# 公共依赖模块common的实现

# common 模块的简介

Commom模块旨在多个模块之间共享相同的工具类、配置类、异常类、全局异常处理、全局返回值对象，公共依赖等等。使用Common模块的可以尽可能的将代码的复用性提升，但是，若复用类的提取不符合共用的原则，则可能导致高耦合的情况。

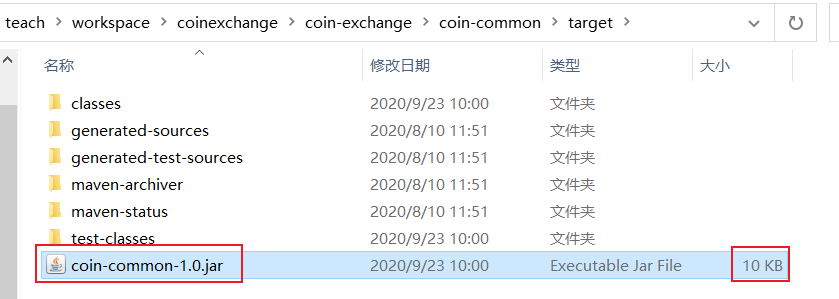
# common的工作原理

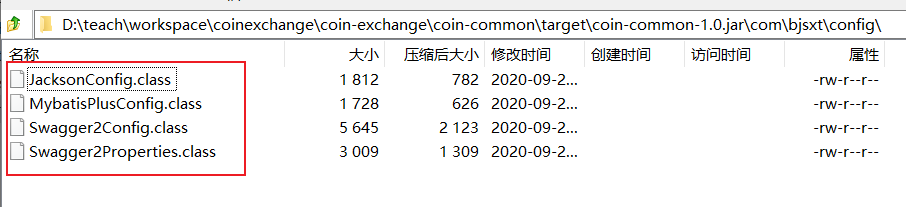
## 2.1 common的打包方式

Common **并不会**使用spring-boot-maven-plugin打包插件来打包自己，而是使用Maven自带的打包方式，也就是：

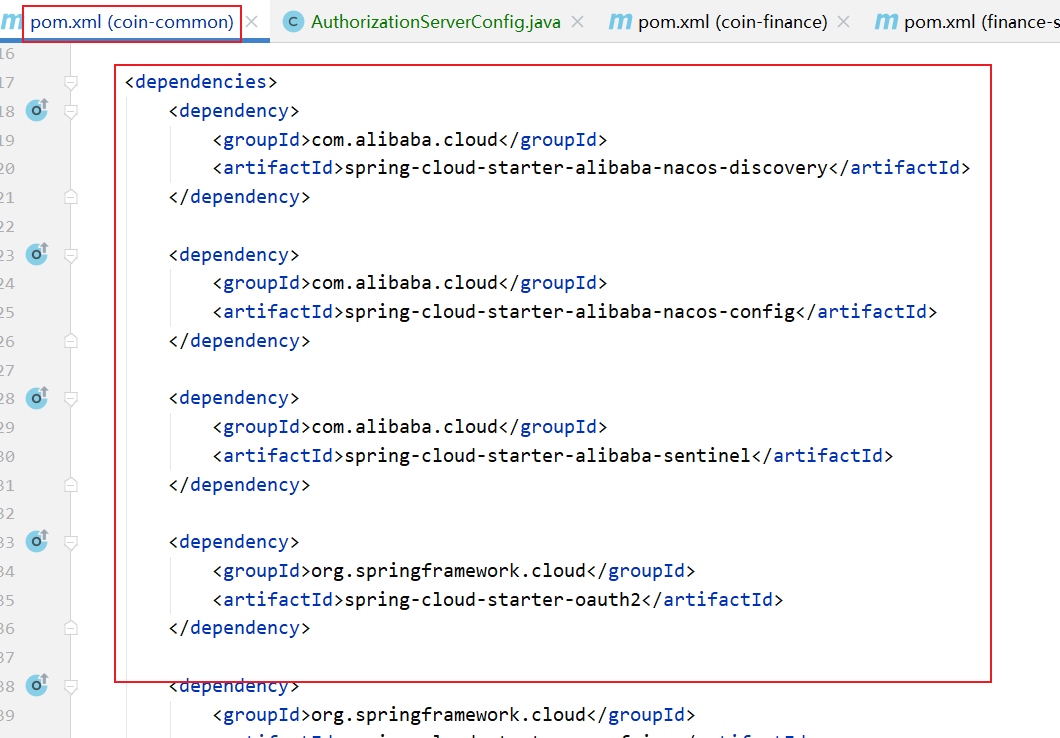
|  |
| --- |
| <packaging>jar</packaging> |

这种方式来打包的，这种方式的打包只会把类编译后生成的class文件打包在一个jar 里面，不会打包类包含的第三方依赖，这就保证了common的轻量级。





并且，我们在该项目的pom.xml文件里面声明的依赖也不会被打包进来。这就保证了绝对的轻量级。



但是需要注意：虽然common 里面不打包第三方的依赖，但是common 模块里面类实际被使用时，该类需要的第三方依赖必须存在，否则会导致ClassNotFound这样的异常。

## 2.2对于配置类

在配置类上，我们都会添加spring 提供的@Configuration这样的注解，只要我们类所在的包能被Spring扫描到，则我们的配置对象均会被Spring创建出来并放在IOC容器里面。Spring 默认的包扫描会扫描jar 里面的配置类。

## 2.3 对应全局异常处理

全局异常处理我们使用了@RestControllerAdvice 进行标记，它的工作方式也是将当前的类创建出来对象放在IOC里面，和@Configuration区别不大。

## 2.4 全局返回值、工具类、公共类

这些类只有在使用时，该类的class文件才会被加载，因此我们的公共类只要能被类加载器加载到就可以了。Maven 会自动的把我们依赖的jar 放在classpath 里面，使用者只有想使用，就会能加载到。

## 2.5 使用Maven的pom.xml 文件抽取公共模块里面需要重复添加的依赖

在pom.xml文件里面只有添加了第三方的依赖，我们的依赖它的所有模块都会默认的依赖它，这就是Maven的继承好处，因此，我们可以将子模块里面公用的依赖抽取出来，放在pom.xml 文件里面。

# pom.xml处理

|  |
| --- |
| <dependencies>  *<!-- 服务的注册中心-->* <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>  </dependency>   *<!-- 服务的配置中心-->* <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>  </dependency>  *<!-- 服务的熔断和降级限流-->* <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>  </dependency>  *<!-- 资源服务器-->* <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-oauth2</artifactId>  </dependency>  *<!--远程调用-->* <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-openfeign</artifactId>  </dependency>  *<!-- mybatis-plus-->* <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  </dependency>  *<!-- mysql 驱动-->* <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>   *<!-- redis 操作-->* <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency>  *<!-- jetcache 缓存-->* <dependency>  <groupId>com.alicp.jetcache</groupId>  <artifactId>jetcache-starter-redis</artifactId>  </dependency>  *<!-- spring 里面添加的一些工具类-->* <dependency>  <groupId>io.springside</groupId>  <artifactId>springside-utils</artifactId>  </dependency>  *<!-- swagger2-->* <dependency>  <groupId>io.springfox</groupId>  <artifactId>springfox-swagger2</artifactId>  </dependency>  *<!-- swagger2的web界面-->* <dependency>  <groupId>io.springfox</groupId>  <artifactId>springfox-swagger-ui</artifactId>  </dependency>  *<!-- web-->* <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>   *<!-- hutool工具类-->* <dependency>  <groupId>cn.hutool</groupId>  <artifactId>hutool-all</artifactId>  </dependency>    *<!--mapStruct依赖 高性能对象映射-->* <dependency>  <groupId>org.mapstruct</groupId>  <artifactId>mapstruct-jdk8</artifactId>  </dependency>  <dependency>  <groupId>org.mapstruct</groupId>  <artifactId>mapstruct-processor</artifactId>  <scope>provided</scope>  </dependency>  *<!-- 配置文件处理-->* <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-configuration-processor</artifactId>  </dependency>  </dependencies> |

Ps: 如果以后我们添加依赖，一定要先去coin-exchange 里面做依赖管理！！

例子: 添加hutool的依赖

* 1、先coin-exchange 里面做依赖的管理：

添加版本号

|  |
| --- |
| <properties>  <hutool.version>4.1.2</hutool.version>  </properties> |

依赖管理：

|  |
| --- |
| <dependencyManagement>  <dependency>  <groupId>cn.hutool</groupId>  <artifactId>hutool-all</artifactId>  <version>${hutool.version}</version> </dependency>  </dependencyManagement> |

* 2、在子项目里面依赖

|  |
| --- |
| *<!-- hutool工具类-->* <dependency>  <groupId>cn.hutool</groupId>  <artifactId>hutool-all</artifactId>  </dependency> |

# 常用的配置类

放在com.bjsxt.config项目

## 4.1 Swagger相关

Swagger2用做做API文档的产出工具

### 4.1.1 SwaggerProperties

|  |
| --- |
| @Data @ConfigurationProperties(prefix = "swagger2") public class SwaggerProperties {   */\*\*  \* 包扫描的路径  \*/* private String basePackage ;   */\*\*  \* 联系人的名称  \*/* private String name ;   */\*\*  \* 联系人的主页  \*/* private String url ;   */\*\*  \* 联系人的邮箱  \*/* private String email ;   */\*\*  \* API的标题  \*/* private String title ;   */\*\*  \* API的描述  \*/* private String description ;   */\*\*  \* API的版本号  \*/* private String version ;   */\*\*  \* API的服务团队  \*/* private String termsOfServiceUrl ;   } |

### 4.1.2 SwaggerAutoConfiguration

|  |
| --- |
| @Configuration @EnableSwagger2 @EnableConfigurationProperties(SwaggerProperties.class) public class SwaggerAutoConfiguration {   private SwaggerProperties swaggerProperties;   public SwaggerAutoConfiguration(SwaggerProperties swaggerProperties) {  this.swaggerProperties = swaggerProperties;  }   @Bean  public Docket docket() {  Docket docket = new Docket(DocumentationType.*SWAGGER\_2*)  .apiInfo(apiInfo())  .select()  .apis(RequestHandlerSelectors.*basePackage*(swaggerProperties.getBasePackage()))  .paths(PathSelectors.*any*())  .build();  *// 安全的配置* docket.securitySchemes(securitySchemes()) *// 安全规则* .securityContexts(securityContexts()); *// 安全配置的上下问* return docket;  }   */\*\*  \* api 信息的简介  \*  \* @return  \*/* private ApiInfo apiInfo() {  return new ApiInfoBuilder().contact(  new Contact(swaggerProperties.getName(), swaggerProperties.getUrl(), swaggerProperties.getEmail())  )  .title(swaggerProperties.getTitle())  .description(swaggerProperties.getDescription())  .version(swaggerProperties.getVersion())  .termsOfServiceUrl(swaggerProperties.getTermsOfServiceUrl())  .build();   }   */\*\*  \* 安全的规则配置  \*  \* @return  \*/* private List<SecurityScheme> securitySchemes() {  return Arrays.*asList*(new ApiKey("Authorization", "Authorization", "Authorization"));  }   */\*\*  \* 安全的上下问  \*  \* @return  \*/* private List<SecurityContext> securityContexts() {  return Arrays.*asList*(new SecurityContext(  Arrays.*asList*(new SecurityReference("Authorization", new AuthorizationScope[]{new AuthorizationScope("global", "accessResource")})),  PathSelectors.*any*()  ));  }   } |

## 4.2 Jackson相关

|  |
| --- |
| @Configuration public class JacksonConfig {  */\*\*  \* 配置ObjectMapper  \* @return  \*/* @Bean  public ObjectMapper objectMapper() {  ObjectMapper objectMapper = new ObjectMapper();  objectMapper.registerModule(new JavaTimeModule());  SimpleModule simpleModule = new SimpleModule();  simpleModule.addSerializer(Long.class, ToStringSerializer.*instance*);  objectMapper.registerModule(simpleModule);  objectMapper.setTimeZone(TimeZone.*getTimeZone*("Asia/shanghai"));  objectMapper.setDateFormat(new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"));  objectMapper.setVisibility(PropertyAccessor.*ALL*, JsonAutoDetect.Visibility.*ANY*);   return objectMapper;  } } |

## 4.3 Redis 相关

我们并不会使用Redis做缓存，但是手动操作redis时，RedisTemplate需要配置序列化才好用。

|  |
| --- |
| @Configuration public class RedisConfig {   */\*\*  \* 使用RedisTemplate<String, Object> 是需要注意其序列化的方式  \* @param redisConnectionFactory  \* @param redisSerializer  \* @return  \*/* @Bean  public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory, RedisSerializer<Object> redisSerializer) {  RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();  redisTemplate.setConnectionFactory(redisConnectionFactory);  redisTemplate.setKeySerializer(new StringRedisSerializer());  redisTemplate.setValueSerializer(redisSerializer);  redisTemplate.setHashKeySerializer(new StringRedisSerializer());  redisTemplate.setHashValueSerializer(redisSerializer);  redisTemplate.afterPropertiesSet();  return redisTemplate;  }   */\*\*  \* 更换redis的序列化形式为Jackson  \* @param objectMapper  \* @return  \*/* @Bean  public RedisSerializer<Object> redisSerializer(ObjectMapper objectMapper) {  *//创建JSON序列化器* Jackson2JsonRedisSerializer<Object> serializer = new Jackson2JsonRedisSerializer<>(Object.class);  serializer.setObjectMapper(objectMapper);  return serializer;  }  } |

## 4.4 ResourceServer相关

我们的每个应用都是一个资源服务器，外部访问都需要token才能进行

ResourceServerConfig

|  |
| --- |
| @Slf4j @EnableGlobalMethodSecurity(prePostEnabled = true) @EnableResourceServer public class ResourceServerConfig extends ResourceServerConfigurerAdapter {   @Override  public void configure(HttpSecurity http) throws Exception {  http  *// 由于使用的是JWT，我们这里不需要csrf* .csrf().disable()  *// 基于token，所以不需要session* .sessionManagement().disable()  .authorizeRequests().antMatchers(  "/v2/api-docs",  "/swagger-resources/configuration/ui",*//用来获取支持的动作* "/swagger-resources",*//用来获取api-docs的URI* "/swagger-resources/configuration/security",*//安全选项* "/webjars/\*\*",  "/swagger-ui.html"  ).permitAll()  .antMatchers("/\*\*").authenticated()  .and().headers().cacheControl();  }   @Override  public void configure(ResourceServerSecurityConfigurer resources) throws Exception {  resources.tokenStore(tokenStore());  }   private TokenStore tokenStore() {  JwtTokenStore jwtTokenStore = new JwtTokenStore(jwtAccessTokenConverter());  return jwtTokenStore;  }   @Bean  public JwtAccessTokenConverter jwtAccessTokenConverter() {  JwtAccessTokenConverter tokenConverter = new JwtAccessTokenConverter();  ClassPathResource classPathResource = new ClassPathResource("coinexchange.pub");  String publicKey = null;  try {  byte[] bytes = FileCopyUtils.*copyToByteArray*(classPathResource.getInputStream());  publicKey = new String(bytes, "utf-8");  } catch (IOException e) {  *log*.info("读取公钥失败");  }  tokenConverter.setVerifierKey(publicKey);  return tokenConverter;  }  } |

## 4.5 MybatisPlus相关

|  |
| --- |
| @Configuration public class MybatisPlusConfig {    */\*\*  \* 分页插件  \*/* @Bean  public PaginationInterceptor paginationInterceptor(){  PaginationInterceptor paginationInterceptor = new PaginationInterceptor();  paginationInterceptor.setDbType(DbType.*MYSQL*);  return paginationInterceptor ;  }   */\*\*  \* 乐观锁插件  \* @return  \*/* @Bean  public OptimisticLockerInterceptor optimisticLockerInterceptor(){  OptimisticLockerInterceptor optimisticLockerInterceptor = new OptimisticLockerInterceptor();  return optimisticLockerInterceptor ;  }   */\*\*  \* Id 生成器-->*  *\* 特殊的一些类使用*  *\* 默认使用  \* @return  \*/* @Bean  public IKeyGenerator iKeyGenerator(){  H2KeyGenerator h2KeyGenerator = new H2KeyGenerator();  return h2KeyGenerator ;  } |

## 4.6 JectCacheConfig

|  |
| --- |
| @Configuration @EnableCreateCacheAnnotation @EnableMethodCache(basePackages = "com.bjsxt.service.impl") public class JetCacheConfig {   } |

# 常用的常量

|  |
| --- |
| public class Constants {  */\*\*  \* UTF-8 字符集  \*/* public static final String *UTF8* = "UTF-8";   */\*\*  \* GBK 字符集  \*/* public static final String *GBK* = "GBK";   */\*\*  \* http请求  \*/* public static final String *HTTP* = "http://";   */\*\*  \* https请求  \*/* public static final String *HTTPS* = "https://";   */\*\*  \* 成功标记  \*/* public static final Integer *SUCCESS* = 200;   */\*\*  \* 失败标记  \*/* public static final Integer *FAIL* = 500;    */\*\*  \* 验证码 redis key  \*/* public static final String *CAPTCHA\_CODE\_KEY* = "captcha\_codes:";   */\*\*  \* 验证码有效期（分钟）  \*/* public static final long *CAPTCHA\_EXPIRATION* = 2;  } |

# 统一的返回值对象

|  |
| --- |
| */\*\*  \* 公共的返回值对象  \*  \* @param <T>  \*/* public class R<T> implements Serializable {   private static final long *serialVersionUID* = 1L;   */\*\*  \* 成功  \*/* public static final int *SUCCESS* = Constants.*SUCCESS*;   */\*\*  \* 失败  \*/* public static final int *FAIL* = Constants.*FAIL*;   private int code;   private String msg;   private T data;   public static <T> R<T> ok() {  return *restResult*(null, *SUCCESS*, null);  }   public static <T> R<T> ok(T data) {  return *restResult*(data, *SUCCESS*, null);  }   public static <T> R<T> ok(T data, String msg) {  return *restResult*(data, *SUCCESS*, msg);  }   public static <T> R<T> fail() {  return *restResult*(null, *FAIL*, null);  }   public static <T> R<T> fail(String msg) {  return *restResult*(null, *FAIL*, msg);  }   public static <T> R<T> fail(T data) {  return *restResult*(data, *FAIL*, null);  }   public static <T> R<T> fail(T data, String msg) {  return *restResult*(data, *FAIL*, msg);  }   public static <T> R<T> fail(int code, String msg) {  return *restResult*(null, code, msg);  }   private static <T> R<T> restResult(T data, int code, String msg) {  R<T> apiResult = new R<>();  apiResult.setCode(code);  apiResult.setData(data);  apiResult.setMsg(msg);  return apiResult;  }   public int getCode() {  return code;  }   public void setCode(int code) {  this.code = code;  }   public String getMsg() {  return msg;  }   public void setMsg(String msg) {  this.msg = msg;  }   public T getData() {  return data;  }   public void setData(T data) {  this.data = data;  } } |

# 公共的模型对象

|  |
| --- |
| */\*\*  \* web 操作日志记录  \*/* @Data @EqualsAndHashCode(callSuper = false) public class WebLog {  */\*\*  \* 操作描述  \*/* private String description;   */\*\*  \* 操作用户  \*/* private String username;   */\*\*  \* 消耗时间  \*/* private Integer spendTime;   */\*\*  \* 根路径  \*/* private String basePath;   */\*\*  \* URI  \*/* private String uri;   */\*\*  \* URL  \*/* private String url;   */\*\*  \* 请求类型  \*/* private String method;   */\*\*  \* IP地址  \*/* private String ip;   */\*\*  \* 请求参数  \*/* private Object parameter;   */\*\*  \* 返回结果  \*/* private Object result;  } |

# 公用的切面

## 8.1 web日志记录（用与logstash）

|  |
| --- |
| @Aspect @Component @Order(1) @Slf4j public class WebLogAspect {   @Pointcut("execution( \* com.bjsxt.controller.\*.\*(..))")  public void webLog() {  }   @Around(value = "webLog()")  public Object recordWebLog(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {  Object result = null;  StopWatch stopWatch = new StopWatch(); *// 创建计时器* stopWatch.start(); *// 开始计时器* result = proceedingJoinPoint.proceed(proceedingJoinPoint.getArgs()); *// 不需要我们自己处理这个异常* stopWatch.stop(); *// 记时结束   // 获取请求的上下文* ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  HttpServletRequest request = requestAttributes.getRequest();  *// 获取登录的用户* Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  *// 获取方法* MethodSignature methodSignature = (MethodSignature) proceedingJoinPoint.getSignature();  Method method = methodSignature.getMethod();  *// 获取方法上的ApiOperation注解* ApiOperation annotation = method.getAnnotation(ApiOperation.class);  *// 获取目标对象的类型名称* String className = proceedingJoinPoint.getTarget().getClass().getName();  *// 获取请求的url 地址* String requestUrl = request.getRequestURL().toString();  WebLog webLog = WebLog.*builder*()  .basePath(StrUtil.*removeSuffix*(requestUrl, URLUtil.*url*(requestUrl).getPath()))  .description(annotation == null ? "no desc" : annotation.value())  .ip(request.getRemoteAddr())  .parameter(getMethodParameter(method, proceedingJoinPoint.getArgs()))  .method(className + "." + method.getName())  .result(request == null ? "" : JSON.*toJSONString*(request))  .recodeTime(System.*currentTimeMillis*())  .spendTime(stopWatch.getTotalTimeMillis())  .uri(request.getRequestURI())  .url(request.getRequestURL().toString())  .username(authentication == null ? "anonymous" : authentication.getPrincipal().toString())  .build();  *log*.info(JSON.*toJSONString*(webLog, true));  return result;  }   */\*\*  \* {  \* "":value,  \* "":"value"  \* }  \*  \* @param method  \* @param args  \* @return  \*/* private Object getMethodParameter(Method method, Object[] args) {  LocalVariableTableParameterNameDiscoverer localVariableTableParameterNameDiscoverer = new LocalVariableTableParameterNameDiscoverer();  String[] parameterNames = localVariableTableParameterNameDiscoverer.getParameterNames(method);  HashMap<String, Object> methodParameters = new HashMap<>();  Parameter[] parameters = method.getParameters();  if (args != null) {  for (int i = 0; i < parameterNames.length; i++) {  methodParameters.put(parameterNames[i], args[i] == null ? "" : JSON.*toJSONString*(args[i]));  }  }  return methodParameters;  } } |

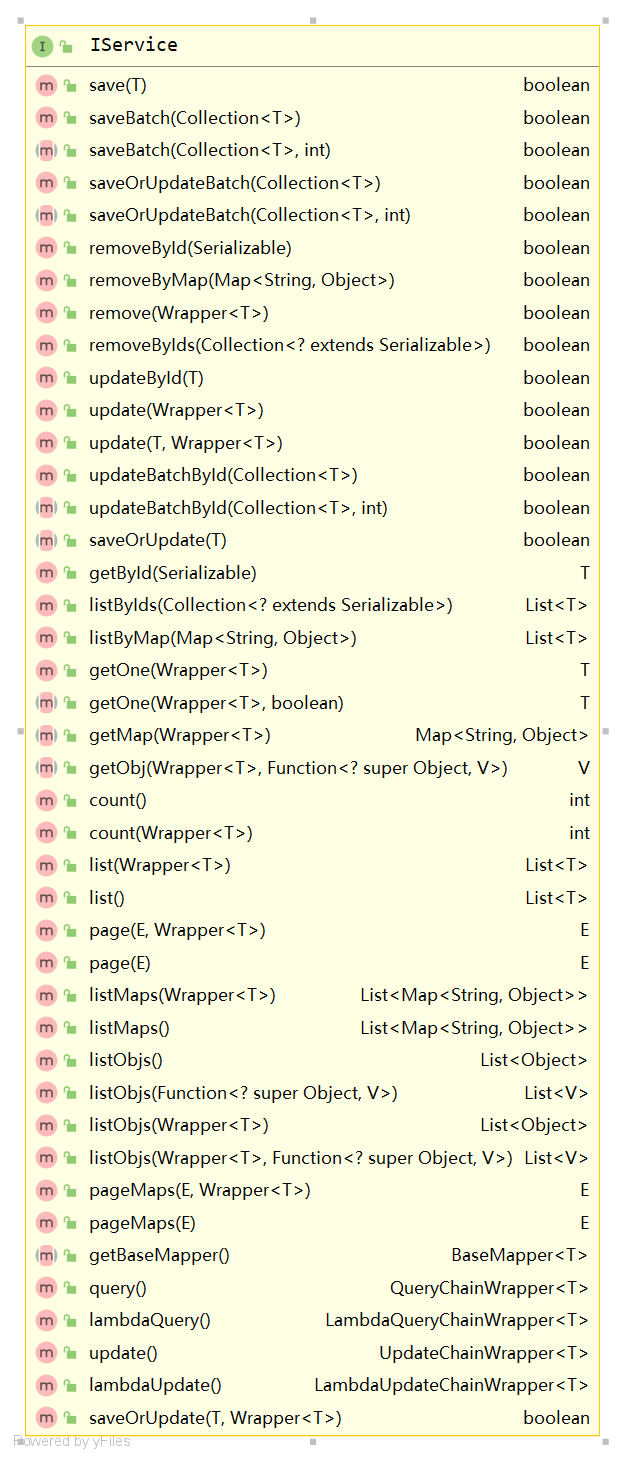
## 8.2 全局Web的异常处理

|  |
| --- |
| @RestControllerAdvice public class GlobalExceptionHandler {   @ExceptionHandler(value = ApiException.class)  public R handle(ApiException e) {  if (e.getErrorCode() != null) {  return R.*fail*(e.getErrorCode());  }  return R.*fail*(e.getMessage());  }    @ExceptionHandler(value = MethodArgumentNotValidException.class)  public R handleValidException(MethodArgumentNotValidException e) {  BindingResult bindingResult = e.getBindingResult();  String message = null;  if (bindingResult.hasErrors()) {  FieldError fieldError = bindingResult.getFieldError();  if (fieldError != null) {  message = fieldError.getField() + fieldError.getDefaultMessage();  }  }  return R.*fail*(message);  }   @ExceptionHandler(value = BindException.class)  public R handleValidException(BindException e) {  BindingResult bindingResult = e.getBindingResult();  String message = null;  if (bindingResult.hasErrors()) {  FieldError fieldError = bindingResult.getFieldError();  if (fieldError != null) {  message = fieldError.getField() + fieldError.getDefaultMessage();  }  }  return R.*fail*(message);  } } |

# MybatisPlus常用的对象

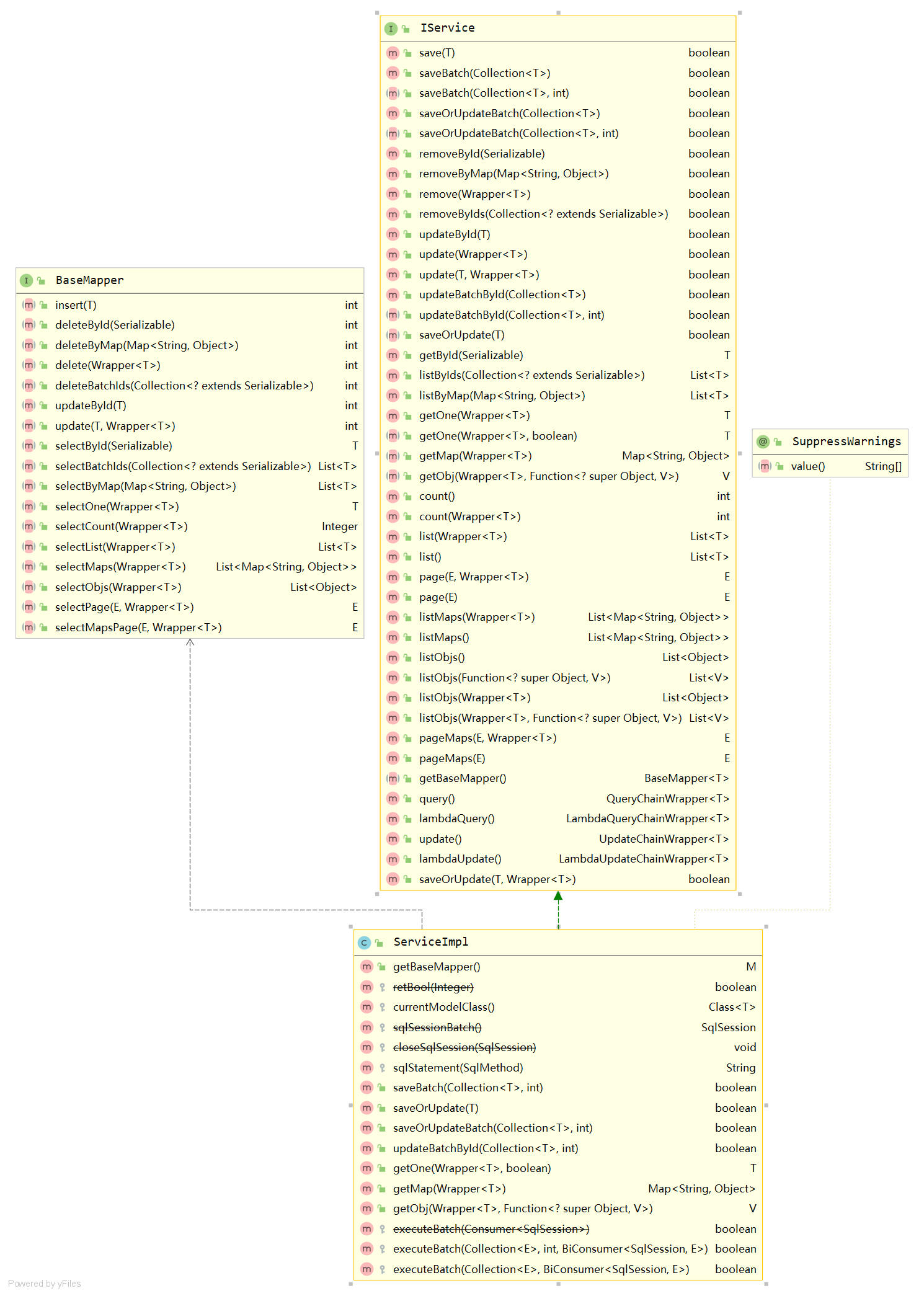
## 9.1 IService<T>

Service接口的抽象，以后我们的所有Service接口都将extends 这个接口



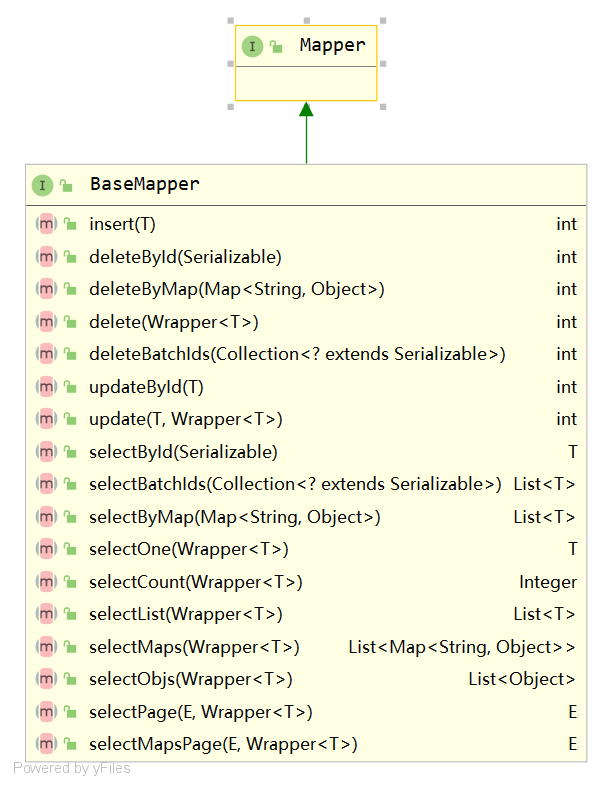
## 9.2 ServiceImpl

借助BaseMapper，实现IService<T> 抽象的方法，以后我们所有的ServiceImpl 都将extends它。



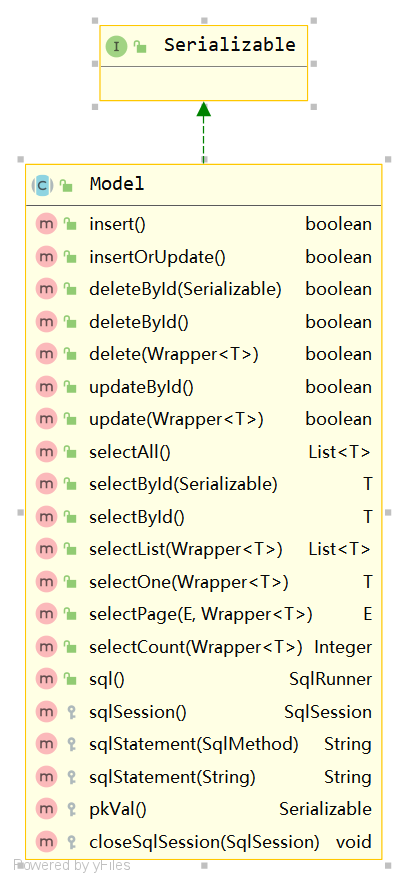
## 9.3 BaseMapper

通用的Mapper 数据库操作对象，我们所有的Mapper接口都将extends它



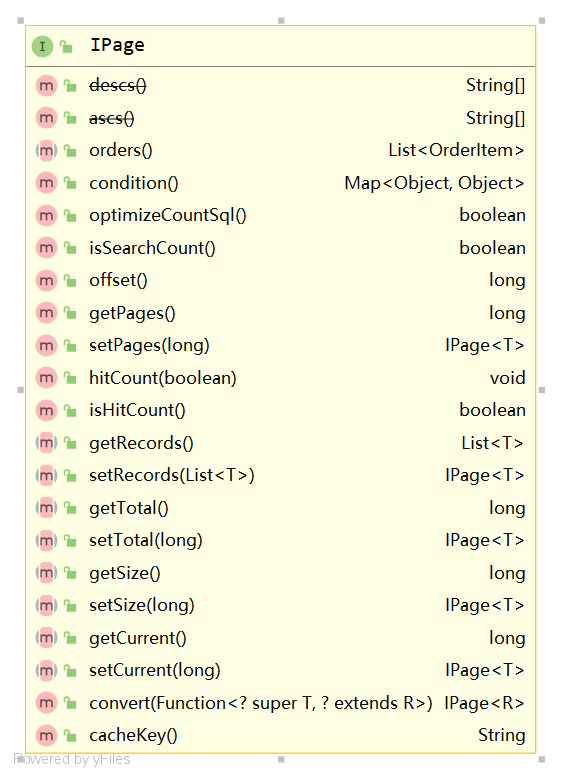
## 9.3 Model(AR模式)

通用的模型对象，该对象在AR模式里面非常的好用，但是我们若用在远程调用传值，是非常不可取的，因为：1、它很重，2、json无法序列化。



## 9.4 分页对象

我们使用它来传递分页的参数，以及返回一个分页的查询对象。





# 测试使用

## 10.1 新建启动类

|  |
| --- |
| @SpringBootApplication @EnableDiscoveryClient public class TestCommonApplication {   public static void main(String[] args) {  SpringApplication.*run*(TestCommonApplication.class ,args) ;  } } |

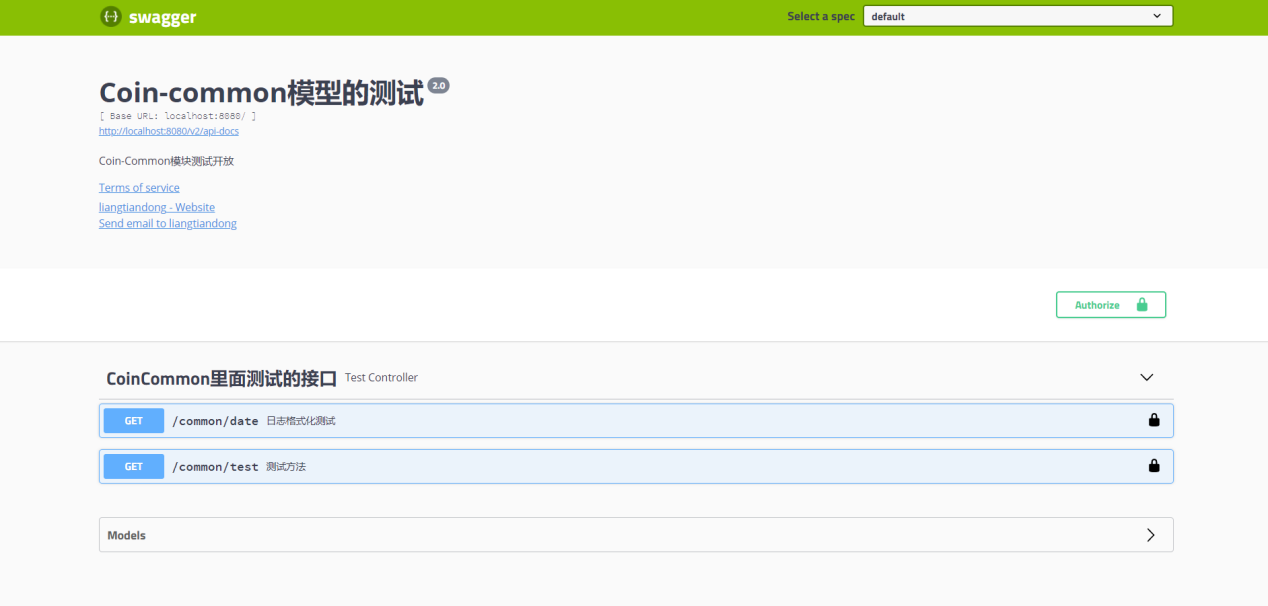
## 10.2 配置文件

|  |
| --- |
| server:  port: 8080 spring:  application:  name: coin-common  profiles:  active: dev  datasource:  driver-class-name: com.mysql.cj.jdbc.Driver  url: jdbc:mysql://mysql-server:3307/coin-exchange  username: root  password: Ltd3411??  redis:  host: redis-server  port: 6380  password: Ltd3411??  cloud:  nacos:  discovery:  server-addr: nacos-server:8848  config:  enabled: false  file-extension: yaml  sentinel:  transport:  dashboard: sentinel-dashboard:8858  mybatis-plus:  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  mapper-locations: classpath:/mapper/\*Mapper.xml jetcache:  statIntervalMinutes: 15  areaInCacheName: false  local:  default:  type: linkedhashmap  keyConvertor: fastjson  remote:  default:  type: redis  keyConvertor: fastjson  valueEncoder: kryo  valueDecoder: kryo  poolConfig:  minIdle: 5  maxIdle: 20  maxTotal: 50  host: ${spring.redis.host}  port: ${spring.redis.port}  password: ${spring.redis.password} |

## 10.3 实现一个测试类的Controller

|  |
| --- |
| @RestController @Api(tags = "CoinCommon里面测试的接口") public class TestController {   @Autowired  private ObjectMapper objectMapper ;   @GetMapping("/common/test")  @ApiOperation(value = "测试方法", authorizations = {@Authorization("Authorization")})  @ApiImplicitParams({  @ApiImplicitParam(name = "param", value = "参数1", dataType = "String", paramType = "query", example = "paramValue"),  @ApiImplicitParam(name = "param1", value = "参数2", dataType = "String", paramType = "query", example = "paramValue")  })  public R<String> testMethod(String param, String param1) {   return R.*ok*("ok");  }   } |

## 10.4 Swagger2测试



## 10.4 日志切面测试



## 10.5 日期格式化测试

在Controller 里面添加：

|  |
| --- |
| @GetMapping("/common/date") @ApiOperation(value = "日志格式化测试", authorizations = {@Authorization("Authorization")}) public R<Date> testMethod() {  return R.*ok*(new Date()); } |

## 10.6 缓存测试

### 10.6.1 新建接口

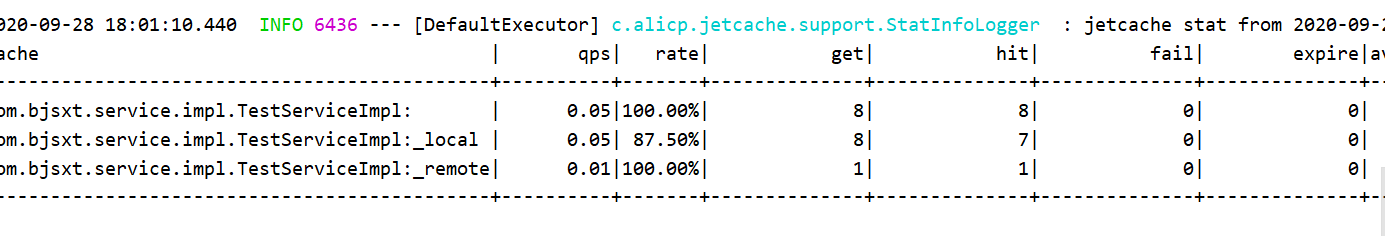
|  |
| --- |
| public interface TestService {   */\*\*  \* 通过username 查询weblog  \*  \*/* WebLog get(String username) ; } |

### 10.6.2 新建实现类

|  |
| --- |
| @Service public class TestServiceImpl implements TestService {   */\*\*  \* 通过username 查询webLog  \* @param username  \* @return  \*/* @Cached(name = "com.bjsxt.service.impl.TestServiceImpl:",key = "#username" ,cacheType = CacheType.*BOTH*)  public WebLog get(String username) {  WebLog webLog = new WebLog();  webLog.setUsername(username);  webLog.setResult("ok");  return webLog;  } } |

### 10.6.3 Controller 添加测试

|  |
| --- |
| @GetMapping("/jetcache/test") @ApiOperation(value = "jetcache缓存的测试",authorizations = {@Authorization("Authorization")}) public R<String> testJetCache(String username){  WebLog webLog = testService.get(username);  System.*out*.println(webLog);  return R.*ok*("OK") ; *//* } |



## 10.5 RedisTemplate<String,Object> 测试

|  |
| --- |
| @Autowired private RedisTemplate<String,Object> redisTemplate ; |

|  |
| --- |
| @GetMapping("/redis/test") @ApiOperation(value = "redis测试",authorizations = {@Authorization("Authorization")}) public R<String> testRedis(){  WebLog webLog = new WebLog();  webLog.setResult("ok");  webLog.setMethod("com.bjsxt.domain.WebLog.testRedis");  webLog.setUsername("1110");  redisTemplate.opsForValue().set("com.bjsxt.domain.WebLog",webLog);  return R.*ok*("OK") ; *//* } |



# 工具类合集

11.1 IpUtils

11.2

# MybaitsPlus日期自动填充

当我们在新增一个数据时、修改一个数据时，需要给表里面的日期字段填充一个值，我们可以把该过程封装起来。以后使用MybatisPlus提供的注解即可：

|  |
| --- |
| */\*\*  \*字段自动填充  \*/* @Component public class AutoFiledValueHandler implements MetaObjectHandler {    */\*\*  \* 新增时填入值  \* @param metaObject  \*/* @Override  public void insertFill(MetaObject metaObject) {  Long userId = getUserId();  */\*\*  \* 3 种情况不填充  \* 1 值为null  \* 2 自动类型不匹配  \* 3 没有改字段  \*/* this.strictInsertFill(metaObject, "lastUpdateTime", Date.class, new Date());  this.strictInsertFill(metaObject, "createBy", Long.class, userId); *// 创建人的填充* this.strictInsertFill(metaObject, "created", Date.class, new Date());   }    */\*\*  \* 修改时填入值  \* @param metaObject  \*/* @Override  public void updateFill(MetaObject metaObject) {  Long userId = getUserId();  this.strictInsertFill(metaObject, "lastUpdateTime", Date.class, new Date());  this.strictInsertFill(metaObject, "modifyBy", Long.class, userId); *// 修改人的填充* }   */\*\*  \* 获取安全上下文里的用户对象 --- 主要是在线程里面获取改值  \* @return  \*/* private Long getUserId() {  Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  Long userId = null;  if (authentication != null) {  String principal = authentication.getPrincipal().toString();  userId = Long.*valueOf*(principal);  }  return userId;  } } |