# springboot集成以及使用swagger

|  |  |  |  |
| --- | --- | --- | --- |
| **操作人** | **操作时间** | **操作内容** | **备注** |
| 胡贺东 | 20170313 | 创建 |  |
| 胡贺东 | 20170317 | 补充内容 | 之前只是谢了大纲，没有具体内容编写 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 使用swagger的优势

1. 解决接口定义与文档脱离问题
2. 解决前后端接口对接不及时以及不合理
3. 解决接口编写文档劳力费力问题

# Springboot集成以及使用swagger

## Springboot集成swagger

### 修改pom.xml

<!-- spring boot 接口文档自动生成相关包 -->

<dependency>

<groupId>io.springfox</groupId>

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

<version>2.6.0</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.6.1</version>

</dependency>

<dependency>

<groupId>org.springframework.restdocs</groupId>

<artifactId>spring-restdocs-mockmvc</artifactId>

<version>1.1.1.RELEASE</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-staticdocs</artifactId>

<version>2.6.1</version>

<scope>test</scope>

</dependency>

### 增加swaggerconfig

package com.pay;

import org.springframework.boot.bind.RelaxedPropertyResolver;

import org.springframework.context.EnvironmentAware;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.env.Environment;

import org.springframework.util.StopWatch;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import springfox.documentation.builders.RequestHandlerSelectors;

import springfox.documentation.service.ApiInfo;

import springfox.documentation.service.Contact;

import springfox.documentation.spi.DocumentationType;

import springfox.documentation.spring.web.plugins.Docket;

import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration

@EnableSwagger2//注意这里

@ComponentScan(basePackages = "com.pay.controller")//接口扫描包名

public class SwaggerConfig extends WebMvcConfigurerAdapter

implements EnvironmentAware {

/\*\*

\* 静态资源映射

\*

\* @param registry

\* 静态资源注册器

\*/

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("swagger-ui.html")

.addResourceLocations("classpath:/META-INF/resources/");

registry.addResourceHandler("/webjars/\*\*")

.addResourceLocations("classpath:/META-INF/resources/webjars/");

super.addResourceHandlers(registry);

}

@Override

public void setEnvironment(Environment environment) {//这里是从配置文件里读相关的字段

this.propertyResolver = new RelaxedPropertyResolver(environment,

"swagger.");

}

@Bean

public Docket swaggerSpringfoxDocket4KAD() {//最重要的就是这里，定义了/test/.\*开头的rest接口都分在了test分组里，可以通过/v2/api-docs?group=test得到定义的json

StopWatch watch = new StopWatch();

watch.start();

Docket swaggerSpringMvcPlugin = new Docket(DocumentationType.SWAGGER\_2)

.groupName("api")

.apiInfo(apiInfo()).select().apis(RequestHandlerSelectors.any())

.build();

watch.stop();

return swaggerSpringMvcPlugin;

}

private ApiInfo apiInfo() {//这里是生成文档基本信息的地方

return new ApiInfo(propertyResolver.getProperty("title"),

propertyResolver.getProperty("description"),

propertyResolver.getProperty("version"),

propertyResolver.getProperty("termsOfServiceUrl"),

new Contact(propertyResolver.getProperty("contact.name"),

propertyResolver.getProperty("contact.url"),

propertyResolver.getProperty("contact.email")),

propertyResolver.getProperty("license"),

propertyResolver.getProperty("licenseUrl"));

}

private RelaxedPropertyResolver propertyResolver;

}

### 增加配置文件内容

swagger.title=北京柚信科技柚信支付接口

swagger.description=柚信支付相内容

swagger.version= 1.0

swagger.termsOfServiceUrl=testtermsOfServiceUrl

swagger.contact.name=胡贺东

swagger.contact.url=testurl

swagger.contact.email=testemail

swagger.license=testlicense

swagger.licenseUrl=testlicenseUrl

### 测试是否成功

启动项目 访问

http://192.168.15.75:8888/api/swagger-ui.html

## Swagger使用

### 标签总体介绍

在不使用 swagger标签的情况下，会自动 获取所有的接口内容。

@Api：用在类上，说明该类的作用。

@ApiOperation：注解来给API增加方法说明。

@ApiImplicitParams : 用在方法上包含一组参数说明。

@ApiImplicitParam：用来注解来给方法入参增加说明。

@ApiResponses：用于表示一组响应

@ApiResponse：用在@ApiResponses中，一般用于表达一个错误的响应信息

    l   **code**：数字，例如400

    l   [**message**](http://www.07net01.com/tags-message-0.html)：信息，例如"请求参数没填好"

    l   **response**：抛出异常的类

@ApiModel：描述一个Model的信息（一般用在请求参数无法使用@ApiImplicitParam注解进行描述的时候）

    l   **@ApiModelProperty**：描述一个model的属性

注意：@ApiImplicitParam的参数说明：

|  |  |
| --- | --- |
| **paramType**：指定参数放在哪个地方 | header：请求参数放置于Request Header，使用@RequestHeader获取  query：请求参数放置于请求地址，使用@RequestParam获取  path：（用于restful接口）-->请求参数的获取：@PathVariable  body：（不常用）  form（不常用） |
| name：参数名 |  |
| dataType：参数类型 |  |
| required：参数是否必须传 | true | false |
| value：说明参数的意思 |  |
| defaultValue：参数的默认值 |  |

### 编码示例

#### 示例代码总体说明

测试代码[TestSwaggerConfig.java](#_TestSwaggerConfig.java_源码)中定义一个模拟查询接口、一个保存接口 [ReqParam](#_ReqParam_源码).java为请求参数、[TestBean](#_TestBean_源码).java为接口模拟业务、[ApiResult](#_ApiResult源码).java为接口返回封装内容

#### 示例代码详细介绍

# 附录

## 自我疑问的一些问题

### 标签过多

在springboot中，spring标签、hibernater valid标签、json标签，Gson标签、swagger标签、其他框架标签。真的太多了，以后融入其他框架 又会有其他框架标签。

如何能够在不使用标签的情况下将功能融入到系统中，

与业务不太相关的内容 切入 业务中 又不给业务系统带来繁琐？？

### 接口改动较为随意

接口后端开发人员定义，很难由项目经理一人把控。接口变动没有主动通知其他人，没有变动记录、或者变动通知。或者全局把控。修改前后区别？

## 参考网址

当时查的时候没有做过记录后来忘了。。。下次一定改

首先要把浏览过比较好的文章做一记录

* 1. <http://www.07net01.com/2016/12/1752448.html>

## 代码附录

### TestSwaggerConfig.java 源码

### ReqParam 源码

### TestBean 源码

### ApiResult源码