



链滴

Spring Boot 中使用自定义注解，AOP 切面打印出入参日志及 Dubbo 链路追踪透传 traceId

作者：[workJun](#)

原文链接：<https://ld246.com/article/1584690795023>

来源网站：[链滴](#)

许可协议：[署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

一，使用背景

开发排查系统问题用得最多的手段就是查看系统日志，在分布式环境中一般使用 ELK 来统一收集日志但是在并发大时使用日志定位问题还是比较麻烦，由于大量的其他用户/其他线程的日志也一起输出行其中导致很难筛选出指定请求的全部相关日志，以及下游线程/服务对应的日志。

二，解决思路

每个请求都使用一个唯一标识来追踪全部的链路显示在日志中，并且不修改原有的

使用Logback的MDC机制日志模板中加入traceld标识，取值方式为%X{traceld}

MDC(Mapped Diagnostic Context, 映射调试上下文)是 log4j 和 logback 提供的一种方便在多线条件下记录日志的功能。MDC 可以看成是一个与当前线程绑定的Map，可以往其中添加键值对。MDC 中包含的内容可以被同一线程中执行的代码所访问。当前线程的子线程会继承其父线程中的 MDC 的内容。当需要记录日志时，只需要从 MDC 中获取所需的信息即可。MDC 的内容则由程序在适当的时候保存进去。对于一个 Web 应用来说，通常是在请求被处理的最开始保存这些数据。

三、方案实现

由于MDC内部使用的是ThreadLocal所以只有本线程才有效，子线程和下游的服务MDC里的值会丢失所以方案主要的难点是解决值的传递问题。

1.logback配置文件模板格式添加标识%X

```
<!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%ms：日志消息，%n是换行符-->
<property name="log.pattern" value="%d{yyyy-MM-dd HH:mm:ss.SSS} [%X{traceld}] [%thread] %-5level %logger{20} - [%method,%line] - %msg%n" />
```

2.添加AOP maven依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

3.自定义日志注解

```
import java.lang.annotation.*;

/**
 * @Description: 自定义日志注解
 */
//什么时候使用该注解，我们定义为运行时
@Retention(RetentionPolicy.RUNTIME)
//注解用于什么地方，我们定义为作用于方法上
@Target({ElementType.METHOD})
//注解是否将包含在 JavaDoc 中
```

```

@Documented
public @interface WebLog {

    /**
     * 日志描述信息
     *
     * @return
     */
    String description() default "";
}

```

4.配置AOP切面

在配置 AOP 切面之前，我们需要了解下 [aspectj](#) 相关注解的作用：

- **@Aspect**: 声明该类为一个注解类；
- **@Pointcut**: 定义一个切点，后面跟随一个表达式，表达式可以定义为切某个注解，也可以切某个 package 下的方法；

切点定义好后，就是围绕这个切点做文章了：

- **@Before**: 在切点之前，织入相关代码；
- **@After**: 在切点之后，织入相关代码；
- **@AfterReturning**: 在切点返回内容后，织入相关代码，一般用于对返回值做些加工处理的场景；
- **@AfterThrowing**: 用来处理当织入的代码抛出异常后的逻辑处理；
- **@Around**: 环绕，可以在切入点前后织入代码，并且可以自由的控制何时执行切点；

```

import com.alibaba.dubbo.rpc.RpcContext;
import com.google.gson.Gson;
import com.common.annotation.WebLog;
import com.common.constant.Constants;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.*;
import org.slf4j.MDC;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;
import javax.servlet.http.HttpServletRequest;
import java.lang.reflect.Method;
import java.util.UUID;

```

```

@Component
@Aspect
@Order(1)
@Slf4j
public class WebLogAspect {

```

```

    /**

```

```

* 换行符
*/
private static final String LINE_SEPARATOR = System.lineSeparator();

/**
 * 以自定义 @WebLog 注解为切点
 */
@Pointcut("@annotation(com.pet.common.annotation.WebLog)")
public void WebLogAspect() {

}

/**
 * 在切点之前织入
 *
 * @param joinPoint
 * @throws Throwable
 */
@Before("WebLogAspect()")
public void doBefore(JoinPoint joinPoint) throws Throwable {
    // 开始打印请求日志
    ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.g
tRequestAttributes();
    HttpServletRequest request = attributes.getRequest();

    // 获取 @WebLog 注解的描述信息
    String methodDescription = getAspectLogDescription(joinPoint);

    String traceId = String.valueOf(UUID.randomUUID());
    MDC.put(Constants.LOG_TRACE_ID, traceId);
    RpcContext.getContext().setAttachment(Constants.LOG_TRACE_ID, traceId);
    // 打印请求相关参数
    log.info("===== Start =====
=====");
    // 打印请求 url
    log.info("URL      : {}", request.getRequestURL().toString());
    // 打印描述信息
    log.info("Description  : {}", methodDescription);
    // 打印 Http method
    log.info("HTTP Method  : {}", request.getMethod());
    // 打印调用 controller 的全路径以及执行方法
    log.info("Class Method  : {}.{}", joinPoint.getSignature().getDeclaringTypeName(), joinPoin
tSignature().getName());
    // 打印请求的 IP
    log.info("IP        : {}", request.getRemoteAddr());
    // 打印请求入参
    log.info("Request Args  : {}", new Gson().toJson(joinPoint.getArgs()));
}

/**
 * 在切点之后织入
 *
 * @throws Throwable
 */

```

```

@After("WebLogAspect()")
public void doAfter() throws Throwable {
    // 接口结束后换行, 方便分割查看
    log.info("=====  

===== End =====  

===== " + LINE_SEPARATOR);
    MDC.clear();
}

/**
 * 环绕
 *
 * @param proceedingJoinPoint
 * @return
 * @throws Throwable
 */
@Around("WebLogAspect()")
public Object doAround(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
    long startTime = System.currentTimeMillis();
    Object result = null;
    try {
        result = proceedingJoinPoint.proceed();
    } catch (Exception e) {
        log.info("exception :{}", e.getMessage());
        throw e;
    } finally {
        // 打印出参
        log.info("Response Args : {}", new Gson().toJson(result));
        // 执行耗时
        log.info("Time-Consuming : {} ms", System.currentTimeMillis() - startTime);
    }
    return result;
}

/**
 * 获取切面注解的描述
 *
 * @param joinPoint 切点
 * @return 描述信息
 * @throws Exception
 */
public String getAspectLogDescription(JoinPoint joinPoint) throws Exception {
    String targetName = joinPoint.getTarget().getClass().getName();
    String methodName = joinPoint.getSignature().getName();
    Object[] arguments = joinPoint.getArgs();
    Class targetClass = Class.forName(targetName);
    Method[] methods = targetClass.getMethods();
    StringBuilder description = new StringBuilder("");
    for (Method method : methods) {
        if (method.getName().equals(methodName)) {
            Class[] clazzs = method.getParameterTypes();
            if (clazzs.length == arguments.length) {
                description.append(method.getAnnotation(WebLog.class).description());
                break;
            }
        }
    }
}

```

```

    }
}
return description.toString();
}

```

5. 下游dubbo服务创建DubboTraceFilter过滤器 服务者端 供扩展

资源文件夹下创建 META-INF/dubbo 文件夹 创建com.alibaba.dubbo.rpc.Filter 文件，并编辑文件内容

```

// xxx为你DubboTraceIdFilter文件所在的位置
dubboTraceIdFilter=com.xxx.DubboTraceIdFilter

import com.alibaba.dubbo.common.extension.Activate;
import com.alibaba.dubbo.rpc.*;
import org.slf4j.MDC;

/**
 * @Description: dubbo跟踪traceId
 */
@Activate(group = {com.alibaba.dubbo.common.Constants.CONSUMER, com.alibaba.dubbo.
ommon.Constants.PROVIDER})
public class DubboTraceIdFilter implements Filter {

    private static final String LOG_TRACE_ID = "traceId";

    @Override
    public Result invoke(Invoker<?> invoker, Invocation invocation) throws RpcException {

        RpcContext rpcContext = RpcContext.getContext();

        // before
        if (rpcContext.isProviderSide()) {
            // get traceId from dubbo consumer, and set traceId to MDC
            String traceId = rpcContext.getAttachment(LOG_TRACE_ID);
            MDC.put(LOG_TRACE_ID, traceId);
        }

        if (rpcContext.isConsumerSide()) {
            // get traceId from MDC, and set traceId to rpcContext
            String traceId = MDC.get(LOG_TRACE_ID);
            rpcContext.setAttachment(LOG_TRACE_ID, traceId);
        }

        Result result = invoker.invoke(invocation);

        // after
        if (rpcContext.isProviderSide()) {
            // clear traceId from MDC
            MDC.remove(LOG_TRACE_ID);
        }
    }
}

```

```
    return result;
}
```

四，如何使用

因为我们的切点是自定义注解 `@WebLog`, 所以我们仅仅需要在 Controller 控制器的每个接口方法添加 `@WebLog` 注解即可, 如果我们不想某个接口打印出入参日志, 不加注解就可以了

```
@GetMapping("/xxxTest")
@WebLog(description = "XXX接口")
public Response xxxTest(HttpServletRequest request) {
    return null;
}
```

五，打印效果

```
.WebLogAspect - [doBefore,52] - ===== Start =====
.WebLogAspect - [doBefore,54] - URL           : http://localhost:8081/xxxTest
.WebLogAspect - [doBefore,56] - Description  : XXX接口
.WebLogAspect - [doBefore,58] - HTTP Method  : GET
.WebLogAspect - [doBefore,60] - Class Method : com. .... .xxxTest
.WebLogAspect - [doBefore,62] - IP           : 0:0:0:0:0:0:1
.WebLogAspect - [doAround,87] - Response Args : {"code":0}
.WebLogAspect - [doAround,89] - Time-Consuming : 282 ms
.WebLogAspect - [doAfter,72] - ===== End =====
```

从上图中可以看到, 每个对于每个请求, 开始与结束一目了然, 并且打印了以下参数:

- **URL:** 请求接口地址;
- **Description:** 接口的中文说明信息;
- **HTTP Method:** 请求的方法, 是 **POST**, **GET**, 还是 **DELETE** 等;
- **Class Method:** 被请求的方法路径: **包名 + 方法名**;
- **IP:** 请求方的 IP 地址;
- **Request Args:** 请求入参, 以 JSON 格式输出;
- **Response Args:** 响应出参, 以 JSON 格式输出;
- **Time-Consuming:** 请求耗时, 以此估算每个接口的性能指数;