

Spring Boot

一、Spring Boot 介绍

1 什么是 Spring Boot

Spring Boot 是一个框架，一种全新的编程规范，他的产生简化了框架的使用，所谓简化是指简化了 Spring 众多框架中所需的大量且繁琐的配置文件，所以 Spring Boot 是一个服务于框架的框架，服务范围是简化配置文件。所以从本质上来说，Spring Boot 其实就是 Spring 框架的另一种表现形式。

2 Spring Boot 特征

- 使用 Spring Boot 可以创建独立的 Spring 应用程序
- 在 Spring Boot 中直接嵌入了 Tomcat、Jetty、Undertow 等 Web 容器，所以在使用 SpringBoot 做 Web 开发时不需要部署 WAR 文件
- 通过提供自己的启动器(Starter)依赖，简化项目构建配置
- 尽量的自动配置 Spring 和第三方库
- 提供了生产就绪特征，如：度量指标，运行状况检查和外部化配置
- 绝对没有代码生成，也不需要 XML 配置文件

3 Spring Boot 版本介绍

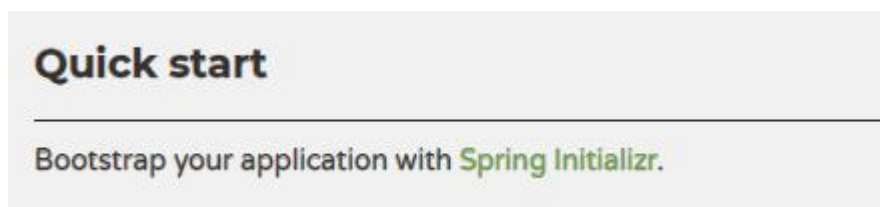
SNAPSHOT：快照版，即开发版。

CURRENT：最新版，但是不一定是稳定版。


GA：General Availability，正式发布的版本。

二、创建基于 Spring Boot 的项目

1 通过官网创建项目



start.spring.io



Spring Initializr
Bootstrap your application

Project

Maven Project | Gradle Project

Language

Java | Kotlin | Groovy

Spring Boot

2.2.1 (SNAPSHOT) | **2.2.0** | 2.1.10 (SNAPSHOT) | 2.1.9

Project Metadata

Group
com.bjsxt

Artifact
springbootdemo

Options

Name
springbootdemo

Description
Demo project for Spring Boot

Package Name
com.bjsxt.springbootdemo

Generate - Ctrl + G | Explore - Ctrl + Space | Share...

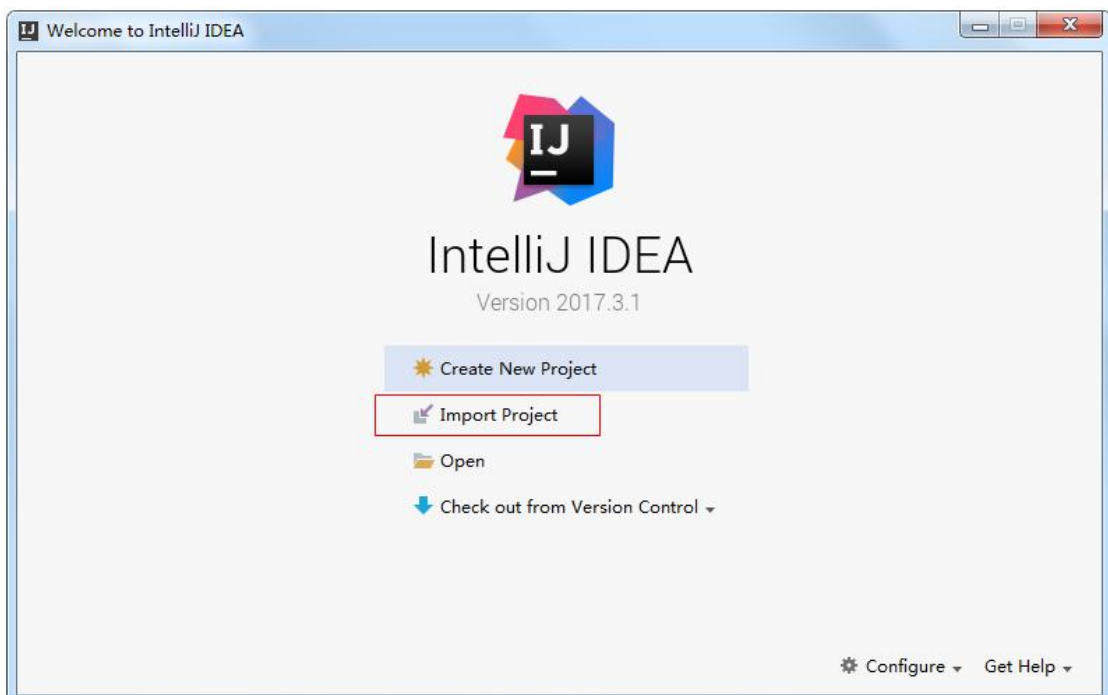
© 2013-2019 Pivotal Software
start.spring.io is powered by
[Spring Initializr](#) and [Pivotal Web Services](#)

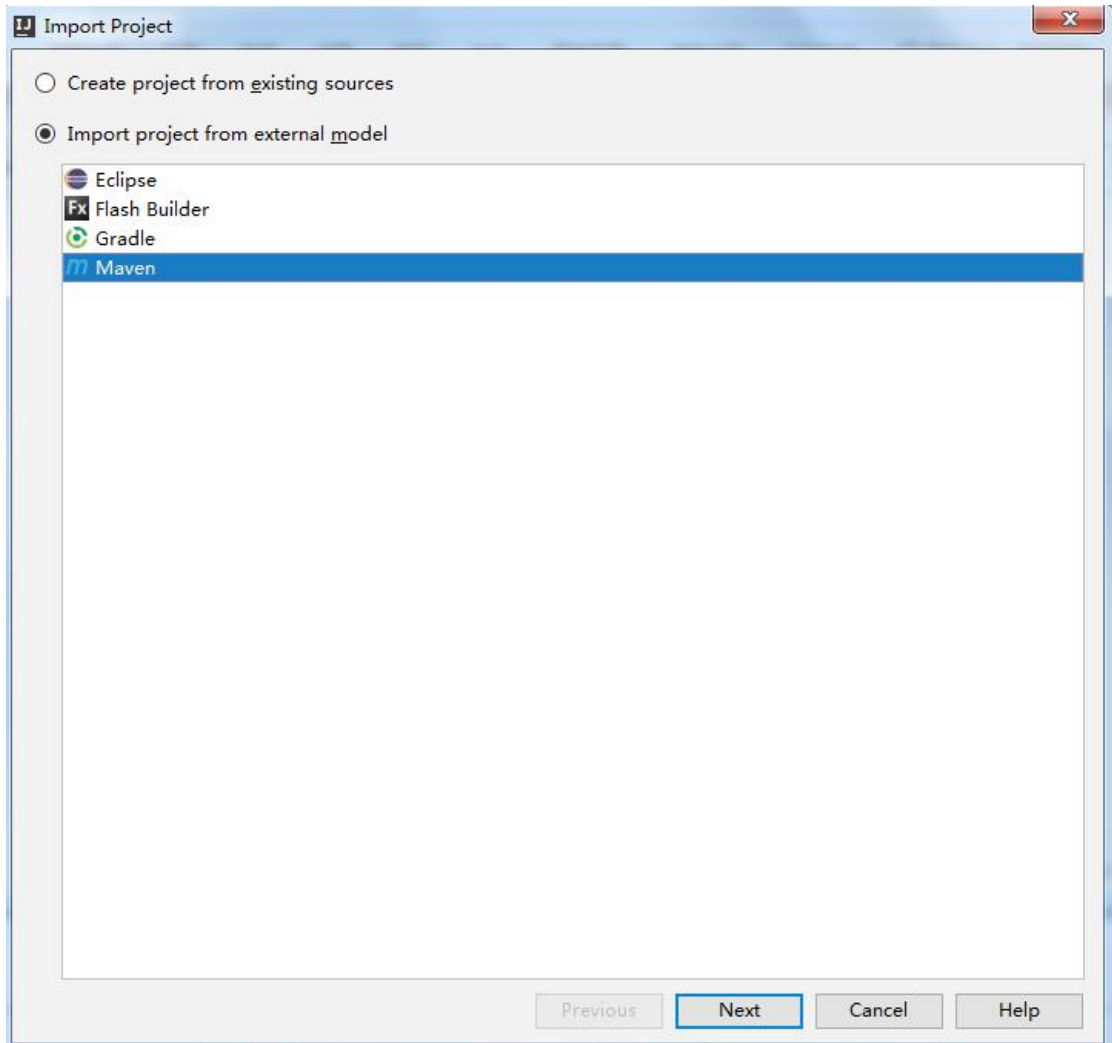
© 2013-2019 Pivotal Software
start.spring.io is powered by
[Spring Initializr](#) and [Pivotal Web Services](#)

Package Name

Generate - Ctrl + G | Explore - Ctrl + Space | Share...

springbootdemo.zip





Import Project

Root directory: C:\springbootdemo\springbootdemo

☐ Search for projects recursively

Project format: .idea (directory based)

☐ Keep project files in:

☐ Import Maven projects automatically

☒ Create IntelliJ IDEA modules for aggregator projects (with 'pom' packaging)

☐ Create module groups for multi-module Maven projects

☒ Keep source and test folders on reimport

☒ Exclude build directory (%PROJECT_ROOT%/target)

☒ Use Maven output directories

Generated sources folders: Detect automatically

Phase to be used for folders update: process-resources

IDEA needs to execute one of the listed phases in order to discover all source folders that are configured via Maven plugins.
Note that all test-* phases firstly generate and compile production sources.

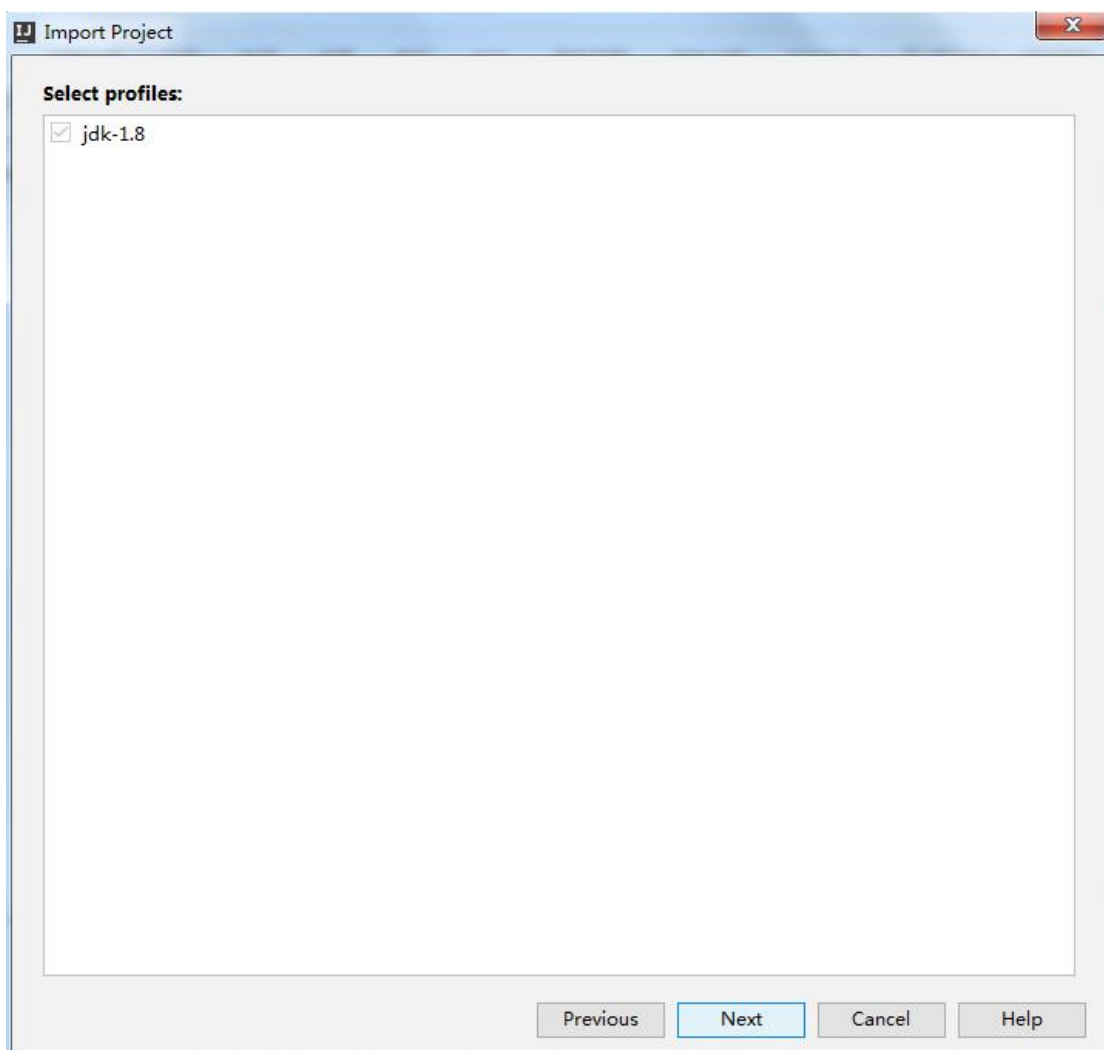
Automatically download: ☐ Sources ☐ Documentation

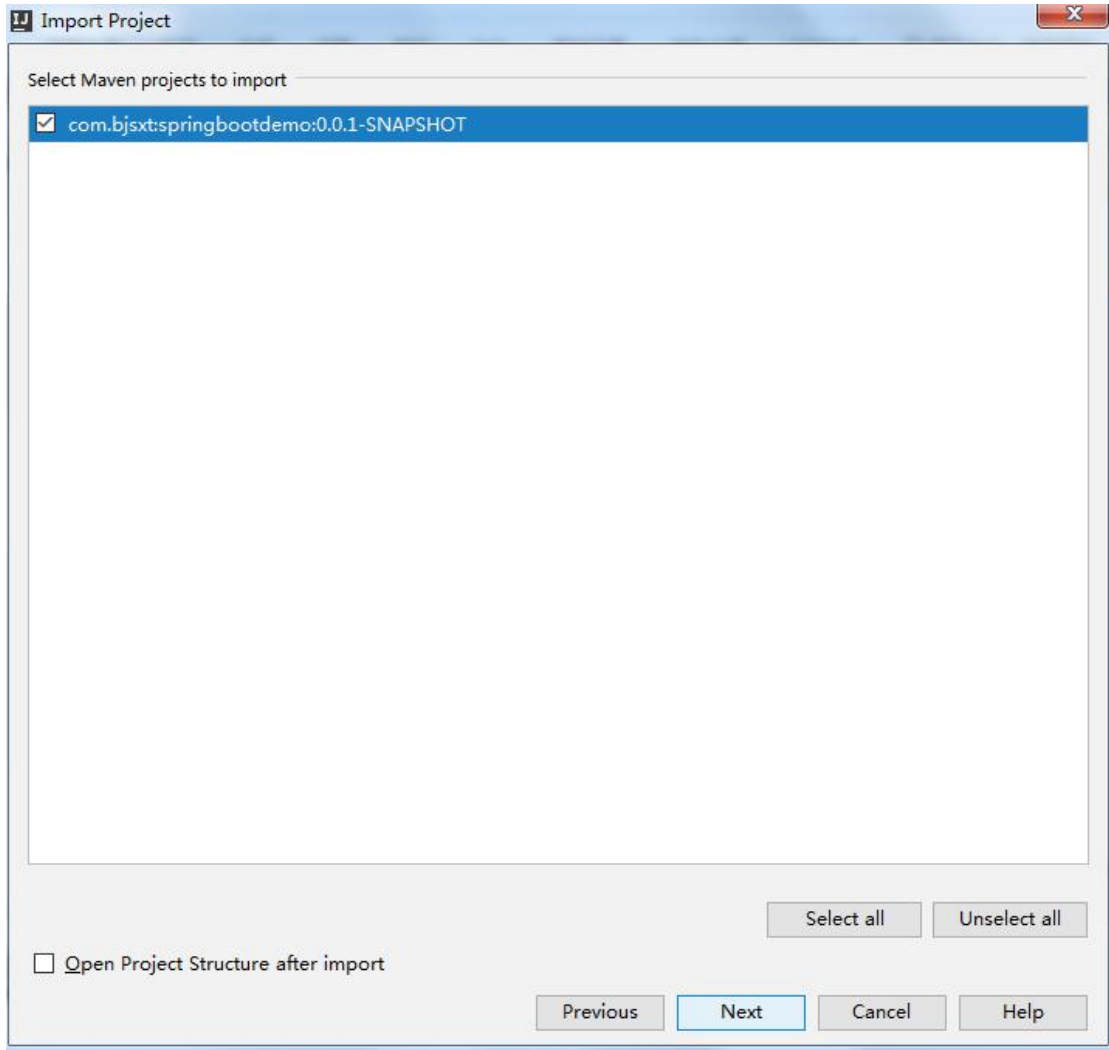
Dependency types: jar, test-jar, maven-plugin, ejb, ejb-client, jboss-har, jboss-sar, war, ear, bundle

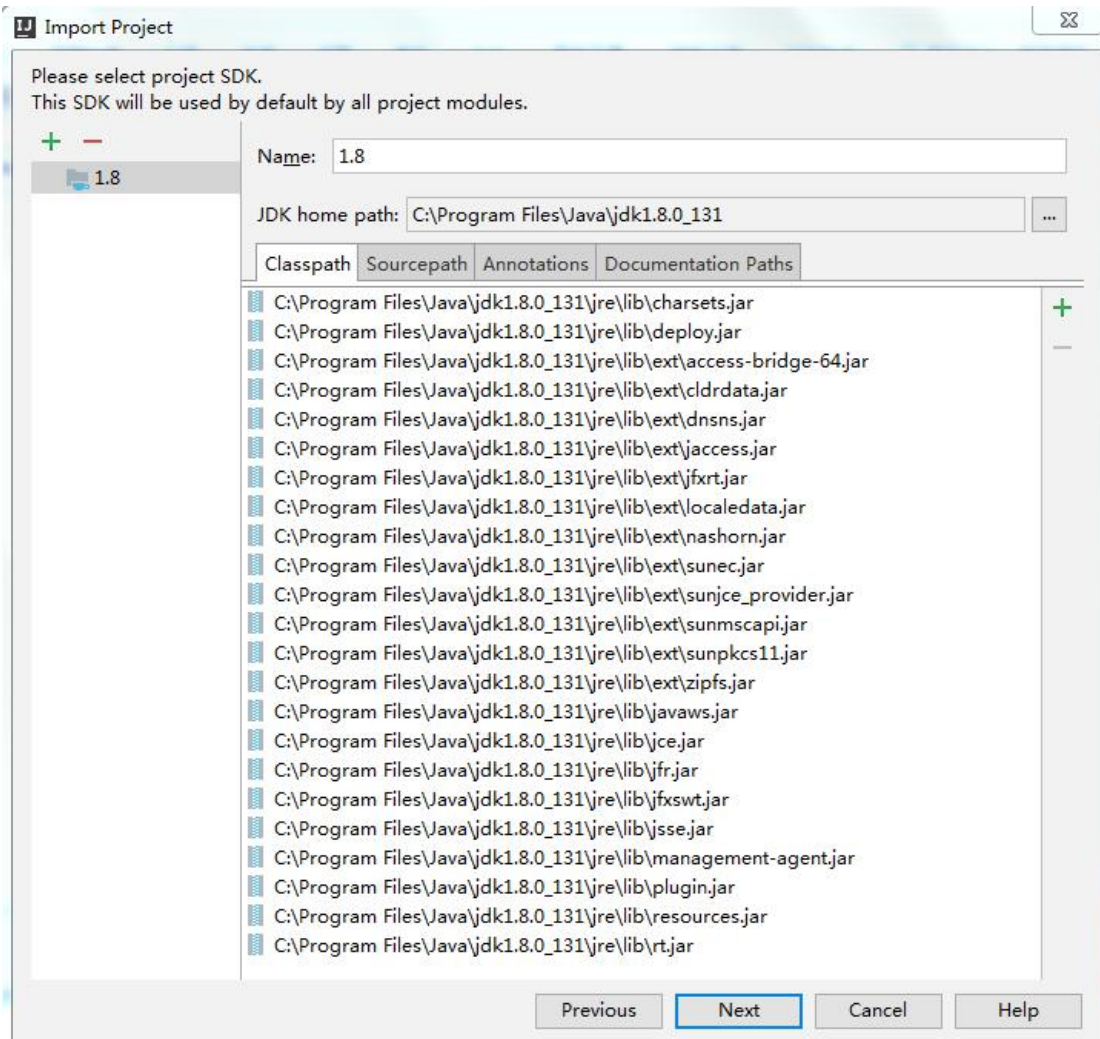
Comma separated list of dependency types that should be imported

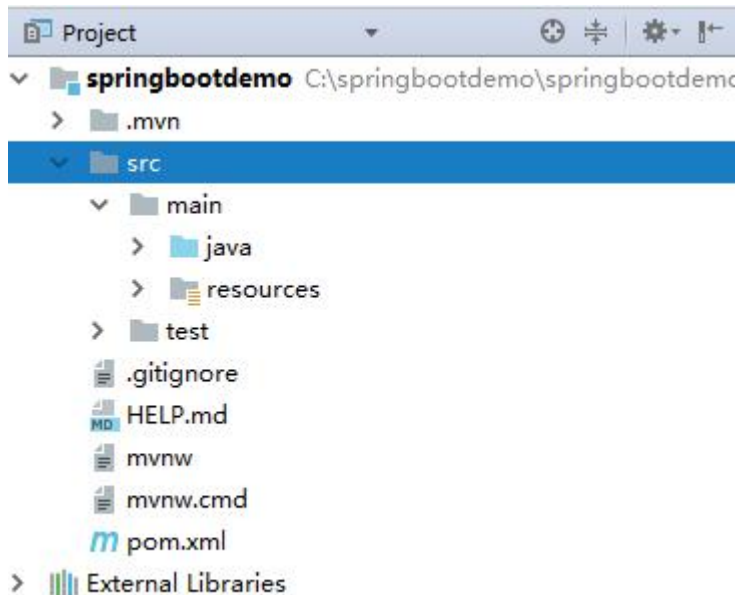
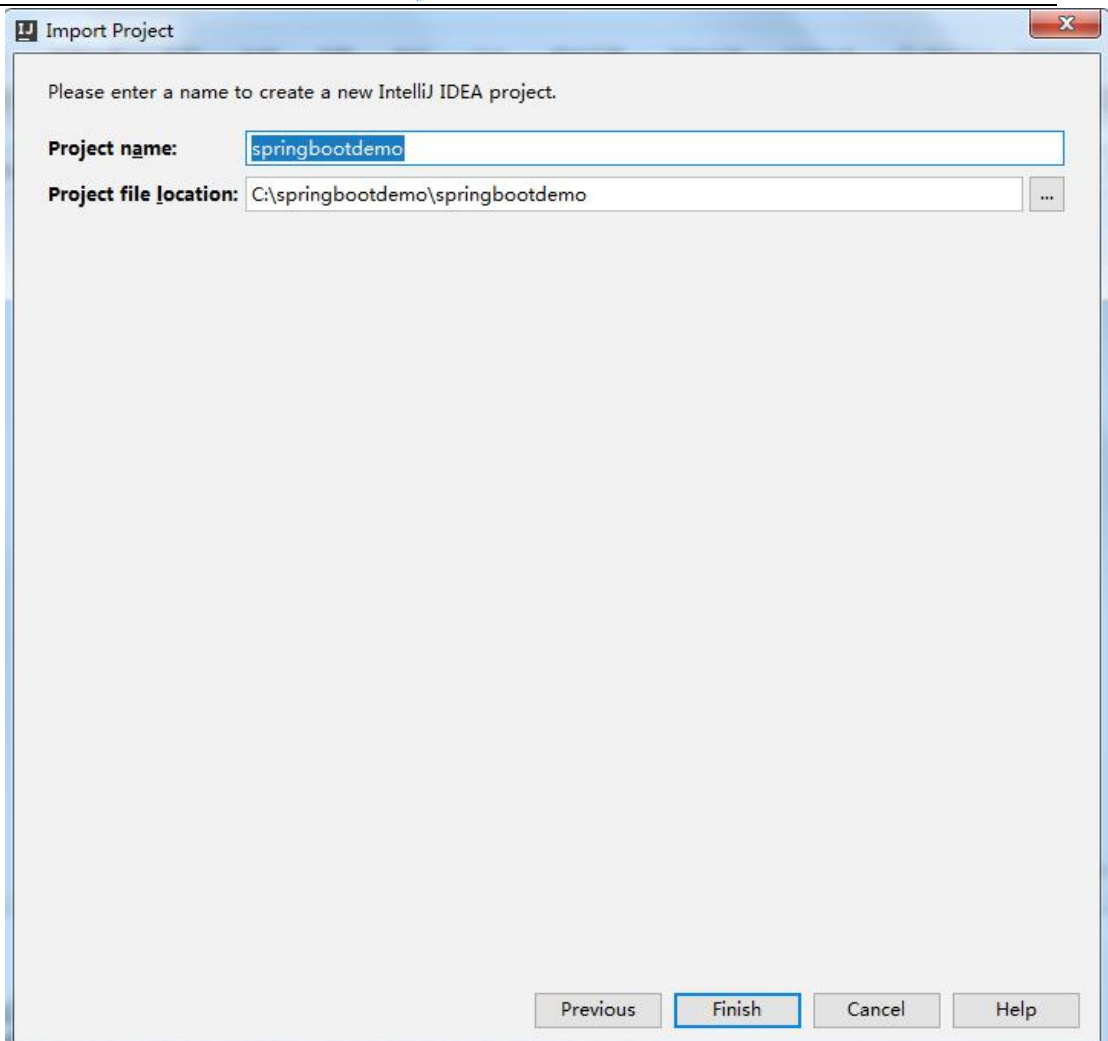
Environment settings...

Previous Next Cancel Help

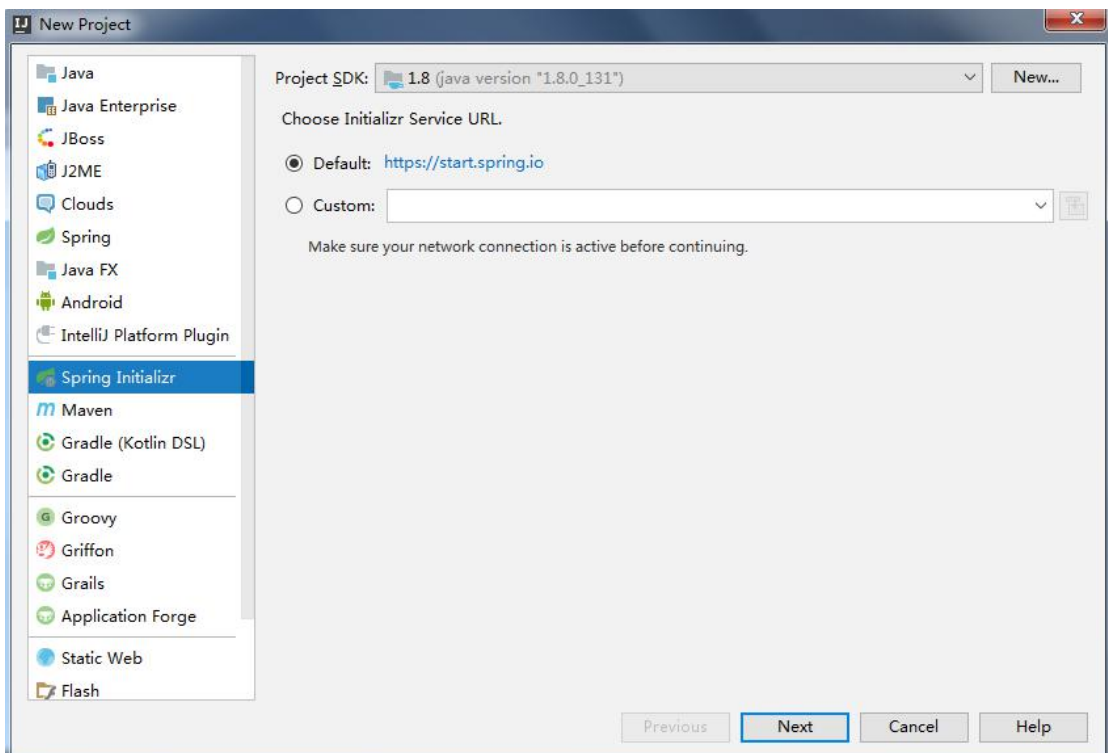








2 通过 IDEA 的脚手架工具创建



New Project

Project Metadata

Group: com.bjsxt

Artifact: springbootdemo2

Type: Maven Project (Generate a Maven based project archive.)

Language: Java

Packaging: Jar

Java Version: 8

Version: 0.0.1-SNAPSHOT

Name: springbootdemo2

Description: Demo project for Spring Boot

Package: com.bjsxt.springbootdemo2

Previous Next Cancel Help

New Project

Dependencies

Spring Boot 2.2.0

Selected Dependencies

Web

Spring Web

Spring Reactive Web

Rest Repositories

Spring Session

Rest Repositories HAL Browser

Spring HATEOAS

Spring Web Services

Jersey

Vaadin

Spring Web

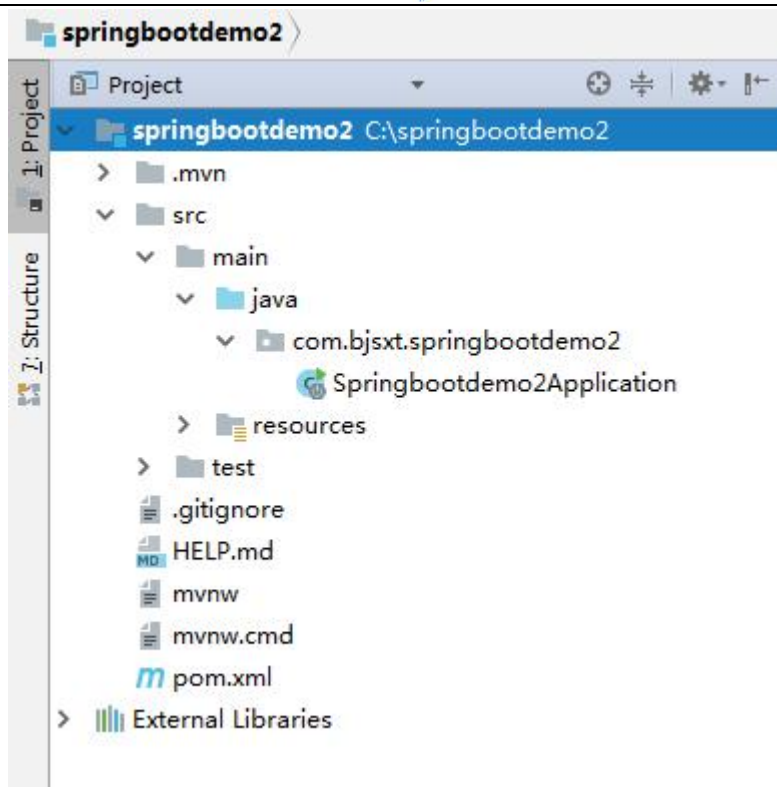
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Building a RESTful Web Service

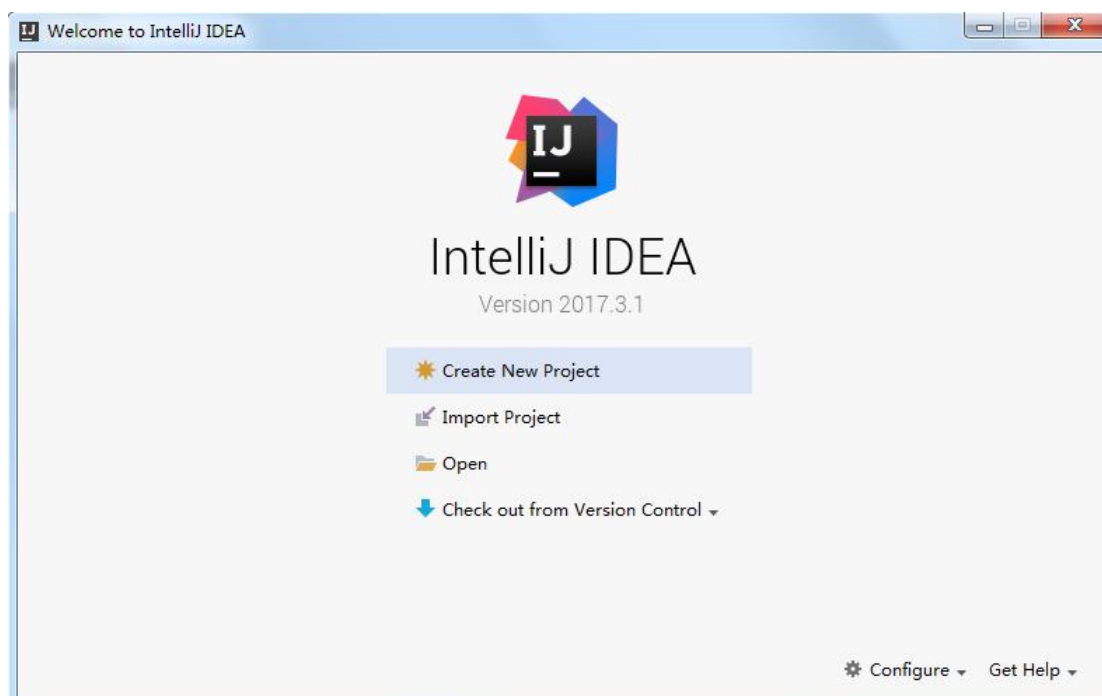
Serving Web Content with Spring MVC

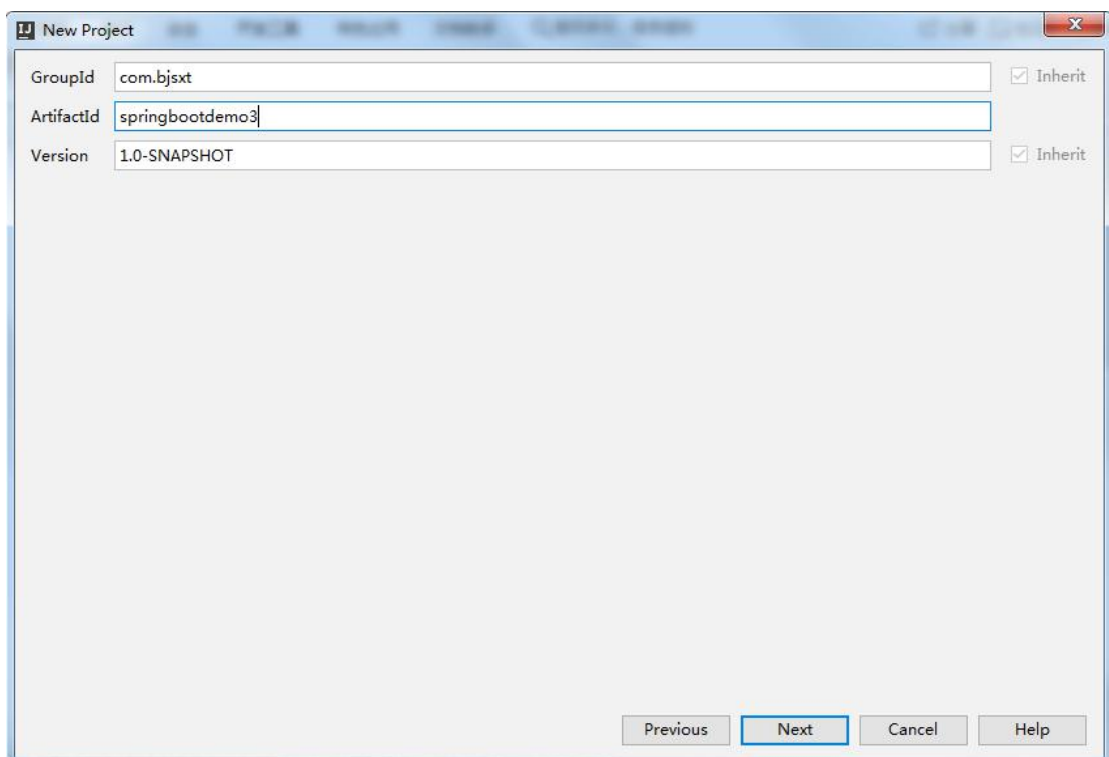
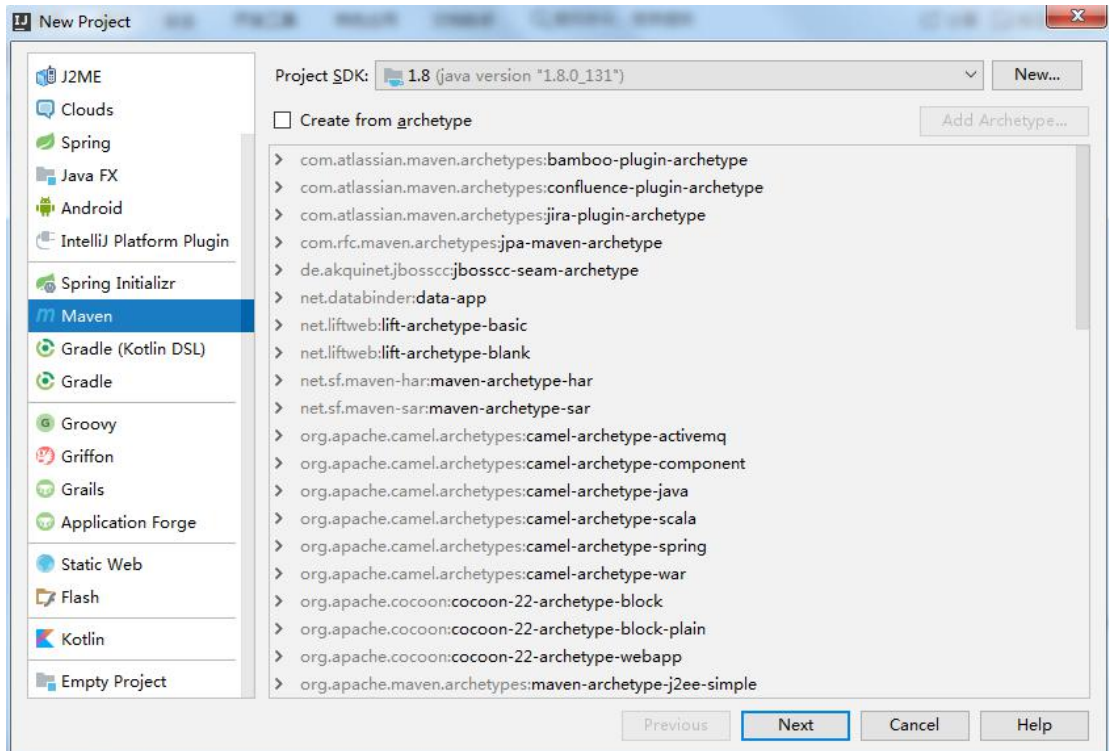
Building REST services with Spring

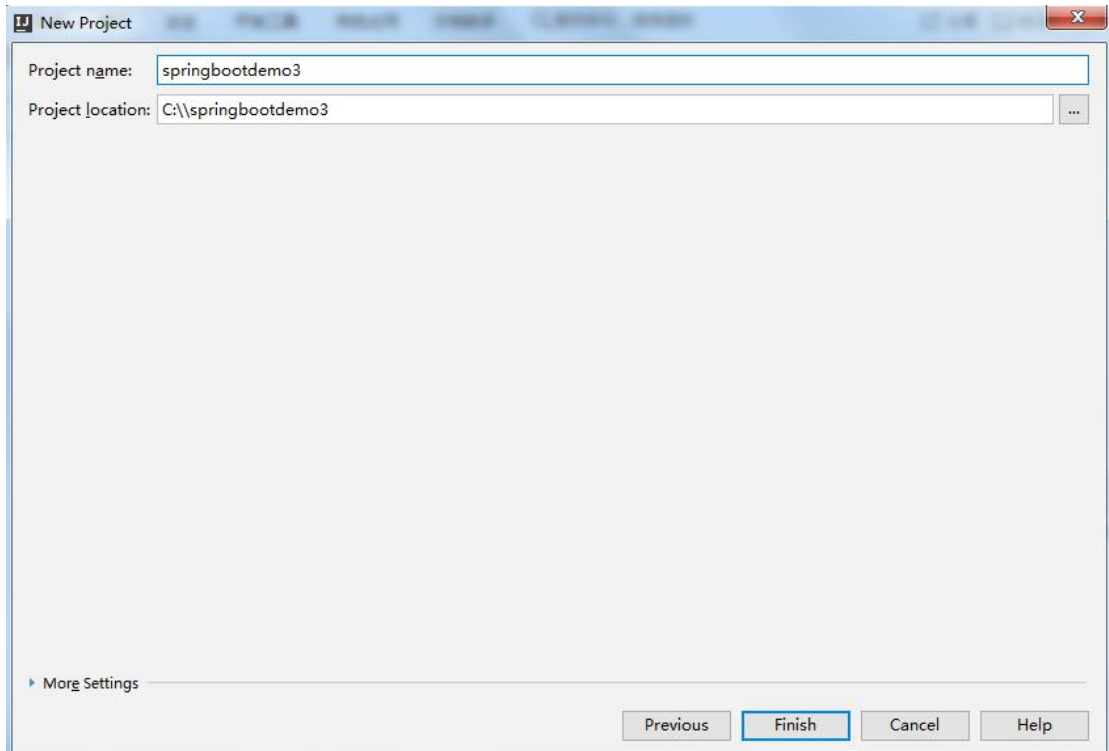
Previous Next Cancel Help



3 通过 IDEA 的 Maven 项目创建







修改 POM 文件

```
<?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.bjsxt</groupId>
  <artifactId>springbootdemo3</artifactId>
  <version>1.0-SNAPSHOT</version>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

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

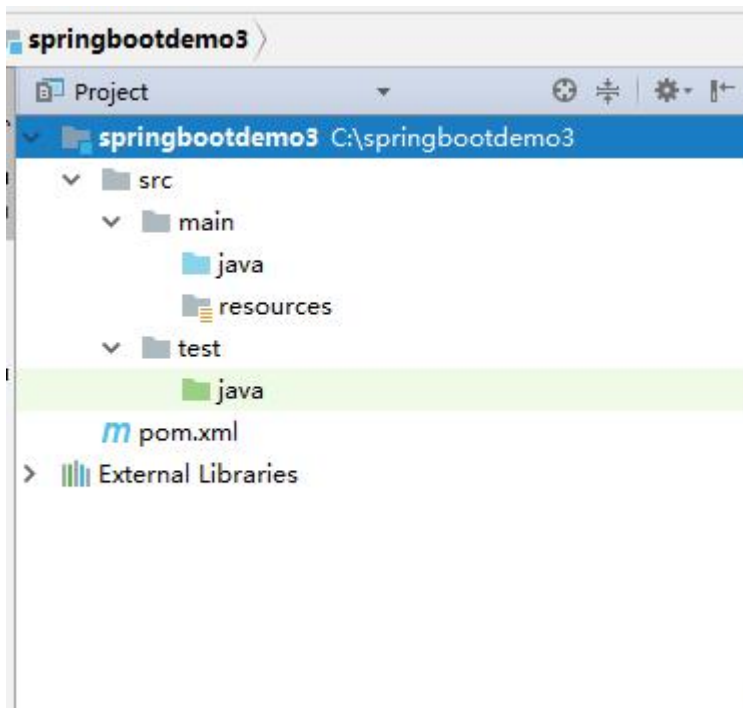
```

<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>
    <exclusions>
      <exclusion>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>

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

```



三、 Spring Boot 项目结构介绍

1 POM 文件

1.1 继承

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.2.0.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Spring Boot 的父级依赖，只有继承它项目才是 Spring Boot 项目。

spring-boot-starter-parent 是一个特殊的 starter，它用来提供相关的 Maven 默认依赖。使用它之后，常用的包依赖可以省去 version 标签。

1.2 依赖

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

启动器依赖

1.3 插件

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

spring-boot-maven-plugin 插件是将 springboot 的应用程序打包成 jar 包的插件。将所有应用启动运行所需要的 jar 包都包含进来，从逻辑上将具备了独立运行的条件。当运行"mvn package"进行打包后，使用"java -jar"命令就可以直接运行。

2 启动类

Spring Boot 的启动类的作用是启动 Spring Boot 项目，是基于 Main 方法来运行的。

注意：启动类在启动时会做注解扫描(@Controller、@Service、@Repository.....)，扫描位置为同包或者子包下的注解，所以启动类的位置应放于包的根下。

2.1 启动类与启动器区别：

- ◆ 启动类表示项目的启动入口
- ◆ 启动器表示 jar 包的坐标

2.2 创建启动类

```
/**
 *Spring Boot 启动类
 */
@SpringBootApplication
public class SpringBootDemo3Application {
    public static void main(String[] args){

SpringApplication.run(SpringBootDemo3Application.class,ar
gs);
    }
}
```

3 启动器

Spring Boot 将所有的功能场景都抽取出来，做成一个个的 **starter**(启动器)，只需要在项目里面引入这些 **starter** 相关场景的所有依赖都会导入进来，要用什么功能就导入什么场景，在 **jar** 包管理上非常方便，最终实现一站式开发。

Spring Boot 提供了多达 44 个启动器。

spring-boot-starter

这是 Spring Boot 的核心启动器，包含了自动配置、日志和 YAML。

spring-boot-starter-actuator

帮助监控和管理应用。

spring-boot-starter-web

支持全栈式 Web 开发，包括 Tomcat 和 spring-webmvc。

spring-boot-starter-amqp

通过 spring-rabbit 来支持 AMQP 协议（Advanced Message Queuing Protocol）。

spring-boot-starter-aop

支持面向方面的编程即 AOP，包括 spring-aop 和 AspectJ。

spring-boot-starter-artemis

通过 Apache Artemis 支持 JMS 的 API（Java Message Service API）。

spring-boot-starter-batch

支持 Spring Batch，包括 HSQLDB 数据库。

spring-boot-starter-cache

支持 Spring 的 Cache 抽象。

spring-boot-starter-cloud-connectors

支持 Spring Cloud Connectors，简化了在像 Cloud Foundry 或 Heroku 这样的云平台上连接服务。

spring-boot-starter-data-elasticsearch

支持 Elasticsearch 搜索和分析引擎，包括 spring-data-elasticsearch。

spring-boot-starter-data-gemfire

支持 GemFire 分布式数据存储，包括 spring-data-gemfire。

spring-boot-starter-data-jpa

支持 JPA（Java Persistence API），包括 spring-data-jpa、spring-orm、Hibernate。

spring-boot-starter-data-mongodb

支持 MongoDB 数据，包括 spring-data-mongodb。

spring-boot-starter-data-rest

通过 spring-data-rest-webmvc，支持通过 REST 暴露 Spring Data 数据仓库。

spring-boot-starter-data-solr

支持 Apache Solr 搜索平台，包括 spring-data-solr。

spring-boot-starter-freemarker

支持 FreeMarker 模板引擎。

spring-boot-starter-groovy-templates

支持 Groovy 模板引擎。

spring-boot-starter-hateoas

通过 spring-hateoas 支持基于 HATEOAS 的 RESTful Web 服务。

spring-boot-starter-hornetq

通过 HornetQ 支持 JMS。

spring-boot-starter-integration

支持通用的 spring-integration 模块。

spring-boot-starter-jdbc

支持 JDBC 数据库。

spring-boot-starter-jersey

支持 Jersey RESTful Web 服务框架。

spring-boot-starter-jta-atomikos

通过 Atomikos 支持 JTA 分布式事务处理。

spring-boot-starter-jta-bitronix

通过 Bitronix 支持 JTA 分布式事务处理。

spring-boot-starter-mail

支持 javax.mail 模块。

spring-boot-starter-mobile

支持 spring-mobile。

spring-boot-starter-mustache

支持 Mustache 模板引擎。

spring-boot-starter-redis

支持 Redis 键值存储数据库，包括 spring-redis。

spring-boot-starter-security

支持 spring-security。

spring-boot-starter-social-facebook

支持 spring-social-facebook

spring-boot-starter-social-linkedin

支持 spring-social-linkedin

spring-boot-starter-social-twitter

支持 spring-social-twitter

spring-boot-starter-test

支持常规的测试依赖，包括 JUnit、Hamcrest、Mockito 以及 spring-test 模块。

spring-boot-starter-thymeleaf

支持 Thymeleaf 模板引擎，包括与 Spring 的集成。

spring-boot-starter-velocity

支持 Velocity 模板引擎。

spring-boot-starter-websocket

支持 WebSocket 开发。

spring-boot-starter-ws

支持 Spring Web Services。

spring-boot-starter-actuator

增加了面向产品上线相关的功能，比如测量和监控。

spring-boot-starter-remote-shell

增加了远程 ssh shell 的支持。

spring-boot-starter-jetty

引入了 Jetty HTTP 引擎（用于替换 Tomcat）。

spring-boot-starter-log4j

支持 Log4J 日志框架。

spring-boot-starter-logging

引入了 Spring Boot 默认的日志框架 Logback。

spring-boot-starter-tomcat

引入了 Spring Boot 默认的 HTTP 引擎 Tomcat。

spring-boot-starter-undertow

引入了 Undertow HTTP 引擎（用于替换 Tomcat）。

4 配置文件

Spring Boot 提供一个名称为 application 的全局配置文件，支持两种格式 properties 格式与 YAML 格式。

4.1 Properties 格式

配置 Tomcat 监听端口

```
server.port=8888
```

4.2 YAML 格式

YAML 格式配置文件的扩展名可以是 yaml 或者 yml。

4.2.1 基本格式要求

- 大小写敏感
- 使用缩进代表层级关系
- 相同的部分只出现一次

配置 Tomcat 监听端口

```
server:  
  port: 8888
```

4.3 配置文件存放位置

- 当前项目根目录中

- 当前项目根目录下的一个/config 子目录中
- 项目的 resources 即 classpath 根路径中
- 项目的 resources 即 classpath 根路径下的/config 目录中

4.4 配置文件加载顺序

4.4.1 不同格式的加载顺序

如果同一个目录下，有 application.yml 也有 application.properties，默认先读取 application.properties。

如果同一个配置属性，在多个配置文件都配置了，默认使用第 1 个读取到的，后面读取的不覆盖前面读取到的。

4.4.2 不同位置的加载顺序

4.4.2.1 当前项目根目录下的一个/config 子目录中(最高)

config/application.properties
config/application.yml

4.4.2.2 当前项目根目录中(其次)

application.properties
application.yml

4.4.2.3 项目的 resources 即 classpath 根路径下的/config 目录中(一般)

resources/config/application.properties
resources/config/application.yml

4.4.2.4 项目的 resources 即 classpath 根路径中(最后)

resources/application.properties
resources/application.yml

4.5 配置文件中的占位符

4.5.1 占位符语法

语法：\${}

4.5.2 占位符作用

- "\${}"中可以获取框架提供的方法中的值如：random.int 等。
- 占位符可以获取配置文件中的键的值赋给另一个键作为值。

4.5.3 生成随机数

`${random.value}` - 类似 uuid 的随机数，没有“-”连接

`${random.int}` - 随机取整型范围内的一个值

`${random.long}` - 随机取长整型范围内的一个值

`${random.long(100,200)}` - 随机生成长整型 100-200 范围内的一个值

`${random.uuid}` - 生成一个 uuid，有短杠连接

`${random.int(10)}` - 随机生成一个 10 以内的数

`${random.int(100,200)}` - 随机生成一个 100-200 范围以内的数

4.6 bootstrap 配置文件

4.6.1 bootstrap 配置文件介绍

Spring Boot 中有两种上下文对象，一种是 bootstrap，另外一种是 application，bootstrap 是应用程序的父上下文，也就是说 bootstrap 加载优先于 application。bootstrap 主要用于从额外的资源来加载配置信息，还可以在本地外部配置文件中解密属性。这两个上下文共用一个环境，它是任何 Spring 应用程序的外部属性的来源。bootstrap 里面的属性会优先加载，它们默认也不能被本地相同配置覆盖。

4.6.2 bootstrap 配置文件特征

- bootstrap 由父 ApplicationContext 加载，比 application 优先加载。
- bootstrap 里面的属性不能被覆盖。

4.6.3 bootstrap 与 application 的应用场景

application 配置文件主要用于 Spring Boot 项目的自动化配置。

bootstrap 配置文件有以下几个应用场景。

- 使用 Spring Cloud Config 配置中心时，这时需要在 bootstrap 配置文件中添加连接到配置中心的配置属性来加载外部配置中心的配置信息。
- 一些固定的不能被覆盖的属性。
- 一些加密/解密的场景。

5 Spring Boot 的核心注解

5.1 @SpringBootApplication

是 SpringBoot 的启动类。

此注解等同于 @Configuration+@EnableAutoConfiguration+@ComponentScan 的组合。

5.2 @SpringBootConfiguration

@SpringBootConfiguration 注解是 @Configuration 注解的派生注解，跟 @Configuration 注解的功能一致，标注这个类是一个配置类，只不过 @SpringBootConfiguration 是 springboot 的注解，而 @Configuration 是 spring 的注解

5.3 @Configuration

通过对 bean 对象的操作替代 spring 中 xml 文件

5.4 @EnableAutoConfiguration

Spring Boot 自动配置（auto-configuration）：尝试根据你添加的 jar 依赖自动配置你的 Spring 应用。是 @AutoConfigurationPackage 和 @Import(AutoConfigurationImportSelector.class) 注解的组合。

5.5 @AutoConfigurationPackage

@AutoConfigurationPackage 注解，自动注入主类下所在包下所有的加了注解的类（@Controller，@Service 等），以及配置类（@Configuration）

5.6 @Import({AutoConfigurationImportSelector.class})

直接导入普通的类

导入实现了 ImportSelector 接口的类

导入实现了 ImportBeanDefinitionRegistrar 接口的类

5.7 @ComponentScan

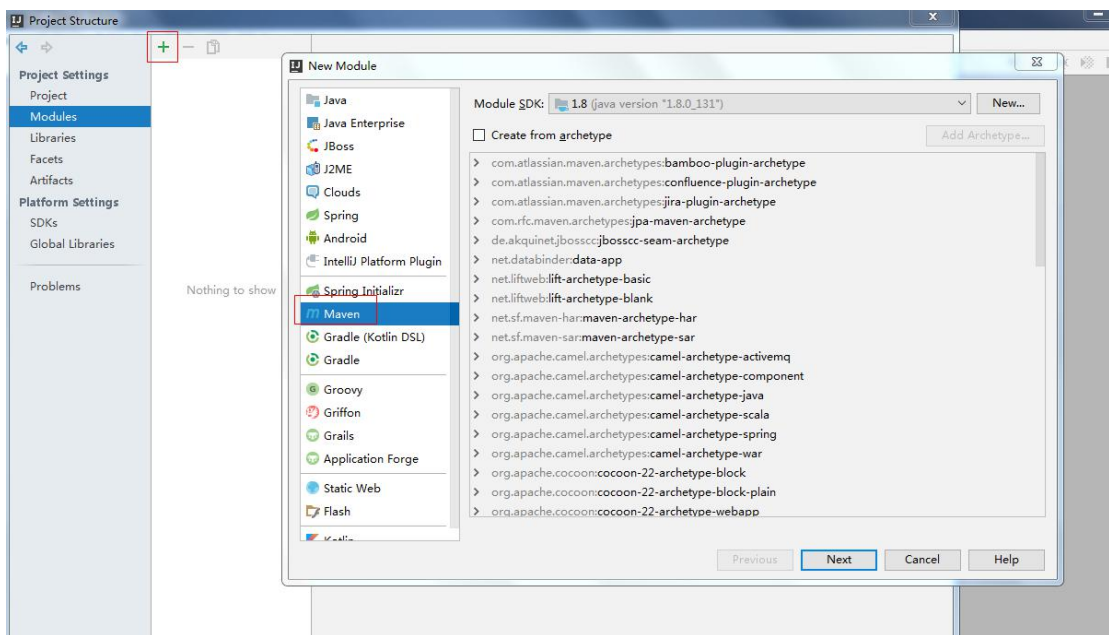
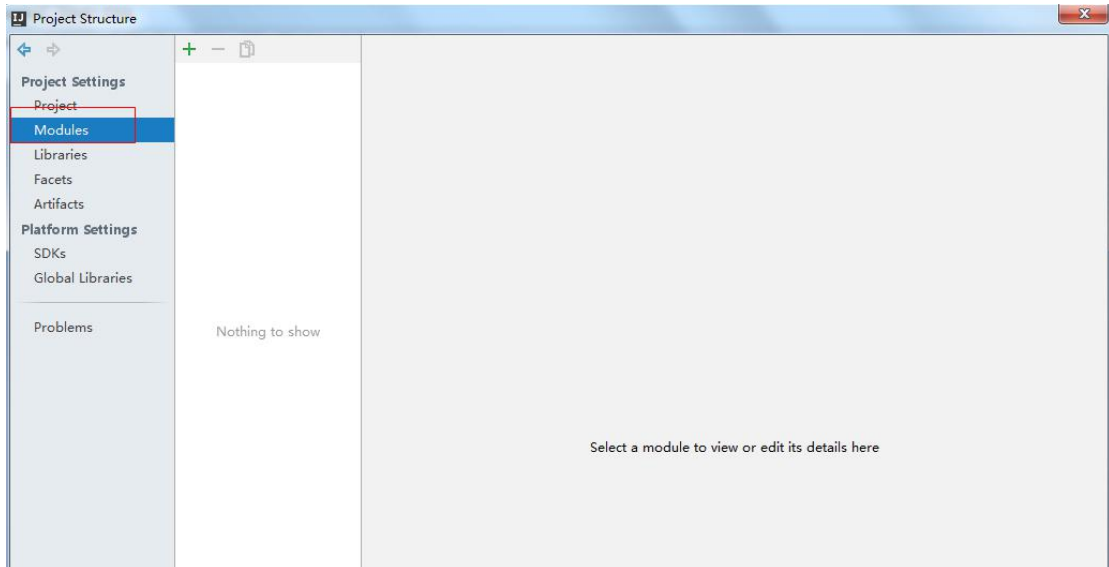
组件扫描，可自动发现和装配一些 Bean。

5.8 @ConfigurationPropertiesScan

@ConfigurationPropertiesScan 扫描配置属性。@EnableConfigurationProperties 注解的作用是使用 @ConfigurationProperties 注解的类生效。

四、 编写 HelloWorld

1 创建项目



New Module

GroupId: ☒ Inherit

ArtifactId:

Version: ☒ Inherit

Previous Next Cancel Help

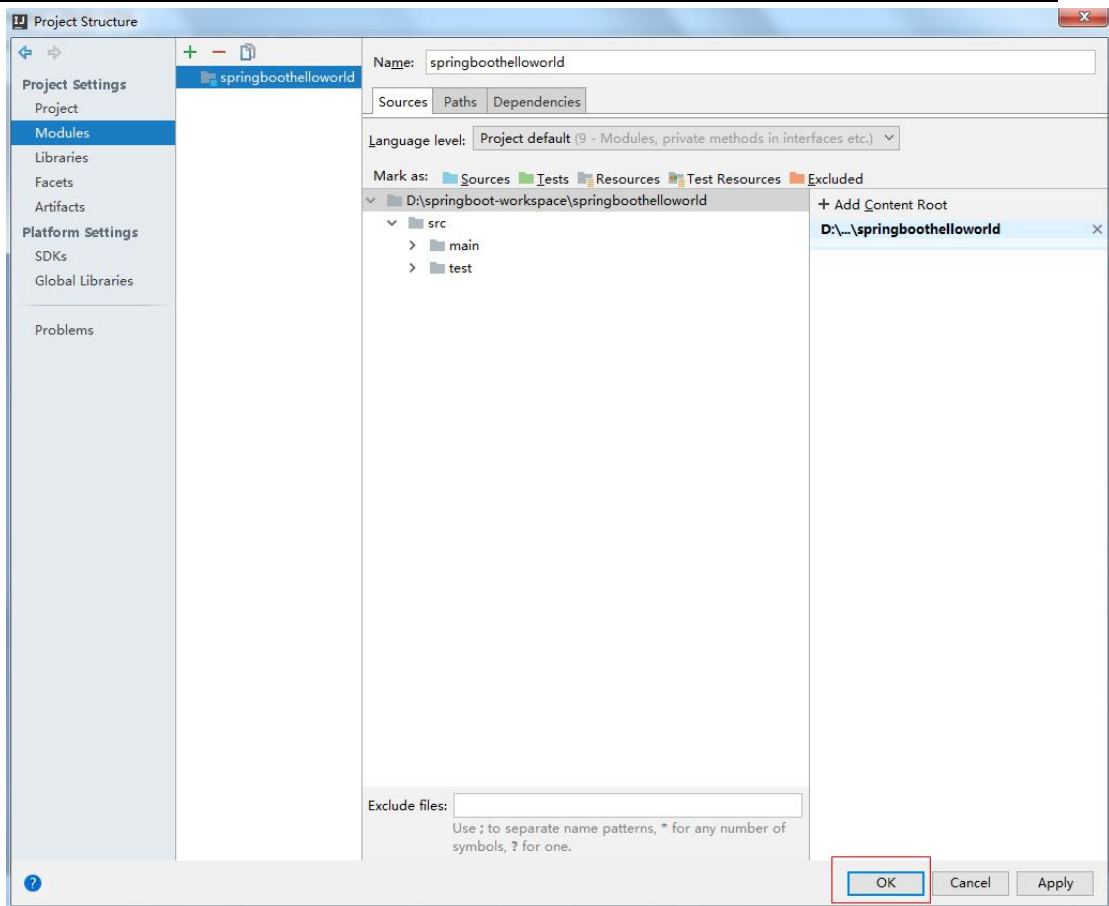
New Module

Module name:

Content root: ...

Module file location: ...

Previous Finish Cancel Help



2 修改 POM 文件

```
<?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.bjsxt</groupId>
  <artifactId>springboothelloworld</artifactId>
  <version>1.0-SNAPSHOT</version>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.0.RELEASE</version>
  </parent>
```

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

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

3 修改 Tomcat 端口

```
server:
  port: 8888
```

4 创建启动类

```
/**
 * 启动类
 */
@SpringBootApplication
public class SpringBootHelloWorldApplication {
    public static void main(String[] args){

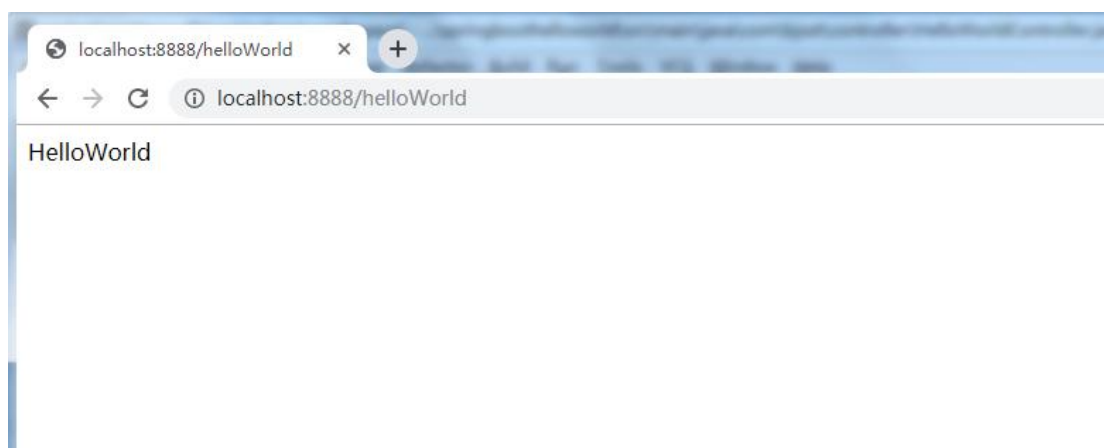
        SpringApplication.run(SpringBootHelloWorldApplication.class, args);
    }
}
```

5 创建 Controller

```
/**
 * 处理请求 Controller
 */

@RestController // @Controller+@ResponseBody 直接返回字符串
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public String showHelloWorld(){
        return "HelloWorld";
    }
}
```



6 Spring Boot 在 Controller 中常用注解

6.1 @RestController

@RestController 相当于@Controller+@ResponseBody 注解

如果使用@RestController 注解 Controller 中的方法无法返回页面，相当于在方法上面自动加了 @ResponseBody 注解，所以没办法跳转并传输数据到另一个页面，所以 InternalResourceViewResolver 也不起作用，返回的内容就是 Return 里的内容。

6.2 @GetMapping

@GetMapping 注解是@RequestMapping(method = RequestMethod.GET)的缩写。

6.3 @PostMapping

@PostMapping 注解是@RequestMapping(method = RequestMethod.POST)的缩写。

6.4 @PutMapping

@PutMapping 注解是@RequestMapping(method = RequestMethod.PUT)的缩写。

6.5 @DeleteMapping

@DeleteMapping 注解是@RequestMapping(method = RequestMethod.DELETE)的缩写。

五、 Spring Boot 整合 Web 层技术

整合 Servlet

1 整合 Servlet 方式一

1.1 通过注解扫描完成 Servlet 组件的注册

1.1.1 创建 Servlet

```
/**
 * 整合 Servlet 方式一
 */
@WebServlet(name = "FirstServlet", urlPatterns = "/first")
public class FirstServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response) {
        System.out.println("First Servlet.....");
    }
}
```


1.1.2修改启动类

```
@SpringBootApplication
@WebServletComponentScan//在 spring Boot 启动时会扫描@WebServlet
注解，并将该类实例化
public class SpringbootwebApplication {

    public static void main(String[] args) {

SpringApplication.run(SpringbootwebApplication.class,
args);
    }
}
```

2 整合 Servlet 方式二

2.1通过方法完成 Servlet 组件的注册

2.1.1创建 Servlet

```
/**
 * 整合 Servlet 方式二
 */
public class SecondServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
HttpServletResponse response){
        System.out.println("Second Servlet.....");
    }
}
```

2.1.2创建 Servlet 配置类

```
/**
 * Servlet 配置类
```

```

*/
@Configuration
public class ServletConfig {

    /**
     * 完成 Servlet 组件的注册
     */
    @Bean
    public ServletRegistrationBean
getServletRegistrationBean() {
        ServletRegistrationBean bean = new
ServletRegistrationBean(new SecondServlet());
        bean.addUrlMappings("/second");
        return bean;
    }
}

```

3 整合 Filter 方式一

3.1 通过注解扫描完成 Filter 组件注册

3.1.1 创建 Filter

```

/**
 * 整合 Filter 方式一
 */
//@WebFilter(filterName = "FirstFilter",urlPatterns =
{"*.do","*.jsp"})
@WebFilter(filterName = "FirstFilter",urlPatterns =
"/first")
public class FirstFilter implements Filter{
    @Override
    public void init(FilterConfig filterConfig) throws
ServletException {

    }
    @Override
    public void doFilter(ServletRequest servletRequest,

```

```
ServletResponse servletResponse, FilterChain filterChain)
throws IOException, ServletException {

    System.out.println("进入 First Filter");

    filterChain.doFilter(servletRequest, servletResponse);

    System.out.println("离开 First Filter");

}
@Override
public void destroy() {

}

}
```

3.1.2 修改启动类

```
@SpringBootApplication
@WebServletScan // 在 spring Boot 启动时会扫描
@WebServlet, @WebFilter 注解，并将该类实例化
public class SpringbootwebApplication {

    public static void main(String[] args) {

        SpringApplication.run(SpringbootwebApplication.class,
            args);
    }

}
```

4 整合 Filter 方式二

4.1 通过方法完成 Filter 组件注册

4.1.1 创建 Filter

```
/**
```

```

* 整合 Filter 方式二
*/
public class SecondFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws
ServletException {

    }
    @Override
    public void doFilter(ServletRequest servletRequest,
ServletResponse servletResponse, FilterChain filterChain)
throws IOException, ServletException {

        System.out.println("进入 Second Filter");

filterChain.doFilter(servletRequest, servletResponse);

        System.out.println("离开 Second Filter");

    }

    @Override
    public void destroy() {

    }

}

```

4.1.2 创建 Filter 配置类

```

/**
 * Filter 配置类
 */
@Configuration
public class FilterConfig {

    @Bean
    public FilterRegistrationBean
getFilterRegistrationBean() {
        FilterRegistrationBean bean = new
FilterRegistrationBean(new SecondFilter());
    }
}

```

```
// bean.addUrlPatterns(new String[]{"*.do","*.jsp"});
bean.addUrlPatterns("/second");
return bean;
}
}
```

5 整合 Listener 方式一

5.1 通过注解扫描完成 Listener 组件注册

5.1.1 编写 Listener

```
/**
 * 整合 Listener
 */
@WebListener
public class FirstListener implements
ServletContextListener{

    public void contextDestroyed(ServletContextEvent event) {

    }

    public void contextInitialized(ServletContextEvent
event) {
        System.out.println("Listener ...Init.....");
    }
}
```

5.1.2 修改启动类

```
@SpringBootApplication
@WebServletComponentScan //在 spring Boot 启动时会扫描
```

@WebServlet,@WebFilter,@WebListener 注解，并将该类实例化

```
public class SpringbootwebApplication {

    public static void main(String[] args) {

SpringApplication.run(SpringbootwebApplication.class,
args);
    }
}
```

6 整合 Listener 方式二

6.1 通过方法完成 Listener 组件注册

6.1.1 编写 Listener

```
/**
 * 整合 Listener 方式二
 */
public class SecondListener implements
ServletContextListener {

    public void contextDestroyed(ServletContextEvent event) {

    }

    public void contextInitialized(ServletContextEvent
event) {

System.out.println("Second....Listener ...Init.....");
    }
}
```

6.1.2 创建 Listener 配置类

```
/**
```

```
* Listener 配置类
*/
@Configuration
public class ListenerConfig {

    @Bean
    public ServletListenerRegistrationBean
getServletListenerRegistrationBean() {
        ServletListenerRegistrationBean bean = new
ServletListenerRegistrationBean(new SecondListener());
        return bean;
    }
}
```

六、 Spring Boot 访问静态资源

在 SpringBoot 项目中没有我们之前常规 web 开发的 WebContent (WebApp)，它只有 src 目录。在 src/main/resources 下面有两个文件夹，static 和 templates。SpringBoot 默认在 static 目录中存放静态页面，而 templates 中放动态页面。

1 static 目录

Spring Boot 通过 classpath/static 目录访问静态资源。注意存放静态资源的目录名称必须是 static。

2 templates 目录

在 Spring Boot 中不推荐使用 jsp 作为视图层技术，而是默认使用 Thymeleaf 来做动态页面。Templates 目录这是存放 Thymeleaf 的页面。

3 静态资源存放其他位置

3.1 Spring Boot 访问静态资源的位置

```
classpath:/META-INF/resources/
classpath:/resources/
classpath:/static/
classpath:/public/
```

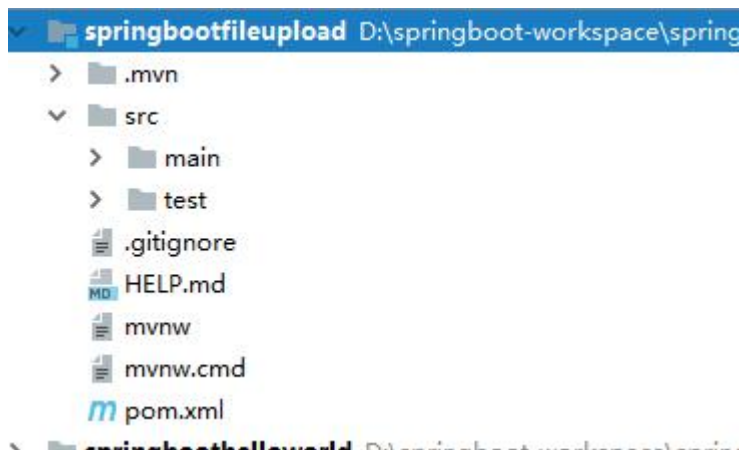

3.2 自定义静态资源位置

#配置静态资源访问路径

```
spring.resources.static-locations=classpath:/suibian/,  
classpath:/static/
```

七、 Spring Boot 文件上传

1 创建项目



2 POM 文件

```
<?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>  
  <parent>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-parent</artifactId>  
    <version>2.2.0.RELEASE</version>  
    <relativePath/> <!-- lookup parent from repository -->  
  </parent>  
  <groupId>com.bjsxt</groupId>  
  <artifactId>springbootfileupload</artifactId>  
  <version>0.0.1-SNAPSHOT</version>
```

```
<name>springbootfileupload</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>
    <exclusions>
      <exclusion>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>

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

3 启动类

```
@SpringBootApplication
public class SpringbootfileuploadApplication {

    public static void main(String[] args) {

SpringApplication.run(SpringbootfileuploadApplication.class, args);
    }

}
```

4 编写上传页面

```
<html>
<head>

</head>

<body>
    <form action="/fileUploadController" method="post"
    enctype="multipart/form-data">
        <input type="file" name="file"/>
        <input type="submit" value="OKOK"/>
    </form>
</body>
```

5 编写 Controller

```
/**
 * 文件上传
 */
@RestController
public class FileUploadController {

    /**
     * 文件上传
     */
}
```

```
@PostMapping("/fileUploadController")
public String fileUpload(MultipartFile file) throws
Exception{
    System.out.println(file.getOriginalFilename());
    file.transferTo(new
File("c:/" + file.getOriginalFilename()));
    return "OK";
}
}
```

6 修改上传文件大小

#配置单个上传文件的大小的限制

spring.servlet.multipart.max-file-size=2MB

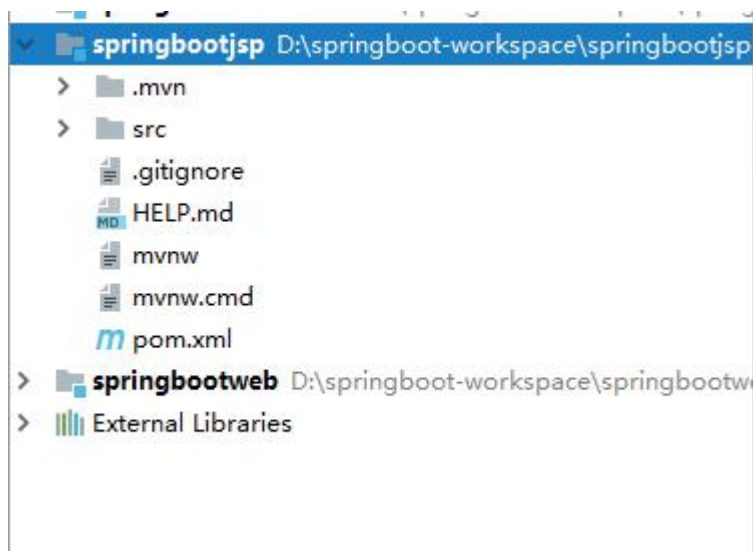
#配置在一次请求中上传文件的总容量的限制

spring.servlet.multipart.max-request-size=20MB

八、 Spring Boot 整合视图层技术

1 Spring Boot 整合 JSP 技术

1.1 创建项目



1.2 修改 POM 文件，添加 JSP 引擎与 JSTL 标签库

```
<?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>2.2.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.bjsxt</groupId>
  <artifactId>springbootjsp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>springbootjsp</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>

    <!--添加 jsp 引擎，SpringBoot 内置的 Tomcat 中没有此依赖-->
    <dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-jasper</artifactId>
    </dependency>

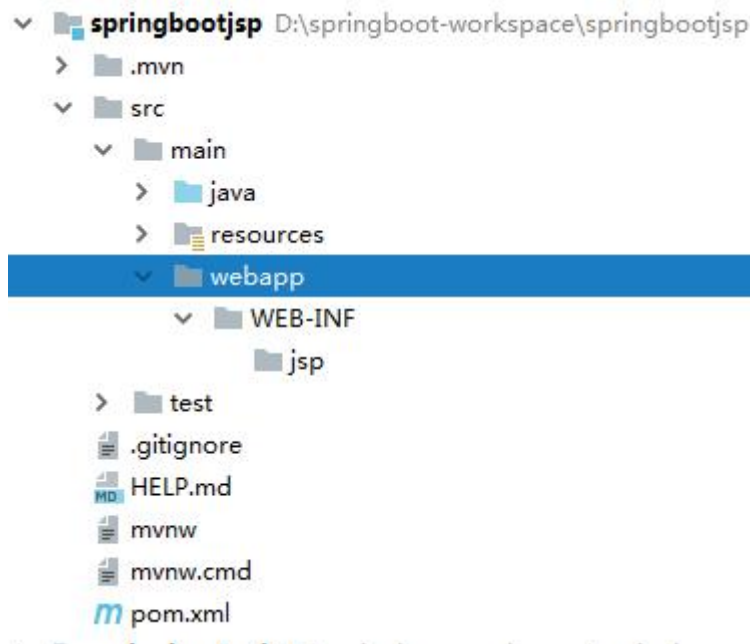
    <!--添加 JSTL 坐标依赖-->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
    </dependency>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <groupId>org.junit.vintage</groupId>
      <artifactId>junit-vintage-engine</artifactId>
    </exclusion>
  </exclusions>
</dependency>
</dependencies>

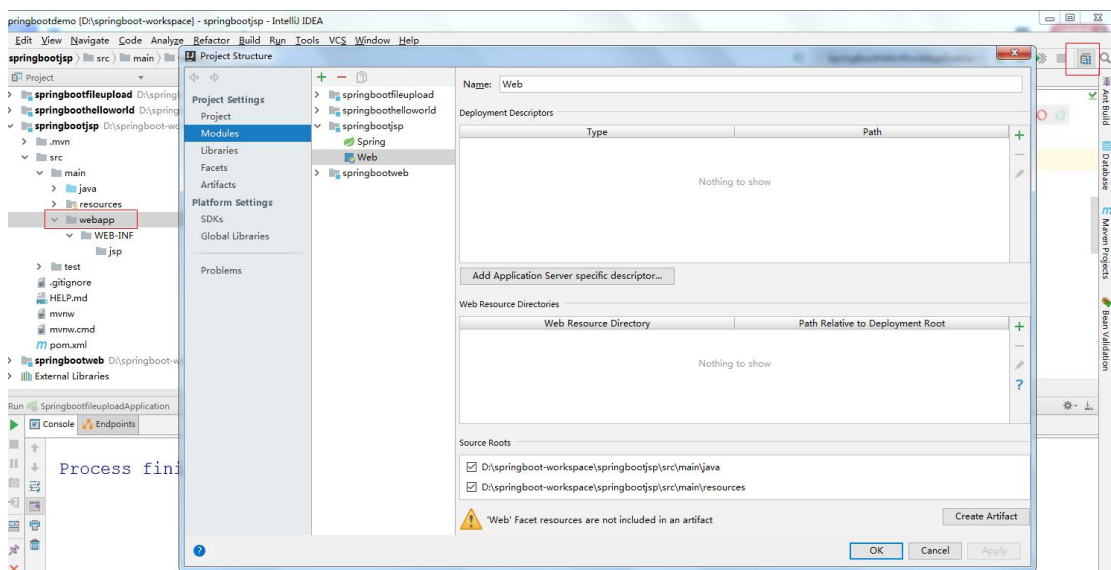
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

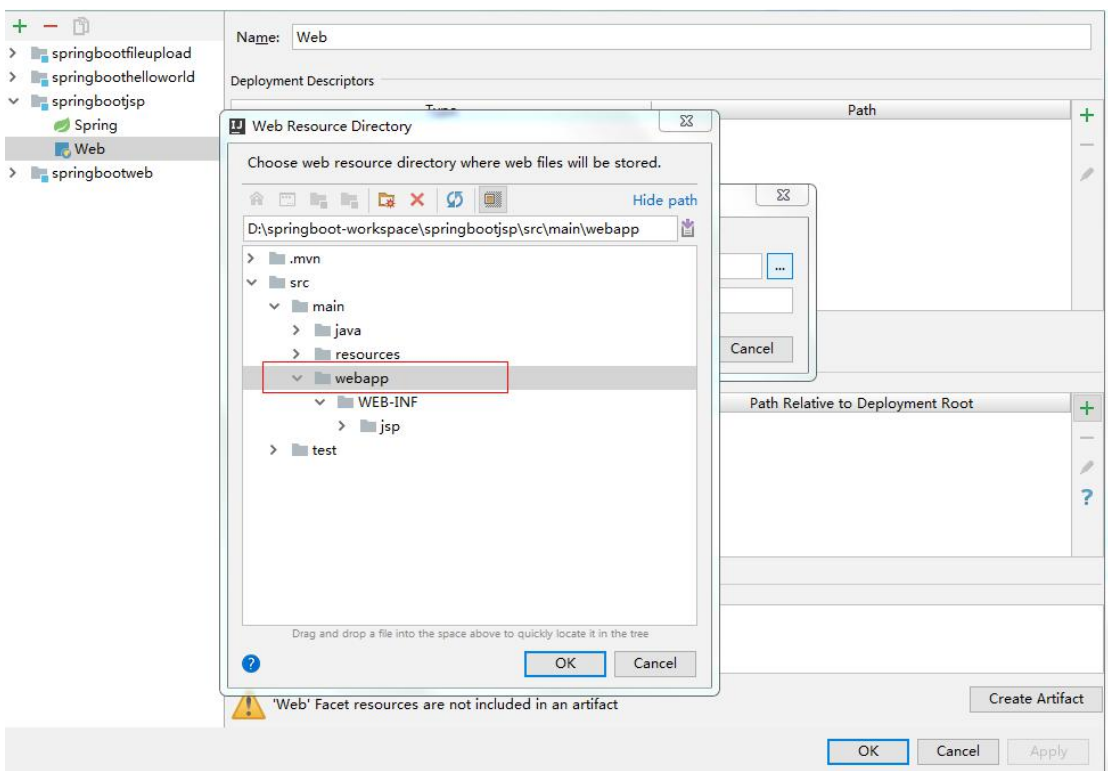
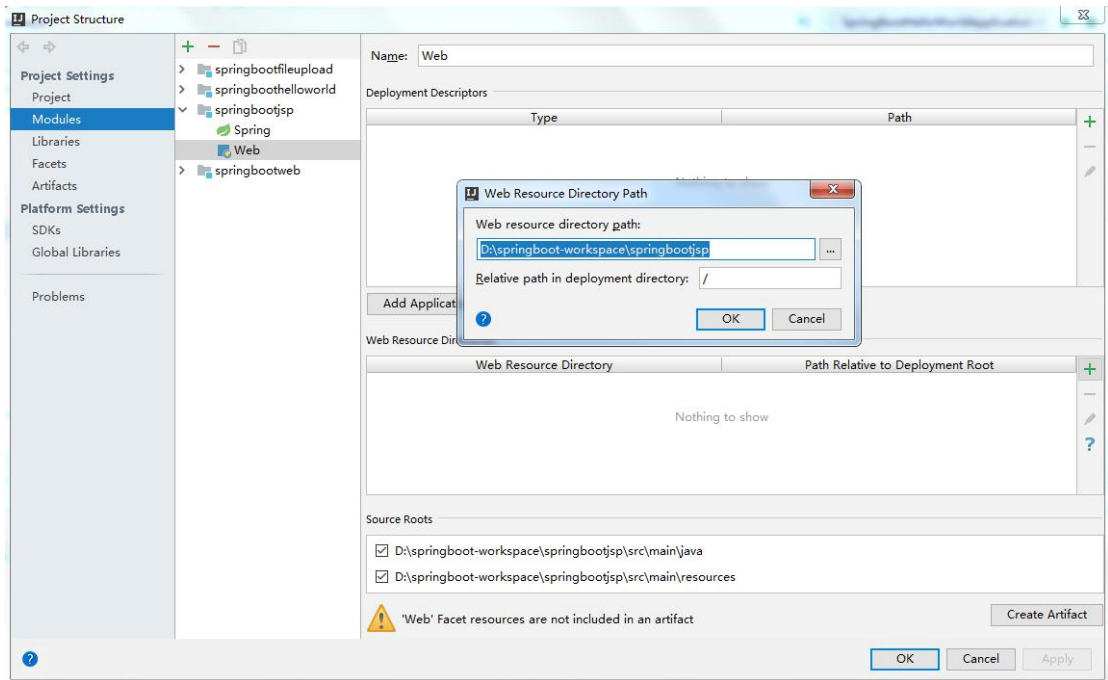
</project>
```

1.3 创建 webapp 目录



1.4 标记为 web 目录





1.5 创建 JSP

```
<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<html>
<head>
```



```
<title>Title</title>
</head>
<body>
  <h2>Hello JSP</h2>
</body>
</html>
```

1.6 修改配置文件，配置视图解析器

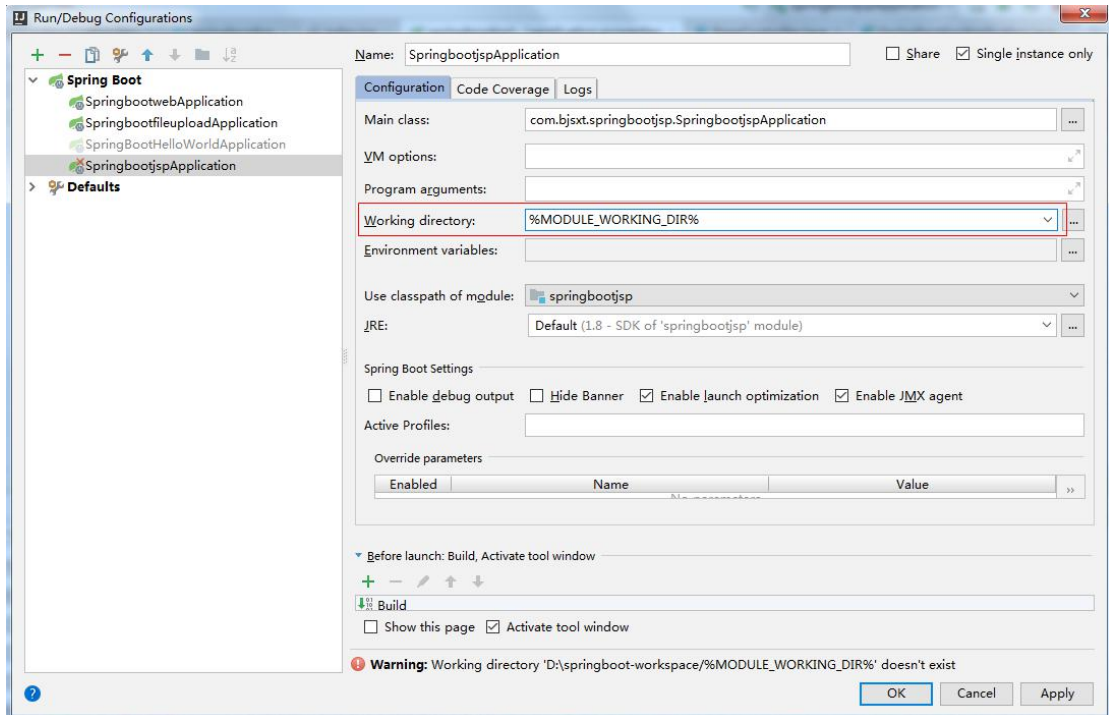
```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

1.7 创建 Controller

```
/**
 * 页面跳转 Controller
 */
@Controller
public class PageController {

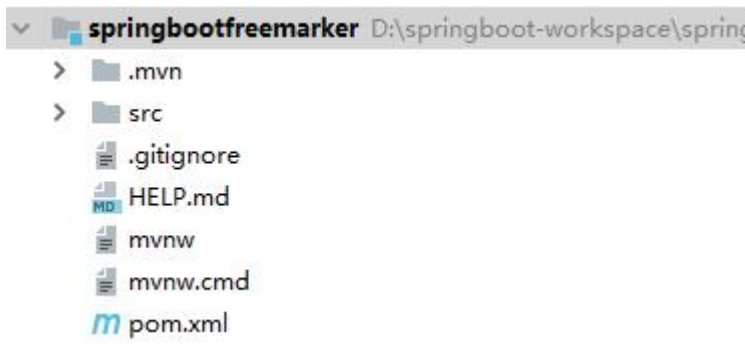
    /**
     * 页面跳转
     */
    @GetMapping("/{page}")
    public String showPage(@PathVariable String page) {
        return page;
    }
}
```

如果在 IDEA 中项目结构为聚合工程。那么在运行 jsp 是需要指定路径。如果项目结构为独立项目则不需要。



2 Spring Boot 整合 Freemarker

2.1 创建项目



2.2 修改 POM 文件，添加 Freemarker 启动器

```
<?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>2.2.0.RELEASE</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.bjsxt</groupId>
<artifactId>springbootfreemarker</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springbootfreemarker</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>

  <!--Freemarker 启动器依赖-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-freemarker</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
      <exclusion>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>

<build>

```

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

</project>
```

2.3 创建 Users 实体

```
public class Users {
    private String username;
    private String usersex;
    private String usage;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getUsersex() {
        return usersex;
    }

    public void setUsersex(String usersex) {
        this.usersex = usersex;
    }

    public String getUsage() {
        return usage;
    }

    public void setUsage(String usage) {
        this.usage = usage;
    }
}
```

```
public Users(String username, String usersex, String
usage) {
    this.username = username;
    this.usersex = usersex;
    this.usage = usage;
}

public Users() {
}
}
```

2.4 创建 Controller

```
/**
 * UsersController
 */
@Controller
public class UsersController {

    /**
     * 处理请求, 返回数据
     */
    @GetMapping("/showUsers")
    public String showUsers(Model model){
        List<Users> list = new ArrayList<>();
        list.add(new Users("admin", "F", "32"));
        list.add(new Users("Lisi", "M", "23"));
        list.add(new Users("xiaoli", "F", "23"));
        model.addAttribute("list", list);
        return "userList";
    }
}
```

2.5 创建视图

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <table border="1" align="center" width="50%">
    <tr>
      <th>Name</th>
      <th>Sex</th>
      <th>Age</th>
    </tr>

    <#list list as user>
      <tr>
        <td>${user.username}</td>
        <td>${user.usersex}</td>
        <td>${user.userage}</td>
      </tr>
    </#list>
  </table>
</body>
```

2.6 修改配置文件添加后缀

```
spring.freemarker.suffix=.ftl
```

3 Spring Boot 整合 Thymeleaf

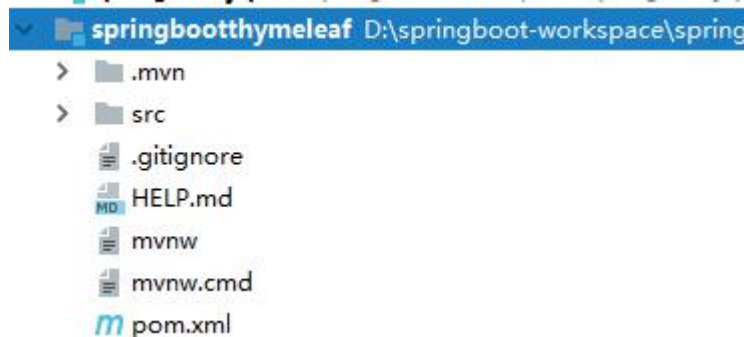
3.1 Thymeleaf 介绍

Thymeleaf 的主要目标是将优雅的自然模板带到开发工作流程中，并将 HTML 在浏览器中正确显示，并且可以作为静态原型，让开发团队能更容易地协作。Thymeleaf 能够处理 HTML，XML，JavaScript，CSS 甚至纯文本。

长期以来,jsp 在视图领域有非常重要的地位,随着时间的变迁,出现了一位新的挑战者:Thymeleaf,Thymeleaf 是原生的,不依赖于标签库.它能够在接受原始 HTML 的地方进行编辑和渲染.因为它没有与 Servlet 规范耦合,因此 Thymeleaf 模板能进入 jsp 所无法涉足的领域。

3.2 Thymeleaf 基本使用

3.2.1 创建项目



3.2.2 修改 POM 文件，添加 Thymeleaf 启动器依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    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>2.2.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.bjsxt</groupId>
  <artifactId>springbootthymeleaf</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>springbootthymeleaf</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>
```

```

<!--添加 Thymeleaf 启动器依赖-->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <groupId>org.junit.vintage</groupId>
      <artifactId>junit-vintage-engine</artifactId>
    </exclusion>
  </exclusions>
</dependency>
</dependencies>

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

```

3.2.3 创建 Controller

```

/**
 * 页面跳转 Controller
 */
@Controller

```



```
public class PageController {

    /**
     * 页面跳转方法
     */
    @GetMapping("/show")
    public String showPage(Model model){
        model.addAttribute("msg", "Hello Thymeleaf");
        return "index";
    }
}
```

3.2.4 创建视图

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>

<title>北京尚学堂首页</title>
</head>
<body>

<span th:text="北京尚学堂"></span>
<hr/>
<span th:text="${msg}"></span>
</body>
</html>
```

3.3 Thymeleaf 语法讲解

命名空间: xmlns:th="http://www.thymeleaf.org"

3.3.1 字符串与变量输出操作

3.3.1.1 th:text

th:text

在页面中输出值

3.3.1.2 th:value

th:value

可以将一个值放入到 input 标签的 value 中

3.3.2 字符串操作

Thymeleaf 提供了一些内置对象，内置对象可直接在模板中使用。这些对象是以#引用的。

3.3.2.1 使用内置对象的语法

- 1) 引用内置对象需要使用#
- 2) 大部分内置对象的名称都以 s 结尾。如：strings、numbers、dates

`${#strings.isEmpty(key)}`

判断字符串是否为空，如果为空返回 true，否则返回 false

`${#strings.contains(msg, 'T')}`

判断字符串是否包含指定的子串，如果包含返回 true，否则返回 false

`${#strings.startsWith(msg, 'a')}`

判断当前字符串是否以子串开头，如果是返回 true，否则返回 false

`${#strings.endsWith(msg, 'a')}`

判断当前字符串是否以子串结尾，如果是返回 true，否则返回 false

`${#strings.Length(msg)}`

返回字符串的长度

`${#strings.indexOf(msg, 'h')}`

查找子串的位置，并返回该子串的下标，如果没找到则返回-1

`${#strings.substring(msg, 2)}`

<code>\${#strings.substring(msg,2,5)}</code>
截取子串，用户与 jdk String 类下 SubString 方法相同
<code>\${#strings.toUpperCase(msg)}</code> <code>\${#strings.toLowerCase(msg)}</code>
字符串转大小写。

3.3.3 日期格式化处理

<code>\${#dates.format(key)}</code>
格式化日期，默认的以浏览器默认语言为格式化标准
<code>\${#dates.format(key, 'yyyy/MM/dd')}</code>
按照自定义的格式做日期转换
<code>\${#dates.year(key)}</code> <code>\${#dates.month(key)}</code> <code>\${#dates.day(key)}</code>
Year: 取年 Month: 取月 Day: 取日

3.3.4 条件判断

3.3.4.1 th:if

th:if
条件判断

3.3.4.2 th:switch / th:case

th:switch / th:case

th:switch / th:case 与 Java 中的 switch 语句等效，有条件地显示匹配的内容。如果有多个匹配结果只选择第一个显示。

th:case="" 表示 Java 中 switch 的 default，即没有 case 的值为 true 时则显示 th:case="" 的内容。

3.3.5 迭代遍历

3.3.5.1 th:each

th:each

迭代器，用于循环迭代集合

3.3.5.2 th:each 状态变量

- 1) index: 当前迭代器的索引 从 0 开始
- 2) count: 当前迭代对象的计数 从 1 开始
- 3) size: 被迭代对象的长度
- 4) odd/even: 布尔值，当前循环是否是偶数/奇数 从 0 开始
- 5) first: 布尔值，当前循环的是否是第一条，如果是返回 true 否则返回 false
- 6) last: 布尔值，当前循环的是否是最后一条，如果是则返回 true 否则返回 false

3.3.6 th:each 迭代 Map

```
<table border="1" width="50%">
  <tr>
    <th>ID</th>
    <th>Name</th>
    <th>Age</th>
    <th>Key</th>
  </tr>
  <tr th:each="m : ${map}">
    <td th:text="${m.value.id}"></td>
    <td th:text="${m.value.name}"></td>
    <td th:text="${m.value.age}"></td>
    <td th:text="${m.key}"></td>
  </tr>
</table>
```

3.3.7 操作域对象

3.3.7.1 HttpServletRequest

<code>request.setAttribute("req", "HttpServletRequest");</code>
<code></code>
<code></code>

3.3.7.2 HttpSession

<code>request.getSession().setAttribute("sess", "HttpSession");</code>
<code>
</code>
<code>
</code>

3.3.7.3 ServletContext

<code>request.getSession().getServletContext().setAttribute("app", "Application");</code>
<code></code>
<code></code>

3.3.8 URL 表达式

3.3.8.1 语法

在 Thymeleaf 中 URL 表达式的语法格式为@{}

3.3.8.2 URL 类型

3.3.8.2.1 绝对路径

<code><a th:href="@{http://www.baidu.com}">绝对路径</code>
--

3.3.8.2.2 相对路径

相对于当前项目的根
<code><a th:href="@{/show}">相对路径</code>
相对于服务器路径的根
<code><a th:href="@{/~/project2/resourcename}">相对于服务器的根</code>

3.3.8.3 在 URL 中传递参数

3.3.8.3.1 在普通格式的 URL 中传递参数

<code><a th:href="@{/show?id=1&name=zhangsan}">普通 URL 格式传参</code>
<code><a th:href="@{/show(id=1,name=zhangsan)}">普通 URL 格式传参</code>
<code><a th:href="@{'/' + show?id=' + \${id} + '&name=' + \${name} + '}">普通 URL 格式传参</code>
<code><a th:href="@{/show(id=\${id},name=\${name})}">普通 URL 格式传参</code>

3.3.8.3.2 在 restful 格式的 URL 中传递参数

<code><a th:href="@{/show/{id}(id=1)}">restful 格式传参</code>
<code><a th:href="@{/show/{id}/{name}(id=1,name=admin)}">restful 格式传参</code>
<code><a th:href="@{/show/{id}(id=1,name=admin)}">restful 格式传参</code>
<code><a th:href="@{/show/{id}(id=\${id},name=\${name})}">restful 格式传参</code>

3.3.9 在配置文件中配置 Thymeleaf

```
spring.thymeleaf.prefix=classpath:/templates/suibian/
spring.thymeleaf.suffix=.html

spring.thymeleaf.mode=HTML #配置视图模板类型，如果视图模板使用的是html5需要配置

spring.thymeleaf.encoding=utf-8

spring.thymeleaf.servlet.content-type=text/html #响应类型

#配置页面缓存
spring.thymeleaf.cache=false
```

九、 Spring Boot 整合持久层技术

1 整合 JDBC

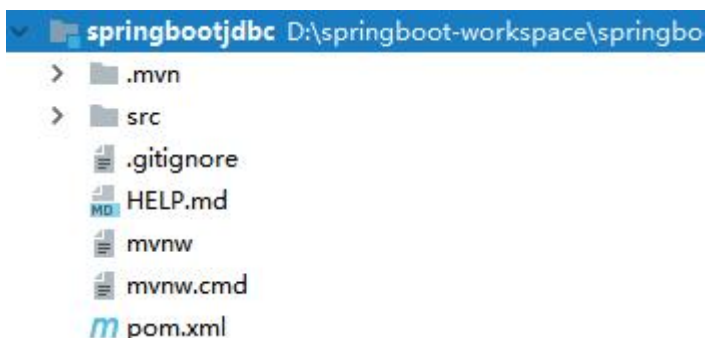
1.1 搭建项目环境

1.1.1 创建表

1.1.1.1 建表语句

```
CREATE TABLE `users` (
  `userid` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(30) DEFAULT NULL,
  `usersex` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

1.1.2 创建项目



1.1.3 修改 POM 文件，添加相关依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    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>2.2.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
```

```

</parent>
<groupId>com.bjsxt</groupId>
<artifactId>springbootjdbc</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springbootjdbc</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>

  <!--Thymeleaf 启动器坐标-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>

  <!--JDBC 启动器坐标-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
  </dependency>

  <!--数据库驱动坐标-->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.38</version>
  </dependency>

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

```



```

        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>

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

1.2 配置数据源

1.2.1 通过自定义配置文件方式配置数据源信息

1.2.1.1 通过@PropertySource 注解读取配置文件

1.2.1.1.1 添加 Druid 数据源依赖

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    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>2.2.0.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    
```

```

<groupId>com.bjsxt</groupId>
<artifactId>springbootjdbc</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springbootjdbc</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>

  <!--Thymeleaf 启动器坐标-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>

  <!--JDBC 启动器坐标-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
  </dependency>

  <!--数据库驱动坐标-->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.38</version>
  </dependency>

  <!--Druid 数据源依赖-->
  <dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.12</version>
  </dependency>

```

```

</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <groupId>org.junit.vintage</groupId>
      <artifactId>junit-vintage-engine</artifactId>
    </exclusion>
  </exclusions>
</dependency>
</dependencies>

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

</project>

```

1.2.1.1.2 创建 Properties 文件

```

jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/test?useUnicode=true
&characterEncoding=utf-8&useSSL=false
jdbc.username=root
jdbc.password=root

```

1.2.1.1.3 创建配置类

```

/**
 * 数据源的 JDBC 配置类
 */

```

```
@Configuration
@PropertySource("classpath:/jdbc.properties") //加载指定的
Properties 配置文件
public class JdbcConfiguration {

    @Value("${jdbc.driverClassName}")
    private String driverClassName;
    @Value("${jdbc.url}")
    private String url;
    @Value("${jdbc.username}")
    private String username;
    @Value("${jdbc.password}")
    private String password;
    /**
     * 实例化 Druid
     */
    @Bean
    public DataSource getDataSource() {
        DruidDataSource source = new DruidDataSource();
        source.setPassword(this.password);
        source.setUsername(this.username);
        source.setUrl(this.url);
        source.setDriverClassName(this.driverClassName);
        return source;
    }
}
```

1.2.1.2 通过@ConfigurationProperties 注解读取配置信息

1.2.1.2.1 创建配置信息实体类

```
/**
 * JDBC 配置信息属性类
 */
@ConfigurationProperties(prefix = "jdbc") //是SpringBoot 的
注解不能读取其他配置文件，只能读取SpringBoot 的application 配置文
```

件

```
public class JdbcProperties {

    private String driverClassName;
    private String url;
    private String username;
    private String password;

    public String getDriverClassName() {
        return driverClassName;
    }

    public void setDriverClassName(String driverClassName) {
        this.driverClassName = driverClassName;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

1.2.1.2.2 修改配置类

```
/**
 * 数据源的 JDBC 配置类
 */
@Configuration
//@PropertySource("classpath:/jdbc.properties") //加载指定
//的 Properties 配置文件

@EnableConfigurationProperties(JdbcProperties.class) //指定
//加载哪个配置信息属性类
public class JdbcConfiguration {

    // @Autowired
    // private JdbcProperties jdbcProperties;

    public JdbcConfiguration(JdbcProperties
    jdbcProperties) {
        this.jdbcProperties = jdbcProperties;
    }

    /**
     * 实例化 Druid
     */
    @Bean
    public DataSource getDataSource(JdbcProperties
    jdbcProperties) {
        DruidDataSource source = new DruidDataSource();
        source.setPassword(jdbcProperties.getPassword());
        source.setUsername(jdbcProperties.getUsername());
        source.setUrl(jdbcProperties.getUrl());

        source.setDriverClassName(jdbcProperties.getDriverClassNa
        me());
        return source;
    }
}
```

1.2.1.2.3 @ConfigurationProperties 注解的优雅使用方式

```
/**
 * 数据源的 JDBC 配置类
 */
@Configuration
//@PropertySource("classpath:/jdbc.properties") //加载指定
//的 Properties 配置文件

//@EnableConfigurationProperties(JdbcProperties.class) //指
//定加载哪个配置信息属性类
public class JdbcConfiguration {

    // @Autowired
    // private JdbcProperties jdbcProperties;

    public JdbcConfiguration(JdbcProperties
jdbcProperties) {
        this.jdbcProperties = jdbcProperties;
    }

    /**
     * 实例化 Druid
     */
    @Bean
    @ConfigurationProperties(prefix = "jdbc")
    public DataSource getDataSource() {
        DruidDataSource source = new DruidDataSource();
        return source;
    }
}
```

1.2.2 通过 Spring Boot 配置文件配置数据源

在 Spring Boot 1.x 版本中的 spring-boot-starter-jdbc 启动器中默认使用的是 org.apache.tomcat.jdbc.pool.DataSource 作为数据源

在 Spring Boot 2.x 版本中的 spring-boot-starter-jdbc 启动器中默认使用的是 com.zaxxer.hikariDataSource 作为数据源

1.2.2.1 使用 Spring Boot 默认的 HikariDataSource 数据源

```
spring.datasource.url=jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf-8&useSSL=false
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=root
```

1.2.2.2 使用第三方的 Druid 数据源

```
spring.datasource.url=jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf-8&useSSL=false
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
```

1.3 添加用户

1.3.1 创建 POJO

```
public class Users {
    private Integer userid;
    private String username;
    private String usersex;

    public Integer getUserid() {
        return userid;
    }

    public void setUserid(Integer userid) {
        this.userid = userid;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
```



```
        this.username = username;
    }

    public String getUsersex() {
        return usersex;
    }

    public void setUsersex(String usersex) {
        this.usersex = usersex;
    }
}
```

1.3.2 创建页面

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.thymeleaf.org">
<head>

    <title>北京尚学堂首页</title>

</head>
<body>

    <form th:action="@{/user/addUser}" method="post">
        <input type="text" name="username"><br/>
        <input type="text" name="usersex"><br/>
        <input type="submit" value="OK"/>
    </form>

</body>
</html>
```

1.3.3 创建 Controller

1.3.3.1 PageController

```
/**
 * 页面跳转 Controller
```

```

*/
@Controller
public class PageController {

    /**
     * 页面跳转方法
     */
    @RequestMapping("/{page}")
    public String showPage(@PathVariable String page){
        return page;
    }
}

```

1.3.3.2 UsersController

```

@Controller
@RequestMapping("/user")
public class UsersController {

    @Autowired
    private UserService userService;

    /**
     * 添加用户
     * @return
     */
    @PostMapping("/addUser")
    public String addUser(Users users){
        try{
            this.userService.addUser(users);
        } catch (Exception e) {
            e.printStackTrace();
            return "error";
        }
        return "redirect:/ok";
    }
}

```

1.3.4创建 Service

```
/**
 * 用户管理业务层
 */
@Service
public class UsersServiceImpl implements UsersService {

    @Autowired
    private UsersDao usersDao;

    /**
     * 添加用户
     * @param users
     */
    @Override
    @Transactional
    public void addUser(Users users) {
        this.usersDao.insertUsers(users);
    }
}
```

1.3.5创建 Dao

```
/**
 * 用户管理持久层
 */
@Repository
public class UsersDaoImpl implements UsersDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    /**
     * 添加用户
     * @param users
     */
    @Override
```

```
public void insertUsers(Users users) {  
    String sql = "insert into users(username,usersex)  
values(?,?)";  
  
    this.jdbcTemplate.update(sql,users.getUsername(),users.ge  
tUsersex());  
}  
}
```

1.3.6 解决 favicon.ico 解析问题

```
<link rel="shortcut icon" href="../resources/favicon.ico" th:href="@{/static/favicon.ico}"/>
```

1.4 查询用户

1.4.1 修改 Controller

```
/**  
 * 查询全部用户  
 */  
@GetMapping("/findUserAll")  
public String findUserAll(Model model){  
    List<Users> list = null;  
    try{  
        list = this.userService.findUsersAll();  
        model.addAttribute("list",list);  
    }catch(Exception e){  
        e.printStackTrace();  
        return "error";  
    }  
    return "showUsers";  
}
```

1.4.2 修改业务层

```
/**
```

```

* 查询全部用户

* @return
*/
@Override
public List<Users> findUsersAll() {
    return this.usersDao.selectUsersAll();
}

```

1.4.3修改持久层

```

/**
 * 查询全部用户
 * @return
 */
@Override
public List<Users> selectUsersAll() {
    String sql = "select * from users";

    return this.jdbcTemplate.query(sql, new
    RowMapper<Users>() {
        /**
         * 结果集的映射
         * @param resultSet
         * @param i
         * @return
         * @throws SQLException
         */
        @Override
        public Users mapRow(ResultSet resultSet, int i) throws
        SQLException {
            Users users = new Users();
            users.setUserid(resultSet.getInt("userid"));

            users.setUsername(resultSet.getString("username"));

            users.setUsersex(resultSet.getString("usersex"));

            return users;
        }
    }
}

```

```
});  
}
```

1.4.4 创建页面显示查询结果

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN"  
    "http://www.w3.org/TR/html4/loose.dtd">  
<html xmlns:th="http://www.thymeleaf.org">  
<link rel="shortcut icon" href="../../resources/favicon.ico"  
th:href="@{/static/favicon.ico}"/>  
<head>  
    <title>北京尚学堂首页</title>  
</head>  
<body>  
    <table border="1" align="center">  
        <tr>  
            <th>用户 ID</th>  
            <th>用户姓名</th>  
            <th>用户性别</th>  
            <th>操作</th>  
        </tr>  
        <tr th:each="u : ${list}">  
            <td th:text="${u.userid}"></td>  
            <td th:text="${u.username}"></td>  
            <td th:text="${u.usersex}"></td>  
            <td>  
                <a  
th:href="@{/user/preUpdateUser(id=${u.userid})}">修改</a>  
                <a  
th:href="@{/user/deleteUser(id=${u.userid})}">删除</a>  
            </td>  
        </tr>
```

```
</table>
</body>
</html>
```

1.5 更新用户

1.5.1 预更新查询

1.5.1.1 修改 Controller

```
/**
 * 预更新用户的查询
 */
@GetMapping("/preUpdateUser")
public String preUpdateUser(Integer id, Model model) {
    try {
        Users user = this.userService.findUserById(id);
        model.addAttribute("user", user);
    } catch (Exception e) {
        e.printStackTrace();
        return "error";
    }
    return "updateUser";
}
```

1.5.1.2 修改业务层

```
/**
 * 预更新查询
 * @param id
 * @return
 */
@Override
public Users findUserById(Integer id) {
    return this.usersDao.selectUserById(id);
}
```

1.5.1.3 修改持久层

```
/**
 * 预更新用户查询
 * @param id
 * @return
 */
@Override
public Users selectUserById(Integer id) {
    Users user = new Users();
    String sql = "select * from users where userid = ?";
    Object[] arr = new Object[]{id};
    this.jdbcTemplate.query(sql, arr, new
    RowCallbackHandler() {
        @Override
        public void processRow(ResultSet resultSet) throws
        SQLException {

            user.setUsername(resultSet.getString("username"));

            user.setUsersex(resultSet.getString("usersex"));
            user.setUserid(resultSet.getInt("userid"));

        }
    });
    return user;
}
```

1.5.1.4 创建用户更新页面

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.thymeleaf.org">
<link rel="shortcut icon" href="../resources/favicon.ico"
th:href="@{/static/favicon.ico}"/>
<head>
    <title>北京尚学堂首页</title>
</head>
```



```
<body>
  <form th:action="@{/user/updateUser}" method="post">
    <input type="hidden" name="userid"
th:value="${user.userid}"/>
    <input type="text" name="username"
th:value="${user.username}"/><br/>
    <input type="text" name="usersex"
th:value="${user.usersex}"/><br/>
    <input type="submit" value="OK"/>
  </form>
</body>
</html>
```

1.5.2 更新用户操作

1.5.2.1 修改 Controller

```
/**
 * 更新用户
 */
@PostMapping("/updateUser")
public String updateUser(Users users){
    try{
        this.userService.modifyUser(users);
    } catch (Exception e) {
        e.printStackTrace();
        return "error";
    }
    return "redirect:/ok";
}
```

1.5.2.2 修改业务层

```
/**
 * 更新用户
 * @param users
 */
@Override
```

```
@Transactional
public void modifyUser(Users users) {
    this.usersDao.updateUsers(users);
}
```

1.5.2.3 修改持久层

```
/**
 * 更新用户
 * @param users
 */
@Override
public void updateUsers(Users users) {
    String sql = "update users set username = ?,usersex=? where
userid = ?";

    this.jdbcTemplate.update(sql,users.getUsername(),users.ge
tUsersex(),users.getUserid());
}
```

1.6 删除用户

1.6.1 修改 Controller

```
/**
 * 删除用户
 */
@GetMapping("/deleteUser")
public String deleteUser(Integer id){
    try{
        this.userService.dropUser(id);
    }catch(Exception e){
        e.printStackTrace();
        return "error";
    }
    return "redirect:/ok";
}
```

1.6.2修改业务层

```
/**
 * 删除用户
 * @param id
 */
@Override
@Transactional
public void dropUser(Integer id) {
    this.usersDao.deleteUserById(id);
}
```

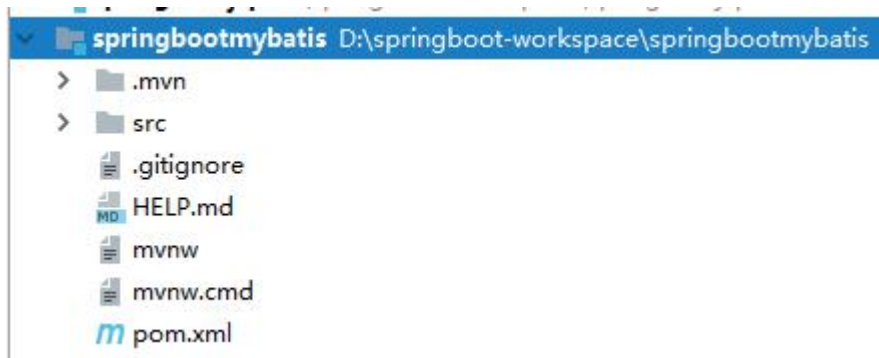
1.6.3修改持久层

```
/**
 * 删除用户
 * @param id
 */
@Override
public void deleteUserById(Integer id) {
    String sql = "delete from users where userid= ?";
    this.jdbcTemplate.update(sql, id);
}
```

2 整合 MyBatis

2.1 搭建项目环境

2.1.1 创建项目



2.1.2 修改 POM 文件，添加相关依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    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>2.2.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.bjsxt</groupId>
  <artifactId>springbootmybatis</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>springbootmybatis</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-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<!--Mybatis 启动器-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.1</version>
</dependency>

<!--数据库驱动坐标-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.38</version>
</dependency>

<!--Druid 数据源依赖-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.12</version>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
    </exclusions>
</dependency>

```

```
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>
```

2.1.3 配置数据源

```
spring:
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url:
jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf-8&useSSL=false
    username: root
    password: root
    type: com.alibaba.druid.pool.DruidDataSource
```

2.2 配置 Maven 的 generator 插件

2.2.1 添加 generator 插件坐标

```
<!--配置 Generator 插件-->
<plugin>
  <groupId>org.mybatis.generator</groupId>

  <artifactId>mybatis-generator-maven-plugin</artifactId>
  <version>1.3.5</version>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
```

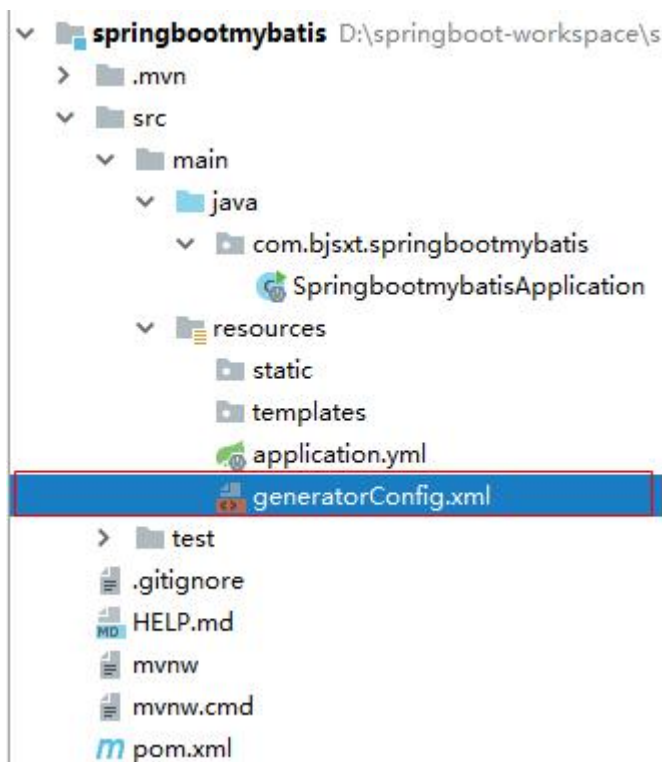
```

<artifactId>mysql-connector-java</artifactId>
<version>5.1.38</version>
</dependency>
</dependencies>

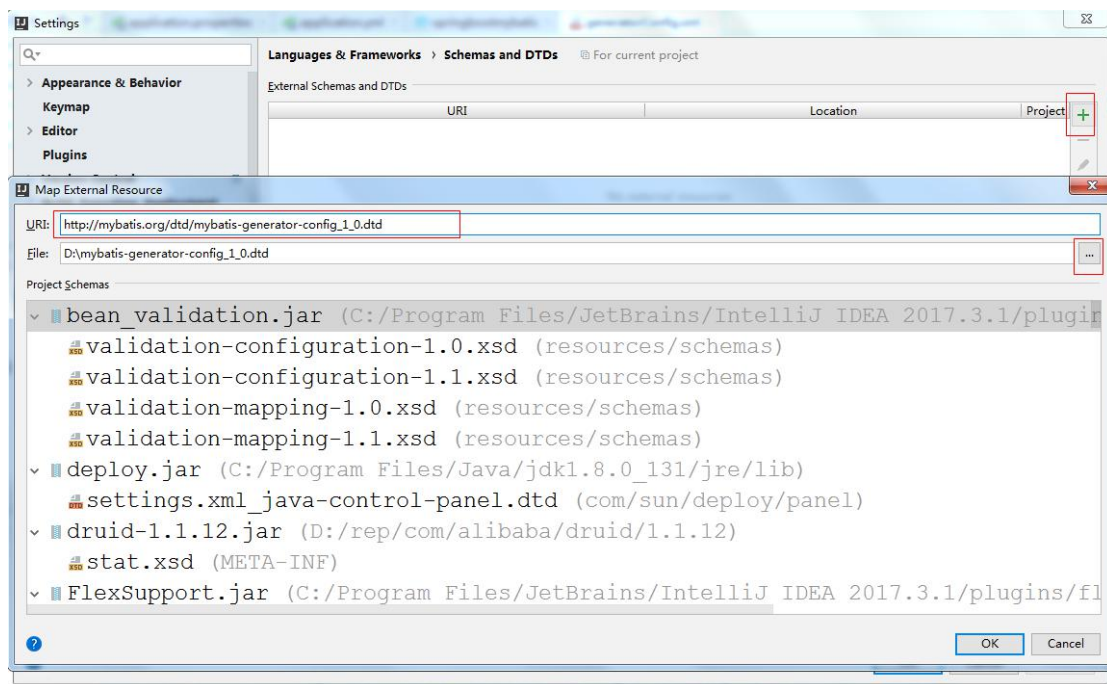
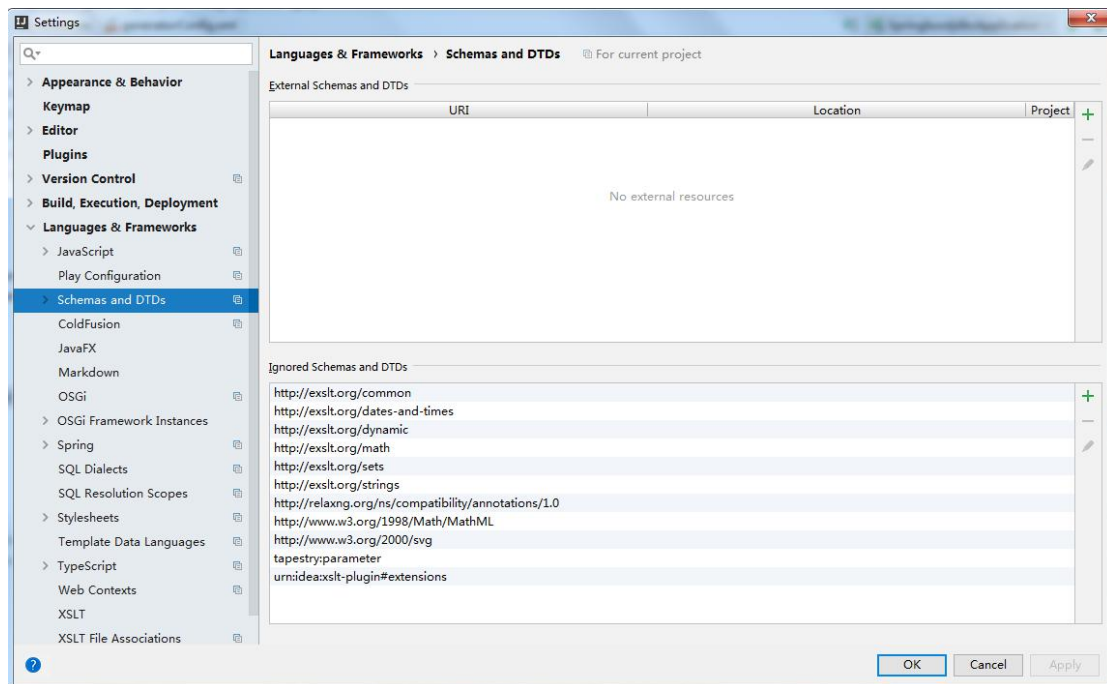
<!--指定配置文件的路径-->
<configuration>
<configurationFile>${project.basedir}/src/main/resources/
generatorConfig.xml</configurationFile>
<verbose>true</verbose>
<overwrite>true</overwrite>
</configuration>
</plugin>

```

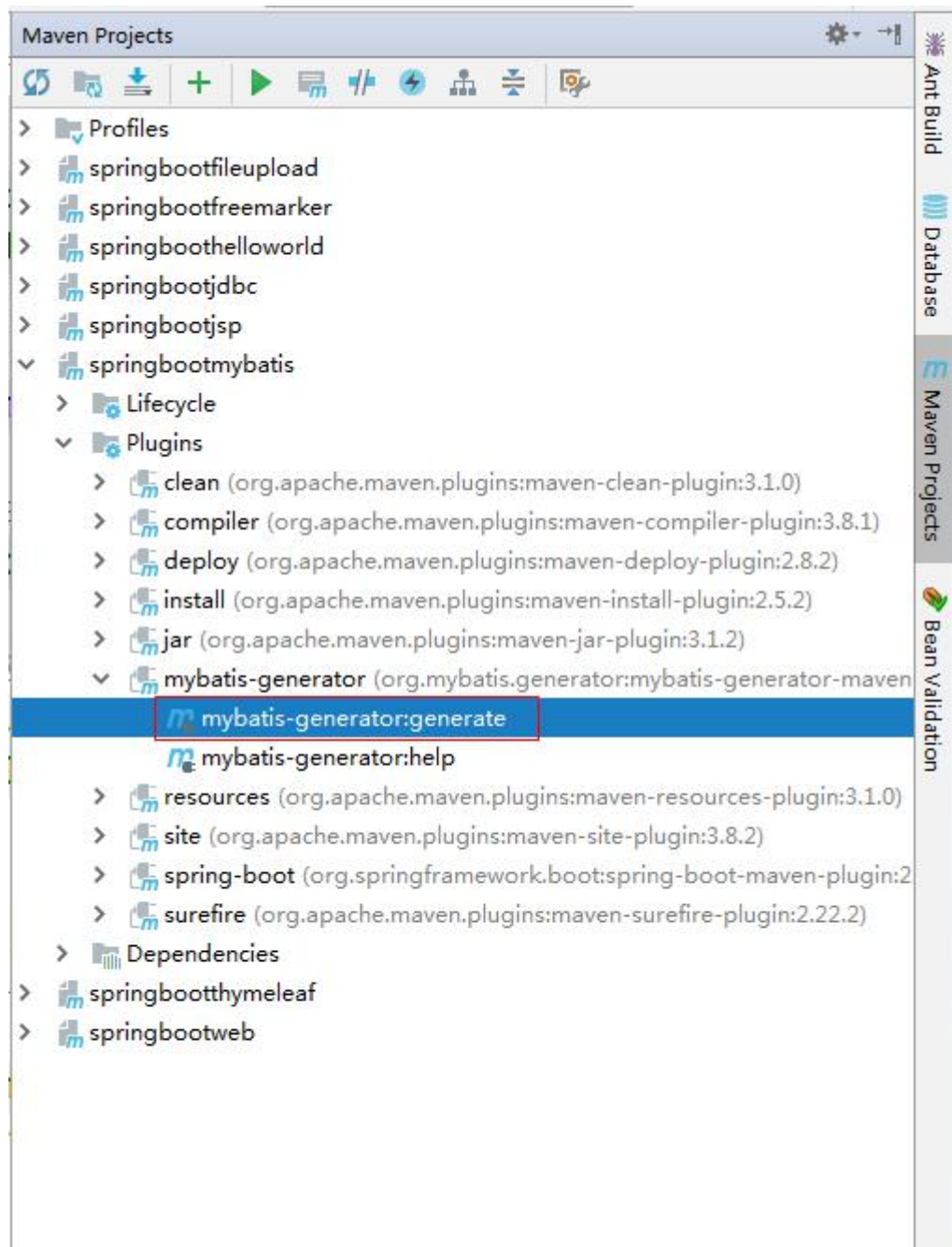
2.2.2 添加 generator 配置文件

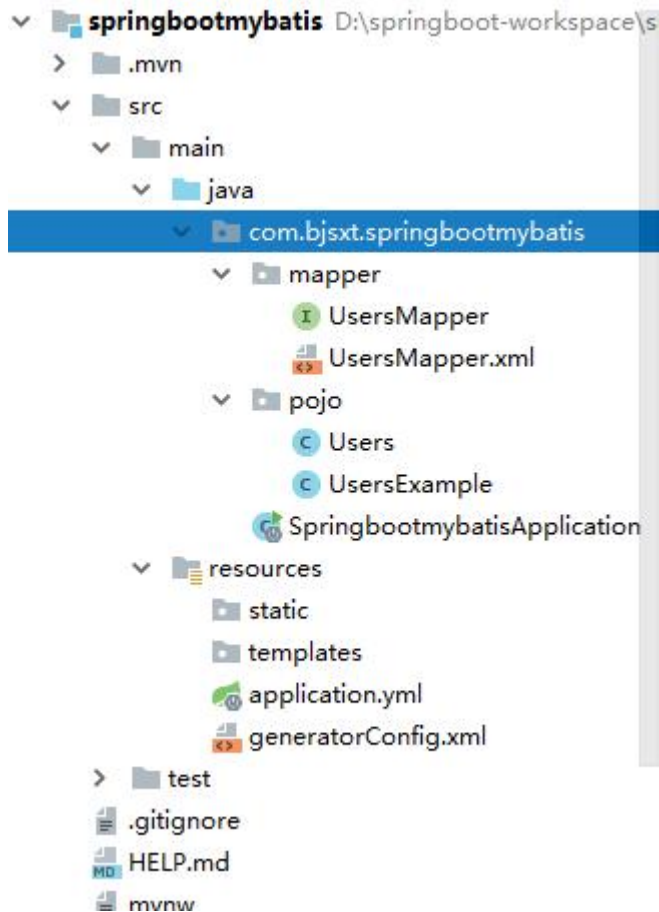


2.2.3 添加 generator 配置文件的 DTD 文件



2.2.4 运行 generator 插件生成代码





2.3 配置资源拷贝插件

2.3.1 添加资源拷贝插件坐标

```
<!--配置资源拷贝插件-->
<resources>
  <resource>
    <directory>src/main/java</directory>
    <includes>
      <include>**/*.xml</include>
    </includes>
  </resource>

  <resource>
    <directory>src/main/resources</directory>
    <includes>
      <include>**/*.yml</include>
      <include>**/*.properties</include>
    </includes>
  </resource>
</resources>
```

```
</resource>  
</resources>
```

2.3.2 修改启动类添加 @MapperScan 注解

```
@SpringBootApplication  
@MapperScan("com.bjsxt.springbootmybatis.mapper") //指定扫描  
接口与映射配置文件的包名  
public class SpringbootmybatisApplication {  
  
    public static void main(String[] args) {  
  
        SpringApplication.run(SpringbootmybatisApplication.class,  
            args);  
    }  
  
}
```

2.4 MyBatis 的其他配置项

```
mybatis:  
  
#扫描 classpath 中 mapper 目录下的映射配置文件，针对于映射配置文件放  
到了 resources 目录下  
  
mapper-locations: classpath:/mapper/*.xml  
  
#定义包别名，使用 pojo 时可以直接使用 pojo 的类型名称不用加包名  
  
type-aliases-package: com.bjsxt.springbootmybatis.pojo
```

2.5 添加用户功能

2.5.1 创建页面

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN"
```

```
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.thymeleaf.org">
<link rel="shortcut icon" href="../../resources/favicon.ico"
th:href="@{/static/favicon.ico}"/>
<head>

    <title>北京尚学堂首页</title>

</head>
<body>

    <form th:action="@{/user/addUser}" method="post">
        <input type="text" name="username"><br/>
        <input type="text" name="usersex"><br/>
        <input type="submit" value="OK"/>
    </form>
</body>
</html>
```

2.5.2 创建 Controller

2.5.2.1 PageController

```
/**
 * 页面跳转 Controller
 */
@Controller
public class PageController {

    /**
     * 页面跳转方法
     */
    @RequestMapping("/{page}")
    public String showPage(@PathVariable String page){
        return page;
    }
}
```

2.5.2.2 UsersController

```
/**
 * 用户管理 Controller
 */
@Controller
@RequestMapping("/user")
public class UsersController {

    @Autowired
    private UsersService userService;
    /**
     * 添加用户
     */
    @PostMapping("/addUser")
    public String addUsers(Users users){
        try{
            this.userService.addUsers(users);
        } catch (Exception e) {
            e.printStackTrace();
            return "error";
        }
        return "redirect:/ok";
    }
}
```

2.5.3 创建 Service

```
/**
 * 用户管理业务层
 */
@Service
public class UsersServiceImpl implements UsersService{
    @Autowired
    private UsersMapper usersMapper;

    @Override
    @Transactional
    public void addUsers(Users users) {
```

```
        this.usersMapper.insert(users);  
    }  
}
```

2.6 查询用户功能

2.6.1 修改 UsersController

```
/**  
 * 查询全部用户  
 */  
@GetMapping("/findUserAll")  
public String findUserAll(Model model){  
    try{  
        List<Users> list = this.userService.findUserAll();  
        model.addAttribute("list", list);  
    } catch (Exception e) {  
        e.printStackTrace();  
        return "error";  
    }  
    return "showUsers";  
}
```

2.6.2 修改业务层

```
/**  
 * 查询全部用户  
 * @return  
 */  
@Override  
public List<Users> findUserAll() {  
    UsersExample example = new UsersExample();  
    return this.usersMapper.selectByExample(example);  
}
```

2.6.3 创建页面显示查询结果

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN"  
    "http://www.w3.org/TR/html4/loose.dtd">  
<html xmlns:th="http://www.thymeleaf.org">  
<link rel="shortcut icon" href="../resources/favicon.ico"  
th:href="@{/static/favicon.ico}"/>  
<head>  
    <title>北京尚学堂首页</title>  
</head>  
<body>  
    <table border="1" align="center">  
        <tr>  
            <th>用户 ID</th>  
            <th>用户姓名</th>  
            <th>用户性别</th>  
            <th>操作</th>  
        </tr>  
        <tr th:each="u : ${list}">  
            <td th:text="${u.userid}"></td>  
            <td th:text="${u.username}"></td>  
            <td th:text="${u.usersex}"></td>  
            <td>  
                <a  
th:href="@{/user/preUpdateUser(id=${u.userid})}">修改</a>  
                <a  
th:href="@{/user/deleteUser(id=${u.userid})}">删除</a>  
            </td>  
        </tr>  
    </table>  
</body>  
</html>
```

2.7 更新用户功能

2.7.1 预更新用户查询

2.7.1.1 修改 UsersController

```
/**
 * 预更新用户查询
 */
@GetMapping("/preUpdateUser")
public String preUpdateUser(Integer id, Model model) {
    try {
        Users user = this.userService.preUpdateUsers(id);
        model.addAttribute("user", user);
    } catch (Exception e) {
        e.printStackTrace();
        return "error";
    }
    return "updateUser";
}
```

2.7.1.2 修改业务层

```
/**
 * 预更新查询
 * @param id
 * @return
 */
@Override
public Users preUpdateUsers(Integer id) {
    return this.usersMapper.selectByPrimaryKey(id);
}
```

2.7.1.3 创建更新用户页面

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```



```
<html xmlns:th="http://www.thymeleaf.org">
<link rel="shortcut icon" href="../../resources/favicon.ico"
th:href="@{/static/favicon.ico}"/>
<head>

    <title>北京尚学堂首页</title>

</head>
<body>
    <form th:action="@{/user/updateUser}" method="post">
        <input type="hidden" name="userid"
th:value="${user.userid}"/>
        <input type="text" name="username"
th:value="${user.username}"/><br/>
        <input type="text" name="usersex"
th:value="${user.usersex}"/><br/>
        <input type="submit" value="OK"/>
    </form>
</body>
</html>
```

2.7.2 更新用户操作

2.7.2.1 修改 UsersController

```
/**
 * 更新用户
 */
@PostMapping("/updateUser")
public String updateUser(Users users) {
    try {
        this.userService.modifyUsers(users);
    } catch (Exception e) {
        e.printStackTrace();
        return "error";
    }
    return "redirect:/ok";
}
```

2.7.2.2 修改业务层

```
/**
 * 更新用户
 * @param users
 */
@Override
@Transactional
public void modifyUsers(Users users) {
    this.usersMapper.updateByPrimaryKey(users);
}
```

2.8 删除用户功能

2.8.1 修改 UsersController

```
/**
 * 删除用户
 */
@GetMapping("/deleteUser")
public String deleteUser(Integer id){
    try{
        this.userService.dropUsersById(id);
    } catch (Exception e) {
        e.printStackTrace();
        return "error";
    }
    return "redirect:/ok";
}
```

2.8.2 修改业务层

```
/**
 * 删除用户
 * @param id
 */
```

```
@Override
@Transactional
public void dropUsersById(Integer id) {
    this.usersMapper.deleteByPrimaryKey(id);
}
```

十、SpringBoot 中异常处理与单元测试

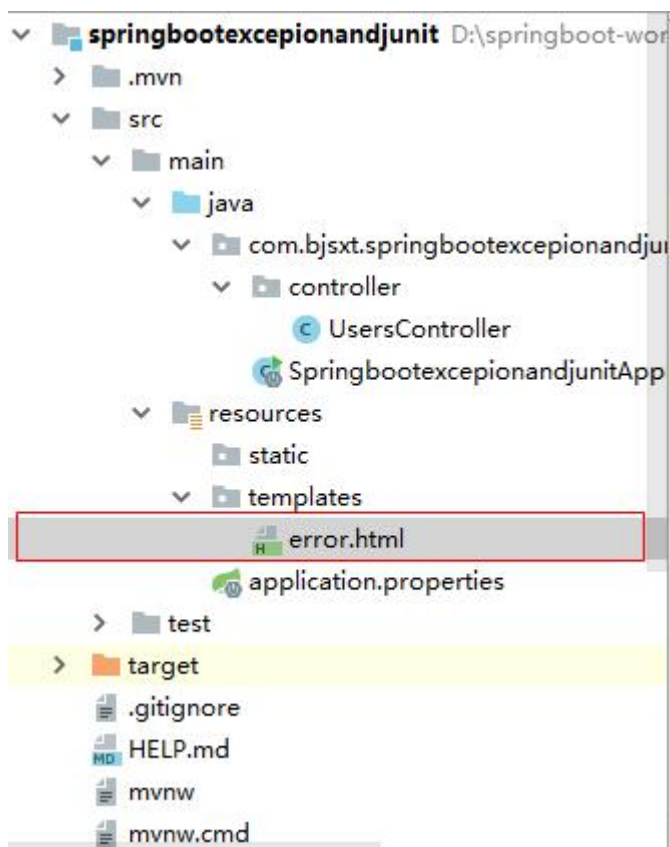
1 异常处理

SpringBoot 中对于异常处理提供了五种处理方式

1.1 自定义错误页面

SpringBoot 默认的处理异常的机制: SpringBoot 默认已经提供了一套处理异常的机制。一旦程序中出现了异常 SpringBoot 会向 /error 的 url 发送请求。在 SpringBoot 中提供了一个名为 BasicErrorController 来处理 /error 请求, 然后跳转到默认显示异常的页面来展示异常信息。

如果我们需要将所有的异常同一跳转到自定义的错误页面, 需要再 src/main/resources/templates 目录下创建 error.html 页面。注意: 页面名称必须叫 error



1.2 通过 @ExceptionHandler 注解处理异常

1.2.1 修改 Controller

```
@Controller
public class UsersController {

    @RequestMapping("showInfo")
    public String showInfo() {
        String str = null;
        str.length();
        return "ok";
    }

    @ExceptionHandler(value =
{java.lang.NullPointerException.class} )
    public ModelAndView nullpointExcepitonHandler(Exception
e) {
        ModelAndView mv = new ModelAndView();
        mv.addObject("err",e.toString());
        mv.setViewName("error1");
        return mv;
    }
}
```

1.2.2 创建页面

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>北京尚学堂首页</title>
</head>
<body>
    出错了。。。
    <span th:text="${err}" />
</body>
```

</html>

1.3 通过@ControllerAdvice 与@ExceptionHandler 注解处理异常

1.3.1 创建全局异常处理类

```
/**
 * 全局异常处理类
 */
@ControllerAdvice
public class GlobalException {

    @ExceptionHandler(value =
    {java.lang.NullPointerException.class} )
    public ModelAndView nullpointExcepitonHandler(Exception
    e) {
        ModelAndView mv = new ModelAndView();
        mv.addObject("err",e.toString());
        mv.setViewName("error1");
        return mv;
    }

    @ExceptionHandler(value =
    {java.lang.ArithmeticException.class} )
    public ModelAndView
    arithmeticExceptionHandler(Exception e){
        ModelAndView mv = new ModelAndView();
        mv.addObject("err",e.toString());
        mv.setViewName("error2");
        return mv;
    }
}
```

1.4 通过 SimpleMappingExceptionResolver 对象处理异常

1.4.1 创建全局异常处理类

```
/**
 * 全局异常
 * SimpleMappingExceptionResolver
 */
@Configuration
public class GlobalException2 {

    /**
     * 此方法返回值必须是 SimpleMappingExceptionResolver 对象
     * @return
     */
    @Bean
    public SimpleMappingExceptionResolver
    getSimpleMappingExceptionResolver() {
        SimpleMappingExceptionResolver resolver = new
        SimpleMappingExceptionResolver();
        Properties properties = new Properties();
        /**
         * 参数一：异常类型，并且是全名
         * 参数二：视图名称
         */
        properties.put("java.lang.NullPointerException", "error3");
        properties.put("java.lang.ArithmeticException", "error4");

        resolver.setExceptionMappings(properties);
        return resolver;
    }
}
```

1.5 通过自定义 HandlerExceptionResolver 对象处理异常

1.5.1 创建全局异常处理类

```
/**
```

```
* 自定义 HandlerExceptionResolver 对象处理异常
* 必须要实现 HandlerExceptionResolver
*/
@Configuration
public class GlobalException3 implements
HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest
httpServletRequest, HttpServletResponse
httpServletResponse, Object handler, Exception e) {
        ModelAndView mv = new ModelAndView();

        //判断不同异常类型, 做不同视图的跳转

        if(e instanceof NullPointerException){
            mv.setViewName("error5");
        }
        if(e instanceof ArithmeticException){
            mv.setViewName("error6");
        }
        mv.addObject("error",e.toString());
        return mv;
    }
}
```

2 Spring Boot 整合 Junit 单元测试

SpringBoot2.x 使用 Junit5 作为测试平台

2.1 修改 POM 文件添加 Test 启动器

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

    <!--junit-vintage-engine 提供了 Junit3 与 Junit4 的运
行平台-->
```

```
<exclusions>
  <exclusion>
    <groupId>org.junit.vintage</groupId>
    <artifactId>junit-vintage-engine</artifactId>
  </exclusion>
</exclusions>
</dependency>
</dependencies>
```

2.2 编写测试代码

```
@SpringBootTest
class SpringbootexceptionandjunitApplicationTests {

    @Autowired
    private UsersServiceImpl usersService;

    @Test
    void suibian() {
        this.usersService.addUser();
    }

}
```


十一、Spring Boot 服务端数据校验

1 Spring Boot 对实体对象的校验

1.1 搭建项目环境

1.1.1 创建项目



1.1.2 创建实体

```
public class Users {  
  
    private Integer userid;  
    private String username;  
    private String usersex;  
  
    public Integer getUserid() {  
        return userid;  
    }  
  
    public void setUserid(Integer userid) {  
        this.userid = userid;  
    }  
  
    public String getUsername() {  
        return username;  
    }  
  
    public void setUsername(String username) {  
        this.username = username;  
    }  
}
```

```
public String getUsersex() {
    return usersex;
}

public void setUsersex(String usersex) {
    this.usersex = usersex;
}
}
```

1.1.3 创建 Controller

```
@Controller
@RequestMapping("/user")
public class UsersController {

    /**
     * 添加用户
     */
    @RequestMapping("/addUser")
    public String addUser(Users users) {
        System.out.println(users);
        return "ok";
    }
}
```

1.1.4 创建页面

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.thymeleaf.org">
<link rel="shortcut icon" href="../resources/favicon.ico"
th:href="@{/static/favicon.ico}"/>
<head>

    <title>北京尚学堂首页</title>

</head>
```

```
<body>

    <form th:action="@{/user/addUser}" method="post">
        <input type="text" name="username"><br/>
        <input type="text" name="usersex"><br/>
        <input type="submit" value="OK"/>
    </form>
</body>
</html>
```

1.2 对实体对象做数据校验

1.2.1 Spring Boot 数据校验的技术特点

Spring Boot 中使用了 Hibernate-validator 校验框架。

1.2.2 对实体对象数据校验步骤

1.2.2.1 修改实体类添加校验规则

```
/**
 * @NotNull: 对基本数据类型的对象类型做非空校验
 * @NotBlank: 对字符串类型做非空校验
 * @NotEmpty: 对集合类型做非空校验
 */
public class Users {
    @NotNull
    private Integer userid;
    @NotBlank
    private String username;
    @NotBlank
    private String usersex;

    public Integer getUserid() {
        return userid;
    }

    public void setUserid(Integer userid) {
```

```

        this.userid = userid;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getUsersex() {
        return usersex;
    }

    public void setUsersex(String usersex) {
        this.usersex = usersex;
    }

    @Override
    public String toString() {
        return "Users{" +
            "userid=" + userid +
            ", username='" + username + '\'' +
            ", usersex='" + usersex + '\'' +
            '}';
    }
}

```

1.2.2.2 在 Controller 中开启校验

```

@Controller
@RequestMapping("/user")
public class UsersController {

    /**
     * 添加用户
     */
    @RequestMapping("/addUser")
    public String addUser(@Validated Users
users, BindingResult result) {

```

```

    if(result.hasErrors()){
        /*List<ObjectError> list = result.getAllErrors();
        for(ObjectError err:list){
            FieldError fieldError = (FieldError) err;
            String fieldName = fieldError.getField();
            String msg = fieldError.getDefaultMessage();
            System.out.println(fieldName+"\t"+msg);
        }*/
        return "addUser";
    }
    System.out.println(users);
    return "ok";
}
}

```

1.2.2.3 在页面中获取提示信息

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns:th="http://www.thymeleaf.org">
<link rel="shortcut icon" href="../resources/favicon.ico"
th:href="@{/static/favicon.ico}"/>
<head>
    <title>北京尚学堂首页</title>
</head>
<body>
    <form th:action="@{/user/addUser}" method="post">
        <input type="text" name="username"><font
color="red"><span
th:errors="${users.username}"/></font><br/>
        <input type="text" name="usersex"><font
color="red"><span
th:errors="${users.usersex}"/></font><br/>
        <input type="submit" value="OK"/>
    </form>
</body>
</html>

```

1.2.3 自定义错误提示信息

1.2.3.1 在注解中定义提示信息

```
@NotNull(message = "用户 ID 不能为空")
private Integer userid;

@NotBlank(message = "用户姓名不能为空")
private String username;

@NotBlank(message = "用户性别不能为空")
private String usersex;
```

1.2.3.2 在配置文件中定义提示信息

配置文件名必须是 ValidationMessages.properties

```
userid.notnull=\u7528\u6237Id\u4e0d\u80fd\u4e3a\u7a7a-pro
username.notnull=\u7528\u6237\u59d3\u540d\u4e0d\u80fd\u4e3a\u7a7a-pro
usersex.notnull=\u7528\u6237\u6027\u522b\u4e0d\u80fd\u4e3a\u7a7a-pro
```

1.2.4 解决页面跳转异常

在跳转页面的方法中注入一个对象，要求参数对象的变量名必须是对象类型名称首字母小写格式。

```
@Controller
public class PageController {

    /**
     * 跳转页面方法
     * 解决异常的方式：可以在跳转页面的方法中注入一个 Users 对象
     * 由于 SprignMVC 会将该对象放入到 Model 中传递，key 的名称会使用
     该对象
```

```
* 的驼峰命名规则来作为 key
*/
@RequestMapping("/{page}")
public String showPage(@PathVariable String page, Users
users){
    return page;
}
}
```

1.2.5修改参数 key 的名称

```
/**
 * 跳转页面方法
 *解决异常的方式：可以在跳转页面的方法中注入一个 Users 对象
 * 由于 SprignMVC 会将该对象放入到 Model 中传递,key 的名称会使用该对
象
 * 的驼峰命名规则来作为 key
*/
@RequestMapping("/{page}")
public String showPage(@PathVariable String page,
@ModelAttribute("aa") Users suibian){
    return page;
}
```

```
/**
 * 添加用户
*/
@RequestMapping("/addUser")
public String addUser(@ModelAttribute("aa")@Validated Users
users, BindingResult result){
    if(result.hasErrors()){
        List<ObjectError> list = result.getAllErrors();
        for(ObjectError err:list){
            FieldError fieldError = (FieldError) err;
            String fieldName = fieldError.getField();
        }
    }
}
```

```
String msg = fieldError.getDefaultMessage();
System.out.println(fieldName+"\t"+msg);
}
return "addUser";
}
System.out.println(users);
return "ok";
}

<form th:action="@{/user/addUser}" method="post">
    <input type="text" name="username"><font
color="red"><span th:errors="${aa.username}"/></font><br/>
    <input type="text" name="usersex"><font
color="red"><span th:errors="${aa.usersex}"/></font><br/>
    <input type="submit" value="OK"/>
</form>
```

1.2.6 其他校验规则

@NotNull: 判断基本数据类型的对象类型是否为 null
@NotBlank: 判断字符串是否为 null 或者是空串(去掉首尾空格)。
@NotEmpty: 判断集合是否为空。
@Length: 判断字符的长度(最大或者最小)
@Min: 判断数值最小值
@Max: 判断数值最大值
@Email: 判断邮箱是否合法

2 Spring Boot 对 Controller 中其他参数的校验

2.1 编写页面

2.2 对参数指定校验规则

```
@PostMapping("/findUser")

public String findUser(@NotBlank(message = "用户名不能为空")
String username){
    System.out.println(username);
    return "ok";
}
```


2.3 在 Controller 中开启校验

```
@Controller
@RequestMapping("/user")
@Validated
public class UsersController {
```

2.4 通过全局异常处理来跳转页面

```
@Configuration
public class GlobalException implements
HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest
httpServletRequest, HttpServletResponse
httpServletResponse, Object handler, Exception e) {
        ModelAndView mv = new ModelAndView();

        //判断不同异常类型，做不同视图的跳转

        if(e instanceof NullPointerException){
            mv.setViewName("error5");
        }
        if(e instanceof ArithmeticException){
            mv.setViewName("error6");
        }
        if(e instanceof ConstraintViolationException){
            mv.setViewName("findUser");
        }
        mv.addObject("error",e.getMessage().split(":")[1]);
        return mv;
    }
}
```

十二、 Spring Boot 热部署

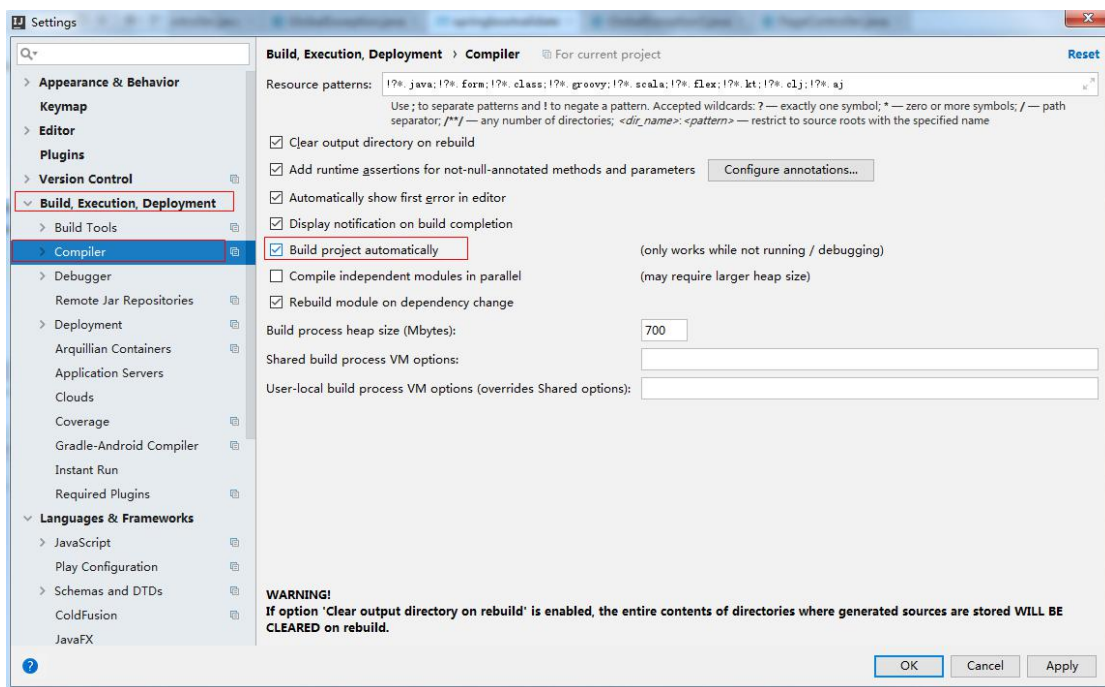
1 通过 DevTools 工具实现热部署

1.1 修改 POM 文件，添加 DevTools 依赖

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

1.2 配置 Idea

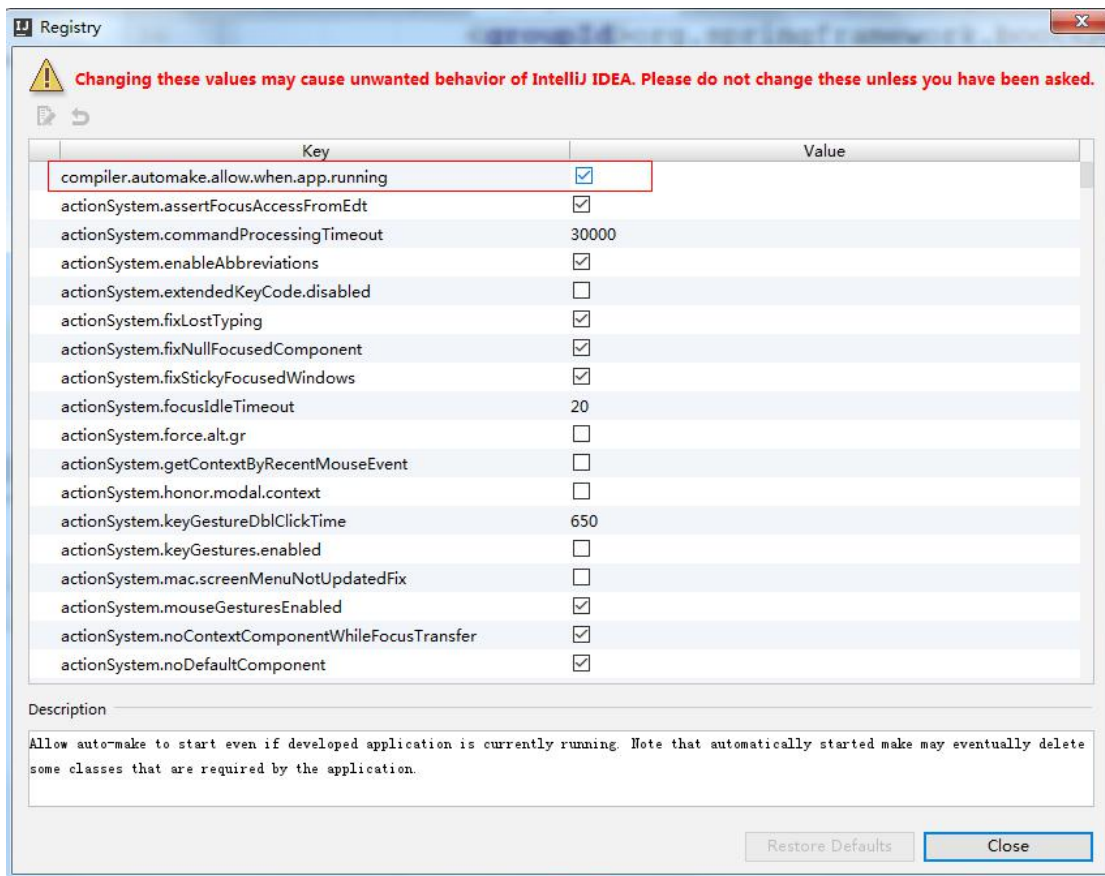
1.2.1 设置自动编译



1.2.2 设置 Idea 的 Registry

通过快捷键打开该设置项：Ctrl+Shift+Alt+/

勾选 compiler.automake.allow.when.app.running



十三、 Spring Boot 度量指标监控与健康检查

1 使用 Actuator 检查与监控

1.1 创建项目



1.2 需改 POM 文件，添加依赖

```
<!--Actuator 坐标依赖-->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

1.3 修改配置文件

```
#配置访问端点的根路径
management.endpoints.web.base-path=/actuator

#配置开启其他端点的 URI
#开启所有的端点访问： *
#指定开启端点访问： 如： beans,env
management.endpoints.web.exposure.include=*
```

1.4 各项监控指标接口 URL 介绍

HTTP 方法	路径	描述
应用配置类		

GET	/configprops	<p>描述配置属性(包含默认值)</p> <ul style="list-style-type: none"> prefix属性代表了属性的配置前缀 properties代表了各个属性的名称和值。
GET	/beans	<p>描述应用程序上下文里全部的Bean，以及它们的关系</p> <ul style="list-style-type: none"> bean：Bean的名称 scope：Bean的作用域 type：Bean的Java类型 resource：class文件的具体路径 dependencies：依赖的Bean名称
GET	/env	<p>获取应用所有可用的环境属性报告。包括：环境变量、JVM属性、应用的配置配置、命令行中的参数</p>
GET	/env/{name}	<p>根据名称获取特定的环境属性值 /env 接口还能用来获取单个属性的值，只需要在请求时在 /env 后加上属性名即可。</p>
GET	/info	<p>该端点用来返回一些应用自定义的信息。默认情况下，该端点只会返回一个空的json内容。 我们可以在application.properties配置文件中通过info前缀来设置一些属性</p>
GET	/mappings	<p>描述全部的URI路径，以及它们和控制器(包含Actuator端点)的映射关系 罗列出应用程序发布的全部接口</p> <ul style="list-style-type: none"> bean属性标识了该映射关系的请求处理器 method属性标识了该映射关系的具体处理类和处理函数。

度量指标类（提供的报告内容则是动态变化的）

GET	/metrics	<p>报告各种应用程序度量信息，比如内存用量、HTTP请求计数、线程信息、垃圾回收信息等。</p> <ul style="list-style-type: none"> 系统信息：包括处理器数量processors、运行时间uptime和instance.uptime、系统平均负载systemload.average。 mem.*：内存概要信息，包括分配给应用的总内存数量以及当前空闲的内存数量。这些信息来自java.lang.Runtime。 heap.*：堆内存使用情况。这些信息来自java.lang.management.MemoryMXBean接口中getHeapMemoryUsage方法获取的java.lang.management.MemoryUsage。 nonheap.*：非堆内存使用情况。这些信息来自java.lang.management.MemoryMXBean接口中getNonHeapMemoryUsage方法获取的java.lang.management.MemoryUsage。 threads.*：线程使用情况，包括线程数、守护线程数(daemon)、线程峰值(peak)等，这些数据均来自java.lang.management.ThreadMXBean。 classes.*：应用加载和卸载的类统计。这些数据均来自java.lang.management.ClassLoadingMXBean。 gc.*：垃圾收集器的详细信息，包括垃圾回收次数gc.ps_scavenge.count、垃圾回收消耗时间gc.ps_scavenge.time、标记-清除算法的次数gc.ps_marksweep.count、标记-清除算法的消耗时间gc.ps_marksweep.time。这些数据均来自java.lang.management.GarbageCollectorMXBean。 httpsessions.*：Tomcat容器的会话使用情况。包括最大会话数httpsessions.max和活跃会话数httpsessions.active。该度量指标信息仅在引入了嵌入式Tomcat作为应用容器的时候才会提供。 gauge.*：HTTP请求的性能指标之一，它主要用来反映一个绝对数值。比如上面示例中的gauge.response.hello: 5，它表示上一次hello请求的延迟时间为5毫秒。 counter.*：HTTP请求的性能指标之一，它主要作为计数器来使用，记录了增加量和减少量。如上示例中counter.status.200.hello: 11，它代表了hello请求返回200状态的次数为11。 <p>对于gauge.*和counter.*的统计，这里有一个特殊的内容请求star-star，它代表了对静态资源的访问。</p>
GET	/metrics/{name}	<p>报告指定名称的应用程序度量值</p>

GET	/health	报告应用程序的健康指标，这些值由HealthIndicator的实现类提供
GET	/dump	获取线程活动的快照。它使用java.lang.management.ThreadMXBean的dumpAllThreads方法来返回所有含有同步信息的活动线程详情。
GET	/trace	提供基本的HTTP请求跟踪信息(时间戳、HTTP头等)。默认情况下，跟踪信息的存储采用org.springframework.boot.actuate.trace.InMemoryTraceRepository实现的内存方式，始终保留最近的100条请求记录。
操作控制类（操作控制类接口默认是关闭的，如果要使用它们的话，需要通过属性来配置开启。）		
POST	/shutdown	关闭应用程序，要求endpoints.shutdown.enabled设置为true

2 使用可视化监控应用 Spring Boot Admin

2.1 使用步骤

Spring Boot Admin 的使用是需要建立服务端与客户端。

服务端：独立的项目，会将搜集到的数据在自己的图形界面中展示。

客户端：需要监控的项目。

对应关系：一个服务端可以监控多个客户端。

2.1.1 搭建服务端

2.1.1.1 创建项目



2.1.1.2 修改 POM 文件

注意：目前在 Spring Boot Admin Starter Server2.1.6 版本中不支持 Spring Boot2.2.x 版本，只支持到 2.1.X

```
<!--
https://mvnrepository.com/artifact/de.codecentric/spring-
boot-admin-starter-server -->
<dependency>
  <groupId>de.codecentric</groupId>
```

```
<artifactId>spring-boot-admin-starter-server</artifactId>
  <version>2.1.6</version>
</dependency>
```

2.1.1.3 修改配置文件

```
server.port=9090
```

2.1.1.4 修改启动类

```
@SpringBootApplication
@EnableAdminServer //开启 Spring Boot Admin 服务端
public class SpringbootactuatorserverApplication {

    public static void main(String[] args) {

SpringApplication.run(SpringbootactuatorserverApplication
.class, args);
    }

}
```

2.2 搭建客户端

2.2.1 修改 POM 文件

```
<!--
https://mvnrepository.com/artifact/de.codecentric/spring-
boot-admin-starter-client -->
<dependency>
  <groupId>de.codecentric</groupId>

  <artifactId>spring-boot-admin-starter-client</artifactId>
  <version>2.1.6</version>
</dependency>
```

2.2.2 修改配置文件

#配置访问端点的根路径

management.endpoints.web.base-path=/actuator

#配置开启其他端点的 URI

#开启所有的端点访问: *

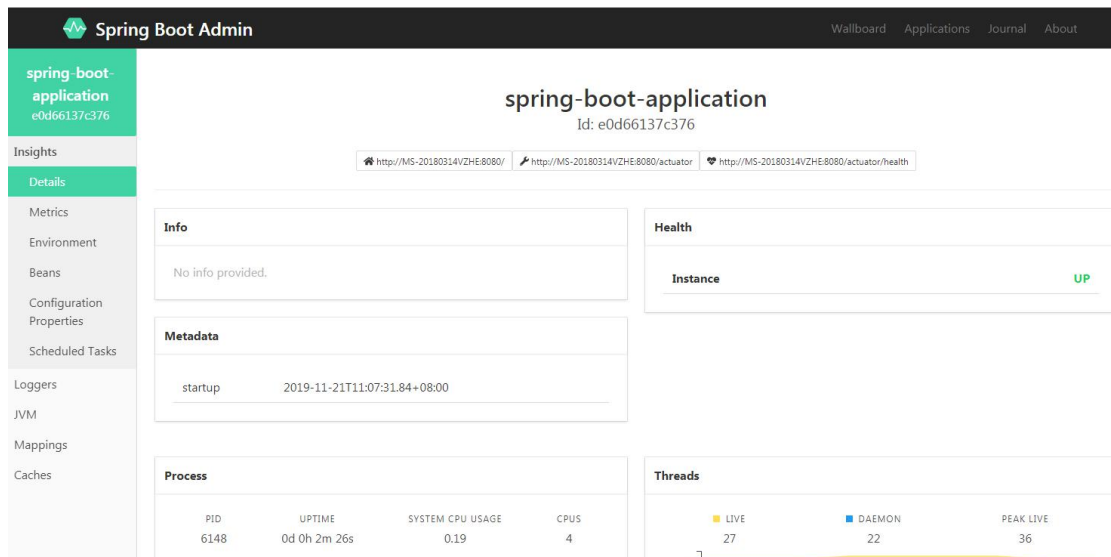
#指定开启端点访问: 如: beans,env

management.endpoints.web.exposure.include=*

#指定服务端的访问地址

spring.boot.admin.client.url=http://localhost:9090

2.2.3 效果图



十四、 Spring Boot 的日志管理

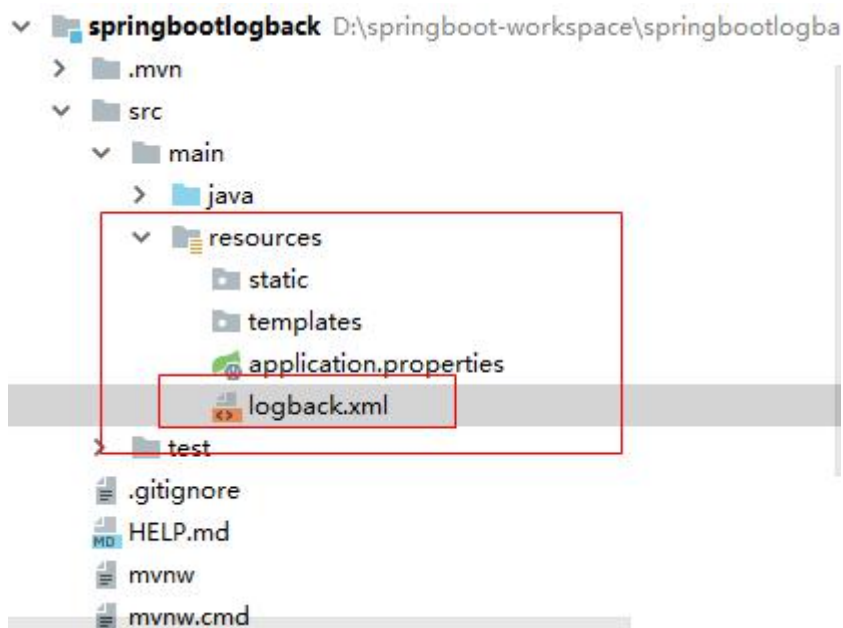
Spring Boot 默认使用 Logback 组件作为日志管理。Logback 是由 log4j 创始人设计的一个开源日志组件。

在 Spring Boot 项目中我们不需要额外的添加 Logback 的依赖,因为在 spring-boot-starter 或者 spring-boot-starter-web 中已经包含了 Logback 的依赖。

1 Logback 读取配置文件的步骤

- (1) 在 classpath 下查找文件 logback-test.xml
- (2) 如果文件不存在,则查找 logback.xml
- (3) 如果两个文件都不存在,LogBack 用 BasicConfiguration 自动对自己进行最小化配置,这样既实现了上面我们不需要添加任何配置就可以输出到控制台日志信息。

2 添加 Logback 配置文件



3 配置 Logback

4 在代码中使用 Logback

```
@RestController
@RequestMapping("/logback")
public class HelloController {

    private final static Logger logger =
```

```
LoggerFactory.getLogger(HelloController.class);

@RequestMapping("/showInfo")
public String showInfo() {

    logger.info("记录日志");

    return "Hello Logback";

}
```

5 在配置文件中屏蔽指定包的日志记录

```
#屏蔽指定包中的日志输出

logging.level.org=off
```

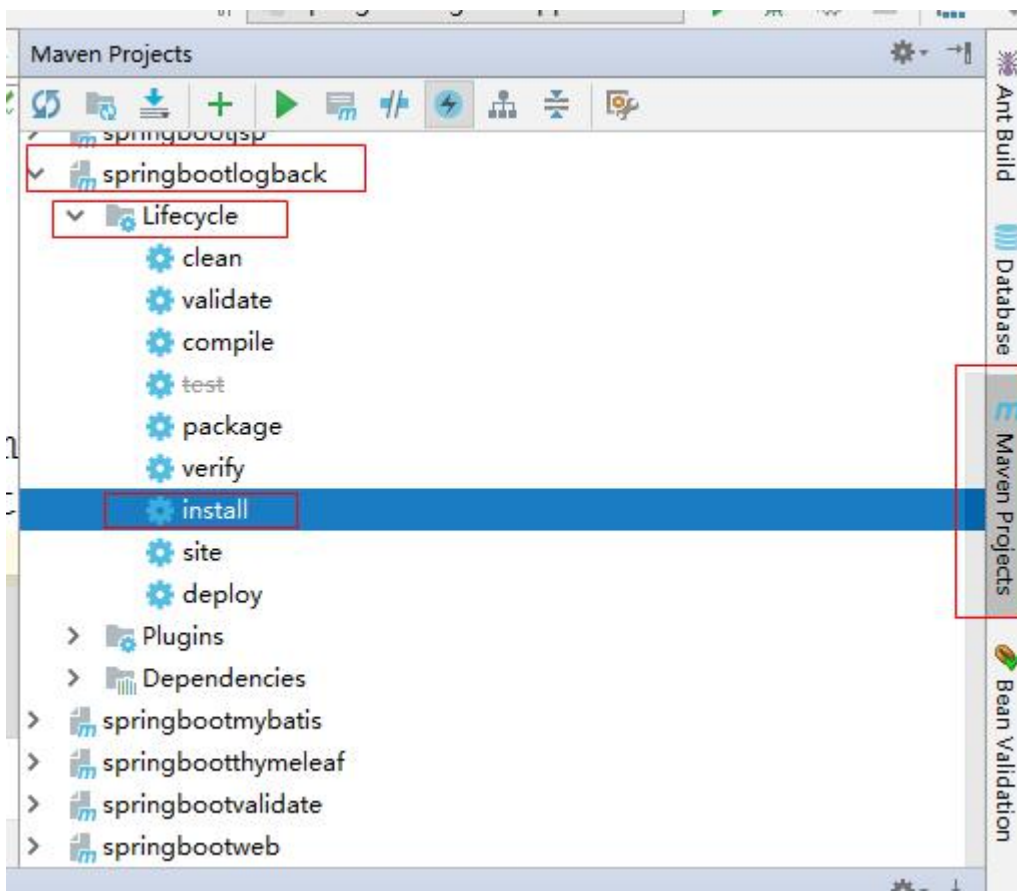
十五、 Spring Boot 项目打包与多环境配置

1 Spring Boot 项目打包

1.1 Spring Boot 的打包插件

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

1.2 项目打包方式



1.3 运行命令

注意：需要正确的配置环境变量。

运行命令：java -jar 项目的名称

2 Spring Boot 的多环境配置

语法结构：application-{profile}.properties/yml

profile：代表某个配置环境的标识

示例：application-dev.properties/yml 开发环境
 application-test.properties/yml 测试环境
 application-prod.properties/yml 生产环境

2.1 Windows 环境下启动方式

java -jar xxx.jar --spring.profiles.active={profile}

2.2 在 Linux 环境下启动方式

2.2.1 安装上传下载工具

安装命令: `yum install lrzsz -y`

上传命令: `rz`

下载命令: `sz` 下载文件名

2.2.2 启动脚本的使用

修改脚本文件中的参数值

将启动脚本文件上传到 Linux 中

分配执行权限: `chmod 777`

通过脚本启动命令: `server.sh start`

通过脚本关闭命令: `server.sh stop`