5.4 knife4j介绍

knife4j是为Java MVC框架集成Swagger生成Api文档的增强解决方案,前身是swagger-bootstrap-ui,取名knife4j是希望它能像一把匕首一样小巧,轻量,并且功能强悍!其底层是对Springfox的封装，使用方式也和Springfox一致，只是对接口文档UI进行了优化。

核心功能：

- **文档说明**：根据Swagger的规范说明，详细列出接口文档的说明，包括接口地址、类型、请求示例、请求参数、响应示例、响应参数、响应码等信息，对该接口的使用情况一目了然。
- **在线调试**：提供在线接口联调的强大功能，自动解析当前接口参数,同时包含表单验证，调用参数可返回接口响应内容、headers、响应时间、响应状态码等信息，帮助开发者在线调试。

5.5 knife4j入门案例

第一步：创建maven工程knife4j_demo并配置pom.xml文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.2.RELEASE</version>
        <relativePath/>
    </parent>
    <groupId>cn.itcast</groupId>
    <artifactId>knife4j_demo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>com.github.xiaoymin</groupId>
            <artifactId>knife4j-spring-boot-starter</artifactId>
            <version>2.0.1</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
    </dependencies>
</project>
```

第二步： 创建实体类User和Menu

```java
package cn.itcast.entity;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@Data
@ApiModel(description = "用户实体")
public class User {
    @ApiModelProperty(value = "主键")
    private int id;
    @ApiModelProperty(value = "姓名")
    private String name;
    @ApiModelProperty(value = "年龄")
    private int age;
    @ApiModelProperty(value = "地址")
    private String address;
}
```

```java
package cn.itcast.entity;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

@Data
@ApiModel(description = "菜单实体")
public class Menu {
    @ApiModelProperty(value = "主键")
    private int id;
    @ApiModelProperty(value = "菜单名称")
    private String name;
}
```

第三步：创建UserController和MenuController

```java
package cn.itcast.controller.user;

import cn.itcast.entity.User;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiImplicitParam;
import io.swagger.annotations.ApiImplicitParams;
import io.swagger.annotations.ApiOperation;
import org.springframework.web.bind.annotation.*;
import java.util.ArrayList;
import java.util.List;

@RestController
@RequestMapping("/user")
@Api(tags = "用户控制器")
public class UserController {
    @GetMapping("/getUsers")
    @ApiOperation(value = "查询所有用户", notes = "查询所有用户信息")
    public List<User> getAllUsers(){
        User user = new User();
        user.setId(100);
        user.setName("itcast");
        user.setAge(20);
        user.setAddress("bj");
        List<User> list = new ArrayList<>();
        list.add(user);
        return list;
    }

    @PostMapping("/save")
    @ApiOperation(value = "新增用户", notes = "新增用户信息")
    public String save(@RequestBody User user){
        return "OK";
    }

    @PutMapping("/update")
    @ApiOperation(value = "修改用户", notes = "修改用户信息")
    public String update(@RequestBody User user){
        return "OK";
    }

    @DeleteMapping("/delete")
    @ApiOperation(value = "删除用户", notes = "删除用户信息")
    public String delete(int id){
        return "OK";
    }

    @ApiImplicitParams({
           @ApiImplicitParam(name = "pageNum", value = "页码",
                        required = true, type = "Integer"),
        @ApiImplicitParam(name = "pageSize", value = "每页条数",
```

```
                        required = true, type = "Integer"),
    })
    @ApiOperation(value = "分页查询用户信息")
    @GetMapping(value = "page/{pageNum}/{pageSize}")
    public String findByPage(@PathVariable Integer pageNum,
                             @PathVariable Integer pageSize) {
        return "OK";
    }
}
```

```java
package cn.itcast.controller.menu;

import cn.itcast.entity.Menu;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiImplicitParam;
import io.swagger.annotations.ApiImplicitParams;
import io.swagger.annotations.ApiOperation;
import org.springframework.web.bind.annotation.*;
import java.util.ArrayList;
import java.util.List;

@RestController
@RequestMapping("/menu")
@Api(tags = "菜单控制器")
public class MenuController {
    @GetMapping("/getMenus")
    @ApiOperation(value = "查询所有菜单", notes = "查询所有菜单信息")
    public List<Menu> getMenus(){
        Menu menu = new Menu();
        menu.setId(100);
        menu.setName("itcast");
        List<Menu> list = new ArrayList<>();
        list.add(menu);
        return list;
    }

    @PostMapping("/save")
    @ApiOperation(value = "新增菜单", notes = "新增菜单信息")
    public String save(@RequestBody Menu menu){
        return "OK";
    }

    @PutMapping("/update")
    @ApiOperation(value = "修改菜单", notes = "修改菜单信息")
    public String update(@RequestBody Menu menu){
        return "OK";
    }

    @DeleteMapping("/delete")
    @ApiOperation(value = "删除菜单", notes = "删除菜单信息")
    public String delete(int id){
        return "OK";
    }

    @ApiImplicitParams({
        @ApiImplicitParam(name = "pageNum", value = "页码",
                          required = true, type = "Integer"),
        @ApiImplicitParam(name = "pageSize", value = "每页条数",
                          required = true, type = "Integer"),
    })
```

```
    @ApiOperation(value = "分页查询菜单信息")
    @GetMapping(value = "page/{pageNum}/{pageSize}")
    public String findByPage(@PathVariable Integer pageNum,
                             @PathVariable Integer pageSize) {
        return "OK";
    }
}
```

第四步：创建配置属性类SwaggerProperties

```java
package cn.itcast.config;

import lombok.*;
import org.springframework.boot.context.properties.ConfigurationProperties;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/*
*配置属性类，用于封装接口文档相关属性，从配置文件读取信息封装成当前对象
*/

@Data
@ConfigurationProperties(prefix = "pinda.swagger")
public class SwaggerProperties {
    private String title = "在线文档"; //标题
    private String group = ""; //自定义组名
    private String description = "在线文档"; //描述
    private String version = "1.0"; //版本
    private Contact contact = new Contact(); //联系人
    private String basePackage = "com.itheima.pinda"; //swagger会解析的包路径
    private List<String> basePath = new ArrayList<>(); //swagger会解析的url规则
    private List<String> excludePath = new ArrayList<>();//在basePath基础上需要排除的
url规则
    private Map<String, DocketInfo> docket = new LinkedHashMap<>(); //分组文档
    public String getGroup() {
        if (group == null || "".equals(group)) {
            return title;
        }
        return group;
    }
    @Data
    public static class DocketInfo {
        private String title = "在线文档"; //标题
        private String group = ""; //自定义组名
        private String description = "在线文档"; //描述
        private String version = "1.0"; //版本
        private Contact contact = new Contact(); //联系人
        private String basePackage = ""; //swagger会解析的包路径
        private List<String> basePath = new ArrayList<>(); //swagger会解析的url规则
        private List<String> excludePath = new ArrayList<>();//在basePath基础上需要排除
的url
        public String getGroup() {
            if (group == null || "".equals(group)) {
                return title;
            }
            return group;
        }
    }
```

```
    @Data
    public static class Contact {
        private String name = "pinda"; //联系人
        private String url = ""; //联系人url
        private String email = ""; //联系人email
    }
}
```

第五步：创建application.yml文件

```yaml
server:
  port: 7788
pinda:
  swagger:
    enabled: true #是否启用swagger
    docket:
      user:
        title: 用户模块
        base-package: cn.itcast.controller.user
      menu:
        title: 菜单模块
        base-package: cn.itcast.controller.menu
```

第六步：创建配置类SwaggerAutoConfiguration

```java
package cn.itcast.config;

import com.google.common.base.Predicate;
import com.google.common.base.Predicates;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.BeanFactoryAware;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.config.ConfigurableBeanFactory;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.bind.annotation.RequestMethod;
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;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

@Configuration
@ConditionalOnProperty(name = "pinda.swagger.enabled", havingValue = "true",
                       matchIfMissing = true)
@EnableSwagger2
@EnableConfigurationProperties(SwaggerProperties.class)
public class SwaggerAutoConfiguration implements BeanFactoryAware {
    @Autowired
    SwaggerProperties swaggerProperties;
    private BeanFactory beanFactory;
    @Bean
    @ConditionalOnMissingBean
    public List<Docket> createRestApi(){
        ConfigurableBeanFactory configurableBeanFactory =
                                            (ConfigurableBeanFactory) beanFactory;
        List<Docket> docketList = new LinkedList<>();
        // 没有分组
        if (swaggerProperties.getDocket().isEmpty()) {
            Docket docket = createDocket(swaggerProperties);
            configurableBeanFactory.registerSingleton(swaggerProperties.getTitle(),
                                                      docket);
            docketList.add(docket);
            return docketList;
        }
```

```
        // 分组创建
        for (String groupName : swaggerProperties.getDocket().keySet()){
            SwaggerProperties.DocketInfo docketInfo =
                swaggerProperties.getDocket().get(groupName);
            ApiInfo apiInfo = new ApiInfoBuilder()
                    //页面标题
                    .title(docketInfo.getTitle())
                    //创建人
                    .contact(new Contact(docketInfo.getContact().getName(),
                            docketInfo.getContact().getUrl(),
                            docketInfo.getContact().getEmail()))
                    //版本号
                    .version(docketInfo.getVersion())
                    //描述
                    .description(docketInfo.getDescription())
                    .build();

            // base-path处理
            // 当没有配置任何path的时候，解析/**
            if (docketInfo.getBasePath().isEmpty()) {
                docketInfo.getBasePath().add("/**");
            }
            List<Predicate<String>> basePath = new ArrayList<>();
            for (String path : docketInfo.getBasePath()) {
                basePath.add(PathSelectors.ant(path));
            }

            // exclude-path处理
            List<Predicate<String>> excludePath = new ArrayList<>();
            for (String path : docketInfo.getExcludePath()) {
                excludePath.add(PathSelectors.ant(path));
            }

            Docket docket = new Docket(DocumentationType.SWAGGER_2)
                    .apiInfo(apiInfo)
                    .groupName(docketInfo.getGroup())
                    .select()
                    //为当前包路径

.apis(RequestHandlerSelectors.basePackage(docketInfo.getBasePackage()))

.paths(Predicates.and(Predicates.not(Predicates.or(excludePath)),Predicates.or(basePat
h)))
                    .build();
            configurableBeanFactory.registerSingleton(groupName, docket);
            docketList.add(docket);
        }
        return docketList;
    }

    //构建 api文档的详细信息
```

```
    private ApiInfo apiInfo(SwaggerProperties swaggerProperties) {
        return new ApiInfoBuilder()
                //页面标题
                .title(swaggerProperties.getTitle())
                //创建人
                .contact(new Contact(swaggerProperties.getContact().getName(),
                                     swaggerProperties.getContact().getUrl(),
                                     swaggerProperties.getContact().getEmail()))
                //版本号
                .version(swaggerProperties.getVersion())
                //描述
                .description(swaggerProperties.getDescription())
                .build();
    }

    //创建接口文档对象
    private Docket createDocket(SwaggerProperties swaggerProperties) {
        //API 基础信息
        ApiInfo apiInfo = apiInfo(swaggerProperties);

        // base-path处理
        // 当没有配置任何path的时候，解析/**
        if (swaggerProperties.getBasePath().isEmpty()) {
            swaggerProperties.getBasePath().add("/**");
        }
        List<Predicate<String>> basePath = new ArrayList<>();
        for (String path : swaggerProperties.getBasePath()) {
            basePath.add(PathSelectors.ant(path));
        }

        // exclude-path处理
        List<Predicate<String>> excludePath = new ArrayList<>();
        for (String path : swaggerProperties.getExcludePath()) {
            excludePath.add(PathSelectors.ant(path));
        }

        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo)
                .groupName(swaggerProperties.getGroup())
                .select()

.apis(RequestHandlerSelectors.basePackage(swaggerProperties.getBasePackage()))

.paths(Predicates.and(Predicates.not(Predicates.or(excludePath)),Predicates.or(basePat
h)))
                .build();
    }

    @Override
    public void setBeanFactory(BeanFactory beanFactory) throws BeansException {
        this.beanFactory = beanFactory;
```

```
    }
}
```

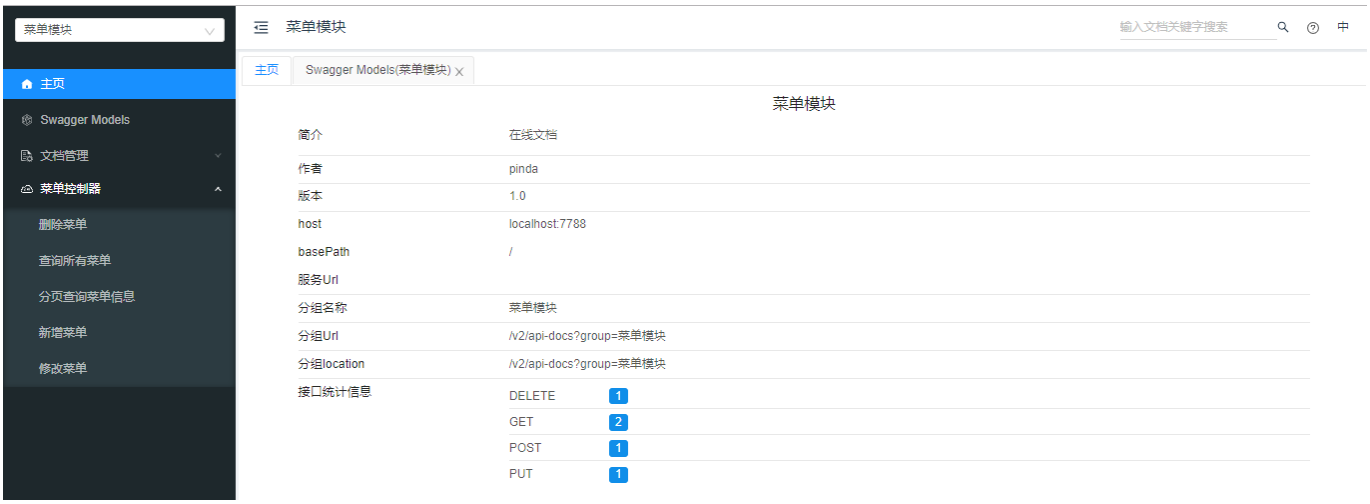
第七步：创建启动类SwaggerDemoApplication

```
package cn.itcast;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SwaggerDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(SwaggerDemoApplication.class, args);
    }
}
```

执行启动类main方法启动项目，访问地址：http://localhost:7788/doc.html



如果接口文档不分组，我们可以修改application.yml文件：

```yaml
server:
  port: 7788
pinda:
  swagger:
    enabled: true #是否启用swagger
    title: test模块
    base-package: cn.itcast.controller
```

再次访问地址：http://localhost:7788/doc.html

可以看到所有的接口在一个分组中。