**Springboot实战**

**目录**

[1 Springboot简介 3](#_Toc24344)

[2 第一个应用程序 4](#_Toc25303)

[2.1 创建应用 4](#_Toc10081)

[2.2 Springboot详解 6](#_Toc3998)

[2.2.1 Springboot启动器： 6](#_Toc1484)

[2.2.2 Springboot注解 7](#_Toc25926)

[2.3 Springboot配置文件 7](#_Toc21690)

[2.4 Springboot打包 7](#_Toc8422)

[3 Springboot实战 8](#_Toc24639)

[3.1 Springboot配置文件 8](#_Toc7242)

[3.2 多环境配置文件用法 10](#_Toc2260)

[3.3 整合mybatis 10](#_Toc29798)

[3.4 LogBack日志集成 12](#_Toc9362)

[3.5 Lombok注解 12](#_Toc26645)

[3.6 参数校验 13](#_Toc23402)

[3.7 集成redis 14](#_Toc23040)

[3.8 AOP应用 15](#_Toc24050)

[3.9 集成Swagger 16](#_Toc30379)

[3.10 单元测试 18](#_Toc5055)

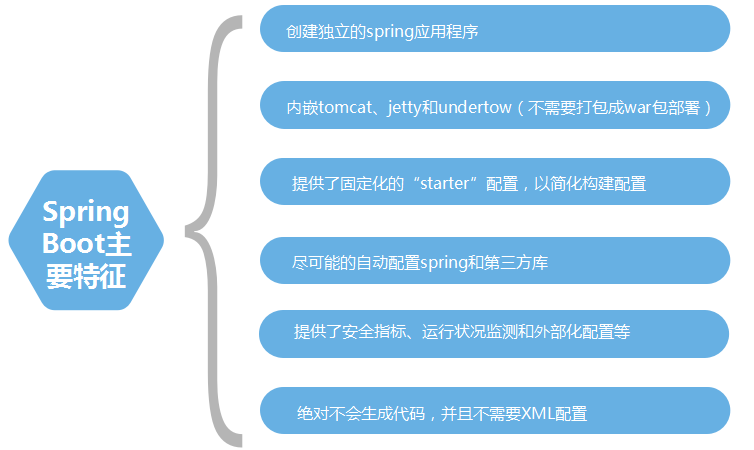
[3.11 Web项目打包 18](#_Toc18423)

[4 参考资料 20](#_Toc1379)

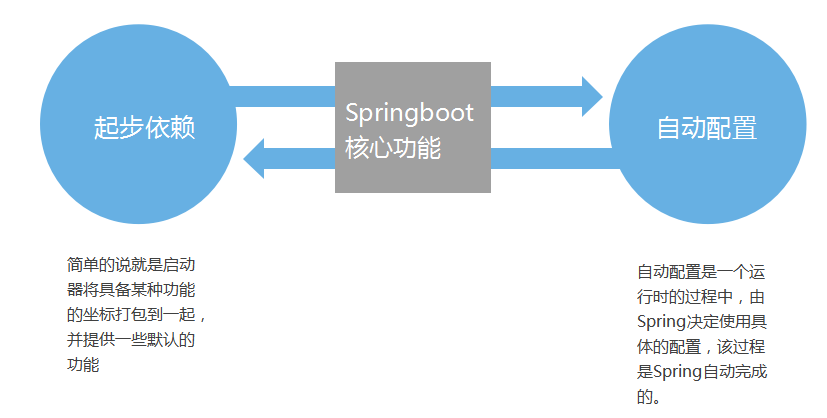
# Springboot简介

Spring的组件代码是轻量级的，但它的配置却是重量级的。一开始，Spring用XML配置，而且是很多XML配置。Spring 2.5引入了基于注解的组件扫描，可以消除大量的xml配置。Springboot官网文档是这样描述的：“Spring Boot你只需要run就可以非常轻易的构建独立的、生产级别的spring应用。我们为spring平台和第三方依赖库提供了一种固定化的使用方式，使你能非常轻松的开始开发你的应用程序。大部分Spring Boot应用只需要很少的配置”。

Springboot主要特性如下图：



Springboot的核心功能如下图：

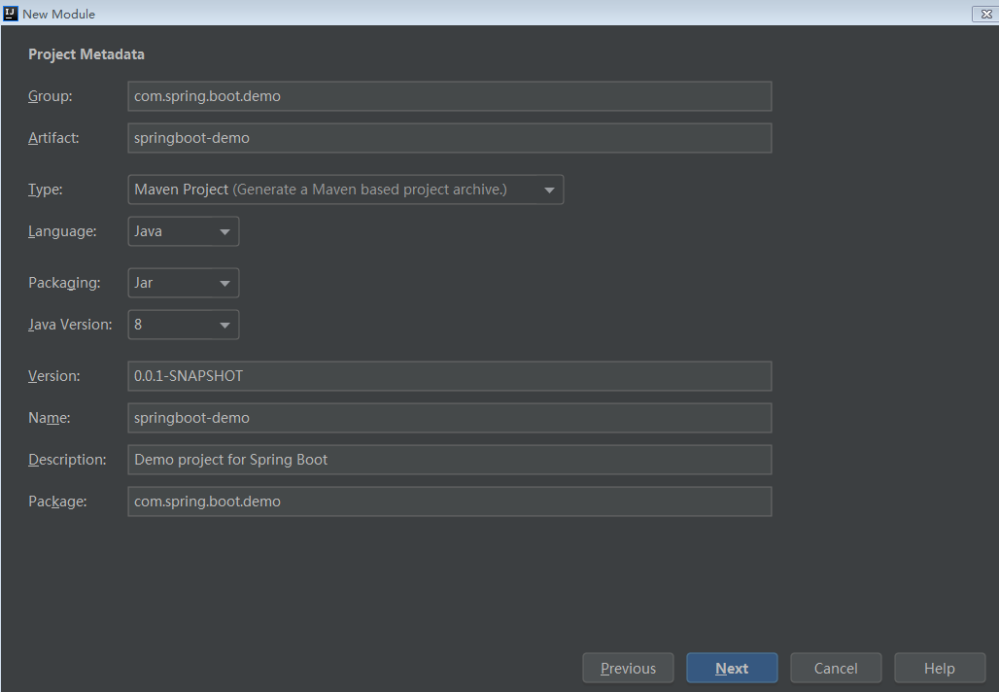


# 第一个应用程序

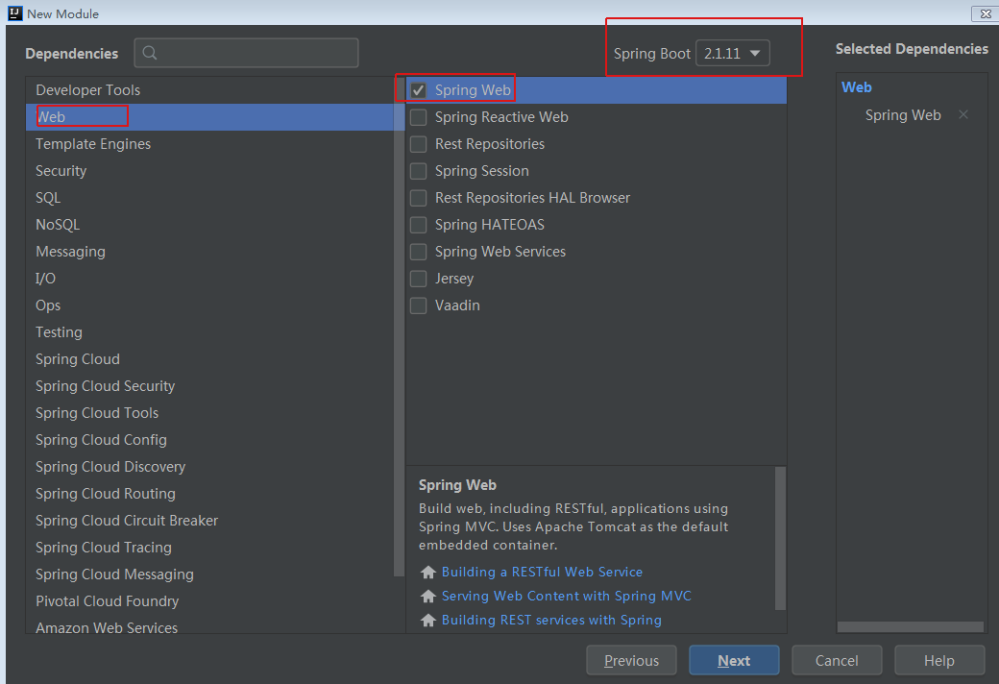
## 创建应用

创建Springboot的应用有很多种方式，可以使用IDE工具生成，也可以用Spring官网的脚手架生成，还可以手动创建并添加依赖。

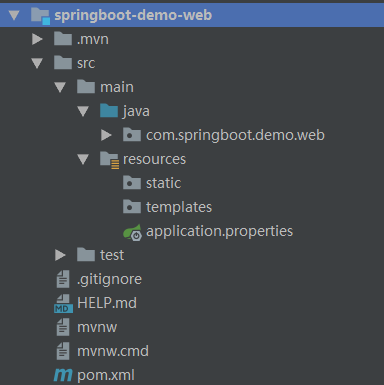
使用IDEA创建Springboot应用 new module ，填入相关信息点击 next



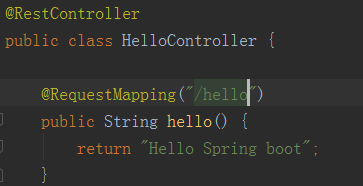
勾选 Spring Web，选择最新版本 2.2.2.RELEASE



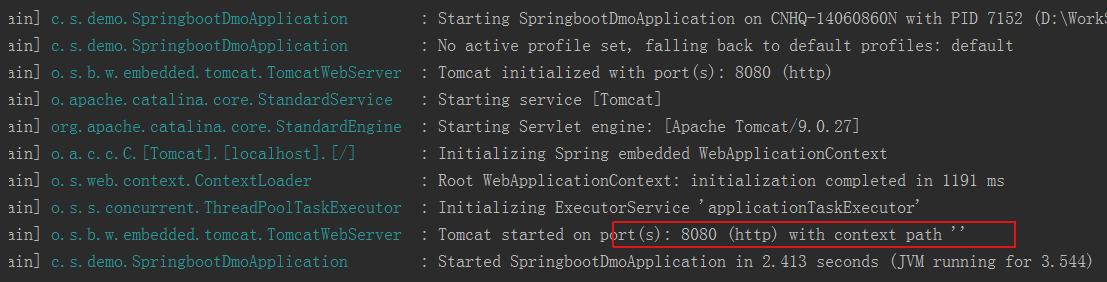
创建完成的工程结构如图：



编写 HelloController.java



启动Springboot应用，运行SpringbootDmoApplication.java 类。



启动完成之后访问 <http://127.0.0.1:8080/hello> 发现网页输出

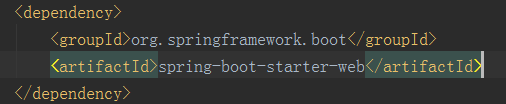
Hello Spring boot

访问成功! 太神奇了，没有任何的配置，竟然可以访问成功

## Springboot详解

### Springboot启动器：

为了让Springboot 自动完成各种配置，创建工程时必须要引入Springboot提供的自动配置的依赖。这些自动配置的依赖我们就称为启动器。我们创建的是web工程，所以就必须引入 spring-boot-starter-web 。 Springboot启动器命名规范是 spring-boot-starter-XXX 而自定义的启动器命名规范是 XXX-spring-boot-starter。



这里不用配置版本号，因为spring-boot-starter-parent 已经帮我们管理好了版本号。

详细的Springboot启动器请参考官网 using spring boot---->build systems---->starters

### Springboot注解

SpringbootDmoApplication 类上加了一个@SpringBootApplication，表示这是一个Springboot的应用，它包含了 @SpringBootConfiguration @EnableAutoConfiguration @ComponentScan 这个三个注解的功能。

@SpringBootConfiguration：继承 @Configuration，功能也是一致的，表示当前类是Springboot的配置类，项目中只有一个无需自己添加。

@EnableAutoConfiguration：开启自动化配置，具体参考

@ComponentScan：扫描器的配置，相当于之前的 <context:component-scan>，spring 会自动发现被 @Component、@Repository、@Service 和 @Controller 标注的类，并注册进容器中。

综上所述，@SpringBootApplication 就是三个注解的集合。

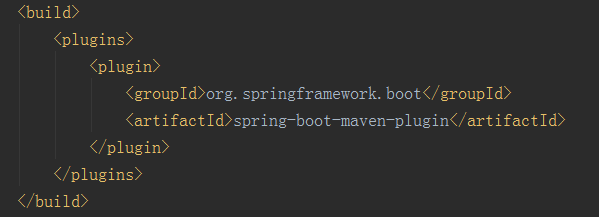
## Springboot配置文件

Springboot是基于约定的，所以有很多配置都是默认的。如果要替换默认配置的话，就可以使用application.properties或者application.yml进行配置。

pringBoot默认会从Resources目录下加载application.properties或application.yml。具体的配置项

## Springboot打包

在pom.xml文件中，引入了 spring-boot-maven-plugin的插件，提供了maven对Springboot的支持，能够将源码打成jar文件或者war文件，并且可以就地运行。可以执行的jar通常称为fat jars。



# Springboot实战

## Springboot配置文件

#### 随机值配置



代码中通过 @Value("${my.secret}") 去使用。

访问 <http://127.0.0.1:8080/random/random.do> 可以将随机值 输出。



#### 命令行配置

server.port配置属于命令行配置的一种，详细的Springboot的命令行配置 查看官网 参考文档的“Application Properties”。

#### 系统配置文件加载

SpringbootApplication加载application.properties配置文件添加到Spring环境中，加载配置文件顺序如下：

1、当前目录下的 /config 子目录

2、当前目录

3、Classpath下的config目录

4、Classpath目录

#### 属性占位符

当application.properties文件中要引用事先定义的值可以使用${key}

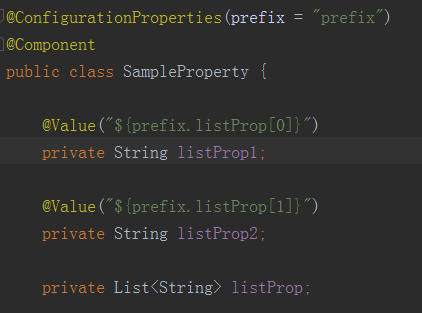
#### yml文件



等同的properties配置：



可以这么使用 ，也可以作用在@Bean注解类上。



YAML是JSON的超集，非常适合定义层级化配置数据，Springboot推荐使用yml文件配置。

## 多环境配置文件用法

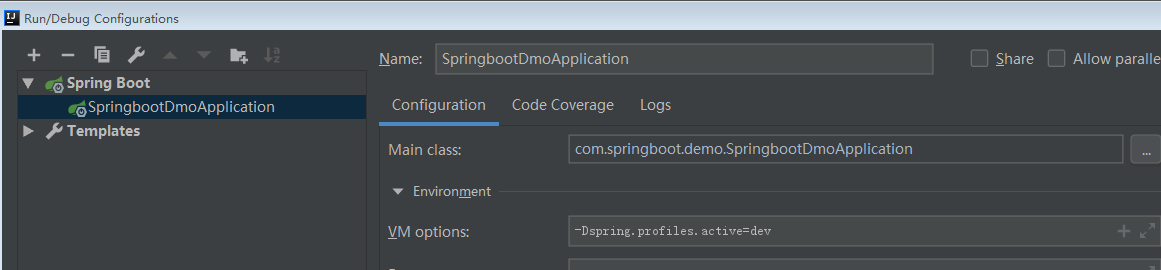
Spring Profiles提供了一种隔离应用程序配置的方式，并让这些配置只能在特定的环境下生效。

在Spring Boot中多环境配置文件名需要使用application-{profile}.properties的格式，这里的{profile}对应的是你的环境标识例如:  
application-dev.properties application-sit.properties

application-pre.properties application-prd.properties

配置文件使用是有以下三种方式

1. application.properties配置文件 中设置 spring.profiles.active={profile}
2. java -jar jar\_name.jar --spring.profiles.active={profile}
3. 在IDEA的中VM options 中设置 -Dspring.profiles.active={profile}



## 整合mybatis

1. 在pom.文件中引入mybatis的启动器和mysql驱动

<!-- spring-boot整合mybatis 启动器 -->

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>2.0.1</version>

</dependency>

<!-- mysql驱动 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

2、在yml文件中配置数据源和mybatis配置

## 数据源配置

spring:

datasource:

url: jdbc:mysql://10.47.154.229:3306/springbootdemo

username: fabu

password: 73R4\_h8td6fE

driver-class-name: com.mysql.jdbc.Driver

##mybatis配置

mybatis:

type-aliases-package: com.springboot.demo.dmo

mapper-locations: classpath:mybatis/mapper/\*.xml

1. 创建 CityInfoMapper.xml 详见源码文件
2. 创建 CityInfoDao CityInfoDaoImpl

在实现类中注入 mybatis启动器的 SqlSessionTemplate

1. 创建 CityInfoService CityInfoServiceImpl 详见源码
2. 创建 CityInfoController 详见源码

最后访问controller的http请求，实现和数据库的交互。

<http://127.0.0.1:8080/city/cityList.do>

<http://127.0.0.1:8080/city/getCityById.do?id=6>

[http://127.0.0.1:8080/city/getCityByName.do?name=南](http://127.0.0.1:8080/city/getCityByName.do?name=%E5%8D%97%E4%BA%AC)京

## LogBack日志集成

Spring Boot内部日志使用Commons Logging，但是开发底层日志实现。会为Java Util Logging, Log4j, Log4j2和Logback提供默认配置。如果使用Springboot的启动器（starters）Logback是默认日志实现，默认输出到控制台，通过配置可以输出到指定的文件中，自定义logback配置文件名称Springboot推荐使用 logback-spring.xml

控制台日志配置：

logging.level.root=warn

logging.level.org.springframework.web=debug

logging.level.org.hibernate=error

输出日志到指定文件：

Logging.file.name= D:\opt\logs\my.log

输出日志到指定目录，文件名称默认

Logging.file.path= D:\opt\logs\

输出日志到指定文件中，文件路径默认

Logging.file.name=my.log

Logback日志输出：

自定义logback-spring.xml(见源码)配置文件放到classpath目录下即可。

访问 <http://127.0.0.1:8080/city/cityList.do> 观察日志输出。

## Lombok注解

lombok 可以通过简单的注解编译的时候生成一些必须有的代码，最常见的就是省略JavaBean 的 Getter/Setter 方法，最终达到的目的（@Getter和@Setter为例）：源码中没有 get、set 方法，编译生成的字节码文件中有 get、set 方法。

引入依赖：

<!--lombok依赖-->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.16.18</version>

</dependency>

Lombok常用注解介绍：

@Getter/@Setter ：作用在类上和属性上生成get、set方法。类:作用于所有的非静态属性，属性：只作用该属性。

@ToString ：类上生成toString()方法。

@NoArgsConstructor ：类上；生成无参的构造

@AllArgsConstructor ：类上；生成全参的构造

@Slf4j : 类上;这里根据自己的项目中使用的是什么日志框架决定使用@Slf4j、@Log4j等等

@Data ：类上；组合注解，包含@ToString、@EqualsAndHashCode、@Getter、@Setter、@RequiredArgsConstructor

## 参数校验

引入spring-boot-starter-validation(使用Hibernate Validator 框架提供验证功能) 的启动器，可以对请求参数校验，由于spring-boot-starter-web，因此就不需要再次引入此依赖了。

基本注解介绍：

@NotEmpty：校验String、collection、map、数组，不能为null，size > 0

@NotBlank: 只校验String，不能为null,也不可以是" "

@NotNull：校验任何类型，不能为null，可以为空

@AssertTrue：必须为true

@AssertFalse：必须为false

@Min(value)：最小值

@Max(value)：最大值

@Size(max,min)：限制参数大小范围

@Future：必须是一个将来的日期

@Past：必须是一个过去的日期

@Pattern(value)：指定正则

1. 创建实体类：ValidEntity
2. 创建 ValidationController
3. 访问 <http://127.0.0.1:8080/valid/valid.do?name=lsh&height=20> 并修改参数验证

## 集成redis

Redis作为一个noSql数据库，在项目经常使用到，下面我么就看一下Springboot是如何集成redis 的。

1. 引入 redis启动器

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

1. 配置yml文件中redis连接

##redis配置

spring:

redis:

database:

host: 127.0.0.1

port: 6379

jedis:

pool:

max-active: 8

max-wait: -1

max-idle: 8

min-idle: 0

timeout: 10000

##redis集群配置

##nodes:

## - 127.0.0.1:6379

## - 127.0.0.1:6378

1. 创建 RedisUtil工具类 祥见源码
2. 创建RedisController 并使用

@Resource

private RedisTemplate<String, Object> redisTemplate;

启动本地redis访问：

<http://127.0.0.1:8080/redis/set.do?key=2222&value=2222>

<http://127.0.0.1:8080/redis/get.do?key=2222>

## AOP应用

AOP是Spring的主要功能之一，看一下AOP在Springboot中是如何使用的。

#### 过滤器

1、创建MyFilter.java类型 详见源码

2、MyFilter类加上 @WebFilter(urlPatterns = "\*.htm", filterName = "MyFilter") 拦截所有的htm的请求

3、在SpringbootDmoApplication类上加 @ServletComponentScan 注解，扫描所有的拦截器。

4、访问 一下链接观察日志可以看出拦截了.htm的请求。

<http://127.0.0.1:8080/city/getCityById.do?id=1>

<http://127.0.0.1:8080/city/getCityById.htm?id=1>

#### 拦截器

1. 创建LoginInterceptor.java 详见源码

2、创建 InterceptorConfig.java

@Configuration

public class InterceptorConfig implements WebMvcConfigurer {

@Override

public void addInterceptors(InterceptorRegistry registry) {

// addPathPatterns 用于添加拦截规则

// excludePathPatterns 用户排除拦截

registry.addInterceptor(new LoginInterceptor())

// .excludePathPatterns("\*.do")

.addPathPatterns("/city/\*.json");

}

}

访问 一下URL观察拦截情况

<http://127.0.0.1:8080/city/getCityById.json?id=1>

<http://127.0.0.1:8080/city/getCityById.do?id=1>

#### Aspect拦截

1、创建MyAspect.java类详见源码，并添加注解@Aspect @Component

1. 添加依赖：

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.5</version>

</dependency>

访问 <http://127.0.0.1:8080/city/cityList.do> 查看切面的执行情况

综上所述：1、Springboot使用 @WebFilter 注解替代web.xml配置 2、拦截器使用WebMvcConfigurer实现替代 在spring-mvc.xml配置文件内添加<mvc:interceptor>标签配置拦截器。

3.Aspect和Spring的区别不大，都可以使用注解。

## 集成Swagger

随着前后端分离越来越流行，编写和维护接口文档的成本就越来越高。而Swagger的出现恰好解决了这个问题，是一个规范和完整的框架，用于生成、描述、调整和可视化的restful风格的web服务。

1. 引入Swagger依赖

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.9.2</version>

</dependency>

<dependency>

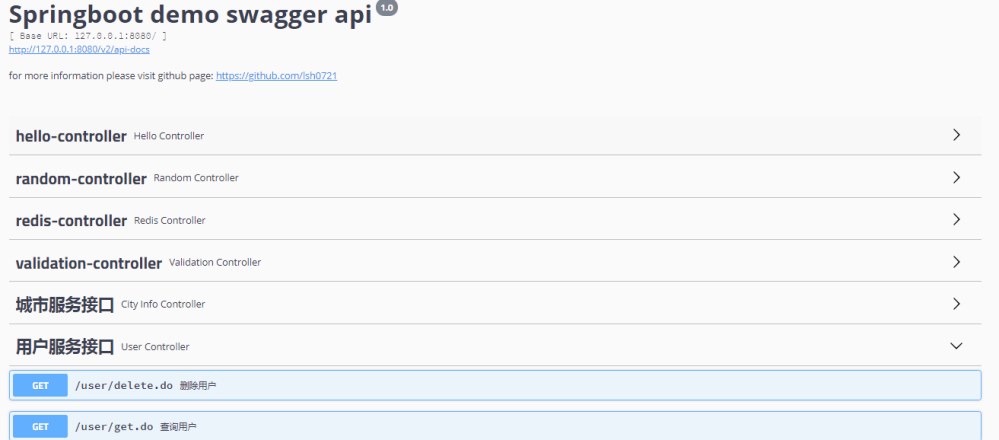
<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.9.2</version>

</dependency>

1. 创建UserController.java类，SwaggerConfig.java 详见源码
2. 访问 <http://127.0.0.1:8080/swagger-ui.html> 使用swagger api页面对数据进行CRUD



Swagger注解如下：

@Api注解可以用来标记当前Controller的功能。

@ApiOperation注解用来标记一个方法的作用。

@ApiImplicitParam注解用来描述一个参数

多个参数用 @ApiImplicitParams({@ApiImplicitParam(...),@ApiImplicitParam(...)})

@ApiModel注解用来标记API中用到的domain，@ApiModelProperty注解用于domain中的属性。

注意！生产环境swagger的API 要在SwaggerConfig类 用注解 @Profile({"dev","sit"}) 来禁用API功能

添加 依赖：

<dependency>

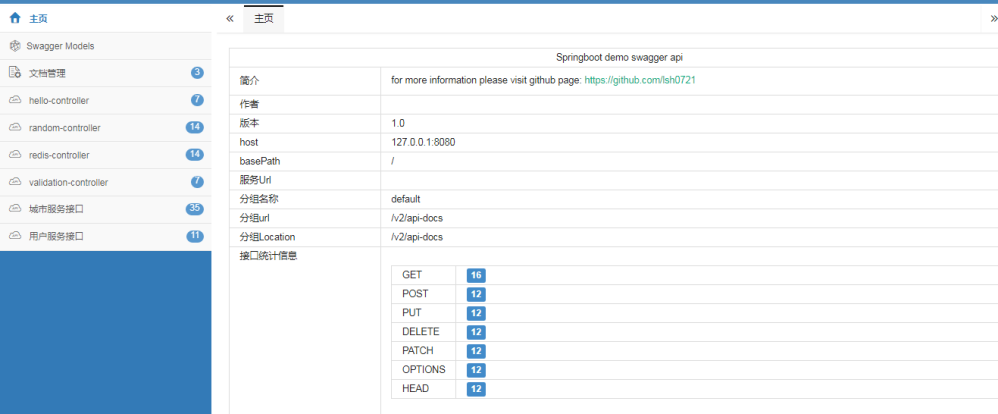
<groupId>com.github.xiaoymin</groupId>

<artifactId>swagger-bootstrap-ui</artifactId>

<version>1.9.6</version>

</dependency>

访问<http://127.0.0.1:8080/doc.html> 查看增强版的swagger api 如下图：



## 单元测试

引入spring-boot-starter-test 启动器，默认集成了junit 、Mockito 、mockmvc .因此 service dao 层可以使用junit、mockito进行测试，API可以使用mockmvc测试。

创建 CityInfoControllerTest.java 详见源码

## Web项目打包

#### 构建war包

修改pom.xml：

<packaging>war</packaging>

排除内嵌tomcat：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<!--排除内嵌tomcat-->

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

执行 mvn clean install -Dmaven.test.skip=true 就可以构成成war包

#### 引入jetty容器

修改pom.xml打包方式为war包

引入jetty依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<!--排除内嵌tomcat-->

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

<!--引入jetty启动器-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jetty</artifactId>

</dependency>

# 参考资料

Springboot参考手册：

<https://docs.spring.io/spring-boot/docs/2.2.2.RELEASE/reference/html/>

Github源码

<https://github.com/lsh0721/springboot-demo>

[git@github.com:lsh0721/springboot-demo.git](mailto:git@github.com:lsh0721/springboot-demo.git)