在线oj项目-接口文档

简介

接口文档，顾名思义就是接口的说明文档，它是我们调用接口的依据。

接口文档一般是由后端开发人员所编写的，经过和前端人员讨论之后最终确定。用来描述系统所提供接口信息的文档。接口文档包含了对接口地址、参数以及输出内容的说明，参照接口文档就能方便的知道接口的作用，以及接口如何进行调用。

作用

- **简化前端开发**：前端开发者无需担心不同接口返回不同格式的数据，他们可以编写统一的解析逻辑来处理响应。
- **易于错误处理**：接口文档定义统一的响应格式包括一个状态码或错误字段，用于指示请求是否成功以及可能的错误原因。这简化了错误处理逻辑，并允许客户端更容易地识别和处理错误
- **代码可维护性**：遵循统一的格式意味着代码更加一致，这有助于减少错误并提高代码的可维护性。当其他开发者接手项目时，他们不需要学习多种不同的响应格式。
- **文档化**：统一的响应格式可以更容易地文档化，因为你可以为整个项目创建一个通用的API文档模板。

内容

- **接口概述**：包括接口名称、接口功能、接口类别等。
- **接口地址**：接口的唯一访问地址。
- **请求方法**：定义接口的请求方式，如GET（查询）、POST（新增）、PUT（修改）、DELETE（删除）等。
- **请求参数**：定义请求时需要传递的参数，包括路径参数（Path Parameters）、查询参数（Query Parameters）、请求头（Headers）、请求体（Body）等。
- **响应数据**：定义接口返回的数据格式，包括状态码（Status Code）、消息（Message）、数据体（Data）等。（注意：包含接口请求出现错误时，返回的状态码和错误信息。不同接口格式统一、状态码含义相同）
- **请求和响应示例：**为了更好地描述接口的使用，接口文档中会提供一些具体的接口请求和响应示例，以供读者参考。

接口文档定义

> 了方便教学，我们先定义出响应数据的格式这部分是所有接口统一的，具体的接口地址、请求参数、请求和响应示例等在开发具体功能时我们我们在逐步给出定义。

响应数据定义

响应数据的格式：

```
public class R<T> {

    /**
     * 消息状态码
     */
    private int code;

    /**
     * 消息内容
     */
    private String msg;

    /**
     * 数据对象
     */
    private T data;

}
```

状态码定义

Http协议状态码

状态码	描述
1XX	信息。表示临时响应并需要请求者继续执行操作
2XX	成功。操作被成功接收并处理
3xx	重定向。表示客户端必须执行一些其他操作才能完成其请求。
4xx	客户端错误。请求包含语法错误或无法完成请求
5xx	服务器错误。这些错误可能是服务器本身的错误，而不是请求出错

但是这些状态码有时不能完全支撑我们的业务我们有时希望通过状态码说明更加详细的信息，另一方面处于安全考虑当服务器出错时我们不直接暴露底层的系统错误。

自定义状态码

状态码	类型	描述
1000	*SUCCESS*	操作成功
服务器报错		
2000	*ERROR*	服务繁忙请稍后重试（前端根据错误码显示：服务繁忙请稍后重试）
操作失败，但是服务器不存在异常		
3000	*FAILED*	操作失败
3001	*FAILED_UNAUTHORIZED*	未授权
3002	*FAILED_PARAMS_VALIDATE*	参数校验失败
3003	*FAILED_NOT_EXISTS*	资源不存在
3004	*FAILED_ALREADY_EXISTS*	资源已存在
3101	*AILED_USER_EXISTS*	用户已存在
3102	*FAILED_USER_NOT_EXISTS*	用户不存在
3103	*FAILED_LOGIN*	用户名或密码错误
3104	*FAILED_USER_BANNED*	您已被列入黑名单, 请联系管理员.

在 **com.bite.common.core.enums** 包下创建**枚举**类型命名为**ResultCode**

```
1 import lombok.AllArgsConstructor;
2 import lombok.Getter;
3
4 @Getter
5 @AllArgsConstructor
6 public enum ResultCode {
7
8     /** 定义状态码 */
```

```
9
10      //操作唱功
11      SUCCESS                     (1000, "操作成功"),
12
13      //服务器内部错误，友好提示
14      ERROR                       (2000, "服务繁忙请稍后重试"),
15
16      //操作失败，但是服务器不存在异常
17      FAILED                      (3000, "操作失败"),
18      FAILED_UNAUTHORIZED         (3001, "未授权"),
19      FAILED_PARAMS_VALIDATE      (3002, "参数校验失败"),
20      FAILED_NOT_EXISTS           (3003, "资源不存在"),
21      FAILED_ALREADY_EXISTS       (3004, "资源已存在"),
22
23      AILED_USER_EXISTS           (3101, "用户已存在"),
24      FAILED_USER_NOT_EXISTS      (3102, "用户不存在"),
25      FAILED_LOGIN                (3103, "用户名或密码错误"),
26      FAILED_USER_BANNED          (3104, "您已被列入黑名单，请联系管理员.");
27
28
29      /**
30       * 状态码
31       */
32      private int code;
33
34      /**
35       * 状态描述
36       */
37      private String msg;
38  }
```


Swagger 引入

官网：https://swagger.io/

什么是swagger

Swagger是一个接口文档生成工具，它可以帮助开发者自动生成接口文档。当项目的接口发生变更时，Swagger可以实时更新文档，确保文档的准确性和时效性。Swagger还内置了测试功能，开发者可以直接在文档中测试接口，无需编写额外的测试代码。

引入swagger

- 在oj-common下创建,oj-common-swagger子module
- 导入依赖

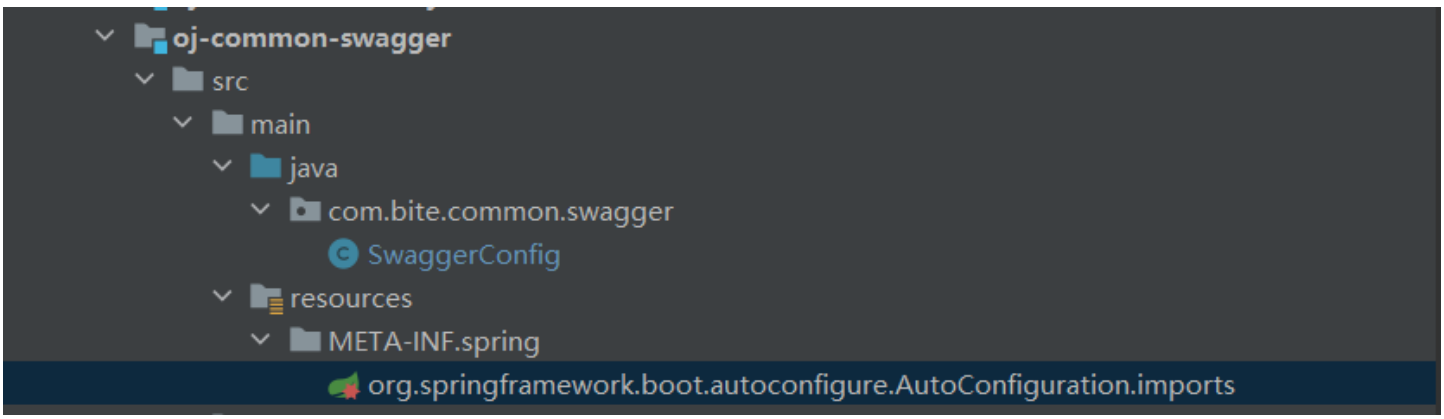
```xml
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.2.0</version>
</dependency>
```

- 录入配置

```java
import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {
    @Bean
    public OpenAPI openAPI() {
        return new OpenAPI()
                .info(new Info()
                        .title("在线oj系统")
                        .description("在线oj系统接口文档")
                        .version("v1"));
    }
}
```

- 创建org.springframework.boot.autoconfigure.AutoConfiguration.imports文件



- 具体服务引入:

```xml
<dependency>
    <groupId>com.bite</groupId>
    <artifactId>oj-common-swagger</artifactId>
    <version>${oj.common.swagger.version}</version>
</dependency>
```

基本注解

- **@Tag：**
  - **介绍：**用于给 接口 分组，用途类似于为 接口文档添加标签。
  - **用于**：方法、类、接口。
  - **常用属性：**
    - name：分组的名称
- **@Operation**
  - **介绍**：用于描述接口的操作。
  - **用于**：方法。
  - **常用属性：**
    - summary：操作的摘要信息。
    - description：操作的详细描述。
- **@Parameters**
  - **介绍**：用于指定**@Parameter**注解对象数组，描述操作的输入参数。
  - **用于**：方法。
- **@Parameter**
  - **介绍**：用于描述输入参数。
  - **用于**：方法。
  - **常用属性：**
    - name：参数的名称。
    - in：参数的位置，可以是 path、query、header、cookie 中的一种。
    - description：参数的描述。
- **@ApiResponse**
  - **介绍**：用于描述操作的响应结果.
  - **用于**：方法。
  - **常用属性：**

    - responseCode：响应的状态码。
    - description：响应的描述。
- **@Schema**
    - **介绍**：用于描述数据模型的属性。
    - **用于**：方法、类、接口。
    - **常用属性：**
        - description：响应的描述。

案例

- controller层修改：

```
@RestController
@RequestMapping("/sysUser")
@Tag(name = "管理员用户API")
public class SysUserController {


    @Operation(summary = "新增管理员", description = "根据提供的信息新增管理员用户")
    @PostMapping("/add")
    @ApiResponse(responseCode = "1000", description = "操作成功")
    @ApiResponse(responseCode = "2000", description = "服务繁忙请稍后重试")
    @ApiResponse(responseCode = "3101", description = "用户已存在")
    public R<Void> add(@RequestBody SysUserSaveDTO saveDTO) {
        return null;
    }

    @DeleteMapping("/{userId}")
    @Operation(summary = "删除用户", description = "通过用户id删除用户")
    @Parameters(value = {
            @Parameter(name = "userId", in = ParameterIn.PATH, description = "用户ID")
    })
    @ApiResponse(responseCode = "1000", description = "成功删除用户")
    @ApiResponse(responseCode = "2000", description = "服务繁忙请稍后重试")
    @ApiResponse(responseCode = "3101", description = "用户不存在")
    public R<Void> delete(@PathVariable Long userId) {
        return null;
    }

    //修改我就不演示了和新增差不多

    @Operation(summary = "用户详情", description = "根据查询条件查询用户详情")
```

```
31      @GetMapping("/detail")
32      @Parameters(value = {
33              @Parameter(name = "userId", in = ParameterIn.QUERY, description =
    "用户ID"),
34              @Parameter(name = "sex", in = ParameterIn.QUERY, description = "用户
    性别")
35      })
36      @ApiResponse(responseCode = "1000", description = "成功获取用户信息")
37      @ApiResponse(responseCode = "2000", description = "服务繁忙请稍后重试")
38      @ApiResponse(responseCode = "3101", description = "用户不存在")
39      public R<SysUserVO> detail(Long userId, @RequestParam(required = false)
    String sex) {
40          return null;
41      }
42  }
```