# 日志追踪

本章描述如何借助于开源框架[TLog](https://tlog.yomahub.com/)，在全链路日志中打印RequestId。进而可以根据RequestId，串联起单个模块中同一请求的全部日志，或者在不同模块之间对同一请求进行日志追踪。项目集成Tlog后日志打印效果如下：

2023-11-17 10:21:47,249 INFO [http-nio-8080-exec-2] (TestController.java:21) <0><XTQg877497500041162752> main thread

2023-11-17 10:21:47,250 INFO [Thread-10] (TestController.java:23) <0><XTQg877497500041162752> sub thread

2023-11-17 10:21:47,250 INFO [common-task-worker-1] (TestController.java:26) <0><XTQg877497500041162752> pool thread

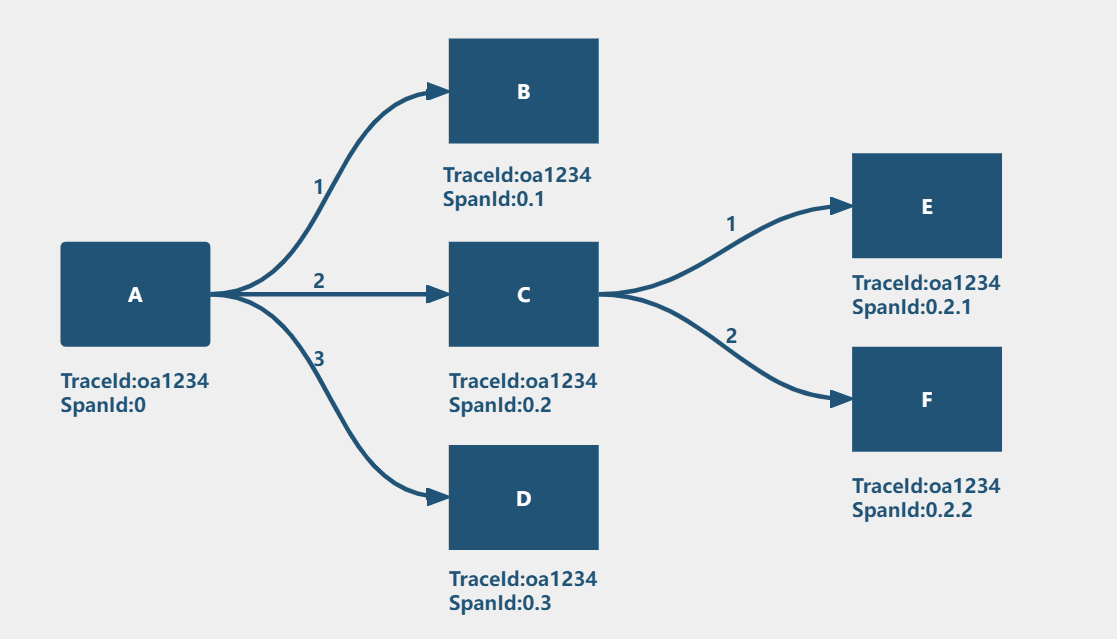
## 1.1. TLog概述

[TLog](https://tlog.yomahub.com/)是一个轻量级的分布式日志标记追踪框架。它不收集日志，也不需要额外的存储空间，它只是自动的对日志进行打标签，自动生成RequestId贯穿微服务的一整条链路。并且提供上下游节点信息。

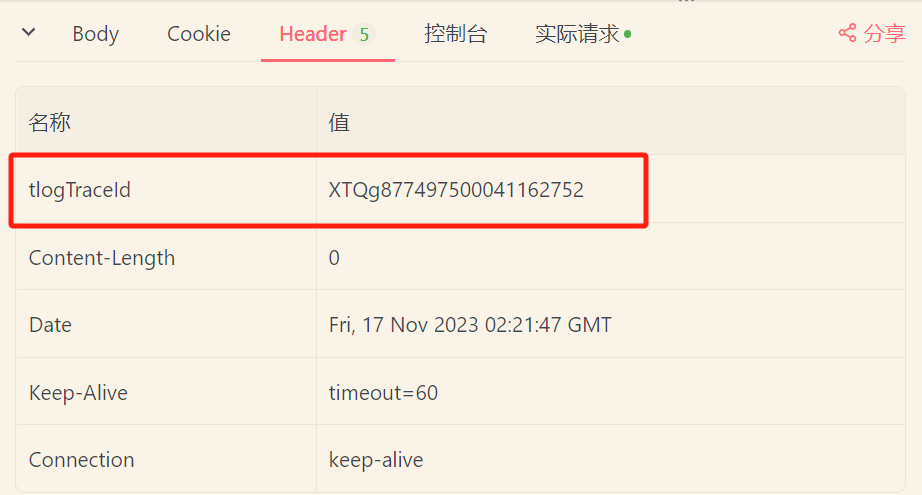
Tlog可实现对单模块接口调用、模块间Feign调用、XXL-JOB任务框架（包括异步任务和定时任务）、以及异步线程（包括线程池，多级异步线程等场景）的日志追踪。基于以上四条，该框架已自动覆盖项目内链路中主要节点，完成全链路日志追踪。

此外，它还具有如下特性：

* 对业务代码无侵入，引入依赖、增加配置即可。
* 几乎无性能损耗，快速稳定，经过压测，损耗在0.01%。
* 支持SpanId，可以打印出链路调用层次，SpanId生成逻辑如下图所示：



* 接口请求结果的header里，携带requestId，如下图所示。



* 可以在日志中打印上游节点信息，包括上游微服务的节点名称、Host Name以及IP地址。
* 通过在方法上增加特定注解，可以在日志中打印部分入参，如资源id、名称等。同时该标签也可以在整条链路中传递。
* 支持通过设置，在日志中打印出参数信息和调用时间信息，效果如下：

2020-12-01 19:20:07.768 [DubboServerHandler-127.0.0.1:30900-thread-2] INFO c.y.tlog.dubbo.filter.TLogDubboInvokeTimeFilter - <0.1><7592057736843136> [TLOG]开始调用接口[DemoService]的方法[sayHello],参数为:["jack"]

2020-12-01 19:20:07.795 [DubboServerHandler-127.0.0.1:30900-thread-2] INFO c.y.tlog.dubbo.filter.TLogDubboInvokeTimeFilter - <0.1><7592057736843136> [TLOG]结束接口[DemoService]中方法[sayHello]的调用,耗时为:90毫秒

关于Tlog的更详细的介绍，请移步官网：https://tlog.yomahub.com

## TLog集成

### 1.2.1. 添加依赖

这里可选择全量依赖，示例如下：

<dependency>  
   <groupId>com.yomahub</groupId>  
   <artifactId>tlog-all-spring-boot-starter</artifactId>  
   <version>1.5.1</version>  
</dependency>

如果不想依赖不必要的包，也可选择按需依赖，示例如下：

<dependency>  
   <groupId>com.yomahub</groupId>  
   <artifactId> tlog-XXX-spring-boot-starter</artifactId>  
   <version>1.5.1</version>  
</dependency>

具体模块和描述如下表:

|  |  |
| --- | --- |
| 模块名 | 描述 |
| tlog-feign-spring-boot-starter | 适用于spring cloud中open feign的项目 |
| tlog-gateway-spring-boot-starter | 适用于spring cloud中的gateway网关服务 |
| tlog-web-spring-boot-starter | 适用于有spring web的项目 |

### 1.2.2. 更新Logback配置文件

将encoder的class设置为AspectLogbackEncoder

<?xml version="1.0" encoding="UTF-8"?>  
<configuration debug="false">  
   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  
      **<!--这里替换成AspectLogbackEncoder-->  
 <encoder class="com.yomahub.tlog.core.enhance.logback.AspectLogbackEncoder">  
  <pattern>%d [%t] %-5level - %msg%n</pattern>  
 </encoder>**   </appender>​  
   <root level="INFO">  
       <appender-ref ref="STDOUT" />  
   </root>  
</configuration>

### 1.2.3. 添加AOP，处理XXL-JOB定时任务

如果不需要在定时任务中打印RequestId，可跳过此步骤，代码如下：

@Aspect  
public class TLogXxlJobAop {  
​  
   private final TLogRPCHandler tLogRPCHandler = new TLogRPCHandler();  
​  
   @Pointcut("execution(\* com.xxl.job.core.handler.IJobHandler+.\*(..))")  
   public void cut() {  
  }  
​  
   @Around("cut()")  
   public Object around(ProceedingJoinPoint jp) throws Throwable {  
       try{  
           tLogRPCHandler.processProviderSide(new TLogLabelBean());  
           return jp.proceed();  
      }finally {  
           tLogRPCHandler.cleanThreadLocal();  
      }  
  }  
}

### 1.2.4. 集成后日志打印效果

集成后日志打印效果如下：

2023-11-17 10:21:47,249 INFO [http-nio-8080-exec-2] (TestController.java:21)

<0><XTQg877497500041162752> main thread

2023-11-17 10:21:47,250 INFO [Thread-10] (TestController.java:23) <0><XTQg877497500041162752> sub thread

2023-11-17 10:21:47,250 INFO [common-task-worker-1] (TestController.java:26) <0><XTQg877497500041162752> pool thread

## TLog原理

### 1.3.1. TLog是怎样将RequestId输出到日志中的？

CMP中当前使用的日志框架是Logback，该框架支持使用方实现其接口，对Logback进行定制化扩展。TLog利用了Logback这个能力，其核心代码如下：

public class AspectLogbackConverter extends ClassicConverter {  
   @Override  
   public String convert(ILoggingEvent event) {  
       //只有在MDC没有设置的情况下才加到message里  
       if (!TLogContext.hasTLogMDC()) {  
           **String logValue = AspectLogContext.getLogValue();**           if (StringUtils.isBlank(logValue)) {  
               return event.getFormattedMessage();  
          } else {  
               **return logValue + " " + event.getFormattedMessage();**  
          }  
      } else {  
           return event.getFormattedMessage();  
      }  
  }  
}

当项目集成TLog后，每次调用logger.info()等方法打印日志前，都会先执行上述代码中的convert方法。该方法会从TLog定义的上下文AspectLogContext中取出包含RequestId等信息的logValue，然后与原本的日志信息拼接返回。

这样就实现了日志中打印RequestId等信息的功能，所以接下来的问题是，针对不同的场景，TLog是如何正确的设置RequestId到AspectLogContext上下文中的。

### 1.3.2. 接口调用时，TLog如何设置RequestId？

TLog实现了Spring的Web拦截器HandlerInterceptor接口，在该实现类中，如果发现上游请求Header中没有携带RequestId，则重新生成一个，并设置到AspectLogContext上下文中。实现该功能的核心代码如下：

public class TLogRPCHandler {  
​  
   protected static final Logger log = LoggerFactory.getLogger(TLogRPCHandler.class);  
​  
   public void processProviderSide(TLogLabelBean labelBean) {

……  
       //如果没有获取到，则重新生成一个traceId  
       if (StringUtils.isBlank(labelBean.getTraceId())) {  
           **labelBean.setTraceId(TLogIdGeneratorLoader.getIdGenerator().withExtData(labelBean.getExtData()).generateTraceId());**  
           log.debug("[TLOG]可能上一个节点[{}]没有正确传递traceId,重新生成traceId[{}]", labelBean.getPreIvkApp(), labelBean.getTraceId());  
      }  
         
       //生成日志标签  
       String tlogLabel = TLogLabelGenerator.generateTLogLabel(labelBean.getPreIvkApp(),  
               labelBean.getPreIvkHost(),  
               labelBean.getPreIp(),  
               TLogContext.getCurrIp(),  
               labelBean.getTraceId(),  
               TLogContext.getSpanId());  
​  
       //往日志切面器里放一个日志前缀  
      **AspectLogContext.putLogValue(tlogLabel);**

……  
  }  
     
}

### 1.3.3. Feign调用时，TLog如何设置RequestId？

TLog实现了Feign的RequestInterceptor接口，在该实现类中TLog将RequestId等信息塞到请求Header里，发送给下游服务，代码如下：

public class TLogFeignFilter implements RequestInterceptor {  
​  
   private static final Logger log = LoggerFactory.getLogger(TLogFeignFilter.class);  
​  
   @Value("${spring.application.name}")  
   private String appName;  
​  
   @Override  
   public void apply(RequestTemplate requestTemplate) {  
**String traceId = TLogContext.getTraceId();**  
​  
       if(StringUtils.isNotBlank(traceId)){  
**requestTemplate.header(TLogConstants.TLOG\_TRACE\_KEY, traceId);**  
           requestTemplate.header(TLogConstants.TLOG\_SPANID\_KEY, SpanIdGenerator.generateNextSpanId());  
           requestTemplate.header(TLogConstants.PRE\_IVK\_APP\_KEY, appName);  
           requestTemplate.header(TLogConstants.PRE\_IVK\_APP\_HOST, LocalhostUtil.getHostName());  
           requestTemplate.header(TLogConstants.PRE\_IP\_KEY, LocalhostUtil.getHostIp());  
      }else{  
           log.debug("[TLOG]本地threadLocal变量没有正确传递traceId,本次调用不传递traceId");  
      }  
  }  
}

Feign下游服务，从Header中取出RequestId等信息，设置到TLog的AspectLogContext上下文中，这一步骤和接口调用时设置方式相同。

### 1.3.4. 在XXL-JOB定时任务中，TLog如何设置RequestId？

TLog使用AOP，拦截所有的XXL-JOB定时任务，并在该AOP执行定时任务代码前，将RequestId设置到AspectLogContext上下文。AOP代码如下：

@Aspect  
public class TLogXxlJobAop {  
​  
   private final TLogRPCHandler tLogRPCHandler = new TLogRPCHandler();  
​  
  **@Pointcut("execution(\* com.xxl.job.core.handler.IJobHandler+.\*(..))")**  
   public void cut() {  
  }  
​  
   @Around("cut()")  
   public Object around(ProceedingJoinPoint jp) throws Throwable {  
       try{  
          **tLogRPCHandler.processProviderSide(new TLogLabelBean());**  
           return jp.proceed();  
      }finally {  
           tLogRPCHandler.cleanThreadLocal();  
      }  
  }  
}

因为该版本TLog适配的XXL-JOB，与项目当前集成的版本不一致，导致TLog在XXL-JOB定时任务中无法正常工作。所以在章节1.2中集成TLog时，才需要额外配置该AOP。

### 1.3.5. 在异步线程中，TLog如何设置RequestId？

JDK提供的ThreadLocal可以在线程内传递数据，InheritableThreadLocal可以在线程之间传递数据，但是JDK并未提供可以在线程池复用线程之间正确传递数据的能力。

因此，TLog选择借助于阿里开源的TransmittableThreadLocal来实现多级异步线程、特别是线程池内复用的线程之间的RequestId等信息的传递，它在前面一直提到的AspectLogContext上下文中发挥作用，代码如下：

public class AspectLogContext {

**private static TransmittableThreadLocal<String> logValueTL = new TransmittableThreadLocal<>();**

public static void putLogValue(String logValue) {

logValueTL.set(logValue);

}

public static string getLogValue() {

return logValueTL.get();

}

public static void remove() {

logValueTL.remove ();

}

}

所以，针对多级异步线程、线程池复用线程场景，TLog不需要再额外做其他处理，就已经可以实现在日志中正确打印RequestId了。