网络案例： <https://github.com/xiaomoinfo/xiaomo-api/blob/master/pom.xml>

这个文档使用的swagger2

官方： <https://swagger.io/tools/swagger-ui/download/>

案例：<https://gitee.com/zx19890628/spring-boot-example/tree/master/lab_003_web_swagger>

# 集成swagger-api

## 环境

<dependency>

<groupId>io.springfox</groupId>

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

<version>2.2.2</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.2.2</version>

</dependency>

## 配置设置了一些默认显示的api相关信息。

@Configuration

@EnableSwagger2

public class Swagger2Config {

@Bean

public Docket createRestApi() {

return new Docket(DocumentationType.SWAGGER\_2)

.apiInfo(apiInfo())

.select()

.apis(RequestHandlerSelectors.basePackage("org.chapter\_03\_swagger"))//API所在的包路径

.paths(PathSelectors.any())

.build();

}

/\*\*

\* @Title: 基本信息

\* @return

\* @date: 2018年8月18日上午10:00:58

\*/

private ApiInfo apiInfo() {

return new ApiInfoBuilder()

.title("Spring Boot中使用Swagger2构建RESTful APIs")

.description("api根地址：http://127.0.0.1:8080/")

.termsOfServiceUrl("http://zx1989.top/")

.contact("张雪")

.version("1.0.0")

.build();

}

}

## 相关注解解读

常用注解：<https://blog.csdn.net/wyb880501/article/details/79576784>

1. @Api用在类上，说明该类的作用@Api(value = "UserController", description = "用户相关api")

2. @ApiOperation用在方法上，说明方法的作用@ApiOperation(value = "查找用户", notes = "查找用户", httpMethod = "GET", produces = MediaType.APPLICATION\_JSON\_UTF8\_VALUE)

3 @ApiImplicitParams用在方法上包含一组参数说明

4. @ApiImplicitParam用在@ApiImplicitParams注解中，指定一个请求参数的各个方面paramType：参数放在哪个地方header–>请求参数的获取：@RequestHeaderquery–>请求参数的获取：@RequestParampath（用于restful接口）–>请求参数的获取：@PathVariablebody（不常用）form（不常用）name：参数名dataType：参数类型required：参数是否必须传value：参数的意思defaultValue：参数的默认值@ApiImplicitParams({

@ApiImplicitParam(name = "id", value = "唯一id", required = true, dataType = "Long", paramType = "path"),

})

5. @ApiResponses用于表示一组响应

6. @ApiResponse用在@ApiResponses中，一般用于表达一个错误的响应信息code：数字，例如400message：信息，例如”请求参数没填好”response：抛出异常的类@ApiResponses(value = {

@ApiResponse(code = 400, message = "No Name Provided")

})

7. @ApiModel描述一个Model的信息（这种一般用在post创建的时候，使用@RequestBody这样的场景，请求参数无法使用@ApiImplicitParam注解进行描述的时候）@ApiModel(value = "用户实体类")

8. @ApiModelProperty描述一个model的属性@ApiModelProperty(value = "登录用户")

9. @ApiIgnore ，有的时候一些controller或是某些方法、字段不想作为api时使用这个

## API

### 定义基础数据模型

@ApiModel

public class User implements Serializable {

private static final long serialVersionUID = 1L;

@ApiModelProperty(value = "主键id", hidden = true, position = 1, example = "10001")

private Long id;

@ApiModelProperty(value = "姓名", position = 2, example = "张三")

private String name;

@ApiModelProperty(value = "年龄", position = 3, example = "26")

private Integer age;

@ApiModelProperty(value = "生日", position = 4, example = "1989-06-28")

private Date brithday;

}

### 定义API

@RestController

@RequestMapping(value = "/user", produces = MediaType.APPLICATION\_JSON\_UTF8\_VALUE)

@Api(value = "用户管理接口",description="用户管理", produces = MediaType.APPLICATION\_JSON\_UTF8\_VALUE)

public class UserController {

private static final Map<Integer, User> users = new HashMap<>();

static {

users.put(1, new User(1L, "张三", 21, new Date()));

users.put(2, new User(2L, "李四", 22, new Date()));

users.put(3, new User(3L, "王五", 23, new Date()));

}

@ApiOperation(value = "用户列表", httpMethod = "GET")

@ApiImplicitParams({

@ApiImplicitParam(name="page", value="页码，从1开始", paramType="query", dataType="int"),

@ApiImplicitParam(name="per\_page", value="每页记录数大小", paramType="query", dataType="int")})

@ApiResponses(value = {

@ApiResponse(code = 200, message = "查询成功"),

@ApiResponse(code = 400, message = "查询失败"),

@ApiResponse(code = 401, message = "没有登录验证"),

@ApiResponse(code = 403, message = "操作没有权限被拒绝"),

@ApiResponse(code = 404, message = "找不到指定的资源或数据"),

@ApiResponse(code = 422, message = "JSON中有未知的字段或类型"),

@ApiResponse(code = 503, message = "操作过于频繁")

})

@RequestMapping(value = "/all", method = RequestMethod.GET)

public List<User> all(@RequestParam(value="page", required=false, defaultValue="0") int pageNum,

@RequestParam(value="per\_page", required=false, defaultValue="100") int pageSize) {

System.out.println("pageNum=" + pageNum);

System.out.println("pageSize=" + pageSize);

return new ArrayList<>(users.values());

}

@ApiOperation(value = "保存用户", httpMethod = "POST")

@ApiParam(name="user", value="用户对象",required = true)

@RequestMapping(value = "/save", method = RequestMethod.POST)

public User save(@RequestBody User user) {

System.out.println(user);

user.setId(4L);

users.put(4, user);

return user;

}

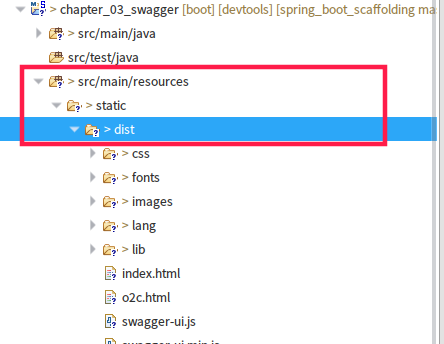
}

## 和Swagger UI的集成

官方自带UI样式：<http://127.0.0.1:8080/swagger-ui.html>

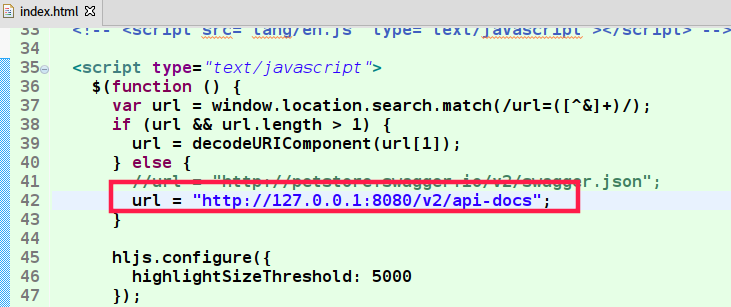
自定义UI样式：<http://127.0.0.1:8080/dist/index.html>

首先，从github swagger-ui 上下载Swagger-UI, 把该项目dist目录下的内容拷贝到项目的resources的目录public下。



将index.html中http://petstore.swagger.wordnik.com/v2/swagger.json修改为

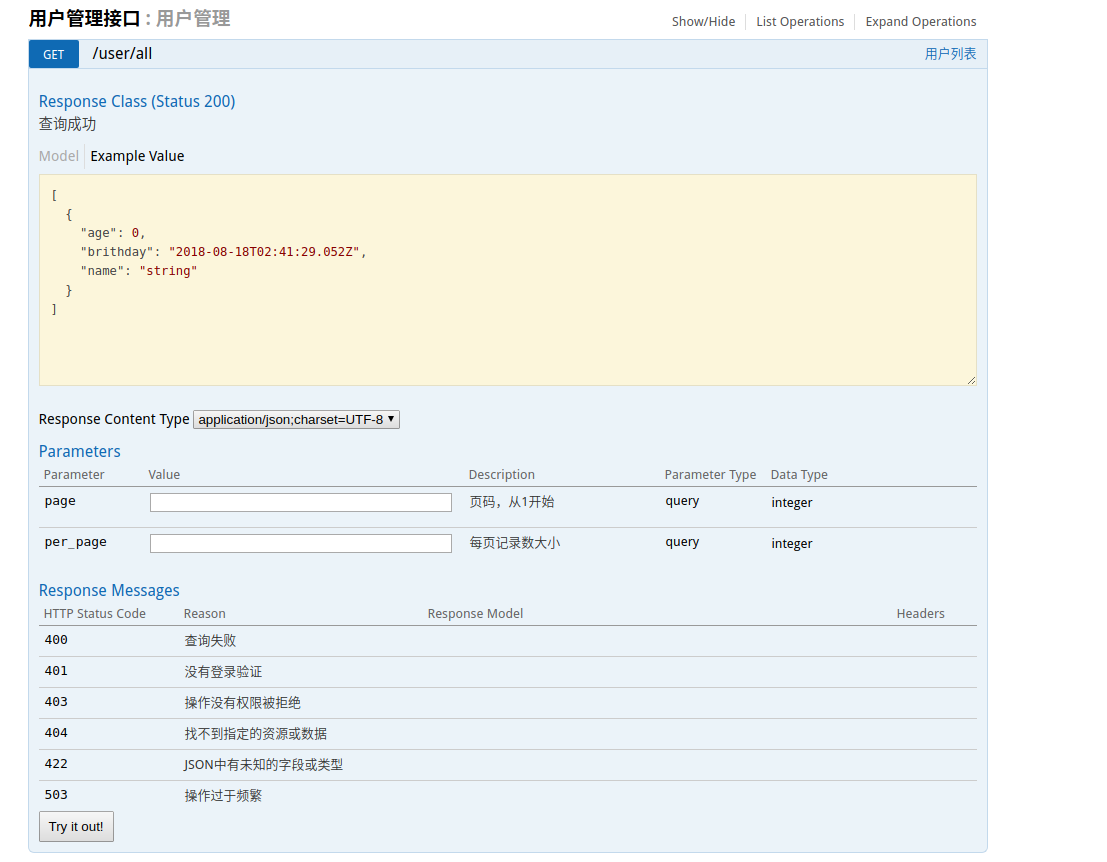
http://localhost:8080/项目名/v2/api-docs



## 访问

http://localhost:8080/dist/index.html 就可以看到效果如下，我们也可以在这里进行功能的测试





# Swagger2Markup生成静态

Swagger2Markup是Github上的一个开源项目。该项目主要用来将Swagger自动生成的文档转换成几种流行的格式以便于静态部署和使用，比如：AsciiDoc、Markdown、Confluence。

项目主页：<https://github.com/Swagger2Markup/swagger2markup>

[案例](https://gitee.com/zx19890628/spring-boot-example/blob/master/lab_003_web_swagger/src/test/java/com/zx/swagger/ApplicationTests.java)

### 如何使用

在使用Swagger2Markup之前，我们先需要准备一个使用了Swagger的Web项目，可以是直接使用Swagger2的项目，也可以使用Spring Boot 2.x基础教程：使用Swagger2构建强大的API文档一文中构建的项目。读者可以通过下面的仓库获取：

* Github：https://github.com/dyc87112/SpringBoot-Learning/tree/2.x
* Gitee：https://gitee.com/didispace/SpringBoot-Learning/tree/2.x

接下来，我们将利用这个项目中的chapter2-2模块作为基础来来生成几种不同格式的静态文档。

### 生成 AsciiDoc 文档

生成 AsciiDoc 文档的方式有两种：

#### 通过Java代码来生成

第一步：编辑pom.xml增加需要使用的相关依赖和仓库

<dependencies>

...

<dependency>

<groupId>io.github.swagger2markup</groupId>

<artifactId>swagger2markup</artifactId>

<version>1.3.3</version>

<scope>test</scope>

</dependency>

</dependencies>

<repositories>

<repository>

<snapshots>

<enabled>false</enabled>

</snapshots>

<id>jcenter-releases</id>

<name>jcenter</name>

<url>http://jcenter.bintray.com</url>

</repository>

</repositories>

本身这个工具主要就临时用一下，所以这里我们把scope设置为test，这样这个依赖就不会打包到正常运行环境中去。

第二步：编写一个单元测试用例来生成执行生成文档的代码

@RunWith(SpringRunner.class)

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED\_PORT)

public class DemoApplicationTests {

@Test

public void generateAsciiDocs() throws Exception {

URL remoteSwaggerFile = new URL("http://localhost:8080/v2/api-docs");

Path outputDirectory = Paths.get("src/docs/asciidoc/generated");

// 输出Ascii格式

Swagger2MarkupConfig config = new Swagger2MarkupConfigBuilder()

.withMarkupLanguage(MarkupLanguage.ASCIIDOC)

.build();

Swagger2MarkupConverter.from(remoteSwaggerFile)

.withConfig(config)

.build()

.toFolder(outputDirectory);

}

}

以上代码内容很简单，大致说明几个关键内容：

MarkupLanguage.ASCIIDOC：指定了要输出的最终格式。除了ASCIIDOC之外，还有MARKDOWN和CONFLUENCE\_MARKUP，分别定义了其他格式，后面会具体举例。

from(remoteSwaggerFile：指定了生成静态部署文档的源头配置，可以是这样的URL形式，也可以是符合Swagger规范的String类型或者从文件中读取的流。如果是对当前使用的Swagger项目，我们通过使用访问本地Swagger接口的方式，如果是从外部获取的Swagger文档配置文件，就可以通过字符串或读文件的方式

toFolder(outputDirectory)：指定最终生成文件的具体目录位置

在执行了上面的测试用例之后，我们就能在当前项目的src目录下获得如下内容：

src

--docs

----asciidoc

------generated

--------definitions.adoc

--------overview.adoc

--------paths.adoc

--------security.adoc

可以看到，这种方式在运行之后就生成出了4个不同的静态文件。

输出到单个文件

如果不想分割结果文件，也可以通过替换toFolder(Paths.get("src/docs/asciidoc/generated")为toFile(Paths.get("src/docs/asciidoc/generated/all"))，将转换结果输出到一个单一的文件中，这样可以最终生成html的也是单一的。

#### 通过 Maven 插件来生成

除了通过上面编写Java代码来生成的方式之外，swagger2markup还提供了对应的Maven插件来使用。对于上面的生成方式，完全可以通过在pom.xml中增加如下插件来完成静态内容的生成。

<plugin>

<groupId>io.github.swagger2markup</groupId>

<artifactId>swagger2markup-maven-plugin</artifactId>

<version>1.3.3</version>

<configuration>

<swaggerInput>http://localhost:8080/v2/api-docs</swaggerInput>

<outputDir>src/docs/asciidoc/generated-by-plugin</outputDir>

<config>

<swagger2markup.markupLanguage>ASCIIDOC</swagger2markup.markupLanguage>

</config>

</configuration>

</plugin>

在使用插件生成前，需要先启动应用。然后执行插件，就可以在src/docs/asciidoc/generated-by-plugin目录下看到也生成了上面一样的adoc文件了。

#### 生成HTML

在完成了从Swagger文档配置文件到AsciiDoc的源文件转换之后，就是如何将AsciiDoc转换成可部署的HTML内容了。这里继续在上面的工程基础上，引入一个Maven插件来完成。

<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>

<backend>html</backend>

<sourceHighlighter>coderay</sourceHighlighter>

<attributes>

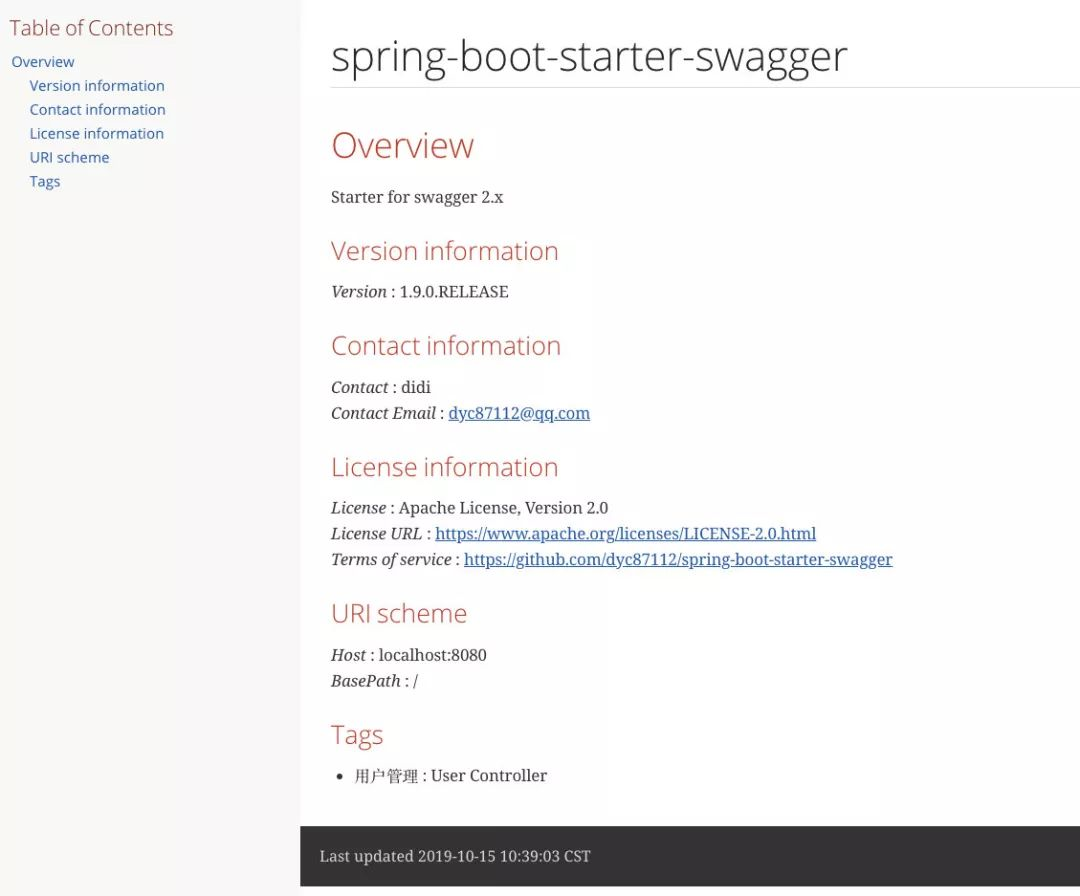
<toc>left</toc>

</attributes>

</configuration>

</plugin>

通过上面的配置，执行该插件的asciidoctor:process-asciidoc命令之后，就能在src/docs/asciidoc/html目录下生成最终可用的静态部署HTML了。在完成生成之后，可以直接通过浏览器来看查看，你就能看到类似下图的静态部署结果：



是不是感觉似曾相识呢？是的，Spring Cloud的E版之前的文档也是这样的！！！

### Markdown 与 Confluence 的支持

要生成Markdown和Confluence的方式非常简单，与上一篇中的方法类似，只需要修改一个参数即可。

生成 Markdown 和 Confluence 文档

生成方式有两种：

#### 通过Java代码来生成

只需要修改withMarkupLanguage属性来指定不同的格式以及toFolder属性为结果指定不同的输出目录。

生成markdown的代码片段：

URL remoteSwaggerFile = new URL("http://localhost:8080/v2/api-docs");

Path outputDirectory = Paths.get("src/docs/markdown/generated");

// 输出Ascii格式

Swagger2MarkupConfig config = new Swagger2MarkupConfigBuilder()

.withMarkupLanguage(MarkupLanguage.MARKDOWN)

.build();

Swagger2MarkupConverter.from(remoteSwaggerFile)

.withConfig(config)

.build()

.toFolder(outputDirectory);

生成confluence的代码片段：

URL remoteSwaggerFile = new URL("http://localhost:8080/v2/api-docs");

Path outputDirectory = Paths.get("src/docs/confluence/generated");

// 输出Ascii格式

Swagger2MarkupConfig config = new Swagger2MarkupConfigBuilder()

.withMarkupLanguage(MarkupLanguage.CONFLUENCE\_MARKUP)

.build();

Swagger2MarkupConverter.from(remoteSwaggerFile)

.withConfig(config)

.build()

.toFolder(outputDirectory);

在执行了上面的设置内容之后，我们就能在当前项目的src目录下获得如下内容：

src

--docs

----confluence

------generated

--------definitions.txt

--------overview.txt

--------paths.txt

--------security.txt

----markdown

------generated

--------definitions.md

--------overview.md

--------paths.md

--------security.md

可以看到，运行之后分别在markdown和confluence目录下输出了不同格式的转换内容。如果读者想要通过插件来生成，直接参考上一节内容，只需要修改插件配置中的swagger2markup.markupLanguage即可支持输出其他格式内容。

最后，我们一起来看看生成的Markdown和Confluence文档要怎么使用

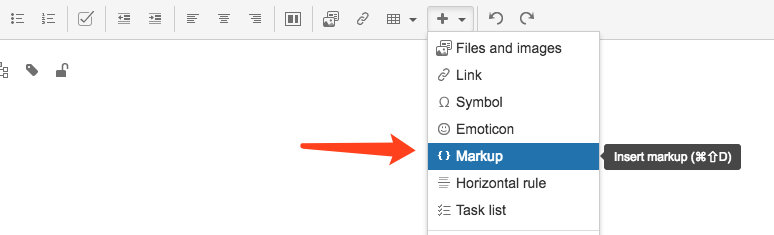
#### Markdown的部署

Markdown目前在文档编写中使用非常常见，所以可用的静态部署工具也非常多，比如：Hexo、Jekyll等都可以轻松地实现静态化部署，也可以使用一些SaaS版本的文档工具，比如：语雀等。具体使用方法，这里按照这些工具的文档都非常详细，这里就不具体介绍了。

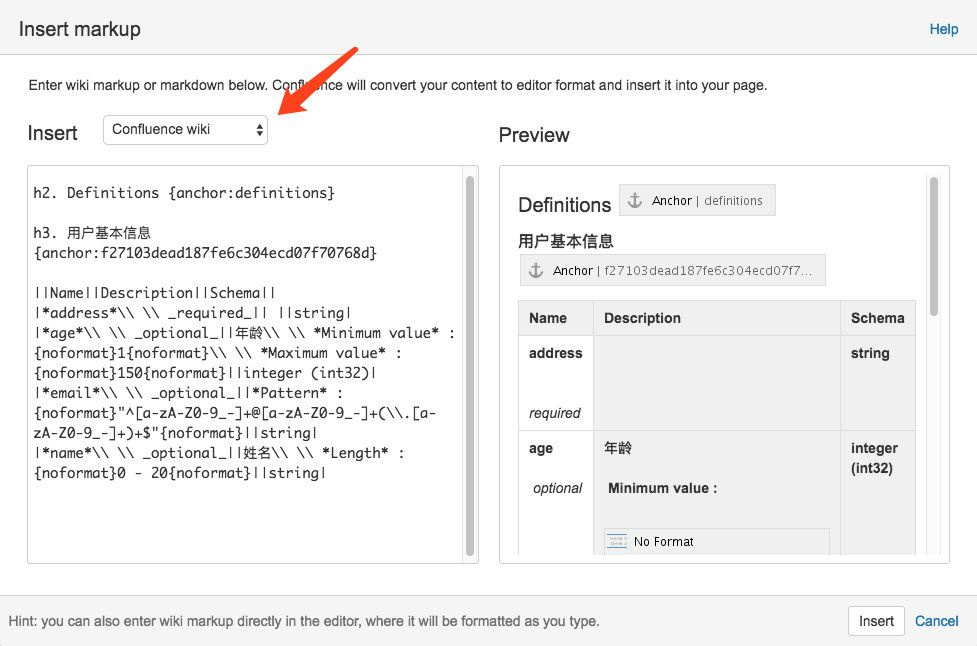
#### Confluence的部署

相信很多团队都使用Confluence作为文档管理系统，所以下面具体说说Confluence格式生成结果的使用。

第一步：在Confluence的新建页面的工具栏中选择{}Markup



第二步：在弹出框的Insert选项中选择Confluence Wiki，然后将生成的txt文件中的内容，黏贴在左侧的输入框中；此时，在右侧的阅览框可以看到如下图的效果了。



注意：所以Insert选项中也提供了Markdown格式，我们也可以用上面生成的Markdown结果来使用，但是效果并不好，所以在Confluence中使用专门的生成结果为佳。

代码示例

Github：https://github.com/dyc87112/SpringBoot-Learning/tree/2.x

Gitee：https://gitee.com/didispace/SpringBoot-Learning/tree/2.x