# 实战篇，微服务日志的伤痛

## 本文内容

* 日志有什么用？
* 日志存在的痛点？
* 构建日志系统

## 日志有什么用？

* 系统出现故障的时候，可以通过日志信息快速定位问题，修复bug，恢复业务
* 提取有用数据，做数据分析使用

本文主要讨论通过日志来快速定位并解决问题。

## 日志存在的痛点

先介绍一下多数公司采用的方式：目前比较流行的是采用spring cloud（或者dubbo）做微服务，按照业务拆分为多个独立的服务，服务采用集群的方式部署在不同的机器上，当一个请求过来的时候，可能会调用到很多服务进行处理，spring cloud一般采用logback（或者log4j）输出日志到文件中。当系统出问题的时候，按照系统故障的严重程度，严重的会回退版本，然后排查bug，轻的，找运维去线上拉日志，然后排查问题。

这个过程中存在一些问题：

1. 日志文件太大太多，不方便查找
2. 日志分散在不同的机器上，也不方便查找
3. 一个请求可能会调用多个服务，完整的日志难以追踪
4. 系统出现了问题，只能等到用户发现了，自己才知道
5. 本文要解决上面的几个痛点，构建我们的日志系统，达到以下要求：
6. 方便追踪一个请求完整的日志
7. 方便快速检索日志
8. 系统出现问题自动报警，通知相关人员

## 构建日志系统

按照上面我们定的要求，一个个解决。

### 方便追踪一个请求完整的日志

当一个请求过来的时候，可能会调用多个服务，多个服务内部可能又会产生子线程处理业务，所以这里面有两个问题需要解决：

* 多个服务之间日志的追踪
* 服务内部子线程和主线程日志的追踪，这个地方举个例子，比如一个请求内部需要给10000人发送推送，内部开启10个线程并行处理，处理完毕之后响应操作者，这里面有父子线程，我们要能够找到这个里面所有的日志

需要追踪一个请求完整日志，我们需要给每个请求设置一个全局唯一编号，可以使用UUID或者其他方式也行。

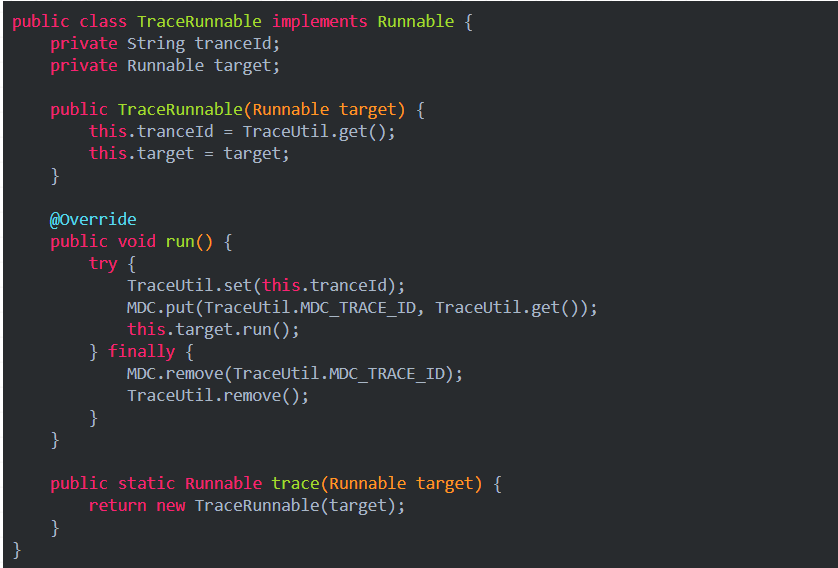
多个服务之间日志追踪的问题：当一个请求过来的时候，在入口处生成一个trace\_id，然后放在ThreadLocal中，如果内部设计到多个服务之间相互调用，调用其他服务的时，将trace\_id顺便携带过去。

父子线程日志追踪的问题：可以采用**InheritableThreadLocal**来存放trace\_id，这样可以在线程中获取到父线程中的trace\_id。

所以此处我们需要使用InheritableThreadLocal来存储trace\_id。

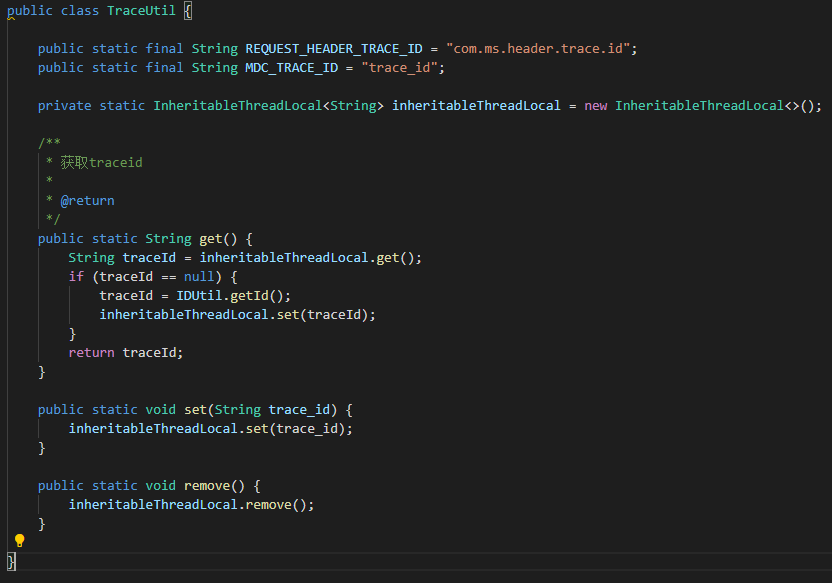
关于ThreadLocal和InheritableThreadLocal可以参考：[ThreadLocal、InheritableThreadLocal（通俗易懂）](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933186&idx=1&sn=079567e8799e43cb734b833c44055c01&chksm=88621b7cbf15926aace88777445822314d6eed2c1f5559b36cb6a6e181f0e543ee14d832ebc2&token=2027319240&lang=zh_CN&scene=21#wechat_redirect)

如果自己使用了线程池处理请求的，由于线程池中的线程采用的是复用的方式，所以需要对执行的任务Runable做一些改造，如代码：



需要用线程池执行的任务使用TraceRunnable封装一下就可以了。

TraceUtil代码：



日志输出中携带上trace\_id，这样最终我们就可以通过trace\_id找到一个请求的完整日志了。

### 方便快速检索日志

日志分散在不同的机器上，如果要快速检索，需要将所有服务产生的日志汇集到一个地方。

关于检索日志的，列一下需求：

1. 我们将收集日志发送到消息中间件中（可以是kafka、rocketmq），消息中间件这块不介绍，选择玩的比较溜的就可以了
2. 系统产生日志尽量不要影响接口的效率
3. 带宽有限的情况下，发送日志也尽量不要去影响业务
4. 日志尽量低延次，产生的日志，尽量在生成之后1分钟后可以检索到
5. 检索日志功能要能够快速响应

关于上面几点，我们需要做的：日志发送的地方进行改造，引入消息中间件，将日志异步发送到消息中间件中，查询的地方采用elasticsearch，日志系统需要订阅消息中间件中的日志，然后丢给elasticsearch建索引，方便快速检索，咱们来一点点的介绍。

**日志发送端的改造**

日志是有业务系统产生的，一个请求过来的时候会产生很多日志，日志产生时，我们尽量减少日志输出对业务耗时的影响，我们的过程如下：

* 业务系统内部引用一个线程池来异步处理日志，线程池内部可以使用一个容量稍微大一点的阻塞队列
* 业务系统将日志丢给线程池进行处理
* 线程池中将需要处理的日志先压缩一下，然后发送至mq

线程池的使用可以参考：[JAVA线程池，这一篇就够了](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933151&idx=1&sn=2020066b974b5f4c0823abd419e8adae&chksm=88621b21bf159237bdacfb47bd1a344f7123aabc25e3607e78d936dd554412edce5dd825003d&token=2027319240&lang=zh_CN&scene=21#wechat_redirect)

**引入mq存储日志**

业务系统将日志先发送到mq中，后面由其他消费者订阅进行消费。日志量比较大的，对mq的要求也比较高，可以选择kafka，业务量小的，也可以选取activemq。

**使用elasticsearch来检索日志**

elasticsearch（以下简称es）是一个全文检索工具，具体详情可以参考其官网相关文档。使用它来检索数据效率非常高。日志系统中需要我们开发一个消费端来拉取mq中的消息，将其存储到es中方便快速检索，关于这块有几点说一下：

建议按天在es中建立数据库，日志量非常大的，也可以按小时建立数据库。查询的时候，时间就是必选条件了，这样可以快速让es定位到日志库进行检索，提升检索效率

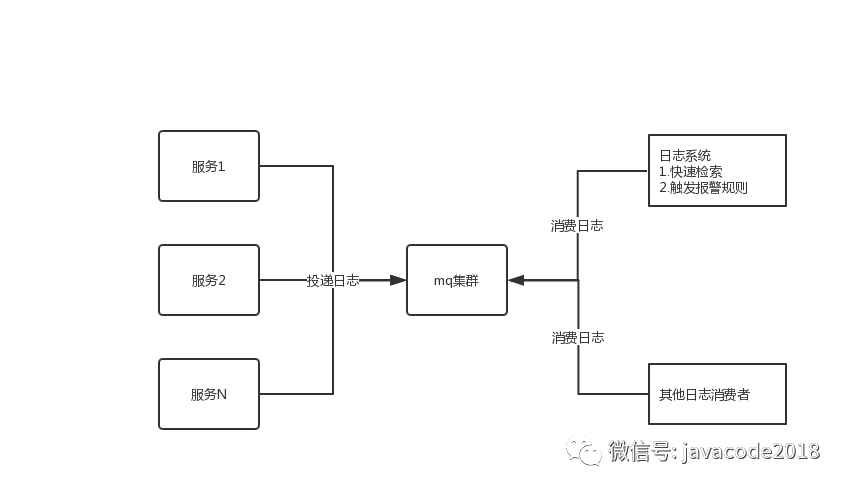
日志常见的需要收集的信息：trace\_id、时间、日志级别、类、方法、url、调用的接口开始时间、调用接口的结束时间、接口耗时、接口状态码、异常信息、日志信息等等，可以按照这些在es中建立索引，方便检索。

### 日志监控报警

日志监控报警是非常重要的，这个必须要有，日志系统中需要开发监控报警功能，这块我们可以做成通过页面配置的方式，支持报警规则的配置，如日志中产生了某些异常、接口响应时间大于多少、接口返回状态码404等异常信息的时候能够报警，具体的报警可以是语音电话、短信通知、钉钉机器人报警等等，这些也做成可以配置的。

日志监控模块从mq中拉取日志，然后去匹配我们启用的一些规则进行报警。

### 结构图如下



## 构建日志系统需要用到的知识点

1. [java中线程池的使用](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933151&idx=1&sn=2020066b974b5f4c0823abd419e8adae&chksm=88621b21bf159237bdacfb47bd1a344f7123aabc25e3607e78d936dd554412edce5dd825003d&token=2027319240&lang=zh_CN&scene=21#wechat_redirect)
2. [ThreadLocal、InheritableThreadLocal（通俗易懂）](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933186&idx=1&sn=079567e8799e43cb734b833c44055c01&chksm=88621b7cbf15926aace88777445822314d6eed2c1f5559b36cb6a6e181f0e543ee14d832ebc2&token=2027319240&lang=zh_CN&scene=21#wechat_redirect)
3. elasticsearch，可以参考其官方文档
4. mq