

新写了个牛逼的 AOP 日志切面，甩锅更方便了！

IT码徒 2023-05-08 22:58 Posted on 河南

点击“IT码徒”，关注，置顶公众号  
每日技术干货，第一时间送达！



Scan to Follow

耗时8个月联合打造 《2023年Java高薪课程》，已更新了 102G 视频，累计更新时长 500+ 个小时，需要的小伙伴可以了解下，一次购买，持续更新，无需2次付费。

- 前言
- 切面介绍
- 切面的使用【基于注解】
- 动手写一个请求日志切面
- 高并发下请求日志切面



最近项目进入联调阶段，服务层的接口需要和协议层进行交互，协议层需要将入参[json字符串]组装成服务层所需的json字符串，组装的过程中很容易出错。入参出错导致接口调试失败问题在联调中出现很多次，因此就想写一个请求日志切面把入参信息打印一下，同时协议层调用服务层接口名称对不上也出现了几次，通过请求日志切面就可以知道上层是否有发起调用，方便前后端甩锅还能拿出证据

PS：本篇文章是实战性的，对于切面的原理不会讲解，只会简单介绍一下切面的知识点

切面介绍

面向切面编程是一种编程范式，它作为OOP面向对象编程的一种补充，用于处理系统中分布于各个模块的横切关注点，比如事务管理、权限控制、缓存控制、日志打印 等等。AOP把软件的功能模块分为两个部分：核心关注点和横切关注点。业务处理的主要功能为核心关注点，而非核心、需要拓展的功能为横切关注点。AOP的作用在于分离系统中的各种关注点，将核心关注点和横切关注点进行分离，使用切面有以下好处：

- 集中处理某一关注点/横切逻辑
- 可以很方便的添加/删除关注点
- 侵入性少，增强代码可读性及可维护性 因此当想打印请求日志时很容易想到切面，对控制层代码0侵入

切面的使用【基于注解】

- @Aspect => 声明该类为一个注解类

切点注解：

- @Pointcut => 定义一个切点，可以简化代码

通知注解：

- @Before => 在切点之前执行代码
- @After => 在切点之后执行代码
- @AfterReturning => 切点返回内容后执行代码，可以对切点的返回值进行封装
- @AfterThrowing => 切点抛出异常后执行
- @Around => 环绕，在切点前后执行代码

动手写一个请求日志切面

- 使用@Pointcut定义切点

```
@Pointcut("execution(* your_package.controller..*(..))")
public void requestServer() {
}
```

@Pointcut定义了一个切点，因为是请求日志切边，因此切点定义的是Controller包下的所有类下的方法。定义切点以后在通知注解中直接使用requestServer方法名就可以了

- 使用@Before再切点前执行

```
@Before("requestServer()")
public void doBefore(JoinPoint joinPoint) {
    ServletRequestAttributes attributes = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
    HttpServletRequest request = attributes.getRequest();

    LOGGER.info("=====Start=====");
    LOGGER.info("IP          : {}", request.getRemoteAddr());
    LOGGER.info("URL          : {}", request.getRequestURL().toString());
    LOGGER.info("HTTP Method   : {}", request.getMethod());
    LOGGER.info("Class Method  : {}.{}", joinPoint.getSignature().getDeclari
}
}
```

在进入Controller方法前，打印出调用方IP、请求URL、HTTP请求类型、调用的方法名

- 使用@Around打印进入控制层的入参

```
@Around("requestServer()")
public Object doAround(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
    long start = System.currentTimeMillis();
    Object result = proceedingJoinPoint.proceed();

    LOGGER.info("Request Params   : {}", getRequestParams(proceedingJoinPoint));
    LOGGER.info("Result          : {}", result);
    LOGGER.info("Time Cost       : {} ms", System.currentTimeMillis() - start);

    return result;
}
```

打印了入参、结果以及耗时

- getRequestParams方法

```
private Map<String, Object> getRequestParams(ProceedingJoinPoint proceedingJoinPoint) {
    Map<String, Object> requestParams = new HashMap<>();

    //参数名
    String[] paramNames = ((MethodSignature)proceedingJoinPoint.getSignature()).getParameterNames();
    //参数值
    Object[] paramValues = proceedingJoinPoint.getArgs();

    for (int i = 0; i < paramNames.length; i++) {
        Object value = paramValues[i];

        //如果是文件对象
        if (value instanceof MultipartFile) {
            MultipartFile file = (MultipartFile) value;
            value = file.getOriginalFilename(); //获取文件名
        }

        requestParams.put(paramNames[i], value);
    }

    return requestParams;
}
```

通过 @PathVariable以及@RequestParam注解传递的参数无法打印出参数名，因此需要手动拼接一下参数名，同时对文件对象进行了特殊处理，只需获取文件名即可

- @After方法调用后执行

```
@After("requestServer()")
public void doAfter(JoinPoint joinPoint) {
    LOGGER.info("=====End=====");
}
```

没有业务逻辑只是打印了End

- 完整切面代码

```
@Component
@Aspect
public class RequestLogAspect {
    private final static Logger LOGGER = LoggerFactory.getLogger(RequestLogAspect.class);

    @Pointcut("execution(* your_package.controller..*(..))")
    public void requestServer() {
    }

    @Before("requestServer()")
    public void doBefore(JoinPoint joinPoint) {
        ServletRequestAttributes attributes = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
        HttpServletRequest request = attributes.getRequest();

        LOGGER.info("=====Start=====");
        LOGGER.info("IP          : {}", request.getRemoteAddr());
        LOGGER.info("URL          : {}", request.getRequestURL().toString());
        LOGGER.info("HTTP Method   : {}", request.getMethod());
        LOGGER.info("Class Method  : {}.{}", joinPoint.getSignature().getDeclaringClass().getName(),
joinPoint.getSignature().getName());
    }

    @Around("requestServer()")
    public Object doAround(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
        long start = System.currentTimeMillis();
        Object result = proceedingJoinPoint.proceed();
        LOGGER.info("Request Params   : {}", getRequestParams(proceedingJoinPoint));
        LOGGER.info("Result          : {}", result);
        LOGGER.info("Time Cost       : {} ms", System.currentTimeMillis() - start);

        return result;
    }

    @After("requestServer()")
    public void doAfter(JoinPoint joinPoint) {
        LOGGER.info("=====End=====");
    }

    /**
     * 获取入参
     * @param proceedingJoinPoint
     *
     * @return
     * */
    private Map<String, Object> getRequestParams(ProceedingJoinPoint proceedingJoinPoint) {
        Map<String, Object> requestParams = new HashMap<>();

        //参数名
        String[] paramNames =
((MethodSignature)proceedingJoinPoint.getSignature()).getParameterNames();
        //参数值
        Object[] paramValues = proceedingJoinPoint.getArgs();

        for (int i = 0; i < paramNames.length; i++) {
            Object value = paramValues[i];

            //如果是文件对象
            if (value instanceof MultipartFile) {
                MultipartFile file = (MultipartFile) value;
                value = file.getOriginalFilename(); //获取文件名
            }
        }
    }
}
```



```
        requestParams.put(paramNames[i], value);
    }

    return requestParams;
}
}
```

高并发下请求日志切面

写完以后对自己的代码很满意，但是想着可能还有完善的地方就和朋友交流了一下。emmmm



果然还有继续优化的地方 每个信息都打印一行，在高并发请求下确实会出现请求之间打印日志串行的问题，因为测试阶段请求数量较少没有出现串行的情况，果然生产环境才是第一发展力，能够遇到更多bug，写更健壮的代码 解决日志串行的问题只要将多行打印信息合并为一行就可以了，因此构造一个对象

- RequestInfo.java

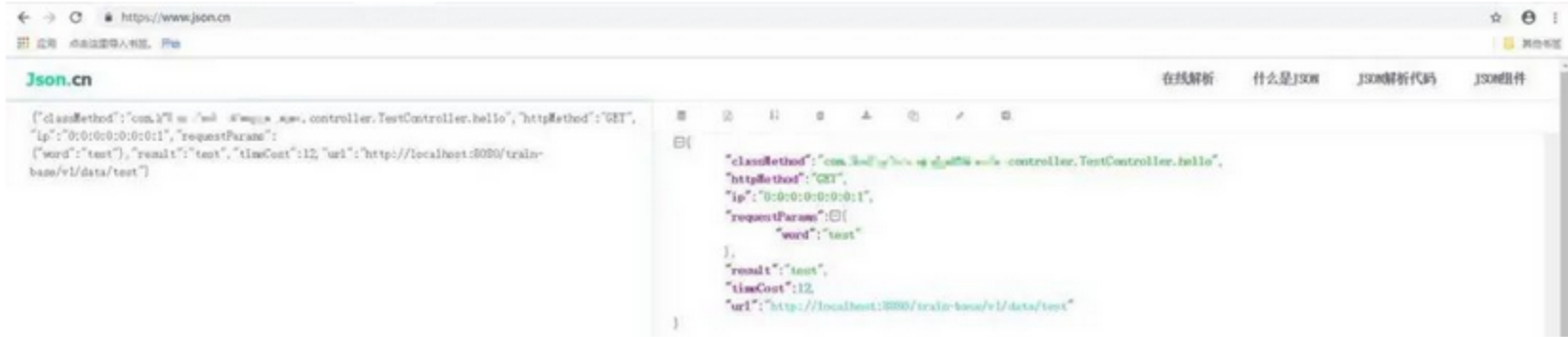
```
@Data
public class RequestInfo {
    private String ip;
    private String url;
    private String httpMethod;
    private String classMethod;
    private Object requestParams;
    private Object result;
    private Long timeCost;
}
```

- 环绕通知方法体

```
@Around("requestServer()")
public Object doAround(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
    long start = System.currentTimeMillis();
    ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
    HttpServletRequest request = attributes.getRequest();
    Object result = proceedingJoinPoint.proceed();
    RequestInfo requestInfo = new RequestInfo();
    requestInfo.setIp(request.getRemoteAddr());
    requestInfo.setUrl(request.getRequestURL().toString());
    requestInfo.setHttpMethod(request.getMethod());
    requestInfo.setClassMethod(String.format("%s.%s", proceedingJoinPoint.getSignature().getDeclaringTypeName(),
        proceedingJoinPoint.getSignature().getName()));
    requestInfo.setRequestParams(getRequestParamsByProceedingJoinPoint(proceedingJoinPoint));
    requestInfo.setResult(result);
    requestInfo.setTimeCost(System.currentTimeMillis() - start);
    LOGGER.info("Request Info      : {}", JSON.toJSONString(requestInfo));

    return result;
}
```

将url、http request这些信息组装成RequestInfo对象，再序列化打印对象 打印序列化 对象结果而不是直接打印对象是因为序列化有更直观、更清晰，同时可以借助在线解析工具对结果进行解析



是不是还不错

在解决高并发下请求串行问题的同时添加了对异常请求信息的打印， 通过使用 @AfterThrowing注解对抛出异常的方法进行处理

- RequestErrorInfo.java

```
@Data
public class RequestErrorInfo {
    private String ip;
    private String url;
    private String httpMethod;
    private String classMethod;
    private Object requestParams;
    private RuntimeException exception;
}
```

- 异常通知环绕体

```
@AfterThrowing(pointcut = "requestServer()", throwing = "e")
public void doAfterThrow(JoinPoint joinPoint, RuntimeException e) {
    ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
    HttpServletRequest request = attributes.getRequest();
    RequestErrorInfo requestErrorInfo = new RequestErrorInfo();
    requestErrorInfo.setIp(request.getRemoteAddr());
    requestErrorInfo.setUrl(request.getRequestURL().toString());
    requestErrorInfo.setHttpMethod(request.getMethod());
    requestErrorInfo.setClassMethod(String.format("%s.%s", joinPoint.getSignature().getDeclaringClass().getName(),
        joinPoint.getSignature().getName()));
    requestErrorInfo.setRequestParams(getRequestParamsByJoinPoint(joinPoint));
    requestErrorInfo.setException(e);
    LOGGER.info("Error Request Info      : {}", JSON.toJSONString(requestErrorInfo));
}
```

对于异常，耗时是没有意义的，因此不统计耗时，而是添加了异常的打印

最后放一下完整日志请求切面代码：

```
@Component
@Aspect

public class RequestLogAspect {

    private final static Logger LOGGER = LoggerFactory.getLogger(RequestLogAspect.class);

    @Pointcut("execution(* your.package.controller..*(..))")

    public void requestServer() {

    }

    @Around("requestServer()")

    public Object doAround(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {

        long start = System.currentTimeMillis();

        ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
        HttpServletRequest request = attributes.getRequest();
        Object result = proceedingJoinPoint.proceed();
        RequestInfo requestInfo = new RequestInfo();
        requestInfo.setIp(request.getRemoteAddr());
        requestInfo.setUrl(request.getRequestURL().toString());
        requestInfo.setHttpMethod(request.getMethod());
        requestInfo.setClassMethod(String.format("%s.%s", proceedingJoinPoint.getSignature().getDeclaringClass().getName(),
            proceedingJoinPoint.getSignature().getName()));
        requestInfo.setRequestParams(getRequestParamsByProceedingJoinPoint(proceedingJoinPoint));
        requestInfo.setResult(result);
        requestInfo.setTimeCost(System.currentTimeMillis() - start);
        LOGGER.info("Request Info      : {}", JSON.toJSONString(requestInfo));

        return result;
    }

    @AfterThrowing(pointcut = "requestServer()", throwing = "e")

    public void doAfterThrow(JoinPoint joinPoint, RuntimeException e) {

        ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
        HttpServletRequest request = attributes.getRequest();
        RequestErrorInfo requestErrorInfo = new RequestErrorInfo();
        requestErrorInfo.setIp(request.getRemoteAddr());
        requestErrorInfo.setUrl(request.getRequestURL().toString());
        requestErrorInfo.setHttpMethod(request.getMethod());
        requestErrorInfo.setClassMethod(String.format("%s.%s", joinPoint.getSignature().getDeclaringClass().getName(),
            joinPoint.getSignature().getName()));
        requestErrorInfo.setRequestParams(getRequestParamsByJoinPoint(joinPoint));
        requestErrorInfo.setException(e);
        LOGGER.info("Error Request Info      : {}", JSON.toJSONString(requestErrorInfo));
    }

    /**
     * 获取入参
     * @param proceedingJoinPoint
     *
     * @return
     * */

    private Map<String, Object> getRequestParamsByProceedingJoinPoint(ProceedingJoinPoint proceedingJoinPoint) {

        //参数名
        String[] paramNames = ((MethodSignature)proceedingJoinPoint.getSignature()).getParameterNames();
        //参数值
        Object[] paramValues = proceedingJoinPoint.getArgs();

        return buildRequestParam(paramNames, paramValues);
    }

    private Map<String, Object> getRequestParamsByJoinPoint(JoinPoint joinPoint) {

        //参数名
        String[] paramNames = ((MethodSignature)joinPoint.getSignature()).getParameterNames();
        //参数值
        Object[] paramValues = joinPoint.getArgs();
    }
}
```



```
        return buildRequestParam(paramNames, paramValues);
    }

    private Map<String, Object> buildRequestParam(String[] paramNames, Object[] paramValues) {
        Map<String, Object> requestParams = new HashMap<>();
        for (int i = 0; i < paramNames.length; i++) {
            Object value = paramValues[i];

            //如果是文件对象
            if (value instanceof MultipartFile) {
                MultipartFile file = (MultipartFile) value;
                value = file.getOriginalFilename(); //获取文件名
            }

            requestParams.put(paramNames[i], value);
        }

        return requestParams;
    }

    @Data
    public class RequestInfo {

        private String ip;

        private String url;

        private String httpMethod;

        private String classMethod;

        private Object requestParams;

        private Object result;

        private Long timeCost;

    }

    @Data
    public class RequestErrorInfo {

        private String ip;

        private String url;

        private String httpMethod;

        private String classMethod;

        private Object requestParams;

        private RuntimeException exception;

    }
}
```


赶紧给你们的应用加上吧【如果没加的话】，没有日志的话，总怀疑上层出错，但是却拿不出证据。

文章来源：<https://juejin.cn/post/6844904087964614670>

— END —

[【福利】2023 高薪课程，全面来袭（视频+笔记+源码）](#)

**PS：防止找不到本篇文章，可以收藏点赞，方便翻阅查找哦。**



IT码徒

专注Java技术栈分享，多线程，JVM，io流，Spring，微服务，数据库等以及开源项目，视...

>

公众号

往期推荐
再见了 shiro
IDE + ChatGPT，这款编辑器真的做到可以自动写代码了
搞定 OAuth 2.0 第三方登录，So Easy !
一行代码搞定Http请求，强得离谱
Java 17 采用率在一年内增长 430% !
Spring Batch 批处理框架，真心强呀!!
几种分布式ID解决方案，总有一款适合你!

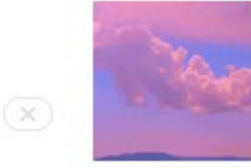
Read more

People who liked this content also liked

Lsposed 技术原理探讨 && 基本安装使用  
数字云信息技术



C++ Best Practices (C++最佳实践)翻译与阅读笔记  
Qt教程



我帮朋友实现了兔女郎自由，现在白送给你  
穿梭在银河的喵喵

