swagger规范

版本号	修订日期	修订内容	修订人
1	2024/3/26	1. A：模块接口文档配置规范 2. A：swagger2升级到openapi3过渡规范	陈雷
2	2024/9/12	A：全面升级到openapi3方案	陈雷

Swagger 规范

1. Swagger Config 配置规范

1.1 各个业务组件独立配置swagger config

- 每个可独立引用的业务组件单独提供swagger配置，如admin-basic、admin-file-manage、amo-logging、lcp-runtime-page等
- 每个业务组件提供的配置只需扫描本模块的controller，如lcp-runtime-page只需扫描 `com.szkingdom.koca.lcp.page.controller` 包下的接口
- 业务组件提供swagger配置时，注意docket bean名称要符合kcoa bean名称前缀规范，避免bean名称冲突
- 业务组件提供swagger配置时，注意docket groupName要唯一，建议为组件maven坐标的artifactId，如admin-basic、amo-logging、lcp-runtime-age等
- 业务组件提供swagger配置时，注意AppInfo的title要唯一，建议为组件的门户+组件中文名，如业务平台-基础管理模块、运维平台-日志监控、开发平台-页面管理
- 门户不提供swagger配置，除非门户本身提供额外接口

下面是lcp-runtime-page的swagger配置，可作参考：

```
/*
 * <p>文件名称: PageConfiguration.java</p>
 * <p>项目描述: KOCA 金证云原生平台 v0.12</p>
 * <p>公司名称: 深圳市金证科技股份有限公司</p>
 * <p>版权所有: (C) 2019-2020</p>
 */

package com.szkingdom.koca.lcp.page.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;
import org.springframework.util.StringUtils;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

/**
 * 配置类。
 * <p>配置类</p>
 *
 * @author qiuhang
 * @since 0.12 2022/10/12
 */
@Configuration("lcp-runtime-page-page-configuration")
@EnableSwagger2
public class PageConfiguration {

    /**
     * swagger配置
     * @param env 环境
```

```
     * @return Docket
     */
    @Bean("lcp-runtime-page-docket")
    public Docket docket(Environment env) {
        return new Docket(DocumentationType.SWAGGER_2).groupName("lcp-runtime-page")
            .apiInfo(appInfo("页面管理", env)).select()
            .apis(RequestHandlerSelectors.basePackage("com.szkingdom.koca.lcp.page.controller"))
            .paths(PathSelectors.any()).build();
    }

    public ApiInfo appInfo(String modelName, Environment env) {
        String serverPort = env.getProperty("server.port");
        String contextPath = env.getProperty("server.servlet.context-path");
        String version = env.getProperty("spring.application.version");

        if (!StringUtils.hasText(contextPath)) {
            contextPath = "/";
        }
        String hostAddress = "localhost";

        String url = String.format("http://%s:%s%s/doc.html", hostAddress, serverPort, contextPath);
        return new ApiInfoBuilder().title(modelName + " API文档").description(modelName + " API文档").termsOfServiceUrl(url)
                .contact(new Contact("chenlei", url, "chenlei@szkingdom.com")).version(version).build();
    }

}
```

1.2 每个controller必须指定分组

- swagger可以为每个controller指定多个分组，接口管理场景下，优先使用第一个分组
- 不允许顶用空分组，即建议不要定义全局分组，因为全局分组定义后，如果没有在任何controller使用该分组，会导致空分组
- tags需要使用中文名以增强可读性

swagger2

使用 `@Api` 注解的tags属性指定分组

```
@Api(tags = "页面管理")
public class Page2Controller {
}
```

swagger3

使用 `@Tag` 注解指定分组

```
@Tag(name = "页面管理")
public class Page2Controller {
}
```

1.3 每个接口必须指定名称

- 接口必须指定名称，名称建议为中文名，增强可读性
- 接口建议指定描述

swagger2

使用 `@ApiOperation` 注解，value属性指定接口名称，notes属性指定接口描述

```
@PostMapping("group/tree")
@ApiOperation(value = "查询页面分组树", notes = "返回所有分组，不分页")
```

```
public ListResult<GroupTreeDto> groupTree(@Valid @RequestBody QueryGroupVo vo) {
    return new ListResult<>(page2Service.groupPageTree(vo));
}
```

swagger3

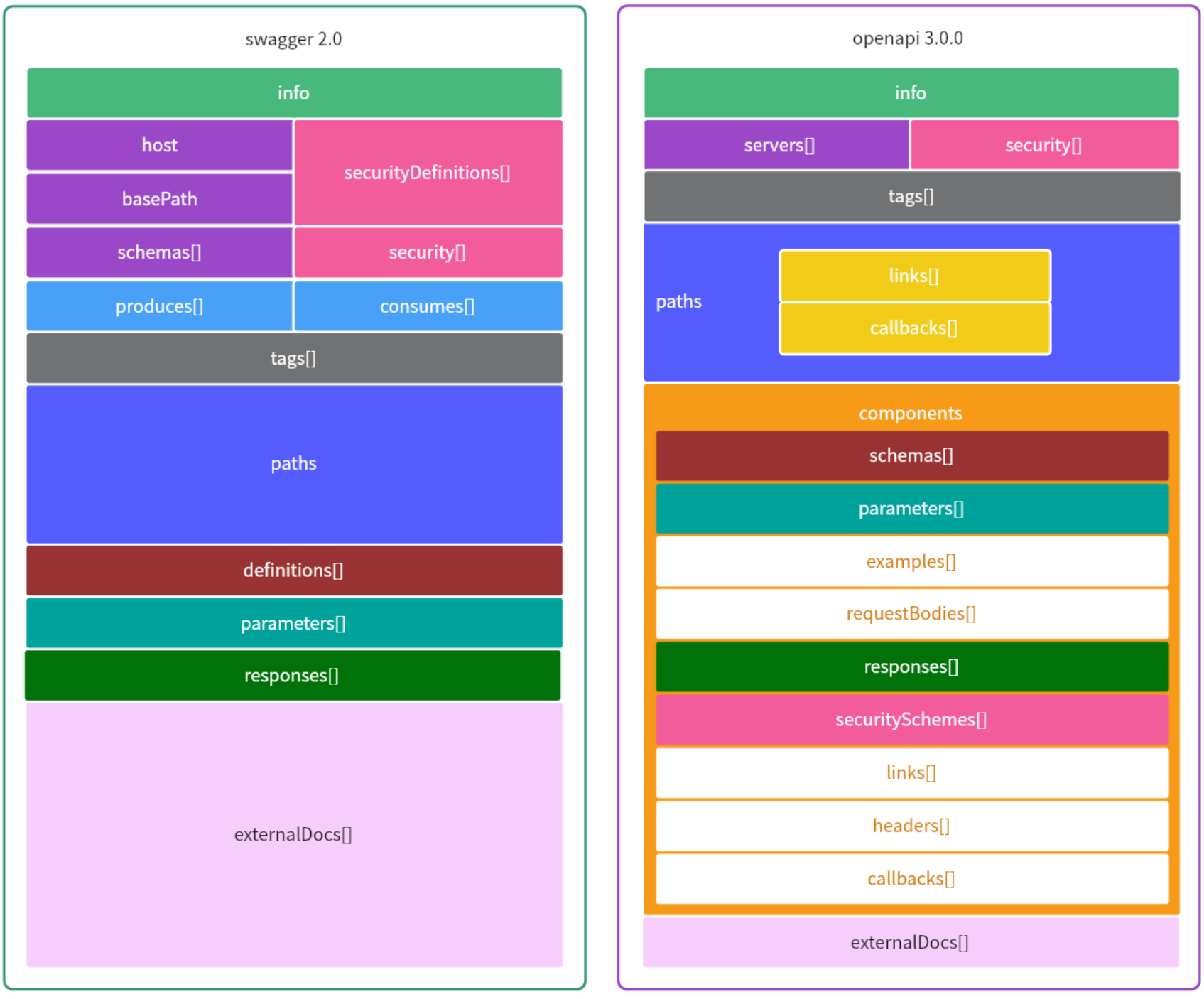
使用 `@Operation` 注解，summary属性指定接口名称，description属性指定接口描述

```
@PostMapping("group/tree")
@Operation(summary = "查询页面分组树", description = "返回所有分组，不分页")
public ListResult<GroupTreeDto> groupTree(@Valid @RequestBody QueryGroupVo vo) {
    return new ListResult<>(page2Service.groupPageTree(vo));
}
```

2. Swagger2和Swagger3对比

Swagger3 在 Swagger2 的基础上做了部分改进，在核心定义不变的情况下， Swagger3 的结构更加清晰，扩展性更高，但是 Swaager3 不能直接兼容 Swagger2 的定义文件，需要通过一次转换才能实现兼容，即 Swagger2 的定义完全可以在 Swagger3 中描述，但部分定义结构发生了变化。

Swaggre2 和 Swagger3 都可以用 json 或 yaml 格式定义，他们在结构上的差异如下：



2.1 springfox swagger

1. 引入 springfox-boot-starter 3.x 版本后，默认包含 swagger2 和 swagger3 的注解
2. 不管使用 swagger2 还是 swagger3 的注解，都可以通过 api/v2/api-docs 和 api/v3/api-docs 获取对应版本的 swagger 定义文件

3. 引入 springfox-boot-starter 3.x 和 springfox-swagger-ui 3.x 版本后，不管使用都会使用 swagger2 还是 swagger3 的注解，swagger-ui 都会访问 api/v3/api-docs 获取元数据
4. 如果引用的 swagger-ui 2.x 版本，会访问 api/v2/api-docs 接口，即 swagger3 兼容两个版本的UI，如果老项目使用的 swagger2 注解，在pom中引入 swagger3 相关依赖后，不需要做任何调整即可完成升级
5. 两个版本的注解可以混用， swagger 解析时可以自动适配
6. swagger3 的注解和模型单独在 io.seagger.core.v3 包下，其他的功能位置不变

2.2 Swagger2和Swagger3注解差异

功能	Swagger2	Swagger3	相同
全局参数定义	`@SwaggerDefinition`	`@OpenAPIDefinition`	❌
服务器	-	`@Server/@Servers/@ServerVariable`	❌
联系人	`@Contact`	`@Contact`	✅
为controller设置Tag	`@Api`	`@Tag/@Tags`	❌
Tag定义	`@Tag`	`@Tag/@Tags`	❌
在controller上配置参数	`@ApiImplicitParam/@ApiImplicitParams/@ApiParam`	`@Parameter/@Parameters`	❌
文档描述	`@Info`	`@Info`	✅
文档License	`@License`	`@License`	✅
扩展属性	`@Extension/@ExtensionProperty`	`@Extension/@Extensions/@ExtensionProperty`	❌
参数示例	`@Example/@ExampleProperty`	`@ExampleObject`	❌
外部文档	`@ExternalDocs`	`@ExternalDocumentation`	❌
接口描述	`@ApiOperation`	`@Operation`	❌
body参数	`@ApiModel/`	`@Schema/@RequestBody/@Content/@ArraySchema`	❌
response	`@ApiResponse/@ApiResponses`	`@ApiResponse/@ApiResponses`	❌
heder	`@ResponseHeader`	`@Header`	❌
对象参数属性	`@ApiModelProperty`	`@Schema`	❌
隐藏属性	`@ApiModelProperty(hidden = true)`	`@Hidden`	❌
隐藏接口	`@ApiOperation(hidden = true)`	`@Hidden`	❌
认证定义	`@SecurityDefinition`	`@SecurityScheme/@SecuritySchemes`	❌
接口认证	`@Authorization`	`@SecurityRequirement/@SecurityRequirements`	❌
Oauth2认证scope	`@Scope/@AuthorizationScope`	`@OAuthScope`	❌
其他认证	`@OAuth2Definition/@BasicAuthDefinition/@ApiKeyAuthDefinition`	`@OAuthFlow/@OAuthFlows`	❌
回调	-	`@Callback/@Callbacks`	❌
接口链	-	`@Link/@LinkParameter`	❌
其他	-	`@Encoding/@DiscriminatorMapping`	❌

升级到Openapi3

Swagger & OpenApi 1.0、2.0、3.0、3.1

> `Swagger`它最初由Tony Tam在2011年创建，并在之后被SmartBear Software公司收购。

1. `Swagger 1.x` 阶段（2011-2014年）： `Swagger`最初是一个简单的`API`文档生成工具，通过对`JAX-RS`和`Jersey`注解的支持自动生成`API`文档，使得`API`文档的维护变得更加容易。 在这个阶段， `Swagger`还没有完全成熟，只能支持基本的`API`描述和文档生成。发展阶段
2. `Swagger 2.x` 阶段（2014-2017年）：在`Swagger 2.x`阶段， `Swagger`发生了重大变化。它不再仅仅是一个文档生成工具，而是一个完整的`API`开发和管理平台。`Swagger 2.x`加 入了强大的注解支持，可以描述`API`的细节信息，如请求参数、返回类型等，以及定义`RESTful API`的元数据，如`API`描述、标签等。 此外， `Swagger 2.x`还引入了`OpenAPI`规范，在`API`定义方面有了更严格的标准和规则。事实标准
3. `OpenAPI`阶段（2017-至今）（也被称为`Swagger 3.x`）：在2017年， `Swagger 2.x`的规范成为了`Linux`基金会旗下的`OpenAPI`规范。这标志着`Swagger`从一款工具演变成为了一个开放且标准的`API`定义框架。`OpenAPI`规范不仅继承了`Swagger 2.x`的特性，还提供了更加全面和严格的`API`定义规范，并且扩展了对非`RESTful API`的支持。公认标准

为什么升级?

1. 随着时间的推移，`Swagger2.x`终究成为历史，`springfox-boot-starter`的坐标从`3.0.0`版本（2020年7月14日）开始就一直没有更新；`springfox-swagger2`坐标和`springfox-boot-start`也是一样的。
2. `springfox`也支持`openapi3`，但在兼容性、灵活性上有些问题，比如`swagger2 swagger3`同时在`classpath`中时，可能导致`group`重复、`tag`混乱等问题。
3. `springdoc`有更加先进的技术架构和更好的扩展性，除了可以生成Swagger UI风格的接口文档，还提供了ReDoc的文档渲染方式，可以自动注入OpenAPI规范的JSON描述文件，支持OAuth2、JWT等认证机制，并且**支持全新的OpenAPI 3.0规范，**同时还拥有更为完善的开发文档和社区支持。

升级顺序

1. `core/base`：剔除`springfox-swagger2`依赖，统一`springdoc`和`sawgger-core`版本，在`dependencies bom`中声明版本，子项目无需关心版本。
2. `admin`：剔除`springfox-swagger2`依赖，替换为`dependencies bom`中的`springdoc`或`swagger-core`。
3. `amo/lcp`：剔除`springfox-swagger2`依赖，替换为`dependencies bom`中的`springdoc`或`swagger-core`。

升级方案(spring-boot 2.7.x)

`core/base`

1. 在`koca-misc-dependencies.xml`中删除`springfox-swagger-ui`和`springfox-swagger2`依赖，删除`<springfox.version>3.0.0</springfox.version>`
2. 在`koca-misc-dependencies/pom.xml`中添加如下`property`和依赖：

   注意：`openapi3`相关的`maven`依赖，`groupId`已经变更为`io.swagger.core.v3`，不再是`io.swagger`

```
<springdoc.version>1.8.0</springdoc.version>
<swagger-core.version>2.2.20</swagger-core.version>


<!-- ui 用于bootapp 如admin门户/amo门户 -->
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>${springdoc.version}</version>
</dependency>

<!-- 基础依赖 用于业务组件 如koca-admin-user/koca-lcp-runtime-page -->
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-common</artifactId>
    <version>${springdoc.version}</version>
</dependency>

<!-- 包括swagger-models和swagger-annotations 用于公共模块 如core/base/common/util等 -->
<dependency>
    <groupId>io.swagger.core.v3</groupId>
    <artifactId>swagger-core</artifactId>
    <version>${swagger-core.version}</version>
</dependency>
```

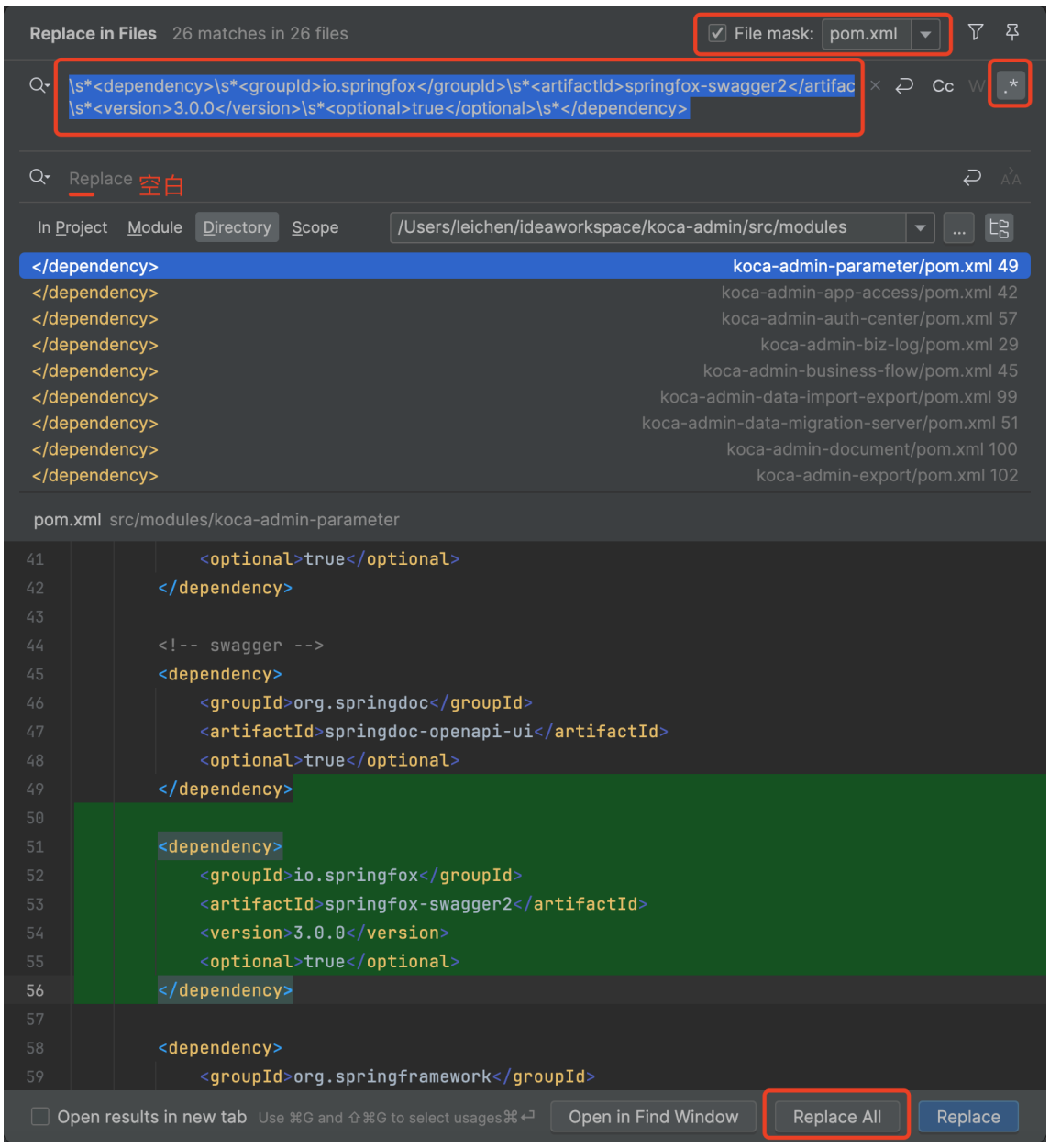
3. 修改`core/base`的其他模块，引入`koca-misc-dependencies`中对应的依赖，部分模块可能需要重新适配，比如`koca-bex-swagger`。

`admin/lcp/amo`

官方文档：pringdoc-openapi

1. 移除所有模块中的`springfox-swagger2`和`springdoc-openapi-ui`依赖(文件夹搜索替换)。

```
\s<dependency>\s<groupId>io.springfox</groupId>\s<artifactId>springfox-swagger2</artifactId>\s<version>3.0.0</version>\s<optional>true</optional>\s</dependency>
```



2. 在`modules/pom.xml`或者所有业务组件的公共模块中引入依赖：

> 如果`modules/pom.xml`或公共模块中引入之后，担心影响范围过大，可通过文件夹搜索删除或替换。

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-common</artifactId>
</dependency>
```

3. `reimport`后，所有使用了`swagger`注解和`Docket`配置的类全部报错，然后通过查找替换的方式完成修改，先修改`swagger`注解：

> 下面为`lcp`升级过程，可作参考。

步骤	搜索	替换
1	`import io.swagger.annotations.ApiModel;\nimport io.swagger.annotations.ApiModelProperty;`	`import io.swagger.v3.oas.annotations.media.Schema;`
2	`import io.swagger.annotations.ApiModelProperty;`	`import io.swagger.v3.oas.annotations.media.Schema;`
3	`@ApiModel(name`	`@ApiModel(description`
4	`@ApiModel("`	`@Schema(description = "`
5	`@ApiModelProperty(value =`	`@Schema(description =`
6	`import io.swagger.annotations.Api;`	`import io.swagger.v3.oas.annotations.tags.Tag;`
7	`import io.swagger.annotations.ApiOperation;`	`import io.swagger.v3.oas.annotations.Operation;`
8	`import io.swagger.annotations.ApiParam;`	`import io.swagger.v3.oas.annotations.Parameter;`
9	`@Api(tags =`	`@Tag(name =`
10	`@Api("`	`@Tag(name = "`
11	`@ApiOperation\(value\s*=\s*("[^"]*"), notes\s*=\s*("[^"]*")\)`	`@Operation(summary = $1, description = $2)`
12	`@ApiParam\(value\s*=\s*("[^"]*"), name\s*=\s*("[^"]*")\)`	`@Parameter(name = $1, description = $2)`

步骤	搜索	替换
13	其他剩余规则	其他替换方案
14	包含不正确 `import` 顺序的多行 `import`	正确的 `import` 顺序，一般是6/7/8三个步骤的顺序不正确

4. 修改配置类，下面为 `lcp` 实现方案，可作参考。

   每个组件新建 `XXOpenapiConfiguration/XXSwaggerConfiguration` 类，如 `PageSwaggerConfiguration`，然后将其他配置类中的 `swagger` 配置全部移动到该类，最后将内容修改为：

```
1  @Configuration("lcp-runtime-page-page-swagger-configuration")
2  @ConditionalOnProperty(value = "springdoc.api-docs.enabled", matchIfMissing = true)
3  public class PageSwaggerConfiguration {
4
5      /**
6       * 创建分组
7       *
8       * @return 分组
9       */
10     @Bean("lcp-runtime-page-create-rest-api-group")
11     @ConditionalOnProperty(value = "springdoc.api-docs.groups.enabled", matchIfMissing = true)
12     public GroupedOpenApi createGroup() {
13         return GroupedOpenApi.builder().group("lcp-runtime-page")
14             .packagesToScan("com.szkingdom.koca.lcp.page.controller").pathsToMatch("/**").build();
15     }
16
17 }
```

   关键点：

   - `line 2`：使用 `springdoc.api-docs.enabled` 判断是否加载该配置类。
   - `line 11`：使用 `springdoc.api-docs.groups.enabled` 判断是否加载分组。
   - `line 12~15`：`GroupedOpenApi` 可指定的属性比 `springfox` 的 `Docket` 少了很多，不再是每个分组都可指定 `server/contact/title/url` 等。

   在 `spring.factories` 中添加改配置类:

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.szkingdom.koca.lcp.page.config.PageConfiguration,\
com.szkingdom.koca.lcp.page.config.PageSwaggerConfiguration
```

5. `bootapp` 或 `koca-application` 中修改 `pom.xml`，删除 `springfox` 相关依赖和 `knif4j` 旧版本依赖，添加 `springdoc-openapi-ui`：

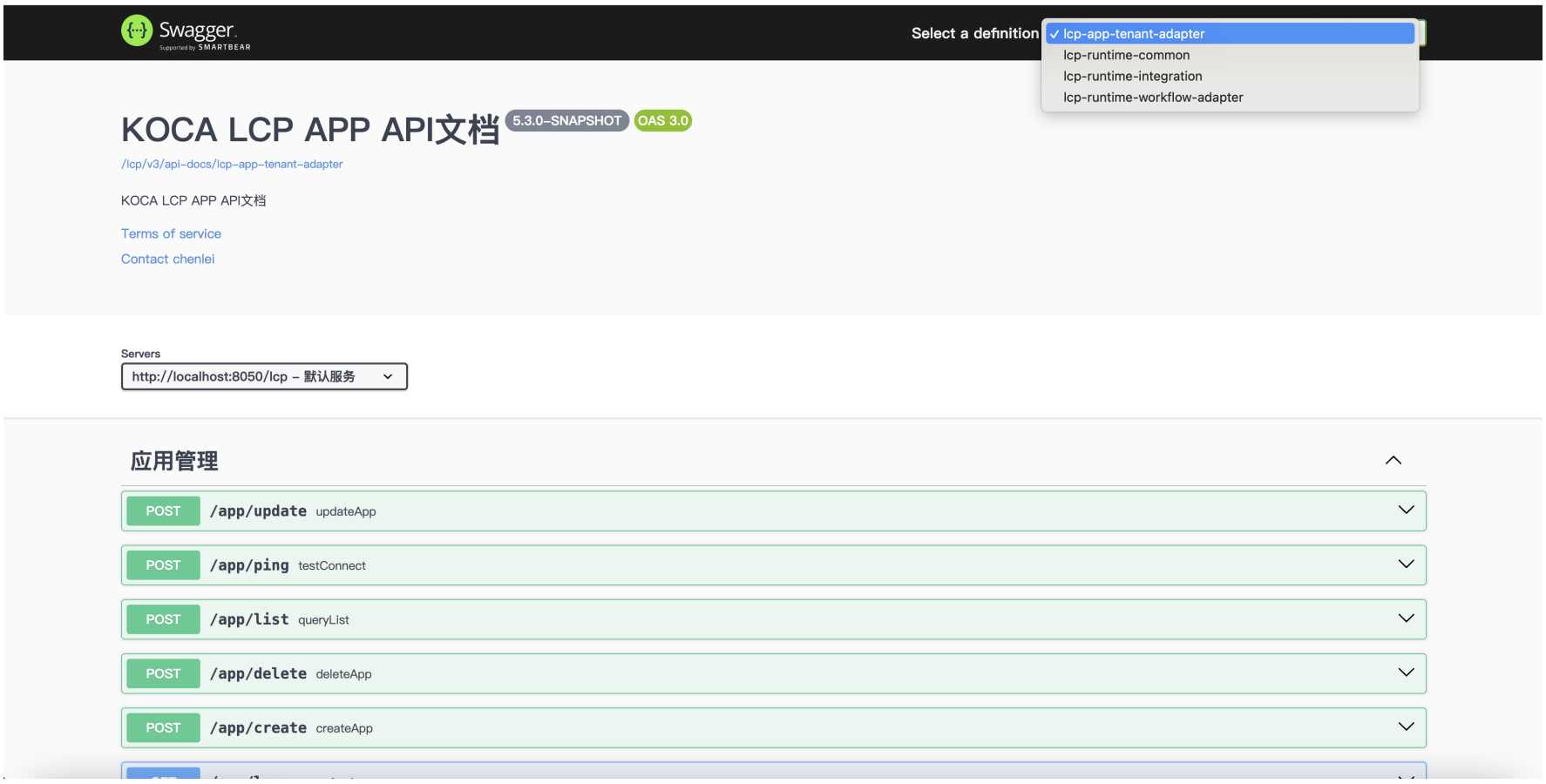
```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
</dependency>
```

   在启动类中添加 `OpeAPI` 配置或新建配置类：

```
@Configuration
@ConditionalOnProperty(value = "springdoc.api-docs.enabled", matchIfMissing = true)
public class OpenApiConfiguration {

    @Bean
    public OpenAPI createLcpApiRestApi(Environment env) {
        String serverPort = env.getProperty("server.port");
        String contextPath = env.getProperty("server.servlet.context-path");
        String version = env.getProperty("spring.application.version");

        if (!StringUtils.hasText(contextPath)) {
            contextPath = "/";
        }
        String hostAddress = "localhost";
```

```
        String url = String.format("http://%s:%s%s/doc.html", hostAddress, serverPort, contextPath);
        Info info = new Info().title("KOCA LCP API文档").description("KOCA LCP API文档").termsOfService(url)
            .contact(new Contact().name("chenlei").email("chenlei@szkingdom.com")).version(version);
        return new OpenAPI().info(info).servers(Collections.singletonList(
            new Server().url(String.format("http://%s:%s%s", hostAddress, serverPort, contextPath))
                .description("默认服务")));
    }

}
```

启动项目，访问`http://ip:port/context/swagger-ui/index.html`

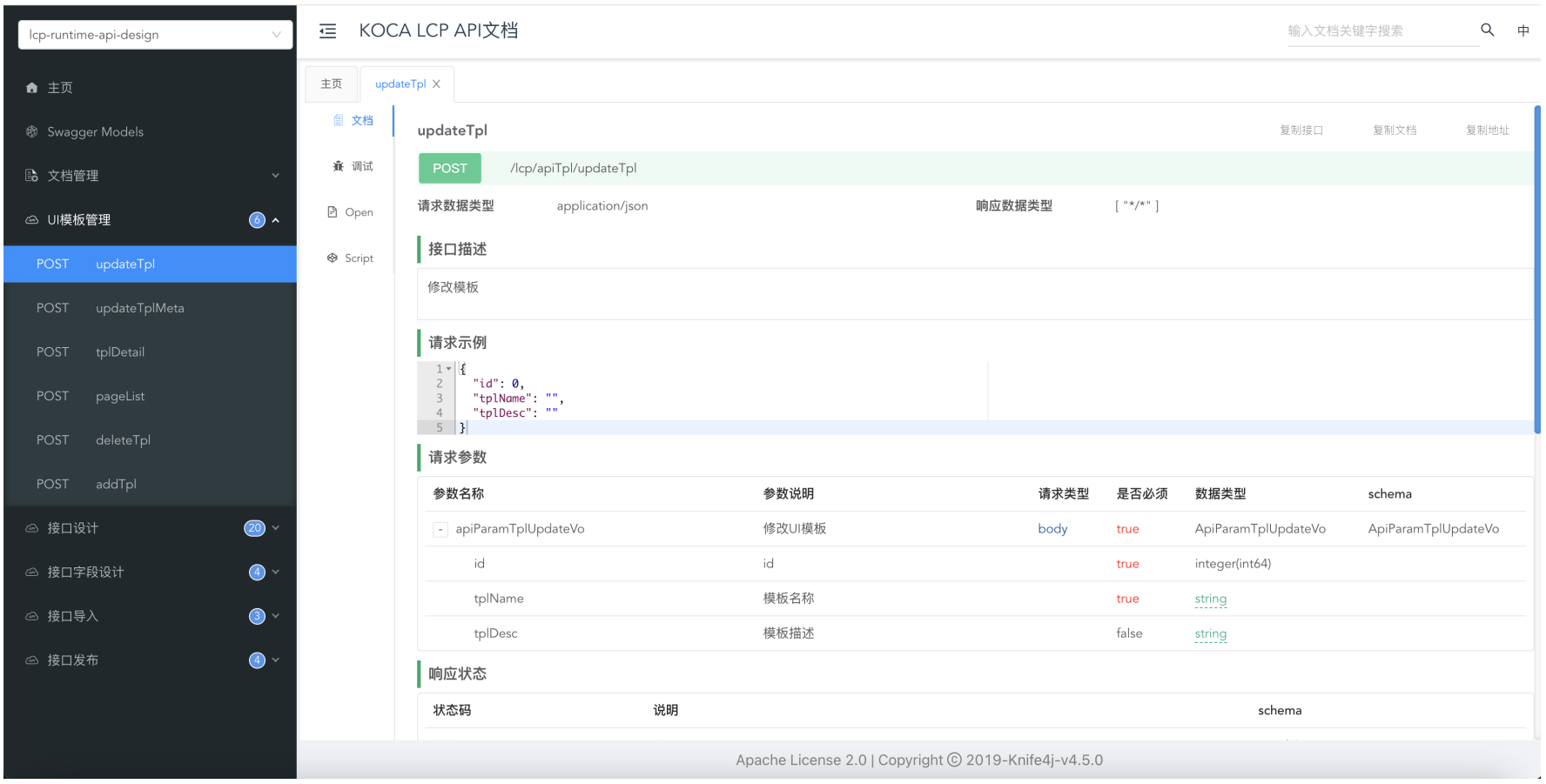


6. 如果想要使用`knife4j`，在5的基础上，新增依赖(**注意不是二选一，是两个都要**):

```
1  <dependency>
2      <groupId>com.github.xiaoymin</groupId>
3      <artifactId>knife4j-openapi3-spring-boot-starter</artifactId>
4      <version>4.5.0</version>
5      <exclusions>
6          <exclusion>
7              <artifactId>springdoc-openapi-ui</artifactId>
8              <groupId>org.springdoc</groupId>
9          </exclusion>
10     </exclusions>
11 </dependency>
```

- 两个都要：平台的`springdoc`版本和最新的`knife4j-openapi3-spring-boot-starter`不一样，需要覆盖。
- `line 5~10`：排除`knife4j-openapi3-spring-boot-starter`中`springdoc-openapi-ui`的依赖。

启动项目，访问`http://ip:port/context/doc.html`



常见问题

1. 问题：`ClassNotFound/NoSuchMethod`，搜索又有对应的包或方法

   方案：一般是包冲突，注意调整各个模块中`springdoc`和`swagger-core`的版本

2. 问题：提示`Docket/EnableSeagger2`等类不存在，打开对应的`java/class`文件飘红

   方案：某些子模块没有升级，或者依赖的其他门户模块没有升级，如果是本门户的模块，升级即可，如果是其他门户的模块，除了通知对应负责人升级并`deploy`外，还可在`bootapp/pom.xml`中添加如下依赖解决，但注意依赖模块升级并发布后需删除：

   ```
   <!-- TODO: DELETE START: 临时解决swagger2没有升级到openapi3的模块报错问题 -->
   <dependency>
       <groupId>io.springfox</groupId>
       <artifactId>springfox-swagger2</artifactId>
       <version>3.0.0</version>
       <exclusions>
           <exclusion>
               <artifactId>swagger-annotations</artifactId>
               <groupId>io.swagger.core.v3</groupId>
           </exclusion>
           <exclusion>
               <artifactId>swagger-annotations</artifactId>
               <groupId>io.swagger</groupId>
           </exclusion>
       </exclusions>
   </dependency>
   ```

3. 问题：生产环境`openapi`接口暴露问题

   方案：生产环境的配置文件添加`springdoc.api-docs.enabled=false`，更多配置请参考 springdoc-openapi properties