

基于SLF4J MDC机制实现日志的链路追踪



xiaolyuh

关注



2

2018.06.03 15:09:26 字数 815 阅读 6,361

问题描述

最近经常做线上问题的排查，而排查问题用得最多的方式是查看日志，但是在现有系统中，各种无关日志穿行其中，导致我没办法快速的找出用户在一次请求中所有的日志。

问题分析

我们没办法快速定位用户在一次请求中对应的所有日志，或者说是定位某个用户操作的所有日志，那是因为在输出的日志的时候没把请求的唯一标示或者说是用户身份标示输出到我们的日志中，导致我们没办法根据一个请求或者用户身份标示来做日志的过滤。所以我们在记录日志的是后把请求的唯一标示（sessionId）或者身份标示（userId）记录到日志中这个问题就可以得到很好的解决了。

解决方案

1. 在每次请求的时候，获取到请求的sessionId（或者自己生成一个伪sessionId），并在每次输出log的时候将这个sessionId输出到日志中。这个方式实现简单，代码侵入型强，每次输出都会多输出一个sessionId参数，工作量大，但是可控粒度高。
2. 我们使用Logback的MDC机制，日志模板中加入sessionId格式。在日志输出格式中指定输出sessionId。如：

```
1 | %d{yyyy-MM-dd HH:mm:ss.SSS} [%X{sessionId}] -%5p ${PID:-} [%15.15t] %-40.40logger{39} :
```

这种方式工作量小，代码侵入小，易扩展，但是可控粒度低。

方案说明

第一种方案很简单，也很容易实现，就是在输出日志的时候多输出一个参数，如：

```
1 | logger.info("sessionId: {}", message: {}, sessionId, "日志信息");
```

我们这里主要说一下第二种方式的实现。

实现思路，这里以Spring MVC为例：

1. 新建一个日志拦截器，在拦截所有请求，在处理请求前将sessionId放到MDC中，在处理完请求后清除MDC的内容。这里就解决了80%的问题。
2. 在原来版本中新起线程时MDC会自动将父线程的MDC内容复制给子线程，因为MDC内部使用的是InheritableThreadLocal，但是因为性能问题在最新的版本中被取消了，所以子线程不会自动获取到父线程的MDC内容。官方建议我们在父线程新建子线程之前调用MDC.getCopyOfContextMap()方法将MDC内容取出来传给子线程，子线程在执行操作



xiaolyuh

关注

总资产261 (约24.28元)

Mybatis 源码（五）Mybatis 中的数据读写

阅读 100

深入理解JVM - 内存溢出实战

阅读 6

深入理解JVM - JVM内存模型

阅读 27

推荐阅读

Spring Cloud gateway 七 Sentinel 注解方式使用

阅读 401

蚂蚁二面，面试官问我零拷贝的实现原理，当场懵了...

阅读 36,220

IDEA插件神器之Grep Console

阅读 4,430

Spring WebFlux运用中的思考与对比

阅读 50

微服务拆分与设计

阅读 13

```
1 | %d{yyyy-MM-dd HH:mm:ss.SSS} [%X{sessionId}] -%5p ${PID:-} [%15.15t] %-40.40logger
```

拦截器 LogInterceptor

```
1 package com.xiaolyuh.interceptors;
2
3 import org.slf4j.MDC;
4 import org.springframework.web.servlet.ModelAndView;
5 import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
6
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9 import java.util.UUID;
10
11 /**
12  * 日志拦截器组件，在输出日志中加上sessionId
13  *
14  * @author yuhao.wang3
15  */
16 public class LogInterceptor extends HandlerInterceptorAdapter {
17     /**
18      * 会话ID
19      */
20     private final static String SESSION_KEY = "sessionId";
21
22     @Override
23     public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2,
24                               throws Exception {
25
26         // 删除sessionId
27         MDC.remove(SESSION_KEY);
28     }
29
30     @Override
31     public void postHandle(HttpServletRequest arg0, HttpServletResponse arg1,
32                           Object arg2, ModelAndView arg3) throws Exception {
33     }
34
35     @Override
36     public boolean preHandle(HttpServletRequest request,
37                              HttpServletResponse response, Object handler) throws Exception {
38
39         // 设置sessionId
40         String token = UUID.randomUUID().toString().replace("-", "");
41         MDC.put(SESSION_KEY, token);
42         return true;
43     }
44 }
```

注册拦截器

```
1 /**
2  * WEB MVC配置类
3  *
4  * @author yuhao.wang3
5  */
6 @Configuration
7 public class WebMvcConfigurer extends WebMvcConfigurerAdapter {
8
9     /**
10      * 把我们的拦截器注入为bean
11      *
12      * @return
13      */
14     @Bean
15     public HandlerInterceptor logInterceptor() {
16         return new LogInterceptor();
17     }
18 }
```

写下你的评论...

评论2

赞27

...

```
23  */
24  @Override
25  public void addInterceptors(InterceptorRegistry registry) {
26      // addPathPatterns 用于添加拦截规则，这里假设拦截 /url 后面的全部链接
27      // excludePathPatterns 用户排除拦截
28      registry.addInterceptor(logInterceptor()).addPathPatterns("/**");
29      super.addInterceptors(registry);
30  }
31 }
```

扩展ThreadPoolTaskExecutor线程池

扩展ThreadPoolTaskExecutor线程池的主要目的是实现将父线程的MDC内容复制给子线程。

```
1 package com.xiaolyuh.utils;
2
3 import org.slf4j.MDC;
4 import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;
5
6 import java.util.Map;
7
8 /**
9  * 这是{@link ThreadPoolTaskExecutor}的一个简单替换，可以在每个任务之前设置子线程的MDC数据。
10  * <p/>
11  * 在记录日志的时候，一般情况下我们会使用MDC来存储每个线程的特有参数，如身份信息等，以便更好的查询日志。
12  * 但是Logback在最新的版本中因为性能问题，不会自动的将MDC的内存传给子线程。所以Logback建议在执行异步线
13  * 先通过MDC.getCopyOfContextMap()方法将MDC内存获取出来，再传给线程。
14  * 并在子线程的运行的最开始调用MDC.setContextMap(context)方法将父线程的MDC内容传给子线程。
15  * <p>
16  * https://logback.qos.ch/manual/mdc.html
17  *
18  * @author yuhao.wang3
19  */
20 public class MdcThreadPoolTaskExecutor extends ThreadPoolTaskExecutor {
21
22     /**
23      * 所有线程都会委托给这个execute方法，在这个方法中我们把父线程的MDC内容赋值给子线程
24      * https://logback.qos.ch/manual/mdc.html#managedThreads
25      *
26      * @param runnable
27      */
28     @Override
29     public void execute(Runnable runnable) {
30         // 获取父线程MDC中的内容，必须在run方法之前，否则等异步线程执行的时候有可能MDC里面的值已经被清
31         Map<String, String> context = MDC.getCopyOfContextMap();
32         super.execute(() -> run(runnable, context));
33     }
34
35     /**
36      * 子线程委托的执行方法
37      *
38      * @param runnable {@link Runnable}
39      * @param context 父线程MDC内容
40      */
41     private void run(Runnable runnable, Map<String, String> context) {
42         // 将父线程的MDC内容传给子线程
43         MDC.setContextMap(context);
44         try {
45             // 执行异步操作
46             runnable.run();
47         } finally {
48             // 清空MDC内容
49             MDC.clear();
50         }
51     }
52 }
```

扩展Hystrix

扩展Hystrix线程池隔离策略日志链路追踪

写下你的评论...

评论2

赞27

...

```

3
4  * @author yuhao.wang3
5  */
6  public class MdcHystrixConcurrencyStrategy extends HystrixConcurrencyStrategy {
7
8      @Override
9      public <T> Callable<T> wrapCallable(Callable<T> callable) {
10         return new MdcAwareCallable(callable, MDC.getCopyOfContextMap());
11     }
12
13     private class MdcAwareCallable<T> implements Callable<T> {
14
15         private final Callable<T> delegate;
16
17         private final Map<String, String> contextMap;
18
19         public MdcAwareCallable(Callable<T> callable, Map<String, String> contextMap) {
20             this.delegate = callable;
21             this.contextMap = contextMap != null ? contextMap : new HashMap();
22         }
23
24         @Override
25         public T call() throws Exception {
26             try {
27                 MDC.setContextMap(contextMap);
28                 return delegate.call();
29             } finally {
30                 MDC.clear();
31             }
32         }
33     }
34 }

```

配置Hystrix

```

1  @Configuration
2  public class HystrixConfig {
3
4      //用来拦截处理HystrixCommand注解
5      @Bean
6      public HystrixCommandAspect hystrixAspect() {
7          return new HystrixCommandAspect();
8      }
9
10     @PostConstruct
11     public void init() {
12         HystrixPlugins.getInstance().registerConcurrencyStrategy(new MdcHystrixConcurrencyStrategy());
13     }
14
15 }

```

Logback配置

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <configuration>
3      <property name="CONSOLE_LOG_PATTERN"
4          value="${CONSOLE_LOG_PATTERN:-%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} [%X{traceId}] - %m%n}" />
5      <property name="FILE_LOG_PATTERN"
6          value="${FILE_LOG_PATTERN:-%d{yyyy-MM-dd HH:mm:ss.SSS} [%X{sessionId}] [%X{traceId}] - %m%n}" />
7      <include resource="org/springframework/boot/logging/logback/base.xml"/>
8
9      <appender name="ASYNC_FILE" class="ch.qos.logback.classic.AsyncAppender">
10         <!-- 不丢失日志。默认的，如果队列的80%已满，则会丢弃TRACT、DEBUG、INFO级别的日志 -->
11         <discardingThreshold>0</discardingThreshold>
12         <!-- 更改默认的队列的深度，该值会影响性能。默认值为256 -->
13         <queueSize>2048</queueSize>
14         <includeCallerData>true</includeCallerData>
15         <!-- 添加附加的appender，最多只能添加一个 -->
16         <appender-ref ref="FILE"/>
17     </appender>
18

```

写下你的评论...

评论2

赞27

...

```
1 @Service
2 public class LogService {
3     Logger logger = LoggerFactory.getLogger(LogService.class);
4
5     public void log() {
6         logger.debug("=====");
7         ThreadTaskUtils.run(() -> run());
8         FutureTask<String> futureTask = new FutureTask<String>(() -> call());
9         ThreadTaskUtils.run(futureTask);
10        try {
11            logger.debug("=====: {}", futureTask.get());
12        } catch (Exception e) {
13            logger.error(e.getMessage(), e);
14        }
15        logger.debug("=====");
16    }
17
18    private String call() {
19        logger.debug("1111111111");
20        return "3333";
21    }
22
23    public void run() {
24        logger.debug("2222222222222");
25    }
26 }
```

日志输出示例

```
1 2018-06-02 23:11:02.376 [a114db8f4be942d891407d57ff74276d] DEBUG 12828 --- [http-nio-80-€
2 2018-06-02 23:11:02.376 [a114db8f4be942d891407d57ff74276d] DEBUG 12828 --- [MdcThreadPoo
3 2018-06-02 23:11:02.377 [a114db8f4be942d891407d57ff74276d] DEBUG 12828 --- [MdcThreadPoo
4 2018-06-02 23:11:02.377 [a114db8f4be942d891407d57ff74276d] DEBUG 12828 --- [http-nio-80-€
5 2018-06-02 23:11:02.377 [a114db8f4be942d891407d57ff74276d] DEBUG 12828 --- [http-nio-80-€
6 2018-06-02 23:11:02.536 [8657ed4f5267489aa323f9422974002b] DEBUG 12828 --- [http-nio-80-€
7 2018-06-02 23:11:02.536 [8657ed4f5267489aa323f9422974002b] DEBUG 12828 --- [MdcThreadPoo
8 2018-06-02 23:11:02.536 [8657ed4f5267489aa323f9422974002b] DEBUG 12828 --- [MdcThreadPoo
9 2018-06-02 23:11:02.536 [8657ed4f5267489aa323f9422974002b] DEBUG 12828 --- [http-nio-80-€
10 2018-06-02 23:11:02.536 [8657ed4f5267489aa323f9422974002b] DEBUG 12828 --- [http-nio-80-€
11 2018-06-02 23:11:02.728 [e85380fb1554463ca156318b0a3ff7c2] DEBUG 12828 --- [http-nio-80-€
12 2018-06-02 23:11:02.728 [e85380fb1554463ca156318b0a3ff7c2] DEBUG 12828 --- [MdcThreadPoo
13 2018-06-02 23:11:02.729 [e85380fb1554463ca156318b0a3ff7c2] DEBUG 12828 --- [MdcThreadPoo
14 2018-06-02 23:11:02.729 [e85380fb1554463ca156318b0a3ff7c2] DEBUG 12828 --- [http-nio-80-€
15 2018-06-02 23:11:02.729 [e85380fb1554463ca156318b0a3ff7c2] DEBUG 12828 --- [http-nio-80-€
```

源码

<https://github.com/wyh-spring-ecosystem-student/spring-boot-student/tree/releases>

spring-boot-student-log 工程

 27人点赞 >



 日志




"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下

写下你的评论...

 评论2  赞27 ...



写下你的评论...

全部评论 2

只看作者

按时间倒序

按时间正序




hylexus

3楼 2019.05.23 17:03

666

 赞


 回复




Nathans

2楼 2018.06.12 19:39

这个功能用于追踪某个请求的链路操作或者用户很方便

 赞

 回复

被以下专题收入，发现更多相似内容

-  程序员
-  Java 杂谈
-  基础资源

推荐阅读

更多精彩内容 >


Java面试宝典Beta5.0

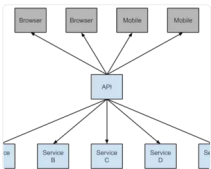
pdf下载地址：Java面试宝典 第一章内容介绍 20 第二章JavaSE基础 21 一、Java面向对象 21 ...

 王震阳 阅读 75,602 评论 25 赞 505

Spring Cloud

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智...

 卡卡罗2017 阅读 73,019 评论 12 赞 116



[翻译] logstash中logback的json编码器插件

原文地址为 <https://github.com/logstash/logstash-logback-encode...>

 飞来来 阅读 7,819 评论 1 赞 5


JMETER中文手册

JMETER中文手册 1. 简介 Apache JMeter是100%纯java桌面应用程序，被设计用来测试客户端...

 捉虫师 阅读 10,677 评论 0 赞 24

iOS 面试宝典 没有比这更全的了（持续更新）

1.iOS高性能编程 (1).内层 最小的内层平均值和峰值(2).耗电量 高效的算法和数据结构(3).初始化时...

 欧辰_OSR 阅读 12,211 评论 7 赞 161