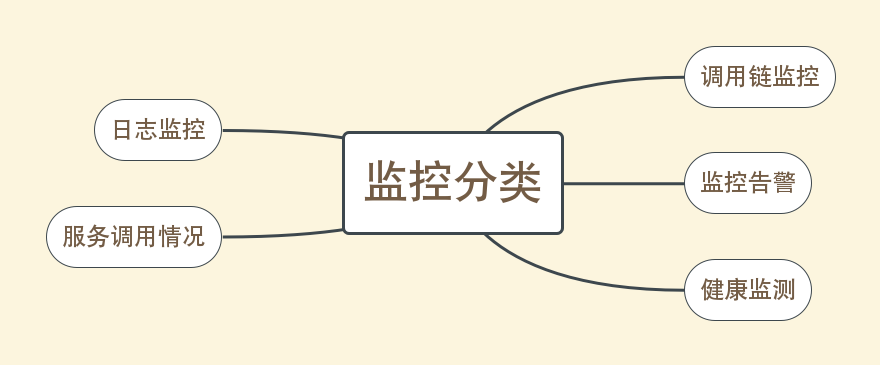
# 微服务下的监控

在这个云原生横行的时代，微服务在生产实践中被大量使用，后台系统服务数量暴增。在以往单应用的环境下，我们可以很快的定位和处理问题，但是在微服务架构下，定位和处理问题就变得并不那么简单了。

## 监控什么

监控是服务治理的重要环节。一般分为以下五个分类



## 日志监控

对于日志监控，我们常用就是ELK，K8S默认的EFK，可是对于微服务来说，我们就需要每一个微服务镜像中都集成日志收集组件。对此我们通常的做法是

|  |  |  |  |
| --- | --- | --- | --- |
|  | 方案 | 优点 | 缺点 |
| 1 | 每个微服务的镜像都集成日志收集组件 | 部署方便，可以每个app自定义日志收集组件 | 强耦合，不方便应用和日志收集组件升级和维护，而且会导致镜像过大 |
| 2 | 单独创建一个日志收集组件跟app的容器一起运行在同一个pod中 | 低耦合，扩展性强，方便维护和升级 | 需要对kubernetes的yaml文件进行单独配置，略显繁琐 |
| 3 | 将所有的Pod的日志都挂载到宿主机上，每台主机上单独起一个日志收集Pod | 完全解耦，性能高，管理方便 | 需要统一日志收集规则、目录和输出方式 |

日志收集之后，我们会发现有一个致命的缺点，每个服务的日志都是独立的，并没有关联起来，所以我们需要用到调用链监控

## 调用链监控

Dapper是Google的分布式系统的跟踪系统（Google论文[Dapper, a Large-Scale Distributed Systems Tracing Infrastructure](http://research.google.com/pubs/pub36356.html)），论文中提出dapper的设计目标

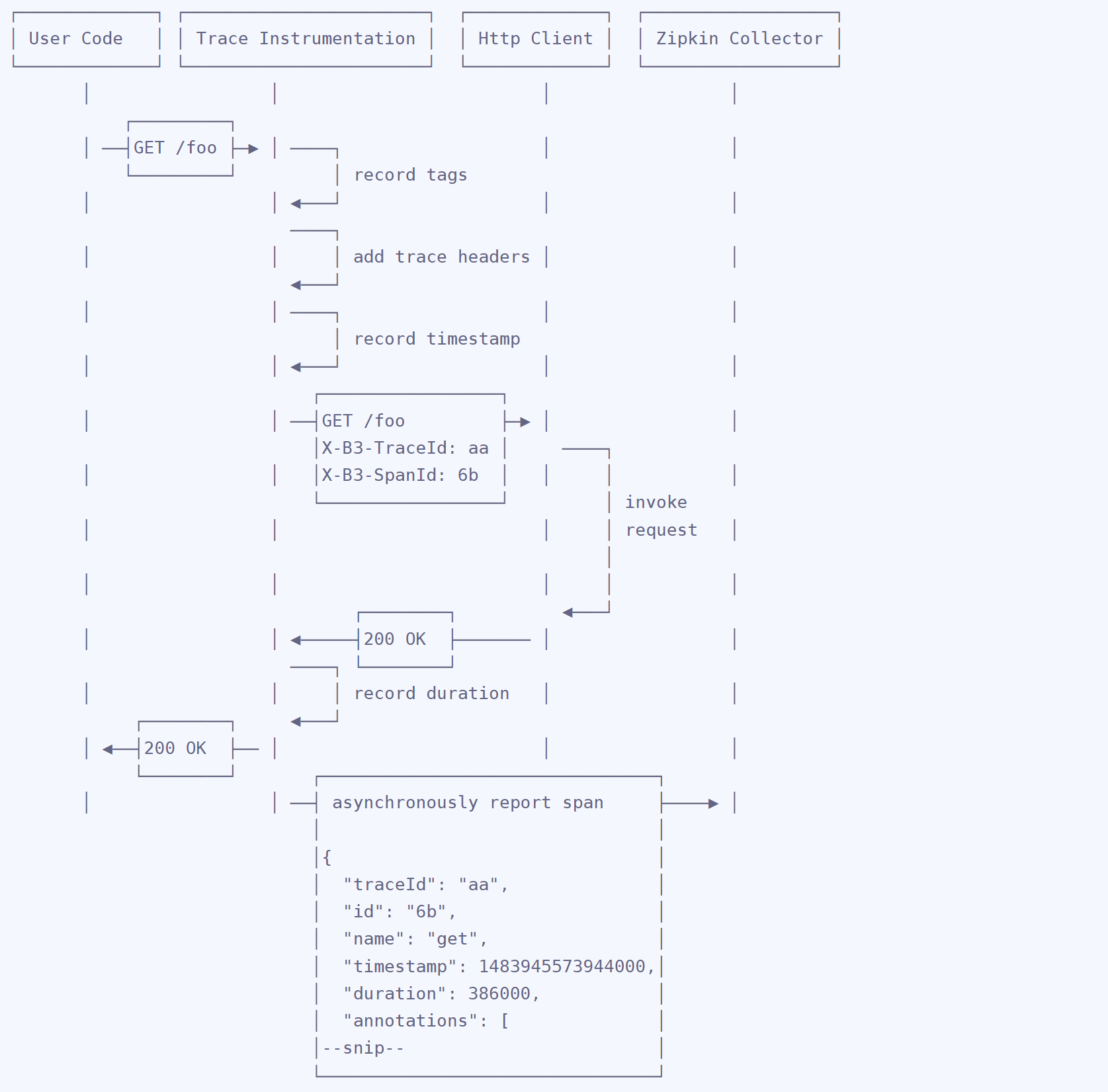
1. 低消耗。跟踪系统对在线服务的影响应该做到足够小。在一些高度优化过的服务，即使一点点损耗也会很容易察觉到，而且有可能迫使在线服务的部署团队不得不将跟踪系统关停。
2. 应用级透明。对于应用的程序员来说，是不需要知道有跟踪系统这回事的。如果一个跟踪系统想生效，就必须需要依赖应用的开发者主动配合，那么这个跟踪系统也太脆弱了，往往由于跟踪系统在应用中植入代码的bug或疏忽导致应用出问题。
3. 扩展性好。对于快速发展的互联网公司而言，用户规模快速增长导致服务以及机器数量越来越多，因此需要dapper需要适应相应的发展

介绍dapper是因为我们的链路追踪系统是基于开源框架zipkin做的，而zipkin是基于dapper论文实现的。

Zipkin分为两部分，zipkin client是zipkin基于不同的语言封装的一系列客户端工具，这些工具生成追踪数据以及上报的功能。

各业务系统在彼此调用时，将特定的跟踪消息传递至zipkin,zipkin在收集到跟踪信息后将其聚合处理、存储、展示等，用户可通过web UI方便获得网络延迟、调用链路、系统依赖等等。

下图是zipkin的原理



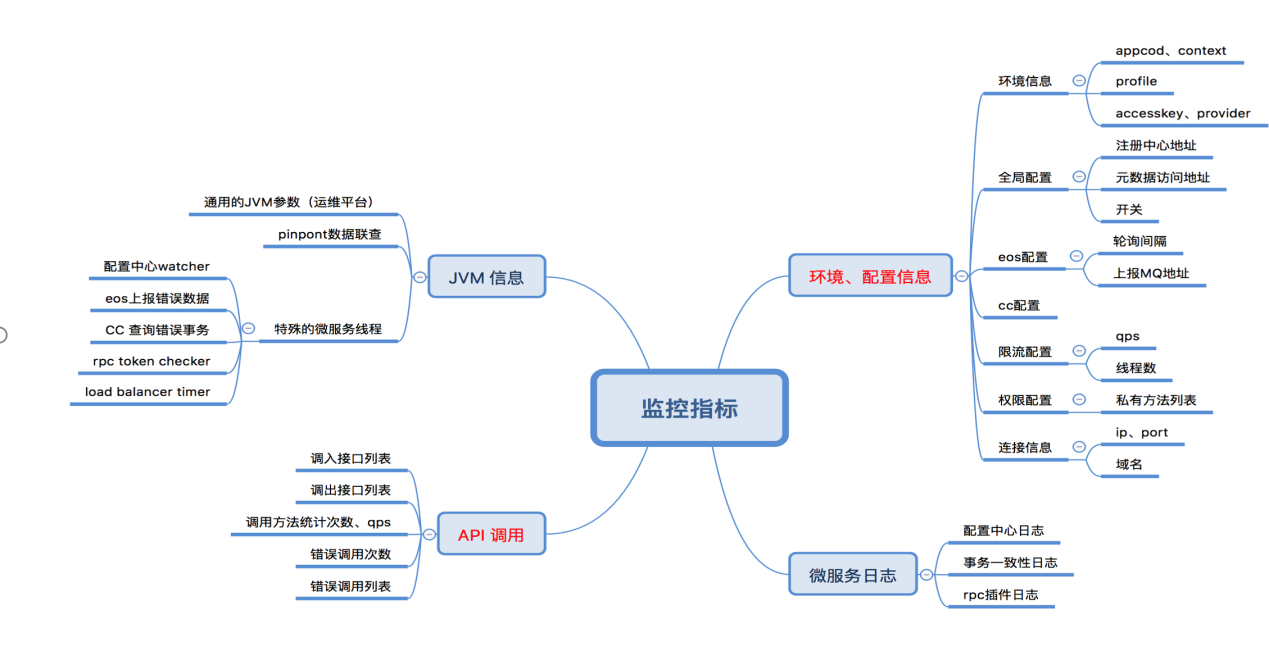
## 服务调用情况

度量标准代表可以在整个系统中观察和收集的资源使用情况或行为的原始度量。 这些可能是操作系统提供的低级使用情况摘要，也可能是与组件的特定功能或工作相关的更高级别的数据类型，例如每秒的请求或一些特殊线程的状态。 一些指标是与总容量相关的，而另一些指标是表示组件“繁忙程度”的比率。

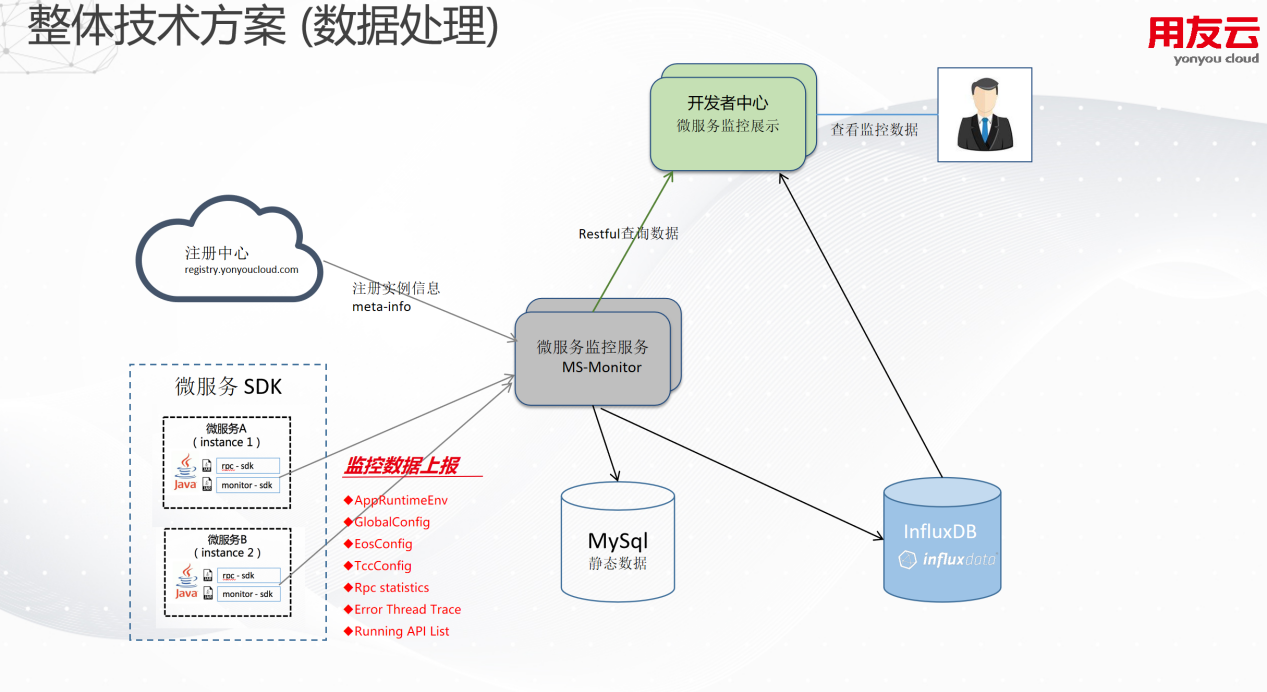
一般来说，度量和监控之间的差异反映了数据和信息之间的差异。 数据由原始的，未经处理的事实组成，而信息则是通过分析和组织数据来创建，以建立提供价值的上下文。 监控将采集指标数据，汇总数据，并以各种方式呈现，从而使人们从各个作品集合中提取见解。

监控系统完成许多相关的功能。 他们的首要责任是接受和存储传入和历史数据。 虽然表示当前时间点的值是有用的，但是查看这些与过去值相关的数字以提供围绕变化和趋势的上下文几乎总是更有帮助。 这意味着一个监控系统应该能够管理一段时间的数据，这可能涉及对旧数据进行采样或汇总。

在我们的微服务中，有以下度量的监控



我们在rpc调用中收集单位时间内rpc调用的次数、响应时间、调用是否成功等度量信息。上报到云端进行计算和持久化。



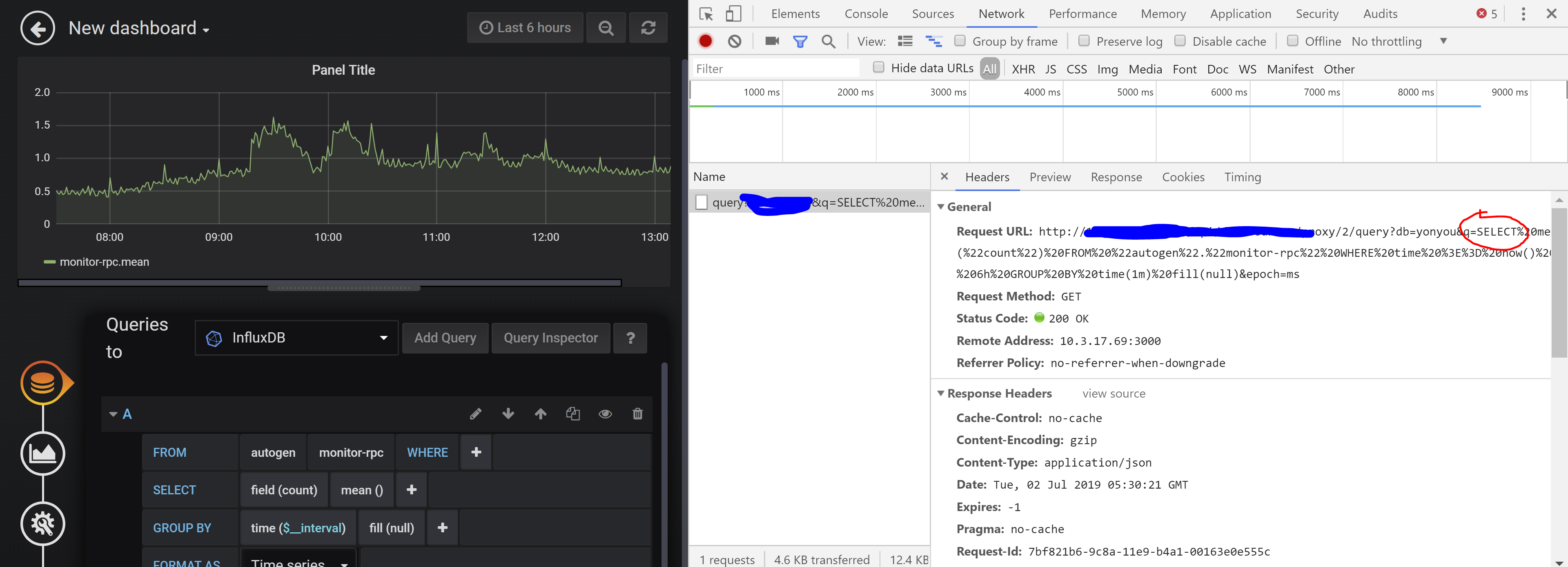
时序数据库我们选用了influxDB，influxDB是开源的分布式时序数据库，使用go编写，无需外部依赖。我们选择influxDB主要是因为以下几个优点。

* 无特殊依赖，开箱即用
* 自带数据过期功能
* 权限管理精细到表级别
* 原生http支持，内置http api
* 强大的类SQL语法，支持min,max,sum,count,mean,median等函数，方便统计

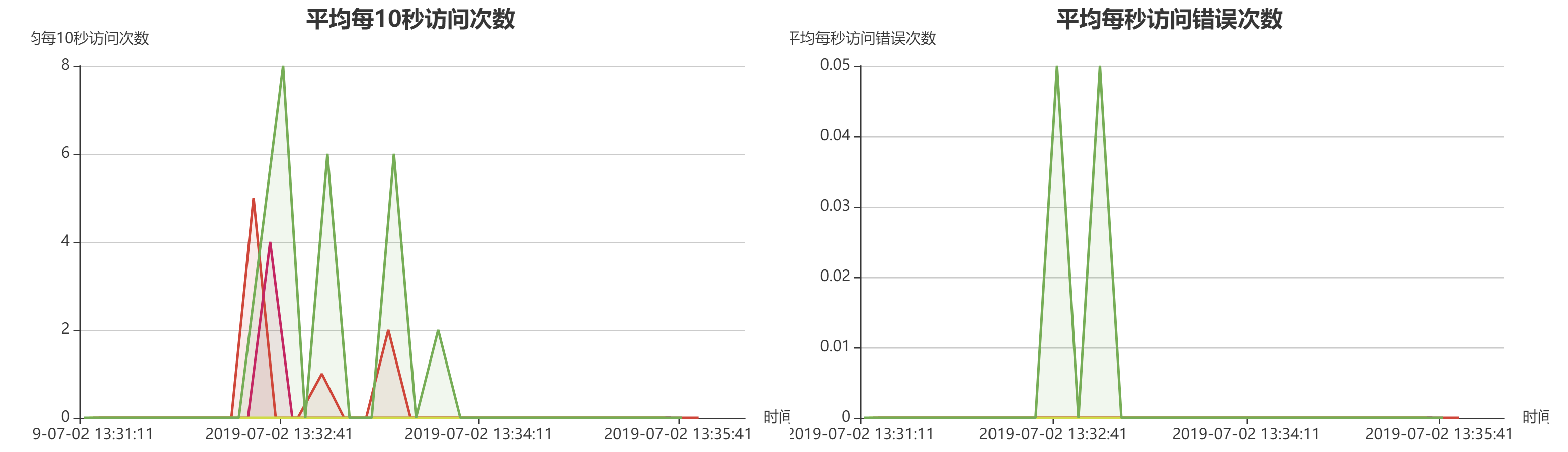
可视化最开始我们选用强大的Grafana，只用配置好数据源，设置好查询条件就可以。

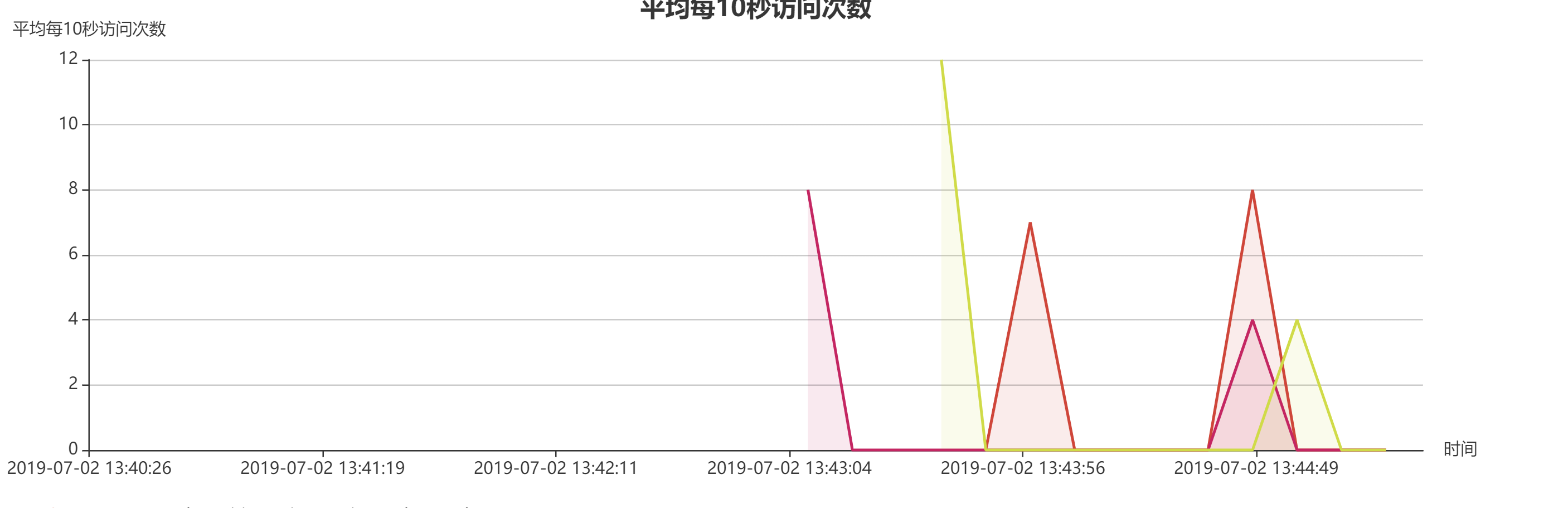


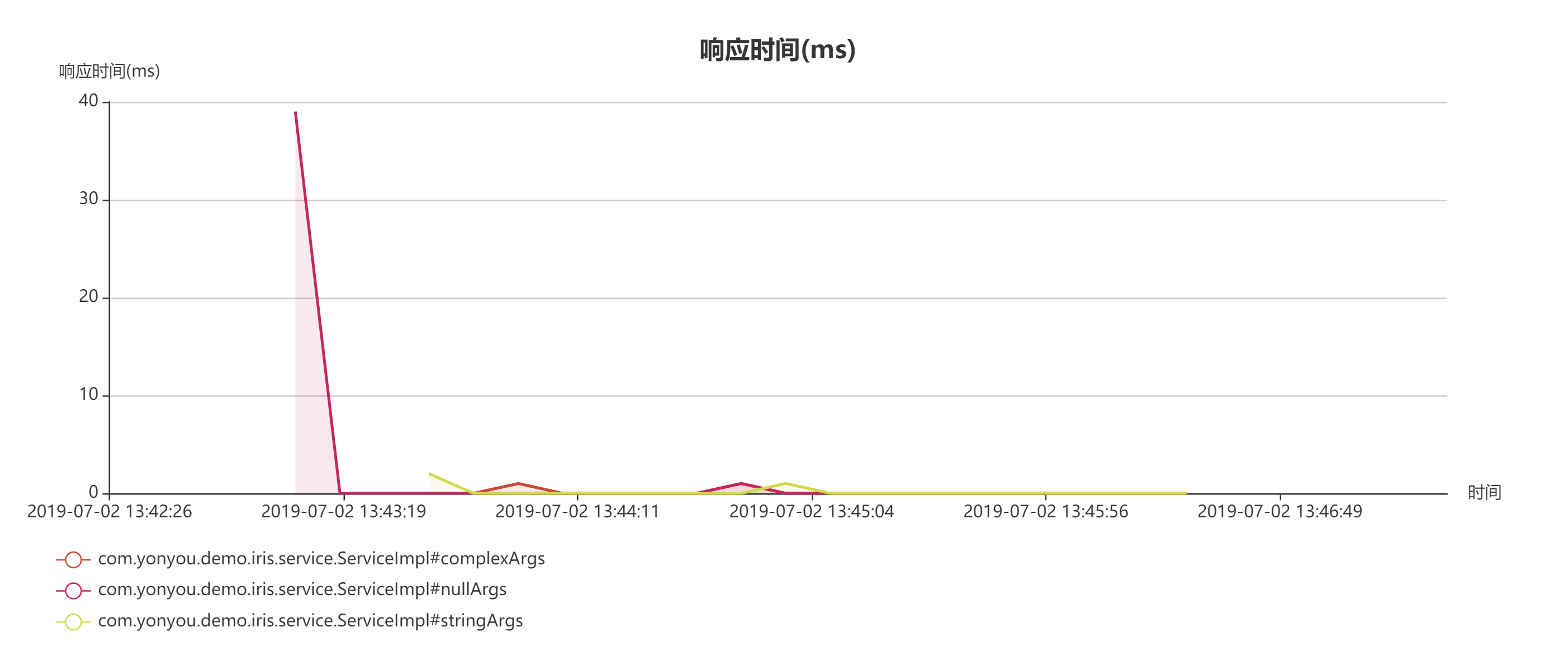
在使用过程中我们遇到了一些问题，grafana在发送查询请求的时候被阿里云认为有sql注入而被拦截。



所有我们自己做了可视化界面。如下图是每个应用的调用情况，我们监控的最细力度是方法级







除了以上的度量信息，我们还监控了微服务的配置信息。用户可以实时的查看，后续我们还会做到在查看页面实时修改并推送给实例，达到动态修改的效果。



## 监控告警

虽然监控系统对于积极的解释和调查是非常有用的，但是完整的监控系统的主要优势之一是让管理员脱离系统。警报允许您定义有意义的情况以积极管理，同时依靠软件的被动监视来监视变化的情况。

尽管通知责任方是最常见的警报行为，但也可以基于阈值违规触发一些程序性响应。 例如，指示您需要更多CPU来处理当前负载的警报可以使用自动调整应用程序层的脚本来响应。 虽然这不是严格的警报，因为它不会导致通知，但同样的监控系统机制通常也可以用来启动这些过程。

但是，提醒的主要目的仍然是引起人们的注意力，以对您系统的当前状态进行评估。 自动化响应是确保只有在知识丰富的人需要考虑的情况下才触发通知的重要机制。 警报本身应该包含有关什么是错误的信息，以及在哪里可以找到更多的信息。 然后，响应警报的个人可以使用监控系统和相关工具（如日志文件）来调查问题的原因并实施缓解策略。

我们通过SDK埋点的方式监控了rpc的调用异常和一些自定义事件，并上报云端。比如我们监控了一个rpc调用耗时超过了预警值，我们就把堆栈信息记录，可以让用户知道服务目前正在执行什么操作。

## 未来展望

我们正在努力实现以下功能：

* 支持更多的rpc
* 更多的度量
* 智能发现问题
* 更多展现方式