链滴

前后端分离时如何优雅的编写 API 文档

作者：kangaroo1122
原文链接：https://ld246.com/article/1573188882137
来源网站：链滴

在前后端分离的项目中，难免会涉及到接口文档的编写维护问题，正好最近的项目就涉及到了这个方的东西，为此，有必要记录一下如何优雅的完成这一巨坑的填补。

本次使用的工具是swagger2，swagger-ui，swagger2markup，knife4j，后边将对这些插件做一简单的介绍。

1 介绍

swagger是一个API接口文档生成工具，官网：https://swagger.io。项目集成swagger后，只需要单的几个注解便可以在编写接口的同时完成文档的编写，配合swagger-ui，便可以在项目启动后，直浏览器访问指定地址，一般是：

http://{IP}:{端口}/{项目名}/swagger-ui.html

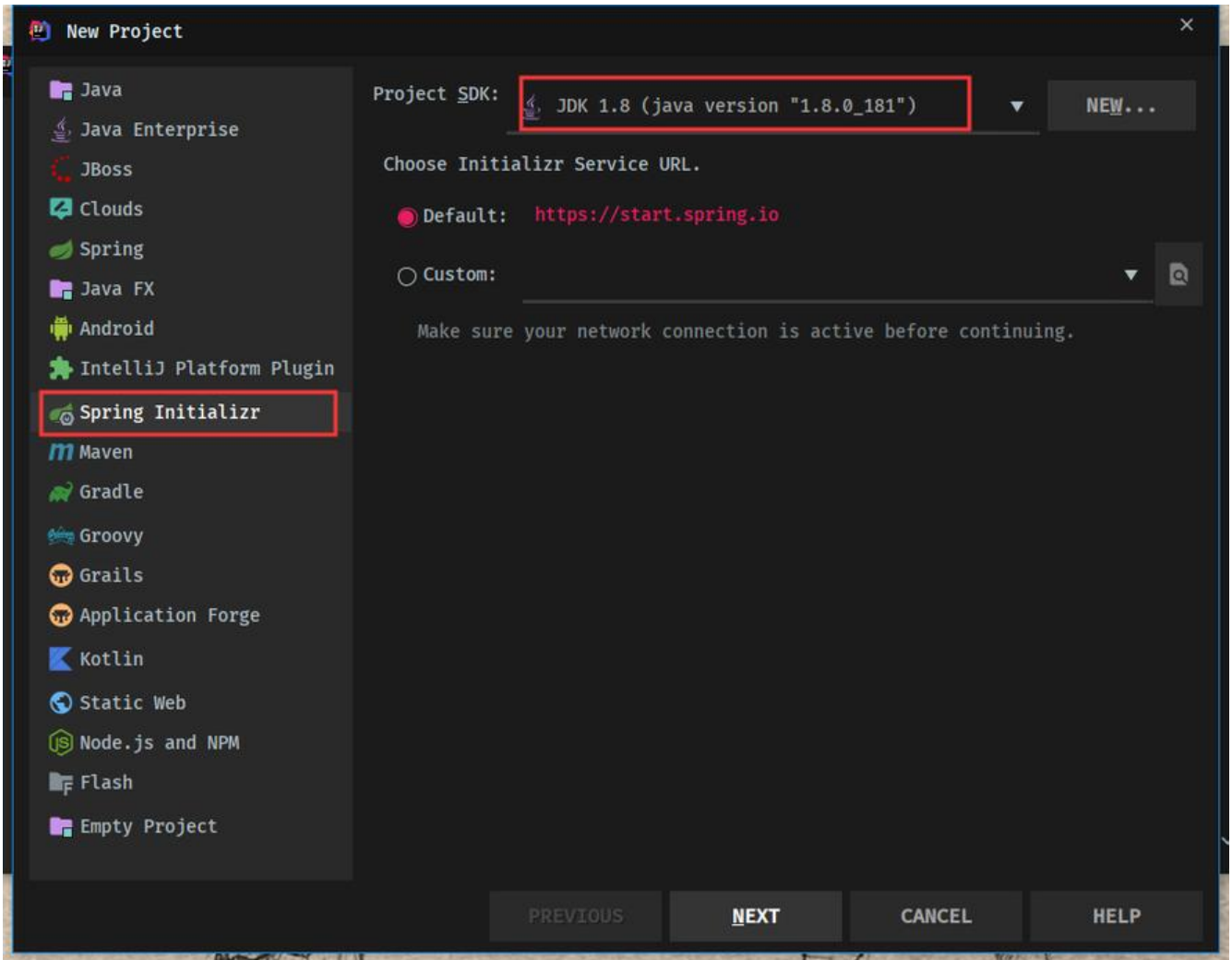
调试API。而整合 swagger2markup 后，还可以将API直接导出成html、markdown等格式的文档。

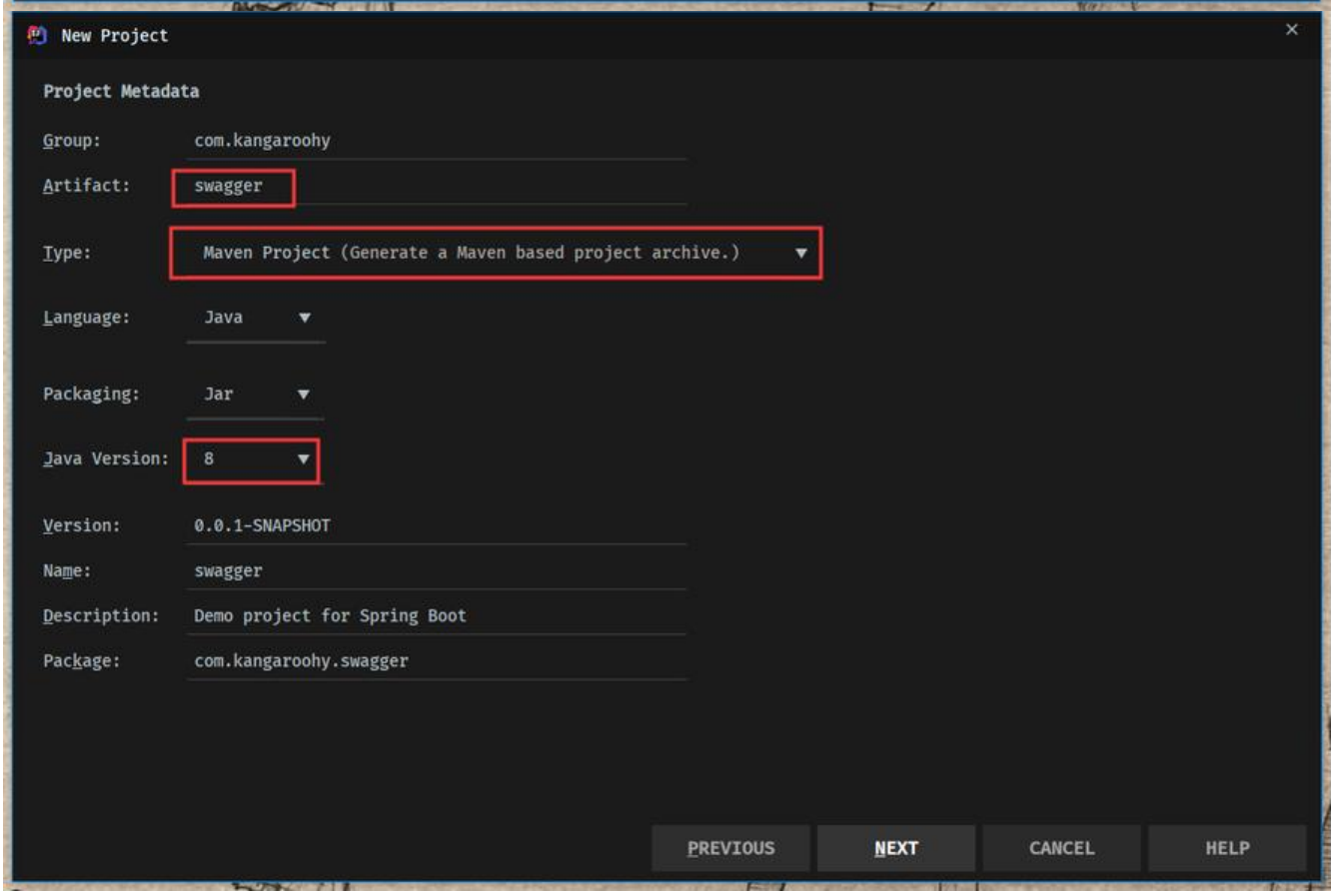
knife4j，前身的swagger-bootstrap-ui，是Swagger生成Api文档的增强解决方案，具体介绍这里再细说，直接上官网：https://doc.xiaominfo.com 。集成这个插件，可以换掉swagger-ui丑陋的皮，还能完成API的排序，同时也能直接生成markdown格式的文档。话不多说，开搞！

2 新建项目

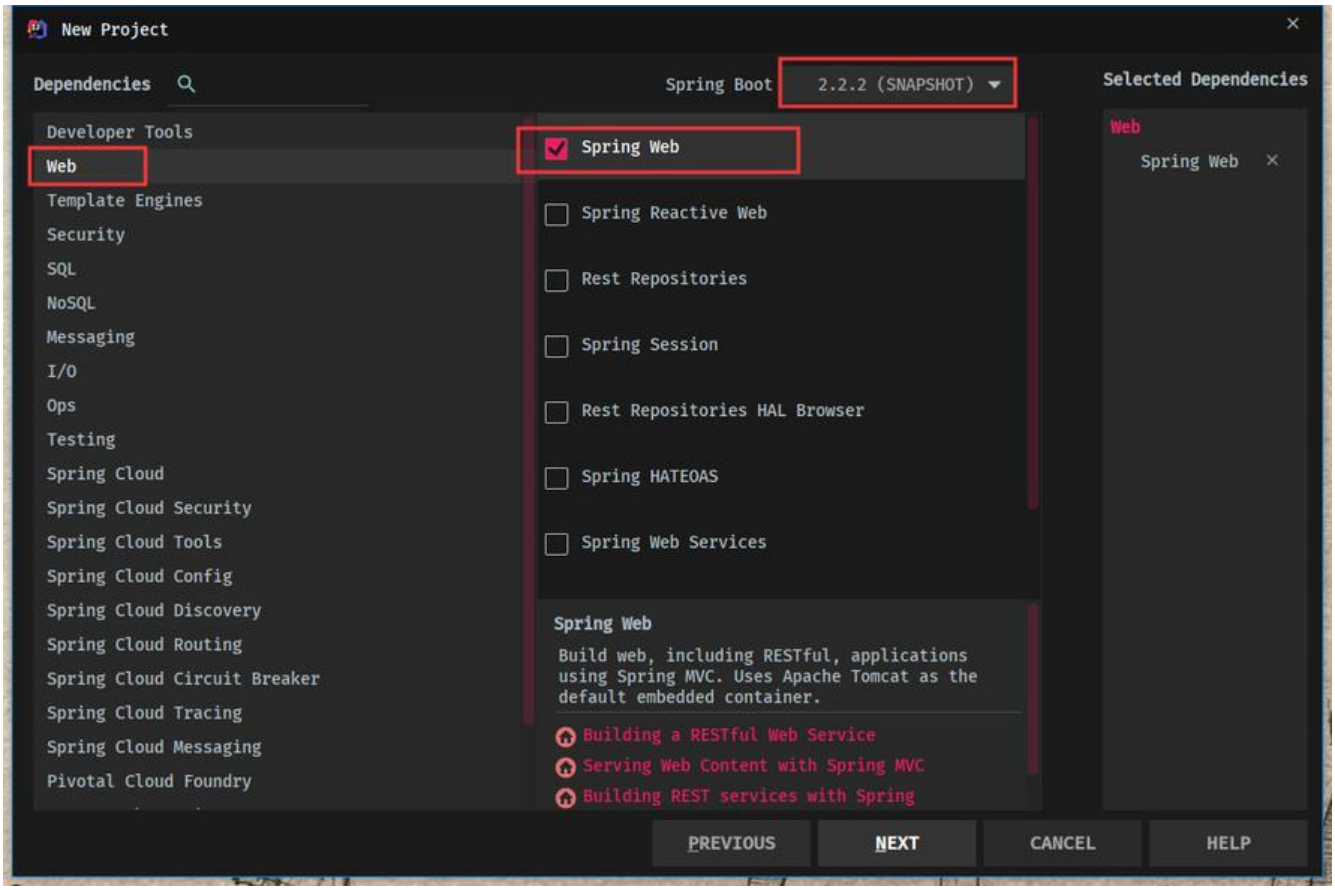
2.1 创建

这里创建一个springboot项目，几个简单的配置项以后，便是等待下载完成。





原文链接：前后端分离时如何优雅的编写 API 文档



2.2 controller

新建一个controller包，并创建一个TestController

```
@RestController
@RequestMapping("/v1")
public class TestController {

    @RequestMapping(value = "/test1", method = RequestMethod.GET)
    public String test1(@RequestParam String username, @RequestParam String password) {
        return username + password;
    }
}
```

3 集成swagger

3.1 添加依赖

项目创建完成后，在pom.xml中添加如下的依赖：

```
    <repositories>
        <repository>
            <snapshots>
                <enabled>true</enabled>
                <updatePolicy>always</updatePolicy>
```

```
        </snapshots>
        <id>jcenter-releases</id>
        <name>jcenter</name>
        <url>http://jcenter.bintray.com</url>
    </repository>
</repositories>

    <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger2</artifactId>
        <version>2.9.2</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger-ui</artifactId>
        <version>2.9.2</version>
    </dependency>
<!-- 导出html、markdown等 -->
    <dependency>
        <groupId>io.github.swagger2markup</groupId>
        <artifactId>swagger2markup</artifactId>
        <version>1.3.1</version>
    </dependency>
```

将来导出html需要的插件配置

```
    <plugins>
        <plugin>
            <groupId>org.asciidoctor</groupId>
            <artifactId>asciidoctor-maven-plugin</artifactId>
            <version>1.5.6</version>
            <configuration>
                <sourceDirectory>src/docs/asciidoc/generated</sourceDirectory>
                <outputDirectory>src/docs/asciidoc/html</outputDirectory>
                <headerFooter>true</headerFooter>
                <doctype>book</doctype>
                <backend>html</backend>
                <sourceHighlighter>coderay</sourceHighlighter>
                <attributes>
                    <!--菜单栏在左边-->
                    <toc>left</toc>
                    <!--多标题排列-->
                    <toclevels>3</toclevels>
                    <!--自动打数字序号-->
                    <sectnums>true</sectnums>
                </attributes>
            </configuration>
        </plugin>
    </plugins>
```

3.2 配置swagger2

新建一个config包，并创建一个配置类：SwaggerConfig.java

```
@Configuration
public class SwaggerConfig {
    // 接口大标题
    private final String title = "测试API";
    // 具体的描述
    private final String description = "测试项目API文档";
    // 接口版本号
    private final String version = "1.0.0";
    // 服务说明url，服务条款
    private final String termsOfServiceUrl = "http://blog.kangaroohy.top";
    // licence，许可证
    private final String license = "MIT";
    // licnce url，许可网址
    private final String licenseUrl = "https://mit-license.org/";
    // 接口作者联系方式
    private final Contact contact = new Contact("kangaroo1122", "http://blog.kangaroohy.top",
"326170945@qq.com");

    @Bean
    public Docket buildDocket() {
        return new Docket(DocumentationType.SWAGGER_2).apiInfo(buildApiInf())
                .groupName("v1-version-api")
                .select()
                .apis(RequestHandlerSelectors.basePackage("com.kangaroohy.swagger.controller"))
                .paths(PathSelectors.any())
                .build();
    }

    private ApiInfo buildApiInf() {
        return new ApiInfoBuilder().title(title).termsOfServiceUrl(termsOfServiceUrl).description(d
escription)
                .version(version).license(license).licenseUrl(licenseUrl).contact(contact).build();
    }
}
```

同时在项目启动类添加一个注解：@EnableSwagger2

```
@SpringBootApplication
@EnableSwagger2
public class SwaggerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SwaggerApplication.class, args);
    }

}
```

接着，需要在controller中加上swagger的注解：

```
@RestController
@RequestMapping("/v1")
@Api(tags = {"Test接口测试"})
```

```
public class TestController {

    @RequestMapping(value = "/test1", method = RequestMethod.GET)
    @ApiOperation(value = "测试url-test1", httpMethod = "GET", produces = "application/json"
 notes = "用于获取测试数据")
    @ApiImplicitParams({
            @ApiImplicitParam(name = "username", value = "姓名", required = true, dataType = "
tring", paramType = "query"),
            @ApiImplicitParam(name = "password", value = "密码", required = true, dataType = "S
ring", paramType = "query")
    })
    public String test1(@RequestParam String username, @RequestParam String password) {
        return username + password;
    }
}
```

各个注解在这里做一个简单的介绍。

3.2.1 @Api 注解

说明每个controller层控制类的作用，即每个请求类

属性：tags：描述该类的作用

3.2.2 @ApiOperation 注解

作用在控制类中的每个方法上，说明该类的作用

属性：value：说明该类的作用

3.2.3 @ApiParam 注解

@ApiParam作用于请求方法上，定义api参数的注解，属性有：

name：api参数的英文名

value：api参数的描述

required：true表示参数必输，false标识参数非必输，默认为非必输

此注解通常与@RequestParam或者@PathVariable集合使用，因为它的作用只是定义每个参数（因可以不使用，但是为了方便测试，加上效果更好），如果要获取前端的参数还需要通过@RequestParam或者@PathVariable来获取

3.2.4 @ApiImplicitParams 注解和@ApiImplicitParam 注解

定义参数的注解除了@ApiParam之外，这两个注解也可以定义参数

①、@ApiImplicitParams定义一组参数

②、@ApiImplicitParam写在@ApiImplicitParams中，定义每个参数的信息，属性为：

name：参数英文名称

value：参数中文名称

paramType：调用的url 参数形式，“query”为问号"?"后面的拼接参数，“path”为绑定的参数

此时，项目已经大概集成完毕，运行项目，看看效果: http://localhost:8080/swagger-ui.html

点开相应的接口，可以做相应的测试。

http://localhost:8080/v2/api-docs?group=v1-version-api效果如下：


```
{
  "swagger": "2.0",
  "info": {
    "description": "测试项目API文档",
    "version": "1.0.0",
    "title": "测试API",
    "termsOfService": "http://blog.kangaroohy.top",
    "contact": {
      "name": "kangaroo1122",
      "url": "http://blog.kangaroohy.top",
      "email": "326170945@qq.com"
    },
    "license": {
      "name": "MIT",
      "url": "https://mit-license.org/"
    }
  },
  "host": "localhost:8080",
  "basePath": "/",
  "tags": [
    {
      "name": "Test接口",
      "description": "Test Controller"
    }
  ],
  "paths": {
    "/v1/test1": {
      "get": {
        "tags": [
          "Test接口"
        ],
        "summary": "测试url-test1",
        "description": "用于获取测试数据",
```

```
            "operationId": "test1UsingGET",
            "produces": [
              "*/*",
              "application/json"
            ],
            "parameters": [
              {
                "name": "password",
                "in": "query",
                "description": "密码",
                "required": true,
                "type": "string"
              },
              {
                "name": "username",
                "in": "query",
                "description": "姓名",
                "required": true,
                "type": "string"
              }
            ],
            "responses": {
              "200": {
                "description": "OK",
                "schema": {
                  "type": "string"
                }
              },
              "401": {
                "description": "Unauthorized"
              },
              "403": {
                "description": "Forbidden"
              },
              "404": {
                "description": "Not Found"
              }
            },
            "deprecated": false,
            "x-order": "1"
          }
        }
    }
}
```

3.3 配置swagger2markup

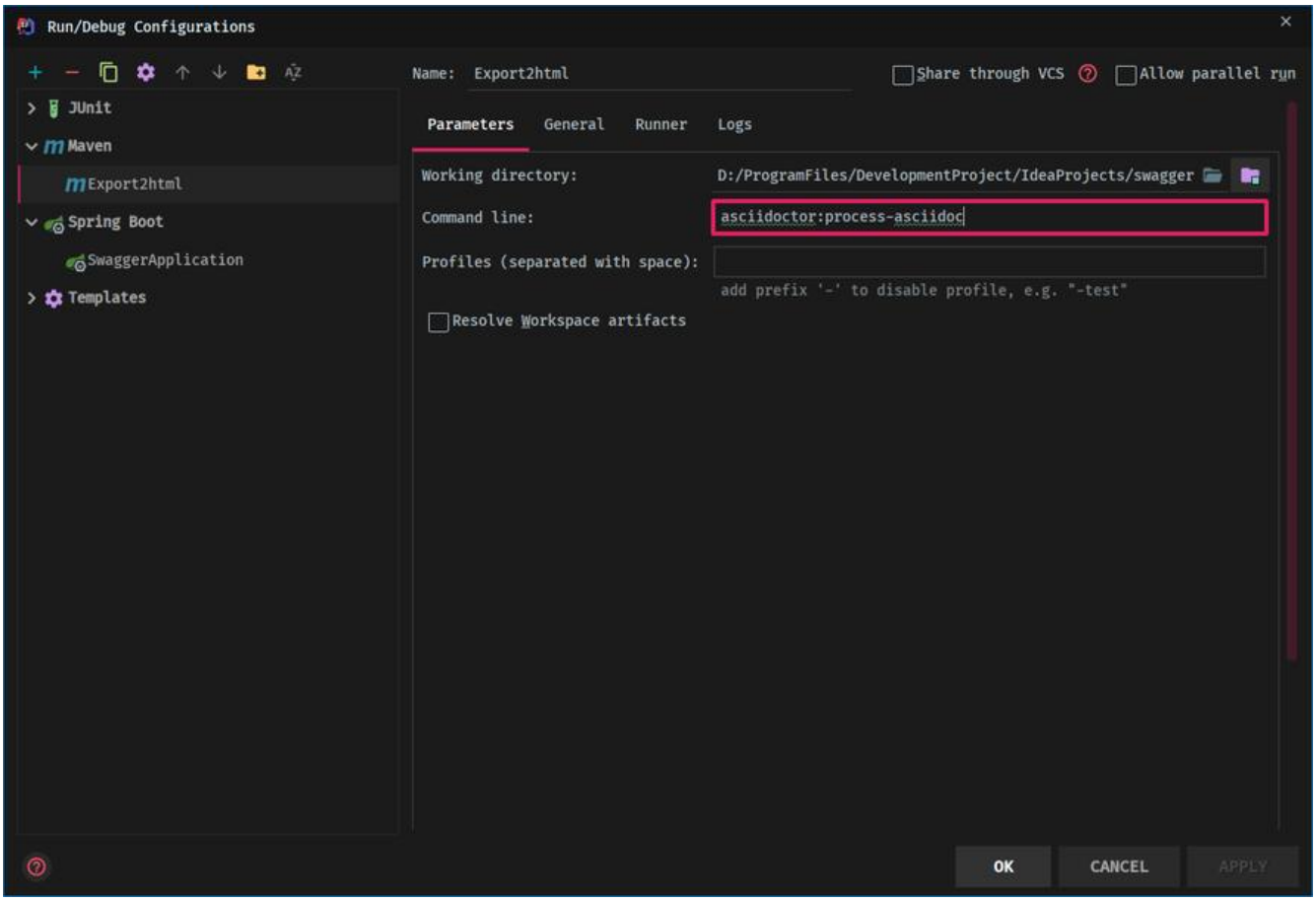
需要添加的依赖和插件在 3.1 已经添加好了，接下来是新建一个导出文档的配置类：ExportConfig.java

```
@RunWith(SpringRunner.class)
public class ExportConfig {
    /**
     * 生成AsciiDocs格式文档
```

```
     * @throws Exception
     */
    @Test
    public void generateAsciiDocs() throws Exception {
        //    输出Ascii格式
        Swagger2MarkupConfig config = new Swagger2MarkupConfigBuilder()
                .withMarkupLanguage(MarkupLanguage.ASCIIDOC)
                .withOutputLanguage(Language.ZH)
                .withPathsGroupedBy(GroupBy.TAGS)
                .withGeneratedExamples()
                .withoutInlineSchema()
                .build();

        Swagger2MarkupConverter.from(new URL("http://localhost:8080/v2/api-docs?group=v1
version-api"))
                .withConfig(config)
                .build()
                .toFolder(Paths.get("src/docs/asciidoc/generated"));
    }
    /**
     * 生成Markdown格式文档
     * @throws Exception
     */
    @Test
    public void generateMarkdownDocs() throws Exception {
        //    输出Markdown格式
        Swagger2MarkupConfig config = new Swagger2MarkupConfigBuilder()
                .withMarkupLanguage(MarkupLanguage.MARKDOWN)
                .withOutputLanguage(Language.ZH)
                .withPathsGroupedBy(GroupBy.TAGS)
                .withGeneratedExamples()
                .withoutInlineSchema()
                .build();

        Swagger2MarkupConverter.from(new URL("http://localhost:8080/v2/api-docs?group=v1
version-api"))
                .withConfig(config)
                .build()
                .toFolder(Paths.get("src/docs/markdown/generated"));
    }
    /**
     * 生成Confluence格式文档
     * @throws Exception
     */
    @Test
    public void generateConfluenceDocs() throws Exception {
        //    输出Confluence使用的格式
        Swagger2MarkupConfig config = new Swagger2MarkupConfigBuilder()
                .withMarkupLanguage(MarkupLanguage.CONFLUENCE_MARKUP)
                .withOutputLanguage(Language.ZH)
                .withPathsGroupedBy(GroupBy.TAGS)
                .withGeneratedExamples()
                .withoutInlineSchema()
                .build();
```

```
        Swagger2MarkupConverter.from(new URL("http://localhost:8080/v2/api-docs?group=v1
version-api"))
                .withConfig(config)
                .build()
                .toFolder(Paths.get("src/docs/confluence/generated"));
    }
    /**
     * 生成AsciiDocs格式文档,并汇总成一个文件
     * @throws Exception
     */
    @Test
    public void generateAsciiDocsToFile() throws Exception {
        //    输出Ascii到单文件
        Swagger2MarkupConfig config = new Swagger2MarkupConfigBuilder()
                .withMarkupLanguage(MarkupLanguage.ASCIIDOC)
                .withOutputLanguage(Language.ZH)
                .withPathsGroupedBy(GroupBy.TAGS)
                .withGeneratedExamples()
                .withoutInlineSchema()
                .build();

        Swagger2MarkupConverter.from(new URL("http://localhost:8080/v2/api-docs?group=v1
version-api"))
                .withConfig(config)
                .build()
                .toFile(Paths.get("src/docs/asciidoc/generated/all"));
    }
    /**
     * 生成Markdown格式文档,并汇总成一个文件
     * @throws Exception
     */
    @Test
    public void generateMarkdownDocsToFile() throws Exception {
        //    输出Markdown到单文件
        Swagger2MarkupConfig config = new Swagger2MarkupConfigBuilder()
                .withMarkupLanguage(MarkupLanguage.MARKDOWN)
                .withOutputLanguage(Language.ZH)
                .withPathsGroupedBy(GroupBy.TAGS)
                .withGeneratedExamples()
                .withoutInlineSchema()
                .build();

        Swagger2MarkupConverter.from(new URL("http://localhost:8080/v2/api-docs?group=v1
version-api"))
                .withConfig(config)
                .build()
                .toFile(Paths.get("src/docs/markdown/generated/all"));
    }
}
```

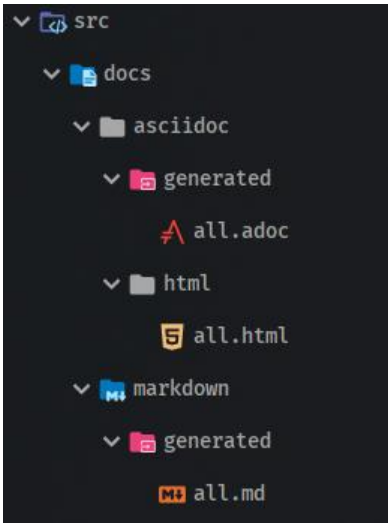
在idea中配置一个maven命令asciidoctor:process-asciidoc，方便执行（也可不用配置，每次需要出时手动输入运行）



3.4 导出html和markdown文档

首先运行ExportConfig.java中的generateAsciiDocsToFile()方法，将文档生成到一个总的AsciiDocs式文档，再执行之前配置的maven命令，生成html文档，完成之后，打开项目下docs文件，便可以到生成的文档。

同理，运行generateMarkdownDocsToFile()将生成一个总的markdown格式的文档，其他的可以自。



4 集成knife4j

原文链接：前后端分离时如何优雅的编写 API 文档

虽然swagger-ui能完成前端文档的展示和调试，但是界面不是很友好，为此，引入knife4j，优点就说了，具体看官方文档。

4.1 依赖

删除官方的swagger-ui，并引入knife4j，其他的依赖不变

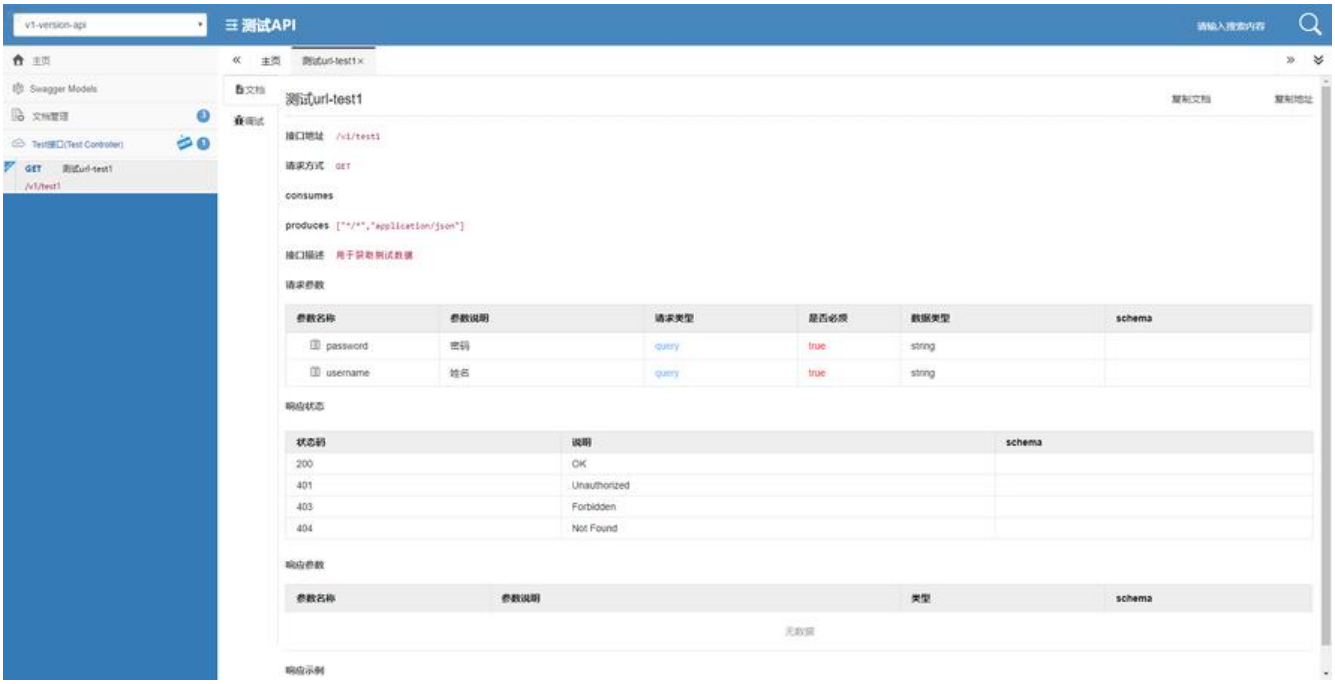
```
<dependency>
    <groupId>com.github.xiaoymin</groupId>
    <artifactId>knife4j-spring-boot-starter</artifactId>
    <version>1.9.6</version>
</dependency>
```

4.2 配置

在swagger的配置类SwaggerConfig.java上加一个注解：@EnableSwaggerBootstrapUi，值得注的是，这个注解的最后一个字母是小写i，如果改成大写I，则只是给前端文档换一个皮肤，不能使用强功能（排序 等）

```
@Configuration
@EnableSwaggerBootstrapUi
public class SwaggerConfig {
```

此时，访问http://{IP}:{端口}/{项目名}/doc.html，即：http://localhost:8080/doc.html（我这里IEA运行项目时没有添加项目名，这个根据自己设定来。）效果如下：



4.3 其他设置

项目接口排序，有两种排序的方式，具体操作，官方说的很清楚，直接上链接：接口排序