



链滴

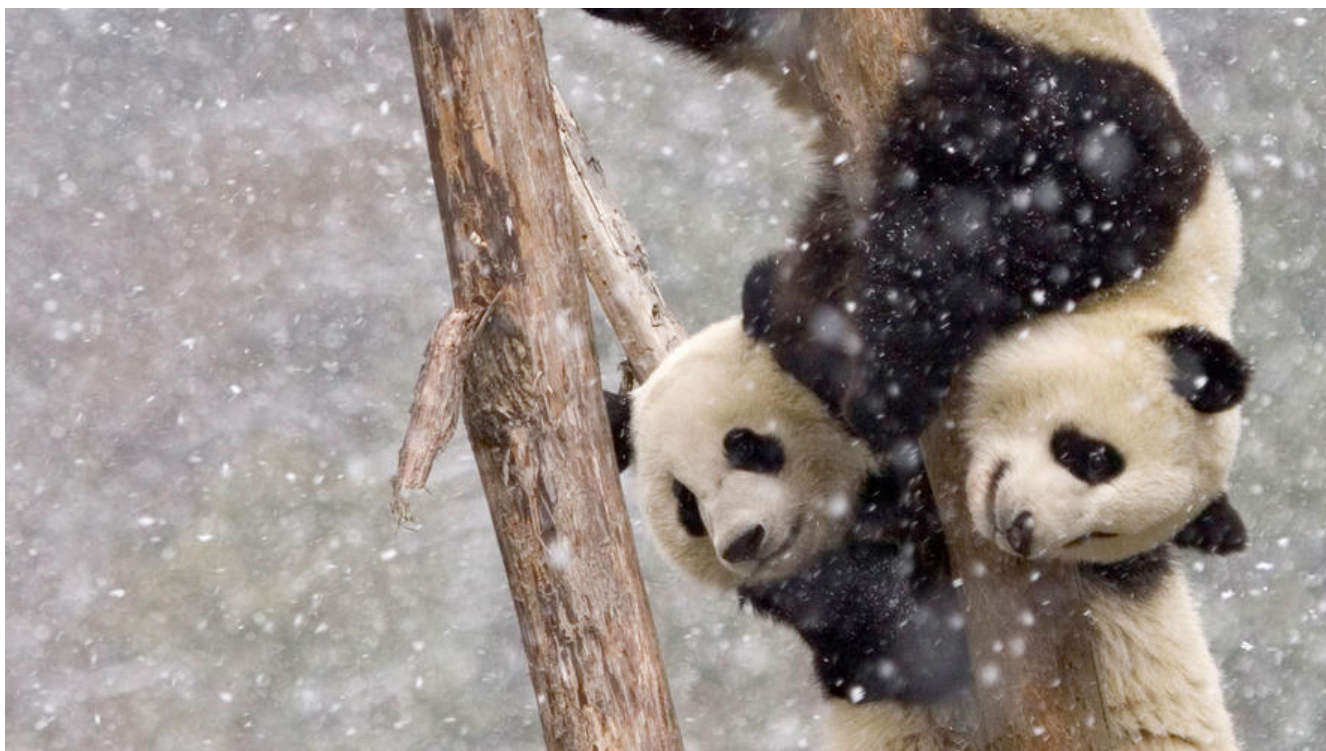
# spring boot 项目集成 knife4j 2.0.5 并实现 入参分组校验显示

作者: [kangaroo1122](#)

原文链接: <https://ld246.com/article/1632022105475>

来源网站: 链滴

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)



之前写过一篇：[前后端分离时如何优雅的编写API文档](#)

不过其中的部分配置还不够完善，本次对其进行一定的优化。

## 1 路径分组配置

项目中，有的路径需要登录，有的不需要登录，需要登录的接口还可能配置全局header，用于校验使用的token等

这里是使用路径进行是否需要登录的匹配，其中，路径包含/pub，则不需要登录，否则，需要登录，时扫描多个路径，路径之前用英文逗号(,) 隔开即可

具体实现如下：

```
@Autowired
private SwaggerProperties properties;

@Bean(value = "defaultApi")
public Docket defaultApi() {
    return new Docket(DocumentationType.SWAGGER_2)
        .useDefaultResponseMessages(false)
        .apiInfo(apiInfo())
        .groupName(properties.getGroupName() + "-待认证")
        .select()
        .apis(getPredicate())
        .paths(PathSelectors.regex("(?!pub).*"))
        .build()
        .securitySchemes(securitySchemes())
        .securityContexts(securityContexts());
}
```

```

@Bean(value = "pubApi")
public Docket pubApi() {
    return new Docket(DocumentationType.SWAGGER_2)
        .useDefaultResponseMessages(false)
        .apiInfo(apiInfo())
        .groupName(properties.getGroupName() + "-无认证")
        .select()
        .apis(getPredicate())
        .paths(PathSelectors.regex("^.*pub.*$"))
        .build();
}

//拼接多路径扫描以实现多包扫描支持
private Predicate getPredicate() {
    String[] split = properties.getBasePackage().split(",");
    Predicate predicate = null;
    for (String s : split) {
        Predicate secPredicate = RequestHandlerSelectors.basePackage(s);
        predicate = predicate == null ? secPredicate : Predicates.or(secPredicate, predicate);
    }
    return predicate;
}

private List<ApiKey> securitySchemes() {
    return new ArrayList(
        new ApiKey(properties.getHeader(), properties.getHeader(), "header"));
}

private List<SecurityContext> securityContexts() {
    return new ArrayList(
        SecurityContext.builder()
            .securityReferences(defaultAuth())
            .forPaths(PathSelectors.regex("(?!pub).*$"))
            .build()
    );
}

private List<SecurityReference> defaultAuth() {
    AuthorizationScope authorizationScope = new AuthorizationScope("global", "accessEverything");
    AuthorizationScope[] authorizationScopes = new AuthorizationScope[1];
    authorizationScopes[0] = authorizationScope;
    return new ArrayList(
        new SecurityReference(properties.getHeader(), authorizationScopes));
}

```

SwaggerProperties 配置文件只是把变化的配置放到了配置文件中配置

这样，访问<http://{IP}:{port}/doc.html> 地址，即可看到效果

## 2 访问控制

knife提供了简单的basic认证，其中

登录访问: SecurityBasicAuthFilter、生产环境屏蔽: ProductionSecurityFilter

由于项目自定义了部分配置, 为了统一, 自己实现注入时机

@Bean

```
@ConditionalOnMissingBean(SecurityBasicAuthFilter.class)
@ConditionalOnProperty(name = "swagger.certifiable", havingValue = "true")
public SecurityBasicAuthFilter securityBasicAuthFilter(SwaggerProperties properties) {
    return new SecurityBasicAuthFilter(properties.getCertifiable(), properties.getUsername(),
    properties.getPassword());
}
```

@Bean

```
@ConditionalOnMissingBean(ProductionSecurityFilter.class)
@ConditionalOnProperty(name = "swagger.prod", havingValue = "true")
public ProductionSecurityFilter productionSecurityFilter(SwaggerProperties properties) {
    return new ProductionSecurityFilter(properties.getProd());
}
```

这样, 在SwaggerProperties中配置相应的信息, 即可实现相关功能

## 3 入参分组

在开发的时候, 会遇到这样一个情况:

一个VO对象在新增的时候某些字段为必填, 在更新的时候又非必填。比如用户新增和编辑功能, 新时id不是必填, 编辑时是必须的, 其他的字段如用户名等都是同样新增、编辑都为必填。

而在swagger中, 不属于新增接口的id字段, 就不应该显示到文档上, 避免前端集成接口不知如何传的问题

以前是新增、编辑使用不同的实体, 或者编辑继承新增实体, 再加上id字段, 但是这样会造成项目中很多的实体, 类臃肿

基于此需求, 可以借助 javax.validation.groups 分组功能以及@Validated 来实现, 同时扩展swagger的功能, 此处参考: <https://blog.csdn.net/jianzhang11/article/details/119632467>

### 3.1 自定义分组

```
import javax.validation.groups.Default;

public interface ValidGroup extends Default {
    interface Create extends ValidGroup {
    }

    interface Update extends ValidGroup {
    }

    interface Query extends ValidGroup {
    }

    interface Delete extends ValidGroup {
    }
}
```

```
}
```

## 3.2 使用

controller层入参上添加分组信息，如：

新增：ValidGroup.Create.class

```
public RestResult<String> insertUser(@RequestBody @Validated(ValidGroup.Create.class) UserBO user) {  
    return userService.insertUser(user);  
}
```

编辑：ValidGroup.Update.class

```
public RestResult<String> updateUser(@RequestBody @Validated(ValidGroup.Update.class) UserBO user) {  
    return userService.updateUser(user);  
}
```

入参实体中：

```
public class UserBO implements Serializable {  
    private static final long serialVersionUID = 5699245096095831445L;  
  
    @ApiModelProperty(value = "ID")  
    @Null(groups = ValidGroup.Create.class)  
    @NotNull(groups = ValidGroup.Update.class, message = "ID不可为空")  
    private Long id;  
}
```

这样，再访问新增、编辑接口，即可看到效果。