链滴

SpringBoot 利用 AOP 记录日志

作者：jianzh5

原文链接：https://ld246.com/article/1589330752488

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)

为什么要用AOP?

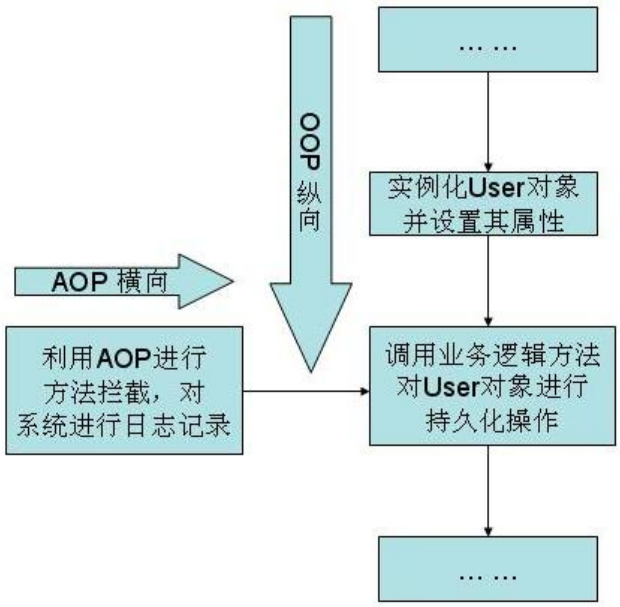
答案是**解耦**!

Aspect Oriented Programming 面向切面编程。解耦是程序员编码开发过程中一直追求的。AOP也为了解耦所诞生。

具体思想是：定义一个切面，在切面的纵向定义处理方法，处理完成之后，回到横向业务流。

AOP 主要是利用代理模式的技术来实现的。具体的代理实现可以参考这篇文章，讲解的非常详细。https://www.cnblogs.com/yanbincn/archive/2012/06/01/2530377.html

通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。利用 AOP 可以对业务逻辑各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了发的效率。



常用的工作场景

1. 事务控制
2. 日志记录

本文没有过度深度学习原理，因为是菜鸟一个，先学会怎么不加班。

必须知道的概念

AOP 的相关术语

通知（Advice）

通知描述了切面要完成的工作以及何时执行。比如我们的日志切面需要记录每个接口调用时长，就需

在接口调用前后分别记录当前时间，再取差值。

- 前置通知（Before）：在目标方法调用前调用通知功能；
- 后置通知（After）：在目标方法调用之后调用通知功能，不关心方法的返回结果；
- 返回通知（AfterReturning）：在目标方法成功执行之后调用通知功能；
- 异常通知（AfterThrowing）：在目标方法抛出异常后调用通知功能；
- 环绕通知（Around）：通知包裹了目标方法，在目标方法调用之前和之后执行自定义的行为。

连接点（JoinPoint）

通知功能被应用的时机。比如接口方法被调用的时候就是日志切面的连接点。

切点（Pointcut）

切点定义了通知功能被应用的范围。比如日志切面的应用范围就是所有接口，即所有 controller 层的口方法。

切面（Aspect）

切面是通知和切点的结合，定义了何时、何地应用通知功能。

引入（Introduction）

在无需修改现有类的情况下，向现有的类添加新方法或属性。

织入（Weaving）

把切面应用到目标对象并创建新的代理对象的过程。

Spring 中使用注解创建切面

相关注解

- @Aspect：用于定义切面
- @Before：通知方法会在目标方法调用之前执行
- @After：通知方法会在目标方法返回或抛出异常后执行
- @AfterReturning：通知方法会在目标方法返回后执行
- @AfterThrowing：通知方法会在目标方法抛出异常后执行
- @Around：通知方法会将目标方法封装起来
- @Pointcut：定义切点表达式

切点表达式

指定了通知被应用的范围，表达式格式：

execution
(方法修饰符

返回类型

方法所属的包.类名.方法名称(方法参数)
//com.ninesky.study.tiny.controller包中所有类的public方法都应用切面里的通知
execution(public * com.ninesky.study.tiny.controller.*.*(..))
//com.ninesky.study.tiny.service包及其子包下所有类中的所有方法都应用切面里的通知
execution(* com.ninesky.study.tiny.service..*.*(..))
//com.ninesky.study.tiny.service.PmsBrandService类中的所有方法都应用切面里的通知
execution(* com.macro.ninesky.study.service.PmsBrandService.*(..))

实战应用-利用AOP记录日志

从传统行业转行，以前都没想过打日志埋点，第一份工作，真的应该选择一个好的平台比较重要。

定义日志信息封装

用于封装需要记录的日志信息，包括操作的描述、时间、消耗时间、url、请求参数和返回结果等信息

```
public class WebLog {
    /**
     * 操作描述
     */
    private String description;
    /**
     * 操作用户
     */
    private String username;
    /**
     * 操作时间
     */
    private Long startTime;
    /**
     * 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;
    /**
     * 请求返回的结果
     */
    private Object result;
    //省略了getter,setter方法
}
```

定义注解,通过注解减少代码量

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface OperationLog {
    String name();//调用接口的名称

    boolean intoDb() default false;//该条操作日志是否需要持久化存储
}
```

统一日志处理切面

```
@Aspect
@Component
@Order(1)
@Slf4j
public class WebLogAspect {
    private static final Logger controlLog = LoggerFactory.getLogger("tmall_control");
    @Pointcut("execution(public * com.yee.walnut.*.*.*(..))")
    public void webLog() {
    }

    @Before(value = "webLog()&& @annotation(OperationLog)")
    public void doBefore(ControllerWebLog controllerWebLog) throws Throwable {
    }

    @AfterReturning(value = "webLog()&& @annotation(OperationLog)", returning = "ret")
    public void doAfterReturning(Object ret, ControllerWebLog controllerWebLog) throws Thr
wable {
    }

    @Around(value = "webLog()&& @annotation(OperationLog)")
    public Object doAround(ProceedingJoinPoint joinPoint, OperationLog operationLog) throw
 Throwable {
        long startTime = System.currentTimeMillis();
```

```
        //获取当前请求对象
        ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.
etRequestAttributes();
        HttpServletRequest request = attributes.getRequest();
        //记录请求信息
        Object[] objs = joinPoint.getArgs();
        WebLog webLog = new WebLog();
        Object result = joinPoint.proceed();//返回的结果，这是一个进入方法和退出方法的一个分界
        Signature signature = joinPoint.getSignature();
        MethodSignature methodSignature = (MethodSignature) signature;
        Method method = methodSignature.getMethod();
        long endTime = System.currentTimeMillis();
        String urlStr = request.getRequestURL().toString();
        webLog.setBasePath(StrUtil.removeSuffix(urlStr, URLUtil.url(urlStr).getPath()));
        webLog.setIp(request.getRemoteUser());
        webLog.setMethod(request.getMethod());
        webLog.setParameter(getParameter(method, joinPoint.getArgs()));
        webLog.setResult(JSONUtil.parse(result));
        webLog.setSpendTime((int) (endTime - startTime));
        webLog.setStartTime(startTime);
        webLog.setUri(request.getRequestURI());
        webLog.setUrl(request.getRequestURL().toString());
        controlLog.info("RequestAndResponse {}", JSONObject.toJSONString(webLog));
        //必须有这个返回值。可以这样理解，Around方法之后，不再是被织入的函数返回值，而是Aro
nd函数返回值
        return result;
    }


    /**
     * 根据方法和传入的参数获取请求参数
     */
    private Object getParameter(Method method, Object[] args) {
        List<Object> argList = new ArrayList<>();
        Parameter[] parameters = method.getParameters();
        for (int i = 0; i < parameters.length; i++) {
            //将RequestBody注解修饰的参数作为请求参数
            RequestBody requestBody = parameters[i].getAnnotation(RequestBody.class);
            if (requestBody != null) {
                argList.add(args[i]);
            }
            //将RequestParam注解修饰的参数作为请求参数
            RequestParam requestParam = parameters[i].getAnnotation(RequestParam.class);
            if (requestParam != null) {
                Map<String, Object> map = new HashMap<>();
                String key = parameters[i].getName();
                if (!StringUtils.isEmpty(requestParam.value())) {
                    key = requestParam.value();
                }
                map.put(key, args[i]);
                argList.add(map);
            } else {
                argList.add(args[i]);
            }
```

```
        }
        if (argList.size() == 0) {
            return null;
        } else if (argList.size() == 1) {
            return argList.get(0);
        } else {
            return argList;
        }
    }
}
```

在方法上加上自定义注解即可

```
@OperationLog(name = "TurnOnOffStrategy")
public String doOperation(GlobalDto globalDto, DeviceOperator deviceOperator) {
}
```