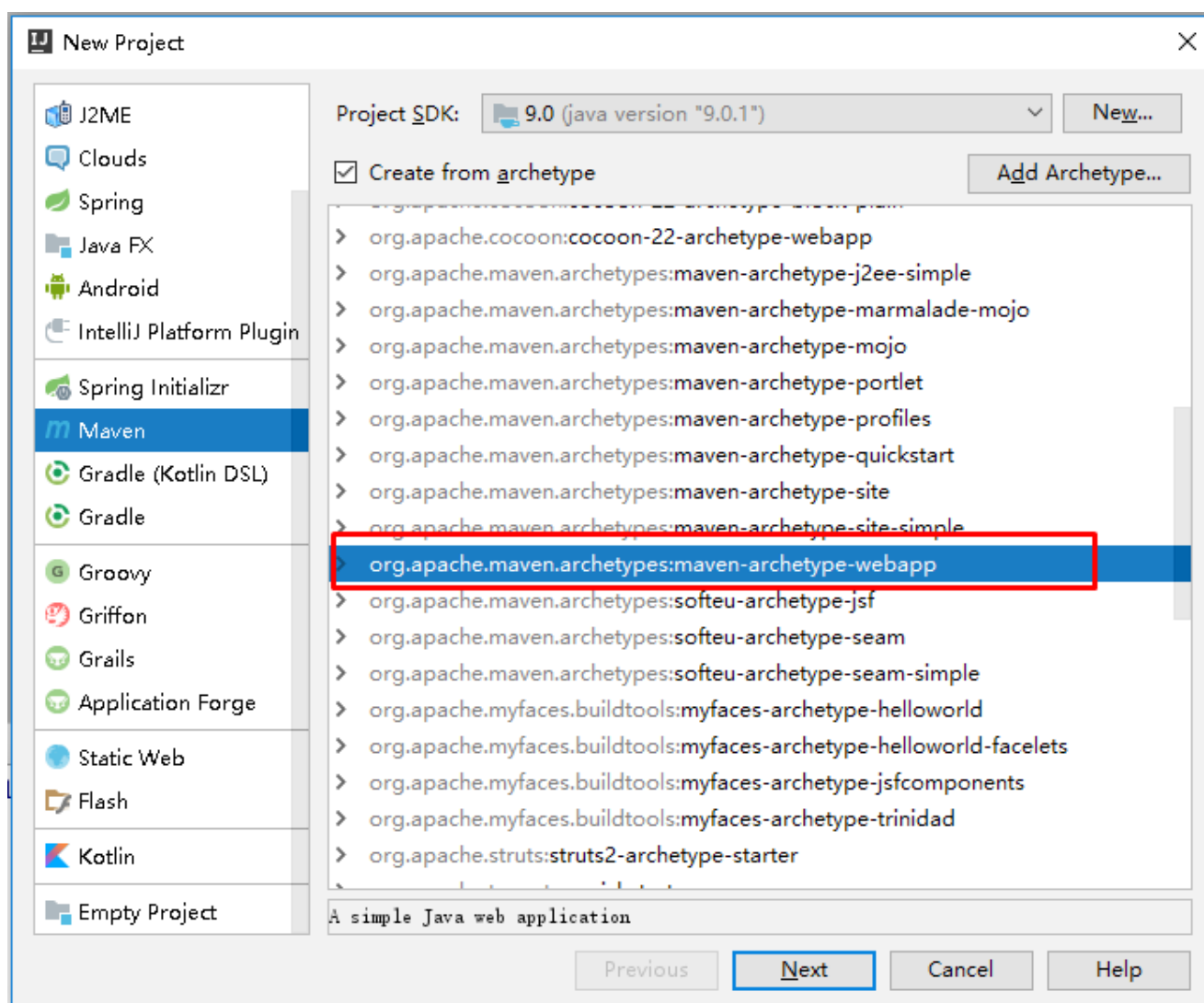
2.3 编写Hello World!



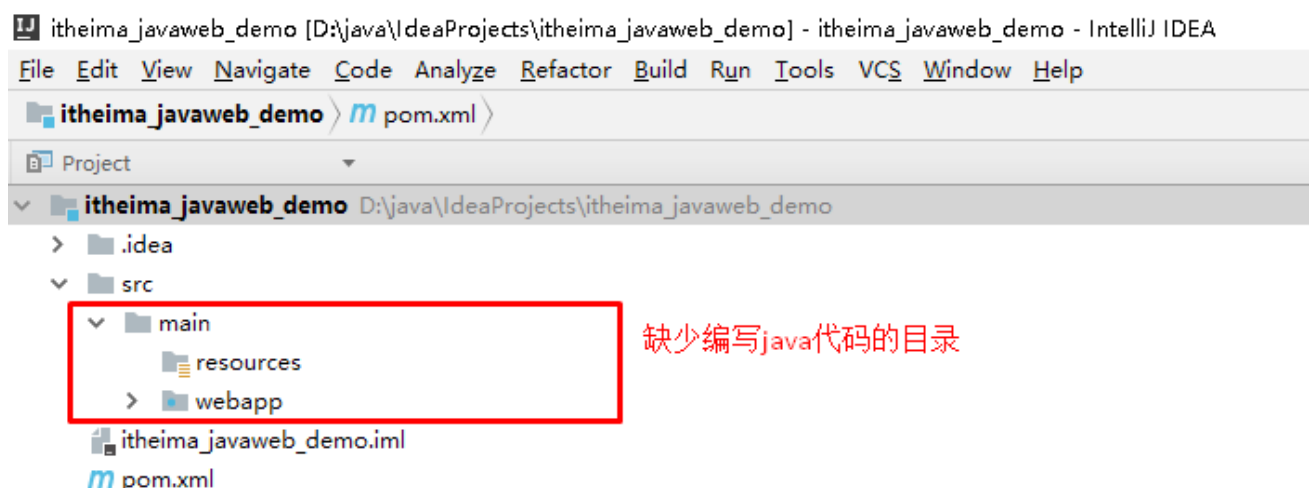
3 创建javaweb工程

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：

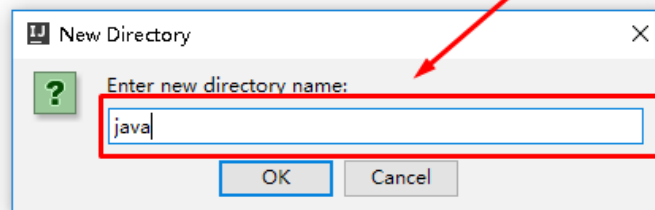
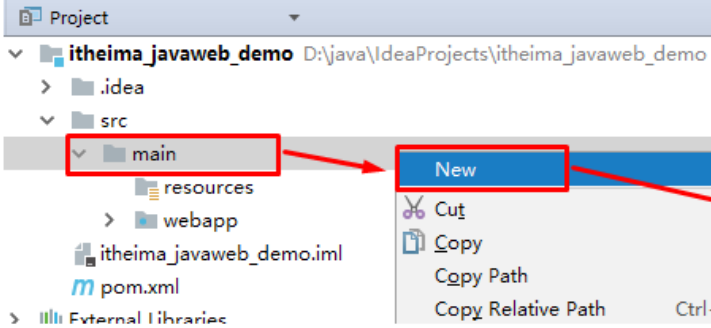


- 所以，要手动创建一个java目录用于编写java代码：

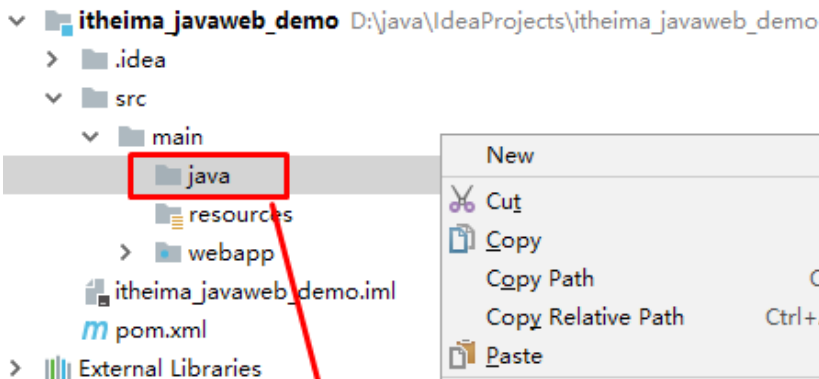
itheima_javaweb_demo [D:\java\IdeaProjects\itheima_javaweb_demo] - itheima_javaweb_demo - IntelliJ IDEA

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

itheima_javaweb_demo > src > main >



- 还要将java目录添加为Source Root:



Messages Maven Goal

[INFO] BUILD SUCCESS

[INFO] -----

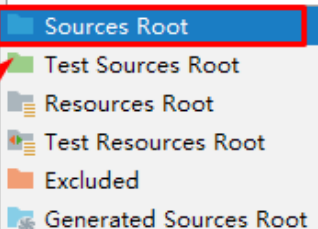
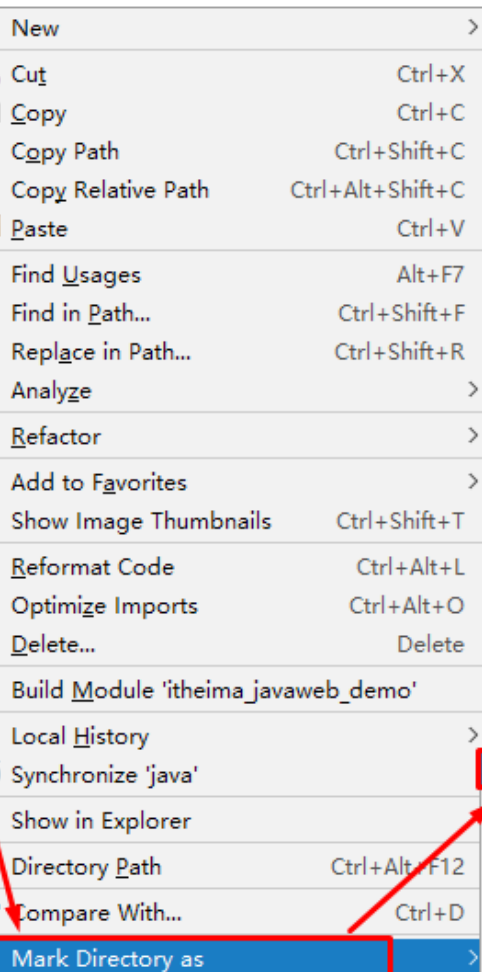
[INFO] Total time: 01:05 min

[INFO] Finished at: 2018-02-

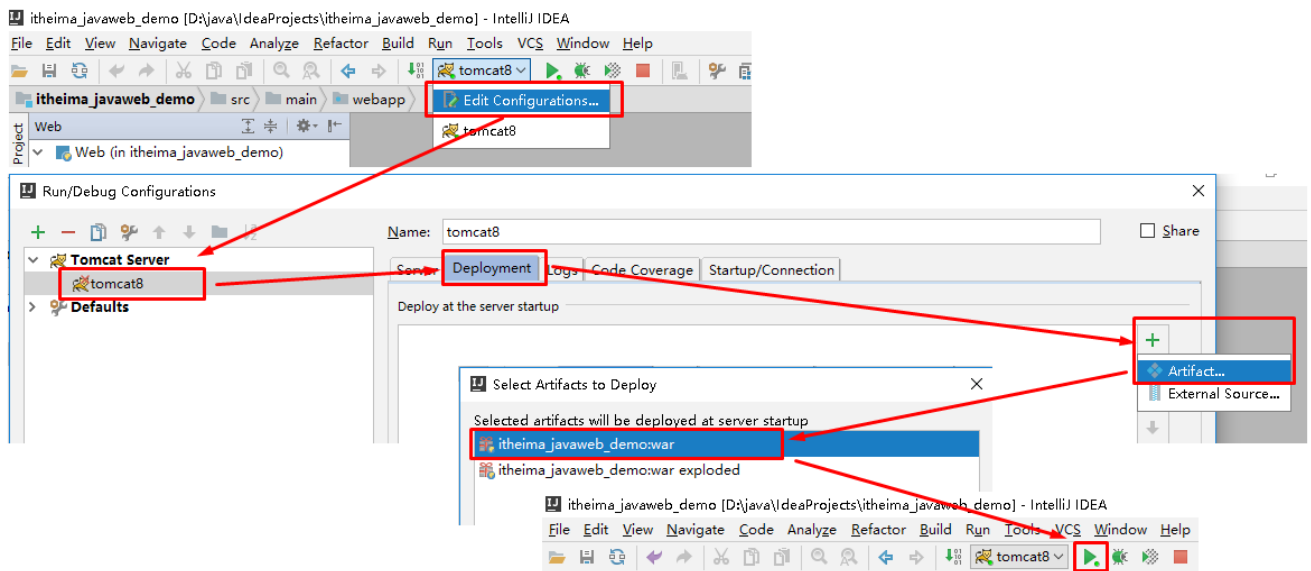
[INFO] Final Memory: 14M/47M

[INFO] -----

[INFO] Maven execution finis



3.2 发布javaweb工程



3.3 浏览器访问效果



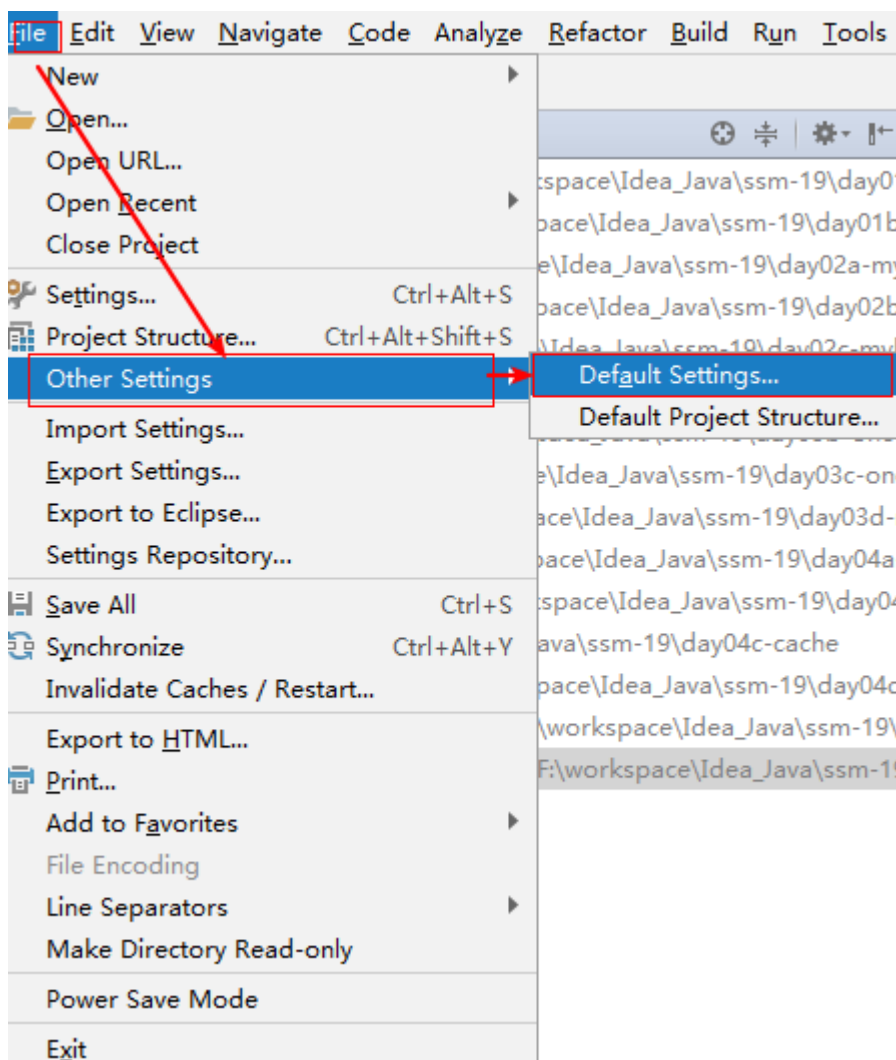
3.4 Maven创建javaweb工程的目录结构



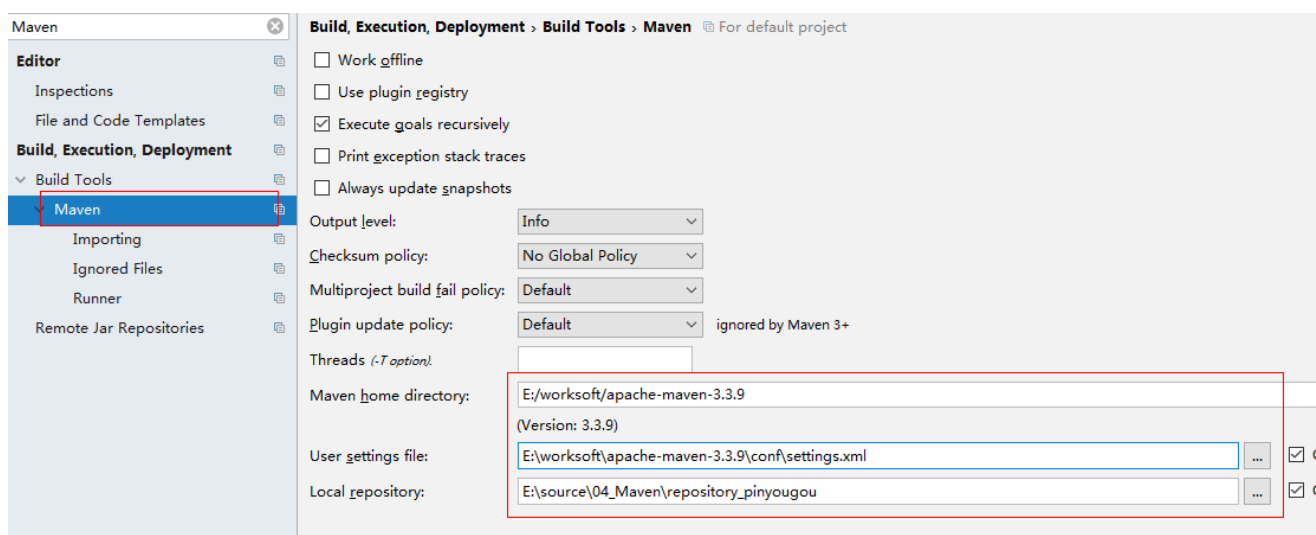
4. 配置默认Maven环境

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：

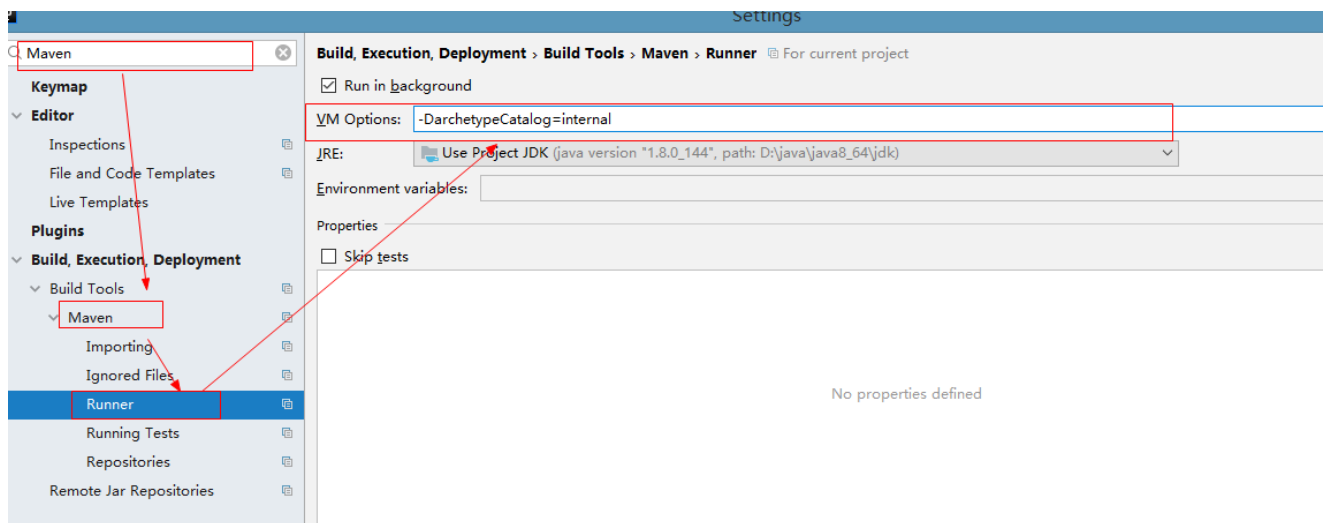
- 选择默认的配置



- 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

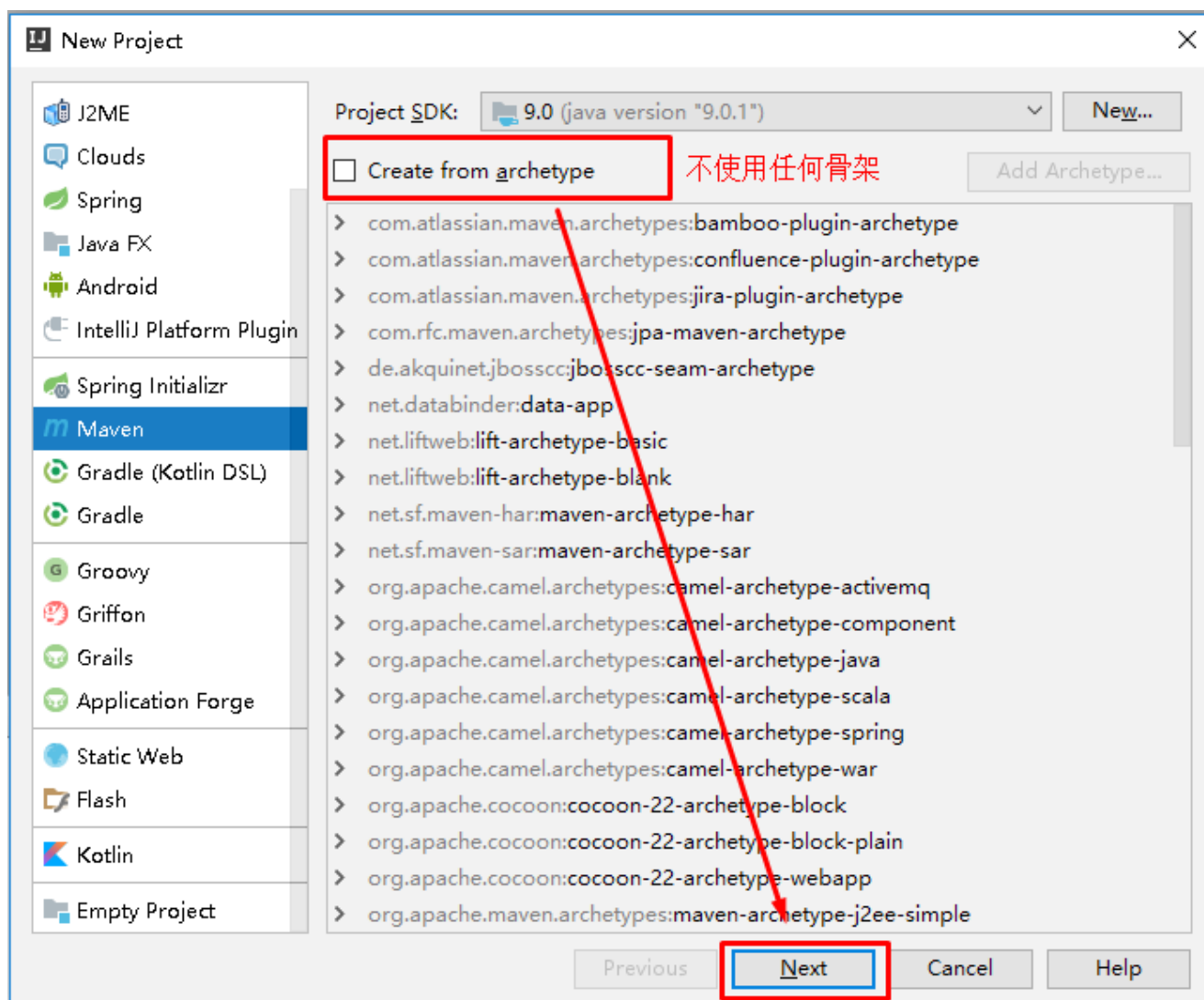


- 重启IDEA, 就可以生效了

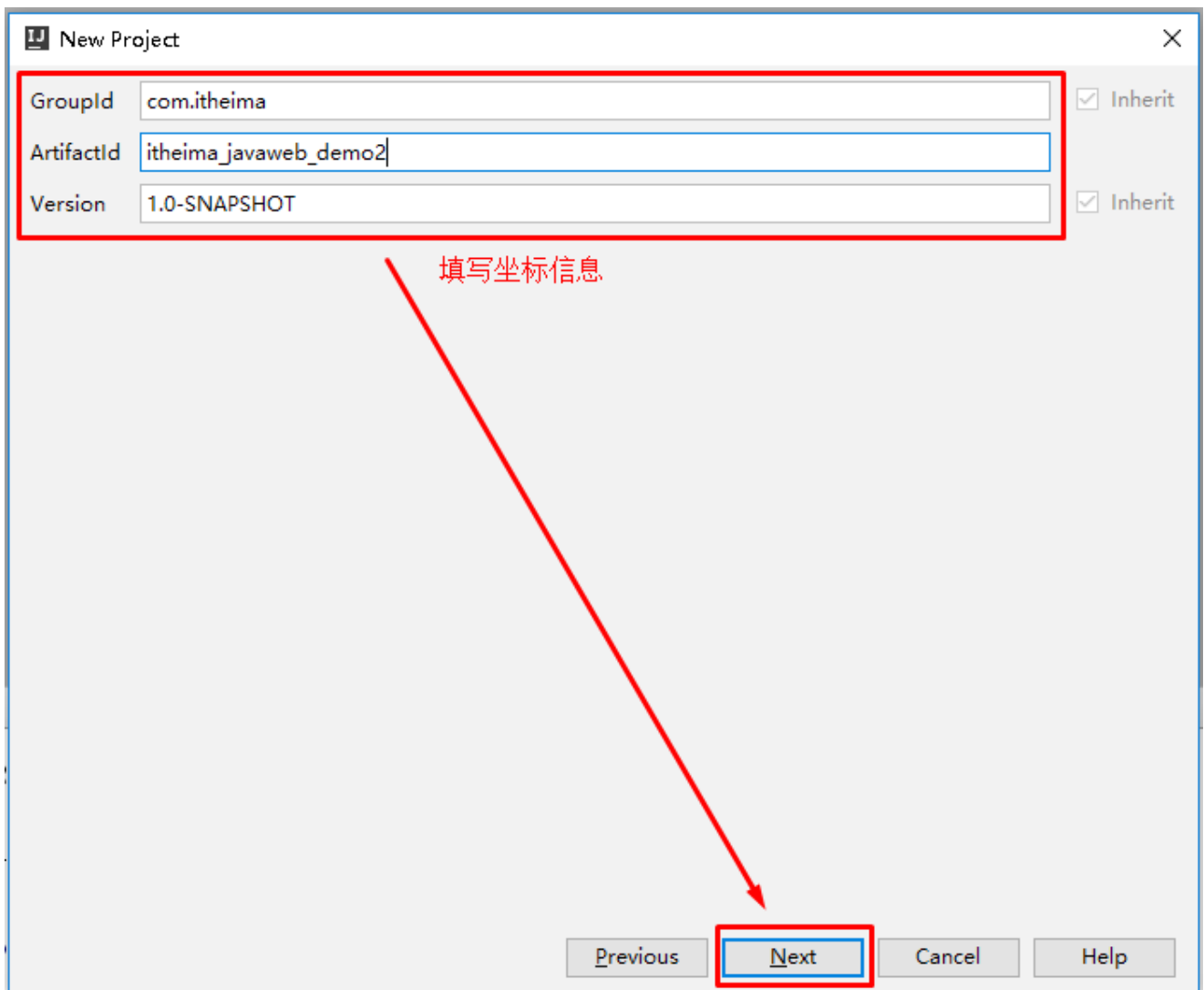
四 创建自定义JavaWeb工程

在三中，在创建javaweb工程时，使用的是maven-archetype-webapp骨架，如果不使用骨架，怎样创建一个javaweb工程呢，见下面的讲解：

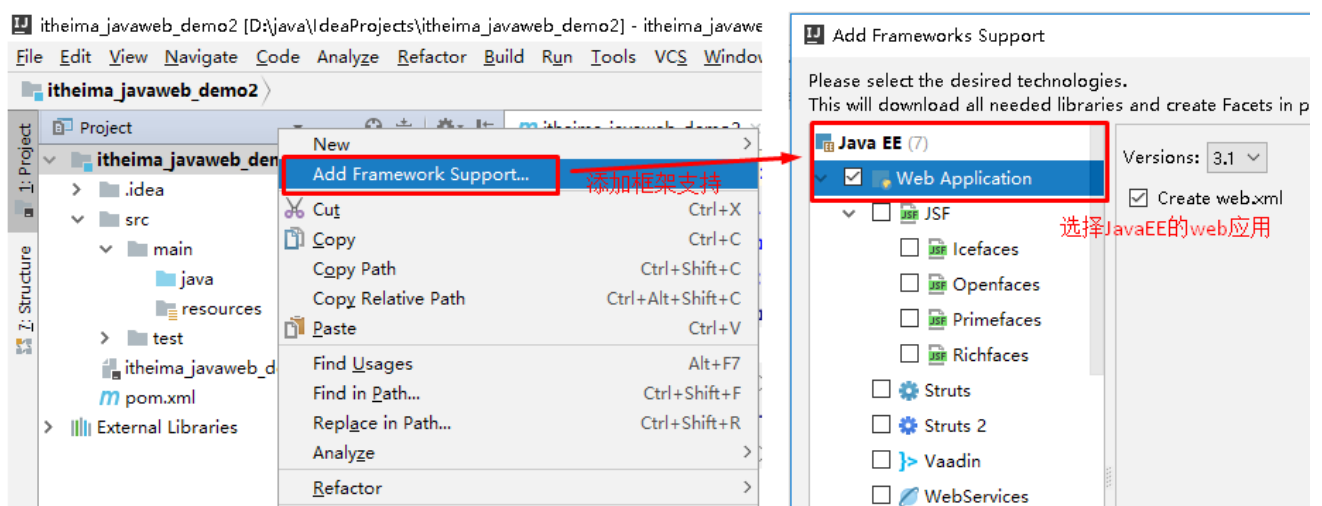
- 创建一个Maven工程，不选择任何骨架



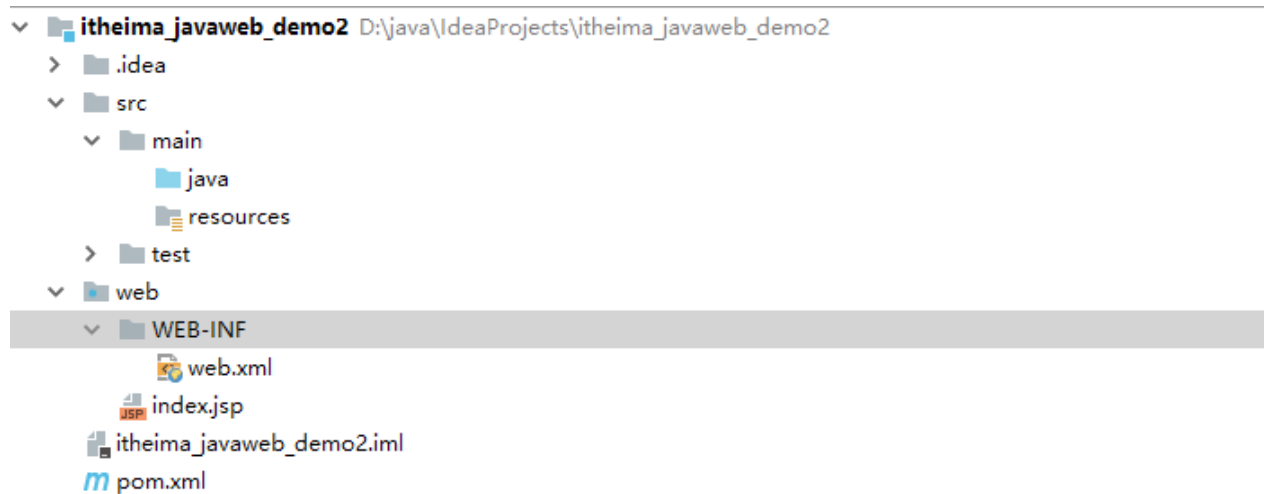
- 填写坐标信息



- 创建web工程信息



- 创建好的javaweb工程的目录结构如下

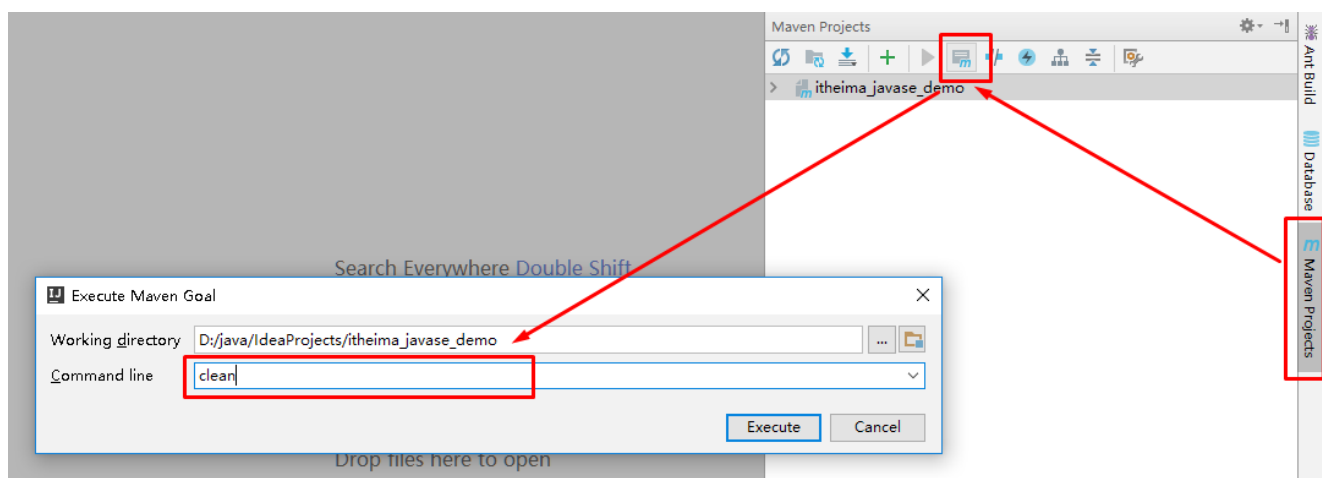


五, Maven的常用命令

1 clean命令

清除编译产生的target文件夹内容，可以配合相应命令一起使用，如mvn clean package, mvn clean test

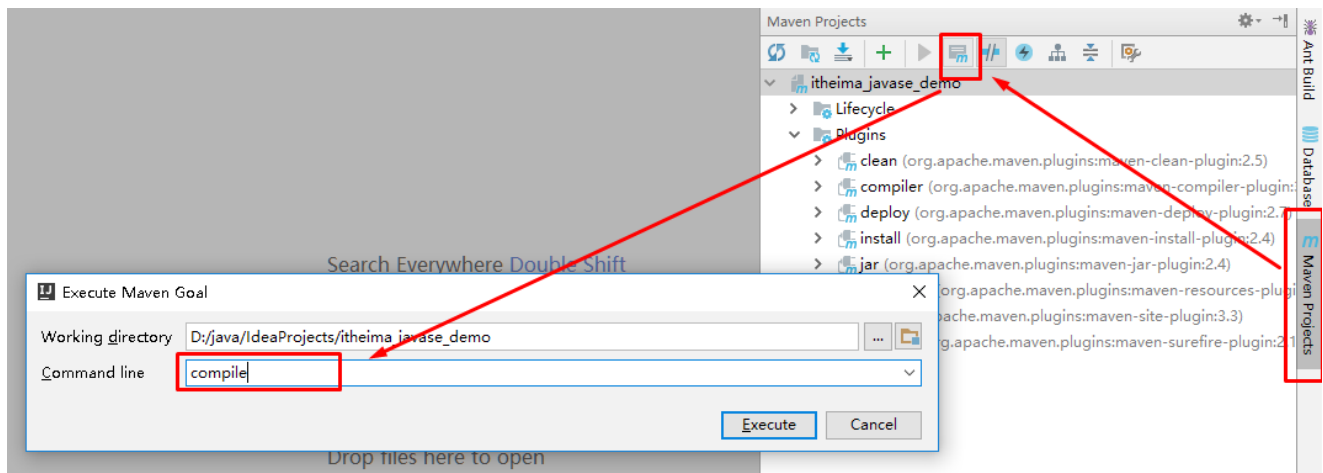
```
F:\workspace\Idea_Java\sz43\web40d-javaweb>mvn clean
```



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
```

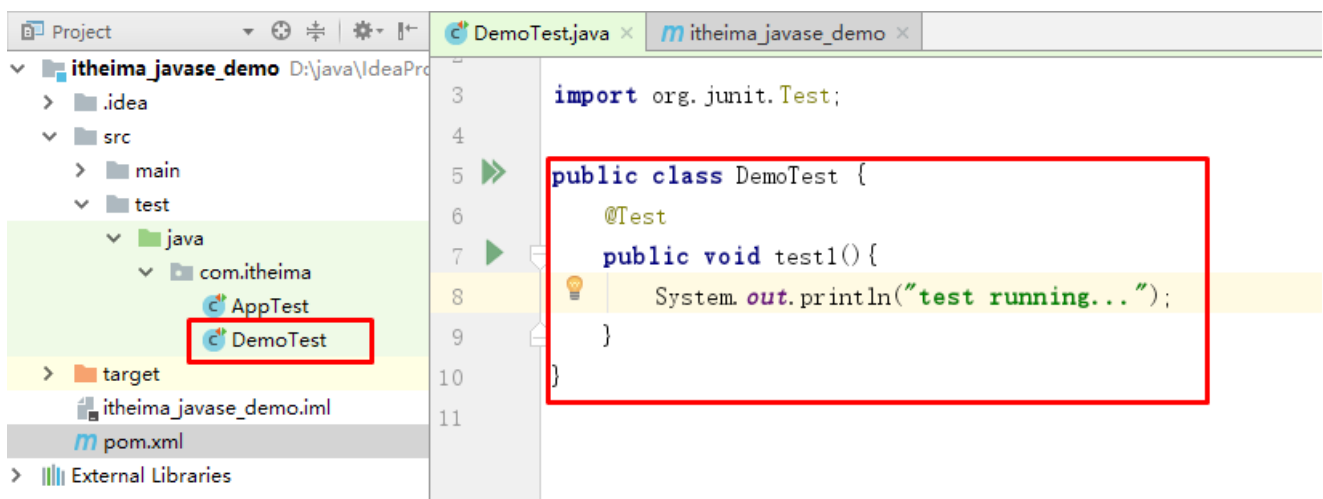


3 test命令

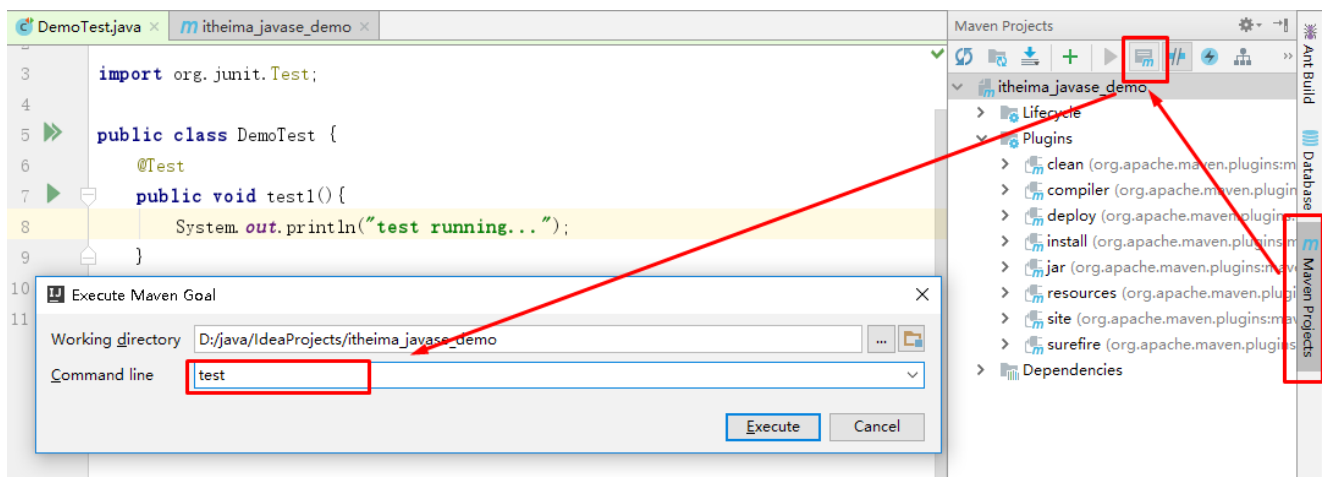
测试命令,或执行src/test/java/下junit的测试用例

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

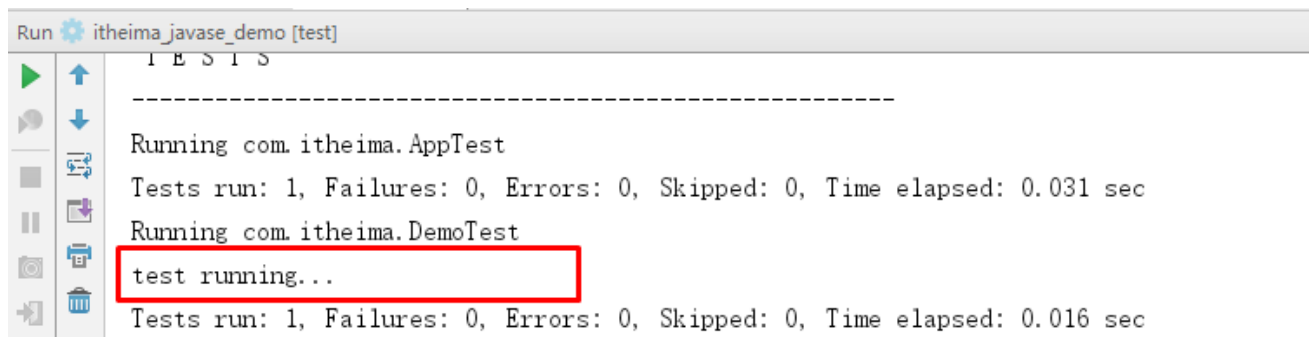
- 在src/test/java下创建测试类DemoTest



- 执行test命令测试



- 控制台显示测试结果

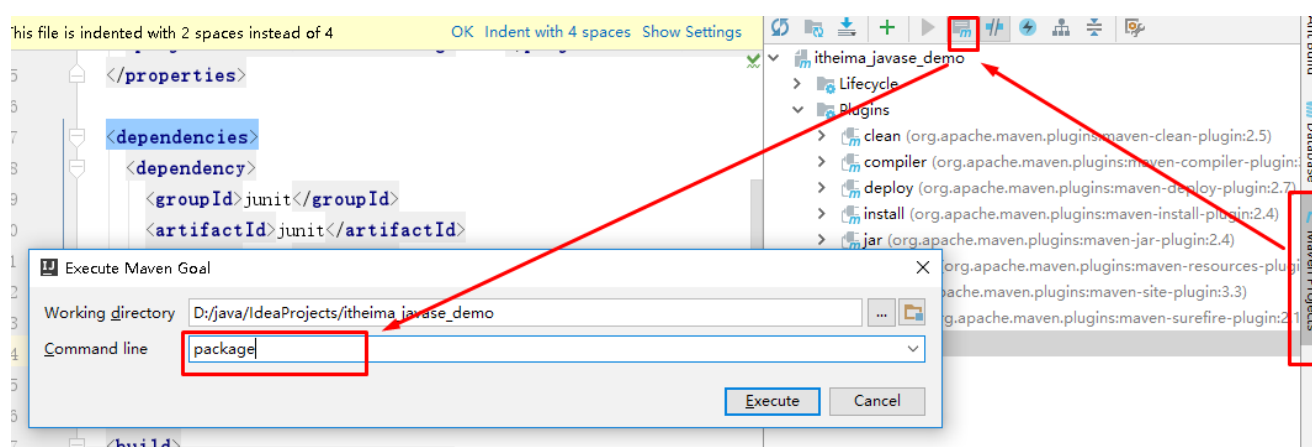


4 package命令

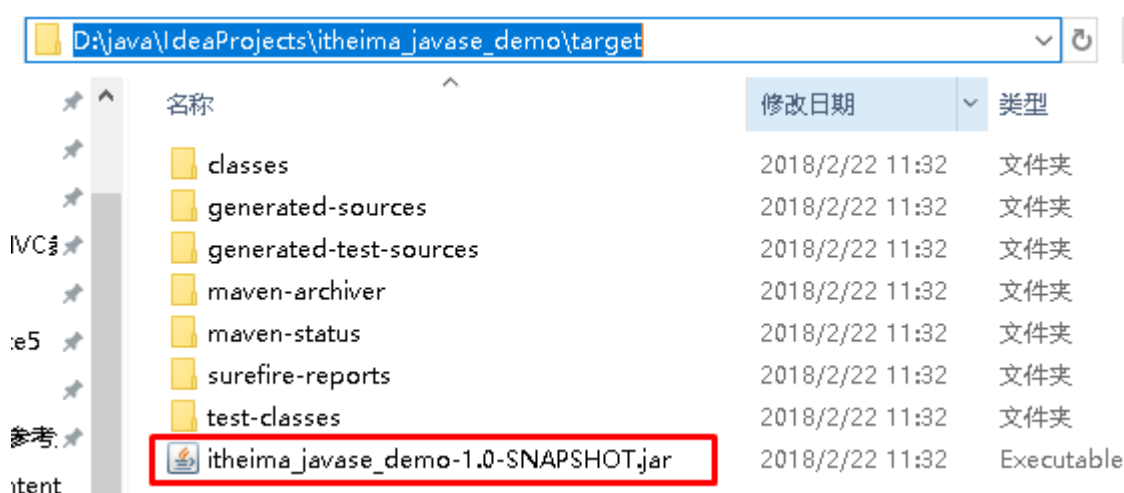
mvn package, 打包项目

- 如果是JavaSe的项目,打包成jar包
- 如果是JavaWeb的项目,打包成war包

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



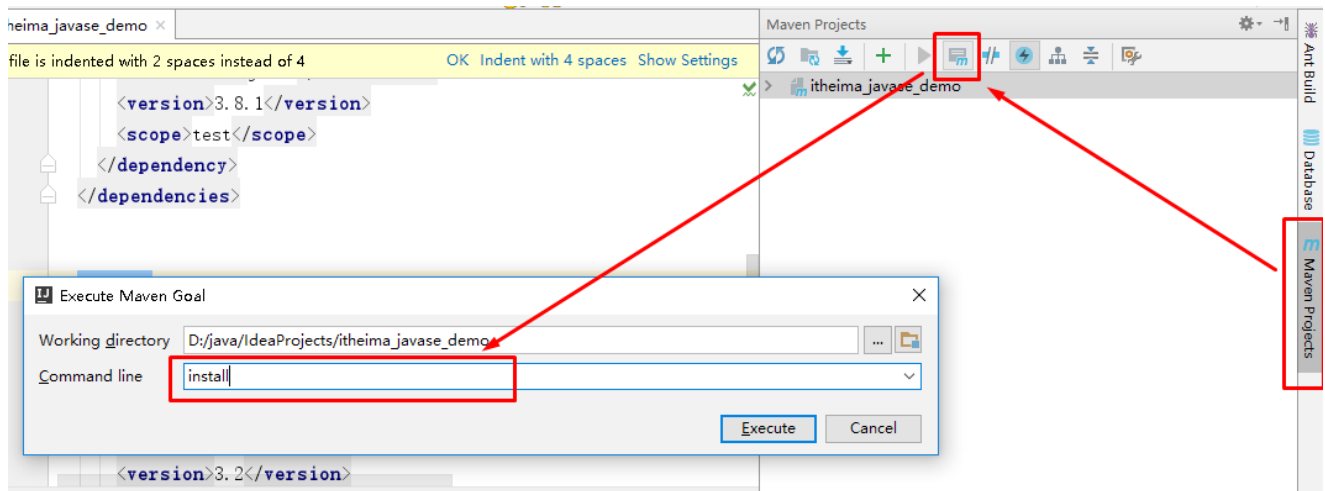
打包后的项目会在target目录下找到



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

com > itheima > itheima_javase_demo > 1.0-SNAPSHOT				
名称	修改日期	类型		
_remote.repositories	2018/2/22 11:34	REP		
itheima_javase_demo-1.0-SNAPSHOT.jar	2018/2/22 11:32	Exe		
itheima_javase_demo-1.0-SNAPSHOT.pom	2018/2/22 11:26	POI		
maven-metadata-local.xml	2018/2/22 11:34	XM		

六,插件和依赖管理

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。

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

1.1 导入junit的依赖

- 导入junit坐标依赖

```

<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>3.8.1</version>
  <scope>test</scope>
</dependency>

```

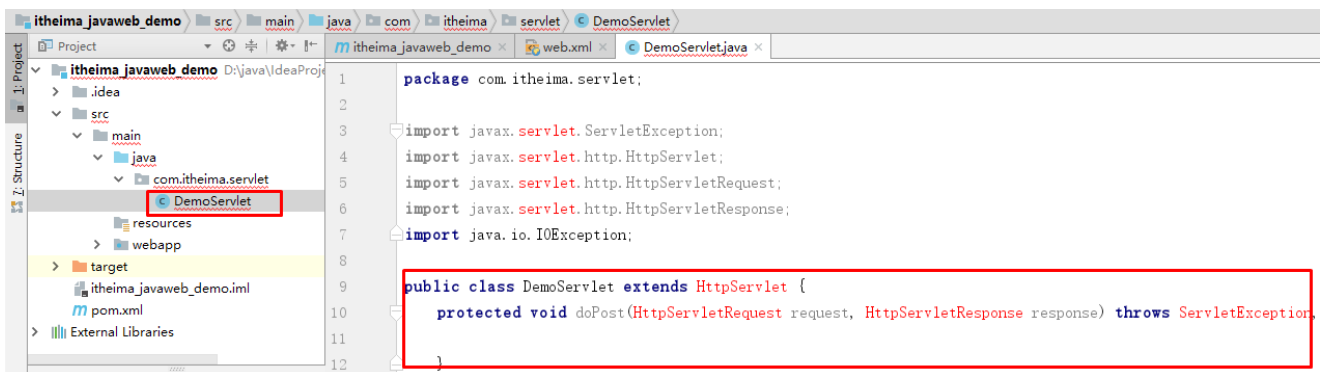
- 进行单元测试


```
import org.junit.Test;

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

1.2 导入servlet的依赖

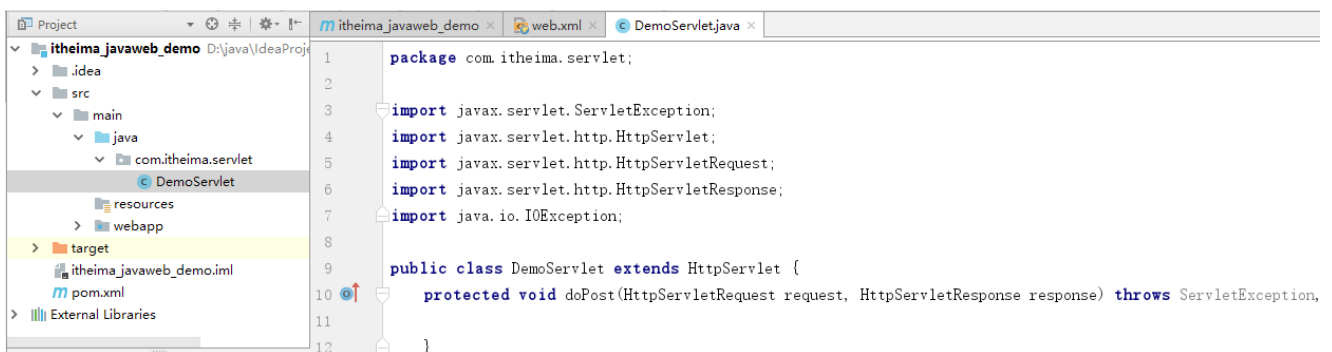
- 创建Servlet，但是发现报错，原因是没有导入Servlet的坐标依赖



- 导入Servlet的坐标依赖

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

- 原有报错的Servlet恢复正常



1.3 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

- **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：strus-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

2 Maven插件

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。

2.1 JDK编译版本的插件

```

<!--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>

```

2.2 Tomcat7服务端的插件

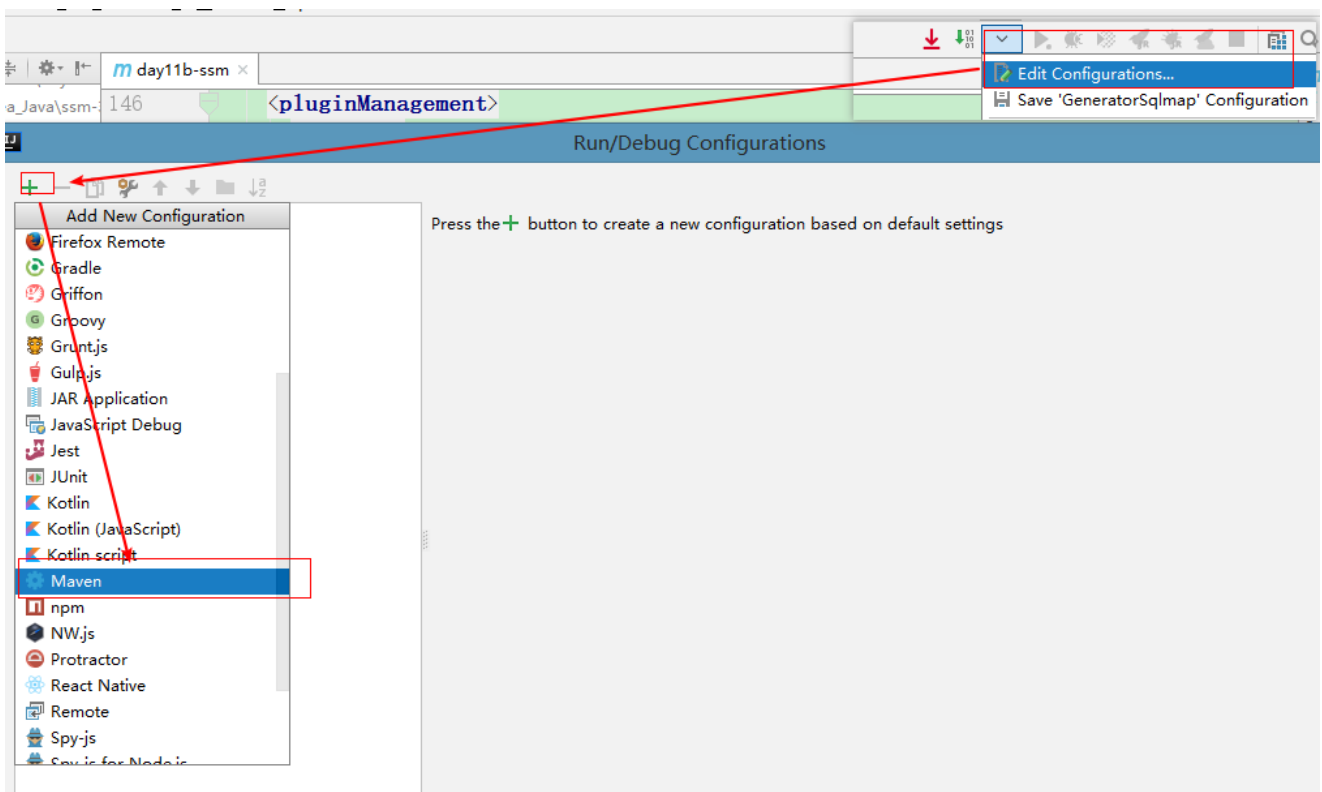
- 添加tomcat7插件

```

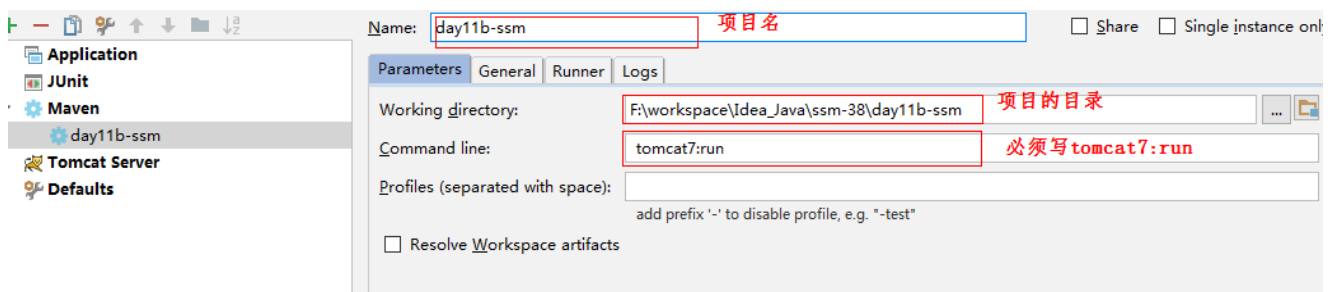
<!-- tomcat7插件 -->
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.1</version>
  <configuration>
    <port>9090</port>
    <path>/</path>
    <uriEncoding>UTF-8</uriEncoding>
    <server>tomcat7</server>
  </configuration>
</plugin>

```

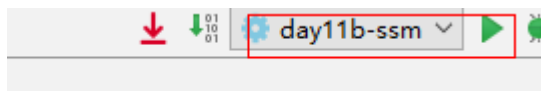
- 配置Maven插件,在Configurations窗口中的左上角 点击 + 号，再选择maven选项



- 然后就可以在右边配置tomcat run的信息，需要配置的有：Name = 项目名， command line = tomcat7:run
Working directory = 项目的路径 ， 配置完成后，点击右下角的Apply，然后点击OK完成



- 启动



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，如果想使用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。

七 使用Maven搭建Servlet+JSP项目

1. 案例需求

完成添加客户信息的操作

客户名称:	345
客户来源:	345
客户级别:	345
客户行业:	345
客户地址:	345
客户电话:	345
<div>保存</div>	

2 案例的准备工作

- 创建数据库

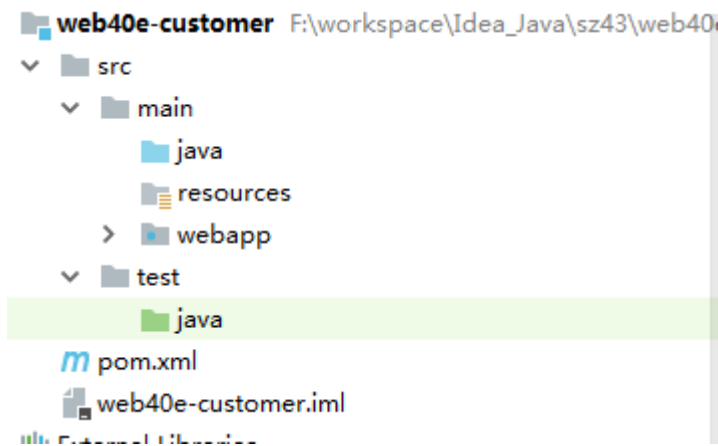
```
CREATE TABLE `cst_customer` (
  `cust_id` bigint(32) NOT NULL AUTO_INCREMENT COMMENT '客户编号(主键)',
  `cust_name` varchar(32) NOT NULL COMMENT '客户名称(公司名称)',
  `cust_source` varchar(32) DEFAULT NULL COMMENT '客户信息来源',
  `cust_industry` varchar(32) DEFAULT NULL COMMENT '客户所属行业',
  `cust_level` varchar(32) DEFAULT NULL COMMENT '客户级别',
  `cust_phone` varchar(64) DEFAULT NULL COMMENT '固定电话',
  `cust_mobile` varchar(16) DEFAULT NULL COMMENT '移动电话',
  PRIMARY KEY (`cust_id`)
)
```

- 创建JavaBean

```
public class Customer implements Serializable {
    private Long custId;
    private String custName;
    private String custSource;
    private String custLevel;
    private String custIndustry;
    private String custMobile;
    private String custPhone;

}
```

- 创建Maven项目



- 导入坐标

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <!--servlet-->
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
  </dependency>
  <!--mysql驱动-->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.26</version>
  </dependency>
  <!--c3p0连接池-->
  <dependency>
    <groupId>c3p0</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.1.2</version>
  </dependency>
  <!--jdbcTemplate-->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>4.1.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>4.1.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>4.1.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>4.1.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.1.1</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

```
<!--beanUtils-->
<dependency>
  <groupId>commons-beanutils</groupId>
  <artifactId>commons-beanutils</artifactId>
  <version>1.9.2</version>
</dependency>
</dependencies>
```

- 导入页面

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<form action="addCustomerServlet" method="post">
  客户名称: <input type="text" name="custName"/><br/>
  客户来源: <input type="text" name="custSource"/><br/>
  客户级别: <input type="text" name="custLevel"/><br/>
  客户行业: <input type="text" name="custIndustry"/><br/>
  客户手机: <input type="text" name="custMobile"/><br/>
  客户电话: <input type="text" name="custPhone"/><br/>
  <input type="submit" value="保存"/><br/>
</form>
</body>
</html>
```

3.思路分析

4.代码实现