# 你是我的眼，带我领略微服务世界的浩瀚

## 前言

在微服务架构应用的建设过程中，我们可能会遇到各种各样的问题。当微服务规模较小，系统间交互较少时，查找定位问题相对容易；然而随着业务的增长，系统越来越繁杂，系统间的交互甚至可以说像城市间的交通网一样错综复杂。伴随而来的问题是系统排查成本增高，系统数据无法有效利用，无法有效支持其他决策系统，产生系统间的协同效应更是难以谈及。

我们需要考虑的是，如何有效的提取并利用系统产生的数据，对其进行分析，以便有效的为系统营销或辅助决策等类似系统提供有价值的数据，最终增强系统协同效应和产品竞争力。

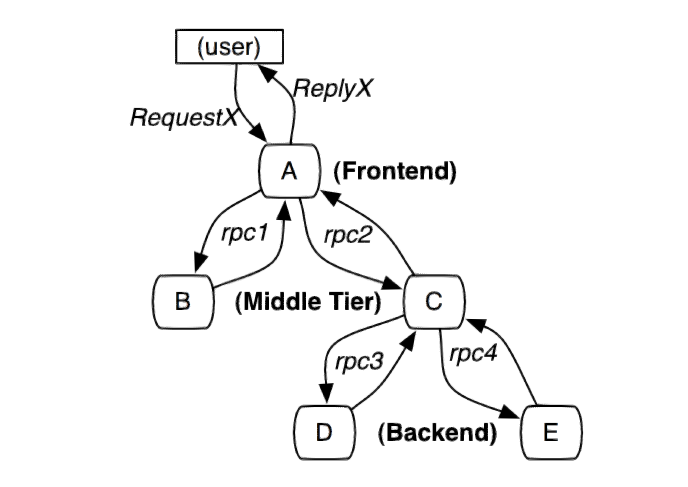
那么，我们如何在错综复杂的系统交互和大量的系统数据中，根据用户的特定行为画出交互时序图呢？当系统出现问题时，如何快速定位是系统bug还是业务流程问题呢？又如何快速找出影响系统性能的瓶颈呢？

利用分布式链路跟踪可以有效解决以上问题。链路跟踪为跨进程通信的事务提供了分析和监控机制，有助于分析bug原因及系统性能问题定位。

本文将以分布式链路跟踪技术为基础，深入剖析用友云技术中的微服务链路跟踪技术，带您领略不一样的日志收集和预警平台。

## 链路跟踪的理论基础

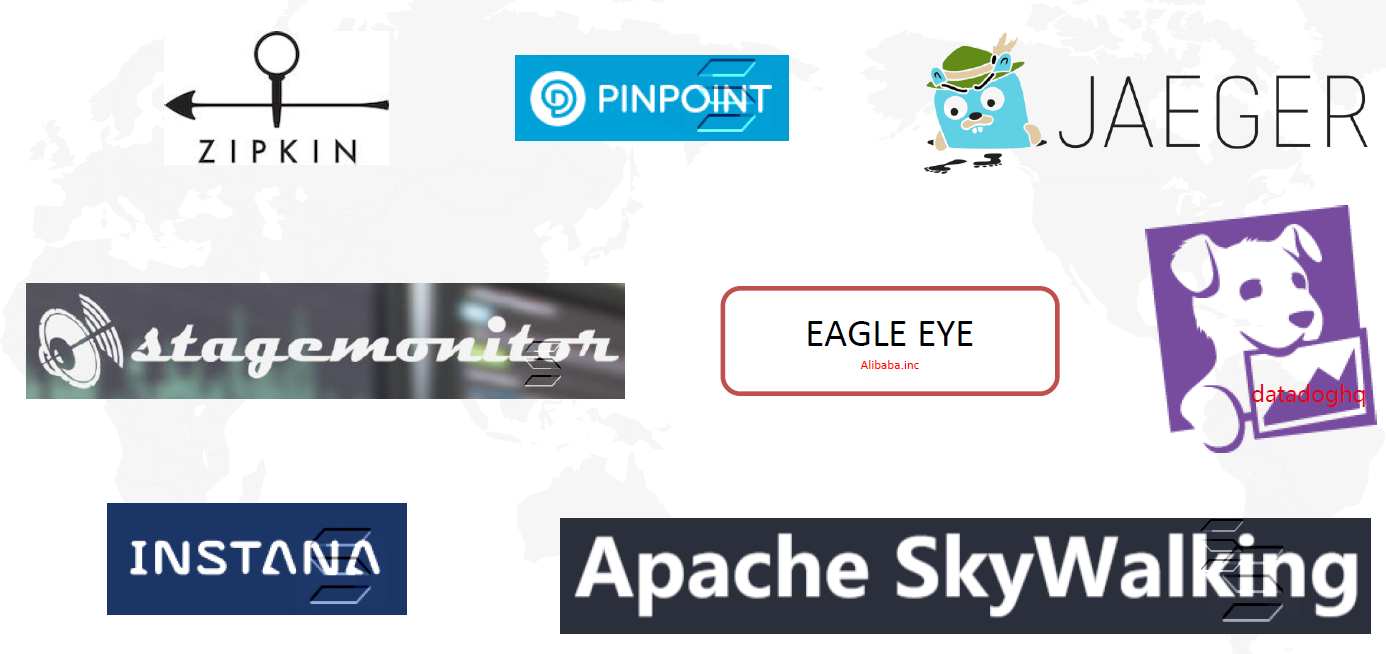
Google Dapper是分布式跟踪理论的重要参考，它定义了链路跟踪的相关概念及模型。



OpenTracing是使用了Dapper的相关概念及模型的链路跟踪的开源标准。它为链路跟踪提供了一个“标准规范”。

遵循OpenTracing规范的产品有很多，如Zipkin、Uber开源的JAEGER、阿里巴巴的“鹰眼”等。

用友云技术中台提供的YYEYE也遵循此规范。



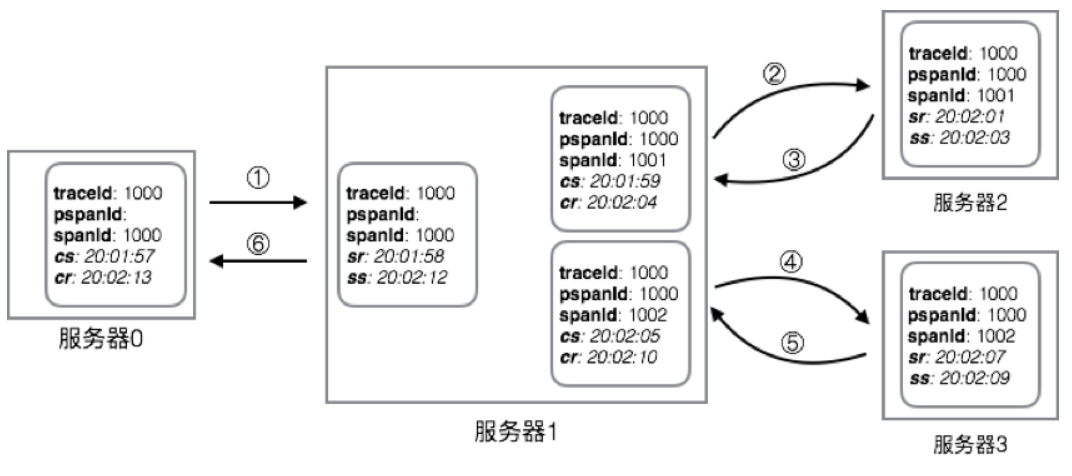
### 链路跟踪的核心的几个概念

* Span：分布式跟踪的主要组件之一，标识在分布式调用中的单个工作单元；
* Tag：用于自定义的标记，用于查询、过滤和理解跟踪数据；
* SpanContext：Span传输的上下文，包含但不限于以下内容：
  + traceId：链路ID，标识整条调用链路；
  + spanId：标识单个工作单元的调用ID；
  + parentSpanId：在此工作单元之前的父调用ID。

### 链路跟踪的传播模型

链路跟踪的传播模型分为四个阶段，分别是：

* CS（ClientSend）：消费者发起远程调用前，消费者创建SpanContext实例，其中包含了traceId（如果没有则生成）和parentSpanId（如有上层调用的话）；
* SR（ServerReceive）：服务提供者接收到请求后，从请求中解析出（traceId，spanId，parentSpanId，如果有上层调用的话），如果没有traceId，则启用新跟踪；
* SS（ServerSend）：服务提供者响应前：关闭服务提供者span对象；
* CR（ClientReceive）：消费者接收到响应后，关闭消费者span对象。



## 链路跟踪相关产品介绍

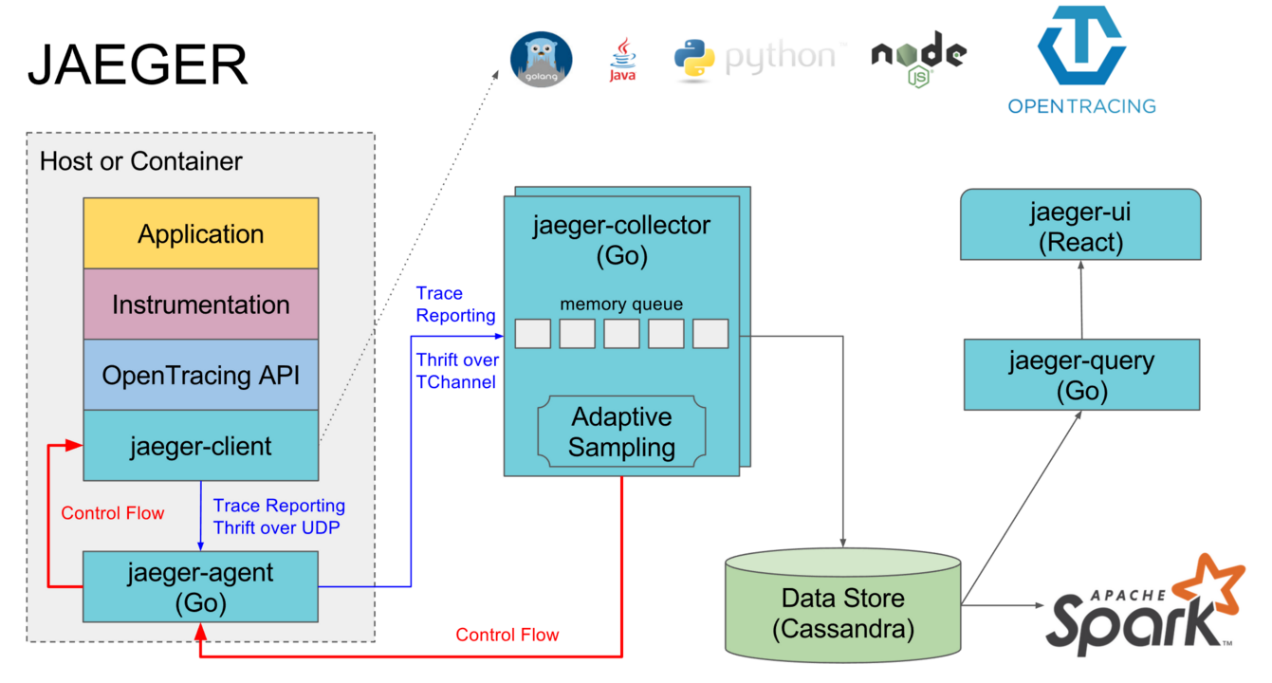
### 链路跟踪产品组成

市面上链路跟踪产品有很多，各类产品的组件构大致相同，一般均包含以下组件：

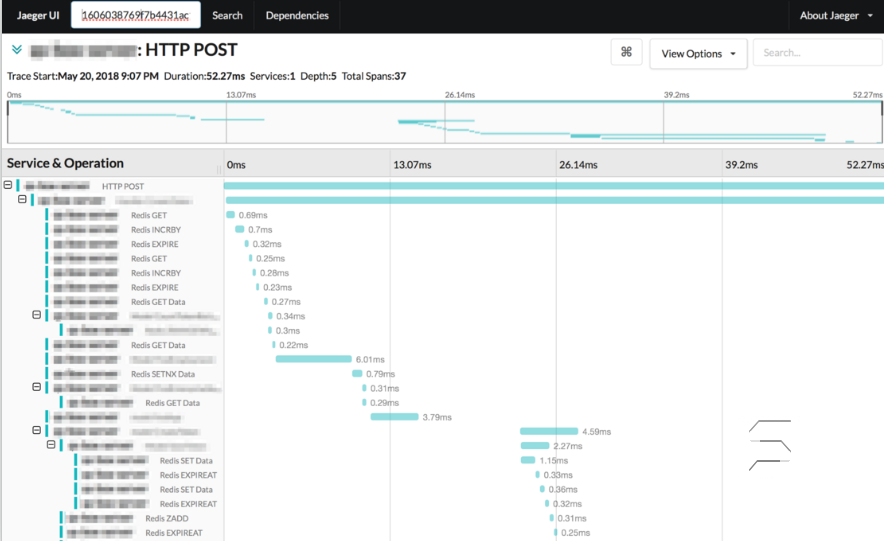
* Client：链路跟踪客户端，用于集成在应用内，记录链路跟踪的因果关系和控制流。一般以自定义应用代码、发布的Library、系统服务、RPC/IPC框架等形式存在，可以调整采样率的大小；
* Agent：通过某种形式（如消息队列、TCP/UDP）发送Client记录的数据到Collector；
* Collector：收集从Agent发送来的数据并放入处理队列，逐条验证、索引和存储队列里的数据到底层存储；
* Query：以各种维度显示链路跟踪数据的分析结果。

### JAEGER介绍

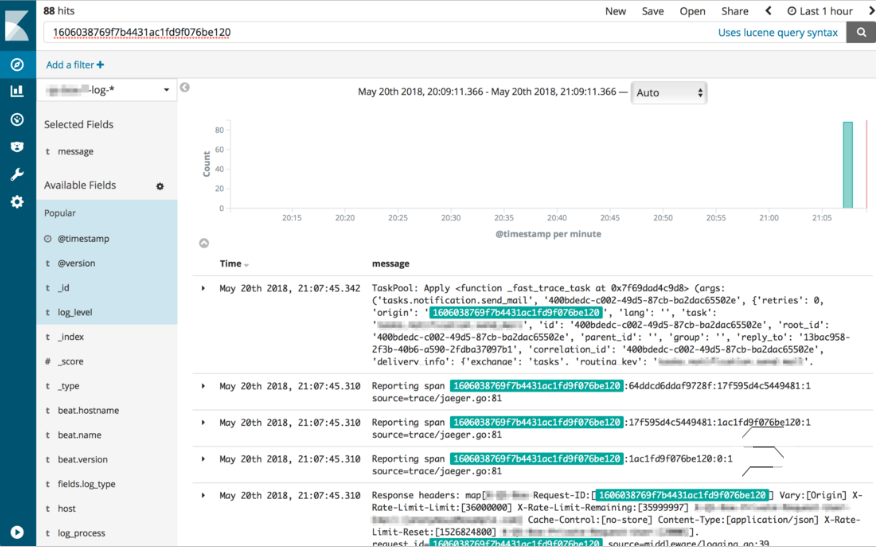
JAEGER是Uber开源的分布式链路跟踪系统，用于基于微服务系统的监控和问题定位。



**JAEGER的组件及架构图**



**JAEGER的系统UI**

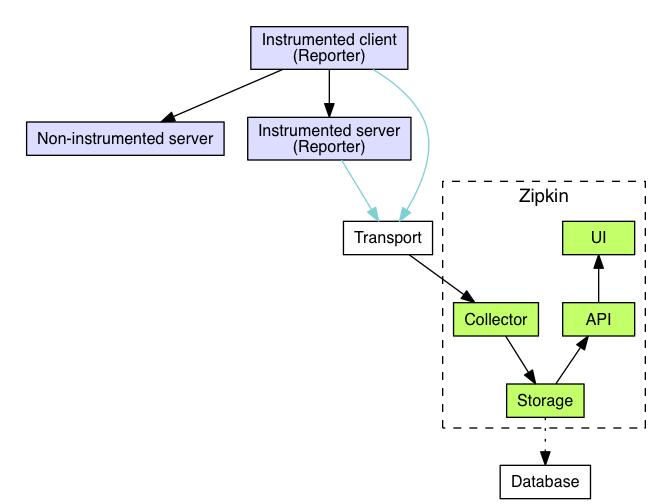


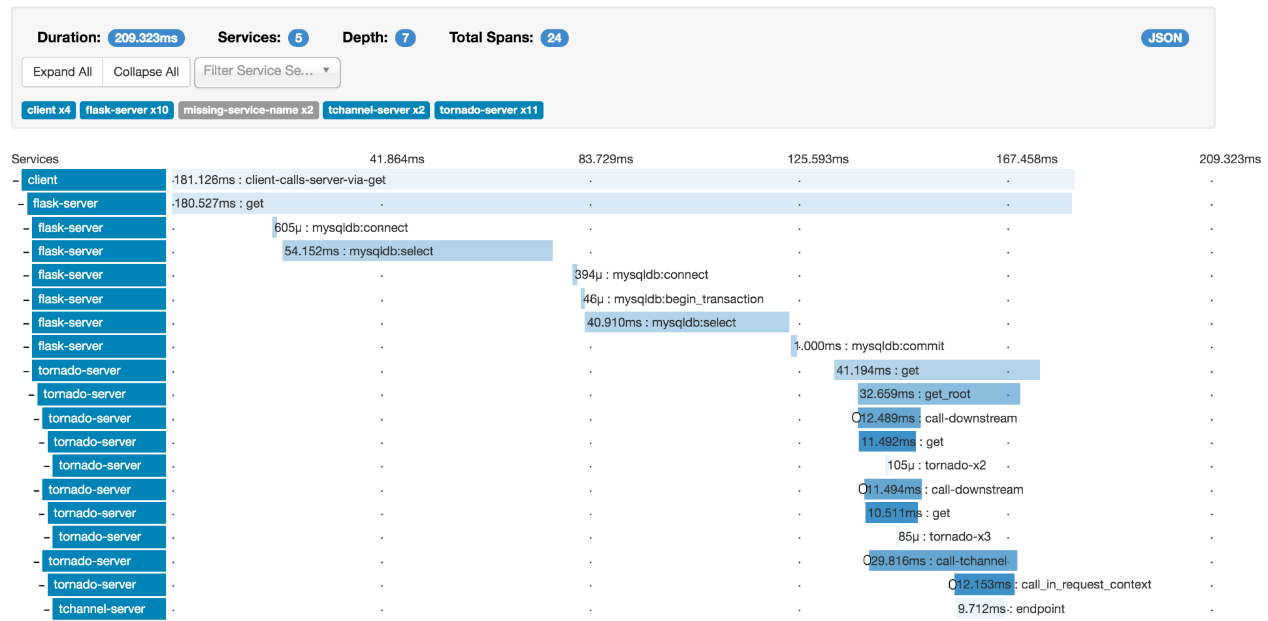
**在JAEGER中查看链路详细信息**

### Zipkin介绍

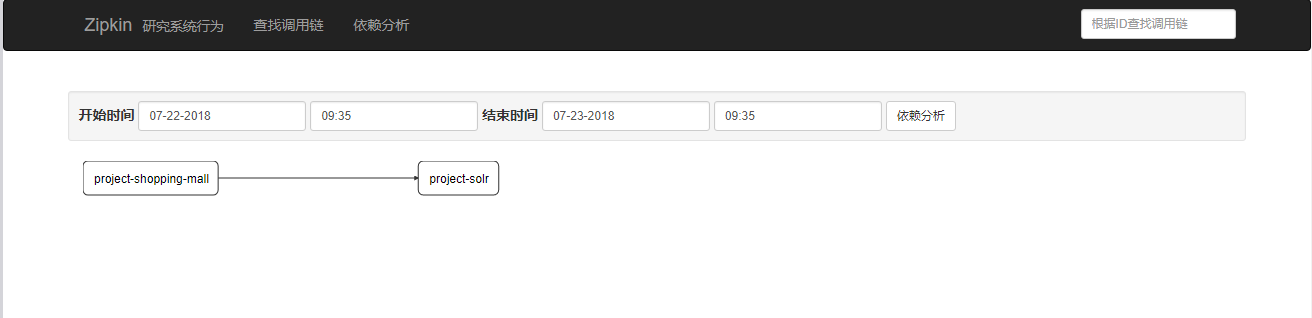
Zipkin 的架构和组件由以下部分组成：

* Reporter：将数据发送到Zipkin的Client；
* Transport：传输通道（Http、Kafka、Log）
* Collector：kafka，SQS；
* Storage：Cassandra，ElasticSearch 或MySQL；
* WebUI：根据服务/耗时/annotation查询。





### Zipkin的查询页面



### Zipkin研究系统性行为

值得一提的是，Zipkin 作为SpringCloud的标准链路跟踪组件，较为容易集成至系统且使用方便。

SpringCloud微服务工程变成ZipkinServer仅需三步操作：

* 加入zipkin依赖 ；
* 配置zipkin-server应用名称；
* 标注@EnableZipkinServer注解。

启动应用后就可看到ZipkinServer的页面了。

SpringCloud微服务工程成为Zipkin的客户端只要两步操作：

* 加入sleuth依赖；
* 配置Zipkin-Server的地址指向和采样率（可选）。

启动后发起微服务间调用就可以在Zipkin里看到链路跟踪和分析的数据了。

## 微服务链路跟踪YYEYE及中台日志收集和预警平台建设

### 微服务链路跟踪YYEYE

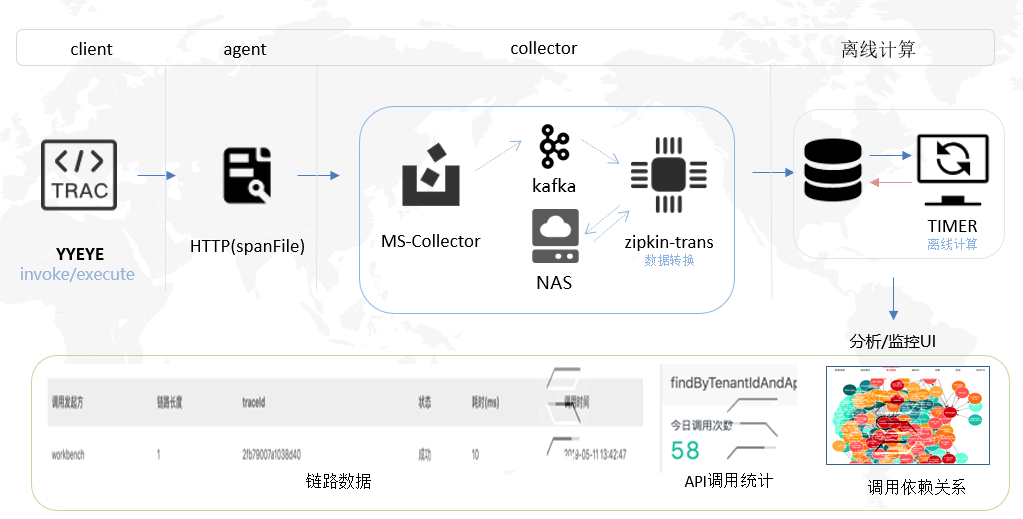
用友YYEYE（yonyou eye）参考了Zipkin和其他同类产品，不断的进行性能及产品方面的优化。

在微服务链路查询维度上，用友YYEYE具有同类产品的链路查询及链路详情查看功能，同时可按接口微服务接口的维度进行查询和显示实时的调用次数，调用正确率统计等信息，还可根据链路调用层级深度或根据单个服务查看调用依赖关系。

在Client收集和Agent发送的性能上，用友YYEYE从刚开始的使用Kafka发送消息，到后来使用HTTP发送链路数据，再到现在的使用NIO写入链路文件，减少了链路跟踪对整体应用的性能影响。Agent将链路跟踪文件进行压缩后，批量发送至Collector。同时，它构建了使用资源少且高效的数据上报流程。

在使用层面，用友YYEYE默认集成到了微服务RPC SDK，使用span.trace.off和span.trace.ratio分别控制是否启用链路采用及调节链路采样率。

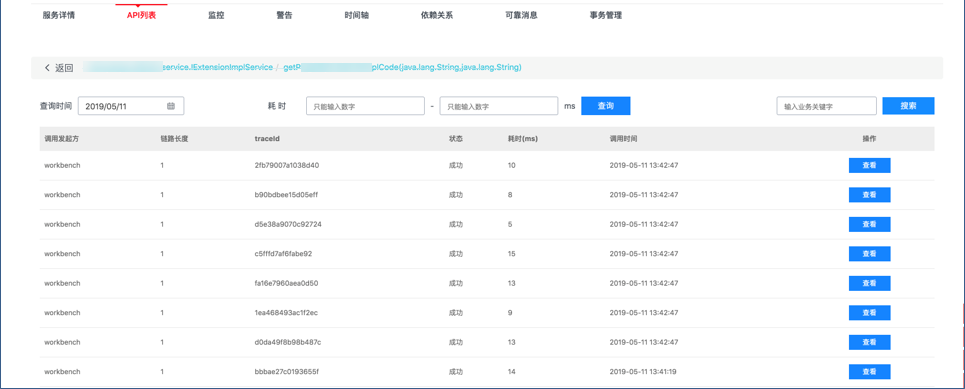
在计算维度层面，它为在线和离线计算分别存储了链路详细信息，实时计算调用次数及调用正确率信息，离线计算了各服务和接口的调用依赖关系。



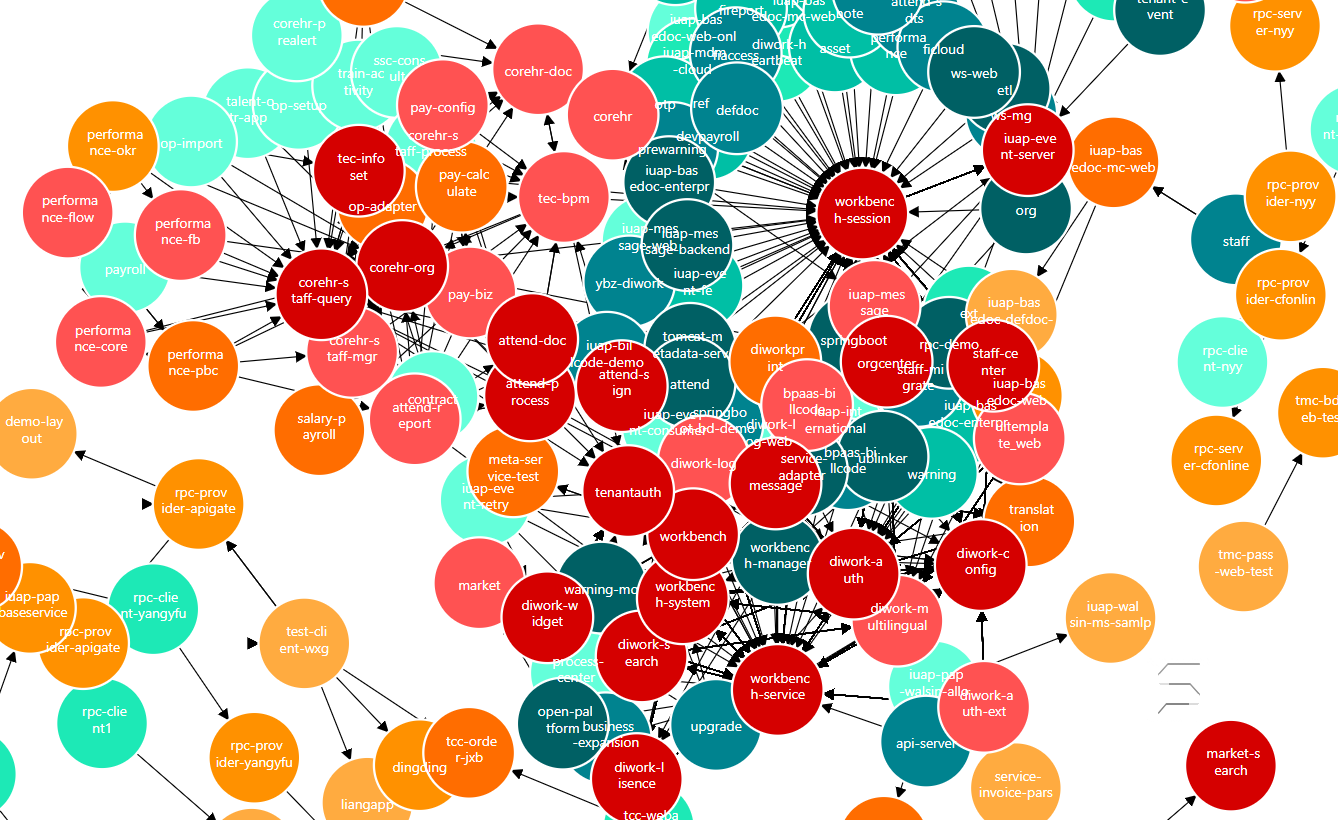
**YYEYE链路跟踪架构及组件**



**YYEYE的查询页面**



**YYEYE针对单个接口的链路跟踪**



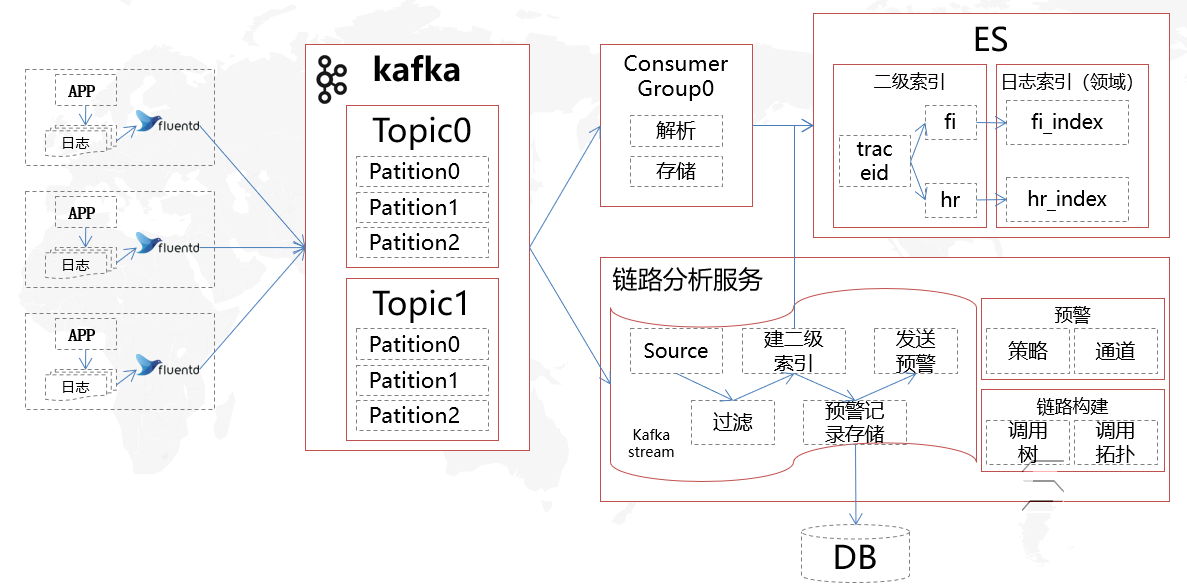
**YYEYE针对服务间调用的关系图**

### 用友中台日志收集和预警平台

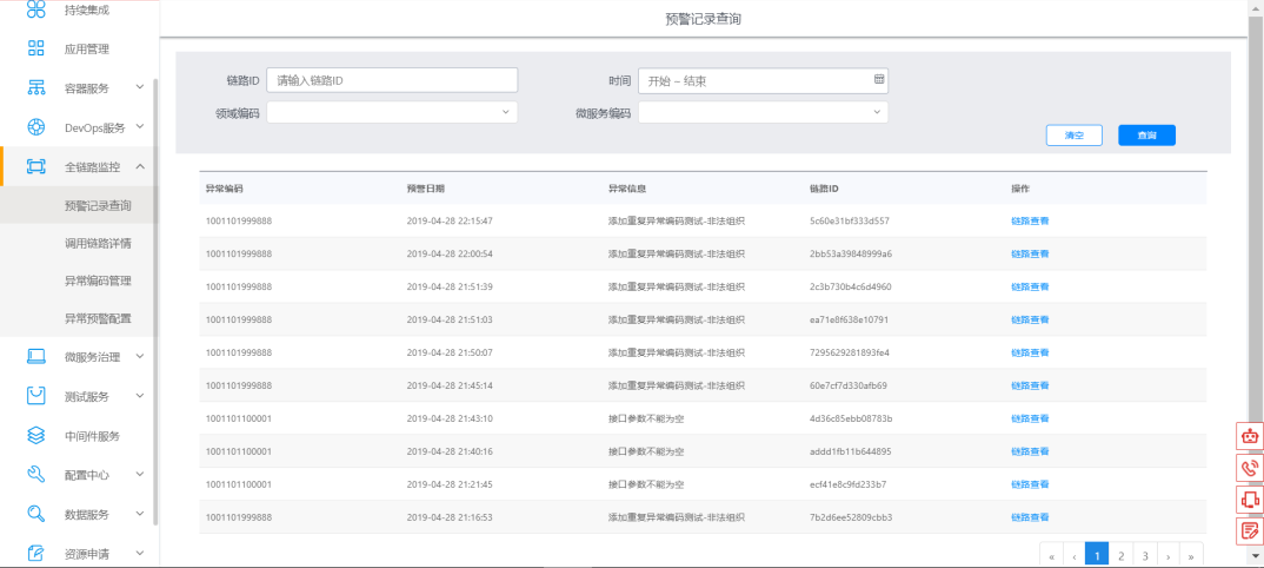
用友中台日志收集和预警平台面向接入用友云的内外部客户，提供异常预警和跨业务的统一链路查询功能。

在功能上说，它具有如下特点：

* 它提供了适合各业务场景的链路跟踪Collector-SDK；
* 它对中间件（如内外网统一域名接入）链路跟踪支持；
* 它制定了链路跟踪日志规范，并推广到各云服务中，方便统一的日志收集；
* 它在统一DevOps平台植入了Agent，方便统一管理和升级；
* 它统一了异常规范和异常预警编码。



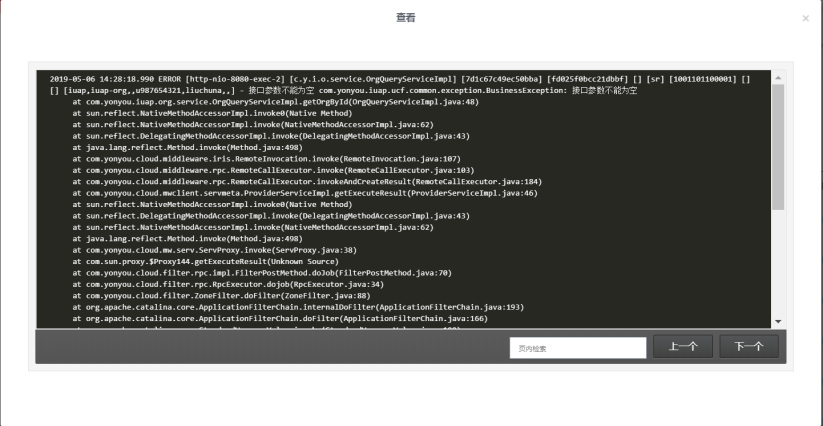
**中台日志收集和预警平台架构**



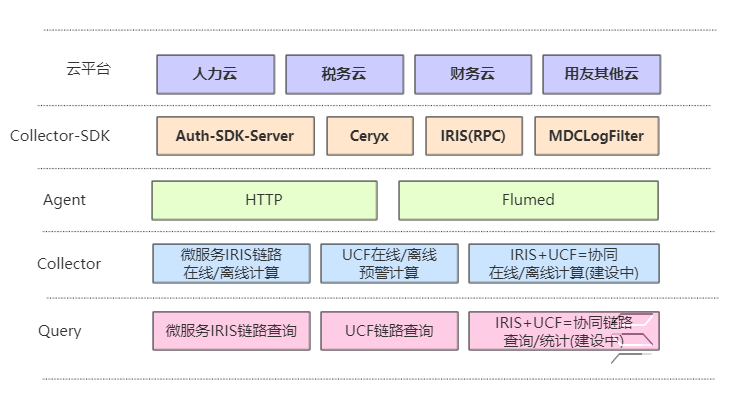
**基于链路跟踪的预警UI及链路查询**



**查看预警详情**



**查看日志信息**



**协同分布式链路跟踪架构**

用友中台日志收集和预警平台和微服务整合后的链路协同，可根据业务主键及事件类型进行跨业务、跨系统、跨中间件的“全息排查”，在业务问题和调用性能方面各具侧重点的精确定位系统问题和性能瓶颈。

## 总结与展望

链路跟踪是微服务时代下不可缺少的业务和性能问题定位利器。有了链路跟踪平台的支撑，可以帮助业务架构师从全局的角度更直观和准确的看到业务流程所出现的漏洞，做出相应的调整和优化，腾出更多的精力聚焦于产品本身的优化上，而不是流程漏洞的细节查找上。

同时，链路跟踪平台也可以帮助开发人员迅速和准确的定位系统性能问题，方便直观的定位性能瓶颈，省去了在问题一次次出现时，反复的找各系统的开发者沟通和定位性能问题，给了开发者定位问题的眼睛（YY EYE）。

YYEYE及用友云技术中台日志收集和预警平台，专业为微服务而生，幻化成微服务世界的双眼，带您领略微服务世界的浩瀚！