# 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. 能够自定义javeweb的Maven工程
9. 能够掌握依赖引入的配置方式
10. 能够了解依赖范围的概念

反馈

|  |
| --- |
| 老师给的文档总是在细节上有些问题,,,导致最后代码跑不出来,比方说今天的jedis.properties,文档里写的是jedis.host,运行后报错无法初始化JdbcUtils....很困扰  我尽量详细。 |
| 宇哥,用你多年的开发经验讲讲适配器模式?  适配器模式适当了解下。 |
| 好不容易笔记代码少，今天怎么没作业啊  预习是重要，昨天给2天预习，今天好好预习javamaill(过滤器那天给了拓展资料，大家好好学习一下，明天会具体应用) |
| 要考虑到经常不修改的数据才添加到redis里面的话，redis用的频率不会很高吧，NOSQL的特点主要体现在哪呢  Redis解决查询性能的，所以主要针对不经常修改又查询量非常大，redis在web阶段使用程度比较浅，在项目阶段使用的更多。redis里面数据过期了，怎么更新，以后大家会学数据同步机制。 |
| redis 如果多次请求，库里面是有值，就会多次读取同一值，其他的选择不会过判断？  Key是唯一的，读取值比mysql等关系型数据库判断逻辑少很多。 |
| 宇哥 上课的时候多带一下我们用DeBug调试 课后自己操作实在低没头绪  Debug,后面会跟大家一起进行 |
| 一个晚上就搞一个案例,debug找了一晚上,感觉效率好低啊!!!  每个人最好都有自己独立debug解决，但是时间控制半个小时，解决不了找同学解决，在解决助教老师，反馈，课下找老师。 |
| 默认的rdb持久化是开启，那么如果开启AOF持久化,是不是两个持久化都开启了  都开启了，redis服务器启动默认读取rdb数据恢复。aof有一个exe工具，他将aof持久化文件进行恢复，在恢复的过程时，综合rdb持久化的数据进行控制统一性。 |
| 是不是想访问某条数据,如果redis中没有数据,都要把mysql数据库的数据放在redis数据库,然后我们再去访问redis中的数据?如果是数据经常变的呢？  Redis存储不经常改变的，具体时间控制在30分之内不要更新。后面大家会学定时器更新redis同步数据。 |
| 今天写代码有点不对劲，一下子就成功了，这样子就比较尴尬了。本来还想DEBUG一下的 一下子就成功了，是不是很不好。 |

# Maven概述

## Maven的概念

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。

## Maven的作用

* 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。
* 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。

## Maven的概念模型

* 项目对象模型 (Project Object Model)。POM对象模型，每个maven工程中都有一个pom.xml文件，定义工程所依赖的jar包、本工程的坐标、打包运行方式。
* 依赖管理系统（基础核心 ）。maven通过坐标对项目工程所依赖的jar包统一规范管理。
* maven定义一套项目生命周期。**清理**、初始化、**编译、测试、**报告 **、打包**、**安装、**部署、站点生成。
* 一组标准集合。maven工程有自己标准的工程目录结构、定义坐标有标准。
* maven 管理项目生命周期过程都是基于插件完成的，例如：开发中使用的tomcat插件。

## 1.4 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。 |

## 1.5 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：定义当前Maven项目名称
* artifactId：定义项目模块
* version：定义当前项目的当前版本

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

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

# 第2章 Maven的安装

## 下载Maven



## 安装Maven

将Maven压缩包解压，即安装完毕

|  |
| --- |
|  |

## Maven目录介绍

|  |
| --- |
|  |

## 配置环境变量

|  |
| --- |
|  |

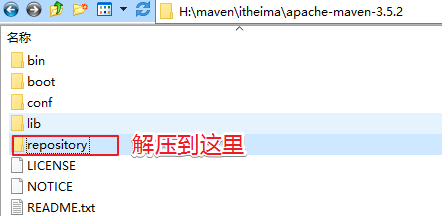
环境变量

MAVEN\_HOME:H:\maven\itheima\apache-maven-3.3.9

Path:%MAVEN\_HOME%\bin

## 配置本地仓库

### 将软件文件夹中的Repository解压



### 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

<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>H:\maven\itheima\apache-maven-3.5.2\repository</localRepository>

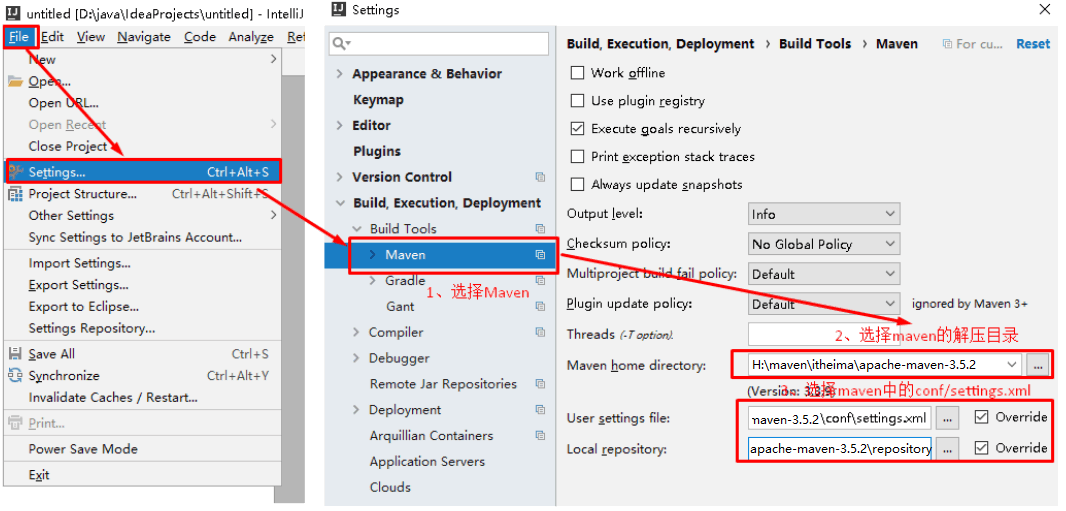
## 测试Maven安装成功

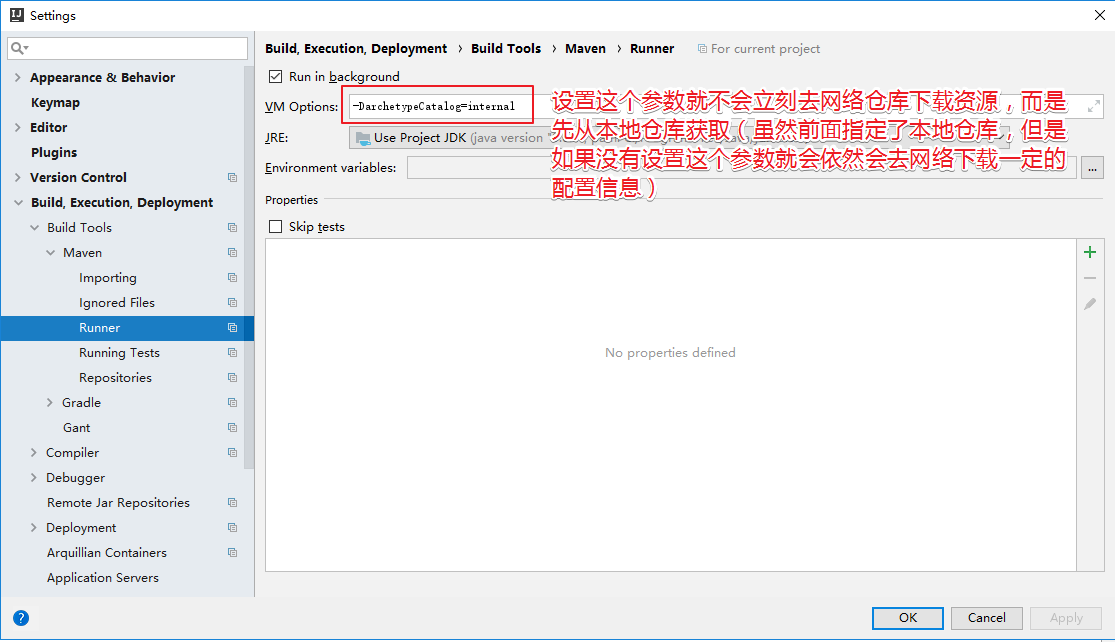
打开cmd本地控制台，输入mvn -version



# IDEA创建Maven工程

## IDEA指定本地Maven





-DarchetypeCatalog=internal

所有资源都先从本地仓库查找，没有再去网络。

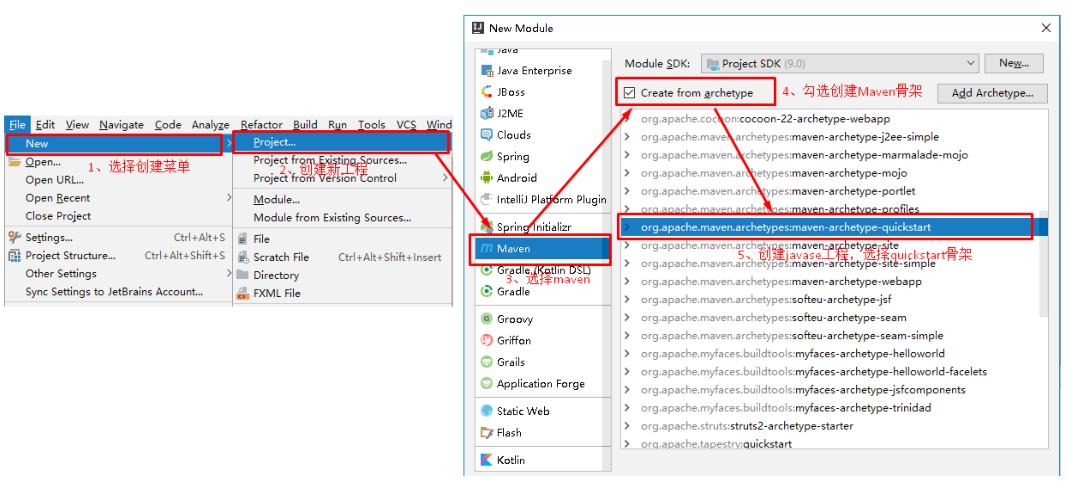
idea使用maven生命周期命令输出中文乱码,解决方案

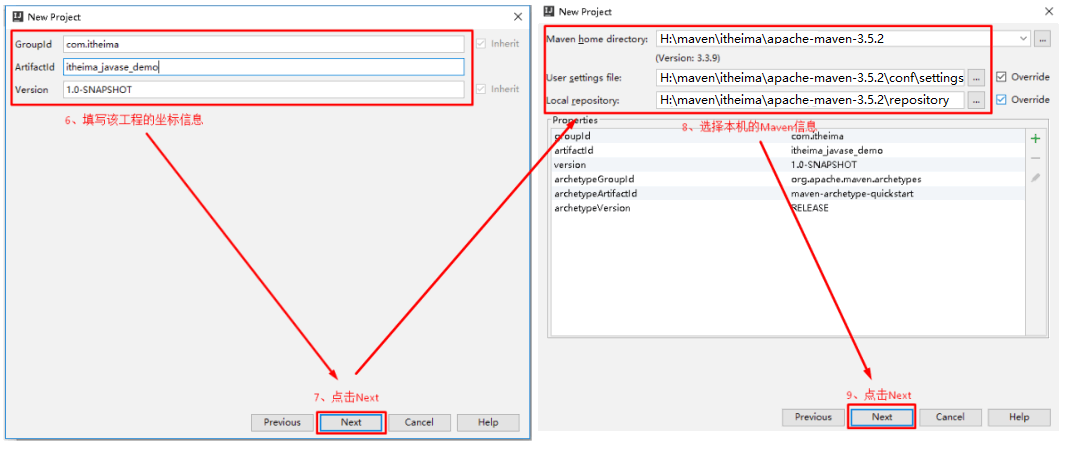
    Setting->maven->runner

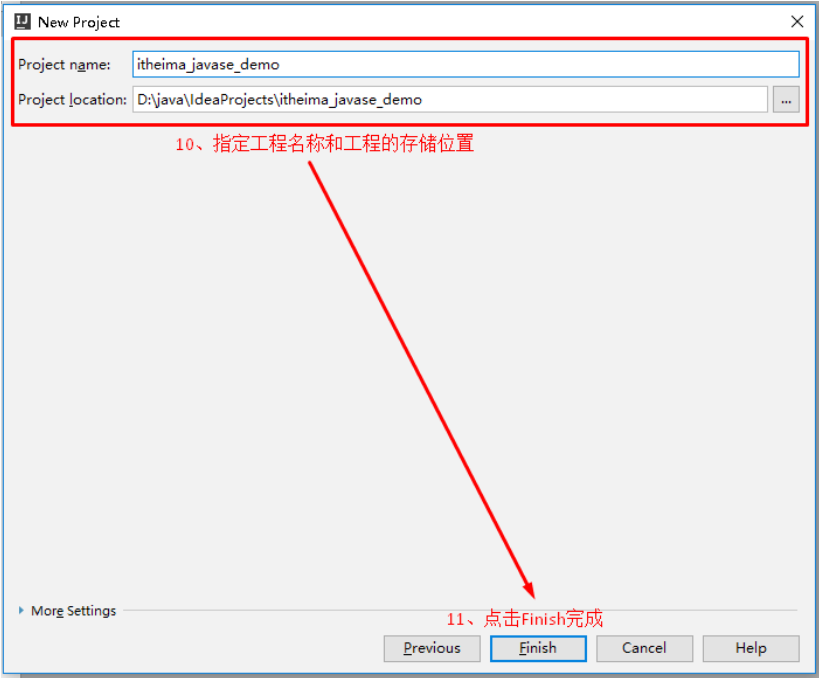
       VMoptions: -Dfile.encoding=GB2312  
    C:\Users\songyu\AppData\Roaming\Foxmail7\Temp-8060-20180523214230\Attach\CatchD2D0(05-11-(05-24-15-35-17).jpg

## 创建java工程

### 创建java工程

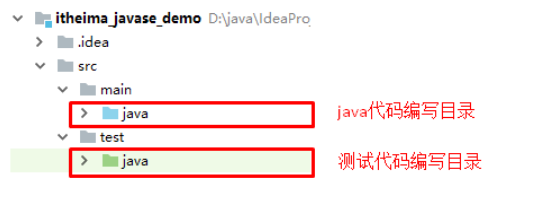








### java工程目录结构



### maven项目目录结构要求

项目名字

src

main

java,存放java源代码（可以部署到服务器上的）

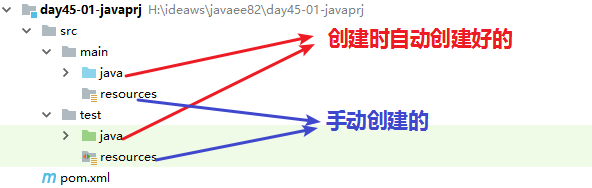
resources,存储配置文件的

test

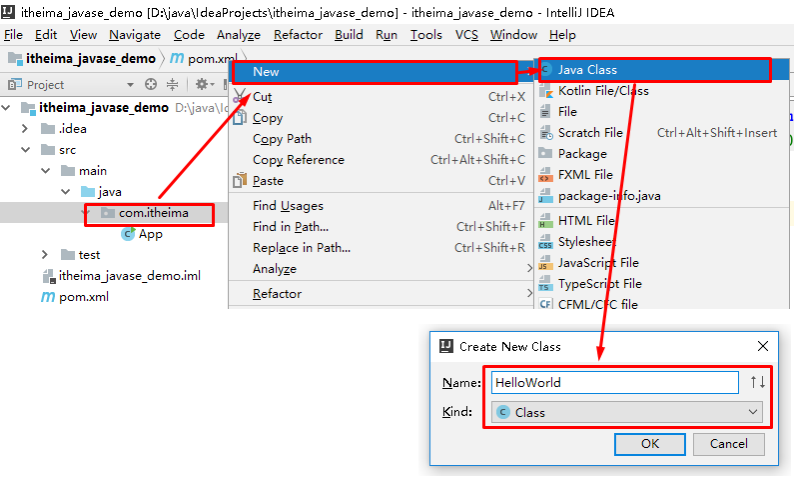
java，存放测试代码（不会部署到服务器上）

resources,测试的配置文件

创建java工程注意的地方



### 编写Hello World！





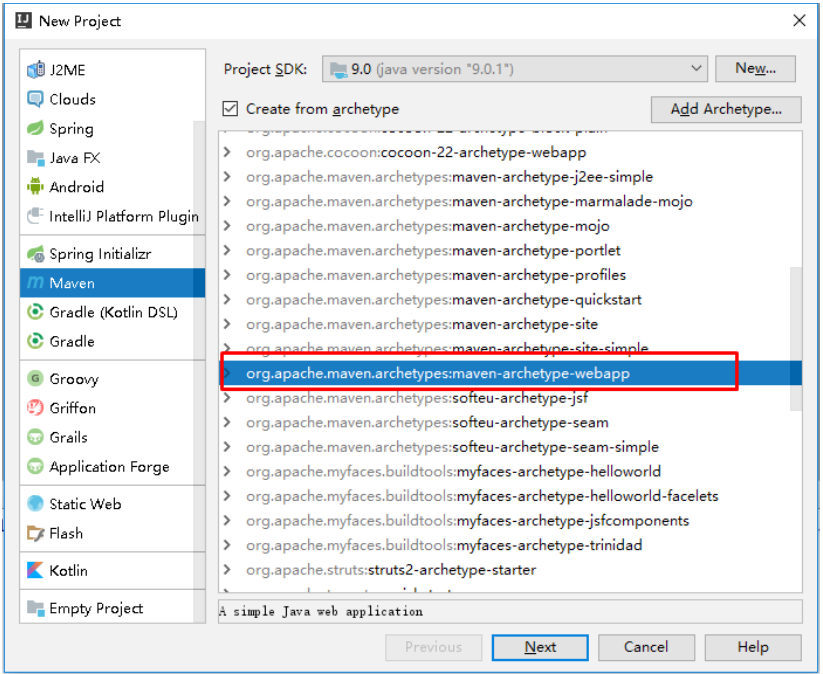
## 使用骨架创建javaweb工程（推荐使用）

### maven创建web项目目录结构要求

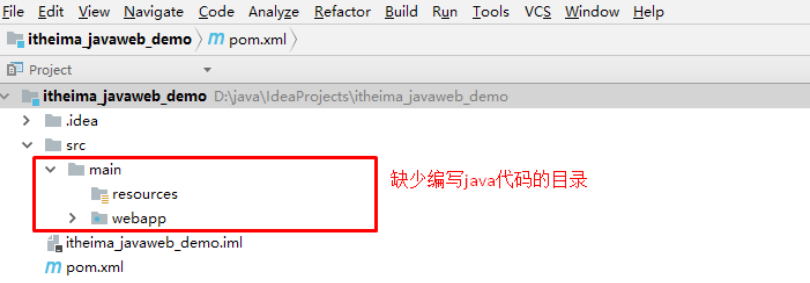
计算机生成了可选文字:
4 ． 1 ． maven 项 目 的 目 录 结 构 
srcknain/java 一 一 存 放 项 目 的 java 文 件 （ 开 发 源 代 码 ） 
ajn/resources _ _ 存 放 项 目 配 置 文 件 ， 如 果 没 有 配 置 文 件 该 目 录 可 无 ， 如 
spring, hibernate 配 置 文 件 
src/main/webapp 一 一 存 放 web 项 目 资 源 文 件 (web 项 目 才 有 ） 
src/test/java 一 一 存 放 所 有 测 试 ． java 文 件 （ 试 源 代 码 ） 
src/test/resources _ _ 测 试 配 置 文 件 ， 如 果 没 有 配 置 文 件 该 目 录 可 无 
target 一 一 项 目 输 出 位 置 （ 可 无 ） 
pom.xml—maven 项 目 核 心 配 置 文 件 
也 就 是 如 果 是 一 个 Maven 项 目 那 么 它 的 根 录 下 必 定 存 在 “ 文 件 夹 和 
pom.xmlo 

### 创建步骤

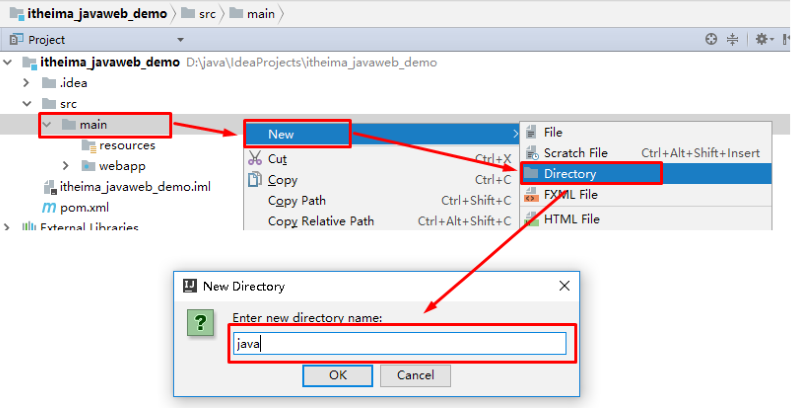
* 创建javaweb工程与创建javase工程类似，但在选择Maven骨架时，选择maven-archetype-webapp即可：



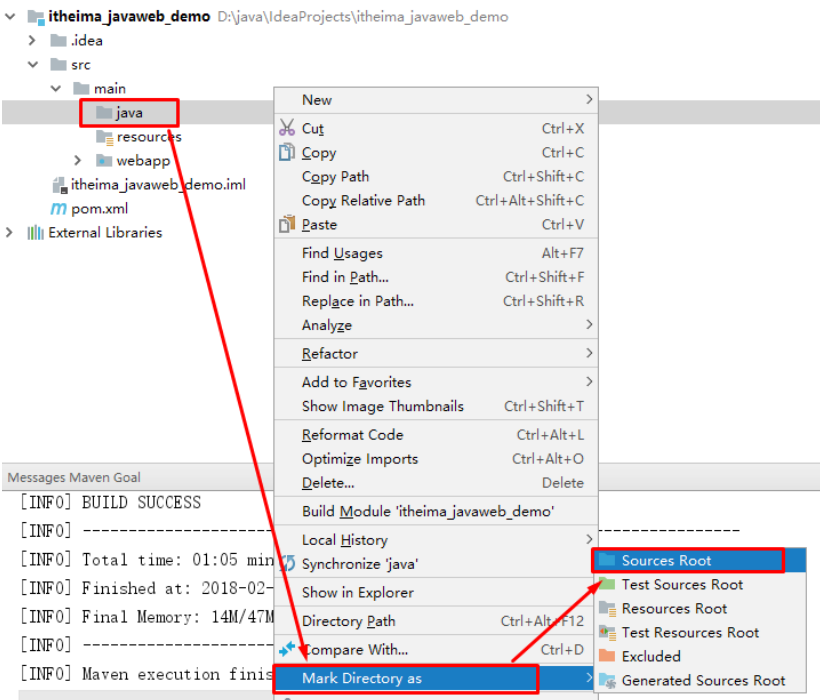
* 创建好的javaweb工程如下：



* 所以，要手动创建一个java目录用于编写java代码：



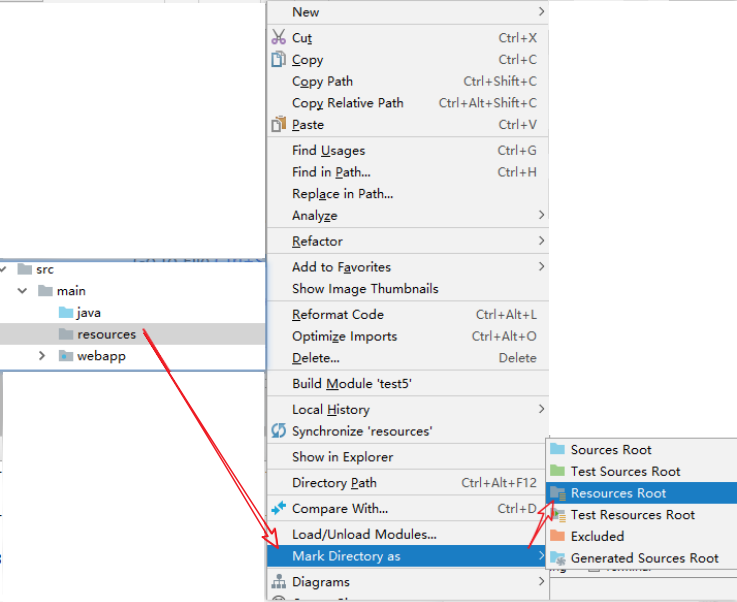
* 还要将java目录添加为Source Root：



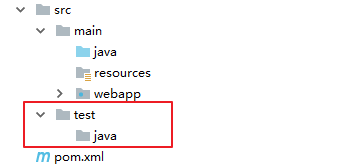
在main下手动创建目录resources

|  |
| --- |
|  |

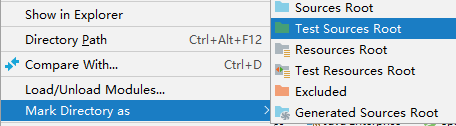
进行如下操作：



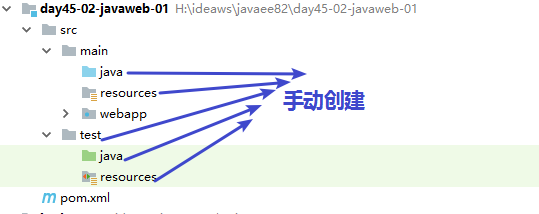
创建test目录，并在test目录下创建java，如图



选择test目录下的java，鼠标右键“Mark Directory as”为“Test Source Root”



最终目录结构



webapp相当于web目录（javaweb项目的web目录，里面放WEB-INF/html/jsp…）

Resources用于存储配置文件

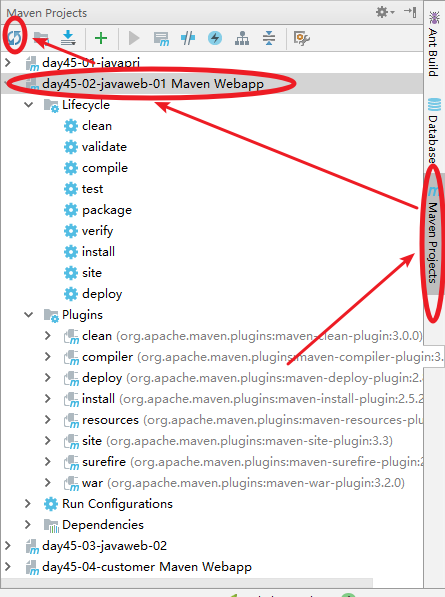
如果pom模板中没有配置过的依赖，可以通过http://mvnrepository.com/ 在线搜索坐标写法

### 创建自定义servlet

第一步：在pom.xml加入依赖

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

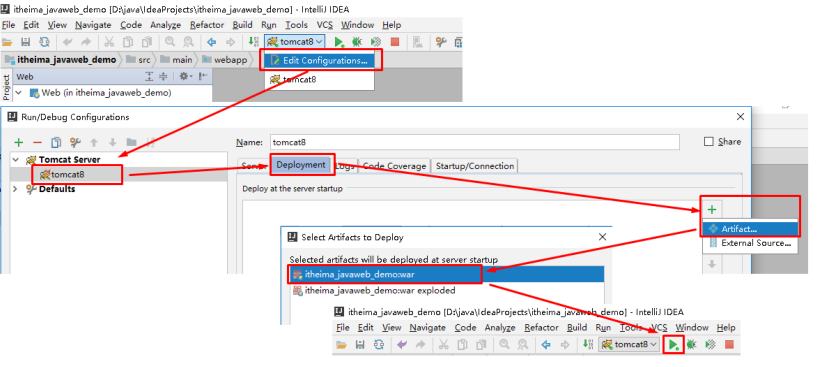
第二步：刷新maven project窗口



第三步：创建servlet

|  |
| --- |
| **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"DemoServlet"**, urlPatterns = **"/DemoServlet"**) **public class** DemoServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  response.getWriter().write(**"hello maven2"**);  } } |

### 发布javaweb工程



### 浏览器访问效果



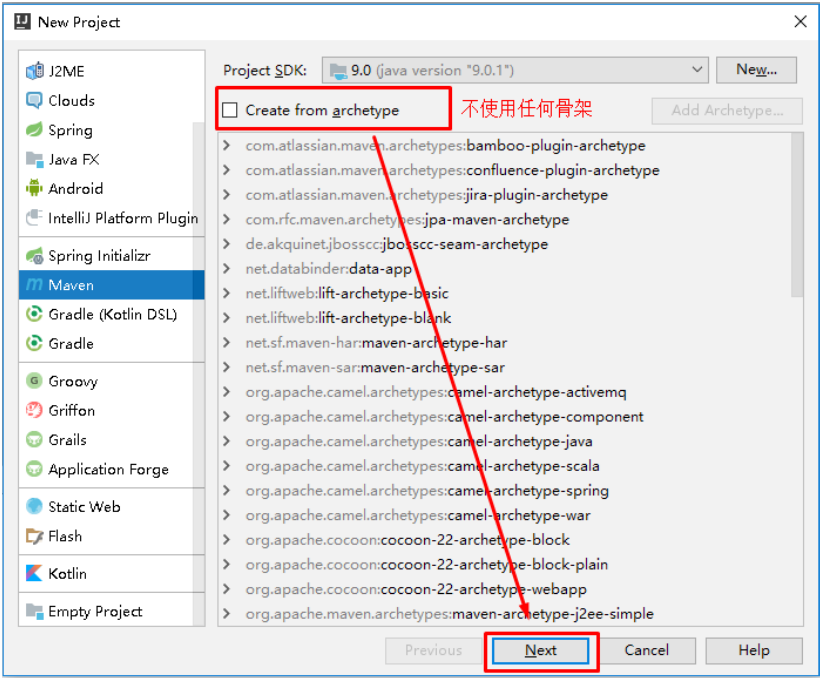
### Maven创建javaweb工程的目录结构



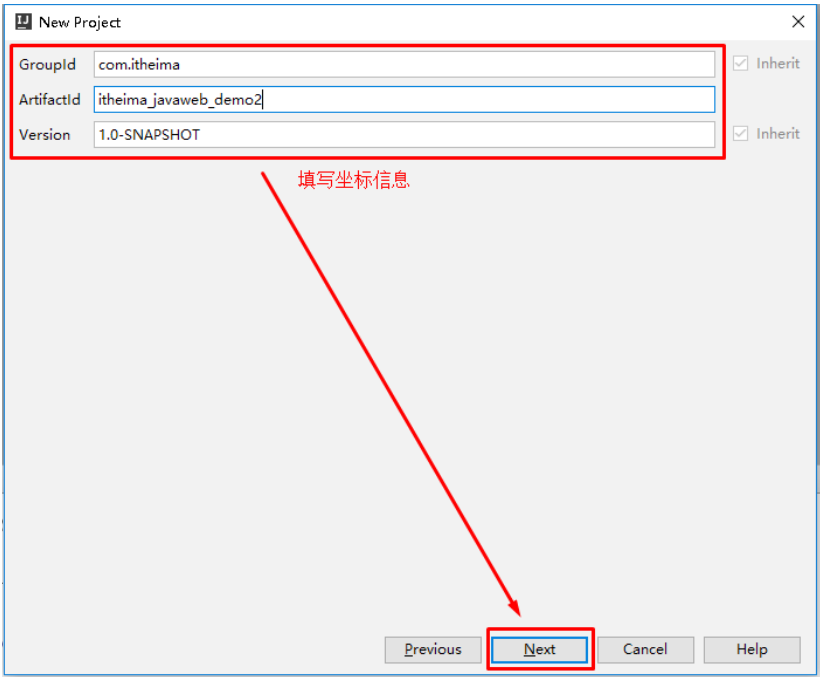
## 创建自定义JavaWeb工程

在上面创建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：

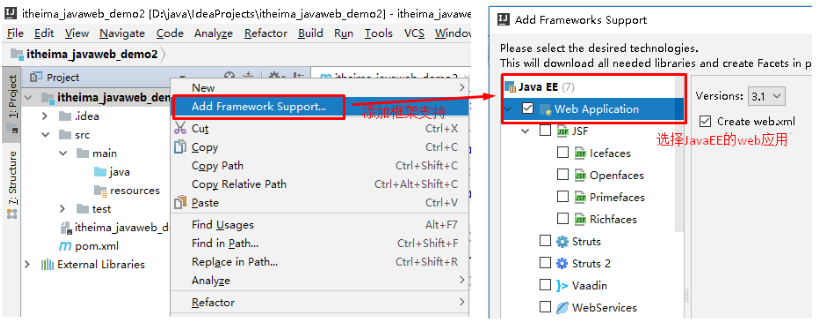
* 创建一个Maven工程，不选择任何骨架



* 填写坐标信息



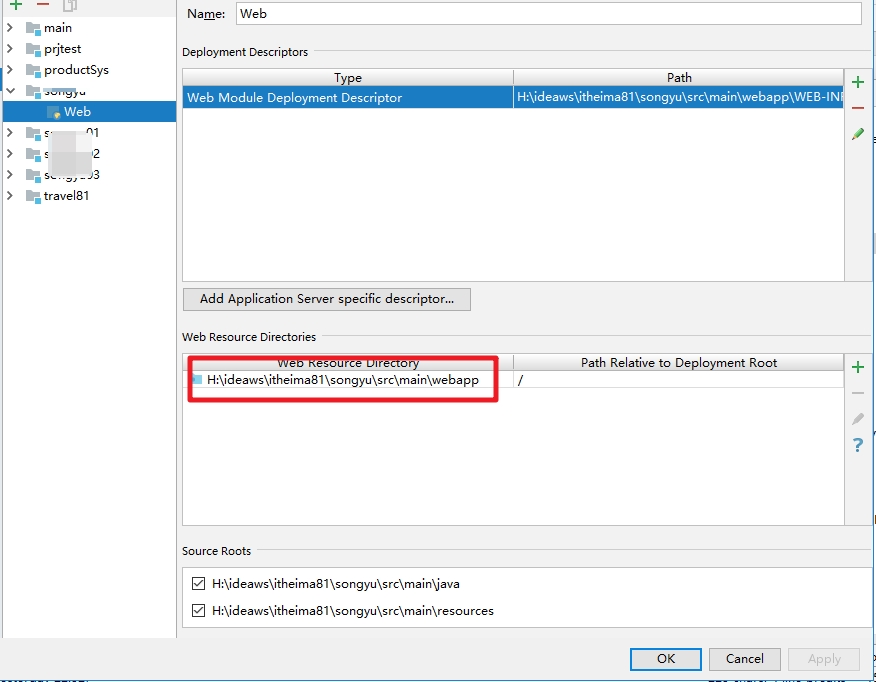
* 创建web工程信息



* 创建好的javaweb工程的目录结构如下



将web目录移入到main目录下,并更改名为webapp,并通过open module setting设置,如下图:



 进行执行maven生命周期命令是报错的，报jdk1.5或1.6过时，需要更新默认使用编译jdk版本,

在pom.xml加入如下配置

    <**properties**>

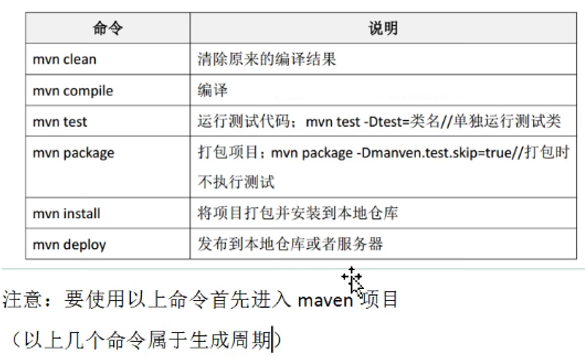
<**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**maven.compiler.source**>1.8</**maven.compiler.source**>  
 <**maven.compiler.target**>1.8</**maven.compiler.target**>  
 </**properties**>

# Maven的常用命令

为了更好的测试运行，需要先在pom.xml中加入如下依赖

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

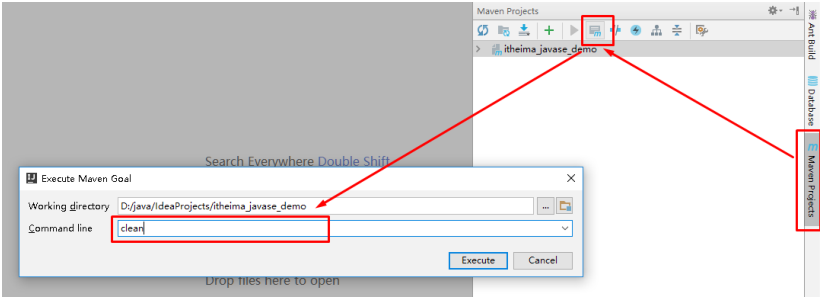
maven生命周期命令



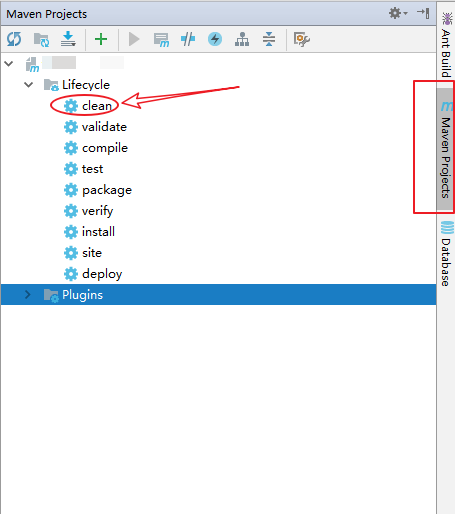


## clean命令

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

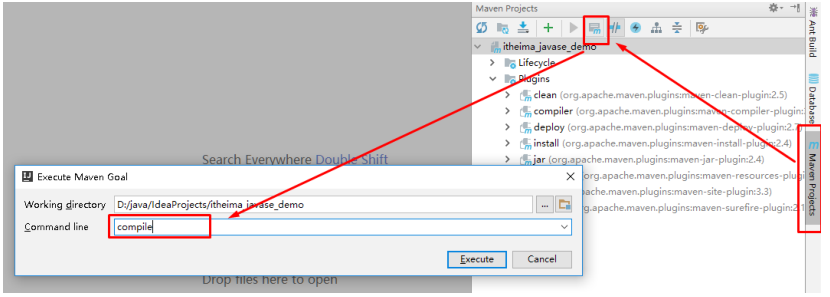


也可以双击如下图



## complie命令

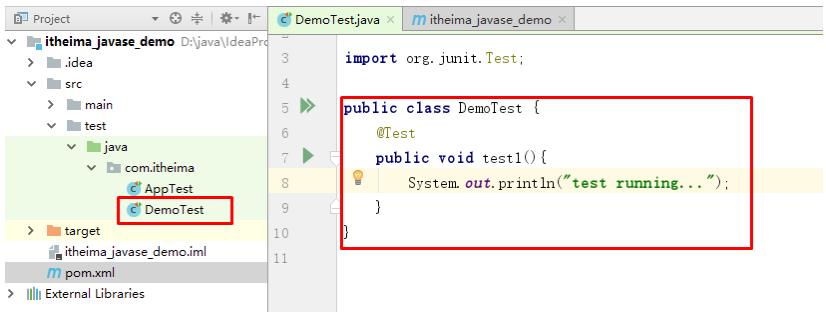
该命令可以对src/main/java目录的下的代码进行编译(不对test目录下java编译)



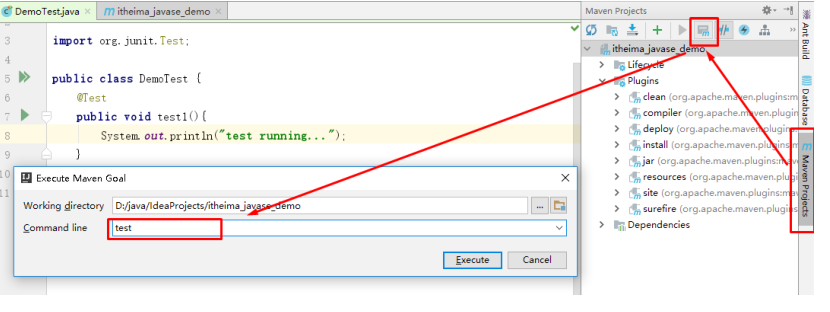
## test命令

测试命令,或执行src/test/java/下junit的测试用例

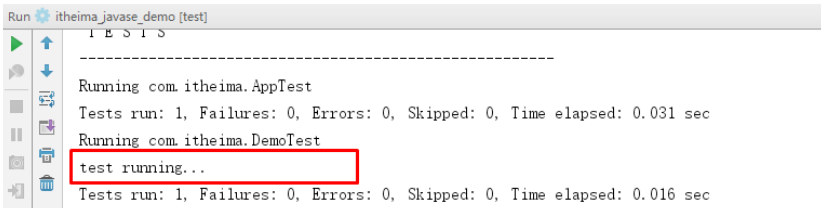
* 在src/test/java下创建测试类DemoTest



* 执行test命令测试

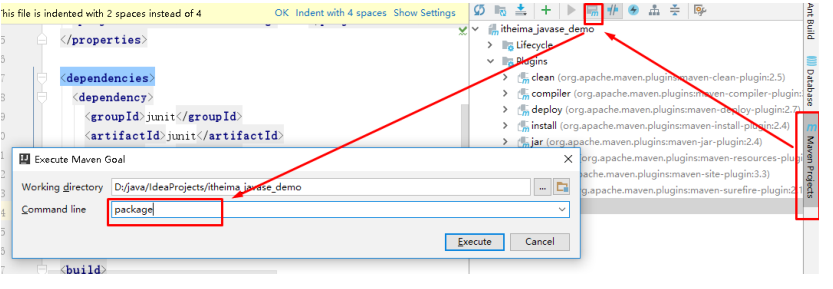


* 控制台显示测试结果

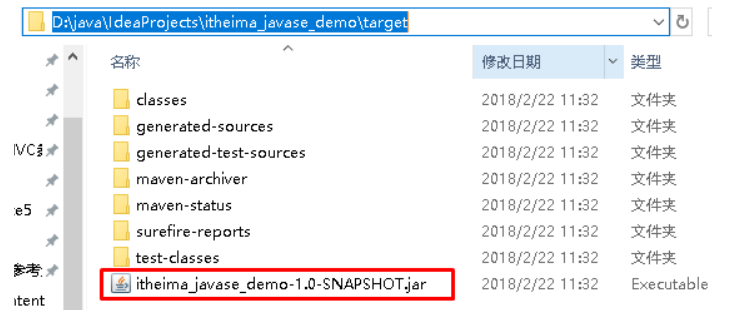


## package命令

mvn package，打包项目

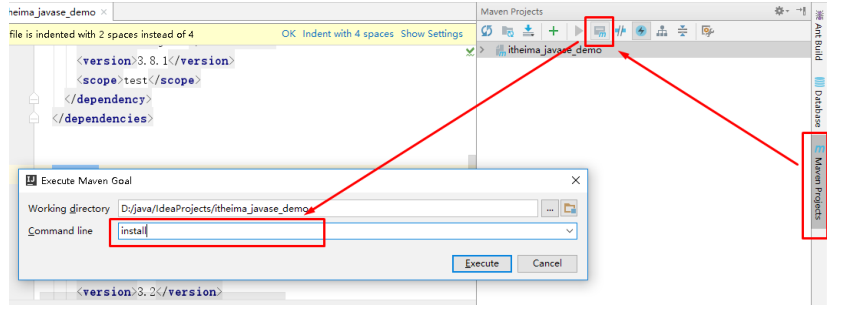


打包后的项目会在target目录下找到

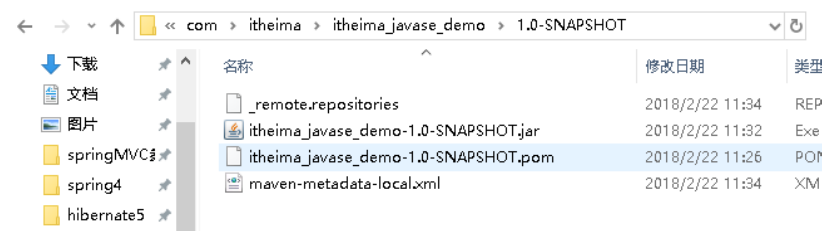


## install命令

mvn install，打包后将其安装在本地仓库



安装完毕后，在本地仓库中可以找到itheima\_javase\_demo的信息



# 依赖管理

## 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。

### JDK编译版本的插件

<!-- jdk版本插件 -->  
<plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-compiler-plugin</artifactId>  
    <version>3.2</version>  
    <configuration>  
        <source>1.7</source>  
        <target>1.7</target>  
        <encoding>UTF-8</encoding>  
        <showWarnings>true</showWarnings>  
    </configuration>  
</plugin>

### Tomcat7服务端的插件

* 配置tomcat7插件

<!--tomcat插件-->

<plugin>

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

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

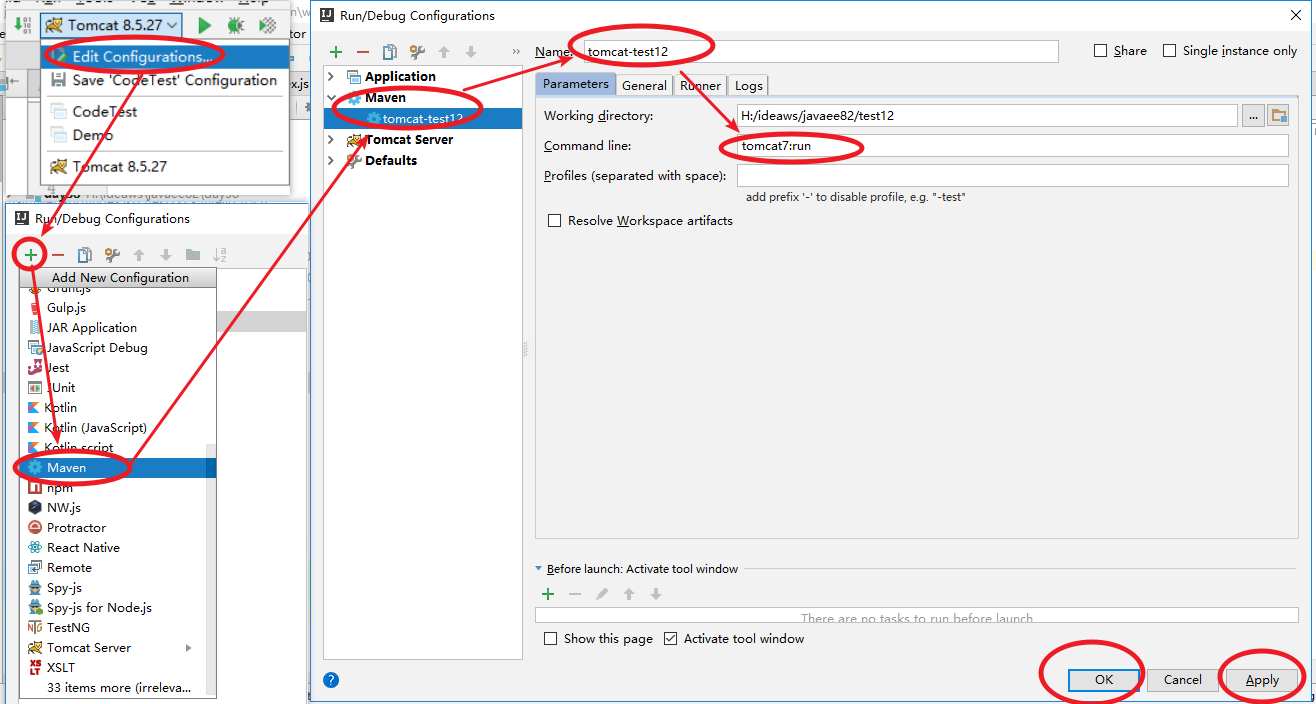
<port>8080</port>

<path>/projectname</path>

</configuration>

</plugin>

* 配置maven运行的tomcat



* 运行tomcat7插件(注意要使用jdk1.8)



* tomcat插件注意问题

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极其以上版本，进行项目的发布。

## 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<dependency>标签引入依赖。

### 6.2.1 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>

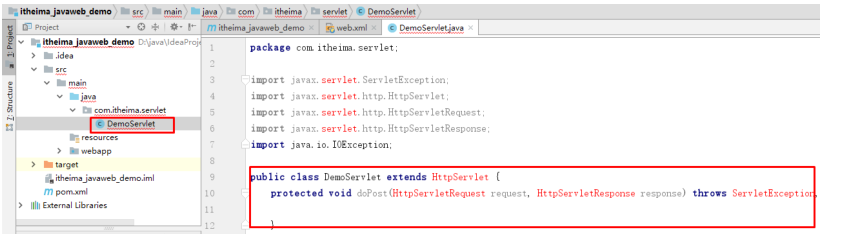
* 进行单元测试

import org.junit.Test;  
​  
public class DemoTest {  
    @Test  
    public void test1(){  
        System.out.println("test running...");  
   }  
}

如果依赖不会写，可以联网查怎么写，<http://mvnrepository.com/>

### 导入servlet的依赖

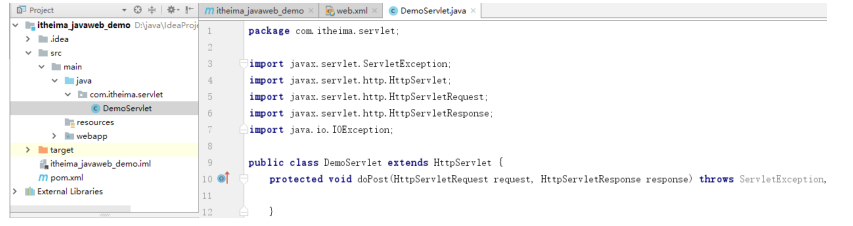
* 创建Servlet，但是发现报错，原因是没有导入Servlet的坐标依赖



* 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



## 6.3 依赖范围



* 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。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

* system：不是使用仓库的jar包，而是使用本地文件系统的jar包，需要域systempath一起配合使用。

例如：junit

不会打到war，如果运行需要使用，需要手动将jar包拷贝到tomcat/lib下

|  |
| --- |
| <dependency> <groupId>javabuilder</groupId> <artifactId>javabuilder</artifactId> <version>1.0</version> <scope>system</scope> <systemPath>${project.basedir}/src/main/webapp/WEB-INF/lib/javabuilder.jar </systemPath> </dependency>  像这样把包拷到项目里面，然后指定到项目下面就可以了。 |

* 按照依赖强度，由强到弱来排序：(理解)

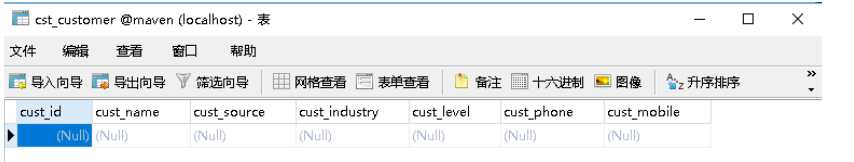
compile> provided> runtime> test

# Maven案例：使用Maven搭建Servlet+JSP+JdbcTemplate项目

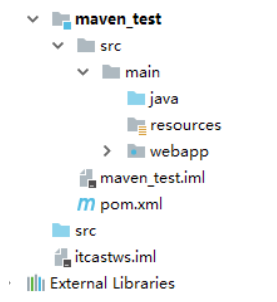
## 需求分析

完成添加客户信息的操作

## 创建数据表



## 创建Maven项目



## 导入坐标

### 导入所需的依赖（jar包）和插件

* mysql驱动程序
* DBUtils
* servlet-api （Servlet的API）
* jsp-api （可以是EL表达式）

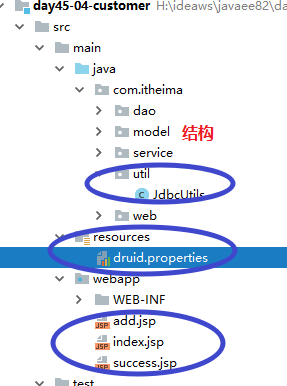
### maven坐标

|  |
| --- |
| *<?***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**>itheima</**groupId**>  <**artifactId**>day45</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  <**packaging**>war</**packaging**>   <**name**>day45 Maven Webapp</**name**>  *<!--* ***FIXME change it to the project's website*** *-->* <**url**>http://www.example.com</**url**>   <**properties**>  <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  <**maven.compiler.source**>1.7</**maven.compiler.source**>  <**maven.compiler.target**>1.7</**maven.compiler.target**>  </**properties**>   <**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**>  <**scope**>compile</**scope**>  </**dependency**>  *<!--druid连接池-->* <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>druid</**artifactId**>  <**version**>1.0.9</**version**>  </**dependency**>  *<!--jdbcTemplate-->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-core</**artifactId**>  <**version**>4.1.2.RELEASE</**version**>  <**scope**>compile</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-jdbc</**artifactId**>  <**version**>4.1.2.RELEASE</**version**>  <**scope**>compile</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-tx</**artifactId**>  <**version**>4.1.2.RELEASE</**version**>  <**scope**>compile</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-beans</**artifactId**>  <**version**>4.1.2.RELEASE</**version**>  <**scope**>compile</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>commons-logging</**groupId**>  <**artifactId**>commons-logging</**artifactId**>  <**version**>1.1.1</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!--beanUtils-->* <**dependency**>  <**groupId**>commons-beanutils</**groupId**>  <**artifactId**>commons-beanutils</**artifactId**>  <**version**>1.9.2</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!--jackson-->* <**dependency**>  <**groupId**>com.fasterxml.jackson.core</**groupId**>  <**artifactId**>jackson-databind</**artifactId**>  <**version**>2.3.3</**version**>  </**dependency**>  <**dependency**>  <**groupId**>com.fasterxml.jackson.core</**groupId**>  <**artifactId**>jackson-core</**artifactId**>  <**version**>2.3.3</**version**>  </**dependency**>  <**dependency**>  <**groupId**>com.fasterxml.jackson.core</**groupId**>  <**artifactId**>jackson-annotations</**artifactId**>  <**version**>2.3.3</**version**>  </**dependency**>  </**dependencies**>   <**build**>  <**finalName**>day45</**finalName**>  <**pluginManagement**>*<!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->* <**plugins**>  <**plugin**>  <**artifactId**>maven-clean-plugin</**artifactId**>  <**version**>3.0.0</**version**>  </**plugin**>  *<!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->* <**plugin**>  <**artifactId**>maven-resources-plugin</**artifactId**>  <**version**>3.0.2</**version**>  </**plugin**>  <**plugin**>  <**artifactId**>maven-compiler-plugin</**artifactId**>  <**version**>3.7.0</**version**>  </**plugin**>  <**plugin**>  <**artifactId**>maven-surefire-plugin</**artifactId**>  <**version**>2.20.1</**version**>  </**plugin**>  <**plugin**>  <**artifactId**>maven-war-plugin</**artifactId**>  <**version**>3.2.0</**version**>  </**plugin**>  <**plugin**>  <**artifactId**>maven-install-plugin</**artifactId**>  <**version**>2.5.2</**version**>  </**plugin**>  <**plugin**>  <**artifactId**>maven-deploy-plugin</**artifactId**>  <**version**>2.8.2</**version**>  </**plugin**>  </**plugins**>  </**pluginManagement**>  </**build**> </**project**> |

​

### 编写代码

#### 代码结构（导入JdbcUtils和druid.properties）



#### 页面编写

* add.jsp表单页面

<%@ page language="java" contentType="text/html; charset=utf-8"  
         pageEncoding="utf-8"%>  
<!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>客户录入页面</title>  
</head>  
<body>  
<form action="SaveCustomer" 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/>  
   客户地址：<input type="text" name="custAddress"/><br/>  
   客户电话：<input type="text" name="custPhone"/><br/>  
    <input type="submit" value="保存"/><br/>  
</form>  
</body>  
</html>

* success.jsp添加成功页面

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
    <h1>save success!!!</h1>  
</body>  
</html>  
​

#### 编写实体类

|  |
| --- |
| **package** com.itheima.model;  **public class** Customer {  **private** Long **custId**;  **private** String **custName**;  **private** String **custSource**;  **private** String **custLevel**;  **private** String **custIndustry**;  **private** String **custMobile**;  **private** String **custPhone**;   **public** Long getCustId() {  **return custId**;  }   **public void** setCustId(Long custId) {  **this**.**custId** = custId;  }   **public** String getCustName() {  **return custName**;  }   **public void** setCustName(String custName) {  **this**.**custName** = custName;  }   **public** String getCustSource() {  **return custSource**;  }   **public void** setCustSource(String custSource) {  **this**.**custSource** = custSource;  }   **public** String getCustLevel() {  **return custLevel**;  }   **public void** setCustLevel(String custLevel) {  **this**.**custLevel** = custLevel;  }   **public** String getCustIndustry() {  **return custIndustry**;  }   **public void** setCustIndustry(String custIndustry) {  **this**.**custIndustry** = custIndustry;  }   **public** String getCustMobile() {  **return custMobile**;  }   **public void** setCustMobile(String custMobile) {  **this**.**custMobile** = custMobile;  }   **public** String getCustPhone() {  **return custPhone**;  }   **public void** setCustPhone(String custPhone) {  **this**.**custPhone** = custPhone;  }   **public** Customer(Long custId, String custName, String custSource, String custLevel, String custIndustry, String custMobile, String custPhone) {  **this**.**custId** = custId;  **this**.**custName** = custName;  **this**.**custSource** = custSource;  **this**.**custLevel** = custLevel;  **this**.**custIndustry** = custIndustry;  **this**.**custMobile** = custMobile;  **this**.**custPhone** = custPhone;  }   **public** Customer() {  } } |

​

#### Servlet编写

|  |
| --- |
| **package** com.itheima.web;  **import** com.itheima.model.Customer; **import** com.itheima.service.CustomerService; **import** org.apache.commons.beanutils.BeanUtils;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.lang.reflect.InvocationTargetException;  @WebServlet(name = **"SaveCustomer"**, urlPatterns = **"/SaveCustomer"**) **public class** SaveCustomer **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   *//实例业务类* **private** CustomerService **customerService** = **new** CustomerService();  **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  *//处理post乱码* request.setCharacterEncoding(**"utf-8"**);    **try** {  *//获取数据封装到对象里* Customer customer = **new** Customer();  BeanUtils.*populate*(customer,request.getParameterMap());  *//调用业务逻辑添加数据* **customerService**.addCustomer(customer);  *//成功跳转页面success.jsp页面* response.sendRedirect(request.getContextPath()+**"/success.jsp"**);   } **catch** (Exception e) {  e.printStackTrace();  *//失败返回添加页面* response.getWriter().write(**"<script>history.back();</script>"**);  }   } } |

#### Service编写

|  |
| --- |
| **package** com.itheima.service;  **import** com.itheima.dao.CustomerDao; **import** com.itheima.model.Customer; */\*\*  \* 业务类  \*/* **public class** CustomerService {   **private** CustomerDao **customerDao** = **new** CustomerDao();   */\*\*  \* 添加客户  \** ***@param customer*** *\** ***@throws*** *Exception  \*/* **public void** addCustomer(Customer customer)**throws** Exception {  **customerDao**.addCustomer(customer);  } } |

#### Dao编写

|  |
| --- |
| **package** com.itheima.dao;  **import** com.itheima.model.Customer; **import** com.itheima.util.JdbcUtils; **import** org.springframework.jdbc.core.JdbcTemplate;  **import** java.sql.SQLException;  */\*\*  \* 数据访问类  \*/* **public class** CustomerDao {   **private** JdbcTemplate **jdbcTemplate** = **new** JdbcTemplate(JdbcUtils.*getDataSource*());    */\*\*  \* 添加客户  \** ***@param customer*** *\** ***@return*** *int  \** ***@throws*** *SQLException  \*/* **public int** addCustomer(Customer customer)**throws** SQLException {  String sql =**"INSERT INTO cst\_customer VALUES(NULL,?,?,?,?,?,?)"**;  **return jdbcTemplate**.update(sql,  customer.getCustName(),  customer.getCustSource(),  customer.getCustIndustry(),  customer.getCustLevel(),  customer.getCustPhone(),  customer.getCustMobile()  );  } } |

​