链滴

自定义注解实现 API 版本管理 实现多版本共存

作者：zsr251
原文链接：https://ld246.com/article/1513526352709
来源网站：链滴

自定义注解实现API版本管理 实现多版本共存

spring boot实现接口多版本共存 灵活修改。GitHub地址：https://github.com/zsr251/SpringBootMybatis-ApiVersion

现状和问题

- 一般设计中无法针对 **单个接口**进行灵活的版本管理
- 接口更新时需要兼容之前的接口，导致接口的 **参数会越来越多**
- 接口更新后会有很多判断当前版本的 if else
- 接口多个版本不能并存

原理

- 自定义方法和参数注解
- 利用拦截器和反射，根据传入的版本参数调用 **Controller方法注解**中指定的类和对应的方法

解决的问题

- 可针对单个Controller方法进行版本控制，允许只升级某一个或几个接口，无需整体版本升级。加版本控制的Controller方法只需要在方法上加上注解即可
- 不更改原URL，兼容以前版本。使用该版本控制可以有效的兼容旧版本的APP，只需要修改服务端无需强制升级APP
- 接口版本完全不用 if else 判断
- 接口多个版本可并存，升级之前的代码完全保留
- 同一个接口每个版本之间参数可以完全不同，无需兼容之前其他版本的参数
- 入口和业务实现完全分离，灵活可配置，升级接口无需修改之前实现

存在的问题

- 依赖与spring mvc
- 没有对返回值进行json处理
- 返回页面的请求无法使用（ps：虽然返回页面的也不建议使用版本控制）
- 没有复用spring获取参数的方法，目前是自己的简单实现
- 目前不支持从body中获取参数
- 目前只支持基础路径参数，不支持路径中添加正则方式获取参数

实现

核心实现在自定义的两个注解和一个拦截器

类名	类型	主要作用

PathVariable	参数注解	spring自带注解，用于引用在URL中的参数
ApiVersion	方法注解	注解在要进行接口版本控制的Controller的对应方法上
ApiParam	参数注解	注解在正处理方法的参数中，**非PathVariable参数必须含有该注解**
DefaultValueEnum	默认参数值枚举	
ApiVersionException	异常	工解析中抛出的异常
ApiVersionInterceptor	接口拦截器	心类，进行所有处理操作

方法注解

```
/**
 * 需要拦截的API接口方法
 * Created by jasonzhu on 2016/11/28.
 */
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface ApiVersion {
    /**
     * 指定执行操作的类名
     */
    Class targetClass();

    /**
     * 指定执行操作的方法名前缀
     */
    String methodPreName() default "";

}
```

targetClass必须指定，指定真正进行业务处理的类，methodPreName如果不指定则为当前Controlle方法的方法名。例如：

```
/**
 * API版本管理测试
 */
@ApiVersion(targetClass = TestApiVersionDo.class)
@RequestMapping("/api/test")
public void test(){}
```

真正执行业务处理的类为：TestApiVersionDo，对应的版本方法是test + 版本号，默认是：test1 方法

参数注解

```
/**
 * 处理方法的参数注解
```

```
 * Created by jasonzhu on 2016/11/30.
 */
@Target({ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface ApiParam {
    /**
     * 参数名
     */
    String value();
    /**
     * 是否必须有值 默认必须有
     */
    boolean required() default true;
    /**
     * 值非必须时 如果未传值 默认值
     */
    DefaultValueEnum defaultValue() default DefaultValueEnum.NULL;
}
```

注解在真正处理方法的参数中，**非PathVariable参数必须含有该注解**,如果没有指定则会报错。

```
/**
 * 版本拦截器之后真正执行的方法
 * Created by jasonzhu on 2016/12/1.
 */
@Controller
public class TestApiVersionDo {
    public String test1(){
        return "调用成功 没有参数";
    }
    public String test2(@ApiParam("av") Integer av,@ApiParam("a") String app,@ApiParam(val
e = "b",required = false) String b){
        return "调用成功 三个参数 app:"+app+" av:"+av+" b:"+b;
    }
}
```

拦截器

代码有点长，摘录一部分，建议直接Github下载源码查看，文章中可直接跳过。

```
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
handler) throws Exception {
        if (!(handler instanceof HandlerMethod)) {
            return true;
        }
        HandlerMethod method = (HandlerMethod) handler;
        ApiVersion apiVersion = method.getMethodAnnotation(ApiVersion.class);
        //判断是否纳入接口版本控制
        if (apiVersion == null) {
            return true;
        }
        //TODO 判断如果是返回页面的方法 则不纳入版本控制
```

```
        //controller中调用的方法
        RequestMapping requestMapping =  method.getMethodAnnotation(RequestMapping.cl
ss);
        String [] mappingPaths = requestMapping.value()[0].split("/");
        String [] requestPaths = request.getRequestURI().split("\\.")[0].split("/");

        Class cls = apiVersion.targetClass();
        Object o;
        try {
            o = context.getBean(cls);
        } catch (Exception e) {
            throw new ApiVersionException("指定的处理类必须纳入spring的bean管理", e);
        }
        String preName = apiVersion.methodPreName();
        if (preName == null || preName.trim().isEmpty()) {
            preName = method.getMethod().getName();
        }
        //默认接口版本号
        String av = "1";
        //参数列表
        Map<String, String[]> requestParam = request.getParameterMap();
        if (requestParam.get(API_VERSION) != null) {
            av = requestParam.get(API_VERSION)[0];
        }
        Method[] methods = cls.getMethods();
        if (methods == null) {
            writeMsg(response, "未找到响应方法");
            return false;
        }
        Method targetMethod = null;
        //找到指定的处理方法
        for (Method me : methods) {
            if (me.getName().equals(preName + av)) {
                targetMethod = me;
                break;
            }
        }
        if (targetMethod == null) {
            writeMsg(response, "非法请求");
            return false;
        }
        if (!targetMethod.getReturnType().equals(String.class)) {
            throw new ApiVersionException("响应方法返回类型必须为String :" + targetMethod.get
ame());
        }
        //获得方法的参数
        Class<?>[] paramTypes = targetMethod.getParameterTypes();
        Integer paramLength = paramTypes.length;

        //调动方法的参数
        Object[] paramList = new Object[paramLength];
        Annotation[][] annotationss = targetMethod.getParameterAnnotations();
        //总注解参数个数
        for (int i = 0; i < annotationss.length; i++) {
```

```
            Annotation[] annotations = annotationss[i];
            if (annotations.length < 1)
                throw new ApiVersionException("存在未添加@ApiParam注解参数的方法 :" + targetM
thod.getName());
            //是否存在ApiParam或PathVariable注解
            boolean hasAnn = false;
            for (int j = 0; j < annotations.length; j++) {
                Annotation annotation = annotations[j];
                if (annotation instanceof ApiParam) {
                    if (paramTypes[i].equals(HttpServletRequest.class)){
                        paramList[i] = request;
                    }else if (paramTypes[i].equals(HttpServletResponse.class)) {
                        paramList[i] = response;
                    }else{
                        //为参数赋值
                        paramList[i] = getParam(requestParam, (ApiParam) annotation, paramTypes[i]);
                    }
                    hasAnn = true;
                    break;
                }
                if (annotation instanceof PathVariable){
                    //为参数赋值
                    paramList[i] = getPathVariable(requestPaths,mappingPaths,(PathVariable) annota
ion,paramTypes[i]);
                    hasAnn = true;
                    break;
                }

            }
            if (!hasAnn)
                throw new ApiVersionException("存在未添加@ApiParam或@PathVariable注解参数
方法 :" + targetMethod.getName());
        }


        //反射方法调用
        String result = (String) targetMethod.invoke(o, paramList);
        writeMsg(response, result);
        return false;
    }
```

针对存在问题的解决方案

干!

其他

没有特殊需求的话，这个版本已经可以使用了。在2017年的最后一个月还是要维护一个版本，毕竟将一年没有更新了 ~

原文链接：自定义注解实现 API 版本管理 实现多版本共存