











# Spring Boot2.x 零基础到高级实战全套视频教程

## 第二章 从零开始认识 Spring Boot

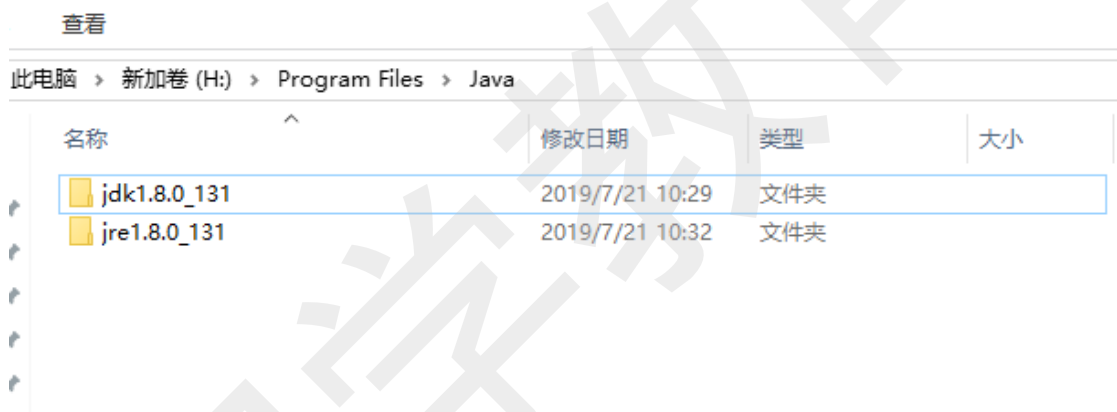
### 1.搭建 Spring Boot2.x 的运行环境

#### 1.1 JDK 安装与环境变量配置

1. 安装JDK Spring Boot 2.0 要求Java 版本必须8以上，Java 6 和 7 不再支持。点击安装选择好安装目录

	apache-activemq-5.9.0-bin	2017/6/22 11:08	360压缩 ZIP 文件	60,428 KB
	apache-maven-3.3.9	2019/6/30 19:51	360压缩 ZIP 文件	8,424 KB
	apache-tomcat-8.5.15-windows-x64	2017/5/25 16:05	360压缩 ZIP 文件	10,757 KB
	ideaIU-2017.1.3	2017/5/25 14:41	应用程序	510,526 KB
	idea注册码	2019/6/30 20:00	文本文档	1 KB
	jdk_8.0.1310.11_64	2018/4/6 21:13	应用程序	202,784 KB
	mysql-5.5.56-winx64	2017/6/17 11:42	Windows Install...	38,508 KB
	Navicat_Premium_11.0.17	2015/1/4 11:01	应用程序	28,698 KB
	Postman-win64-4.10.2-Setup	2017/8/13 12:50	应用程序	64,301 KB
	redis-latest-windws	2017/9/21 18:53	360压缩 ZIP 文件	5,756 KB

2. 安装JDK 选择安装目录 安装过程中会出现两次 安装提示。第一次是安装 jdk(随意选目录一路默认安装)，第二次是安装 jre。建议两个都安装在同一个java文件夹中的不同文件夹中(不能都安装在java文件夹的根目录下，jdk和jre安装在同一文件夹会出错，所以jre需要手动创建一个空文件夹)如下图所示。



3. 安装完JDK后配置环境变量 计算机→属性→高级系统设置→高级→环境变量

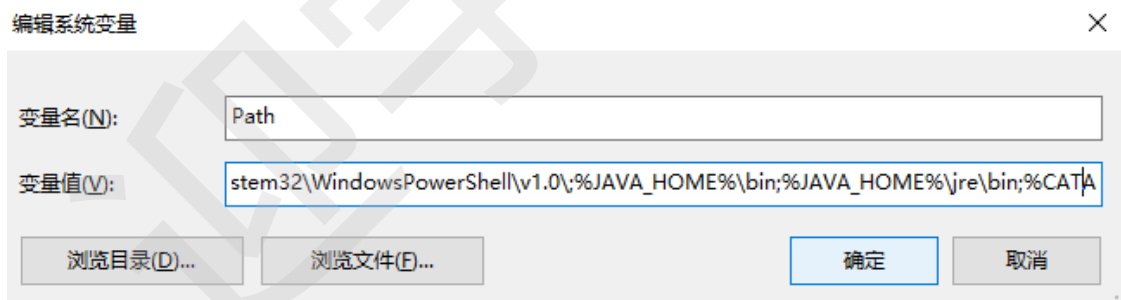


4. 系统变量→新建 JAVA\_HOME 变量。变量值填写jdk的安装目录（本人是 H:\Program Files\Java\jdk1.8.0\_131）

5. 系统变量→寻找 Path 变量→编辑

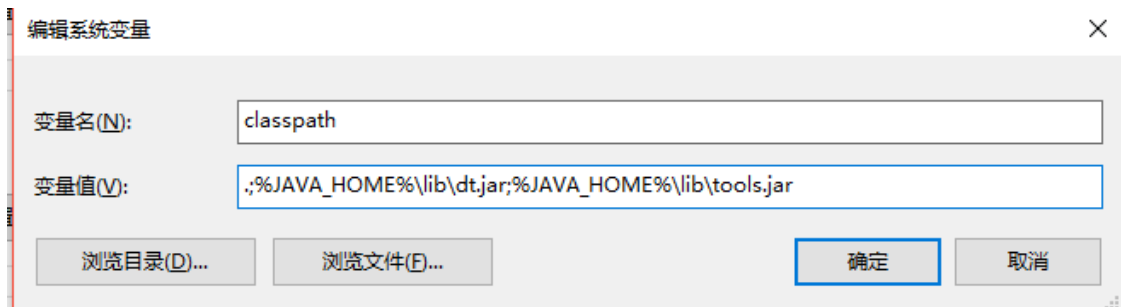
在变量值最后输入 ;%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin

（注意原来Path的变量值末尾有没有;号，如果没有，先输入；号再输入上面的代码）



6. 系统变量→新建 CLASSPATH 变量

变量值填写 .;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar（注意最前面有一点）



7. 检验是否配置成功 运行cmd 输入 java -version（java 和 -version 之间有空格）

若如图所示 显示版本信息 则说明安装和配置成功

```
C:\Users\Administrator>java -version
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
C:\Users\Administrator>
```

## 1.2 Tomcat 安装及配置教程

1. 虽然现在spring boot 内置了tomcat，运行时候不需要单独部署到外部容器，但是后续的课程需要测试用所以这里也顺带安装

此电脑 > 新加卷 (H:) > Spring Boot > 上课常用软件 > 常用软件

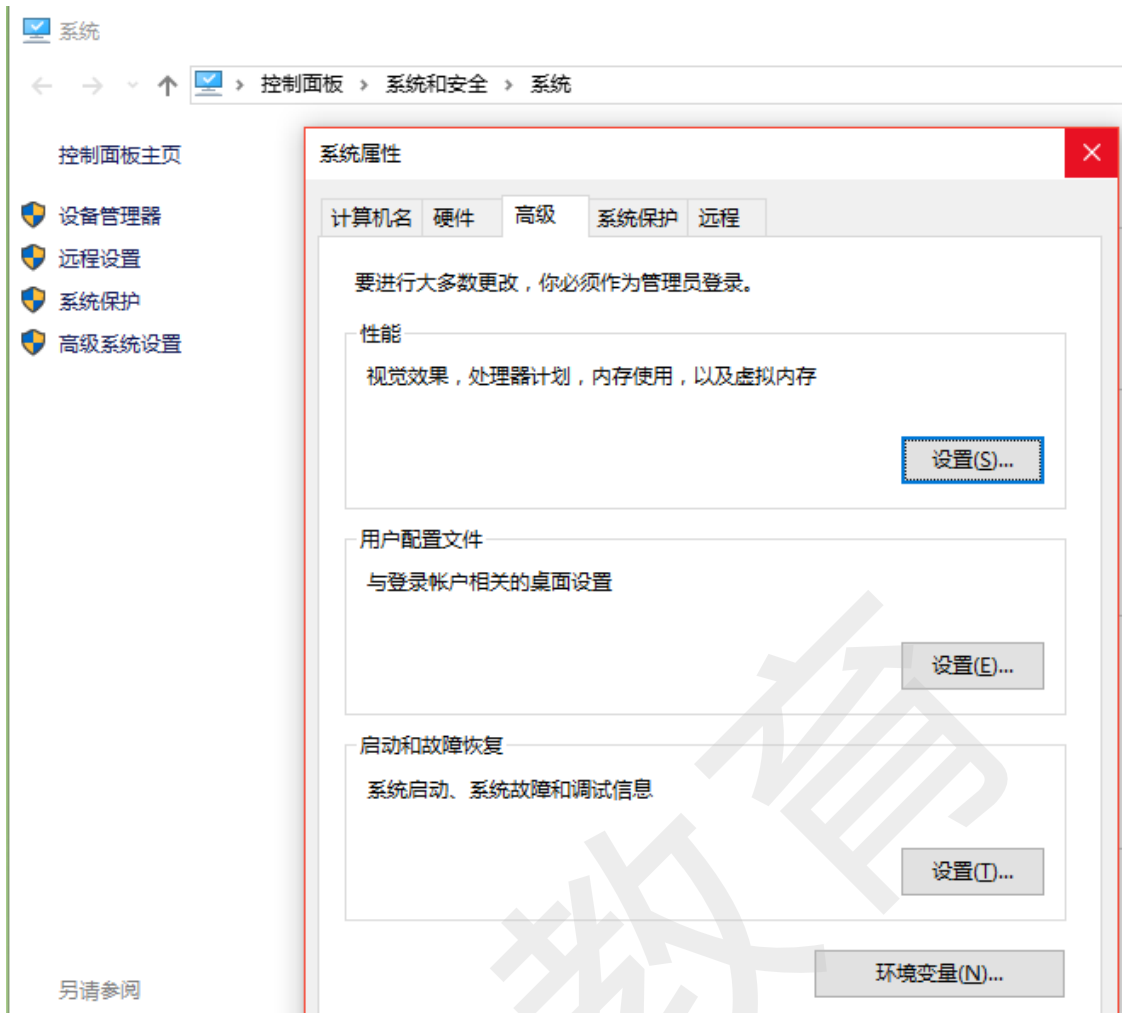
名称	修改日期	类型	大小
apache-activemq-5.9.0-bin	2017/6/22 11:08	360压缩 ZIP 文件	60,428 KB
apache-maven-3.3.9	2019/6/30 19:51	360压缩 ZIP 文件	8,424 KB
apache-tomcat-8.5.15 windows-x64	2017/5/25 16:05	360压缩 ZIP 文件	10,757 KB
ideaU-2017.1.3	2017/5/25 14:41	应用程序	510,526 KB
idea注册码	2019/6/30 20:00	文本文档	1 KB

2. 解压到相应目录(本人是 D:\apache-tomcat-8.5.15)

此电脑 > 本地磁盘 (D:) > apache-tomcat-8.5.15 >

名称	修改日期	类型	大小
bin	2017/11/30 22:48	文件夹	
conf	2017/5/25 16:10	文件夹	
lib	2017/5/25 16:07	文件夹	
logs	2019/7/13 10:41	文件夹	
temp	2017/5/29 16:13	文件夹	
webapps	2019/6/30 20:23	文件夹	
work	2017/5/25 16:10	文件夹	

3. 解压完 Tomcat 后配置环境变量 计算机→属性→高级系统设置→高级→环境变量



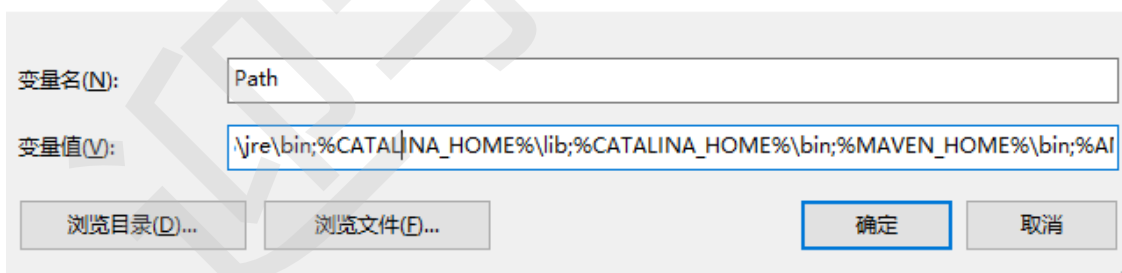
4. 系统变量→新建 CATALINA\_HOME 变量。变量值填写Tomcat的安装目录（本人是 D:\apache-tomcat-8.5.15）

5. 系统变量→寻找 Path 变量→编辑

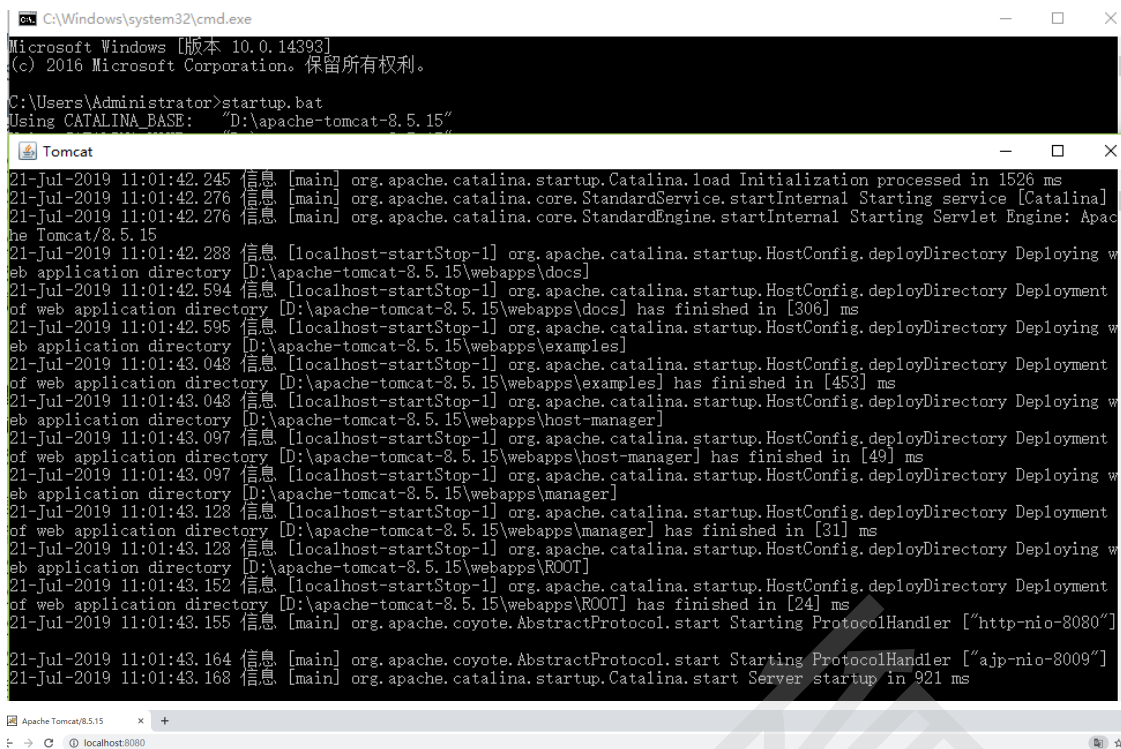
在变量值最后输入 ;%CATALINA\_HOME%\bin

（注意原来Path的变量值末尾有没有;号，如果没有，先输入；号再输入上面的代码）

编辑系统变量



6. 这样 Tomcat 就配置好了。打开 cmd 命令提示符，输入 startup.bat 后回车，就可以看到 Tomcat 成功启动了



### 1.3 Maven 安装及配置教程

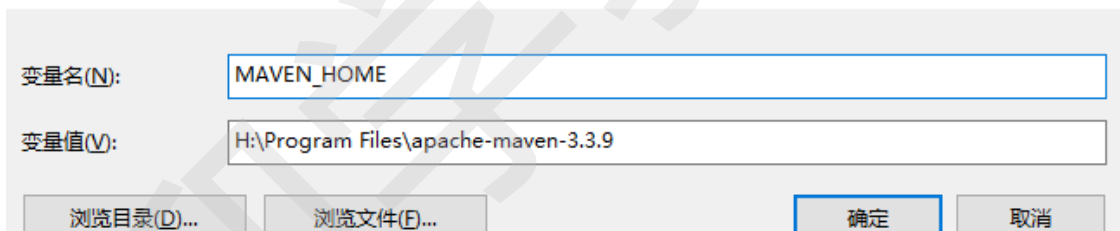
1. 我这里用的是 maven 3.3.9

此电脑 > 新加卷 (H:) > Spring Boot > 上课常用软件 > 常用软件				
名称	修改日期	类型	大小	
apache-activemq-5.9.0-bin	2017/6/22 11:08	360压缩 ZIP 文件	60,428 KB	
apache-maven-3.3.9	2019/6/30 19:51	360压缩 ZIP 文件	8,424 KB	
apache-tomcat-8.5.15-windows-x64	2017/5/25 16:05	360压缩 ZIP 文件	10,757 KB	

2. 解压到特定目录(本人是 H:\Program Files\apache-maven-3.3.9)
3. 安装完 maven 后配置环境变量 计算机→属性→高级系统设置→高级→环境变量



4. 系统变量→新建 MAVEN\_HOME 变量。变量值填写Tomcat的安装目录（本人是 H:\Program Files\apache-maven-3.3.9）  
编辑系统变量

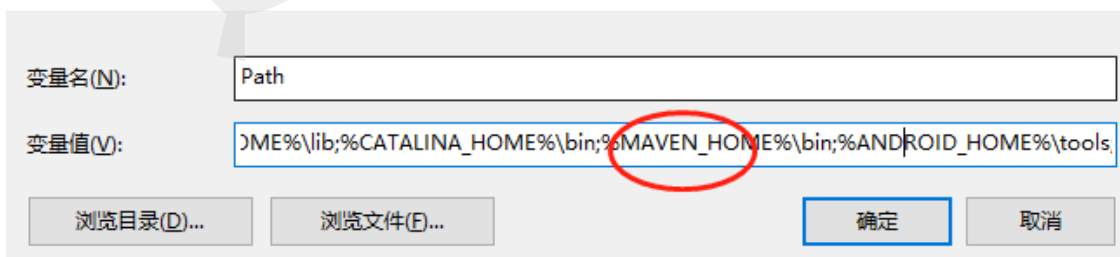


5. 系统变量→寻找 Path 变量→编辑

在变量值最后输入;%MAVEN\_HOME%\bin

（注意原来Path的变量值末尾有没有;号，如果没有，先输入;号再输入上面的代码）

编辑系统变量



6. 这样Maven就配置好了 打开---cmd---输入mvn -version （ mvn和 -version 之间有空格）

如下图说明配置成功

```
Using CLASSPATH: D:\apache-tomcat-8.5.15\bin\bootstrap.jar;D:\apache-tomcat-8.5.15\bin\tomcat-juli.jar
C:\Users\Administrator>mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T00:41:47+08:00)
Maven home: H:\Program Files\apache-maven-3.3.9\bin\..
Java version: 1.8.0_131, vendor: Oracle Corporation
Java home: H:\Program Files\Java\jdk1.8.0_131\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos"
```

7. 修改本地仓库位置(打开maven解压目录----conf---找到setting.xml)

```
| values (values used when the setting is not specified) are provided.
|
|--->
<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>
  </-->
  </-->
```

(一般默认情况下Maven在本机的仓库位于C:\Users\你的电脑用户名\m2\repository)

8. 为了缓解c盘的的压力我们可以自定义配置本地仓库目录 F:\mvnrepository(本人是F:\mvnrepository)

```
|--->
<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.xs
  <!-- 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>F:\mvnrepository</localRepository>
  <!-- interactiveMode
  | This will determine whether maven prompts you when it needs input. If set to false,
  | maven will use a sensible default value, perhaps based on some other setting, for
  | the interactive mode.
```

加上这句话即可

9. 设置镜像我们用Maven的时候，因为Maven自带的远程中央仓库在国外，所以经常会很慢。我们可以把远程中央仓库改为国内阿里的远程仓库。

还是找到setting.xml----打开-----找到mirros节点 咱贴

```
</mirror>
  <mirror>
    <id>alimaven</id>
    <name>aliyun maven</name>
    <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    <mirrorOf>central</mirrorOf>
  </mirror>
```

```
|--->
<mirrors>
  <!-- mirror
  | Specifies a repository mirror site to use instead of a given repository. The repository that
  | this mirror serves has an ID that matches the mirrorOf element of this mirror. IDs are used
  | for inheritance and direct lookup purposes, and must be unique across the set of mirrors.
  |
  <!-->
  <mirror>
    <id>mirrorId</id>
    <mirrorOf>repositoryId</mirrorOf>
    <name>Human Readable Name for this Mirror.</name>
    <url>http://my.repository.com/repo/path</url>
  </mirror>
  <mirror>
    <id>alimaven</id>
    <name>aliyun maven</name>
    <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    <mirrorOf>central</mirrorOf>
  </mirror>
</mirrors>
```

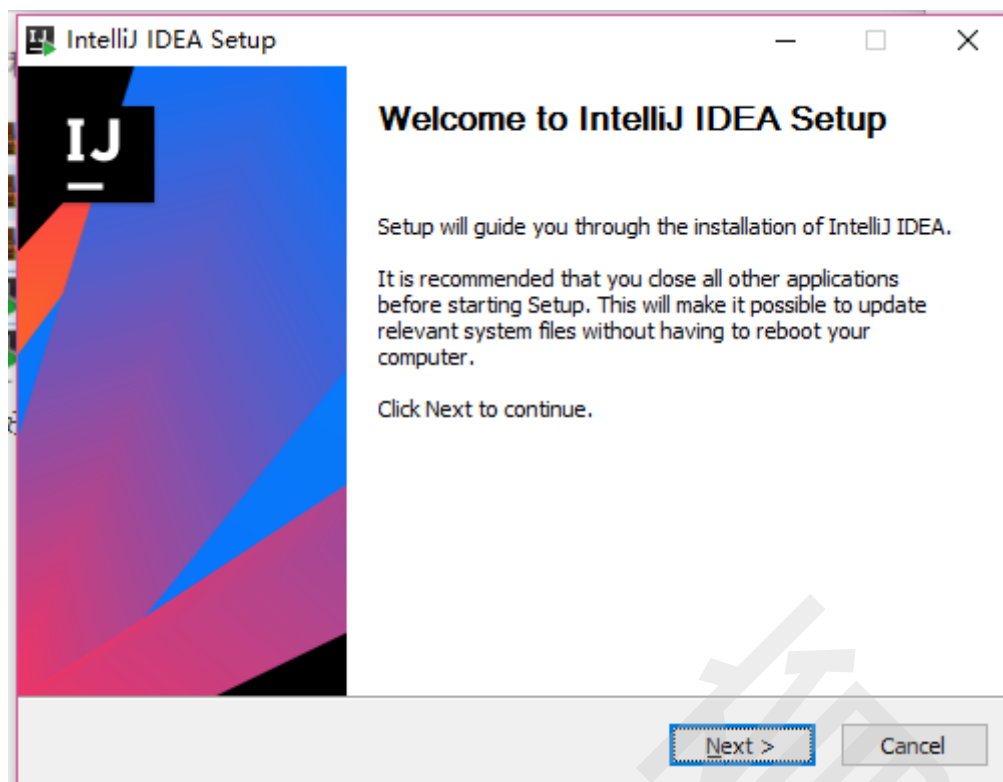
## 1.4 Idea 安装及配置教程

1. idea 是一款你用了就不想换的编译器，你可以去[官网](#)下载  
这里我用的是 18 年

此电脑 > 新加卷 (H:) > Spring Boot > 上课常用软件 > 常用软件

名称	修改日期	类型	大小
apache-activemq-5.9.0-bin	2017/6/22 11:08	360压缩 ZIP 文件	60,428 KB
apache-maven-3.3.9	2019/6/30 19:51	360压缩 ZIP 文件	8,424 KB
apache-tomcat-8.5.15-windows-x64	2017/5/25 16:05	360压缩 ZIP 文件	10,757 KB
ideaIU-2017.1.3	2017/5/25 14:41	应用程序	510,526 KB
ideaIU-2018.2.8	2019/7/21 17:13	应用程序	533,465 KB

2. 找到我们下载好的 .exe 执行文件，然后双击打开



3. 点击【next】下一步，进入选择安装目录，选择我们要安装的安装位置(本人是D:\idea\IntelliJ IDEA 2018.2.8)



**Choose Install Location**

Choose the folder in which to install IntelliJ IDEA.

Setup will install IntelliJ IDEA in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

## Destination Folder

Space required: 1.2 GB

Space available: 32.3 GB

&lt; Back

Next &gt;

Cancel

**Installation Options**

Configure your IntelliJ IDEA installation

## Create Desktop Shortcut

☐ 32-bit launcher ☒ 64-bit launcher

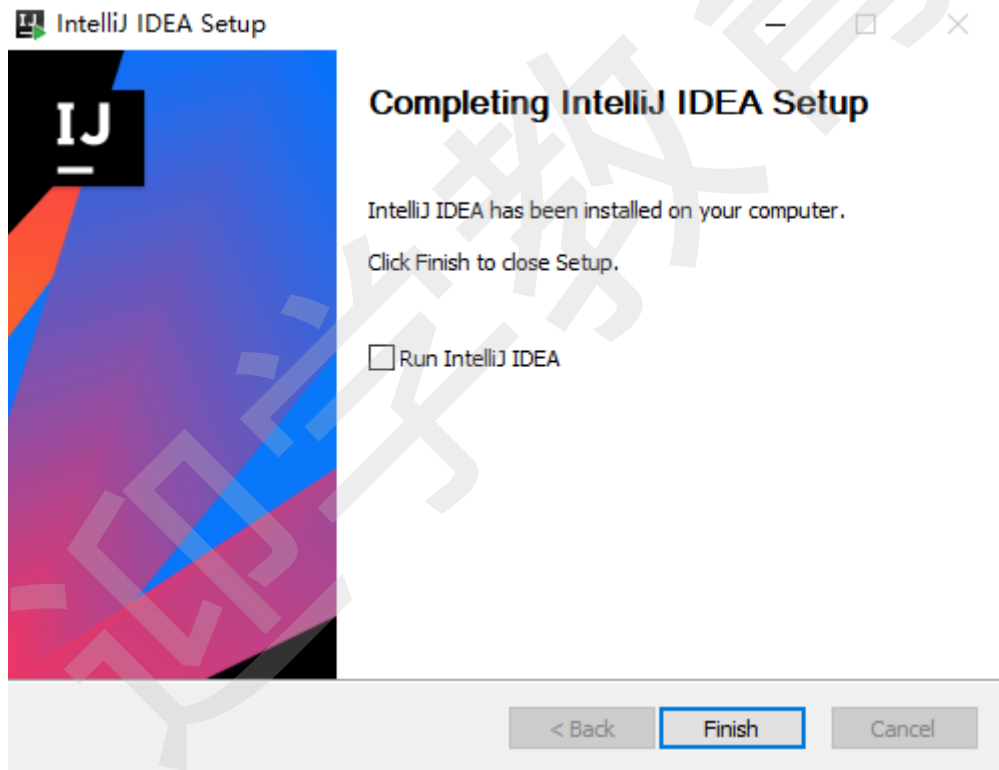
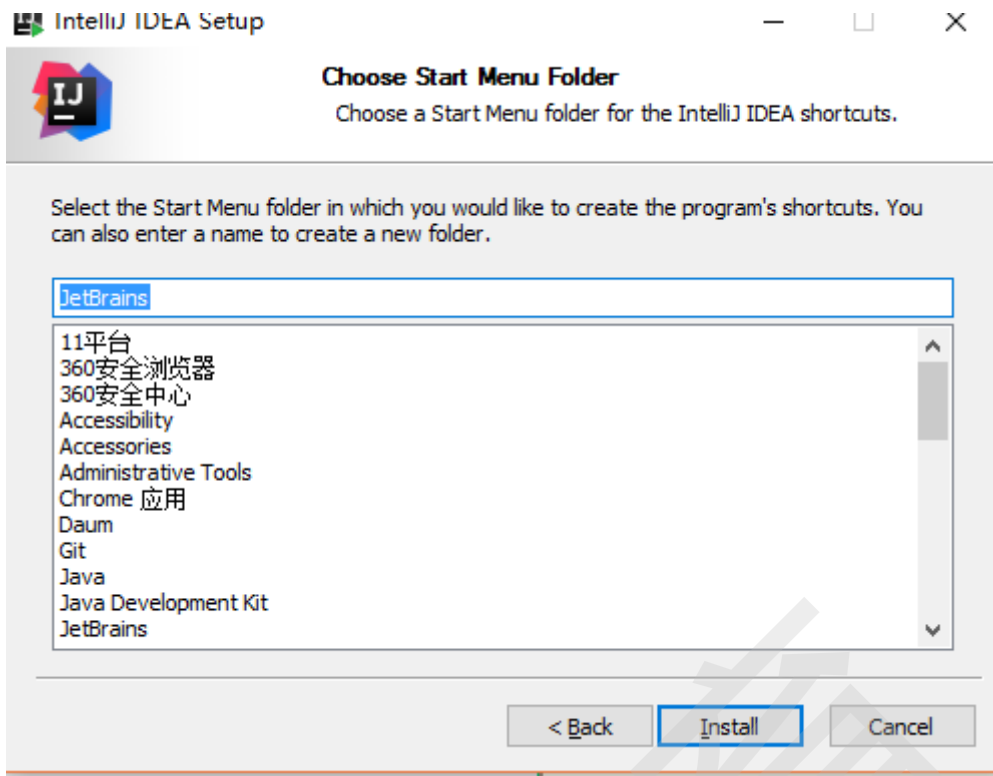
## Create Associations

☒ .java ☐ .groovy ☐ .kt☐ Download and install JRE x86 by JetBrains

&lt; Back

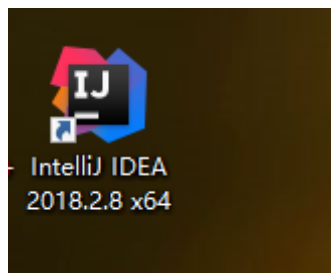
Next &gt;

Cancel

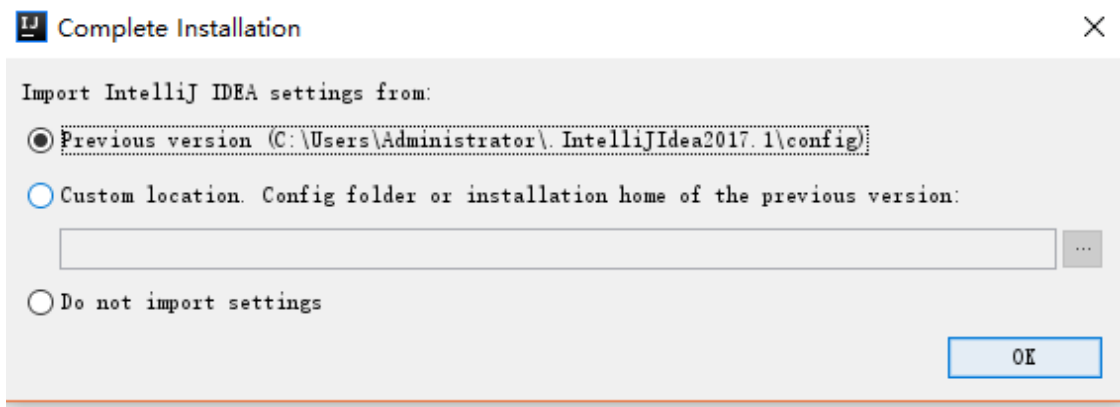


点击Finish，就安装好了。

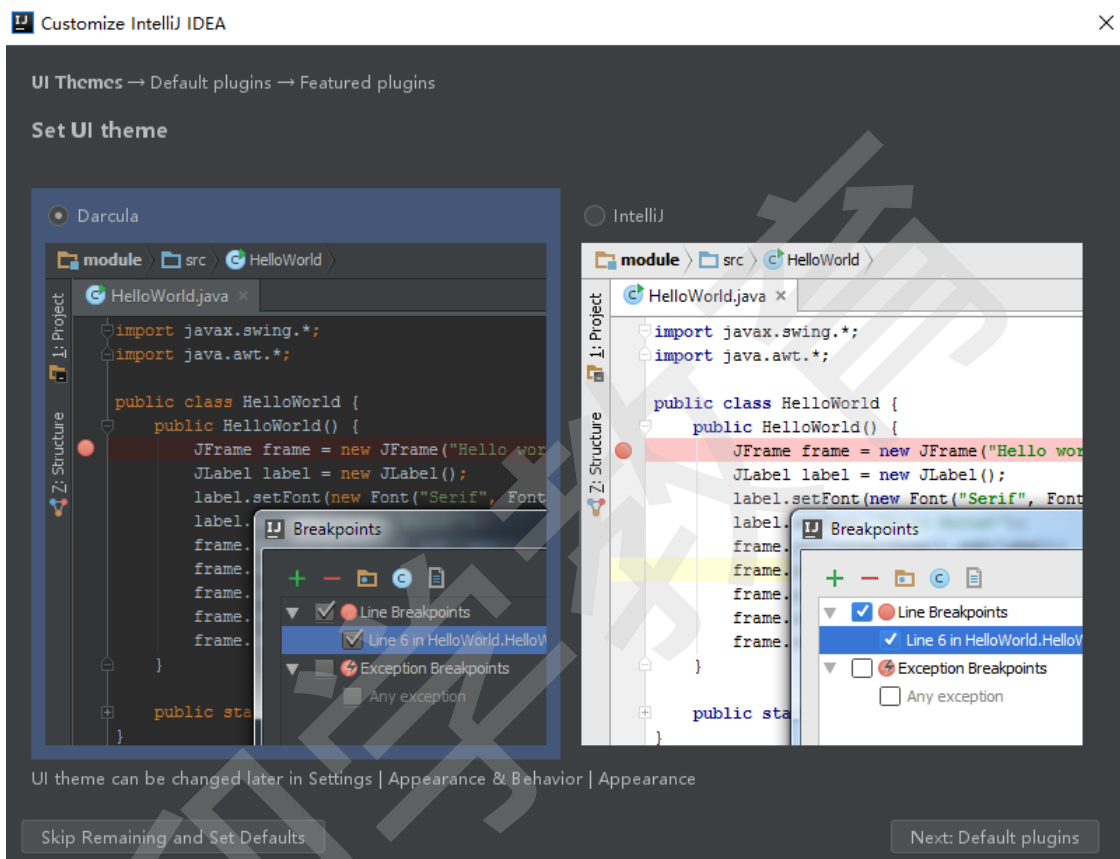
这是会在我的桌面生成一个64位的快捷方式：



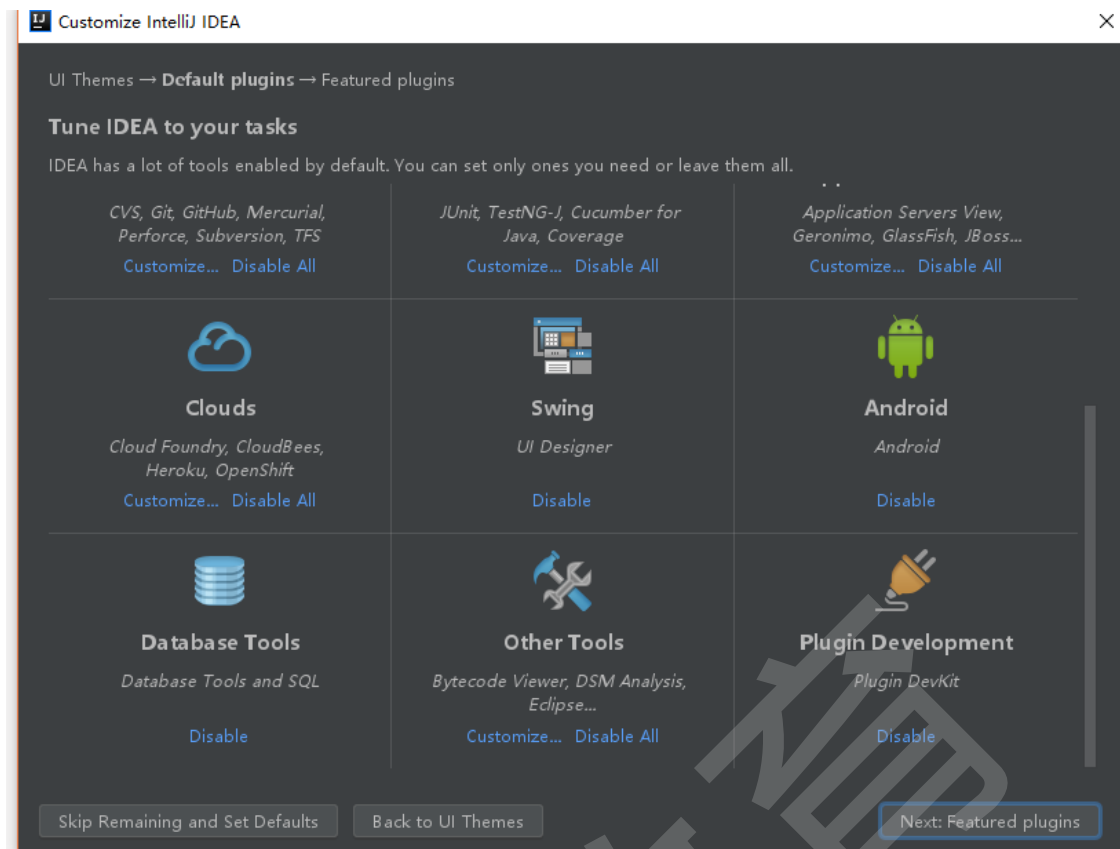
4. 到现在，我们的安装过程就讲完了，我们讲解下如何配置IDEA，假如已经有配置的话，可以直接导入之前的配置（或者不选择导入以前配置选择第三个 Do not import settings）一路默认安装即可



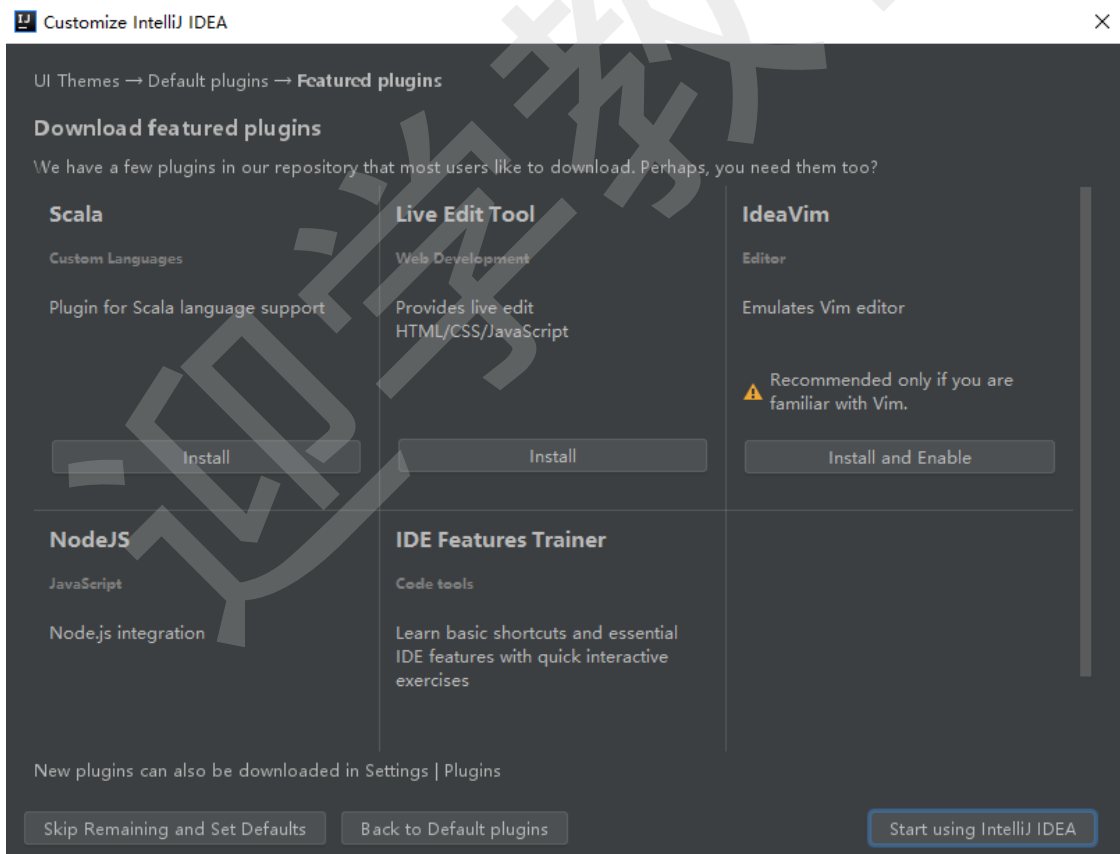
5. 选择默认皮肤



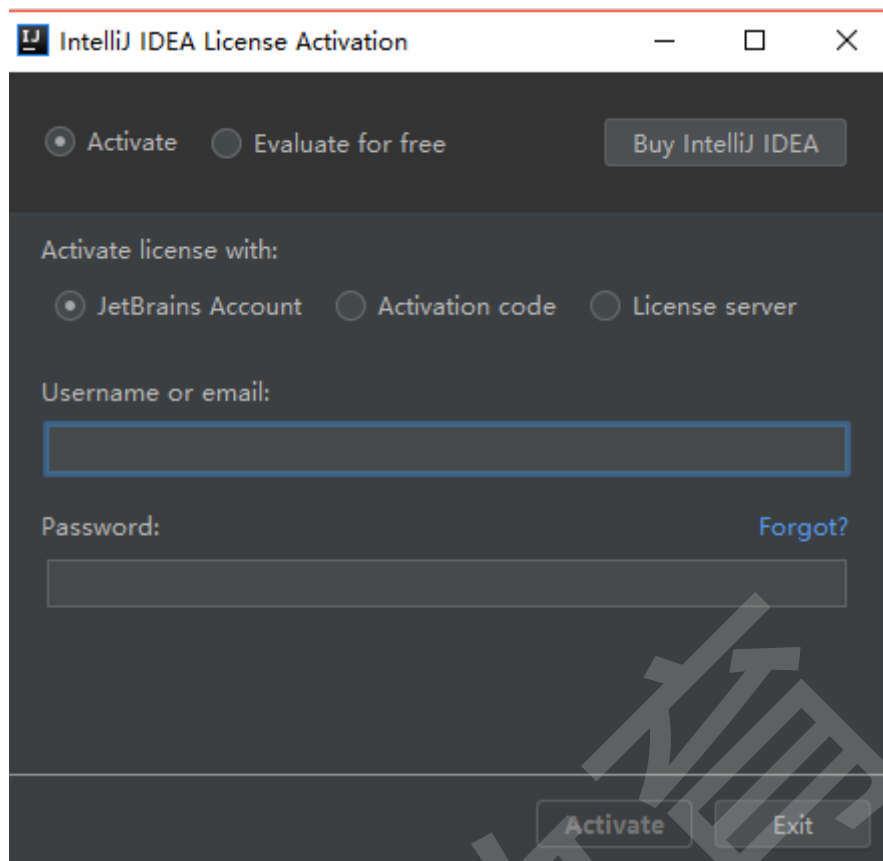
6. 下一步，默认插件



7. 下一步，功能插件



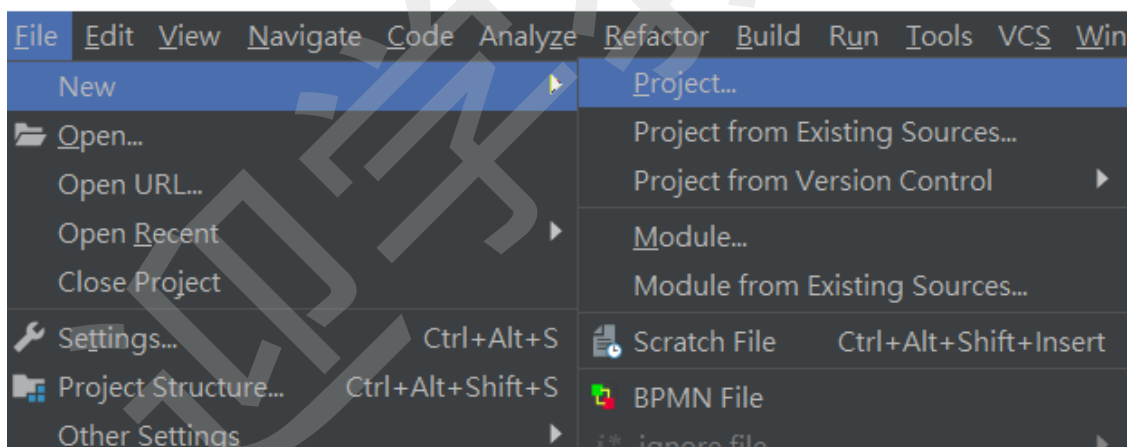
8. 注册方法和注册码([https://blog.csdn.net/gnail\\_oung/article/details/53977118](https://blog.csdn.net/gnail_oung/article/details/53977118)) 这个博客超详细跟着配置就可以使用了

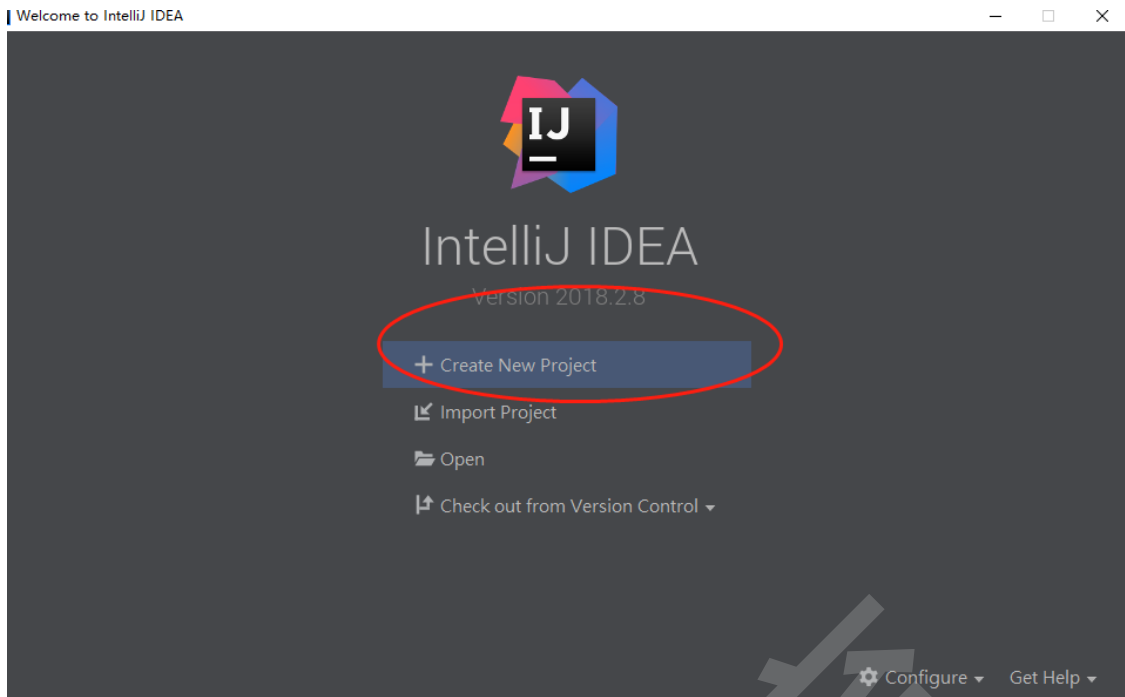


## 2. maven 手工创建 Spring Boot2.x web应用

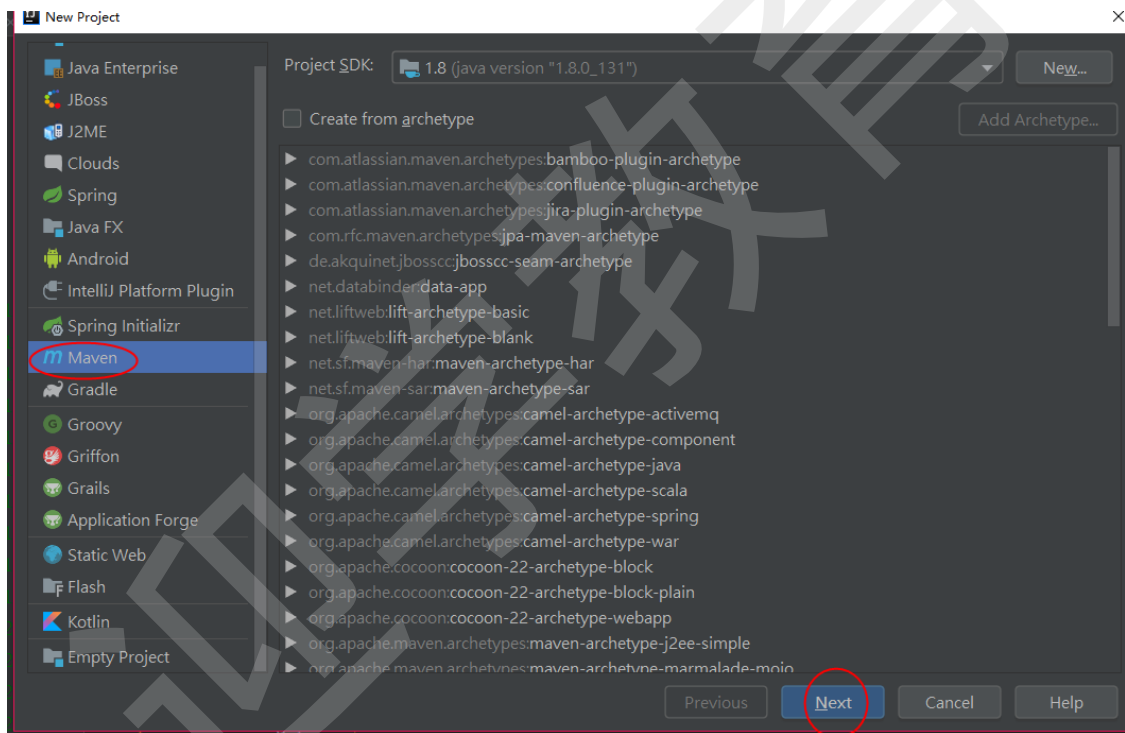
### 2.1 创建工程

1. 创建一个新的工程

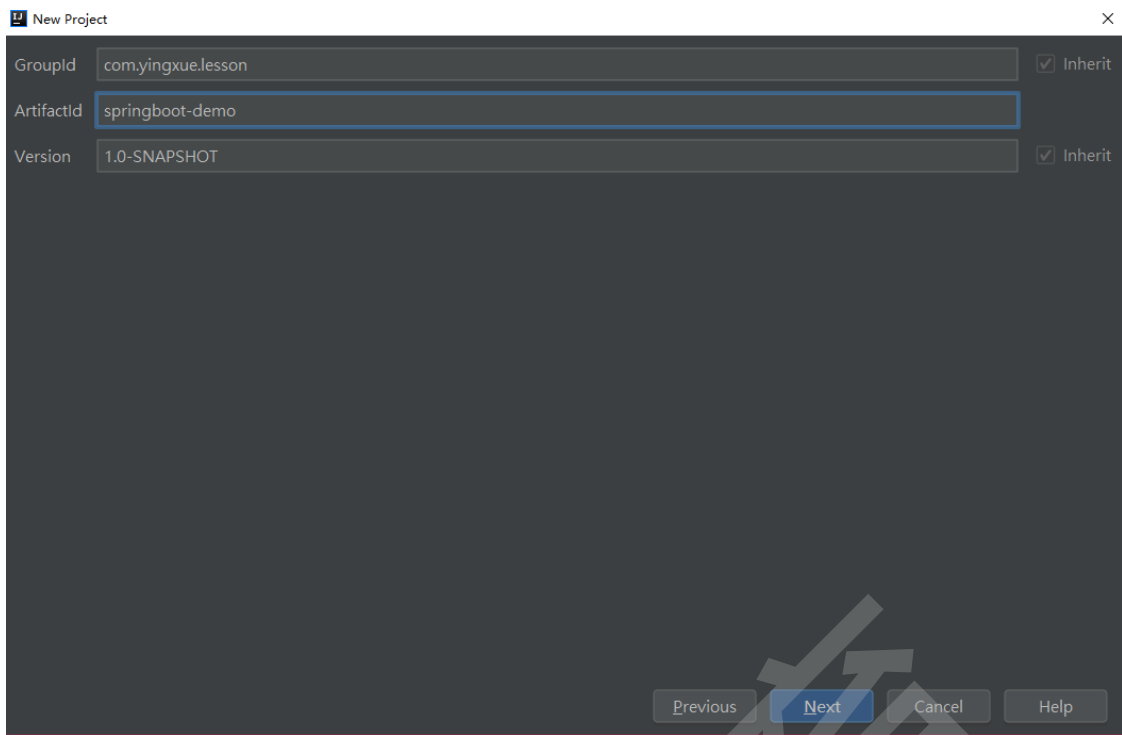




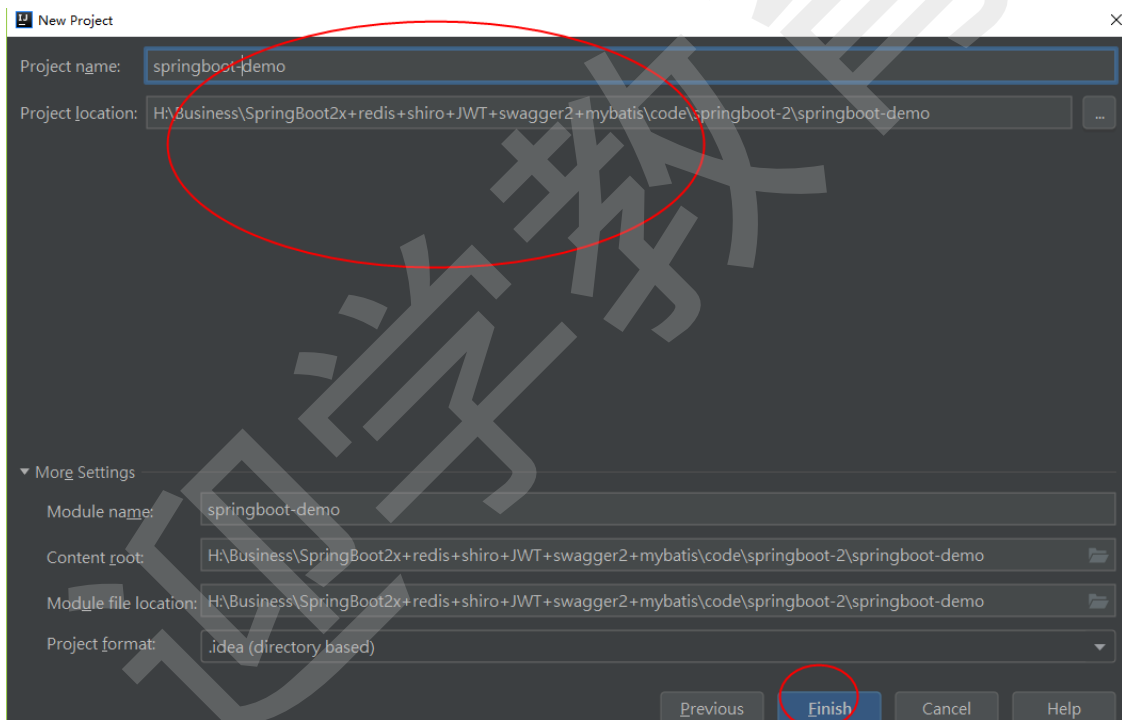
## 2. 使用 maven 来构建



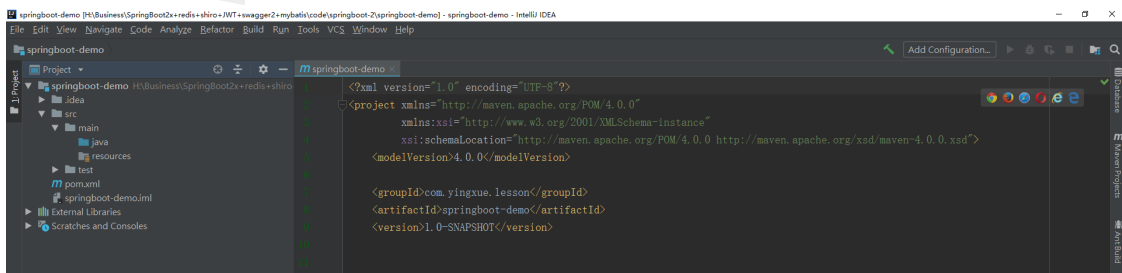
## 3. 然后填写坐标



#### 4. 目录结构



#### 5. 项目结构



## 2.2 添加依赖

### 2.2.1 添加父工程坐标

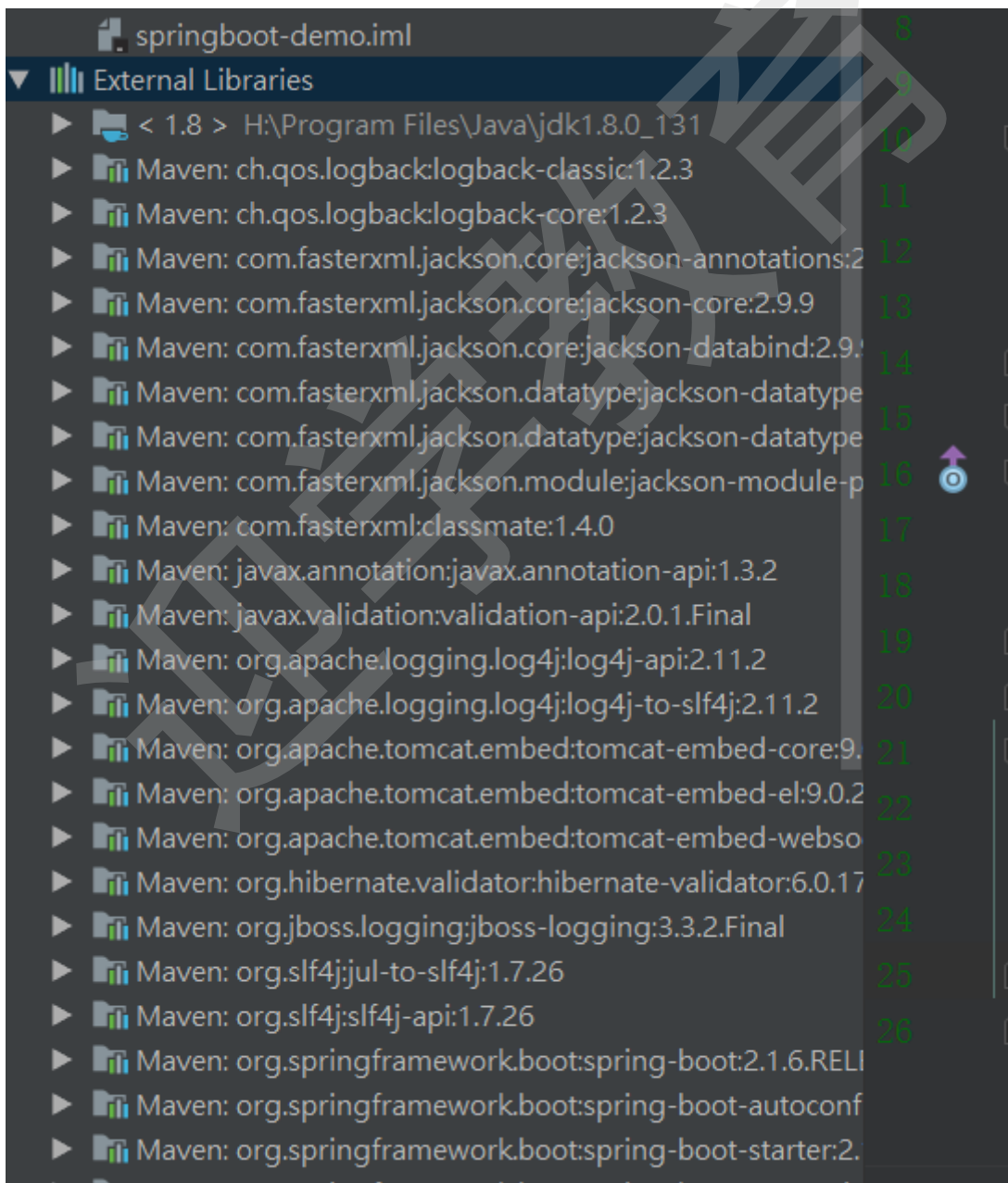
```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.6.RELEASE</version>
</parent>
```

### 2.2.2 添加web启动器

为了让 SpringBoot 帮我们完成各种自动配置，我们必须引入 SpringBoot 提供的自动配置依赖，我们称为启动器。因为我们是 web 项目，这里我们引入 web 启动器：

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
```

需要注意的是，我们并没有在这里指定版本信息。因为SpringBoot的父工程已经对版本进行了管理了。这个时候，我们会发现项目中多出了大量的依赖：



这些都是 SpringBoot 根据spring-boot-starter-web 这个依赖自动引入的，而且所有的版本都已经管理好，不会出现冲突。

### 2.2.3 添加单元测试



```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

#### 2.2.4 管理jdk版本

默认情况下，maven工程的jdk版本是1.5，而我们开发使用的是1.8，因此这里我们需要修改jdk版本，只需要简单的添加以下属性即可：

```
<properties>
  <java.version>1.8</java.version>
</properties>
```

#### 2.2.5 完整的pom

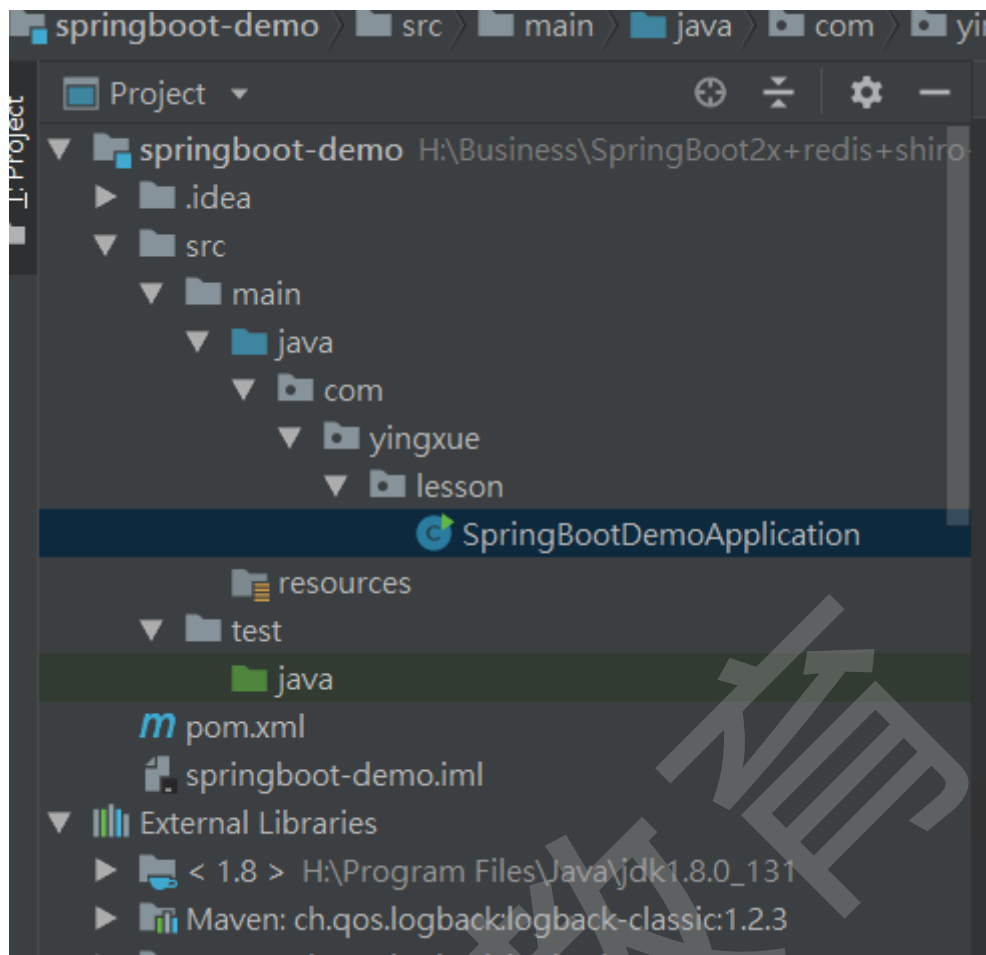
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.yingxue.lesson</groupId>
  <artifactId>springboot-demo</artifactId>
  <version>1.0-SNAPSHOT</version>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.6.RELEASE</version>
  </parent>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

### 2.3 启动类

Spring Boot项目通过main函数即可启动，我们需要创建一个启动类：

注意这个启动类必须放在包目录下



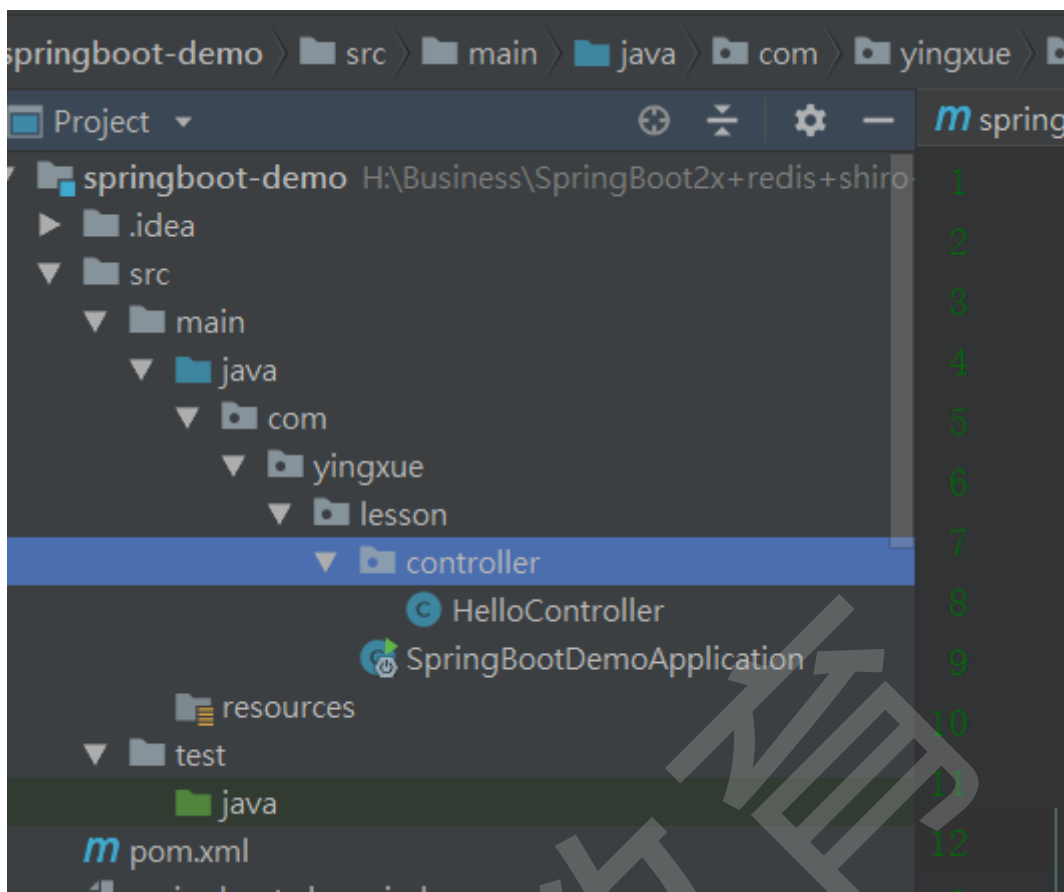
然后编写main函数

```
package com.yingxue.lesson;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootDemoApplication.class, args);
    }
}
```

## 2.4 编写测试类 controller

1. 创建 HelloController.java



```
package com.yingxue.lesson.controller;

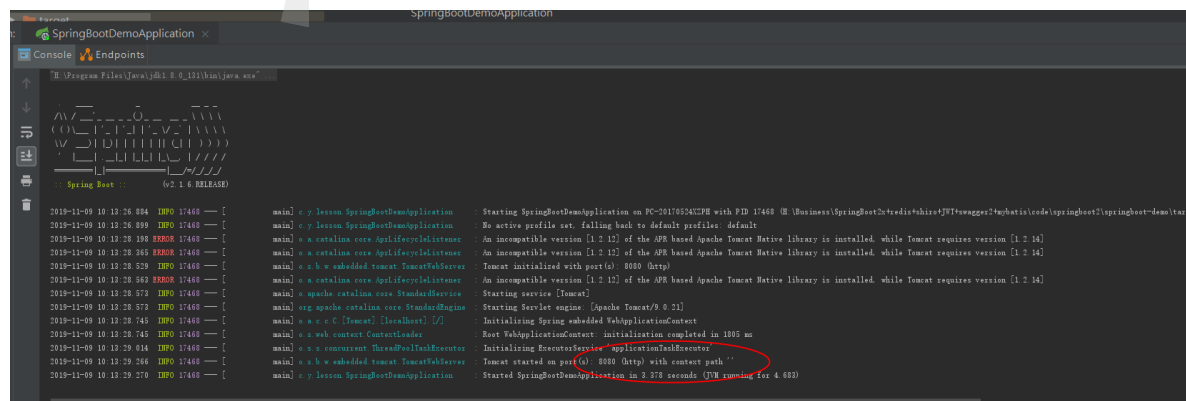
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/hello")
public class HelloController {

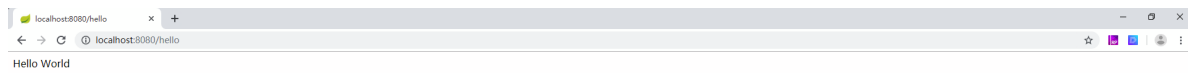
    @GetMapping("/hello")
    public String helloWorld(){
        return "Hello World";
    }
}
```

## 2.5 启动测试

接下来，我们运行main函数，查看控制台：



浏览器访问：<http://localhost:8080/hello>

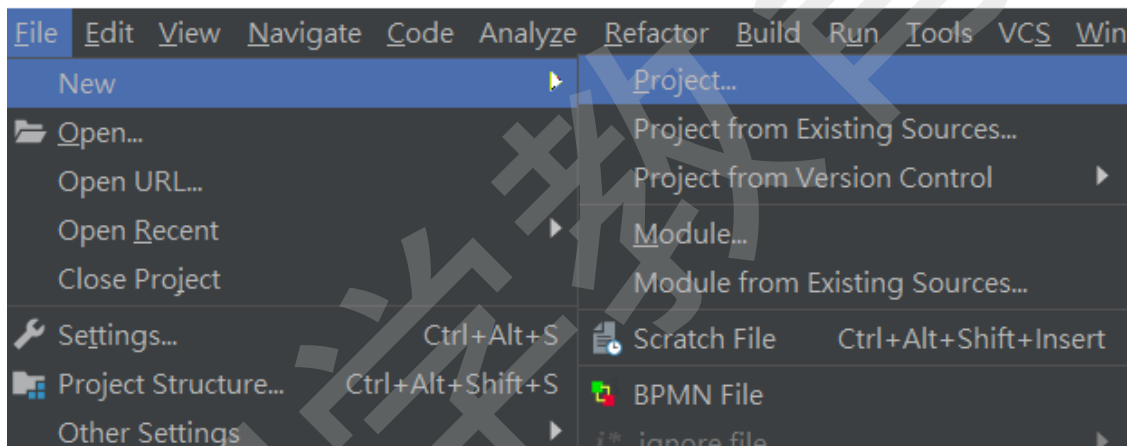


启动测试成功

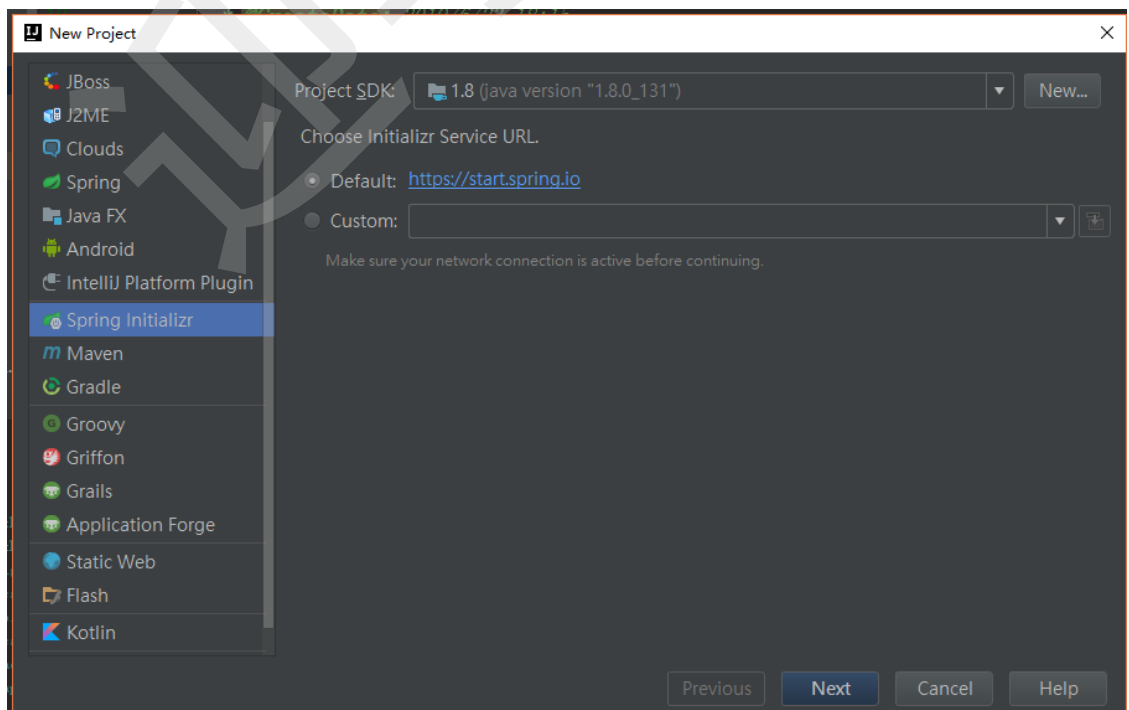
- @RestController 注解相当于 @Controller+@ResponseBody 合在一起的作用，如果 Web 层的类上使用了 @RestController 注解，就代表这个类中所有的方法都会以 JSON 的形式返回结果，也相当于JSON 的一种快捷使用方式。
- @GetMapping("/hello")是一个组合注解 等于@RequestMapping(name="/hello",method = RequestMethod.GET)的缩写，以 /hello的方式去请求， 如果@PostMapping 则为method= RequestMethod.POST 是指只可以使用 Post 的方式去请求，如果使用 Get 的方式去请求的话，则会报 405 不允许访问的错误。 @PutMapping、@DeleteMapping 则 method 会以此类推。

### 3 Spring工具创建Spring Boot2.x web应用

1. 创建一个新的工程

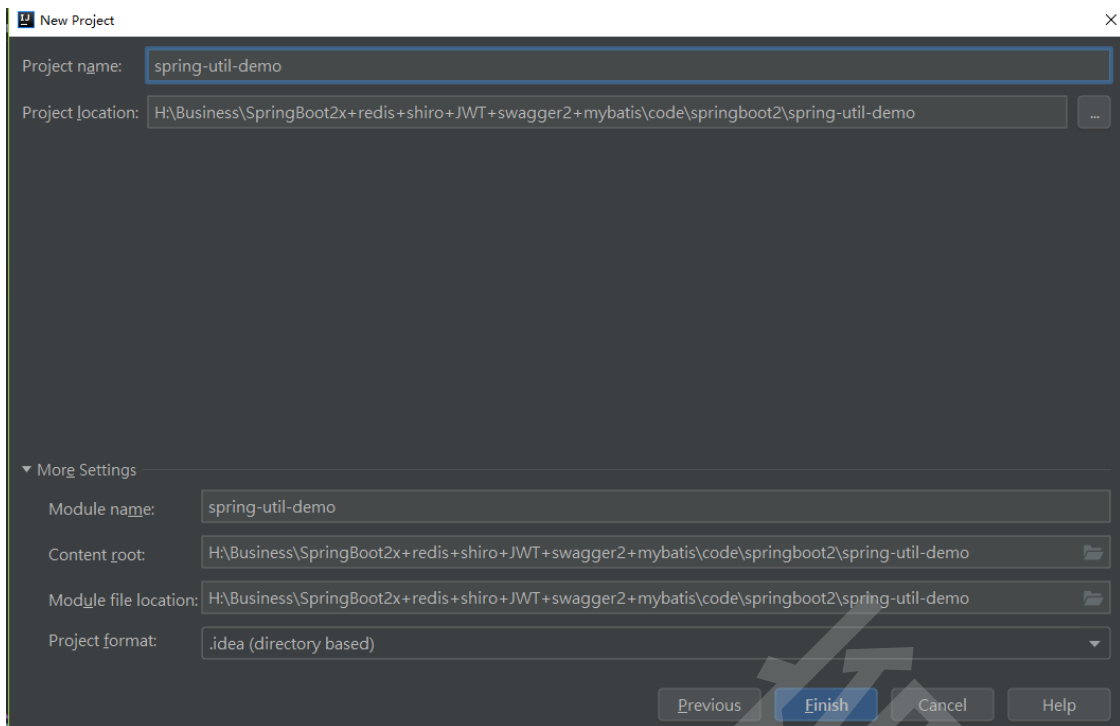


2. 使用 Spring Initializr 来构建

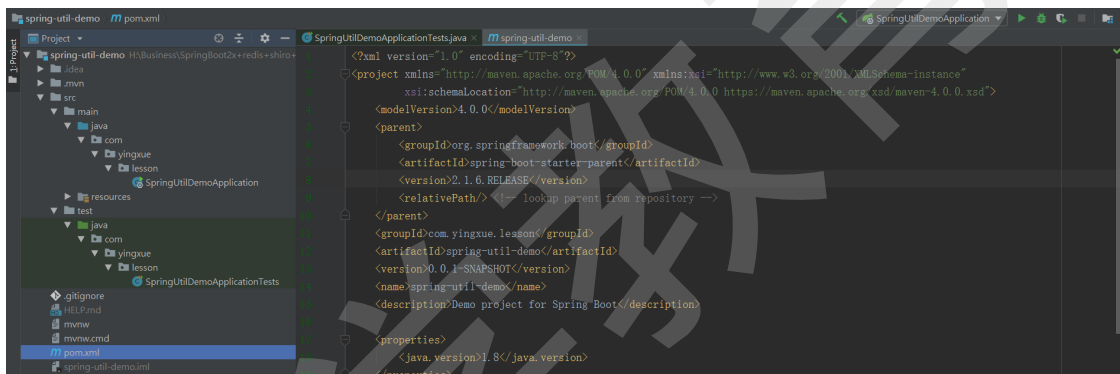


3. 填写坐标





## 6. 完整的目录如下



- o pom文件为基本的依赖管理文件
- o resources 资源文件
- o static 静态资源
- o templates 模板资源
- o application.properties 配置文件
- o SpringInitializrDemoApplication 程序的入口。

## 7. 完整的pom

```
<?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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.6.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.yingxue.lesson</groupId>
    <artifactId>spring-util-demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-util-demo</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
```

```

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

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>

```

## 4. maven-多模块 ( module ) 项目搭建教程

很多同学在搭建maven 多模块项目过程中经常会遇到jar包冲突、子类引入不进jar包，归其原因是分不清楚dependencies与dependencyManagement的区别。

### 4.1 DependencyManagement & dependencies区别

**dependencies**：即使在子项目中不写该依赖项，那么子项目仍然会从父项目中继承该依赖项（全部继承）

**dependencyManagement**：里只是声明依赖，并不实现引入，因此子项目需要显示的声明需要用的依赖。如果不在子项目中声明依赖，是不会从父项目中继承下来的；只有在子项目中写了该依赖项，并且没有指定具体版本，才会从父项目中继承该项，并且version和scope都读取自父pom;另外如果子项目中指定了版本号，那么会使用子项目中指定的jar版本。

### 4.2 工程目录结构

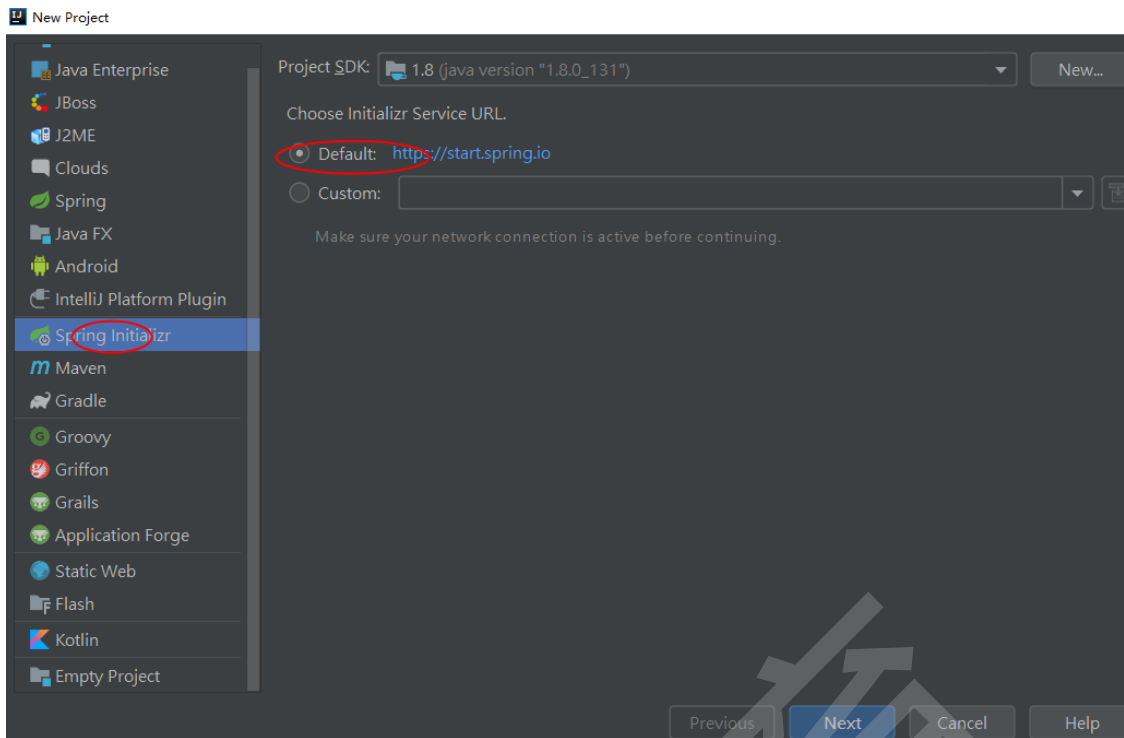
- yingxue-service：服务层
- yingxue-web：控制层

### 4.3 搭建步骤

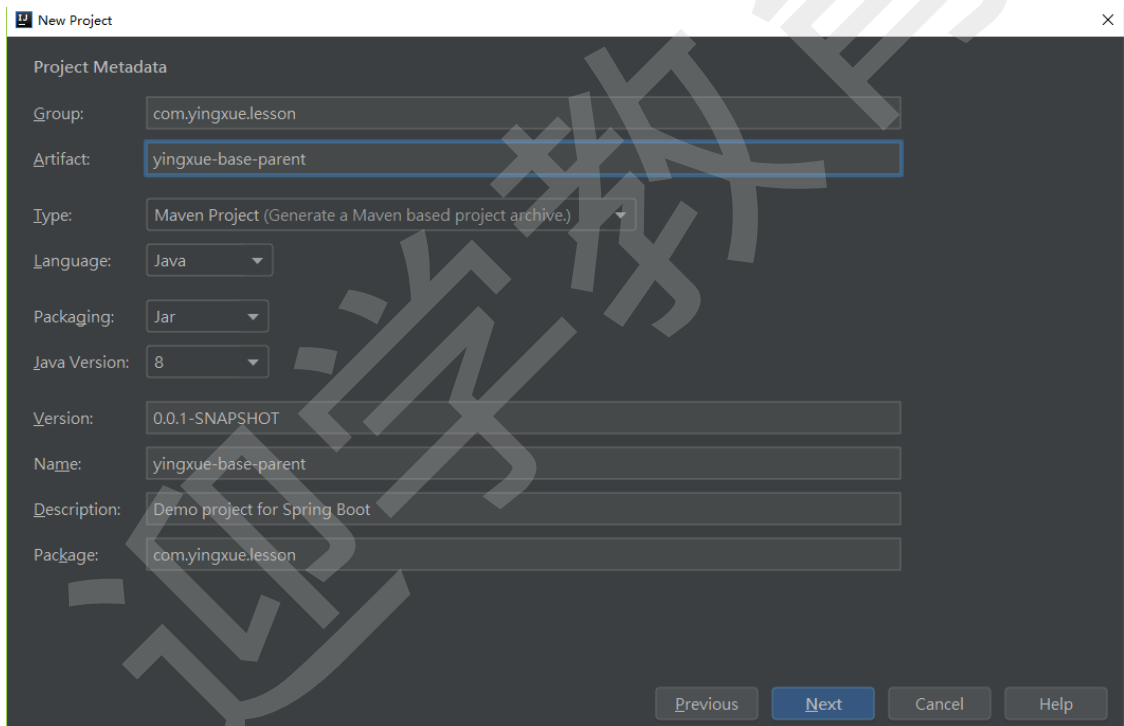
由于我们是多模块，所以我们抽出一个yingxue-base-parent来管理子项目的公共的依赖。在我们项目顶层的POM文件中，我们会看到dependencyManagement元素。通过它元素来管理jar包的版本，让子项目中引用一个依赖而不用显示的列出版本号。Maven会沿着父子层次向上走，直到找到一个拥有dependencyManagement元素的项目，然后它就会使用在这个dependencyManagement元素中指定的版本号。

#### 4.3.1 父类工程

1. IDEA 工具栏选择菜单 File -> New -> Project...
2. 选择Spring Initializr，Initializr默认选择Default，点击Next

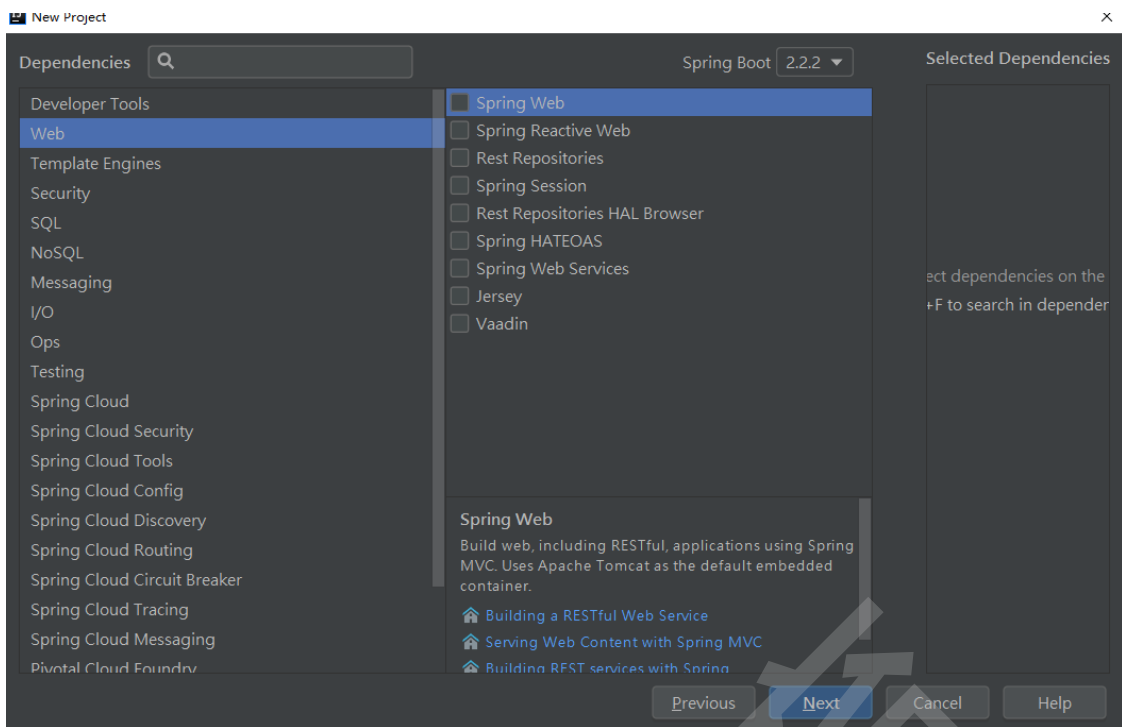


3. 填写坐标，点击Next

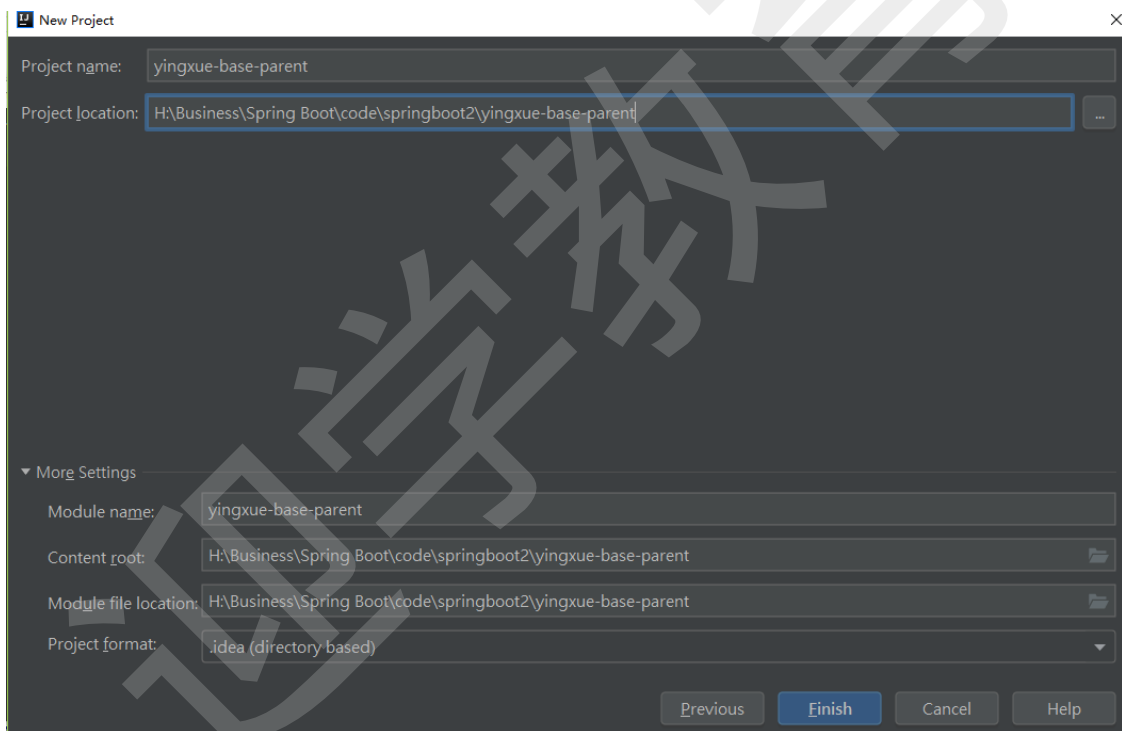


4. 这步不需要选择直接点Next

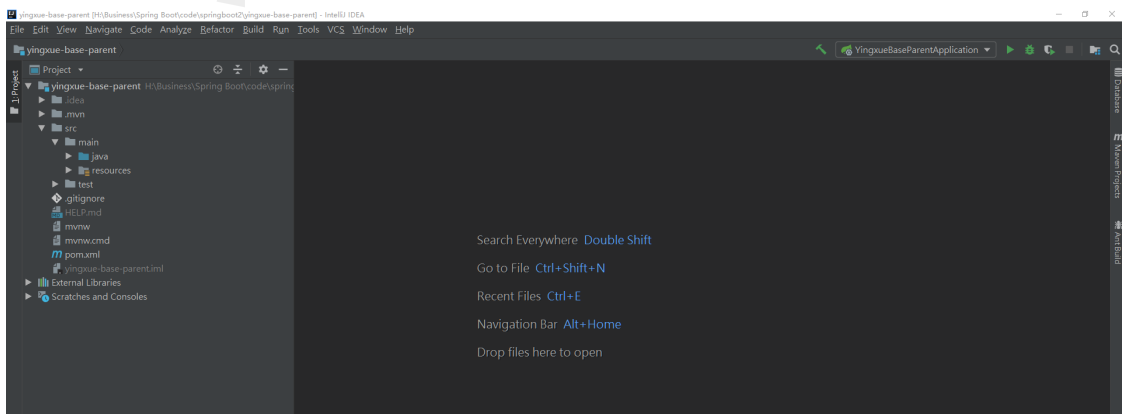




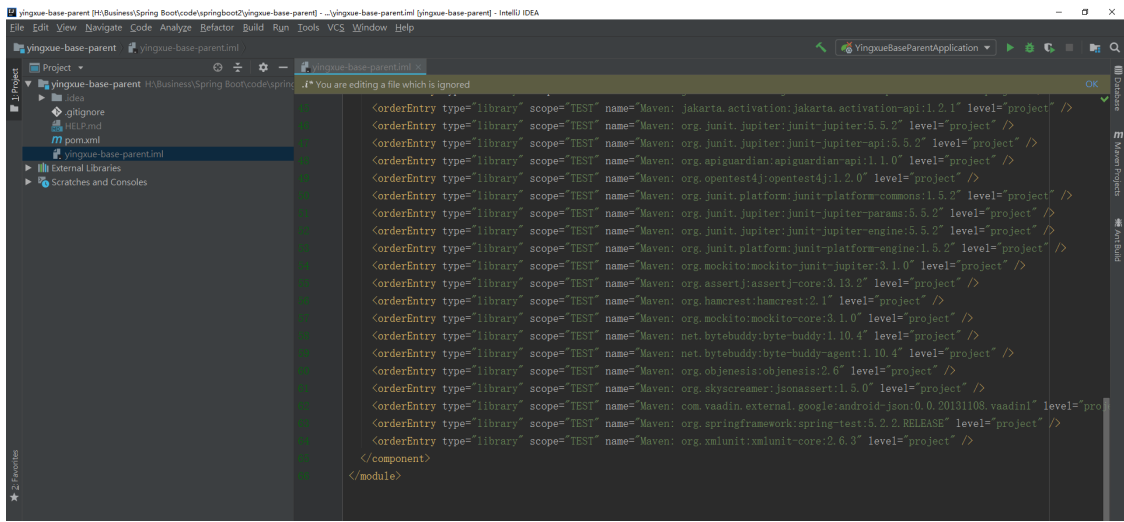
5. 点击Finish



6. 最终得到的项目目录结构如下

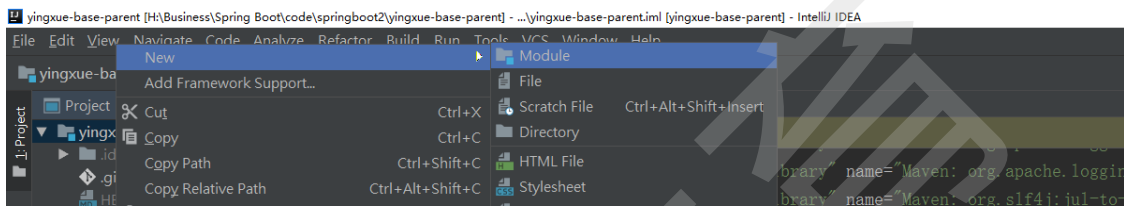


7. 删除无用的.mvn目录、src目录、mvnw及mvnw.cmd文件，最终只留.gitignore和pom.xml

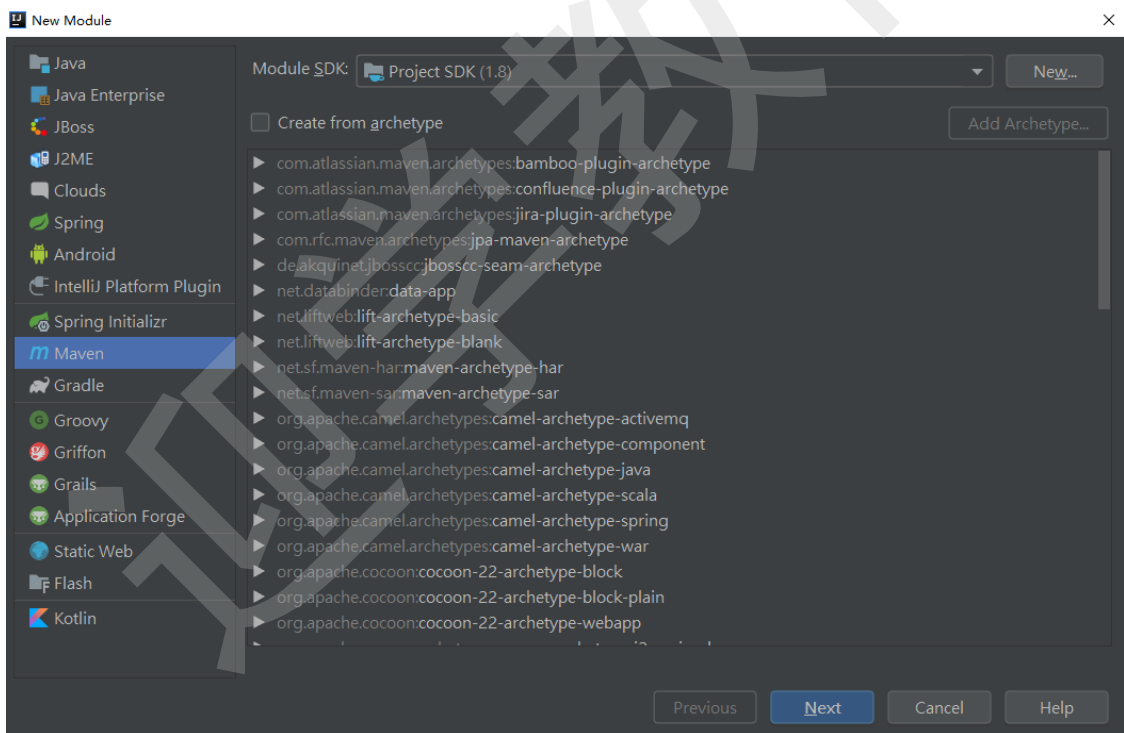


#### 4.3.2 子类工程

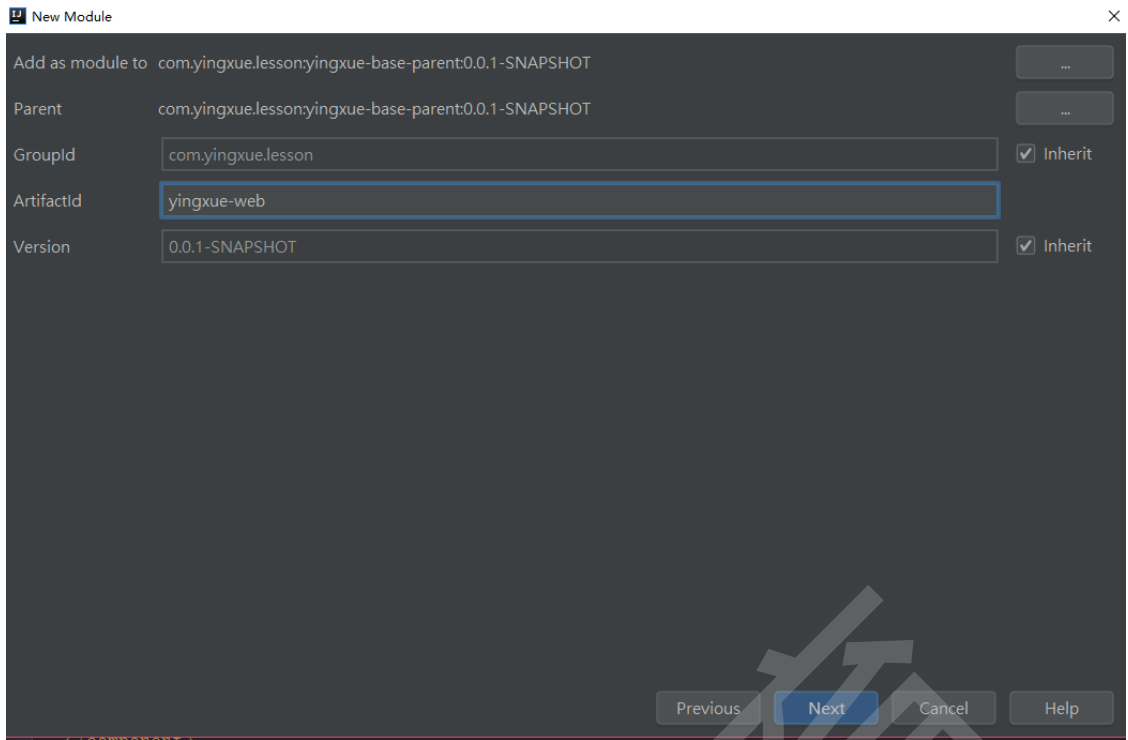
1. 选择项目根目录yingxue-base-parent右键呼出菜单，选择New -> Module



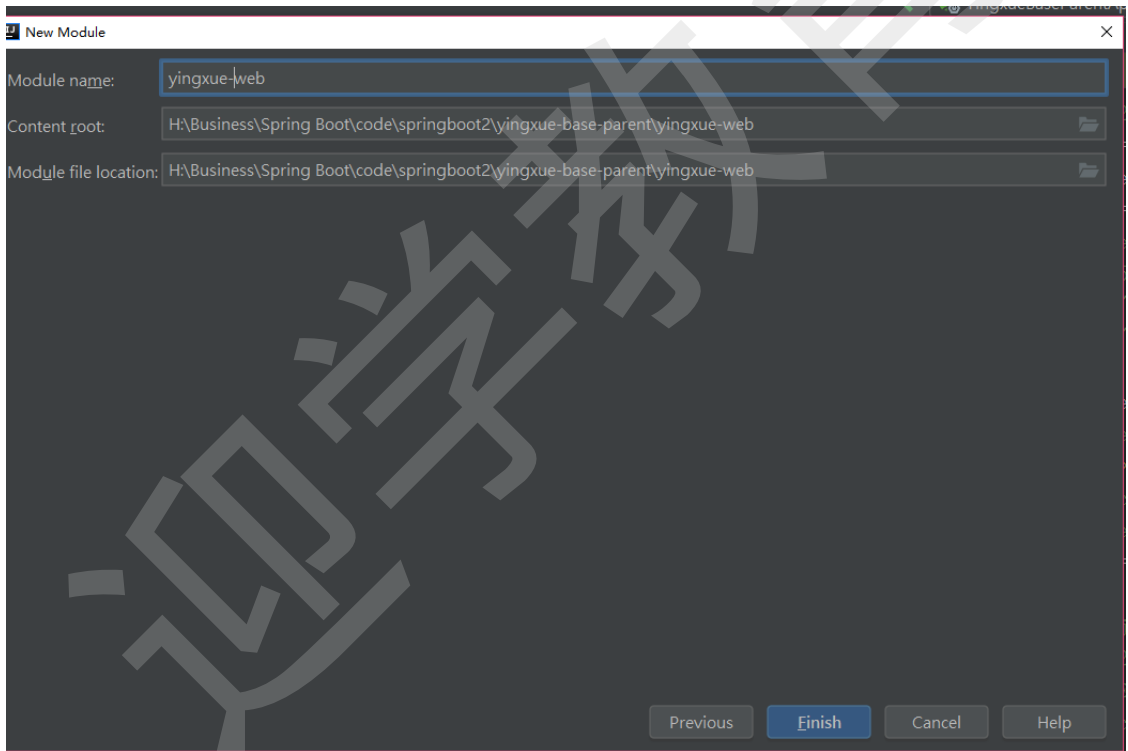
2. 选择Maven，点击Next



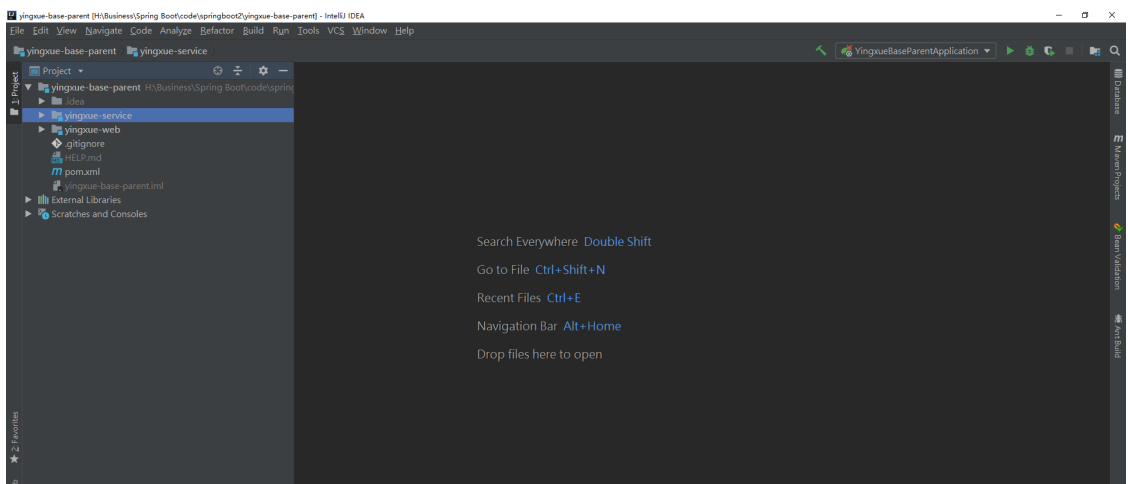
3. 填写ArtifactId，点击Next



4. 修改Module name 加上‘-’增加可读性，点击Finish



5. 同理添加【yingxue-service】子模块，最终得到项目目录结构如下图



### 4.3.3 整理父 pom 文件

1. 修改 spring-boot-starter-parent 版本统一2.1.6.RELEASE
2. 删除 dependencies 标签及其中的依赖，因为 Spring Boot 提供的父工程已包含，并且父 pom 原则上都是通过 dependencyManagement 标签管理依赖包。
3. 删除 build 标签及其中的所有内容，spring-boot-maven-plugin 插件作用是打一个可运行的包，多模块项目仅仅需要在入口类所在的模块添加打包插件，这里父模块不需要打包运行。而且该插件已被包含在 Spring Boot 提供的父工程中，这里删掉即可。
4. 整理后的pom文件如下

```
<?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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <packaging>pom</packaging>
    <modules>
        <module>yingxue-web</module>
        <module>yingxue-service</module>
    </modules>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.6.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.yingxue.lesson</groupId>
    <artifactId>yingxue-base-parent</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>yingxue-base-parent</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>
</project>
```

### 4.3.4 运行项目

1. yingxue-web pom 加入必须的依赖包

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

2. 在yingxue-web层创建com.yingxue.lesson.web包（注意：这是多层目录结构并非单个目录名，com >> yingxue>> lesson>>web）并添加入口类WebApplication.java

```
package com.yingxue.lesson.web;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * @ClassName: WebApplication
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@SpringBootApplication
public class WebApplication {
```

```

public static void main(String[] args) {
    SpringApplication.run(WebApplication.class, args);
}
}

```

3. 在com.yingxue.lesson.web包中添加controller目录并新建一个Testcontroller，添加hello方法测试接口是否可以正常访问

```

package com.yingxue.lesson.web.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @ClassName: Testcontroller
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
@RequestMapping("/test")
public class TestController {
    @GetMapping("/hello")
    public String hello(){
        return "Hello world";
    }
}

```

4. 运行WebApplication类中的main方法启动项目，默认端口为8080，访问<http://localhost:8080/test/hello>得到如下效果



#### 4.3.5配置模块间的依赖关系

通常 JAVA Web 项目会按照功能划分不同模块，模块之间通过依赖关系进行协作，下面将完善模块之间的依赖关系。

1. 首先在父 pom 文件中使用「 dependencyManagement 」标签声明所有子模块依赖

```

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>yingxue-web</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>yingxue-service</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>
</dependencyManagement>

```

2. 其次在 yingxue-service 层中的 pom 文件中添加 spring-boot-starter-web 依赖

```

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

```

- 其次在 yingxue-web 层中的 pom 文件中添加 yingxue-service 去掉 spring-boot-starter-web 依赖

```

<dependency>
    <groupId>${parent.groupId}</groupId>
    <artifactId>yingxue-service</artifactId>
</dependency>

```

#### 4.3.6 web 层调用服务层接口测试

模块依赖关系配置完成之后，通过 web 层 测试下 service 层的接口是否可以正常调用。

- 首先在 yingxue-service 层创建 com.yingxue.lesson 包，添加 service 目录并在其中创建 UserService 接口类及 impl 目录（用于存放接口实现类）。

```

package com.yingxue.lesson.service;

/**
 * @ClassName: UserService
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */

public interface UserService {
    String testService();
}

```

```

package com.yingxue.lesson.service.impl;

import com.yingxue.lesson.service.UserService;
import org.springframework.stereotype.Service;

/**
 * @ClassName: UserServiceImpl
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Service
public class UserServiceImpl implements UserService {
    @Override
    public String testService() {
        return "testService";
    }
}

```

- TestController 通过 @Autowired 注解注入 UserService，修改 TestController 的 test 方法使之调用 UserService 的 testService 方法

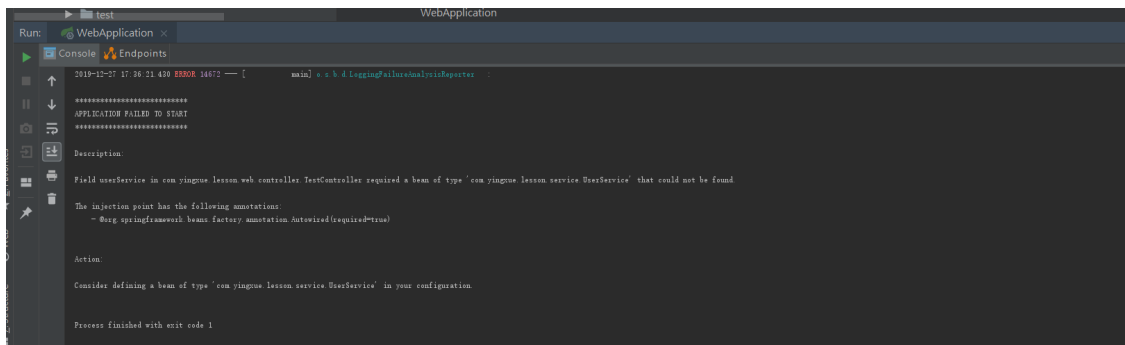
```

@Autowired
private UserService userService;

@GetMapping("/hello")
public String hello(){
    return userService.testService();
}

```

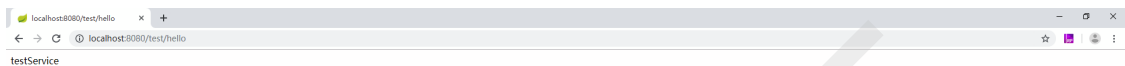
- 再次运行 WebApplication 类中的 main 方法启动项目，发现如下报错



4. 在 WebApplication 入口类中增加包扫描，设置 @SpringBootApplication 注解中的 scanBasePackages 值为 com.yingxue.lesson

```
@SpringBootApplication(scanBasePackages = "com.yingxue.lesson")
```

5. 再次运行 WebApplication 类中的 main 方法启动项目 访问<http://localhost:8080/test/hello>得到如下效果



## 5. Spring Boot 配置文件语法

### 5.1 配置详解

当我们使用spring 初始化器构建完一个Spring Boot项目后，只需引入一个web启动器的依赖，它就变成一个web项目了，而且呢我们什么都没有配置就能通过localhost:8080进行访问了，那这是为什么呢？

那是因为Spring Boot在底层已经把配置信息都给我们自动配置好了。

那我们怎么去修改默认配置信息？

在使用Spring 初始化器创建一个Springboot项目的时候会在resources目录下自动生成一个文件 application.properties，这是一个空文件，它的作用是提供我们修改默认配置信息的入口。

Spring Boot还提供给我们另外一种风格的配置文件 application.yml，虽然是两个不同的文件但是本质是一样的，区别只是其中的语法略微不同。下面我们就来——介绍一下这两种不同风格的配置文件。

#### 5.1.1 Properties 语法

application.properties 配置文件比较简单，形式如下

- key = value

不需要空格进行区分，父属性和子属性之间是以"."进行区分的

#### 5.1.2 YML 语法

1. 大小写敏感。
2. k:(空格)v：表示一对键值对（空格必须有），以空格的缩进来控制层级关系。
3. 只要是左对齐的一列数据，则表示都是同一个层级的。
4. “#”表示注释，从这个字符一直到行尾，都会被解析器忽略。

#### 5.1.3 例子

我们写一个简单的例子看看它们之间的区别：配置端口号

- 以前是这样配置的

```
<server>
  <port>8080</port>
</server>
```

- 现在是这样配置的
  - application.yml

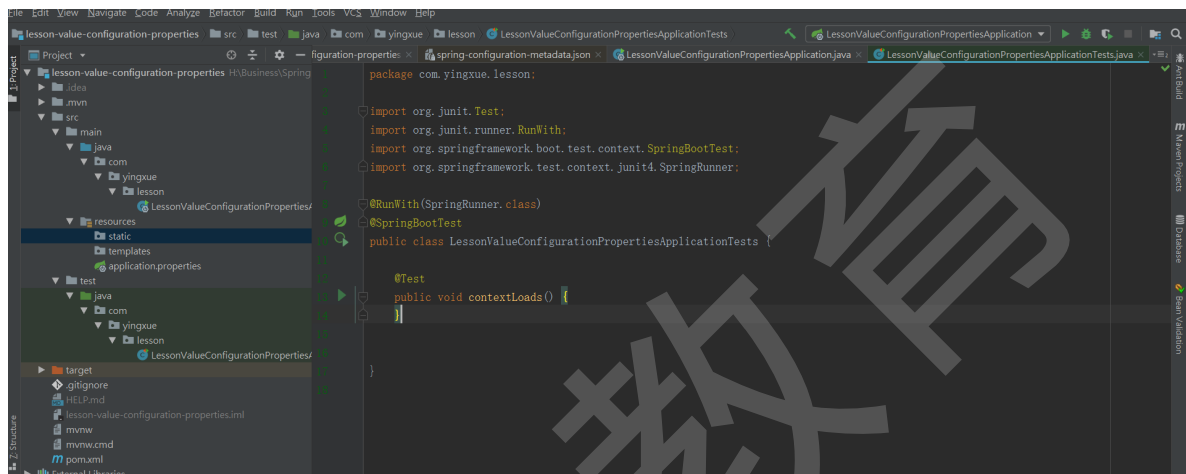
```
server:
  port: 8080
```

- application.properties

```
server.port=8080
```

## 6. 配置文件读取

### 6.1 创建工程



### 6.2 简单的数据类型读取-@Value

- 在配置文件加上配置信息
- 创建一个Person实体类用来验证@Value 读取配置文件属性是否成功
- 创建一个PersonConfig 配置类读取属性
- 把属性set到Person注入到容器
- 单元测试拿到person实例打印它

#### 6.2.1修改默认配置文件

- 修改 application.properties

```
#基本类型
person.userName=zhangsan
person.age=29
person.salary=22000
person.sex=male
```

- 创建 application.yml(和application.properties本质一样两者选其一)

```
person:
  userName: zhangsan
  age: 29
  salary: 220000
  sex: male
```

#### 6.2.2 属性注入类

创建 Person javabean

```
package com.yingxue.lesson.entity;

/**
 * @ClassName: Person
```



```

* TODO:类文件简单描述
* @Author: 小霍
* @UpdateUser: 小霍
* @Version: 0.0.1
*/
public class Person {
    private String userName;
    private int age;
    private double salary;
    private String sex;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    @Override
    public String toString() {
        return "Person{" +
            "userName='" + userName + '\'' +
            ", age=" + age +
            ", salary=" + salary +
            ", sex='" + sex + '\'' +
            '}';
    }
}

```

### 6.2.3 声明配置类

创建 PersonConfig 配置类

```

package com.yingxue.lesson.config;

import com.yingxue.lesson.entity.Person;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @ClassName: PersonConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */

```

```

@Configuration
public class PersonConfig {
    @Value("${person.userName}")
    private String userName;
    @Value("${person.age}")
    private int age;
    @Value("${person.salary}")
    private double salary;
    @Value("${person.sex}")
    private String sex;

    @Bean
    public Person getPerson(){
        Person person=new Person();
        person.setSex(sex);
        person.setAge(age);
        person.setUserName(userName);
        person.setSalary(salary);
        return person;
    }
}

```

## 6.2.4 单元测试

创建 SpringBootDemoApplicationTests Junit 单元测试

```

@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringUtilDemoApplicationTests {

    @Autowired
    private Person person;

    @Test
    void contextLoads() {
    }

    @Test
    public void testPerson(){
        System.out.println(person.toString());
    }

}

```

### 1. 测试输出

```

19:58:14.529 [main] INFO org.springframework.test.annotation.ProfileValueUtils: Identified candidate component class: file [/D:/Business/SpringBoot/Tests/Chapter11/PersonConfig.class]
19:58:14.539 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper: Found @SpringBootConfiguration com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests for test class com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests
19:58:14.566 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper: @TestExecutionListeners is not present for class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests], using defaults
19:58:14.567 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper: Loaded default TestExecutionListener class names from location [META-INF/spring.factories]: [org.springframework.test.context.transaction.TransactionalTestExecutionListener, org.springframework.test.context.jdbc.SqlScriptsTestExecutionListener, org.springframework.test.context.web.WebTestExecutionListener]
19:58:14.579 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper: Skipping candidate TestExecutionListener [org.springframework.test.context.transaction.TransactionalTestExecutionListener] due to a missing @Transactional annotation
19:58:14.582 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper: Using TestExecutionListeners: [org.springframework.test.context.web.WebTestExecutionListener@e2e18f2, org.springframework.test.context.jdbc.SqlScriptsTestExecutionListener@e2e18f2, org.springframework.test.context.transaction.TransactionalTestExecutionListener@e2e18f2]
19:58:14.583 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Retrieved @ProfileValueSourceConfiguration [null] for test class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests]
19:58:14.593 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Retrieved @ProfileValueSourceConfiguration [null] for test class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests]
19:58:14.593 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Retrieved ProfileValueSource type [class org.springframework.test.annotation.SystemProfileValueSource] for class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests]
19:58:14.594 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Retrieved ProfileValueSource type [class org.springframework.test.annotation.SystemProfileValueSource] for class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests]
19:58:14.595 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Retrieved @ProfileValueSourceConfiguration [null] for test class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests]
19:58:14.595 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Retrieved ProfileValueSource type [class org.springframework.test.annotation.SystemProfileValueSource] for class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests]
19:58:14.599 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Retrieved @ProfileValueSourceConfiguration [null] for test class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests]
19:58:15.000 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Retrieved ProfileValueSource type [class org.springframework.test.annotation.SystemProfileValueSource] for class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests]
19:58:15.001 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Retrieved ProfileValueSource type [class org.springframework.test.annotation.SystemProfileValueSource] for class [com.yingus.lesson.lesson.value.configuration.properties.ApplicationTests]
19:58:15.023 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils: Adding inline properties to environment: {spring.jmx.enabled=false, org.springframework.boot.test.context.SpringBootTestContextBootstrapper}

.. Spring Boot .. (v2.1.4 RELEASE)

2019-11-09 10:58:15.475 INFO 14708 [main] e.configuration.properties.ApplicationTests Starting LessonValueConfigurationPropertiesApplicationTests on PC-20170524NPH with PID 14708 (started by Administrator in E:\Buaa)
2019-11-09 10:58:15.477 INFO 14708 [main] e.configuration.properties.ApplicationTests No active profile set, falling back to default profiles: default
2019-11-09 10:58:16.894 ERROR 14708 [main] o.a.catalina.core.AprLifecycleListener An incompatible version [1.2.12] of the APR based Apache Tomcat Native library is installed, while Tomcat requires version [1.2.1]
2019-11-09 10:58:17.461 INFO 14708 [main] o.s.s.concurrent.ThreadPoolExecutor Initializing ExecutorService 'applicationTaskExecutor'
2019-11-09 10:58:17.798 INFO 14708 [main] e.configuration.properties.ApplicationTests Started LessonValueConfigurationPropertiesApplicationTests in 2.774 seconds (JVM running for 3.810)
Person{userName='shangsan', age=29, salary=22000.0, sex='male'}
2019-11-09 10:58:18.132 INFO 14708 [main] Thread-2 o.s.s.concurrent.ThreadPoolExecutor Shutting down ExecutorService 'applicationTaskExecutor'

Process finished with exit code 0

```

### 2. 解读

- @Configuration : 声明我们PersonConfig 是一个配置类
- 通过@Value 为属性注入值
- 通过@Bean Spring会自动调用该方法，将方法的返回值加入Spring容器中。

## 6.3 复杂的配置文件读取- @ConfigurationProperties

- 配置文件配置复杂的内容
- 创建映射实体
- 屏蔽PersonConfig 配置类的代码
- 修改 Person 实体类
- 打印person实体

### 6.3.1 修改配置文件置

- application.Properties

```
#基本类型
person.userName=zhangsan
person.age=29
person.salary=22000
person.sex=male
#array
person.pets=dog,cat
#map
person.friend.userName=wangwu
person.friend.age=28
#list
person.list[0]=value1
person.list[1]=value2
#list嵌套Map/对象
person.children[0].name=ting
person.children[0].weight=2
person.children[1].name=hao
person.children[1].weight=5
#对象嵌套对象
person.employee.name=lisi
person.employee.age=21
```

- application.yml

```
person:
  userName: zhangsan
  age: 29
  salary: 220000
  sex: male
  pets: cat,dog
  friend:
    userName: wangwu
    age: 27
  list:
    - value1
    - value2
  children:
    - name: ting
      weight: 2
    - name: hao
      weight: 5
  employee:
    name: lisi
    age: 21
```

### 6.3.2 注释 PersonConifg

### 6.3.3 创建 Employee

```
package com.yingxue.lesson.entity;

/**
 * @ClassName: Employee
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍

```

```

* @Version: 0.0.1
*/
public class Employee {
    private String name;
    private String age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAge() {
        return age;
    }

    public void setAge(String age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Employee{" +
            "name='" + name + '\'' +
            ", age='" + age + '\'' +
            '}';
    }
}

```

### 6.3.3 修改 peson

```

package com.yingxue.lesson.entity;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import java.util.Arrays;
import java.util.List;
import java.util.Map;

/**
 * @ClassName: Person
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@ConfigurationProperties(prefix = "person")
@Component
public class Person {
    private String userName;
    private int age;
    private double salary;
    private String sex;
    //array
    private String[] pets;
    //map
    private Map<String,String> friend;
    //list
    private List<String> list;
    //list嵌套对象或者map
    private List<Map<String,String>> children;
    //对象嵌套对象
    private Employee employee;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {

```

```
        this.userName = userName;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public String[] getPets() {
        return pets;
    }

    public void setPets(String[] pets) {
        this.pets = pets;
    }

    public Map<String, String> getFriend() {
        return friend;
    }

    public void setFriend(Map<String, String> friend) {
        this.friend = friend;
    }

    public List<String> getList() {
        return list;
    }

    public void setList(List<String> list) {
        this.list = list;
    }

    public List<Map<String, String>> getChildren() {
        return children;
    }

    public void setChildren(List<Map<String, String>> children) {
        this.children = children;
    }

    public Employee getEmployee() {
        return employee;
    }

    public void setEmployee(Employee employee) {
        this.employee = employee;
    }

    @Override
    public String toString() {
        return "Person{" +
            "userName='" + userName + '\'' +
            ", age=" + age +
```

```

        ", salary=" + salary +
        ", sex='" + sex + '\'' +
        ", pets=" + Arrays.toString(pets) +
        ", friend=" + friend +
        ", list=" + list +
        ", children=" + children +
        ", employee=" + employee +
        '}}';
    }
}

```

### 6.3.3 单元测试

运行 textPerson 方法

#### 1. 运行结果

```

Tests passed: 1 of 1 test - 188 ms

11:55:02.422 [main] INFO org.springframework.test.annotation.ProfileValueUtils - Retrieved @ProfileValueSourceConfiguration [null] for test class [com.yingxue.lesson.LessonValueConfigurationPropertiesApplicationTests]
11:55:02.422 [main] INFO org.springframework.test.annotation.ProfileValueUtils - Retrieved ProfileValueSource type [class org.springframework.test.annotation.SystemProfileValueSource] for class [com.yingxue.lesson.LessonValueConfigurationPropertiesApplicationTests]
11:55:02.422 [main] INFO org.springframework.test.annotation.ProfileValueUtils - Before test class: context: [DefaultTestContext@4d719e testClass=LessonValueConfigurationPropertiesApplicationTests, testInstance=[null], testMethod=
11:55:02.422 [main] INFO org.springframework.test.annotation.ProfileValueUtils - Retrieved @ProfileValueSourceConfiguration [null] for test class [com.yingxue.lesson.LessonValueConfigurationPropertiesApplicationTests]
11:55:02.422 [main] INFO org.springframework.test.annotation.ProfileValueUtils - Retrieved ProfileValueSource type [class org.springframework.test.annotation.SystemProfileValueSource] for class [com.yingxue.lesson.LessonValueConfigurationPropertiesApplicationTests]
11:55:02.459 [main] INFO org.springframework.test.annotation.ProfileValueUtils - Adding inlined properties to environment: (spring.jmx.enabled=false, org.springframework.boot.test.context.SpringBootTestContextBootstrapper=true, server.port=0)

Spring Boot :: (v2.1.4.RELEASE)

2019-11-09 11:55:03.019 INFO 11652 [main] s@ConfigurationPropertiesApplicationTests Starting LessonValueConfigurationPropertiesApplicationTests on PC-DE176541CFE with PID 11652 (started by Administrator in D:\Business\SpringBoot\redis\redis\JITFree
2019-11-09 11:55:03.021 INFO 11652 [main] s@ConfigurationPropertiesApplicationTests No active profile set, falling back to default profiles: default
2019-11-09 11:55:04.114 INFO 11652 [main] s-w-cavalink-test-ApplicationTests An incompatible version (3.2.12) of the JPA based Apache Tomcat Native library is installed, while Tomcat requires version (3.2.14)
2019-11-09 11:55:04.160 INFO 11652 [main] s-w-cavalink-test-ApplicationTests Initializing ExecutorService 'applicationDefaultExecutor'
2019-11-09 11:55:05.181 INFO 11652 [main] s@ConfigurationPropertiesApplicationTests Started LessonValueConfigurationPropertiesApplicationTests in 2.729 seconds (JVM running for 3.864)
2019-11-09 11:55:05.284 INFO 11652 [main] s-w-cavalink-test-ApplicationTests Person{name='zhangsan', age=29, sex='male', pet='dog', cat='dog', friend='liu', name='zhangsan', age=29, name='zhangsan', list='zhangsan, liu', value='zhangsan', children='zhangsan, weight=0', liu='liu', weight=0', employee='Employee{name='liu', age='31'}}
2019-11-09 11:55:05.284 INFO 11652 [main] s-w-cavalink-test-ApplicationTests Thread-2 s s concurrent ThreadPooledExecutor Shutting down ExecutorService 'applicationDefaultExecutor'
Process finished with exit code 0

```

#### 2. 解读

- @ConfigurationProperties 注解向Spring Boot声明该类中的所有属性和配置文件中相关的配置进行绑定。
  - prefix = "person" : 声明配置前缀，将该前缀下的所有属性进行映射。
- @Component 将该组件加入Spring Boot容器，只有这个组件是容器中的组件，配置才生效。

## 7. 静态工具类读取配置文件

在日常开发中经常会遇到这种情况，就是有一些方法经常用到但是呢又需要外部的一些静态常量(各个环境的值是不一样的)进行组装，所以呢我们会把这些方法抽出来统一放在一个工具类里面，大家也知道通常来说工具类里面都是一些静态的方法，后续在其它地方使用的时候直接工具类.方法就可以了。但是呢我们的方法改成静态方法后，里面用到的变量也同样要改成静态的变量，这里就涉及到了静态属性的注入问题了，如果我们还像上课那样用 @Value 直接注入是注入不进来的。那么我们该怎么去改造呢？怎么才能把它注入到静态的变量呢？好下面我们就来实践一下。

- 加入配置文件信息
- 新增一个配置读取类：TokenSettings
- 新增静态工具类：JwtTokenUtil
- 创建一个代理工具类：StaticInitializerUtil
- 单元测试

#### 1. 配置文件信息

##### 1. application.yml

```

#JWT 密钥
jwt:
  secretKey: xxxxxfdsfxxx
  issuer: yingxue.com

```

##### 2. application.properties

```

#JWT 密钥
jwt.secretKey=xxxxxfdsfxxx
jwt.issuer=yingxue.com

```

#### 2. 新增 一个配置读取类 TokenSettings

```
@Configuration
@ConfigurationProperties(prefix = "jwt")
@Data
public class TokenSettings {
    private String secretKey;
    private String issuer;
}
```

### 3. 创建一个JwtTokenUtil

```
public class JwtTokenUtil {

    private static String secretKey;
    private static String issuer;

    public static void setTokenSettings(TokenSettings tokenSettings){
        secretKey=tokenSettings.getSecretKey();
        issuer=tokenSettings.getIssuer();
    }
    public static String getSecretKey(){

        return secretKey;
    }
    public static String getIssuer(){

        return issuer;
    }

}
```

### 4. 创建一个代理类 StaticInitializerUtil

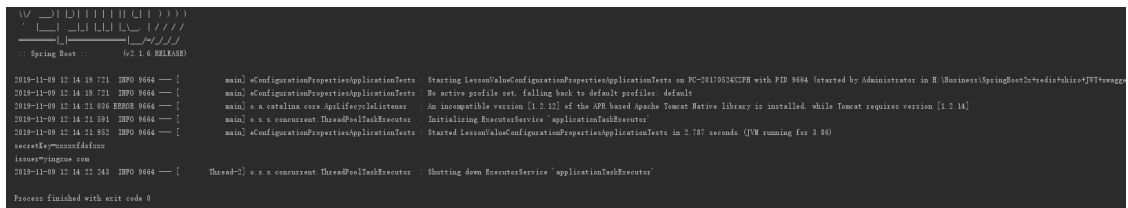
```
@Component
public class StaticInitializerUtil {
    private TokenSettings tokenSettings;

    public StaticInitializerUtil(TokenSettings tokenSettings) {
        JwtTokenUtil.setTokenSettings(tokenSettings);
    }
}
```

### 5. 单元测试

```
@Test
public void testStaticUtil(){
    System.out.println("secretKey="+JwtTokenUtil.getSecretKey());
    System.out.println("issuer="+JwtTokenUtil.getIssuer());
}
```

### 6. 输出



```

2018-11-09 12:14:19.721 INFO 9664 --- [main] oConfigurationPropertiesApplicationTests Starting Lesson\oConfigurationPropertiesApplicationTests on PC-201705247278 with PID 9664 (started by Administrator in E:\Business\SpringBoot\src\distribution\JWT\src\main\resources\application.yml)
2018-11-09 12:14:19.721 INFO 9664 --- [main] oConfigurationPropertiesApplicationTests No active profile set, falling back to default profiles: default
2018-11-09 12:14:21.039 ERROR 9664 --- [main] o.a.catalina.core.AprLifecycleListener: An incompatible version (1.2.12) of the APR based Apache Tomcat Native library is installed, while Tomcat requires version [1.2.14]
2018-11-09 12:14:21.591 INFO 9664 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor Initializing ExecutorService 'applicationTaskExecutor'
2018-11-09 12:14:21.952 INFO 9664 --- [main] oConfigurationPropertiesApplicationTests Started Lesson\oConfigurationPropertiesApplicationTests in 2.787 seconds (JVM running for 3.86)
2018-11-09 12:14:22.249 INFO 9664 --- [Thread-2] o.s.s.concurrent.ThreadPoolTaskExecutor Shutting down ExecutorService 'applicationTaskExecutor'
Process finished with exit code 0

```

## 8. Spring Boot Profile 多环境配置

我们在开发项目时，通常同一套程序会被发布到几个不同的环境，比如：开发、测试、生产等。其中每个环境的数据库地址、redis地址、服务器端口等等配置都会不同，如果在为不同环境打包时都要频繁修改配置文件的话，那必将是个非常繁琐且容易发生错误的事。

对于多环境的配置，各种项目构建工具或是框架的基本思路是一致的，通过配置多份不同环境的配置文件，再通过打包命令指定需要打包的内容之后进行区分打包，而Spring Boot 就更简单了，打好包后，只需在不同的环境下启动时候指定读取的配置文件就可以了。

在Spring Boot中多环境配置文件名需要满足application-{profile}.properties的格式，其中{profile}对应你的环境标识，比如：

application-dev.properties：开发环境

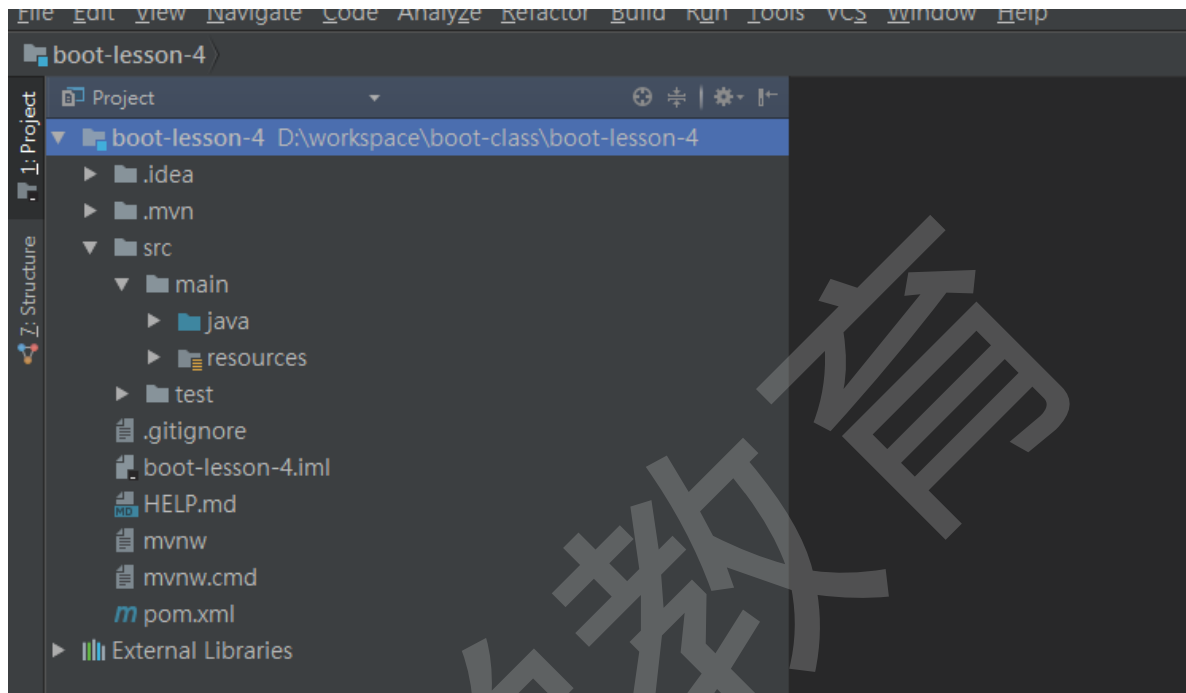
application-test.properties：测试环境

application-prod.properties：生产环境

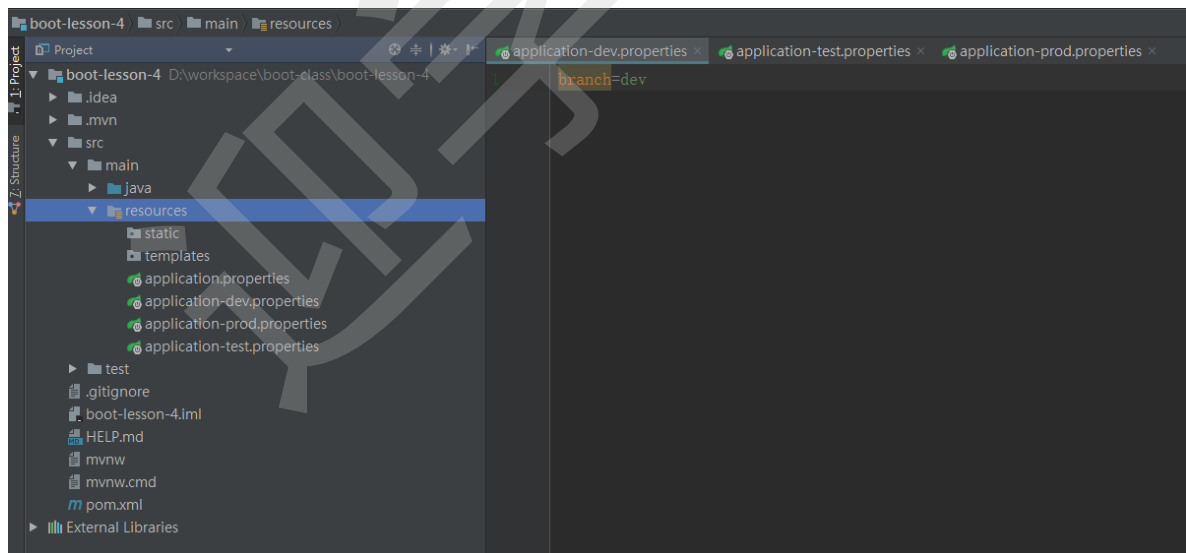
至于哪个具体的配置文件会被加载，需要在application.properties文件中通过spring.profiles.active属性来设置，其值对应{profile}值。

如：spring.profiles.active=test就会加载application-test.properties配置文件内容

## 8.1 创建工程



## 8.2 创建多个 配置文件



## 8.3 每个环境的properties 设置branch分支的值

- application.properties 默认dev

```
#默认使用dev的配置
spring.profiles.active=dev
```

- application-dev.properties 端口8080

```
server.port=8080
branch=dev
```



- application-test.properties 端口8081

```
server.port=8081  
branch=test
```

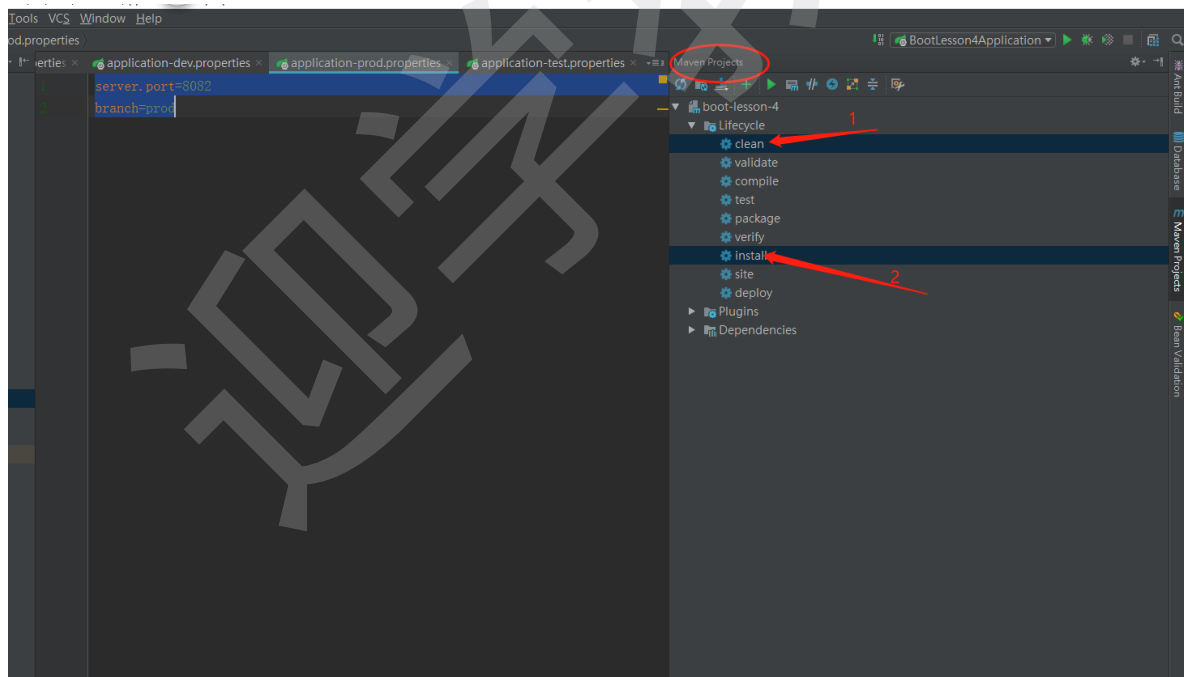
- application-prod.properties 端口8082

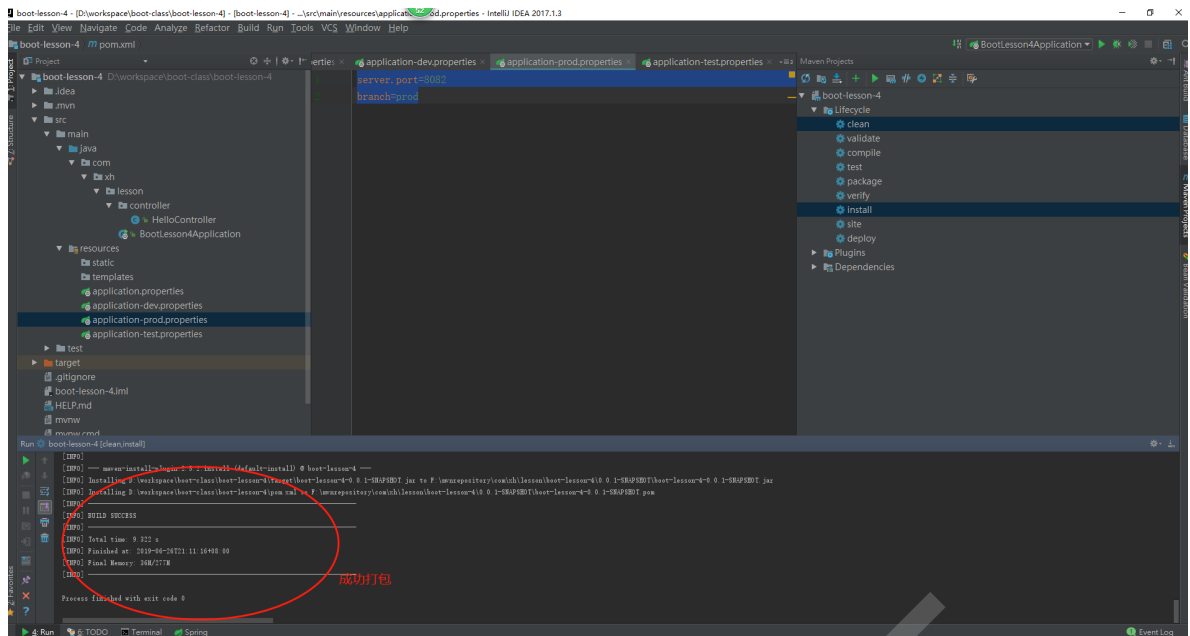
```
server.port=8082  
branch=prod
```

## 8.4 创建一个测试类 controller

```
package com.yingxue.lesson.controller;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("/test")  
public class HelloController {  
    @Value("${branch}")  
    private String branch;  
    @GetMapping("/branch")  
    public String test(){  
        return "Spring Boot Current branch "+branch;  
    }  
}
```

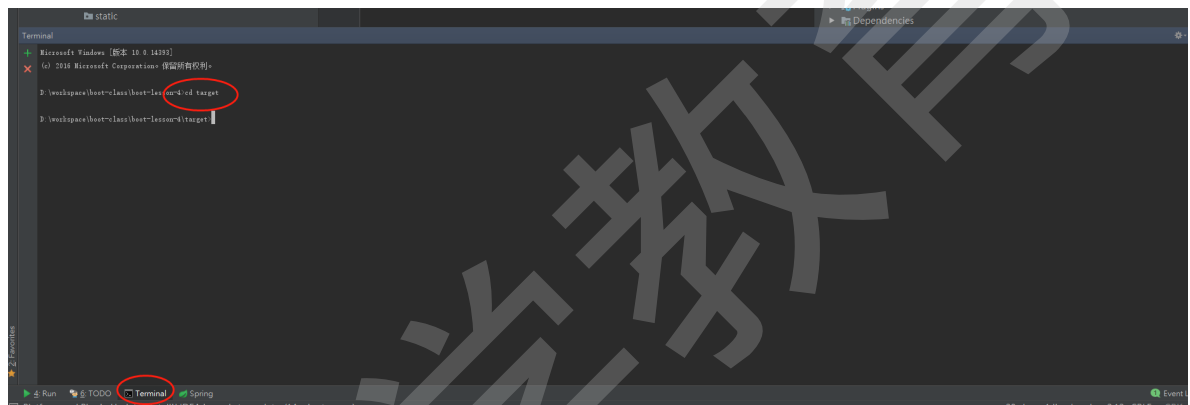
## 8.5 maven project clean install





## 8.6 测试

- cd target



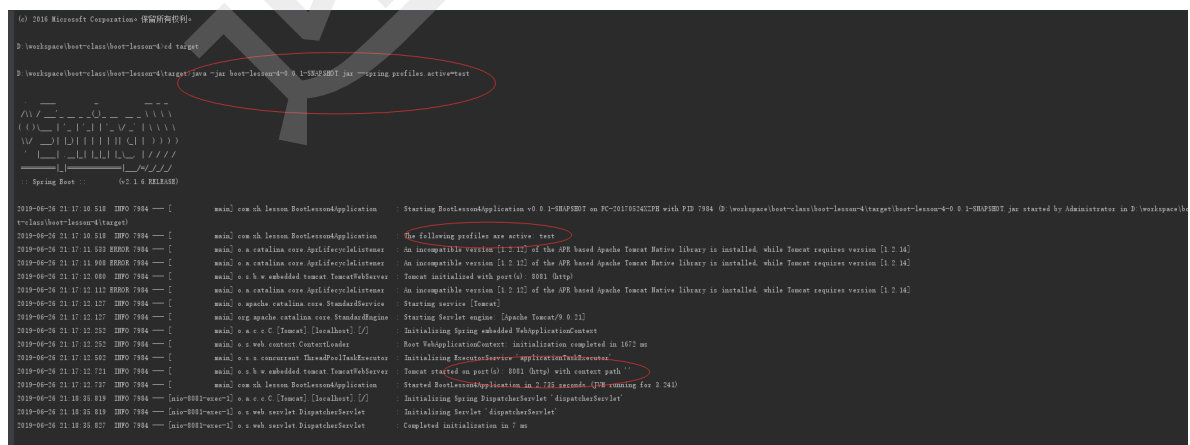
- 进入到 jar 同级执行 java -jar xxx.jar --spring.profiles.active=test

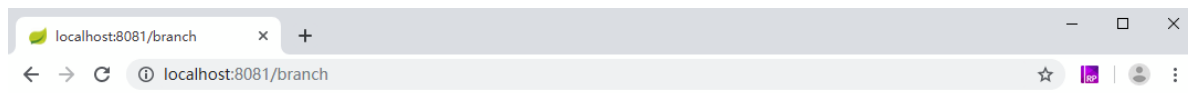
xxx:指的是你项目打包后jar包名称

--spring.profiles.active=test 启动制定执行哪个环境的properties

cd target

java -jar boot-lesson-4-0.0.1-SNAPSHOT.jar --spring.profiles.active=test

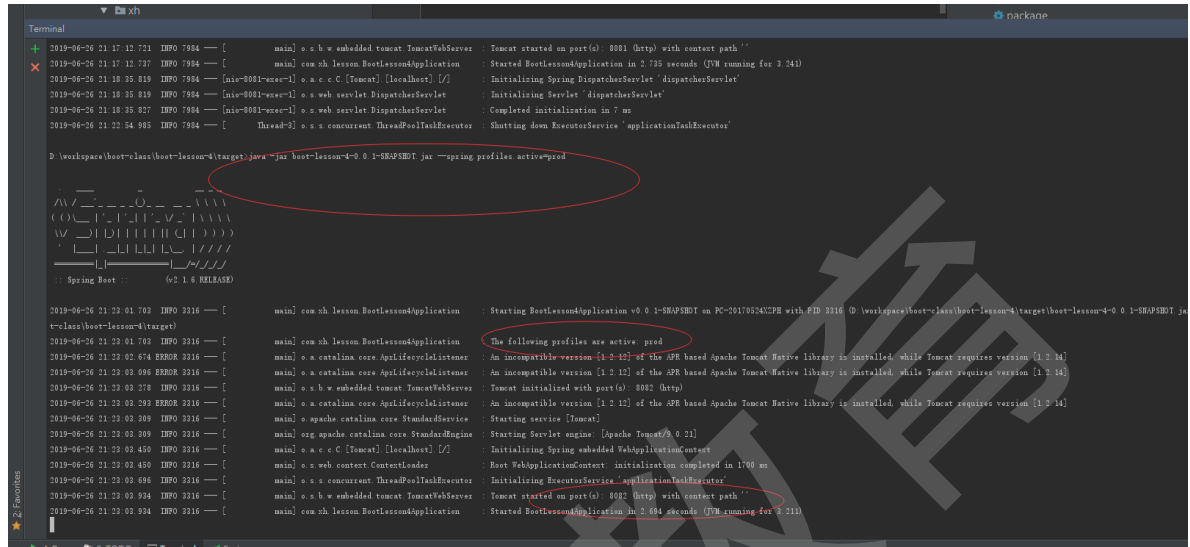




Spring Boot Current branch test

cd target

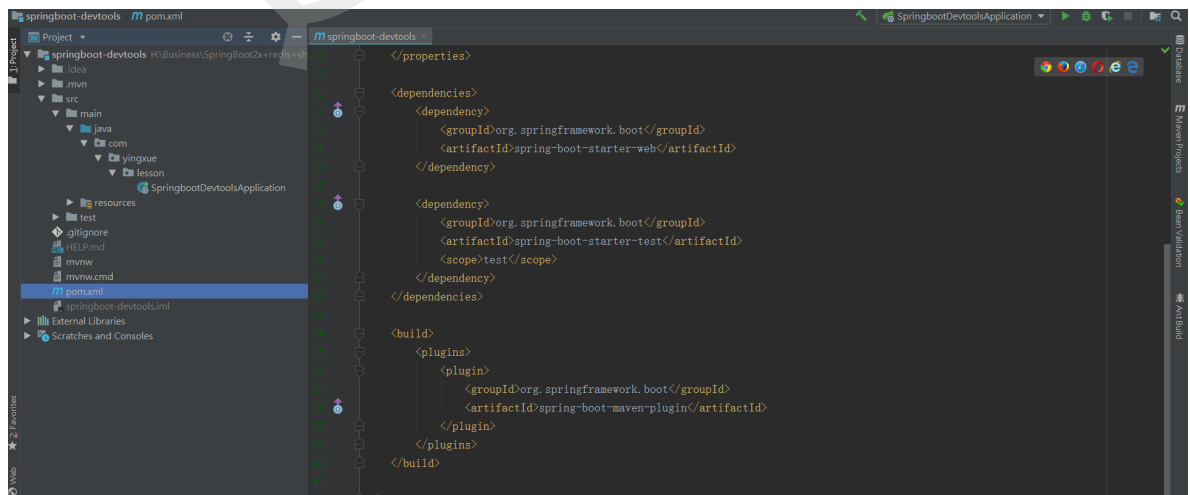
java -jar boot-lesson-4-0.0.1-SNAPSHOT.jar --spring.profiles.active=prod



Spring Boot Current branch prod

## 9. Spring Boot devtools热部署

### 9.1 创建工程



### 9.1 引入 依赖

要想成功配置热部署功能必须在 pom 文件引入 spring-boot-devtools 依赖 它是 Spring Boot 提供的一组开发工具包，其中就包含我们需要的热部署功能。

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

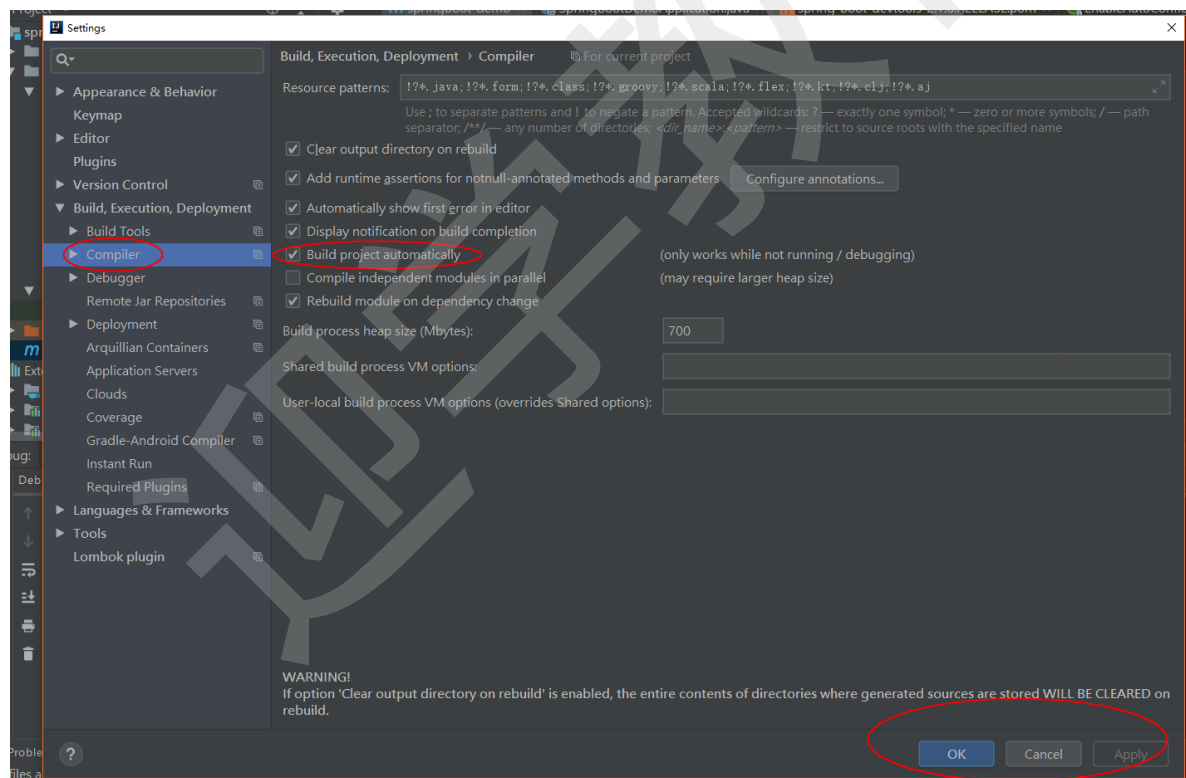
## 9.2 配置 plugin

加入 plugin 且配置一个属性 fork 为 true

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <fork>true</fork>
      </configuration>
    </plugin>
  </plugins>
</build>
```

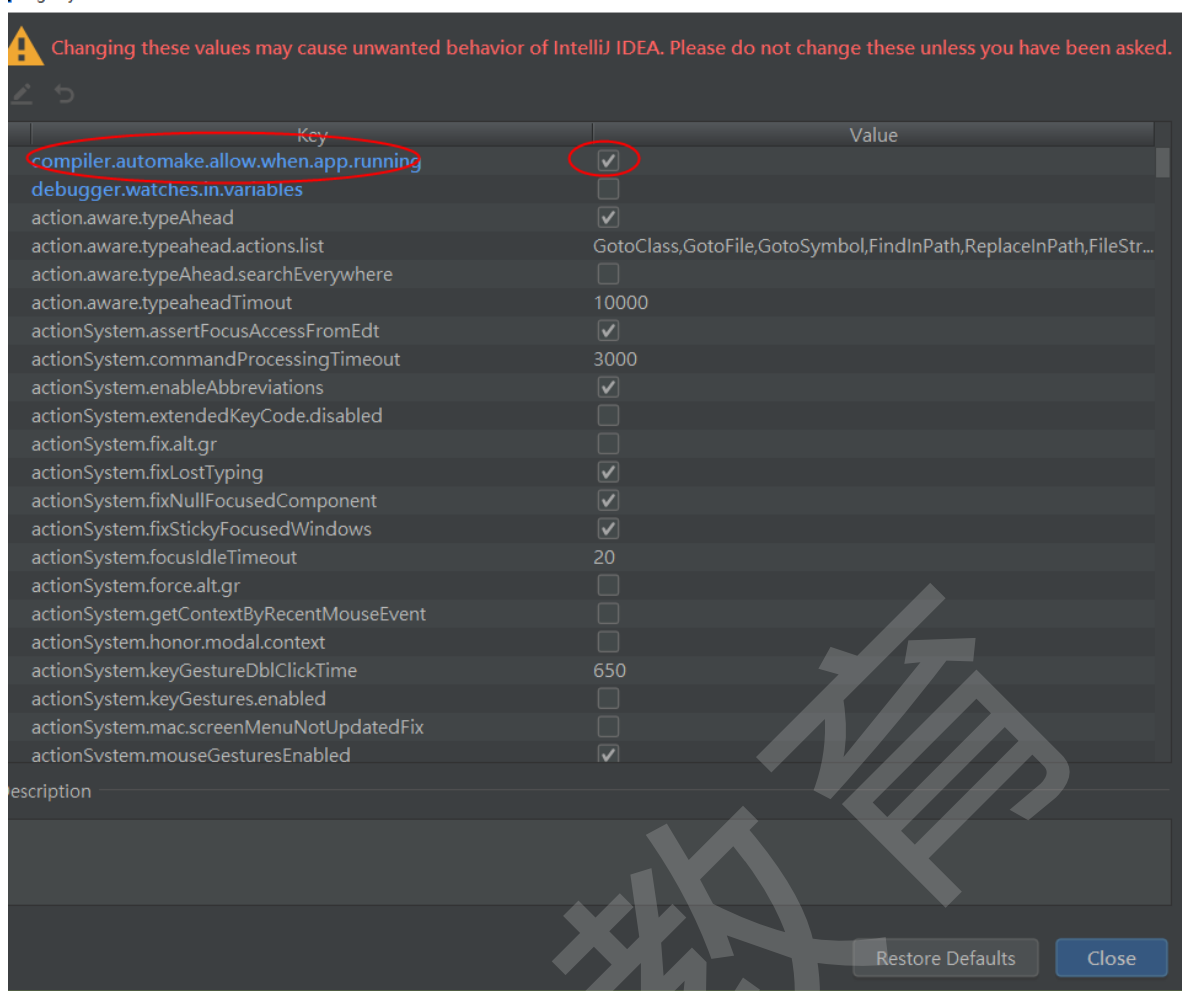
## 9.3 开启自动编译

选择 File | Settings | Compiler 命令，然后勾选 Build project automatically 复选框，低版本的 IDEA 请勾选 make project automatically 复选框。

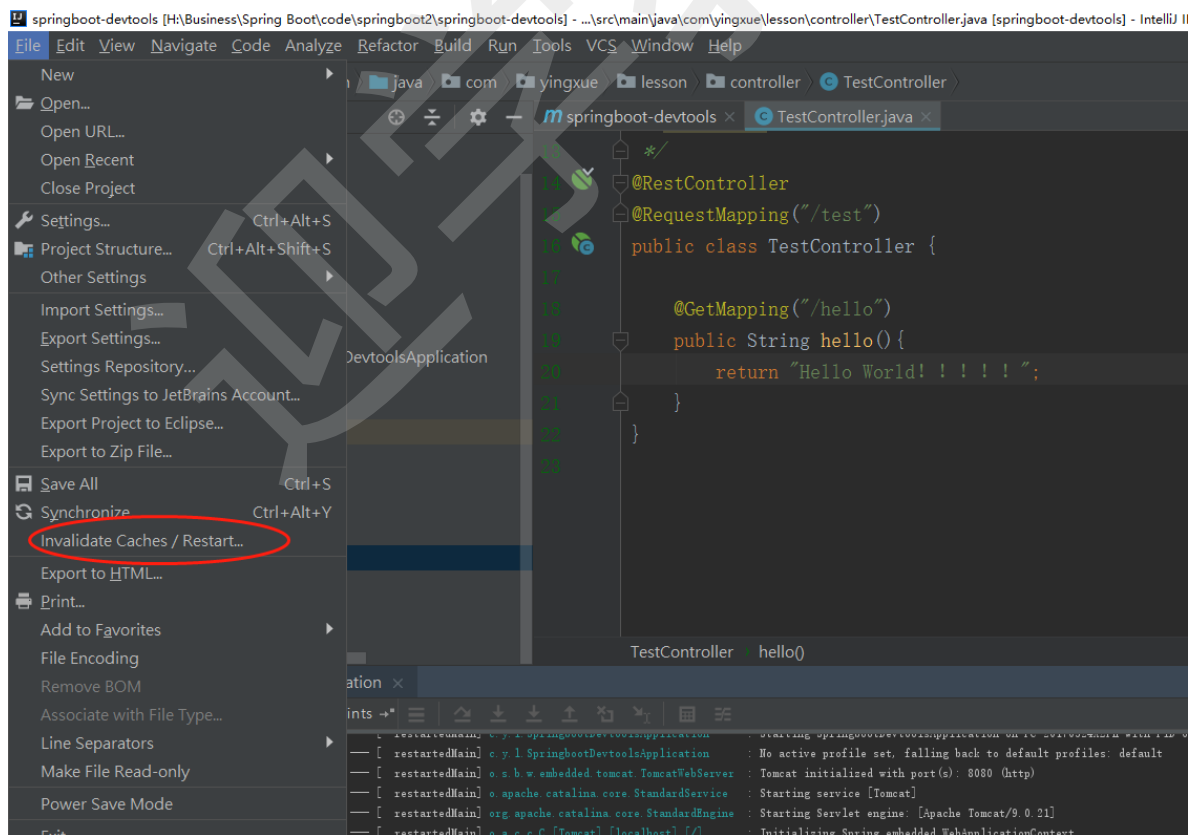


## 9.4 IDEA 设置运行中，允许自动编译

操作：ctrl + shift + alt + /，选择Registry，勾选上 Compiler autoMake allow when app running



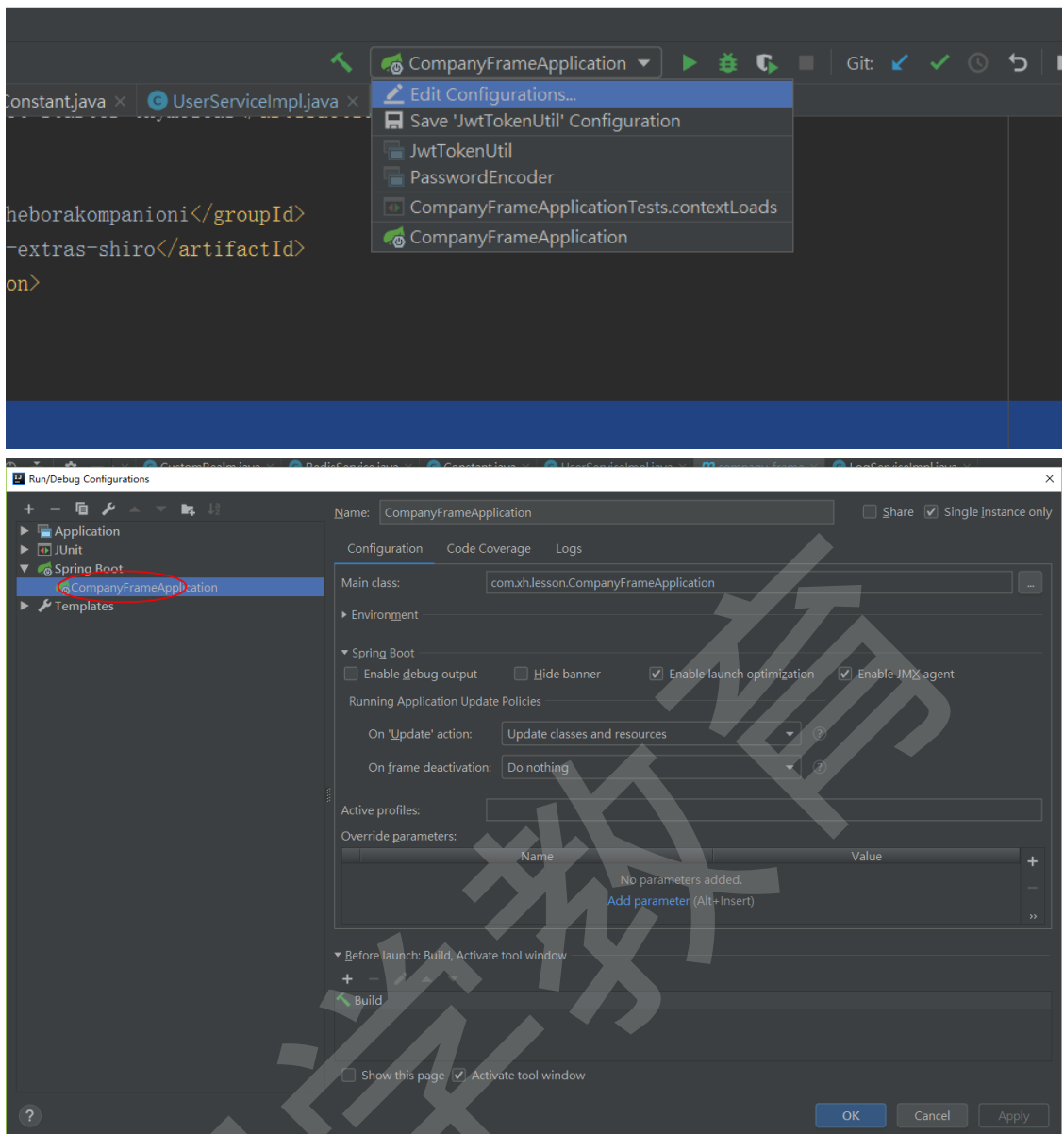
## 重启 IDEA



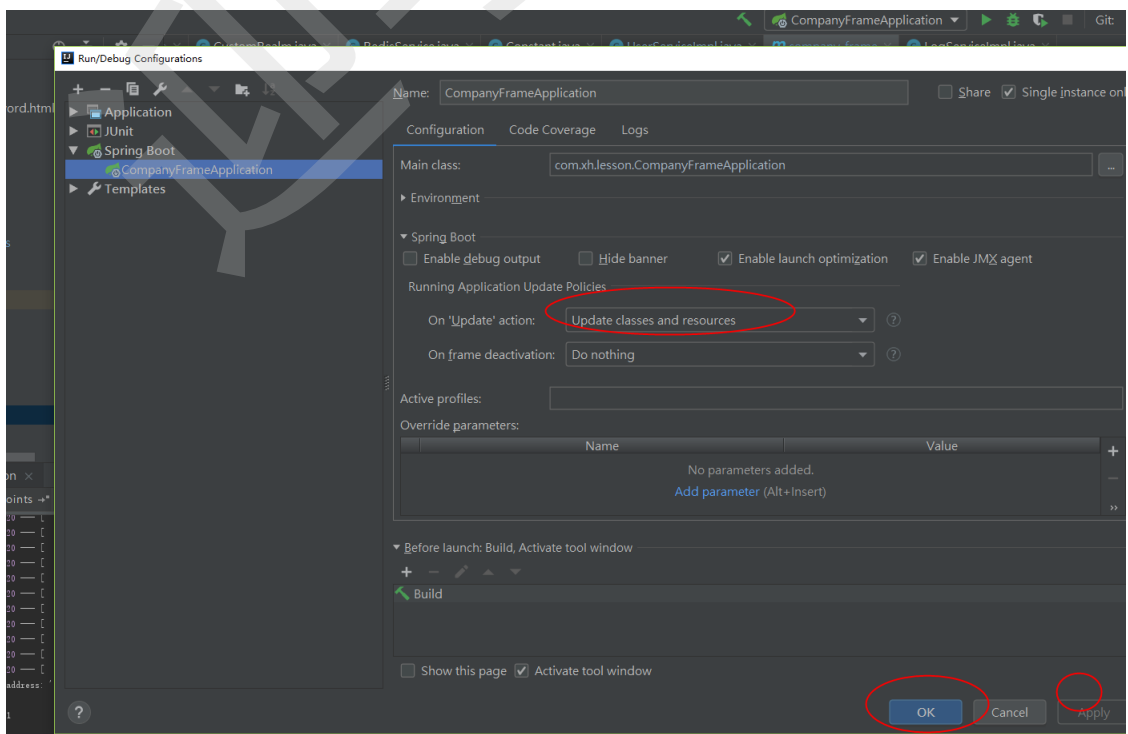
## 9.5 手动触发编译

### 9.5.1 设置策略

- 打开 edit configurations

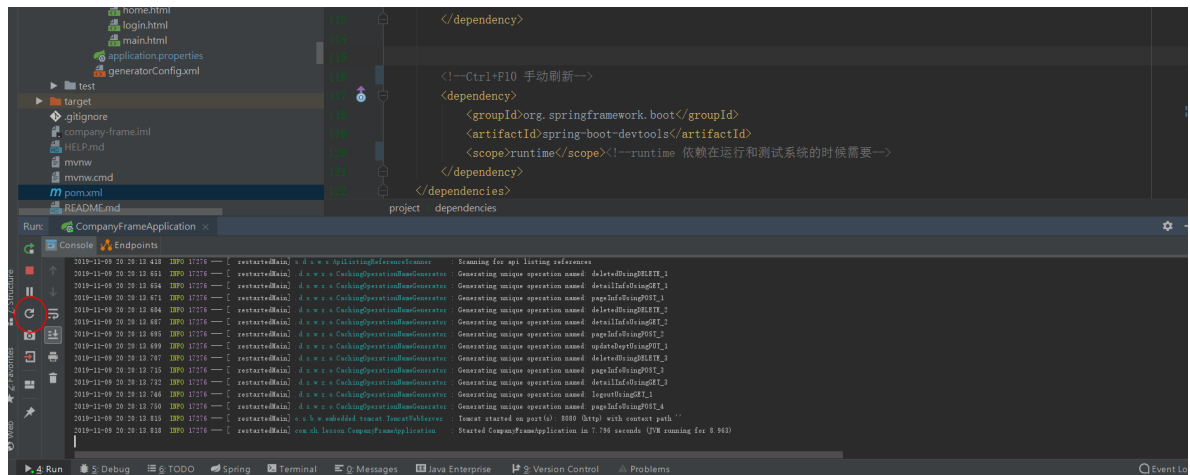


- 找到 Running Application Update Policies---->On Update action 选择 Update classes and resources



## 9.5.2 触发自动编译

按一下左下角的 Run 栏目中的 Update xxx application 才会触发热切换。也可以用快捷键 `ctrl+F10`



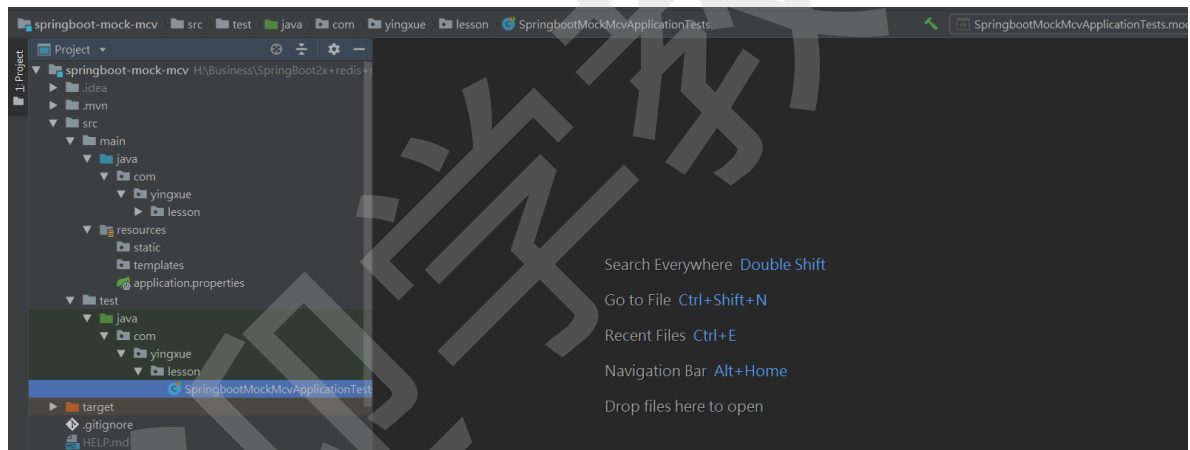
## 10. Spring Boot 单元测试 MockMvc

### 10.1 简介

为何使用MockMvc？对模块进行集成测试时，希望能够通过输入 URL 对接口进行测试，如果通过启动服务器，建立http client进行测试，这样会使得测试变得麻烦，假如我们的项目启动速度慢、网络环境差，这样就会造成我们的测试效率底下而且由于网络的原因出现一些不可描述bug。所以为了方便我们对接口的测试，我们引入了 MockMVC。MockMvc 实现了对 Http 请求的模拟，能够直接转换到 接口的调用，而且不依赖网络环境，还提供了一套验证的工具，这样我们就能直观的看到认证的结果，大大提高了我们的自测效率。

### 10.2 例子

#### 10.2.1 创建工程



#### 10.2.2 创建 UserInfo javaBean

```
package com.yingxue.lesson.model;  
  
/**  
 * @ClassName: UserInfo  
 * TODO:类文件简单描述  
 * @Author: 小霍  
 * @UpdateUser: 小霍  
 * @Version: 0.0.1  
 */  
public class UserInfo {  
    private String userId;  
    private String name;  
  
    public String getUserId() {  
        return userId;  
    }  
  
    public void setUserId(String userId) {  
        this.userId = userId;  
    }  
}
```

```

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```

### 10.2.3 创建接口

```

package com.yingxue.lesson.controller;

import com.yingxue.lesson.model.UserInfo;
import org.springframework.web.bind.annotation.*;

/**
 * @ClassName: MockmvcController
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
public class MockmvcController {

    @GetMapping("/user")
    public UserInfo getUser(@RequestParam(required = false)String userId,@RequestParam(required = false)
String name){
        UserInfo userInfo=new UserInfo();
        userInfo.setName(name);
        userInfo.setUserId(userId);
        return userInfo;
    }

    @PostMapping("/user")
    public UserInfo getUser(@RequestBody UserInfo userInfo){
        System.out.println(userInfo.getName());
        return userInfo;
    }
}

```

### 10.2.4 创建单元测试

```

package com.yingxue.lesson;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.ObjectWriter;
import com.fasterxml.jackson.databind.util.JSONPObject;
import com.yingxue.lesson.model.UserInfo;
import net.minidev.json.JSONObject;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.skyscreamer.jsonassert.JSONCompareResult;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@RunWith(SpringRunner.class)

```



```

@SpringBootTest
public class SpringbootMockMvcApplicationTests {
    private MockMvc mockMvc;
    @Autowired
    private WebApplicationContext context;

    @Before
    public void setUp(){
        mockMvc=MockMvcBuilders.webAppContextSetup(context).build();
    }
    @Test
    public void contextLoads() {
    }

    @Test
    public void mockMvcGet()throws Exception{
        MvcResult mvcResult = mockMvc.perform(
            MockMvcRequestBuilders.get("/user")
                .contentType(MediaType.APPLICATION_JSON_UTF8)
                .accept(MediaType.APPLICATION_JSON_UTF8) //accept指定客户端能够接收的内容类型
                param("userId", "1").
                param("name", "zhangsan"))
            .andDo(print())
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.userId").value("1"))//验证userId是否为1, jsonPath的使用
            .andExpect(jsonPath("$.name").value("zhangsan"))//验证name是否为zhangsan, jsonPath的使用
            .andReturn();
        System.out.println(mvcResult.getResponse().getContentAsString());
    }

    @Test
    public void mockMvcPost()throws Exception{
        UserInfo userInfo=new UserInfo();
        userInfo.setName("lisi");
        userInfo.setUserId("2");
        ObjectMapper mapper = new ObjectMapper();
        ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();
        String json=ow.writeValueAsString(userInfo);
        MvcResult mvcResult = mockMvc.perform(
            MockMvcRequestBuilders.post("/user")
                .contentType(MediaType.APPLICATION_JSON_UTF8)
                .content(json)
                .accept(MediaType.APPLICATION_JSON_UTF8))
            .andDo(print())
            .andExpect(jsonPath("$.userId").value("1"))//验证userId是否为1, jsonPath的使用
            .andExpect(jsonPath("$.name").value("zhangsan"))//验证name是否为zhangsan, jsonPath的使用
            .andReturn();
        System.out.println(mvcResult.getResponse().getContentAsString());
    }
}

```

### 10.3 MockMvc 具体流程

我们怎么使用MockMvc呢？

1. MockMvcBuilder构造MockMvc的构造器
2. mockMvc调用perform，执行一个RequestBuilder请求，调用controller的业务处理逻辑
3. perform返回ResultActions，返回操作结果，通过ResultActions，提供了统一的验证方式
4. 使用StatusResultMatchers对请求结果进行验证
5. 使用ContentResultMatchers对请求返回的内容进行验证

### 10.4 MockMvcBuilder

MockMvc是spring测试下的一个非常好用的类，他们的初始化需要在setUp中进行。

MockMvcBuilder是用来构造MockMvc的构造器

MockMvcBuilders.webAppContextSetup(WebApplicationContext context)：指定WebApplicationContext，将会从该上下文获取相应的控制器并得到相应的MockMvc

### 10.5 MockMvcRequestBuilders

从名字可以看出，RequestBuilder用来构建请求的，主要的api有：

MockHttpServletRequestBuilder get(String urlTemplate, Object... urlVariables)：根据 uri 模板和 uri 变量值得到一个GET请求方式的RequestBuilder，如果在controller的方法中method选择的是RequestMethod.GET，那在 mockMvc 单元测试中对应就要使用MockMvcRequestBuilders.get。后续的post、put、deleted、等请求方式以此类推。

## 10.6 ResultActions

调用MockMvc.perform(RequestBuilder requestBuilder)后将得到ResultActions，对ResultActions有以下三种处理：

1. ResultActions.**andExpect**：添加执行完成后的断言。添加ResultMatcher验证规则，验证控制器执行完成后结果是否正确；
2. ResultActions.**andDo**：添加一个结果处理器，比如此处使用.andDo(MockMvcResultHandlers.print())输出整个响应结果信息，可以在调试的时候使用。
3. ResultActions.**andReturn**：表示执行完成后返回相应的结果