

# Swagger

---

## 1 简介

Swagger 是一款RESTful接口的文档在线自动生成+功能测试功能软件，随着现在许多公司实现了前后端分离，swagger越来越受欢迎了。

随着互联网技术的发展，现在的网站架构基本都由原来的后端渲染，变成了：**前端渲染、前后端分离**的形态，而且前端技术和后端技术在各自的道路上越走越远。

前端和后端的唯一联系，变成了**API接口**；API文档变成了前后端开发人员联系的纽带，变得越来越重要，swagger 就是一款让你更好的书写API文档的框架。

- 号称世界上最流行的API框架！
- RESTful风格API文档在线自动生成工具 ==》API文档与API定义同步更新
- 直接运行，可以在线测试API接口
- 支持多种语言：Java、PHP

官网：<https://swagger.io/>

## 2 时代背景

### 2.1 后端时代【过去】

在过去的后端时代，前端只用管理静态页面；

HTML和后端进行数据交互，通过模板引擎【例如SP，更早的EJB】

这个时代，后端是主力。

### 2.2 前后端分离时代【现在】

- 后端：控制层，服务层、数据访问层【后端团队】
- 前端：控制层，视图层【前端团队】
  - 伪造后端数据，JSON，【不需要后端，前端写的工程依旧可以跑起来】
- 前后端如何交互？
  - 通过API接口【双方遵守某个约定】
- 优点
  - 前后端相对独立，松耦合
  - 前后端甚至可以部署在不用的服务器上
- 但是会产生一个问题：前后端无法集成联调，前端人员和后端人员无法做到“及时协商，尽早解决”，最终导致问题集中爆发。
  - 解决方案
    - 首先制定一个schema【计划书】，实时更新API，降低集成的风险
    - 早些年使用Word计划文档
    - 前后端分离：
      - 前端测试后端端口：postman、yapi等工具
      - 后端提供接口，需要实时更新最新的消息和改动
      - 所以，Swagger就是为了解决这个问题！

### 3 在SpringBoot项目中集成

注意：在Spring项目中也可以使用，步骤一样！

- 1 要想在项目中使用Swagger，需要Springfox
- 2 - Swagger2
- 3 - UI

- 新建一个SpringBoot Web项目====》添加Web支持
- 导入依赖

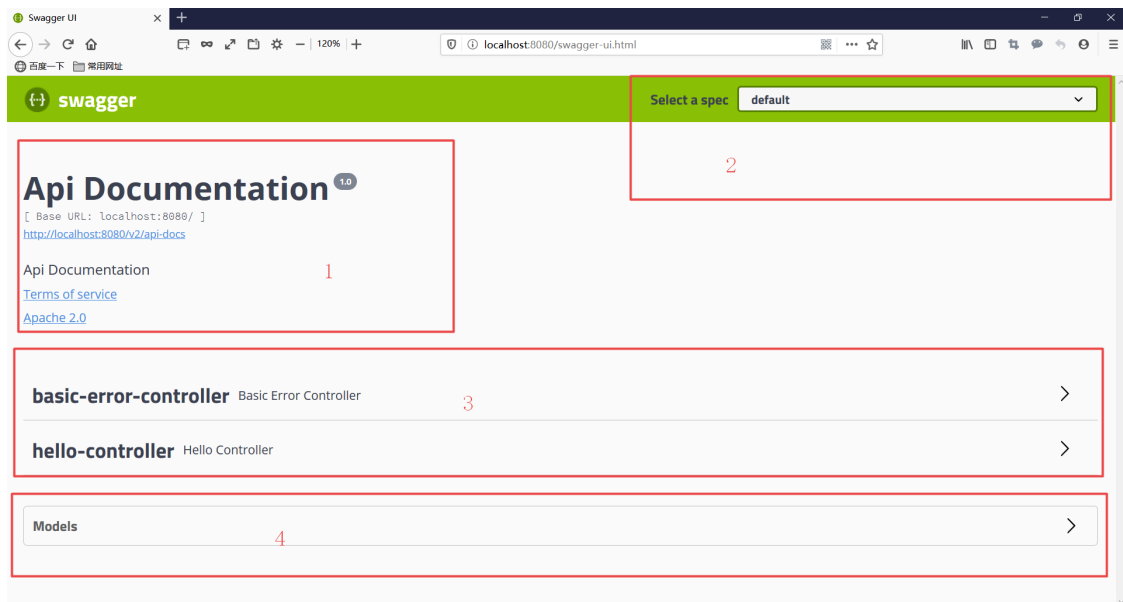
```
1 <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2
-->
2 <dependency>
3     <groupId>io.springfox</groupId>
4     <artifactId>springfox-swagger2</artifactId>
5     <version>2.9.2</version>
6 </dependency>
7
8 <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-
ui -->
9 <dependency>
10     <groupId>io.springfox</groupId>
11     <artifactId>springfox-swagger-ui</artifactId>
12     <version>2.9.2</version>
13 </dependency>
```

- 配置Swagger

```
1 package com.kuang.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import springfox.documentation.swagger2.annotations.EnableSwagger2;
5
6 @Configuration //配置到Spring中
7 @EnableSwagger2 //开启Swagger2
8 public class SwaggerConfig {
9 }
```

这样只是最简单的使用一下Swagger

- 测试运行
- 进入网页: <http://localhost:8080/swagger-ui.html>



可以看得出分成四块：

- 1: Swagger信息
- 2: 选择框【组】
- 3: 接口信息
- 4: 实体类信息

现在已经可以进行最基本的使用了。

## 4 配置Swagger

Swagger的Bean实例Docket，所有的配置都基于Docket。

点进源码：

```
public Docket(DocumentationType documentationType) {  
    this.apiInfo = ApiInfo.DEFAULT;  
    this.groupName = "default";  
    this.enabled = true;  
    this.genericsNamingStrategy = new DefaultGenericTypeNamingStrategy();  
    this.applyDefaultResponseMessages = true;  
    this.host = "";  
    this.pathMapping = Optional.absent();  
    this.apiSelector = ApiSelector.DEFAULT;  
    this.enableUrlTemplating = false;  
    this.vendorExtensions = Lists.newArrayList();  
    this.documentationType = documentationType;  
}
```

可以看到可配置的信息。

### 4.1 配置Swagger的基本信息==>apiInfo

看源码：

```

1 public class ApiInfo {
2
3     public static final Contact DEFAULT_CONTACT = new Contact("", "", "");
4     public static final ApiInfo DEFAULT = new ApiInfo("Api Documentation",
5         "Api Documentation",
6         "1.0",
7         "urn:tos",
8         DEFAULT_CONTACT,
9         "Apache 2.0", "http://www.apache.org/licenses/LICENSE-2.0",
10        new ArrayList<VendorExtension>());

```

## Api Documentation <sup>1.0</sup>

[ Base URL: localhost:8080/ ]  
<http://localhost:8080/v2/api-docs>

Api Documentation

[Terms of service](#)

[Apache 2.0](#)

可以看到这个页面的信息和apiInfo对应。所以，我们可以自行更改。

```

1 package com.kuang.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import springfox.documentation.service.ApiInfo;
6 import springfox.documentation.service.Contact;
7 import springfox.documentation.service.VendorExtension;
8 import springfox.documentation.spi.DocumentationType;
9 import springfox.documentation.spring.web.plugins.Docket;
10 import springfox.documentation.swagger2.annotations.EnableSwagger2;
11
12 import java.util.ArrayList;
13
14 @Configuration //配置到Spring中
15 @EnableSwagger2 //开启Swagger2
16 public class SwaggerConfig {
17     //配置了Swagger的Docket的Bean实例
18     @Bean
19     public Docket docket() {
20         return new Docket(DocumentationType.SWAGGER_2)
21             .apiInfo(apiInfo());
22     }
23
24     //配置Swagger信息===》apiInfo
25     private ApiInfo apiInfo() {
26
27         return new ApiInfo(
28             "Geekst的Swagger API文档", //标题
29             "Swagger Demo", //描述
30             "1.0", //版本号
31             "https://blog.csdn.net/Geekst",
32             new Contact("Geekst", "https://blog.csdn.net/Geekst",
33                 "1084987683@qq.com"), //作者信息
34             "Apache 2.0",
35             "http://www.apache.org/licenses/LICENSE-2.0",
36             new ArrayList<VendorExtension>());

```

# Geekst的Swagger API文档<sup>1.0</sup>

[ Base URL: localhost:8080/ ]  
<http://localhost:8080/v2/api-docs>

Swagger Demo

[Terms of service](#)

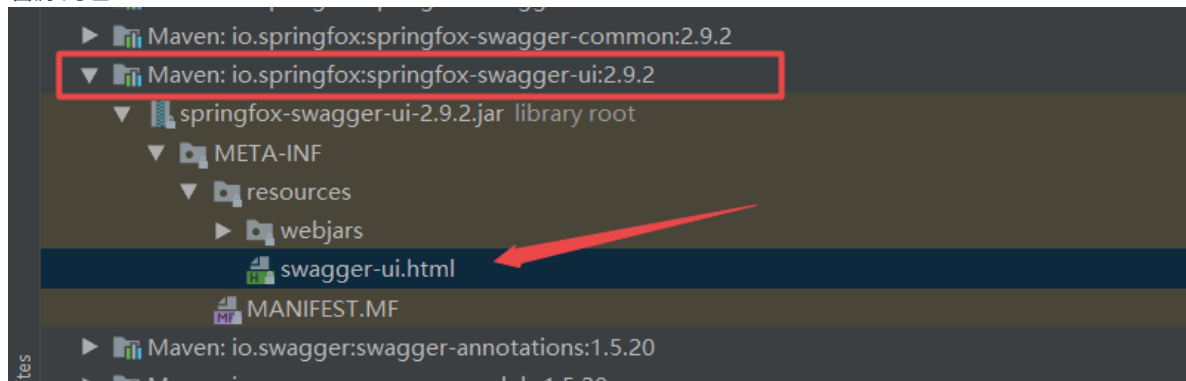
[Geekst - Website](#)

[Send email to Geekst](#)

[Apache 2.0](#)

好奇这个swagger-ui.html界面怎么来的?

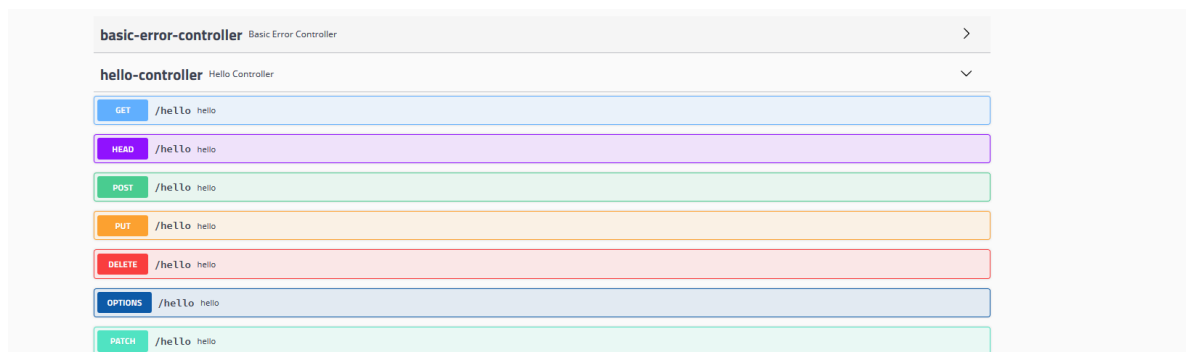
看源码包:



在resources, 直接进行访问。

## 4.2 自定义配置扫描接口

未配置的时候:



扫描了所有。

```

1 //配置了Swagger的Docket的Bean实例
2 @Bean
3 public Docket docket() {
4     return new Docket(DocumentationType.SWAGGER_2)
5         .apiInfo(apiInfo())
6         .select()
7         /*
8         RequestHandlerSelectors 配置要扫描接口的方式
9         basePackage(): 指定要扫描的包
10        any(): 扫描全部
11        none(): 都不扫描
12        withClassAnnotation(): 扫描类上的注解, 参数是一个注解的反射对象

```

```

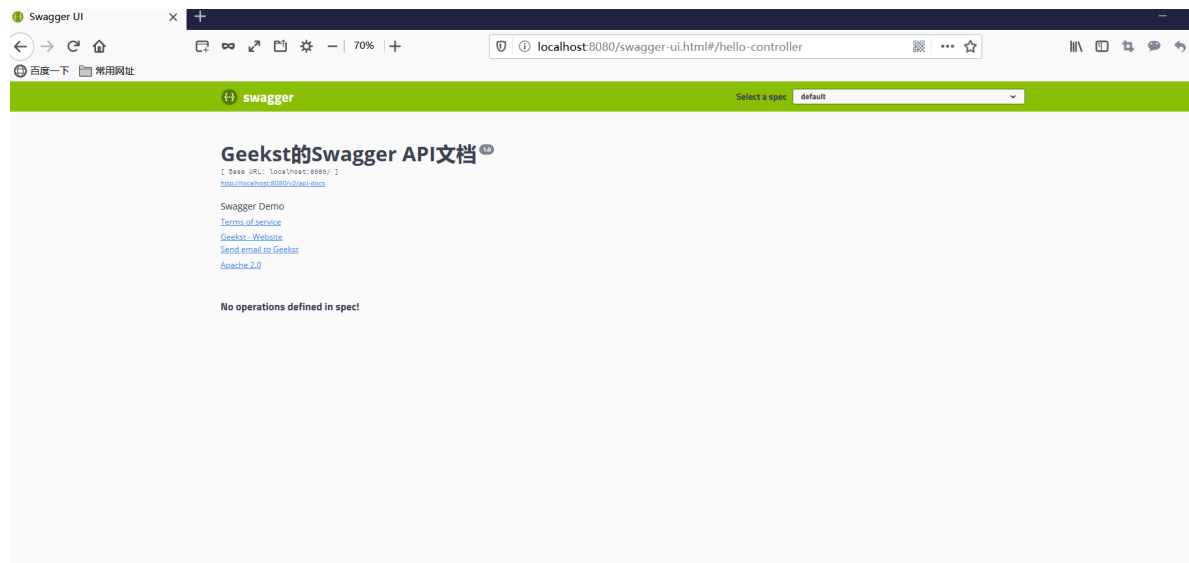
13         withMethodAnnotation(): 扫描方法上的注解
14         */
15
16     .apis(RequestHandlerSelectors.basePackage("com.kuang.swagger.controller"))
17         //指定扫描的位置
18         //ant(): 只扫描该请求下的
19         .paths(PathSelectors.ant("/kuang/**"))
20         .build();

```

刚才配置中加上两个地方：

- 配置要扫描接口的方式
- 指定扫描的位置

由于我们没有/kuang/\*\*下的请求，所以没有扫描：



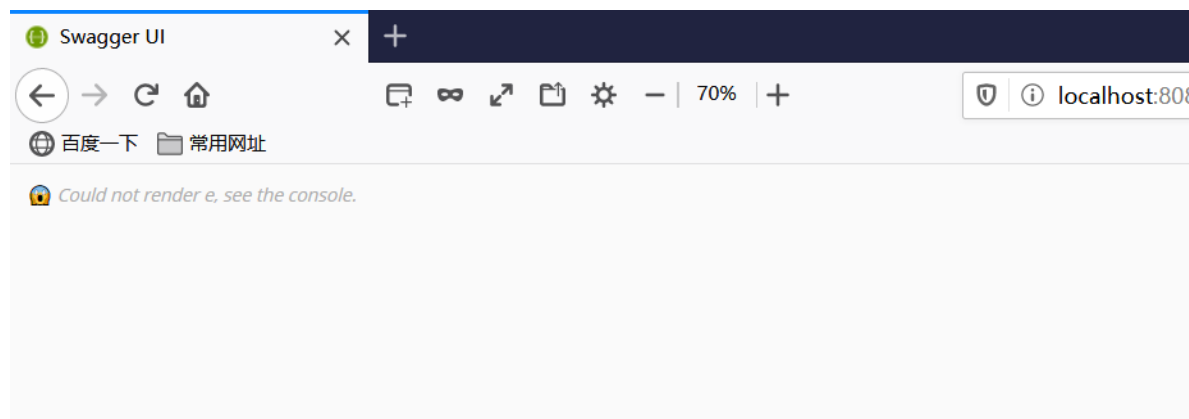
### 4.3 配置是否启动Swagger

一样在Docket下：

```
1 | .enable(true)
```

默认为true===》启动

关掉了会这样：



### 4.4 多环境配置

案例：只希望在生产环境中使用Swagger，项目上线后不使用。

- 首先, 判断是不是生产环境 flag
- 注入enable(flag)

来两个环境: dev【生产环境】、pro【上线后的环境】

application-dev.properties:

```
1 | server.port=8080
```

application-pro.properties:

```
1 | server.port=8081
```

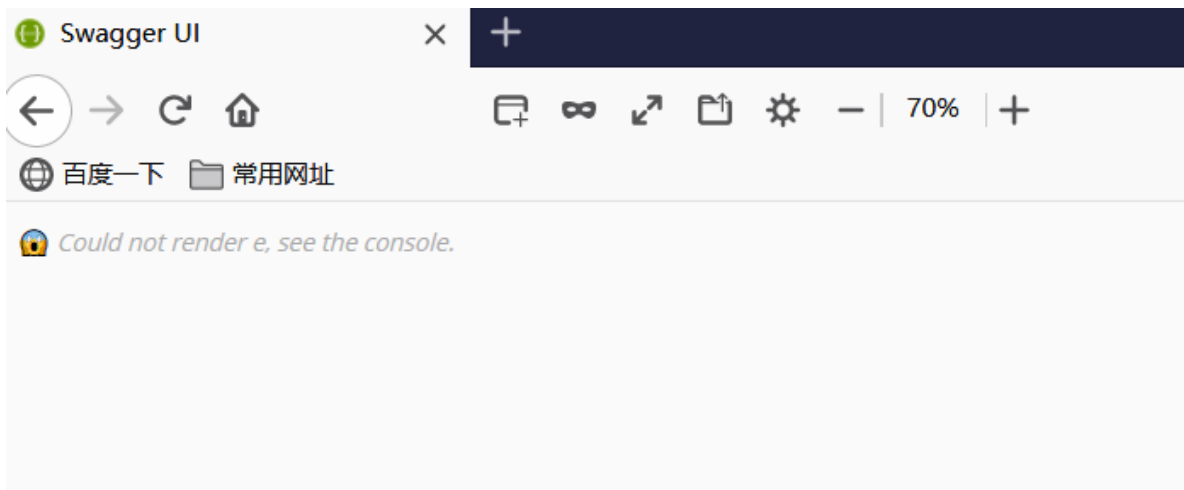
application.properties:

```
1 | spring.profiles.active=pro
```

这里启动上线环境。

```
1  @Bean
2  public Docket docket(Environment environment) {
3      //设置要显示的Swagger环境【假设这里去取生产环境dev】
4      Profiles profiles = Profiles.of("dev","test");
5
6      //通过environment.acceptsProfiles来判断是否处于自己设置的环境中【生产环境】
7      boolean flag = environment.acceptsProfiles(profiles);
8
9      //获取项目环境
10     environment.getActiveProfiles();
11
12     return new Docket(DocumentationType.SWAGGER_2)
13         .apiInfo(apiInfo())
14         .enable(flag)//是否启动, 默认为true【启动】
15         .select()
16         /*
17         RequestHandlerSelectors 配置要扫描接口的方式
18         basePackage(): 指定要扫描的包
19         any(): 扫描全部
20         none(): 都不扫描
21         withClassAnnotation(): 扫描类上的注解, 参数是一个注解的反射对象
22         withMethodAnnotation(): 扫描方法上的注解
23         */
24
25     .apis(RequestHandlerSelectors.basePackage("com.kuang.swagger.controller"))
26         //指定扫描的位置
27         //ant(): 只扫描该请求下的
28         .paths(PathSelectors.ant("/kuang/**"))
29         .build();
30 }
```

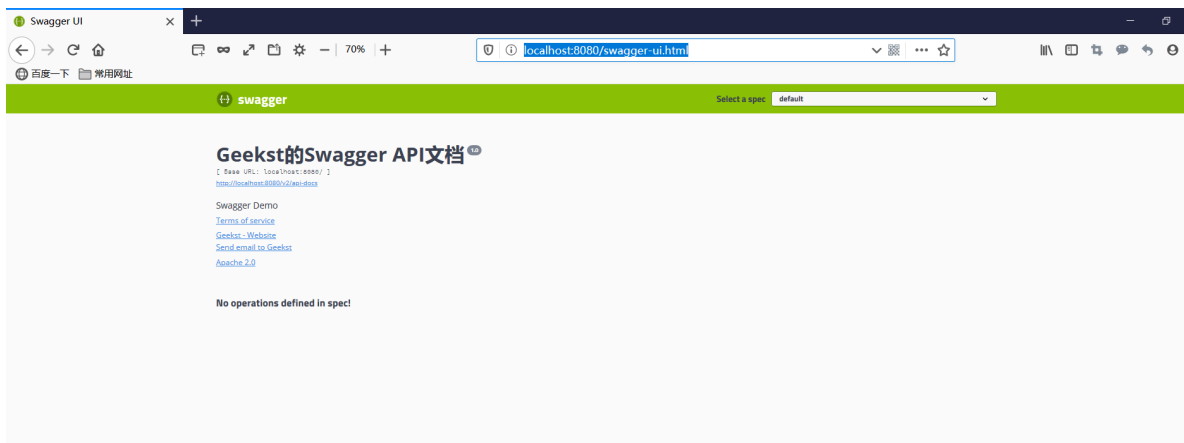
测试一下: <http://localhost:8081/swagger-ui.html>



我们切换环境：【只需要改配置文件】

```
1 | spring.profiles.active=dev
```

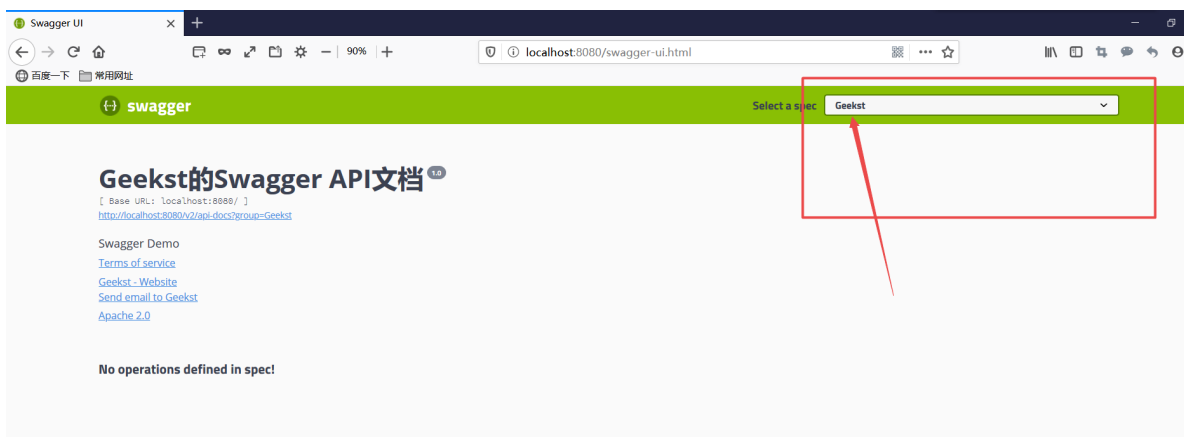
测试: <http://localhost:8080/swagger-ui.html>



## 4.5 配置API文档的分组

配置groupName。

```
1 | .groupName("Geekst")
```



## 4.6 配置多个Docket的实例，多个分组

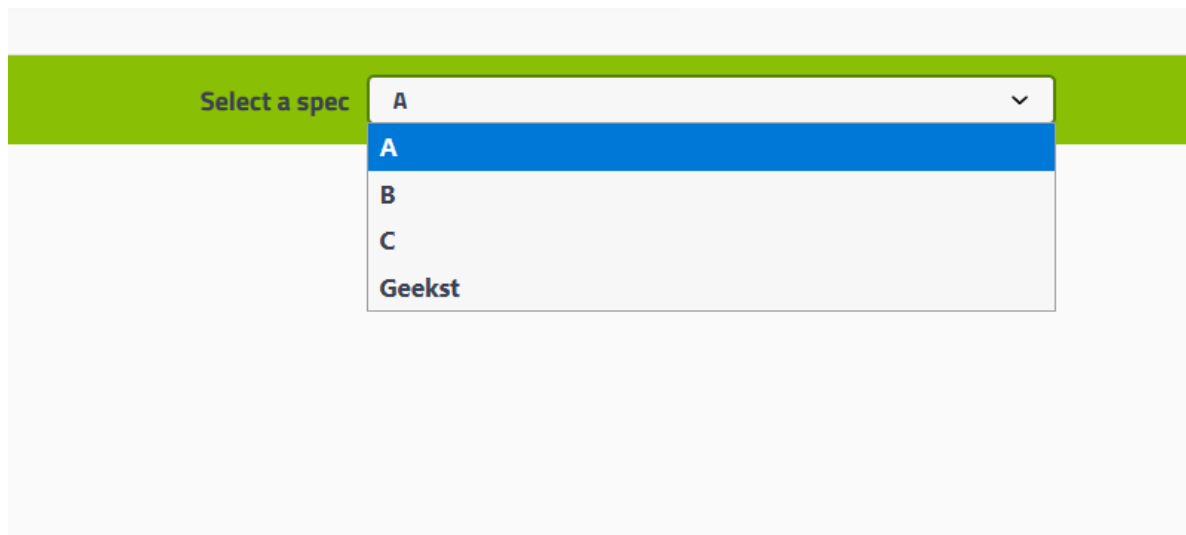
```
1 | //配置了Swagger的Docket的Bean实例
2 | @Bean
3 | public Docket docket(Environment environment) {
```



```

4      //设置要显示的Swagger环境【假设这里去取生产环境dev】
5      Profiles profiles = Profiles.of("dev", "test");
6
7      //通过environment.acceptsProfiles来判断是否处于自己设置的环境中【生产环境】
8      boolean flag = environment.acceptsProfiles(profiles);
9
10     //获取项目环境
11     environment.getActiveProfiles();
12
13     return new Docket(DocumentationType.SWAGGER_2)
14         .apiInfo(apiInfo())
15         .enable(flag)//是否启动，默认为true【启动】
16         .groupName("Geekst")
17         .select()
18         /*
19         RequestHandlerSelectors 配置要扫描接口的方式
20         basePackage(): 指定要扫描的包
21         any(): 扫描全部
22         none(): 都不扫描
23         withClassAnnotation(): 扫描类上的注解，参数是一个注解的反射对象
24         withMethodAnnotation(): 扫描方法上的注解
25         */
26
27     .apis(RequestHandlerSelectors.basePackage("com.kuang.swagger.controller"))
28     //指定扫描的位置
29     //ant(): 只扫描该请求下的
30     .paths(PathSelectors.ant("/kuang/**"))
31     .build();
32
33     @Bean
34     public Docket docket1() {
35         return new Docket(DocumentationType.SWAGGER_2)
36             .groupName("A");
37     }
38
39     @Bean
40     public Docket docket2() {
41         return new Docket(DocumentationType.SWAGGER_2)
42             .groupName("B");
43     }
44
45     @Bean
46     public Docket docket3() {
47         return new Docket(DocumentationType.SWAGGER_2)
48             .groupName("C");
49     }

```



- 用处
  - 不同的分组分别进行不同的开发需求
  - 协同开发

## 4.7 接口注释

实体类配置：

```
1 @ApiModelProperty("用户实体类")
2 public class User implements Serializable {
3     @ApiModelProperty("用户名")
4     public String username;
5     @ApiModelProperty("密码")
6     public String password;
7 }
```

控制层中的配置：

```
1 @Api(tags = "Hello的控制层")
2 @RestController
3 public class HelloController {
4     @GetMapping("/hello")
5     public String hello() {
6         return "hello";
7     }
8
9
10    //只要在我们的接口中，返回值中存在实体类，它就会被扫描到Swagger中
11    @PostMapping(value = "/user")
12    public User user() {
13        return new User();
14    }
15
16    @ApiOperation("hello有username")
17    @GetMapping("/hello2")
18    public String hello(@ApiParam("用户名") String username) {
19        return "hello" + username;
20    }
21 }
```

Apache 2.0

Hello的控制层

Hello Controller

GET

/hello

hello

GET

/hello2

hello有username

POST

/user

user

Models

GET

/hello2

hello有username

Parameters

Try it out

Name

username

(body)

Description

用户名

Example Value / Model

"string"

Parameter content type

application/json

Responses

Response content type \*/\*

Models

用户实体类

password

string

密码

username

string

用户名

}

## 5 接口测试的使用

这里拿我们的postt方法进行测试：

POST

/postt

Post测试类

Parameters

Try it out

Name

Description

password

string

(query)

密码

username

string

(query)

用户名

Responses

Response content type \*/\*

Curl

curl -X POST "http://localhost:8080/postt?password=123456&username=lili" -H "accept: \*/\*"

点击 Try it out进行测试：

Parameters

Cancel

Name

Description

password

string

(query)

密码

password - 密码

username

string

(query)

用户名

username - 用户名

Execute

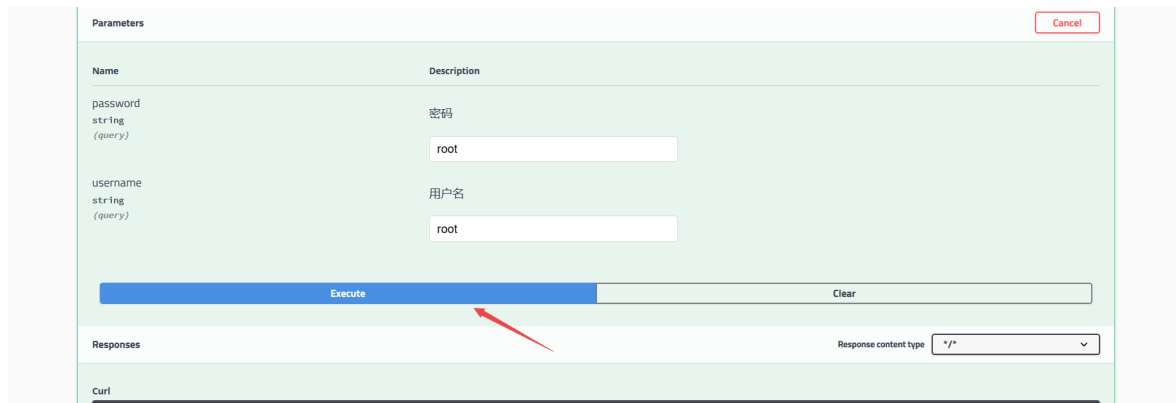
Clear

Responses

Response content type \*/\*

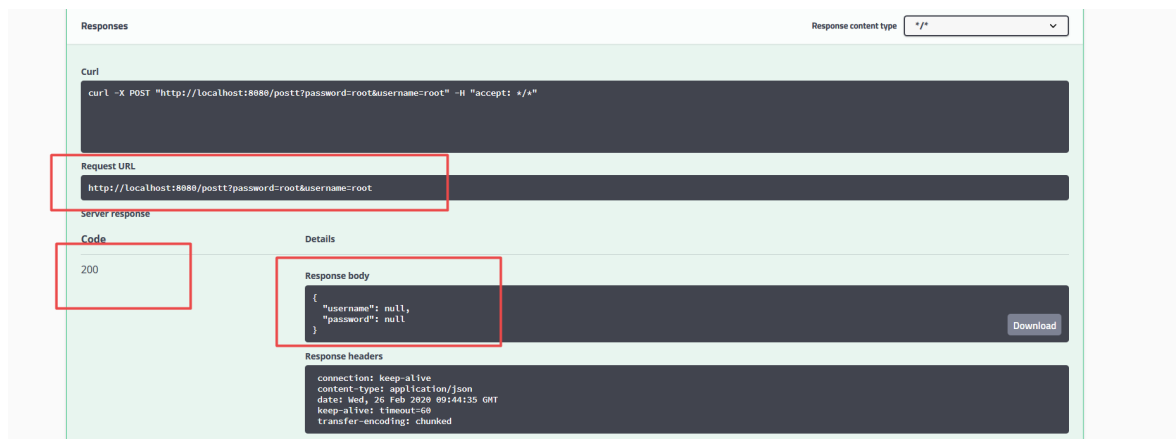
这些字段都是我们方法中有的，这里自动生成。

我们测试一个：



The image shows the 'Parameters' section of a Swagger UI interface. It contains a table with two columns: 'Name' and 'Description'. The first row is for 'password', a string query parameter with the description '密码' (password), and a text input field containing 'root'. The second row is for 'username', a string query parameter with the description '用户名' (username), and a text input field containing 'root'. Below the table are two buttons: 'Execute' (highlighted in blue) and 'Clear'. A red arrow points to the 'Execute' button. At the bottom, there is a 'Responses' section with a 'Response content type' dropdown set to '\*/\*' and a 'Curl' section.

输入需要测试的数据之后的，点击Excute进行测试：



The image shows the 'Responses' section of the Swagger UI interface. It displays the 'Curl' command: `curl -X POST "http://localhost:8888/postt?password=root&username=root" -H "accept: */*"`. Below this, the 'Request URL' is shown as `http://localhost:8888/postt?password=root&username=root`. The 'Server response' section shows a 'Code' of 200. The 'Details' section shows the 'Response body' as `{ "username": null, "password": null }` and the 'Response headers' as `connection: keep-alive, content-type: application/json, date: Wed, 26 Feb 2020 09:44:35 GMT, keep-alive: timeout=60, transfer-encoding: chunked`. Red boxes highlight the 'Request URL', the 'Code' 200, and the 'Response body'.

会生成URL，执行代码（200、404之类的）。

基本使用步骤：选择要进行测试的接口方法——>Try it out ——>输入要测试的值——>Excute——>观察返回的结果

## 6 总结

- 给一些比较难理解的属性或者接口增加注释信息（见4.7）
- 接口文档实时更新
- 可以在线测试接口
- 符合迭代开发的需求

【注意：出于安全考虑，在项目正式上线的时候，必须禁用Swagger，不然会有暴露接口的风险！而且还能节省运行的内存】