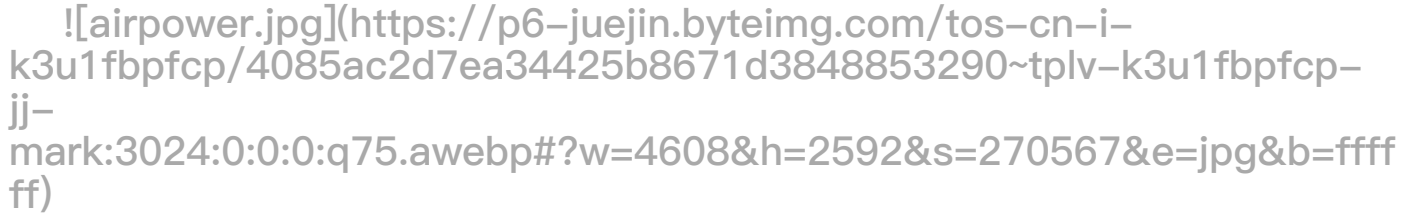
不使用Swagger生成文档，我们写了个“丝袜妹”

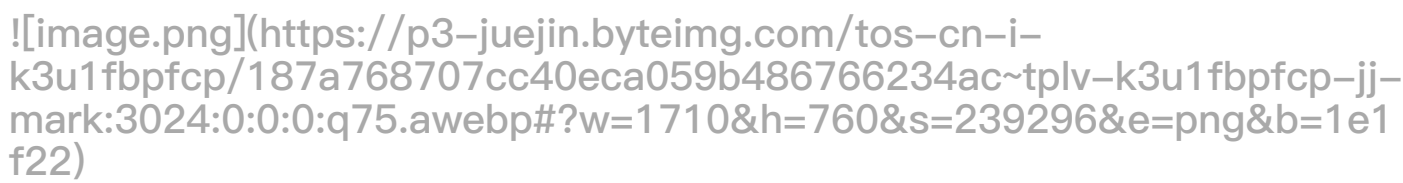


写在前面

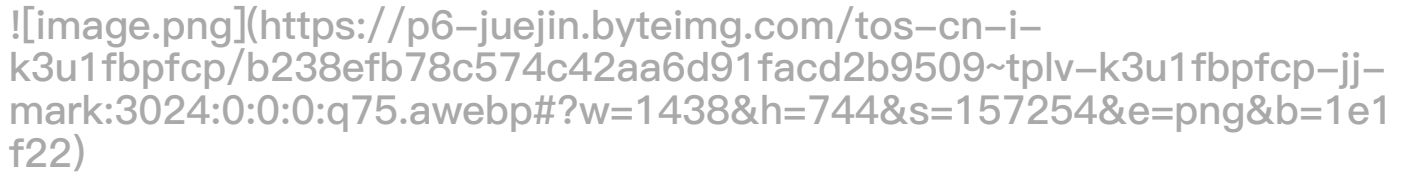
最开始我们是没集成 **Swagger** 的，然后也没有大量的文档的需求，于是一开始没太在意。

但基本在项目都快写完的情况下，我们发现利用现有的代码，巧妙的写一些解析规则，即可自动生成文档：

* 项目使用了 **JPA**，实体上标记了大量的解释注解；
* 使用了很多的内置和自定义的验证注解：



* 项目没有使用 **VO**/**EO**/**DTO**/**POJO** 等设计，全程是 **Entity** 一把梭；
* 使用了 **@Description** 作为系统的文案翻译工具，以提供在各种错误文案信息的提示



* 约束了前端统一使用 **POST** 进行数据交互（`window.open`类的下载请

求除外)
* 还有很多，想到再补。

开始设计
====

为了尽可能在不改动业务代码的情况下（如 **Swagger** 需要去对应的 **Controller** **VO** 上添加注解来声明文档），我们设计出如下的文档生成逻辑：

* **POST** 作为API提供服务，对应的 **GET** 请求作为文档输出：

> 如 `POST: /user/login` 作为登录接口，`GET: /user/login` 即为登录接口的文档输出。

* 直接读取 **Controller** 配置的注解、验证器，来生成文档所需要的各种信息：

> * @Description() 统一的文案：接口名称、属性名称等
> * @Validated() 验证器： 标记属性的必填、类型等
> * @Dictionary() 字典： 字典的可选值等
> * 还有很多，想到再补。

开始开发
====

如上设计的思路，我们直接在拦截器里对请求进行拦截，如果是 **GET** 请求，则开始反射读取访问的控制器名称和方法，得到接口文档的一些必要信息，如接口名称、备注、访问地址等等。

```
if (HttpMethod.GET.name().equalsIgnoreCase(request.getMethod()) &&
        globalConfig.isEnableDocument()) {
    // 如果是GET 方法，并且开启了文档
    GetMapping getMapping =
ReflectUtil.getAnnotation(GetMapping.class, method);
```

```
    if (Objects.isNull(getMapping)) {
        // 如果没有GetMapping注解，则直接返回文档
        ApiDocument.writeApiDocument(response, clazz, method);
        return false;
    }
}
```

开始生成接口的必要信息

```
public static void writeApiDocument(HttpServletResponse response,
Class<?> clazz, Method method) {
    ApiDocument apiDocument = new ApiDocument();
    String className = ReflectUtil.getDescription(clazz);
    String methodName = ReflectUtil.getDescription(method);
    apiDocument.setTitle(className + " " + methodName + " Api接口文档
");

    apiDocument.setDocument(ReflectUtil.getDocument(method));

    apiDocument.setRequestParamList(getRequestParamList(clazz,
method));
    .....
}
```

开始反射读取方法的参数和验证器，生成属性的列表：

```
private static List<ApiRequestParam> getRequestParamList(Class<?>
currentClass, Method method) {
    Parameter[] parameters = method.getParameters();
    if (parameters.length == 0) {
        return new ArrayList<>();
    }
    Parameter parameter = parameters[0];

    List<ApiRequestParam> params = new ArrayList<>();
    RequestBody requestBody =
parameter.getAnnotation(RequestBody.class);
    if (Objects.isNull(requestBody)) {
```

```
        return params;
    }

    Class<?> action = Void.class;
    Validated validated = parameter.getAnnotation(Validated.class);
    if (Objects.nonNull(validated)) {
        if (validated.value().length == 0) {
            return params;
        }
        action = validated.value()[0];
    }

    Class<?> paramClass = parameter.getType();
    if
(!parameter.getParameterizedType().getTypeName().contains(PACKAGE_
SPLIT)) {
        // 泛型
        paramClass = (Class<?>) ((((ParameterizedType)
currentClass.getGenericSuperclass()).getActualTypeArguments())[0]);     }


    List<Field> fields = ReflectUtil.getFieldList(paramClass);

    return getFieldList(fields, currentClass, action);
}
```

组装参数数组

```
private static List<ApiRequestParam> getFieldList(List<Field> fields,
Class<?> currentClass, Class<?> action) {
    List<ApiRequestParam> params = new ArrayList<>();
    for (Field field : fields) {
        ReadOnly readOnly = field.getAnnotation(ReadOnly.class);
        if (Objects.nonNull(readOnly)) {
            continue;
        }
        ApiRequestParam apiRequestParam = new ApiRequestParam();
        apiRequestParam.setName(field.getName());
        apiRequestParam.setDescription(ReflectUtil.getDescription(field));
        apiRequestParam.setDocument(ReflectUtil.getDocument(field));
        apiRequestParam.setType(field.getType().getSimpleName());
        if (ReflectUtil.isModel(field.getType())) {
            apiRequestParam.setLink(field.getType().getName());
```

```
        }

        // 获取字段的泛型类型
        if
(!field.getGenericType().getTypeName().contains(PACKAGE_SPLIT)) {
            Class<?> clazz = (Class<?>) (((ParameterizedType)
currentClass.getGenericSuperclass()).getActualTypeArguments())[0];
            apiRequestParam.setType(clazz.getSimpleName());
            if (ReflectUtil.isModel(clazz)) {
                apiRequestParam.setLink(clazz.getName());
            }
        }

        jakarta.validation.constraints.NotNull notNull =
field.getAnnotation(jakarta.validation.constraints.NotNull.class);
        NotBlank notBlank = field.getAnnotation(NotBlank.class);

        if (!action.equals(Void.class)) {
            if (Objects.nonNull(notBlank) &&
Arrays.stream(notBlank.groups()).toList().contains(action)) {
                apiRequestParam.setRequired(true);
            }
            if (Objects.nonNull(notNull) &&
Arrays.stream(notNull.groups()).toList().contains(action)) {
                apiRequestParam.setRequired(true);
            }
        }

        Dictionary dictionary = field.getAnnotation(Dictionary.class);
        if (Objects.nonNull(dictionary) &&
Arrays.stream(dictionary.groups()).toList().contains(action)) {
apiRequestParam.setDictionary(DictionaryUtil.getDictionaryList(dictionar
y.value()));
        }

        Phone phone = field.getAnnotation(Phone.class);
        if (Objects.nonNull(phone) &&
Arrays.stream(phone.groups()).toList().contains(action)) {
            if (phone.mobile() || phone.tel()) {
                apiRequestParam.setPhone(true);
            }
        }

        Email email = field.getAnnotation(Email.class);
        if (Objects.nonNull(email) &&
Arrays.stream(email.groups()).toList().contains(action)) {
            apiRequestParam.setEmail(true);
        }
```

```
        params.add(apiRequestParam);
    }
    return params;
}

```

最后的生成文档：

```
public static boolean writeEntityDocument(String packageName,
HttpServletResponse response) {
    System.out.println(packageName);
    try {
        Class<?> clazz = Class.forName(packageName);
        if (!ReflectUtil.isModel(clazz)) {
            return false;
        }
        List<ApiRequestParam> params =
getFieldList(ReflectUtil.getFieldList(clazz), clazz, Void.class);

        ApiDocument apiDocument = new ApiDocument();
        apiDocument.setTitle(ReflectUtil.getDescription(clazz) + " " +
clazz.getSimpleName());

        apiDocument.setDocument(ReflectUtil.getDocument(clazz));

        apiDocument.setRequestParamList(params);

        String html = """
                <!DOCTYPE html>
                    <html>
                        <head>
                            <title>AirPower4J 文档</title>
                            <style>
                            </style>
                        </head>
                        <body>
                            <div id="app" v-cloak>
                            </div>
                        </body>
                        <script src="//cdn.hamm.cn/js/vue-
2.6.10.min.js"></script>
                        <script
src="//cdn.hamm.cn/js/axios.min.js"></script>
                        <script src="//cdn.hamm.cn/js/element.js"></script>
```

```
                        <script src="//cdn.hamm.cn/js/vue-
clipboard.min.js"></script>
                        <script>
                        const json =
                        """
                + JSONUtil.toJsonStr(apiDocument) +
                """
                                </script>
                                <script>
                                new Vue({
                                    el: '#app',
                                    data() {
                                        return {
                                            url: window.location.pathname,
                                            api: json,
                                        }
                                    },
                                    created() {
                                        console.log(this.api)
                                    },
                                    updated() {},
                                    methods: {}
                                });
                                </script>

                            </html>
                        """;
        try {
            response.reset();
            response.setCharacterEncoding("UTF-8");
            response.getWriter().write(html);
            response.flushBuffer();
        } catch (IOException ignored) {

        }
        return true;
    } catch (ClassNotFoundException e) {
        return false;
    }
}
```

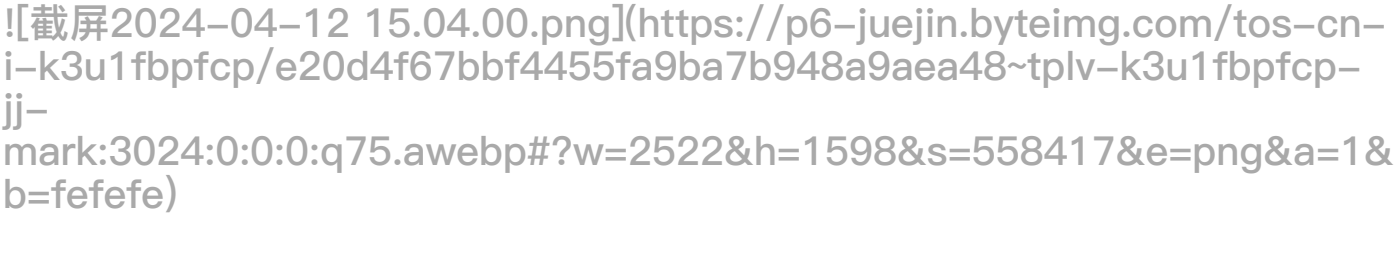
> 这里就直接输出一个 **HTML** 文件给前端即可显示文档

实现的效果

=====

通过 **POST** 请求这个URL，可以正常进行数据交互；

通过 **GET** 浏览器直接打开这个URL，则显示如下的文档：



后期计划
====

目前只实现了 **接口文档** **类的属性列表文档** 等，后续计划的项目：

* 直接测试发起请求
* 直接显示响应的结构体文档
* 支持在代码里通过 **@Document** 写 Markdown 说明。
* 优化文档的样式
* 还没想到，想到再说。

欢迎体验
====

可以查看我们的开源项目：

[github.com/HammCn/AirP...](http://cxyroad.com/”https://github.com/HammCn/AirPower4J”)

写完了

===

就酱，**丝袜妹** 就介绍到这，欢迎有兴趣的与我们一起交流。

原文链接: https://juejin.cn/post/7356757888087556096