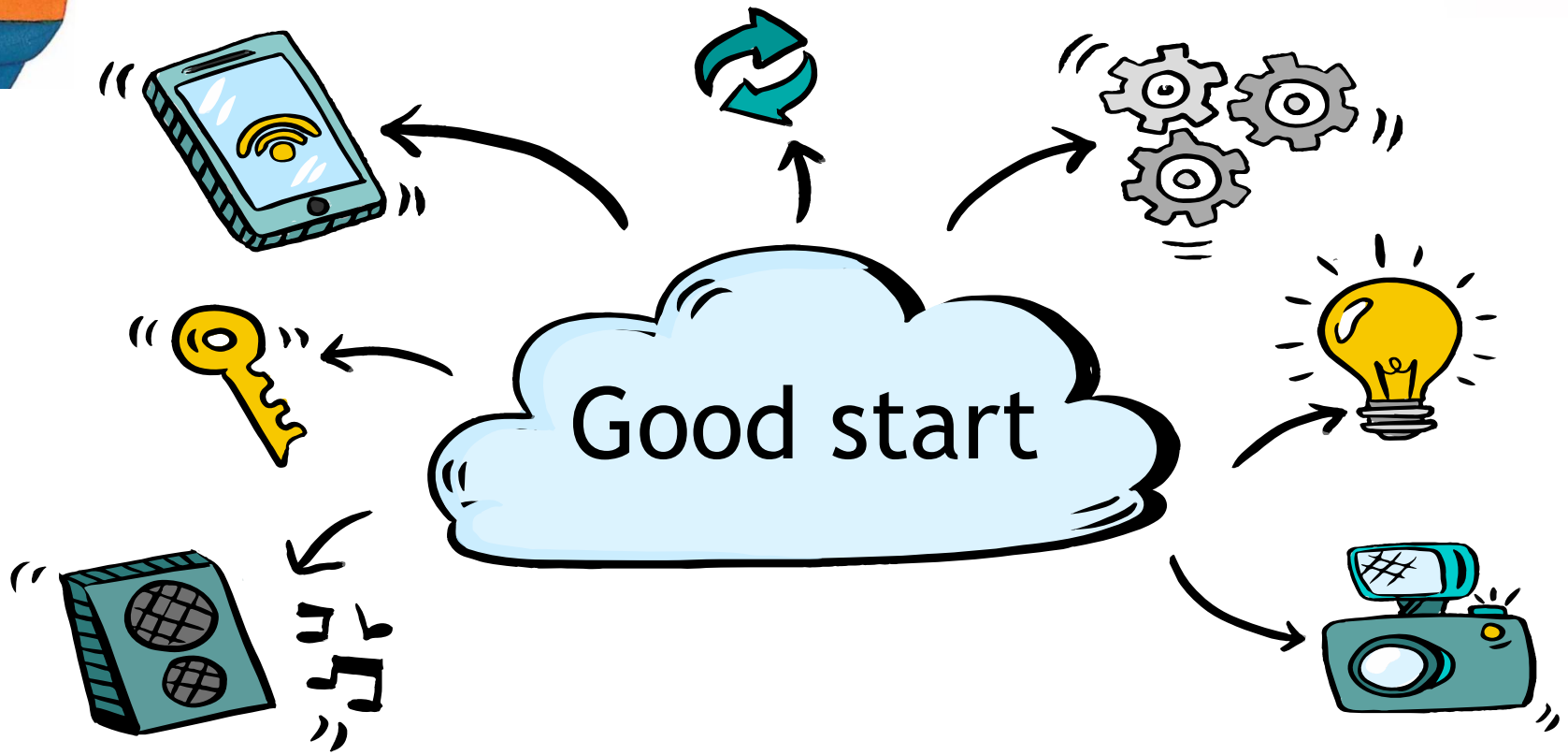
郑晖

- 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

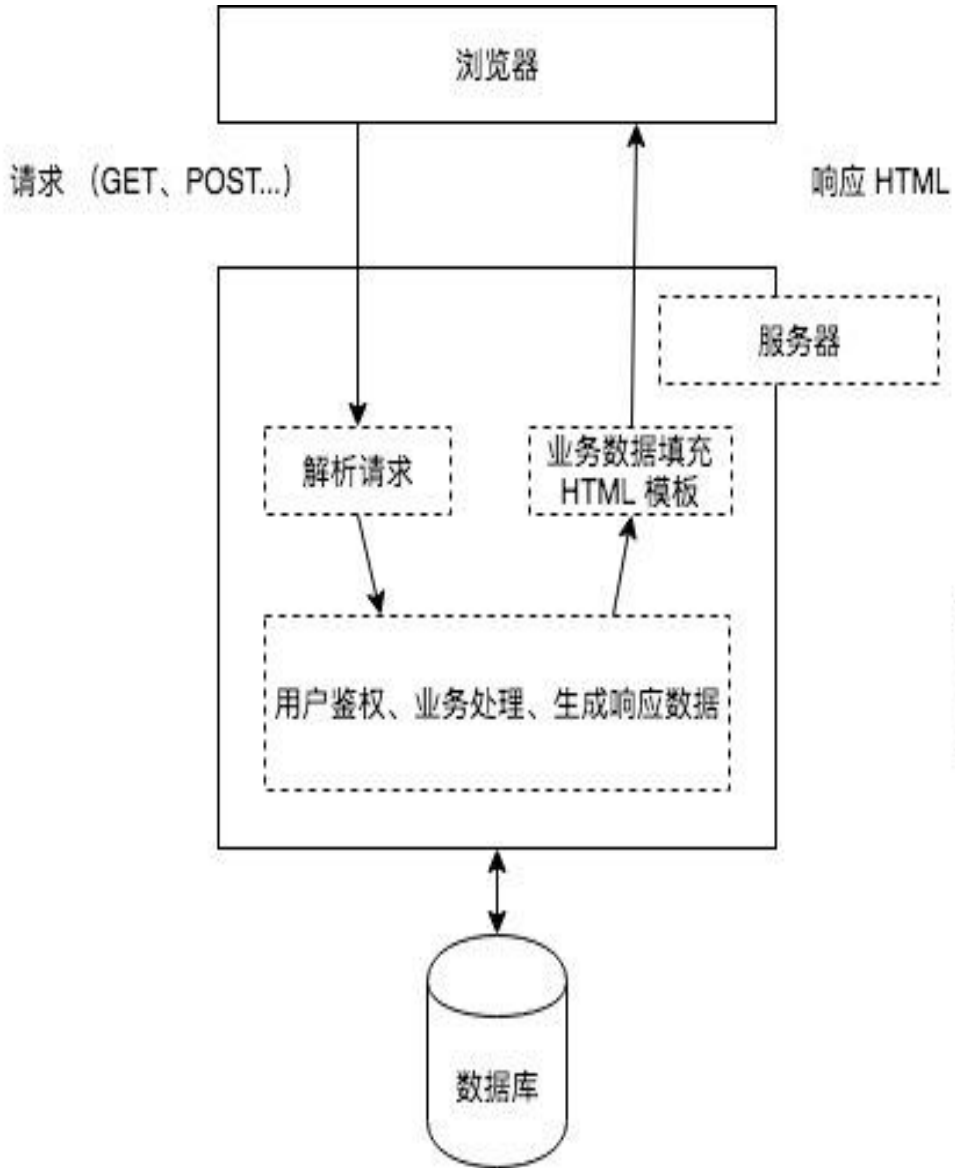
03 你将会了解truedei-swagger-plugin框架

CONTENTS

CONTENTS

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



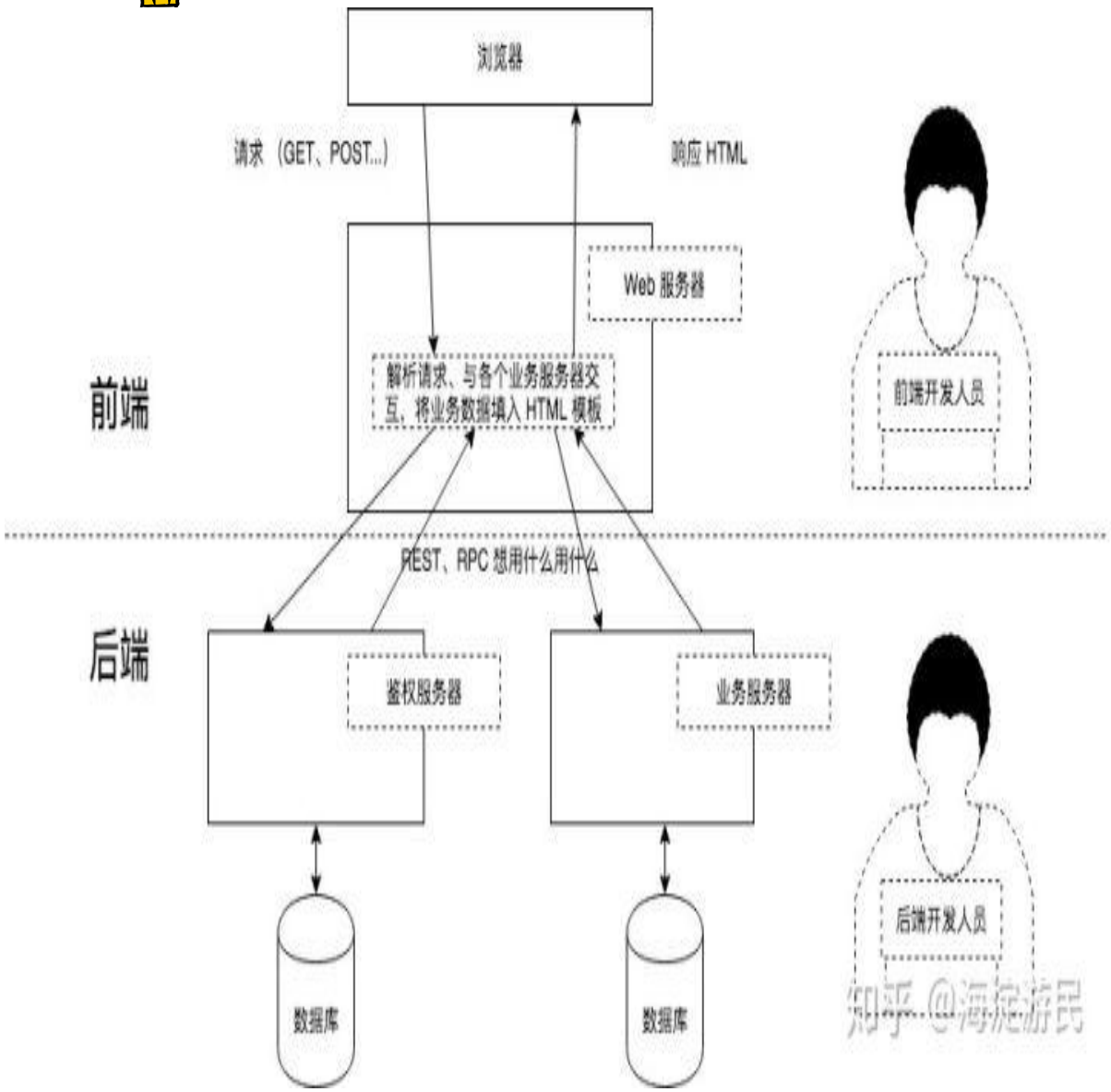
- 存在很大的问题
- 有什么问题呢？
- 问题点在图右边那位全寨（栈）开发人员！
- 开人人员不仅要会服务器端开发（Java），又要会前端开发（HTML、CSS、JS）。还要懂数据库开发（SQL）；

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



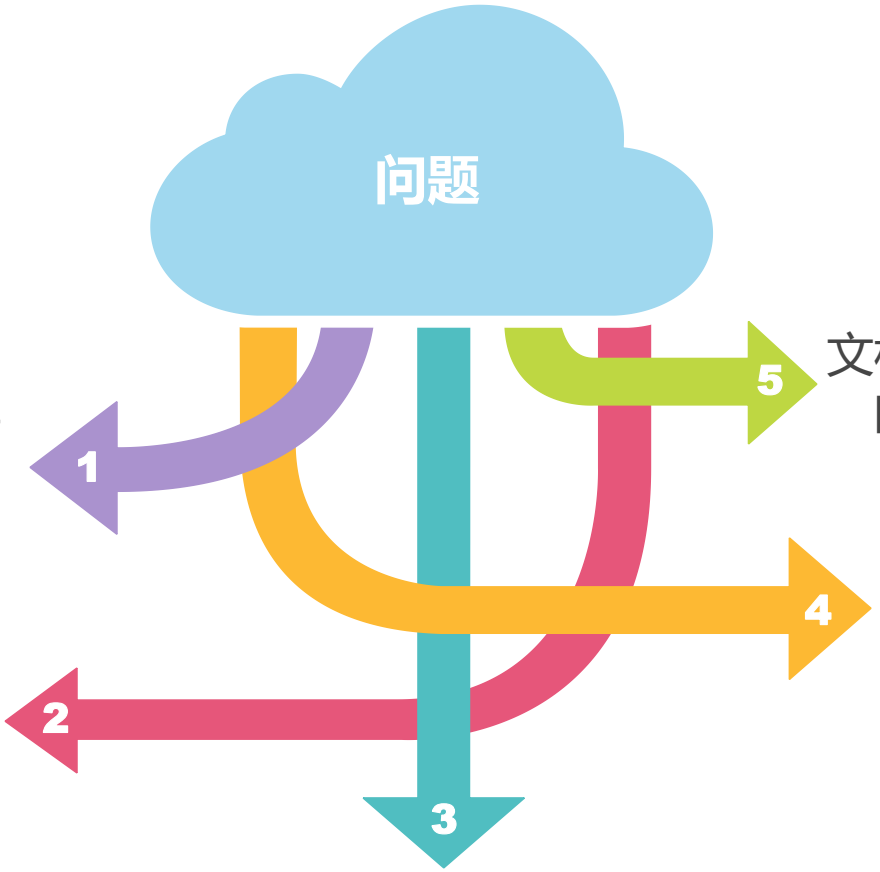
前后端分离模式

- 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。



文档维护困难，需要一定的人力去专门做维护

文档类型难统一，一个团队可能存在各种各样的文档

经常是改变了code，文档不能及时更新

在微服务的盛行下，有成千上万的接口。靠人力来编写，费时还费劲

工欲善其事；

必先利其器。

孔子 -《论语·魏灵公》

CONTENTS

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

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

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

如何结合SpringBoot项目进行应用

step1

pom.xml文件引入相关依赖

step2

使用注解，开启Swagger相关功能；

step3

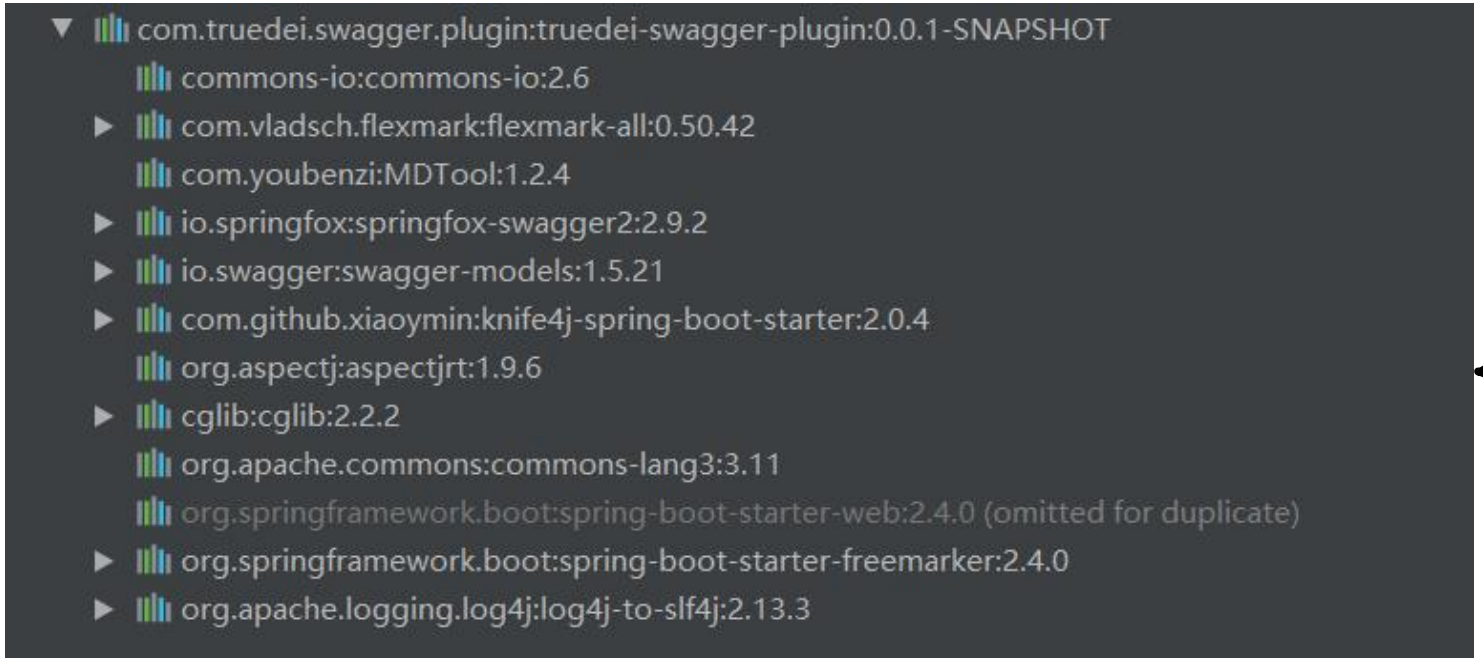
配置Swagger基本信息

step4

使用特定的注解来描述某个需要展示的接口。

Step1:引入Swagger需要的依赖

```xml
<!--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>
```



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）
@ApiModel()	类	表示对类进行说明，用于参数用实体类接收
@ApiModelProperty()	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
@EnableApiFlleURI：开启从配置文件中读取数据的功能

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();
    }
}
```

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();
}
```

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

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

Step4:使用了@APiFileInfo注解后，然后编写特定的文件信息

springbootswagger1 - TestController.md - IntelliJ IDEA

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

springbootswagger1 〉 src 〉 main 〉 resources 〉 md 〉 TestController.md

OcrbootApplication

TestController.java

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

    @PostMapping("/one")
    @ResponseBody
    @ApiOperation(value = "one", notes = "one",position = 1)
    @APiFileInfo("/test/one")
    public String testOne(@Apicp(
                                modelName = "tetsOne",
                                noValues = {"name","address"},
                                noValueTypes = {"string","string"},
                                noVlaueExplains = {"名字","地址"},
                                noVlaueExample = {"郑晖"},
                                noVlaueRequired = {true}
                            )
                          @RequestBody String str){
        return "one";
    }

    @GetMapping("/tow")
    @ResponseBody
    @ApiOperation(value = "tow", notes = "tow",position = 2)
    public String testTow(@Apicp(
            modelName = "Refund",
            noValues = {"orderId","images","remark","addressId"},
            noValueTypes = {"Long","String","String","Long"},
            noVlaueExplains = {"订单ID","图片(列表)最多8个","退款退货原因","收货地址ID"},
            noVlaueExample = {"1","2","不想要了","1"}
    )@RequestBody String str){
        return "tow";
```

TestController.md

```
# URL:/test/one

这是接口描述

---

code:200

**请求成功返回的数据**

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

code:400
请求失败返回的数据
```json
    {
        "code": 400,
        "message": "访问失败了",
        "result":""
    }
```

```

6: TODO Anonymous Apex Log Analyzer Illuminated Cloud SOQL Query Terminal Java Enterprise Spring Event Log

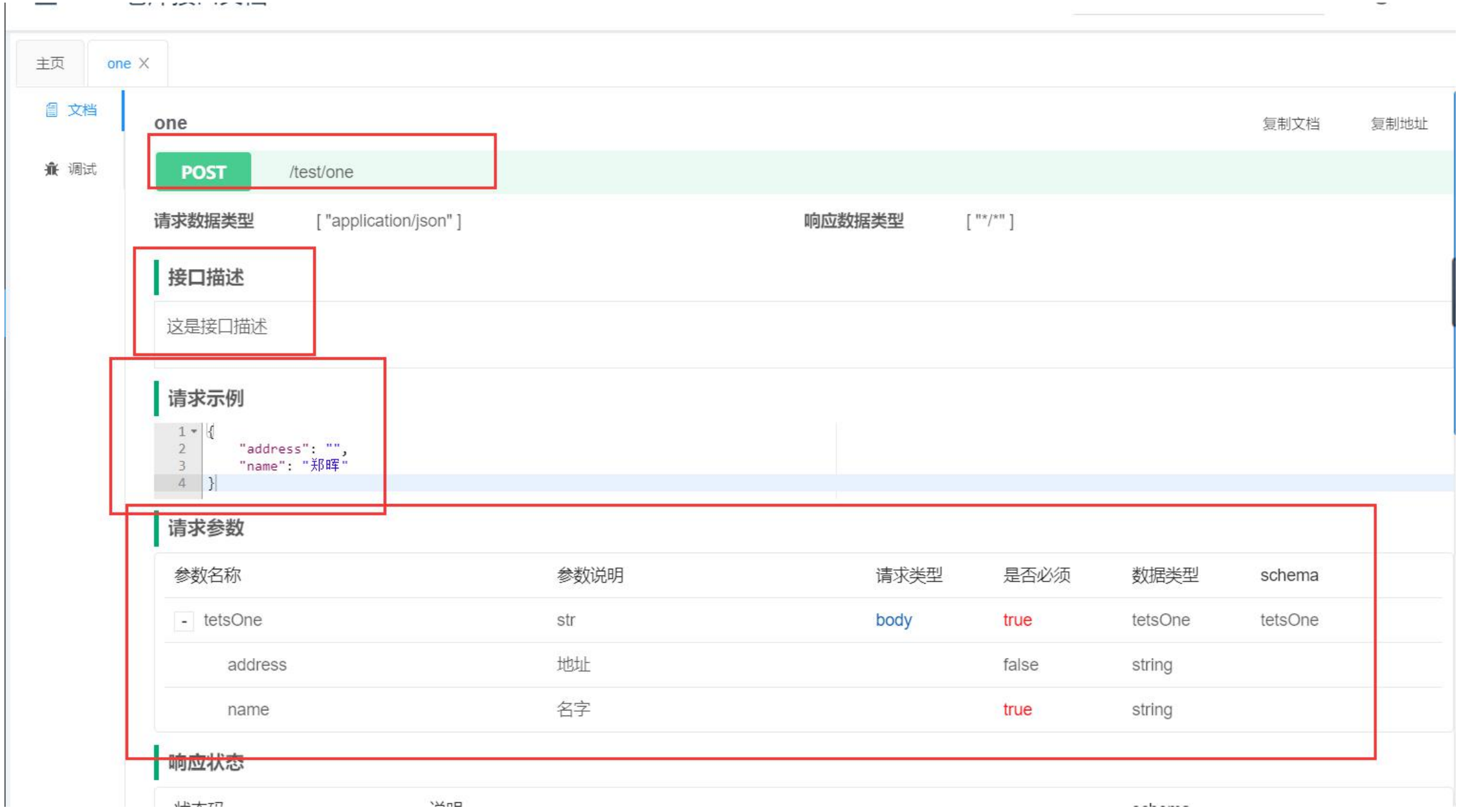
Illuminated Cloud: Your evaluation will expire in 29 days. (2 minutes ago)

19:1 CRLF UTF-8 4 spaces

结果


API仓库接口文档
127.0.0.1:8080/doc.html#/home
API仓库
1
分组的信息
API仓库接口文档
4
标题
输入文档关键字搜索
中
主页
Swagger Models
文档管理
1.test接口
one
tow
2
扫描的有效接口
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
Apache License 2.0 | Copyright © 2019-Knife4j

结果


主页
one
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
{
"address": "",
"name": "郑晖"
}
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

API仓库接口文档

输入文档关键字搜索

主页 | one

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)

```
{
    "address": "",
    "name": "郑晖"
}
```

响应内容 | Raw | Headers | Curl

显示说明 响应码: 200 耗时: 123ms 大小: 3 b

```
one
```

CONTENTS

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

```
@ResponseBody
@RequestMapping(value = "/editRobot", method = RequestMethod.POST)
@ApiOperation(position = 3,value = "15.3.编辑机器人",
        notes = "<h1><strong>编辑机器人</strong></h1>\n" +
        "<p>//说明：robotName和categotyIds二者不能都为空</p>\n" )
public Object editRobot(  @Apicp(
            classPath=RobotConfig.class, //指定原始类
            values = {"robotId","robotName"}, //指定原始类中已经存在的属性
            noValues = {"categotyIds"}, //指定新增的属性
            noValueTypes = {"string"}, //指定新增属性的类型，(忽略大小写)
            noVlaueExplains = {"所选分类，例如：2,3,4,5,6"}, //指定新增属性的描述
            modelName = "EditRobotModel") //指定新生成的类的名字
         @RequestBody String str) {

        String userId = getOperatorId(request);
        if(userId == null || "".equals(userId)){
            return BaseResponseMsg.getDefaultTokenError();
        }

        Map map = JSONObject.parseObject(str, Map.class);

        String robotIdStr;
        if(map == null || map.get("robotId") == null || "".equals(robotIdStr = m
            return BaseResponseMsg.getConstructMsg(BaseResponseMsg.RESPONSE_CODE

        try {
            long robotId = Long.valueOf(robotIdStr);
            RobotConfig robotConfig = robotConfigMpg.selectByRobotId(robotId);
```

Run: Application

Console Endpoints

知识库接口文档

输入文档关键字搜索

主页 15.3.编辑机器人

文档 调试

15.3.编辑机器人

复制文档 复制地址

POST /robotConfig/editRobot

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

接口描述

编辑机器人

//说明：robotName和categotyIds二者不能都为空

请求示例

```
{
    "categotyIds": "",
    "robotId": 0,
    "robotName": ""
}
```

1 可以看到请求的示例和请求的参数都有了

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
categotyIds	所选分类，例如：2,3,4,5,6		false	string	
robotId			false	integer(int64)	
robotName			false	string	

响应状态

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

解决之前

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 ["*/*"]

接口描述

新增机器人

请求示例

```
{
    "pcode": "",
    "robotName": "",
    "zidingyiIntType": 0,
    "zidingyide": ""
}
```

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，维护不方便的解决办法

```
        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();
    }
```

```
250 ---
251
252
253
254 # URL:/robotConfig/totalCount
255 ---
256 code:200
257 ```
258 {
259     "messageCode": 200,
260     "message": "success",
261     "result": {
262         "totalCount": 15
263     }
264 }
265 ```
266 ---
267 code:510
268 ```json
269 {
270     "messageCode": 510,
271     "message": "system error"
272 }
273 ```
274 ---
```

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，维护不方便的解决办法

解决之前

```
@ResponseBody
@RequestMapping(value = "/edit", method = RequestMethod.POST)
@ApiOperation(position = 3,value = "9.3.编辑人工监管配置",notes = "<p><strong>请求的url:</strong></p>\n" +
        "<p>/api/background/config/robotMonitorConfig/edit</p>\n" +
        "<p><strong>说明:</strong></p>\n" +
        "<p>1.authStatus/warnStatus/warnRule/warnKeywords不能都为空</p>\n" +
        "<p>2.如果需要删除warnKeywords，则传空字符或者空数组皆可</p>\n" +
        "<p><strong>请求body参数示例:</strong></p>\n" +
        "<details> \n" +
        "<summary>点击展开查看</summary> \n" +
        "<pre><code class=\"language-json\">{\n" +
        "    robotId: 18,\n" +
        "    authStatus: 1,\n" +
        "    warnStatus: 1,\n" +
        "    warnRule: \n" +
        "    [\n" +
        "        {\n" +
        "            &quot;angry&quot;:1\n" +
        "        },\n" +
        "        {\n" +
        "            &quot;unknown&quot;:1\n" +
        "        },\n" +
        "        {\n" +
        "            &quot;down&quot;:1\n" +
        "        },\n" +
        "        {\n" +
        "            &quot;sameAnswer&quot;:1\n" +
        "        },\n" +
        "        {\n" +
        "            &quot;noClick&quot;:1\n" +
        "        },\n" +
        "        {\n" +
        "            &quot;keywords&quot;:1\n" +
        "        }\n" +
        "    ],\n" +
        "    warnKeywords: \n" +
        "    [\n" +
        "        &quot;转人工&quot;,\n" +
        "        &quot;人工回复&quot;\n" +
        "    ],\n" +
```

解决之后

在code中**减少**了**99%**的**无用代码**

```
@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();
```

总结与回顾

总结

70%

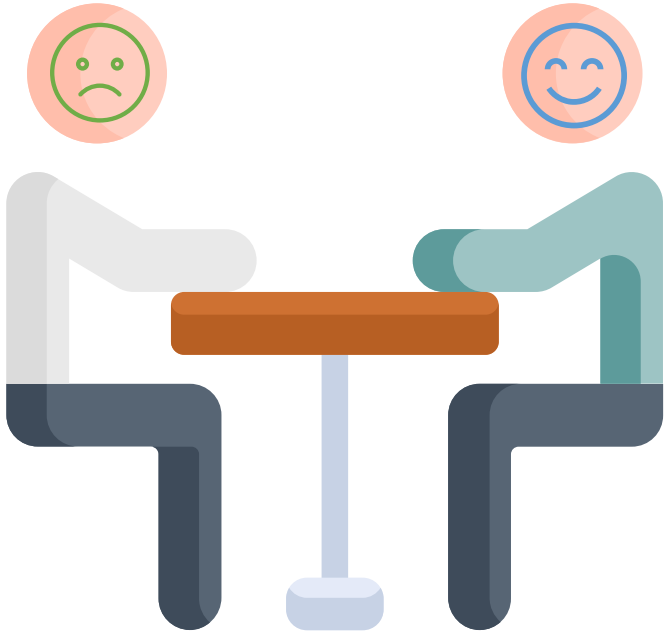
90%

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；



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

谢谢观看