本方案描述如何在日志中打印RequestId，进而根据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进行链路追踪。

# 一、自研方案

本方案以创建云主机为例，说明方案的具体实施过程。

1. 由于UserSession会从网关开始，传递到链路中的各个节点，包括异步任务中。因此，本方案选择使用UserSession在链路中传递RequestId。UserSession新增的代码片段如下：

public class UserSession implements Serializable {  
  /\*\*  
    \* 请求ID，用来在日志中打印，进行同一请求的链路追踪  
    \*/  
   private String requestId;  
​  
   public String getRequestId() {  
       return requestId;  
  }  
​  
   public void setRequestId(String requestId) {  
       this.requestId = requestId;  
  }  
}

1. 创建云主机的请求，首先打到网关，由网关生成RequestId，并存储到UserSession。
2. 请求接着到达ICompute模块，由ICompute中拦截所有Controller的AOP，将UserSession中的RequestId设置为Logback变量。AOP示例代码如下：

@Slf4j  
@Aspect  
@Component  
public class RequestIdAdvice {  
​  
   @Pointcut("within(com.inspur.incloud.icompute.openstack.controller..\*)")  
   private void allController() {}  
​  
   @Around("allController()")  
   public Object doAround(ProceedingJoinPoint joinPoint) throws Throwable {  
       try {  
           // 1.将requestId设置为logback变量  
           MDC.put("REQUEST\_ID",SessionCheckUtil.getSession().getRequestId());  
​  
           // 2.业务逻辑处理  
           return joinPoint.proceed();  
      } finally {  
           // 3.清理logback变量  
           MDC.clear();  
      }  
  }  
}

1. 更新Logback配置文件，取出步骤3中设置的RequestId，作为日志的一部分。以下XML配置示例中的%X{REQUEST\_ID}即是取出Logback变量值的语法。

<?xml version="1.0" encoding="UTF-8"?>  
<configuration>    
 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  
 <encoder>  
 <pattern>%d %p **%X{REQUEST\_ID}** (%file:%line\)- %m%n</pattern>  
 <charset>UTF-8</charset>  
 </encoder>  
 </appender>  
 <root level="INFO">  
 <appender-ref ref="STDOUT" />  
 </root>  
</configuration>

1. ICompute继续添加ITask任务，异步创建云主机。由于Logback是绑定到线程的，因此步骤3中设置的RequestId，在异步任务中取不到。

为了解决这个问题，需要对ITask中AsynServiceImpl类addWork方法进行改造，在异步任务中，再次将UserSession中的RequstId再次设置为Logback变量。

1. 在创建云主机的异步任务中，ICompute会调用INetwork、ICinder等模块的接口。这些模块也依照步骤3、4进行配置，从ICompute传递过来的UserSession中取出RequestId，打印到自己的模块日志中。
2. 完成上述步骤后，ICompute及请求链路上的其他模块的日志，就会带上RequestId，控制台输出示例如下：

2023-10-31 20:14:30,481 ERROR 8a6f34e58b7432ce018b752420ca0026 (OpenstackVmCreateServiceImpl.java:363)- vm name :zhvm02 existed

1. 更新项目内公共返回Model，添加RequestId并在构造器中赋值，返回给前端或经由网关的接口调用。

public class OperationResult<T> implements java.io.Serializable {  
     
   private String requestId;  
​  
   public String getRequestId() {  
       return requestId;  
  }  
​  
   public OperationResult() {  
       this.requestId = SessionCheckUtil.getSession().getRequestId();  
  }  
}

返回结果示例：

{  
   "flag": true,  
   "errCode": null,  
   "errMessage": null,  
   "exceptionMsg": null,  
   "resData": { },  
   "requestId": "8a6f34e58b7432ce018b752420ca0026"  
}

# 二、开源方案

## 2.1. 概述特性

在Gitee上，存在一个轻量级的分布式日志标记追踪框架[TLog](https://tlog.yomahub.com/)，可以完美匹配项目对RequestId的需求。它具有如下特性：

* 可实现全链路日志打印RequestId。
* 对业务代码无侵入式设计，使用简单，10分钟即可接入。
* 支持Logback日志框架（当前项目日志框架），并提供自动检测，完成适配。
* 默认支持对异步线程的追踪，包括线程池，多级异步线程等场景。
* 默认支持对XXL-JOB任务框架的追踪，包括异步任务、定时任务。
* 支持SpanId，可以打印出链路调用层次。如初始调用是0，下一级调用是0.1。
* 几乎无性能损耗，快速稳定，经过压测，损耗在0.01%。
* 支持feigin调用---

## 2.2. 集成步骤

1. 添加依赖

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

1. 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>