链滴

java aop 实现 http 接口日志记录

作者：crick77
原文链接：https://ld246.com/article/1541226397969
来源网站：链滴

http log

本文旨在讨论如何利用aop打印springboot http日志，包括访问日志、接口返回response，以及异堆栈信息。

背景

为什么需要log

有过线上问题排查经历的同学都知道，日志可以给你提供最直白、最明确的信息。此外，通过对日志收集、审计，可以对BI以及系统健壮性提供建议。尤其是新系统上线之初，系统稳定性不足，更需要志监控。

为什么不实用Greys+access log

Greys 作为Java线上问题诊断工具，可以监控实时访问参数，但是对历史访问无记录，尤其是在前后离or微服务架构下，日志是可以作为呈堂证供的存在。而access log不够灵活和强大。

方案优势

1. 简化日志打印
2. 统一的日志格式，可以配合收集机制
3. 灵活，可以自由选择是否记录日志、是否忽略response、是否忽略某些参数（如file参数显然不适记录日志）、以及是否打印header参数
4. 日志记录比例可配置（系统稳定，流量增大之后，可以通过配置降低记录比例）
5. 方便回溯异常请求参数
6. 记录接口耗时

实现

AOP-面向切面编程

一句话描述，在java对象方法中增加切点，在不改变对象领域的前提下通过代理扩展功能。

切面选择

网上常见教程会选择拦截所有http请求，并打印request.parameters。但是存在问题：

1. 不够灵活，部分参数不想打印，如文件数据（过大）、敏感数据（身份证）等。
2. 隐式的aop，容易给维护人员造成疑惑。(个人习惯偏向于避免隐藏的aop逻辑)
3. 部分参数通过header传输

因此，自定义日志输出@annotation **HttpLog**作为切点

```
/**
```

```
 * 用于打印http请求访问日志
 *
 * @author yuheng.wang
 */
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface HttpLog {
    /**
     * 忽略参数，避免文件or无意义参数打印
     *
     * @return 忽略参数数组
     */
    String[] exclude() default {};

    /**
     * 需要打印的header参数
     *
     * @return header参数名数组
     */
    String[] headerParams() default {};

    boolean ignoreResponse() default false;

    /**
     * 日志记录比例
     *  >100 or < 0时，每次请求都会被记录
     *  = 0 时，不会记录
     */
    int percent() default 100;
}


@Aspect
@Slf4j
public class HttpLogHandler {

    @Pointcut("@annotation(wang.crick.study.httplog.annotation.HttpLog)")
    public void logAnnotation() {
    }

    private Optional<HttpLog> getLogAnnotation(JoinPoint joinPoint) {
    if (joinPoint instanceof MethodInvocationProceedingJoinPoint) {
        Signature signature = joinPoint.getSignature();
        if (signature instanceof MethodSignature) {
            MethodSignature methodSignature = (MethodSignature) signature;
            Method method = methodSignature.getMethod();
            if (method.isAnnotationPresent(HttpLog.class)) {
                return Optional.of(method.getAnnotation(HttpLog.class));
            }
        }
    }
        return Optional.empty();
    }
}
```

打印异常信息

1. 为了方便回溯，定义UUID形式的traceId，并通过ThreadLocal持有。
2. 为了记录接口相应时间，定义startTime，并通过ThreadLocal持有。
3. AfterThrowing方法不会阻碍Exception的抛出，不会影响全局异常捕获。

```
@AfterThrowing(throwing = "e", pointcut = "logAnnotation()")
public void throwing(JoinPoint joinPoint, Exception e) {
    try {
        Optional<HttpLog> httpLog = getLogAnnotation(joinPoint);
        httpLog.ifPresent(anno -> {
            if (!anno.ignoreResponse()) {
                log.info("ERROR_LOG. traceId:{}, cost:{}",
                        traceIdThreadLocal.get(), costTime(), e);
            }
        });
    } catch (Exception exception) {
        log.warn("print error log fail!", exception);
    } finally {
        startTimeThreadLocal.remove();
        traceIdThreadLocal.remove();
    }
}
```

打印response

1. 通过JSONObject.toJSONString()转换，因为restful接口常见为返回json格式。
2. 需要判断是否忽略了response打印

```
@AfterReturning(returning = "restApiResponse", pointcut = "logAnnotation()")
public void response(JoinPoint joinPoint, Object restApiResponse) {
    try {
        Optional<HttpLog> httpLog = getLogAnnotation(joinPoint);
        httpLog.ifPresent(anno -> {
            if (!anno.ignoreResponse()) {
                log.info("RESPONSE_LOG. traceId:{}, result:{}, cost:{}",
                        traceIdThreadLocal.get(), JSONObject.toJSONString(restApiResponse), costTim
());
            }
        });
    } catch (Exception exception) {
        log.warn("print response log fail!", exception);
    } finally {
        startTimeThreadLocal.remove();
        traceIdThreadLocal.remove();
    }
}
```

获取请求日志

获取HttpServletRequest

重点在于获取HttpServletRequest对象，而spring已经很贴心的在initContextHolders时通过Reques ContextHolder持有了RequestAttributes参数(基于ThreadLocal)。 所以我们可以很轻松的获取请参数

```
private HttpServletRequest getRequest() {
    RequestAttributes ra = RequestContextHolder.getRequestAttributes();
    ServletRequestAttributes sra = (ServletRequestAttributes) ra;
    return sra.getRequest();
}
```

获取RequestBody

！！上面获取参数的方式有一个缺陷，无法获取RequestBody参数，因为RequestBody是以流的形读取，只能被读取一次，如果在aop中的提前读取了内容信息，后面的业务方法则无法获取。

一般的解决方案为封装一层HttpServletRequestWrapper暂存流的信息，实现多次读取。但是此种案对需要修改HttpServletRequest，侵入性较大，所以我们采取另一种方案，基于aop+反射的特性取java方法参数，遍历判断是否为@RequestBody参数。

```
private Optional<Object> getRequestBodyParam(JoinPoint joinPoint){
    if (joinPoint instanceof MethodInvocationProceedingJoinPoint) {
        Signature signature = joinPoint.getSignature();
        if (signature instanceof MethodSignature) {
            MethodSignature methodSignature = (MethodSignature) signature;
            Method method = methodSignature.getMethod();
            Parameter[] methodParameters = method.getParameters();
            if (null != methodParameters
                    && Arrays.stream(methodParameters).anyMatch(p-> AnnotationUtils.findAnnota
ion(p, RequestBody.class) != null)) {
                return Optional.of(joinPoint.getArgs());
            }
        }
    }
    return Optional.empty();
}
```

获取uri

根据拦截规则不同，getServletPath()和request.getPathInfo()可能为空，简单的做一次健壮性判断。

```
private String getRequestPath(HttpServletRequest request) {
   return (null != request.getServletPath() && request.getServletPath().length() > 0)
           ? request.getServletPath() : request.getPathInfo();

}
```

日志记录百分比

1. 接口请求量一般为平稳持续
2. 并不需要十分精确的比例

所以通过接口访问时间戳(startTime)尾数进行判断

```
private boolean needLog(int percent, long startTimeMillis){
    if (percent< 0 || percent >= 100) {
        return true;
    } else if (percent == 0) {
        return false;
    } else {
        return percent > startTimeMillis % 100;
    }
}
```

使用

加载HttpLogAspect对象

1. 可以在@SpringBootApplication类下直接加载，也可以在HttpLogAspect中增加@Component解。

2. 不需要显示声明@EnableAspectJAutoProxy注解

```
@Bean
public HttpLogAspect httpLogAspect(){
    return new HttpLogAspect();
}
```

Controller配置日志打印

在controller中在接口方法中通过HttpLog配置，即可实现日志记录

```
@GetMapping("/log/pwdExcludeResponse/{id}")
@HttpLog(headerParams="username", ignoreResponse = true, percent = 50, exclude={"avata
"})
public RestApiResponse<User> getInfoWithPwd(@PathVariable("id") int id,
                                            @RequestParam("age") int age,
                                            @RequestParam("avatar") MultipartFile avatar,
                                            @RequestHeader("password") String password){
    User user = new User();
    user.setId(id);
    user.setPassword(password);
    user.setAge(age);
    return RestApiResponse.success(user);
}
```

效果

```
REQUEST_LOG. traceId:737001cf-f546-4092-9c77-f7762275ddcf. requestUrl: /user/log/7 -PAR
MS- age: [6],
RESPONSE_LOG. traceId:737001cf-f546-4092-9c77-f7762275ddcf, result:{"code":10000,"data":
"age":6,"id":7,"username":"8025373076402792850"}}, cost:171
REQUEST_LOG. traceId:8c907a7e-f36a-4ccc-a931-f167b55697ee. requestUrl: /user/log/pwd/0
PARAMS- age: [7],  -HEADER_PARAMS- password: 6460630360418207541,
RESPONSE_LOG. traceId:8c907a7e-f36a-4ccc-a931-f167b55697ee, result:{"code":10000,"data":
"id":0,"password":"6460630360418207541","username":"2250855379452327045"}}, cost:9
```