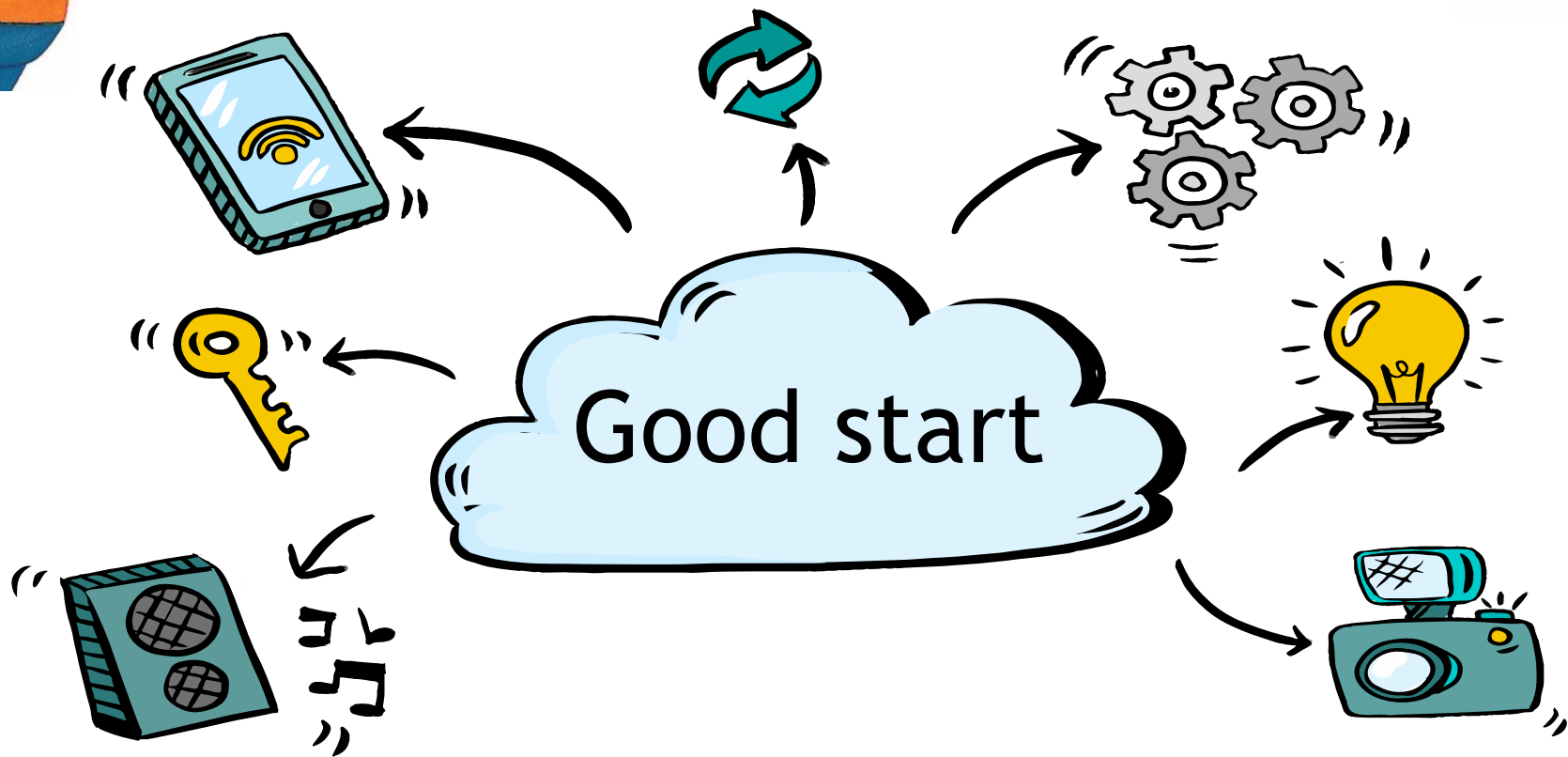


# 郑晖

---



- ✓ 大四学生，即将成为后端开发队伍的打工人
- ✓ truedei-swagger-plugin框架作者
- ✓ CSDN博客专家
- ✓ 腾讯云社区优秀作者
- ✓ 对Linux比较熟悉，高中就参加Linux比赛拿到了全国的金牌
- ✓ 大学里参加了无数计算机类比赛，拿到的证书可以按斤算
- ✓ 现在在某一线公司实习



**前后端分离模式下如何保证开发人员不打架？**

# 说在前面

01

你将会了解前后端分离模式







02

你将会了解到一些有趣的事情

03

你将会了解truedei-swagger-plugin框架

# CONTENTS

-  **01 传统的web开发模式VS前后端分离开发模式**
-  **02 truedei-swagger-plugin是什么框架，为什么能解决现状呢？**
-  **03 swagger是什么？和Springfox-swagger有什么关系？**
-  **04 为什么要用Springfox-swagger？**
-  **05 如何结合SpringBoot项目进行应用**
-  **06 总结**

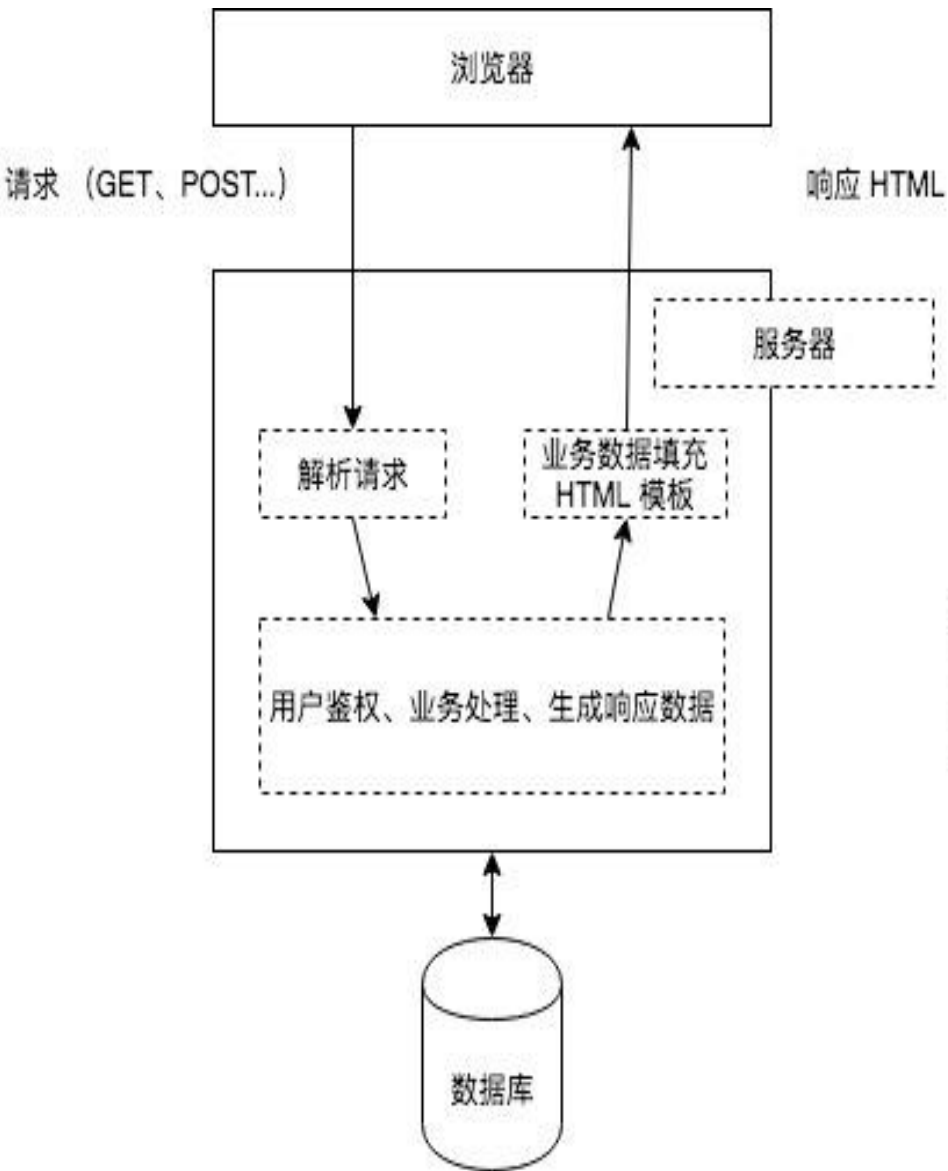
# CONTENTS

- 01 传统的web开发模式VS前后端分离开发模式
- 02 truedei-swagger-plugin是什么框架，为什么能解决现状呢？
- 03 swagger是什么？和Springfox-swagger有什么关系？
- 04 为什么要用Springfox-swagger？
- 05 如何结合SpringBoot项目进行应用
- 06 总结



# 传统的web开发模式 VS 前后端分离开发模式

## 传统的web开发模式



- 存在很大的问题
- 有什么问题呢?
- 问题点在图右边那位全栈（栈）开发人员!
- 开发人员不仅要会服务器端开发（Java），又要会前端开发（HTML、CSS、JS）。还要懂数据库开发（SQL）；



样样通，不如一样精；

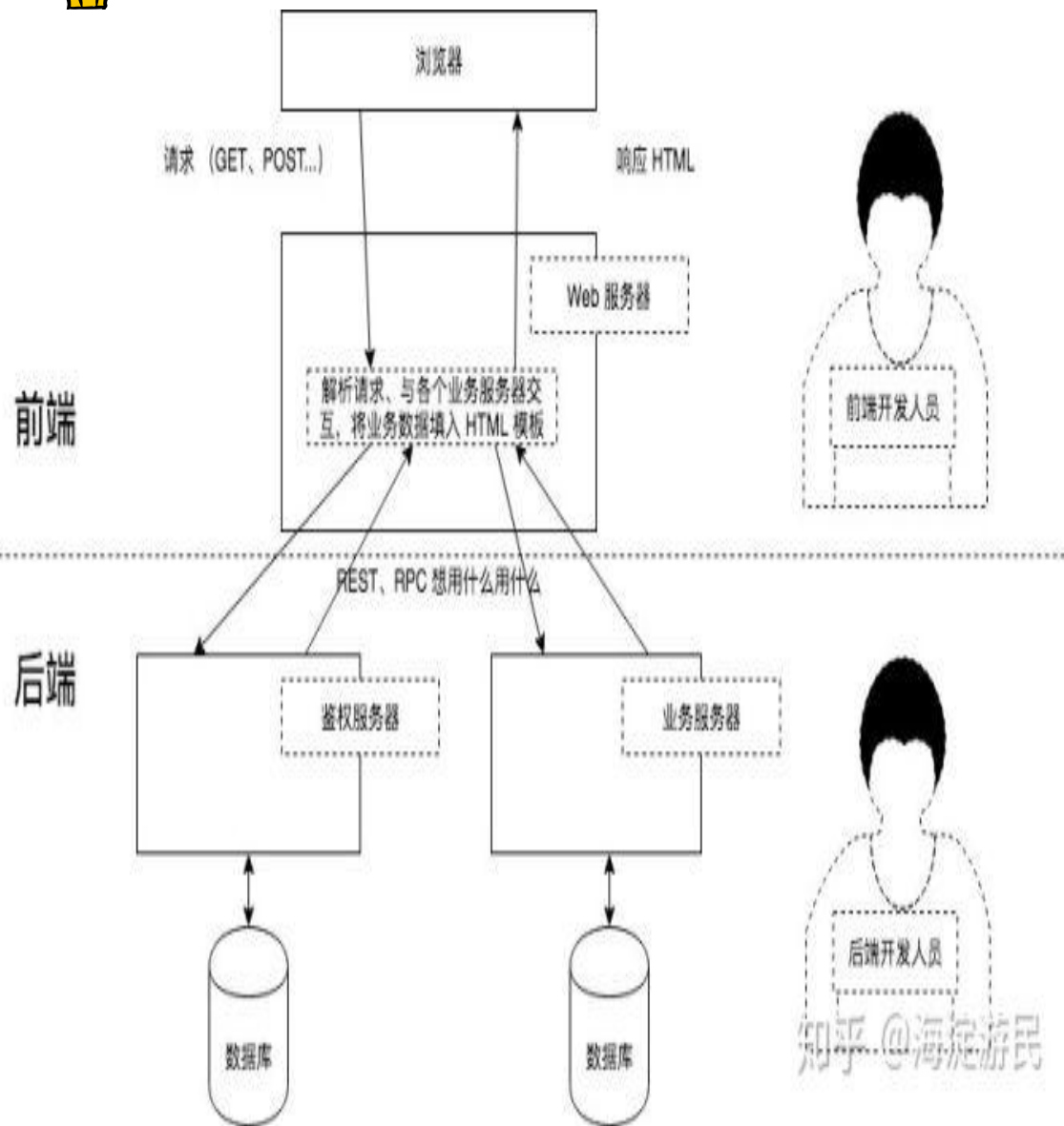
什么都懂，往往什么都不懂。

老子 - 《道德经》





# 传统的web开发模式 VS 前后端分离开发模式



## 前后端分离模式

- 事先约定好接口;



- 1、前端人员只需要把界面做好就可以了;
- 2、后端人员只需要做好业务逻辑处理即可。



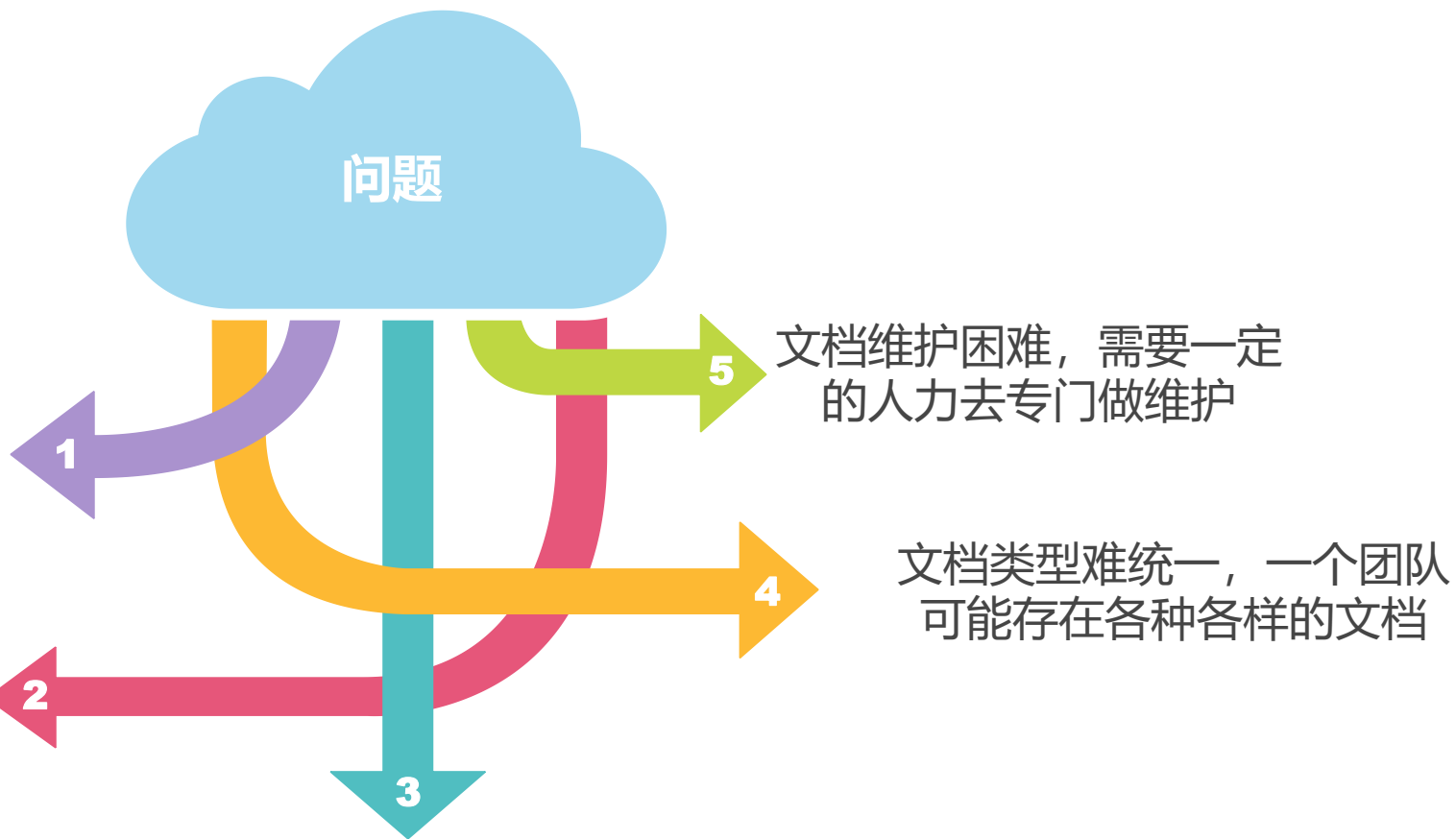


# 前后端分离带来的问题



前后端集成联调，前端人员和后端人员无法做到，及时协商。

经常是改变了code，文档不能及时更新



工欲善其事；

必先利其器。

孔子 - 《论语·魏灵公》

# CONTENTS

- 01 传统的web开发模式VS前后端分离开发模式
- 02 **truedei-swagger-plugin是什么框架，为什么能解决现状呢？**
- 03 swagger是什么？和Springfox-swagger有什么关系？
- 04 为什么要用Springfox-swagger？
- 05 如何结合SpringBoot项目进行应用
- 06 总结



# truedei-swagger-plugin是什么?

## 1、是一个开源的框架

是一个对swagger二次开发，扩展了多种功能的框架，简单，方便，快捷，高效。

## 2、是多种api接口管理的解决方案

解决了多种swagger在真实当中应用的问题。

## 3、有效的解决了前后端、测试人员交流的问题

提供了在线调试、在线阅读api说明，及时更新等功能。



# CONTENTS

- 01 传统的web开发模式VS前后端分离开发模式
- 02 truedei-swagger-plugin是什么框架，为什么能解决现状呢？
- 03 **swagger是什么？和Springfox-swagger有什么关系？**
- 04 为什么要用Springfox-swagger？
- 05 如何结合SpringBoot项目进行应用
- 06 总结



# Swagger和Springfox-swagger的关系是什么

## 1、swagger是什么？

swagger有5个项目，有Swagger Codegen、Swagger UI、Swagger Editor、Swagger Inspector、Swagger Hub。更准确的说，swagger是一套规范。

## 2、Springfox-swagger是什么？

Springfox-swagger 是基于swagger规范和Spring的一个框架；

可以将基于Spring或Spring Boot项目进行融合，扫描SpringMVC或者SpringBoot项目的代码，自动生成JSON格式的数据；

**Springfox-swagger本身不是属于Swagger官方提供的。**

## 3、swagger和Springfox-swagger有什么区别？

Swagger 是一套规范；

Springfox-swagger 是基于Swagger规范和Spring的一个框架；

# CONTENTS

- 01 传统的web开发模式VS前后端分离开发模式
- 02 truedei-swagger-plugin是什么框架，为什么能解决现状呢？
- 03 swagger是什么？和Springfox-swagger有什么关系？
- 04 为什么要用Springfox-swagger？
- 05 如何结合SpringBoot项目进行应用
- 06 总结



# 为什么要用springfox-swagger



1、可以和Spring项目或者SpringBoot项目很好的整合；



2、可以自动扫描并生成项目的所有的接口，并展示在UI界面，无需手动特意编写API文档；



3、通过web方式查看每个接口的作用，以及描述



4、可以查看参数的是什么类型，是否必须输入等信息；



5、可以像postman一样测试API接口



6、可以动态的查看接口请求的状态码和返回的数据结构；



# CONTENTS

- 01 传统的web开发模式VS前后端分离开发模式
- 02 truedei-swagger-plugin是什么框架，为什么能解决现状呢？
- 03 swagger是什么？和Springfox-swagger有什么关系？
- 04 为什么要用Springfox-swagger？
- 05 如何结合SpringBoot项目进行应用
- 06 总结



# 如何结合SpringBoot项目进行应用





## Step1:引入Swagger需要的依赖

```
<!--引入依赖，此依赖包是对Swagger的一个扩展-->  
<dependency>  
  <groupId>com.truedei.swagger.plugin</groupId>  
  <artifactId>truedei-swagger-plugin</artifactId>  
  <version>0.0.1-SNAPSHOT</version>  
</dependency>
```

```
▼ com.truedei.swagger.plugin:truedei-swagger-plugin:0.0.1-SNAPSHOT  
  commons-io:commons-io:2.6  
  ▶ com.vladsch.flexmark:flexmark-all:0.50.42  
  com.youbenzi:MDTool:1.2.4  
  ▶ io.springfox:springfox-swagger2:2.9.2  
  ▶ io.swagger:swagger-models:1.5.21  
  ▶ com.github.xiaoymin:knife4j-spring-boot-starter:2.0.4  
  org.aspectj:aspectjrt:1.9.6  
  ▶ cglib:cglib:2.2.2  
  org.apache.commons:commons-lang3:3.11  
  org.springframework.boot:spring-boot-starter-web:2.4.0 (omitted for duplicate)  
  ▶ org.springframework.boot:spring-boot-starter-freemarker:2.4.0  
  ▶ org.apache.logging.log4j:log4j-to-slf4j:2.13.3
```

需要的依赖，已全部集成到了truedei-swagger-plugin中。  
无需关心其他的依赖。





# Springfox-Swagger2常用注解总结

注解	作用对象	说明
@Api()	类	表示标识这个类是swagger的资源
@ApiOperation()	方法	表示一个http请求的操作
@ApiParam()	方法	参数, 字段说明, 表示对参数的添加元数据 (说明或是否必填等)
@ApiModelProperty()	类	表示对类进行说明, 用于参数用实体类接收
@ApiModelPropertyProperty()	方法, 字段	表示对model属性的说明或者数据操作更改
@ApiIgnore()	类, 方法, 方法参数	表示这个方法或者类被忽略
@ApiImplicitParam()	方法	表示单独的请求参数
@ApiImplicitParams()	方法	包含多个 @ApiImplicitParam



## Step2:开启Swagger相关功能

需要用到的注解:

```
@Configuration //Spring的注解将该类注入到Bean中
@EnableSwagger2 //开启Springfox-swagger2的功能
@EnableKnife4j //开启使用第三方UI
@EnableSwaggerPlugin //开启自定义扩展的功能
public class Swagger2Config { .... }
```

注解解释:

**@Configuration**: Spring的注解将该类注入到Bean中  
**@EnableSwagger2**: 开启Springfox-swagger2的功能  
**@EnableKnife4j**: 开启使用第三方UI  
**@EnableSwaggerPlugin**: 开启自定义扩展的功能  
**@EnableApiFileURI**: 开启从配置文件中读取数据的功能

这些注解是必须的哦!



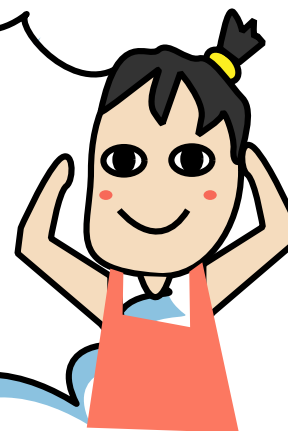


## Step3:配置Swagger的基本信息

....省略注解

```
public class Swagger2Config {  
  
    @Bean  
    public Docket appApi() {  
        return new Docket(DocumentationType.SWAGGER_2)  
            // 不适用swagger自带的response的消息 (404,500,200等)  
            .useDefaultResponseMessages(false)  
            .groupName("API仓库") //分组名称  
            .genericModelSubstitutes(ResponseEntity.class)  
            .apiInfo(apiInfo()) //swagger基本信息  
            .select()  
            //第一种: 扫描指定的包  
            // .apis(RequestHandlerSelectors.basePackage("com.glodon.demo.mybatis"))  
            //第二种: 扫描只包含Swagger的注解, 这种方式灵活  
            .apis(RequestHandlerSelectors.withMethodAnnotation(ApiOperation.cl  
ass))  
            .paths(PathSelectors.any())  
            .build();  
    }  
}
```

更详细的配置可以  
去查看swagger的  
配置!





## Step3:配置Swagger的基本信息

```
/**
 * 配置Swagger信息
 * @return
 */
private ApiInfo apiInfo() {
    return new ApiInfoBuilder()
        .title("API仓库接口文档")
        .description("这是描述信息")
        .contact(new springfox
            .documentation
            .service
            .Contact(
                "我们是TrueDei团队",
                "https://www.truedei.com/",
                "8042965@qq.com"))
        .version("0.0.1")
        .build();
}
```

Swagger api文档的  
标题等配置





## Step4:使用注解描述特定的类，使被Swagger扫描到

需要用到的注解:

```
@Controller
@RequestMapping("/test")
@Api(value = "1.test接口",tags = "1.test接口",position = 1)
public class TestController { ... }
```

注解解释:

**@Api()**: 描述当前类是一个Controller层的类，加上之后，项目在启动时，此接口就可以被Swagger自动扫描到

包含参数:

**value**、**tags**: 都是用来描述当前类的，相当于显示的别名

**position**: 排序使用的序号

原来@Api是用来描述Controller层的类的。







## Step4:使用注解描述特定的接口

需要用到的注解:

```
@ResponseBody
@RequestMapping(value = "/one", method = RequestMethod.POST)
@ApiOperation(position = 1,value = "1.one",notes = "one" )
@APIFileInfo("/test/one")
public Object testOne(    @Apicp(
    modelName = "testOne",//新生成的对象的名字
    noValues = {"name","address"},//不存在的属性的名字
    noValueTypes = {"string","string"},//不存在的属性对应的类型
    noVlaueExplains = {"名字","地址"},//不存在的属性对应的说明
    noVlaueExample= {"郑晖","北京市昌平区"},//参数示例
    noVlaueRequired= {true,flase},//是否必填
) @RequestBody String str) { .... }
```

注解解释:

**@ApiOperation:** 描述一个接口, 此接口就可以被Swagger自动扫描到

**@APIFileInfo:** 描述当前接口的一些信息都来自文件;

**@Apicp:** 描述这个参数需要生成新的类

哇塞, 原来@Apicp是用来描述接口参数, 来生成实体类的。





# Step4:使用了@ApiFileInfo注解后，然后编写特定的文件信息

The screenshot displays the IntelliJ IDEA IDE with two files open: `TestController.java` and `TestController.md`.

**TestController.java:**

```
11 import org.springframework.web.bind.annotation.*;
12 @Controller
13 @RequestMapping("/test")
14 @Api(value = "1.test接口", tags = "1.test接口", position = 1)
15 public class TestController {
16
17     @PostMapping("/one")
18     @ResponseBody
19     @ApiOperation(value = "one", notes = "one", position = 1)
20     @ApiFileInfo("/test/one")
21     public String testOne(@ApiParam(
22         modelName = "tetsOne",
23         noValues = {"name", "address"},
24         noValueTypes = {"string", "string"},
25         noVlaveExplains = {"名字", "地址"},
26         noVlaveExample = {"郑晖"},
27         noVlaveRequired = {true}
28     )
29     @RequestBody String str){
30         return "one";
31     }
32
33     @GetMapping("/tow")
34     @ResponseBody
35     @ApiOperation(value = "tow", notes = "tow", position = 2)
36     public String testTow(@ApiParam(
37         modelName = "Refund",
38         noValues = {"orderId", "images", "remark", "addressId"},
39         noValueTypes = {"Long", "String", "String", "Long"},
40         noVlaveExplains = {"订单ID", "图片(列表)最多8个", "退款退货原因", "收货地址ID"},
41         noVlaveExample = {"1", "2", "不要了", "1"}
42     ) @RequestBody String str){
43         return "tow";
44     }
45 }
```

**TestController.md:**

```
1 # URL:/test/one
2
3 这是接口描述
4
5 ---
6
7 code:200
8
9 **请求成功返回的数据**
10
11 ```json
12 {
13     "code": 100,
14     "message": "",
15     "result": "name='TrueDei'"
16 }
17 ```
18 ---
19
20 code:400
21 **请求失败返回的数据**
22
23 ```json
24 {
25     "code": 400,
26     "message": "访问失败了",
27     "result": ""
28 }
29 ```
30 ---
```



API仓库 1 分组的信息

主页

Swagger Models

文档管理

1.test接口

one

tow

2 扫描的有效接口

API仓库接口文档 4 标题

输入文档关键字搜索

主页

## API仓库接口文档

简介 该文档主要提供知识库后端的接口  
请求服务:http//IP:Port/doc.html

3 基本信息

作者 我们是TrueDei团队

版本 0.0.1

host 127.0.0.1:8080

basePath /

服务Url

分组名称 API仓库

分组Url /v2/api-docs?group=API仓库

分组location /v2/api-docs?group=API仓库

接口统计信息 POST 1

GET 1



# 结果

主页 one X

文档

调试

one 复制文档 复制地址

**POST** /test/one

请求数据类型 ["application/json"]      响应数据类型 ["\*\*/\*"]

**接口描述**

这是接口描述

**请求示例**

```
1 {
2   "address": "",
3   "name": "郑晖"
4 }
```

**请求参数**

参数名称	参数说明	请求类型	是否必须	数据类型	schema
- tetsOne	str	body	true	tetsOne	tetsOne
address	地址		false	string	
name	名字		true	string	

**响应状态**



# 结果

页

one X

name

名字

true

string

## 响应状态

状态码

说明

schema

200

### 请求成功返回的数据

▼ 点击展开查看

```
{
  "code": 100,
  "message": "",
  "result": "name='TrueDei'"
}
```

400

### 请求失败返回的数据

▼ 点击展开查看

```
{
  "code": 400,
  "message": "访问失败了",
  "result": ""
}
```



# 结果

主页

one X

文档

调试

POST

/test/one

发送

请求头部

请求参数

x-www-form-urlencoded  form-data  raw JSON(application/json) v

```
1 {  
2   "address": "",  
3   "name": "郑晖"  
4 }
```

响应内容

Raw

Headers

Curl

显示说明 响应码: 200 耗时: 123ms 大小: 3 b

```
1 one
```

# CONTENTS

- 01 传统的web开发模式VS前后端分离开发模式
- 02 truedei-swagger-plugin是什么框架，为什么能解决现状呢？
- 03 swagger是什么？和Springfox-swagger有什么关系？
- 04 为什么要用Springfox-swagger？
- 05 如何结合SpringBoot项目进行应用
- 06 **总结**



# 总结与回顾

## 问题1：后端接收json字符串参数时，无法在界面显示的解决办法

```
...  
@EnableSwaggerPlugin //开启自定义扩展的功能  
public class Swagger2Config {  
...  
}
```

开启自定义注解的扩展功能

```
94  
95  
96  
97 @ResponseBody  
98 @RequestMapping(value = "/editRobot", method = RequestMethod.POST)  
99 @ApiOperation(position = 3, value = "15.3.编辑机器人",  
100 notes = "<h1><strong>编辑机器人</strong></h1>\n" +  
101 "<p>//说明: robotName和categoryIds二者不能都为空</p>\n")  
102 public Object editRobot( @ApiParam(  
103 classPath=RobotConfig.class, //指定原始类  
104 values = {"robotId","robotName"}, //指定原始类中已经存在的属性  
105 noValues = {"categoryIds"}, //指定新增的属性  
106 noValueType = {"string"}, //指定新增属性的类型, (忽略大小写)  
107 noValueExplains = {"所选分类, 例如: 2,3,4,5,6"}, //指定新增属性的描述  
108 modelName = "EditRobotModel") //指定新生成的类的名字  
109 @RequestBody String str) {  
110  
111 String userId = getOperatorId(request);  
112 if(userId == null || "".equals(userId)){  
113 return BaseResponseMsg.getDefaultTokenError();  
114 }  
115  
116 Map map = JSONObject.parseObject(str, Map.class);  
117  
118 String robotIdStr;  
119 if(map == null || map.get("robotId") == null || "".equals(robotIdStr) ||  
120 map.get("robotId").equals(""))  
121 return BaseResponseMsg.getConstructMsg(BaseResponseMsg.RESPONSE_CODE  
122  
123 try {  
124 long robotId = Long.valueOf(robotIdStr);  
125 RobotConfig robotConfig = robotConfigMsg.selectByRobotId(robotId);  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500
```

知识库接口文档

输入文档关键字搜索

15.3.编辑机器人

POST /robotConfig/editRobot

请求数据类型 [application/json] 响应数据类型 [Map]

### 编辑机器人

//说明: robotName和categoryIds二者不能都为空

请求示例

```
1 {  
2   "categoryIds": "",  
3   "robotId": 0,  
4   "robotName": ""  
5 }
```

请求参数

参数名称	参数说明	请求类型	是否必须	数据类型	schema
EditRobotModel	str	body	true	EditRobotModel	EditRobotModel
categoryIds	所选分类, 例如: 2,3,4,5,6		false	string	
robotId			false	integer(int64)	
robotName			false	string	

响应状态

1 可以看到请求的示例和请求的参数都有了





# 总结与回顾

## 问题1：后端接收json字符串参数时，无法在界面显示解决办法

### 解决之前

```
public Object newRobot(@RequestBody String str) {}
```

请求参数

参数名称
str

响应状态

200

响应参数

参数名称
------

无法得知传递的是什么

### 解决之后

#### 15.2.新增机器人

POST /robotConfig/newRobot

请求数据类型 [application/json] 响应数据类型 [\*/json]

接口描述

新增机器人

请求示例

```
1 {  
2   "pcode": "",  
3   "robotName": "",  
4   "zidingyiIntType": 0,  
5   "zidingyide": ""  
6 }
```

请求参数

参数名称	参数说明	请求类型	是否必须	数据类型
NewRobotModel	str	body	true	NewRobotModel
pcode			false	string
robotName			false	string
zidingyiIntType	自定义2		false	integer(int32)
zidingyide	自定义1		false	string



## 总结与回顾

问题2：接口描述的文本太多，全写在code里侵入性太大，维护不方便的办法

```
return BaseResponseMsg.getDefaultSystemError();
}

@ResponseBody
@RequestMapping(value = "/totalCount", method = RequestMethod.GET)
@ApiOperation(position = 6, value = "15.6.查询机器人总数", notes = "查询机器人总数")
@ApiFileInfo("/robotConfig/totalCount")
public Object totalCount() {
    int totalCount = robotConfigMng.count();
    try {
        Map result = new HashMap<>();
        result.put("totalCount", totalCount);
        return BaseResponseMsg.getConstructSuccessMsg(result);
    } catch (Exception e) {
        logger.error("/robotConfig/totalCount error:{}", e.getMessage());
    }
    return BaseResponseMsg.getDefaultSystemError();
}

---
# URL:/robotConfig/totalCount
---
code:200
```
{
  "messageCode": 200,
  "message": "success",
  "result": {
    "totalCount": 15
  }
}
```
---
code:510
```
json
{
  "messageCode": 510,
  "message": "system error"
}
```
---
```



# 总结与回顾

## 解决之前

问题2：接口描述的文本太多，全写在code里侵入性太大，维护不方便的解决办法

## 解决之后

在code中**减少了99%的无用代码**

```
209 @ResponseBody
210 @RequestMapping(value = "/edit", method = RequestMethod.POST)
211 @ApiOperation(position = 3, value = "9.3. 编辑人工监管配置", notes = "<p><strong>请求的url:</strong></p>\n" +
212     "<p>/api/background/config/robotMonitorConfig/edit</p>\n" +
213     "<p><strong>说明:</strong></p>\n" +
214     "<p>1.authStatus/warnStatus/warnRule/warnKeywords不能都为空</p>\n" +
215     "<p>2. 如果需要删除warnKeywords, 则传空字符或者空数组皆可</p>\n" +
216     "<p><strong>请求body参数示例:</strong></p>\n" +
217     "<details> \n" +
218     "<summary>点击展开查看</summary> \n" +
219     "<pre><code class='\"language-json\">{\n" +
220     "  robotId: 18,\n" +
221     "  authStatus: 1,\n" +
222     "  warnStatus: 1,\n" +
223     "  warnRule: \n" +
224     "    [\n" +
225     "      {\n" +
226     "        &quot;angry&quot;:1\n" +
227     "      },\n" +
228     "      {\n" +
229     "        &quot;unknown&quot;:1\n" +
230     "      },\n" +
231     "      {\n" +
232     "        &quot;down&quot;:1\n" +
233     "      },\n" +
234     "      {\n" +
235     "        &quot;sameAnswer&quot;:1\n" +
236     "      },\n" +
237     "      {\n" +
238     "        &quot;noClick&quot;:1\n" +
239     "      },\n" +
240     "      {\n" +
241     "        &quot;keywords&quot;:1\n" +
242     "      }\n" +
243     "    ],\n" +
244     "    warnKeywords: \n" +
245     "    [\n" +
246     "      &quot;转人工&quot;,\n" +
247     "      &quot;人工回复&quot;\n" +
248     "    ],\n" +
```

```
@ResponseBody
@RequestMapping(value = "/totalCount", method = RequestMethod.GET)
@ApiOperation(position = 6, value = "15.6. 查询机器人总数", notes = "查询机器人总数")
@ApiFileInFo("/robotConfig/totalCount")
public Object totalCount() {
    int totalCount = robotConfigMag.count();
}
```



# 总结与回顾

## 总结

70%

### 使用前

- 1、无法及时更新文档（遗漏）；
- 2、需要手动的编写太多与文档无关的内容；
- 3、文档与代码不在一起，维护困难；



90%

### 使用后

- 1、做到了更改代码后可以及时更新文档；
- 2、无需要手动的编写太多与文档无关的内容，会自动生成页面等；
- 3、文档与代码在一起，维护简单；





# 说在最后

## truedei-swagger-plugin框架



项目地址

<https://github.com/truedei/truedei-swagger-plugin>

说明书

<https://truedei.github.io/#/zh-cn/truedei-swagger-plugin/>

联系qq

8042965

Q交流群

1093169351 (快满了)



## 说在最后

可以关注官方公众号，及时得到最新版本的更新消息



微信搜一搜



TrueDei

