# 说明

# 版权归属：鑫学猿公司

微信：yuanhouduanyanfa

Java开发工程师转行辅导体系,保障拿到java工程师offer

图形用户界面, 文本, 应用程序

描述已自动生成

**SpringBoot项目AOP统一打印**

**背景**

在实际Web项目中，我们通常都会在比如请求进来时或者返回后能够记录日志方便日后排查问题等。甚至有时候我们需要在内部的某些方法逻辑执行前、执行后做一些操作，比如：对参数进行解密，对所有的返回值进行加密等。这些与业务无关的操作，传统做法就是在每个期望的地方都去手动记录日志，但有了AOP的思想和实现，其实没有必要在每个方法中都写一遍，这里我们有两种方式实现，一是自定义个注解和切面方法来实现，可以适用任何你希望的方法上；二是针对controller的方法可以使用 springmvc 中的@ControllerAdvice 和 RequestBodyAdvice、ResponseBodyAdvice 来对请求前后进行处理，本质上也是 aop 的思想

**自定义方式**

自定义注解和切面方法，在方法执行开始之前和结束之后，打印出参、入参日志

**RestControllerAdvice方式**

RequestBodyAdvice：会对被@RquestBody和@PostMapping（@PutMapping等也生效）注解的请求进行增强处理，比如所有请求的数据都加密之后放在 body 中，在到达 controller 的方法之前，需要先进行解密，那么就可以通过 RequestBodyAdvice 来进行统一的解密处理，无需在 controller 方法中去做这些通用的操作。

ResponseBodyAdvice：类似RequestBodyAdvice，对 Controller 中@ResponseBody 类型返回值进行增强处理，也就是说可以拦截@ResponseBody 类型的返回值，进行再次处理，比如加密、日志等操作

**教程**

**自定义方式**

1. 引入依赖

|  |
| --- |
| XML <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-aop</artifactId> </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.73</version> </dependency> |

1. 定义切点

|  |
| --- |
| Java @Retention(RetentionPolicy.RUNTIME) @Target({ElementType.METHOD}) @Documented public @interface MethodLog { } |

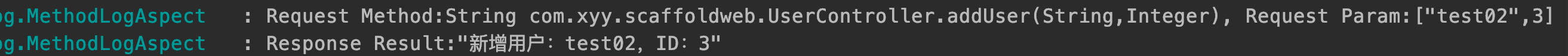
1. 编写切面方法

|  |
| --- |
| Java @Aspect @Slf4j @Component public class MethodLogAspect {   /\*\*  \* 自定义 @MethodLog 切点  \*/  @Pointcut("@annotation(com.xyy.scaffoldcommon.log.MethodLog)")  public void MethodLog() {  }   @Before("MethodLog()")  public void doBefore(JoinPoint joinPoint) {  log.info("Request Method:{}, Request Param:{}", joinPoint.getSignature(), JSON.toJSONString(joinPoint.getArgs()));  }   @AfterReturning(returning = "o", pointcut = "MethodLog()")  public void doAfterReturning(Object o) {  log.info("Response Result:{}", JSONObject.toJSONString(o));  } } |

1. 编写controller

|  |
| --- |
| Java @RestController @Api(tags = "用户数据接口") public class UserController {  @Resource  private UserService userService;    @MethodLog  @ApiOperation(value = "查询所有用户", notes = "查询所有用户")  @GetMapping("/user/all")  public Result<List<UserDTO>> queryAllUser() {  List<UserDTO> users = userService.queryAll();  if (CollectionUtils.isEmpty(users)) {  return Result.Result(ErrorEnum.DATA\_NOT\_FOUND, null);  }  return Result.Result(ErrorEnum.SUCCESS, users);  }   @ApiOperation(value = "新增用户", notes = "根据用户名和ID添加新用户")  @ApiImplicitParams({  @ApiImplicitParam(paramType = "query", name = "userName", value = "用户名",  required = true),  @ApiImplicitParam(paramType = "query", name = "userID", value = "用户ID",  required = true)  })  @PostMapping("/user")  @MethodLog  public String addUser(@RequestParam String userName, @RequestParam Integer userID) {  userService.createUser(userName);  return "新增用户：" + userName + "，ID：" + userID;  } } |

1. 演示



**RestControllerAdvice方式**

1. 日志依赖引入，这里我们日志框架就简单使用slf4j和logback

|  |
| --- |
| 如果引了mybatis-spring-boot-starter都会包含前两个包，只需要单独引入lombok即可 |

|  |
| --- |
| XML <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>1.7.25</version>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-classic</artifactId>  <version>1.2.3</version>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>1.18.20</version>  <scope>compile</scope>  </dependency> |

1. 编写Request请求日志代码

|  |
| --- |
| Java @Slf4j @RestControllerAdvice public class LogTrace4Request implements RequestBodyAdvice {  /\*\*  \* 该方法用于判断当前请求，是否要执行beforeBodyRead方法  \* methodParameter 方法的参数对象  \* type 方法的参数类型  \* aClass 将会使用到的Http消息转换器类类型  \* 注意：此判断方法，会在beforeBodyRead 和 afterBodyRead方法前都触发一次。  \*  \* @return 返回true则会执行beforeBodyRead  \*/  @Override  public boolean supports(MethodParameter methodParameter, Type type, Class<? extends HttpMessageConverter<?>> aClass) {  return true;  }   /\*\*  \* 在Http消息转换器执转换，之前执行  \* inputMessage 客户端的请求数据  \* parameter方法的参数对象  \* targetType方法的参数类型  \* converterType 将会使用到的Http消息转换器类类型  \*  \* @return 返回 一个自定义的HttpInputMessage  \*/  @Override  public HttpInputMessage beforeBodyRead(HttpInputMessage httpInputMessage, MethodParameter methodParameter, Type type, Class<? extends HttpMessageConverter<?>> aClass) throws IOException {  return httpInputMessage;  }   /\*\*  \* 在Http消息转换器执转换，之后执行  \* body 转换后的对象  \* inputMessage 客户端的请求数据  \* parameter handler方法的参数类型  \* targetType handler方法的参数类型  \* converterType 使用的Http消息转换器类类型  \* @return 返回一个新的对象  \*/  @Override  public Object afterBodyRead(Object o, HttpInputMessage httpInputMessage, MethodParameter methodParameter, Type type, Class<? extends HttpMessageConverter<?>> aClass) {  if (Objects.isNull(o)) {  log.info("请求追踪:请求参数为空");  } else {  log.info("请求追踪:{}", o);  }  return o;  }   /\*\*  \* 参数与afterBodyRead相同，不过这个方法处理的是，body为空的情况  \*/  @Override  public Object handleEmptyBody(Object o, HttpInputMessage httpInputMessage, MethodParameter methodParameter, Type type, Class<? extends HttpMessageConverter<?>> aClass) {  return o;  } } |

1. 编写response响应日志代码

|  |
| --- |
| Java @RestControllerAdvice @Slf4j public class LogTraceResponse implements ResponseBodyAdvice {  @Override  public boolean supports(MethodParameter methodParameter, Class aClass) {  return true;  }   @Override  public Object beforeBodyWrite(Object o, MethodParameter methodParameter, MediaType mediaType, Class aClass, ServerHttpRequest serverHttpRequest, ServerHttpResponse serverHttpResponse) {  if (Objects.isNull(o)) {  log.info("响应追踪:请求参数为空");  } else {  log.info("响应追踪:{}", o);  }  return o;  } } |

1. 编写controller

|  |
| --- |
| Java @RestController @Api(tags = "用户数据接口") public class UserController {  @Resource  private UserService userService;   @ApiOperation(value = "查询用户", notes = "根据姓名查询用户")  @PostMapping("/user/query")  public Result<UserDTO> queryUserByName(@RequestBody UserQueryRequest request) {  UserDTO user = userService.queryUserByName(request.getName());  if (Objects.isNull(user)) {  return Result.Result(ErrorEnum.DATA\_NOT\_FOUND, null);  }  return Result.Result(ErrorEnum.SUCCESS, user);  }   @ApiOperation(value = "修改用户信息", notes = "更新用户信息")  @PutMapping("/user")  public String updateUser(@RequestBody UserDTO userDTO) {  return "成功";  } } |

1. 演示

