

02 | 基本概念：监控圈子有哪些行业黑话？

2023-01-11 秦晓辉 来自北京

《运维监控系统实战笔记》



你好，我是秦晓辉。

上一讲我们介绍了监控解决的问题域，并对业界常见开源方案做了横评对比。这一讲我们继续学习监控的相关概念，包括监控、监控指标、指标类型、时序库，还有告警收敛与闭环等。理清监控圈子的这些常用术语之后，我们学习起来会更轻松一些。

监控这个词在不同的上下文含义会有一些区别，一般我们说监控 MySQL、监控 Redis 的时候，都是指能够采集到 MySQL、Redis 的监控数据，并能可视化展示。这时候监控表示数据采集和可视化，不包括告警引擎和事件处理。但当我们讲监控系统的时候，因为说的是整个系统，所以也会包含告警和事件发送等相关功能。

监控体系中最基础的是监控指标，监控系统就是围绕指标的采集、传输、存储、分析、可视化的一个系统，下面我们就从监控指标这个概念讲起。


监控指标

监控指标是指数值类型的监控数据，比如某个机器的内存利用率，某个 MySQL 实例的当前连接数，某个 Redis 的最大内存上限等等。不同的监控系统，对于监控指标有不同的描述方式，典型的方式有三种，下面我们分别介绍一下。

全局唯一字符串作为指标标识

监控指标通常是一个全局唯一的字符串，比如某机器的内存利用率


host.10.2.3.4.mem_used_percent，这个字符串中包含了机器的信息，也包含了指标名，可以唯一标识一条监控指标。假设监控数据采集的频率是 30 秒，2 分钟内采集了 4 个数据点，一个数据点包含一个时间戳和一个值，我们看一下如何用 JSON 表示这个监控指标及其监控数据。

 复制代码

```
1 {
2   "name": "host.10.2.3.4.mem_used_percent",
3   "points": [
4     {
5       "clock": 1662449136,
6       "value": 45.4
7     },
8     {
9       "clock": 1662449166,
10      "value": 43.2
11    },
12    {
13      "clock": 1662449196,
14      "value": 44.9
15    },
16    {
17      "clock": 1662449226,
18      "value": 44.8
19    }
20  ]
21 }
```

相对老一些的监控系统，比如 Graphite，就是使用这种方式来标识监控指标的。一些老的监控数据采集器，比如 Collectd，也是这样标识监控指标的。

虽然这种方式一目了然，非常清晰，但是缺少对维度信息的描述，不便于做聚合计算。比如下面几条用于描述 HTTP 请求状态码的指标。

 复制代码


```
1 myhost.service1.http_request.200.get
2 myhost.service1.http_request.200.post
3 myhost.service1.http_request.500.get
4 myhost.service1.http_request.500.post
5 myhost.service2.http_request.200.get
6 myhost.service2.http_request.500.post
```


myhost 这个机器上部署了两个服务，分别是 service1 和 service2。有些请求状态码是 200，有些是 500，有些 HTTP method 是 get，有些是 post。我们想分别统计这些指标的数量，就要把这些分类维度信息都拼到指标标识中。但是这样做会产生两个弊端：一是看起来比较混乱，二是不方便聚合统计。

举个例子，比如我想计算 service1 这个服务的成功率，要把 service1 里所有状态码为 200 的请求数量拿到，除以 service1 所有请求的数量。如果预先知道所有状态码、所有 HTTP method，则可以枚举指标名称，拉取监控数据。如果不知道所有状态码和 HTTP method，就要先用正则匹配指标标识，查询出指标列表再拉取监控数据做计算，处理起来非常麻烦。

其实我这是用马后炮视角解释这个问题的，实际上，在 Graphite、Collectd、Zabbix、Cacti 这些软件盛行的时代，这个问题并不明显。因为那个时代主要关注的是机器设备、数据库、中间件的监控，不会有很多维度的信息。直到业界开始关注应用层面监控的时候，才暴露出这个问题。

标签集的组合作为指标标识

2010 年左右，有一款名叫  OpenTSDB 的时序数据库诞生了。虽然现在已经较少使用了，但是 OpenTSDB 描述指标的方式，对业界有很大影响。下面是通过文本协议推给 OpenTSDB 的数据示例，我们可以从中看出指标标识的定义方式。


 复制代码

```
1 mysql.bytes_received 1287333217 327810227706 schema=foo host=db1
```

```
2 mysql.bytes_sent 1287333217 6604859181710 schema=foo host=db1
3 mysql.bytes_received 1287333232 327812421706 schema=foo host=db1
4 mysql.bytes_sent 1287333232 6604901075387 schema=foo host=db1
5 mysql.bytes_received 1287333321 340899533915 schema=foo host=db2
6 mysql.bytes_sent 1287333321 5506469130707 schema=foo host=db2
```

上面这 6 条监控指标，都通过空格把指标分隔成了多个字段。第一段是指标名，第二段是时间戳（单位是秒），第三段是指标值，剩下的部分是多个标签（tags/labels），每个标签都是 key=value 的格式，多个标签之间使用空格分隔。

除了 OpenTSDB，新时代的时序库大都引入了标签的概念，比如 Prometheus，它们甚至认为指标名也是一种特殊的标签（其标签 key 是 __name__），所以 Prometheus 仅仅使用标签集作为指标标识，从 Prometheus 的数据结构定义中就可以看出来。

 复制代码

```
1 message Sample {
2     double value = 1;
3     int64 timestamp = 2;
4 }
5
6 message Label {
7     string name = 1;
8     string value = 2;
9 }
10
11 message TimeSeries {
12     repeated Label labels = 1 [(gogoproto.nullable) = false];
13     repeated Sample samples = 2 [(gogoproto.nullable) = false];
14 }
```

TimeSeries 这个结构中并没有一个单独的 metric 字段，指标名的信息实际是放到了 labels 数组中了。

优雅高效的 Influx 指标格式

InfluxData 公司有一款开源时序库非常有名，叫 [InfluxDB](#)，InfluxDB 在接收监控数据写入时，设计了一个非常精巧的指标格式，一行可以传输多个指标。

[复制代码](#)

```
1 measurement,labelkey1=labelval1,labelkey2=labelval2 field1=1.2,field2=2.3 timestamp
```

总体来看，分为 4 个部分，measurement,tag_set field_set timestamp，其中 tag_set 是可选的，tag_set 与前面的 measurement 之间用逗号分隔，其他各个部分之间都是用空格来分隔的。我们可以通过下面的例子来理解。

[复制代码](#)

```
1 weather,location=us-midwest temperature=82 1465839830100400200
2 |-----|
3 |           |           |           |
4 |           |           |           |
5 +-----+-----+-----+-----+
6 |measurement|,tag_set| |field_set| |timestamp|
7 +-----+-----+-----+-----+
```

我们把上面 OpenTSDB 的指标示例改写成 Influx 格式，结果是这样的。

[复制代码](#)

```
1 mysql,schema=foo,host=db1 bytes_received=327810227706,bytes_sent=6604859181710 12
2 mysql,schema=foo,host=db1 bytes_received=327812421706,bytes_sent=6604901075387 12
3 mysql,schema=foo,host=db2 bytes_received=340899533915,bytes_sent=5506469130707 12
```

注意，时间戳的单位是纳秒。这种写法设计很精巧，标签重复度低，field 越多的情况下它的优势越明显，网络传输的时候可以节省更多带宽。当然了，OpenTSDB 的格式或者 Prometheus 的格式如果做了数据压缩，节省带宽的效果也是不错的，因为字符的压缩效果一般比较明显。现如今机房内部通信动辄万兆网卡，这个流量的大小区别倒也不用太关注。

指标标识	优点	缺点
全局唯一的字符串	简单	缺少维度信息不便于做聚合计算和灵活筛选
标签集的组合	灵活	稍显冗余
优雅高效的 Influx	灵活、精巧、语义丰富	理解成本稍高

监控指标的概念非常重要，可以说监控系统中的一切都是围绕监控指标来的，所以我们用了大量篇幅来解释监控指标，还讲解了几种不同指标的描述方式。除了这些描述方式之外，监控指标还分为各种不同的类型，下面我们一起来看一下。

指标类型

有些监控系统是不区分指标类型的，有些会做区分。90 年代左右社区就出现了一款作品 [RRDtool](#)，它是一个环形数据库，也是一个绘图引擎，很多监控工具都是使用 RRDtool 来存储或绘制监控趋势图的，比如 Cacti、MRTG、Zabbix 等等。RRDtool 还提出了数据类型的概念，支持 GAUGE、COUNTER、DERIVE、DCOUNTER、DDERIVE、ABSOLUTE 等多种数据类型。

Prometheus 生态也支持数据类型，分为 Gauge、Counter、Histogram、Summary 4 种，下面我们简单了解一下 Prometheus 的这 4 种类型。

Gauge

测量值类型，可大可小，可正可负。这种类型的数据，我们通常关注的是**当前值**，比如房间里的人数、队列积压的消息数、今年公司的收入和净利润。

Counter

表示**单调递增的值**，比如操作系统自启动以来网卡接收到的所有流量包的数量。每接收到一个包，操作系统就会加 1，所以这个值是持续递增的。但是操作系统可能会重启，导致这个值出现重置，比如第一次是从 0 一直涨到了 239423423，然后机器重启，新采集的数据是一些比

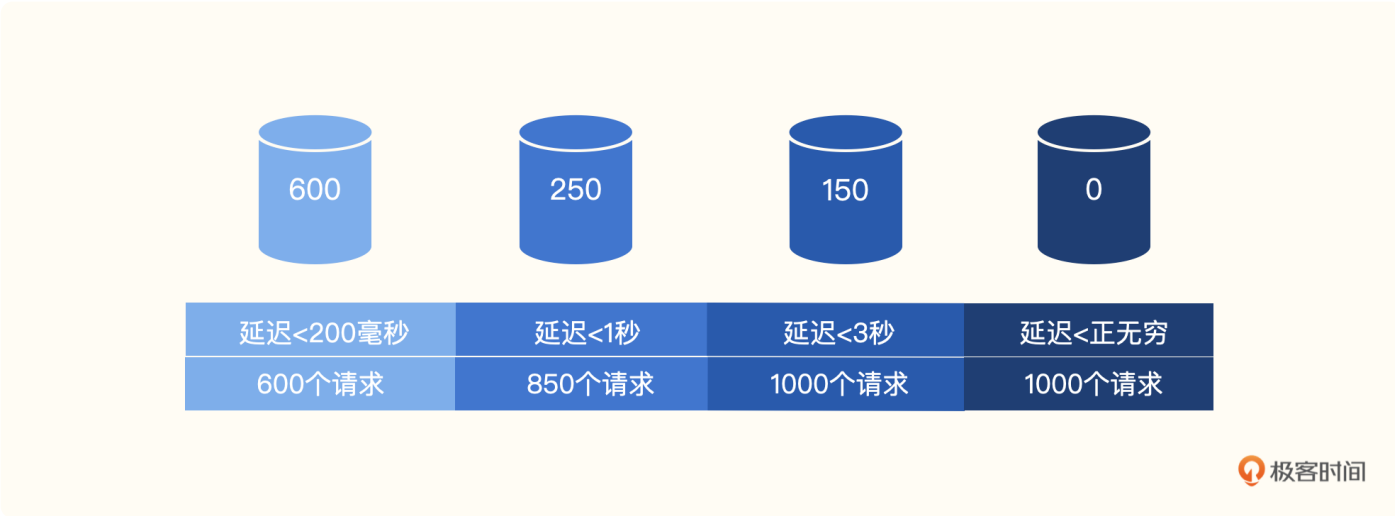
239423423 小很多的值，这种情况怎么办？此时 Prometheus 看到值没有递增，就能感知到重置的情况，会把新采集的值加上 239423423 再做计算。

Histogram

直方图类型，用于**描述数据分布**，最典型的应用场景就是监控延迟数据，计算 90 分位、99 分位的值。所谓的分位值，就是把一批数据从小到大排序，然后取 X% 位置的数据，90 分位就是指样本数据第 90% 位置的值。

有些服务访问量很高，每秒几百万次，如果要把所有请求的延迟数据全部拿到，排序之后再计算分位值，这个代价就太高了。使用 Histogram 类型，可以用较小的代价计算一个大概值。当然，不是特别准确，但是在监控场景足够用了，监控毕竟只是一个采样的系统，对数据准确性要求没有那么多高。

Histogram 的做法是根据数据的 value 范围，规划多个桶（bucket），把样本数据点放入不同的桶来统计。比如我们有个服务 service1，它的接口延迟最小的通常在一两百毫秒，最大的通常在 1 秒，如果超过 3 秒，大概率就是系统不正常了。此时我们可以规划 4 个桶，假设有 1000 个请求，我们来看下各个桶和对应的样本统计值。



延迟小于 200 毫秒的，有 600 个请求落到了这个区间。

延迟小于 1 秒的，有 850 个请求落到了这个区间，其中大于 200 毫秒的有 250 个请求。

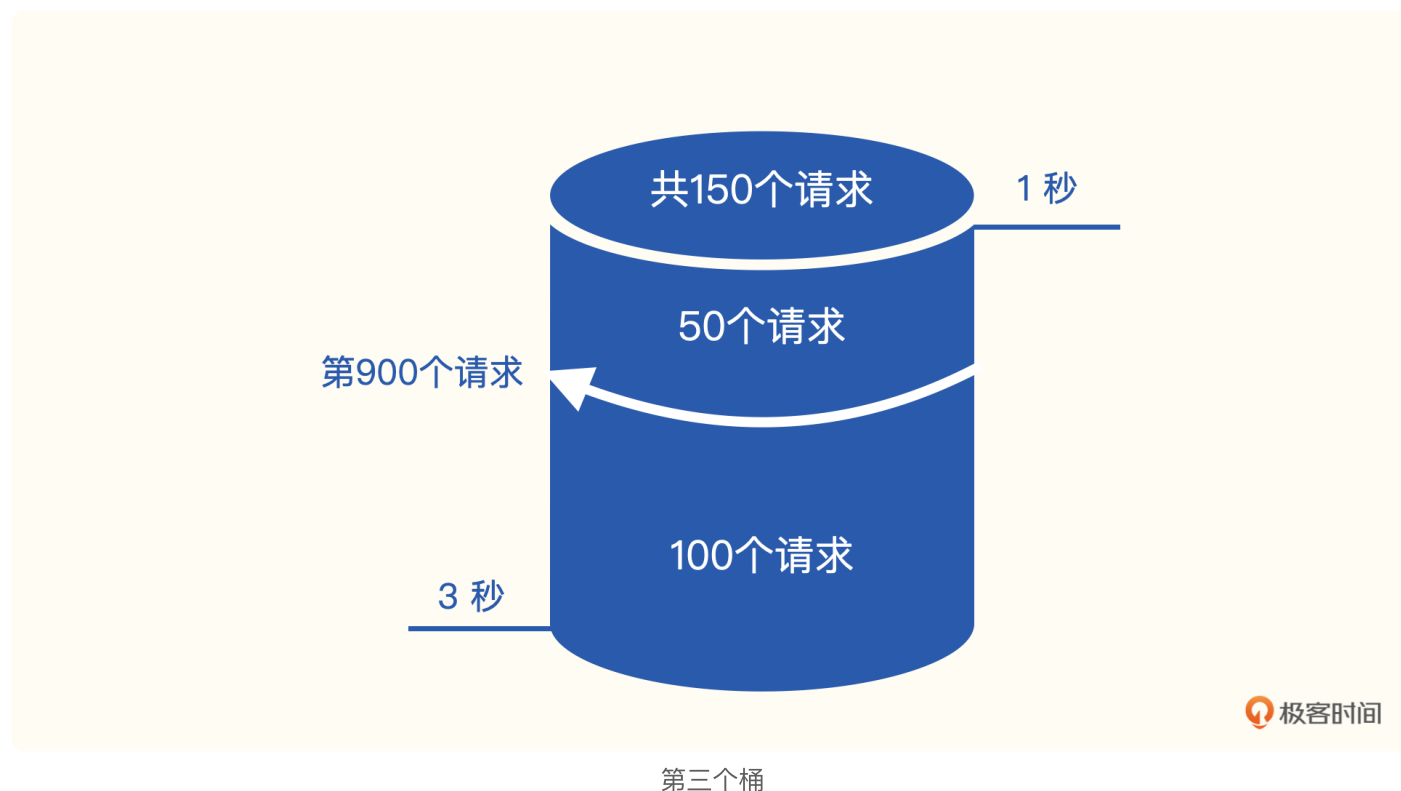
延迟小于 3 秒的，有 1000 个请求落到了这个区间，其中大于 1 秒的有 150 个请求。

延迟小于正无穷的，也就是总量，有 1000 个请求落到了这个区间，说明没有大于 3 秒的请求。

现在我们来计算 90 分位值，也就是第 900 个请求，说明这个值落到了第 3 个桶，延迟小于 3 秒的桶，于是我们可以得出结论，90 分位的延迟大小是 1~3 秒之间。

虽然知道了区间范围，但是我们还无法得出具体的值。为了计算出具体的值，Prometheus 有个假设，它认为各个桶里的样本数据是均匀分布的，即第三个桶的这 150 个请求，延迟最小的请求恰好延迟了 1 秒，延迟最大的恰好延迟了 3 秒，总量的第 900 个请求，是这个桶里的第 50 个请求，最终 90 分位的延迟数据计算方法是：

$$(3 - 1) \times (50 \div 150) + 1 = 1.67\text{秒}$$



这就是 histogram_quantile 的计算方法。histogram_quantile 是 Prometheus 的一个函数，这里为了说明 Histogram 类型，提前给你讲一下这个函数的计算逻辑。

虽然 Histogram 这种做法相比于把所有请求延迟数据都存储起来再计算延迟，性能有了巨大的提升，但是要同时计算成千上万个接口的分位值延迟数据，还是非常耗费资源的，甚至会造成服务端 OOM。于是就有了 Summary 类型。

Summary

Summary 这种类型是**在客户端计算分位值**，然后把计算之后的结果推给服务端存储，展示的时候直接查询即可，不需要做很重的计算，性能大幅提升。

听起来不错，但是有个问题，Summary 的计算是在客户端计算的，也就意味着不是全局的分位值，比如某个服务 service1，部署在两个机器上，服务代码里通过内嵌 Prometheus 的 SDK 做了埋点监控，SDK 里会计算 Summary 数据。也就是说，分位值延迟数据是进程粒度的，不是整个服务粒度的。

这个问题很严重吗？其实也没什么大不了的，这两个机器前面肯定有负载均衡，负载均衡会保证把请求均匀地打给后端的两个实例。一个实例内部计算的分位值，理论上和全局计算的分位值差别不会很大。另外，如果某个实例有故障，比如硬盘问题，导致落在这个实例的请求延迟大增，我们在实例粒度计算的延迟数据反而更容易发现问题。

到这里，我们就把 Prometheus 的四种数据类型介绍完了，现在请你闭上眼睛回顾一下这 4 种类型，再往上翻一下 Prometheus 的数据传输结构，就是那个 TimeSeries 的 Proto 结构，看看能否发现一些问题？

类型扩展知识

TimeSeries 数据结构中没有包含类型信息！惊不惊喜？意不意外？我现在采集了一些监控数据，传给 Prometheus（比如通过 remote write 协议），我的数据是分多种类型的，传输的时候却没有办法告诉 Prometheus 这些数据的类型是什么，难道它不需要知道我的数据类型吗？

其实从存储角度还真的不需要知道，存储的时候只要知道指标标识、时间戳、值，就足够了。后续做 PromQL 查询计算的时候，不同的函数有不同的行为，比如 rate、increase 函数，我们就给它传入 Counter 类型的数据作为参数即可。对于 histogram_quantile 函数，就传入带

有 le 标签的 bucket 指标。悄悄告诉你，其实你给 rate 函数传入一个 Gauge 类型的指标，也照样可以拿到值，虽然这个值没有合乎情理的业务语义。

那为什么还需要划分这么多类型呢？最主要的作用是在采集侧埋点的时候，SDK 会根据数据类型做不同的计算逻辑，比如 Histogram 类型，每次请求进来的时候，代码里调用一下 SDK 的 Observe 方法通知 SDK，SDK 就会自动计算生成多个指标，提升埋点便利性。

了解了监控系统中指标的采集和传输之后，接下来就是如何存储这些数据了。监控数据是周期性采集的，每条数据都关联一个时间戳，所以都是时序数据，使用时序库存储，下面我们就来看看时序库的概念。

时序库

时序库（Time series database）是一种专门处理时序数据的数据库。我们常见的数据库中，MySQL 是关系型数据库，Redis 是 KV 数据库，MongoDB 是文档数据库，而 InfluxDB、VictoriaMetrics、M3DB 等都是时序库，Prometheus 其实也内置实现了一个时序存储模块。

那什么是**时序数据**呢？时序数据最大的特点是每一条数据都带有时间戳，通常是单调顺序，不会乱序，流式发给服务端，通常不会修改，比如指标数据和日志数据，都是典型的时序数据。存储领域没有银弹，不同的数据场景侧重点不同，所以针对时序数据这个特定场景，产生了时序库这个专门的细分领域。在 DB-Engines 网站上，有一个 [🔗时序库的流行度排序](#)，你可以了解到当下哪些时序库比较流行。

之前我们提到监控系统有两个核心能力，一个是监控，一个是告警。告警部分也有一些关键概念，比如告警收敛、告警闭环，下面我们一起来看一下。

告警收敛

基础设施层面的故障，比如基础网络问题，可能会瞬间产生很多告警事件，形成告警风暴，导致接收告警的媒介拥塞，比如手机不停接收到短信和电话呼入，没办法使用。这个时候，我们就要想办法**让告警事件变少**，用的方法就是告警收敛。

最典型的手段是告警聚合发送，聚合可以采用不同的维度，比如时间维度、策略维度、监控对象维度等等。如果 100 台机器同时报失联，就可以合并成一条告警通知，减少打扰。

另外一个典型的收敛手段是把多个事件聚合成告警，把多个告警聚合成故障。比如某个机器的 CPU 利用率告警，监控系统可能每分钟都会产生一条事件，这多个事件的告警规则、监控对象、监控指标都相同，所以可以收敛为一条告警。假设有 100 台机器都告警了，其中 50 台属于业务 A，另外 50 台属于业务 B，我们可以按照业务来做聚合，聚合之后产生两个故障，这样就可以起到很好的收敛效果。

告警闭环

闭环这个词是个互联网黑话，表示某个事情有始有终，告警怎么判断是否闭环了呢？问题最终被解决，告警恢复，就算是闭环了。产品怎么设计才能保证告警闭环呢？通常来讲，没人响应的告警能够升级通知，告警 oncall 人员可以认领告警，基本就有比较好的保障了。

小结

这一讲我们主要讲解一些监控领域的关键概念，这些基础知识是整个监控体系知识蓝图的根基，一定要掌握。为了让你记忆更加深刻，下面我对这一讲的内容做一个简单总结。

监控：这个词在不同的上下文会有不同的语义，有的时候表示数据采集和可视化，有的时候表示整个监控系统。不过不管怎么理解，通常都不影响交流。

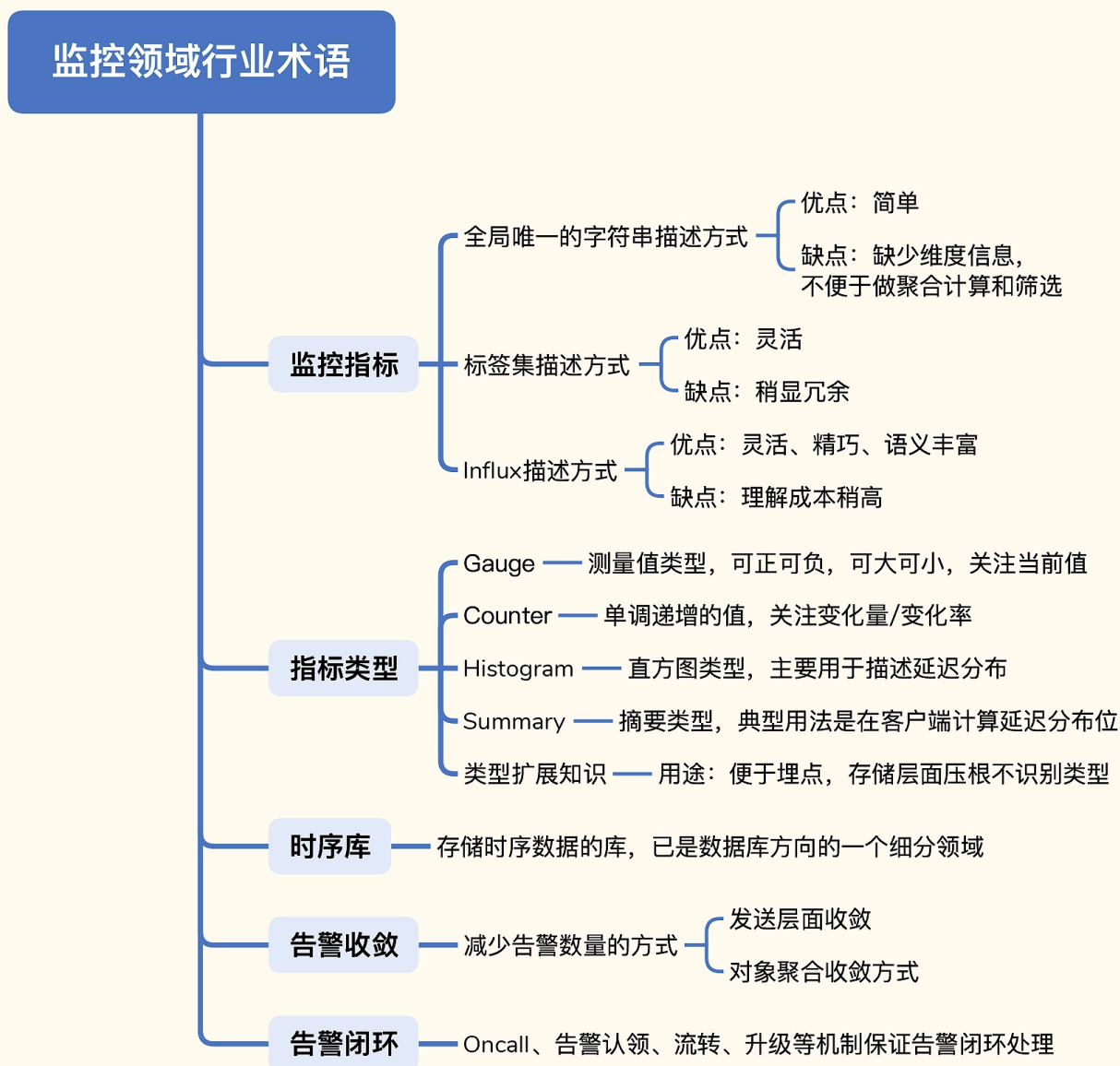
监控指标：这个概念很关键，不同的监控产品有不同的描述方式，不过随着 OpenMetrics 标准的建立，指标描述方式会渐渐趋于一致。重点要了解 Prometheus 的指标描述方式 metric + labels，当然 metric 也可以看作一个特殊的 label。Influx 格式也很重要，建议你掌握，如果使用 Telegraf 作为采集器，就绕不过去这个格式。

指标类型：针对时下流行的 Prometheus，我们讲解了 4 种指标类型及每个类型的适用场景，最后明确了指标类型最核心的作用：在采集侧埋点时，SDK 会根据数据类型做不同的计算逻辑。

时序库：存储时序序列数据的数据库，它已经成为了一个单独的数据库细分方向，而且随着 IoT 的场景越来越多，以及微服务的发展，时序库这个话题也越来越流行。

告警收敛和告警闭环：告警事件层面的话题是所有监控系统都需要处理的。当然也可以作为一个专门的产品和多种监控系统对接，专注处理告警事件，希望国内能有超越 Bigpanda、Pagerduty 的产品出现。

最后我把这节课的重点总结成了一张脑图，你可以多看几遍，加深记忆。你也可以自己试着总结一下这节课的重点，画一画图。



这一讲我们聊了很多监控方向的黑话，现在我们来互动一下，你能否用一句话或者一个词，来证明你是圈内人士，让我们一起来看看有多少同道中人。也欢迎你把今天的内容分享给你身边的朋友，邀他一起学习。我们下一讲再见！

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (9)



蓝熊船长

2023-01-11 来自广东

来催个更。正在负责新项目的监控这块。补齐一下地图。期待



不是苏苏

2023-01-11 来自北京

太强了😂 看完清晰多了



怀朔

2023-01-11 来自浙江

这是一个误报



hshopeful

2023-01-11 来自湖北

关于监控，脑海中有几句话想跟大家分享下：

- 1、当 facebook 还是 facebook 的时候，它开源的 gorilla 时序数据压缩算法特别适合监控领域的数据存储，prometheus 中已经采用
- 2、对监控指标进行流式聚合计算的时候，数据准确性和时效性需要做出 tradeoff
- 3、当你的监控告警指标太多的时候，相当于没有指标，一定要筛选出核心待关注的监控告警指标
- 4、怎么对监控系统本身进行监控





kaizen

2023-01-11 来自北京

influx line protocol 理解成本高?

共 1 条评论 >



无名无姓

2023-01-11 来自北京

针对prom每个类型展现一个实例比较好



DBRE

2023-01-11 来自北京

老师能根据Summary给个示例吗? 没太理解

共 1 条评论 >



臭猫

2023-01-11 来自广东

warning:xx节点cpu使用率超过80%，持续时间超过5分钟



呵呵

2023-01-11 来自山东

指标格式对比没太看懂，是想表达influx的全是kv，opentsdb的不全是。所以同一时间的消息，influx能少传几条，省了n个时间戳？怎么就标签重复的低了？不还是kv吗？

共 2 条评论 >

