

05 | Prometheus中有哪些关键设计？

2023-01-18 秦晓辉 来自北京



天下无鱼

<https://shikey.com/>

[课程介绍 >](#)

《运维监控系统实战笔记》



讲述：秦晓辉

时长 15:20 大小 14.01M



你好，我是秦晓辉。

上一讲我们介绍了如何搭建 Prometheus 系统，演示了基本的使用方法，这一讲我们深入进去，梳理一下 Prometheus 的关键设计，看看这些设计是如何奠定 Prometheus 江湖地位的。

标准先行，注重生态

推拉模式

动态发现机制

基于配置文件做规则管理

灵活的查询语言

标准先行，注重生态

Prometheus 最重要的规范就是**指标命名方式**，数据格式简单易读，在 [第 2 讲](#) 中我们已经聊过了，它用标签集来标识指标。有些监控系统会把一些特殊的字段单独提出来，最典型的就是 `hostname` 字段，这种做法在一些特定场景会显得更有效。但是显然，**统一的标签集表达方式是最通用、最灵活的**。

虽然标签集很灵活，但是在实际落地时，我强烈建议你在公司推行一个标签定义规范，标签 Key 不能随便起名，该有的标签也不能缺失。既减少了理解成本，也保证了数据的规整完备，便于后续做数据分析。比如，对于应用层面的监控，可以要求必须具备这几个信息。

- 指标名称 `metric`

Prometheus 内置建立的规范就是叫 `metric`（即 `__name__`）。如果是 `Counter` 类型，单调递增的值，指标名称以 `_total` 结尾。

- 服务名称 `service`

服务名称 `service` 要全局唯一，比如 `n9e-webapi`，`p8s-alertmanager`，一般是系统名称加上模块名称，组成最终的服务名称。如果公司比较大，就需要一个全局的服务目录做参考，否则不同的团队可能会起相同的名称，我们可以考虑使用 Git 里的 `GroupName + RepoName`。系统名称最好也单独做成一个标签，比如 `system=n9e` `system=p8s`。

- 实例名称 instance

一个服务一般会部署多个实例，可以直接使用机器名或 Pod 名作为 instance 名称。如果在物理机部署，有实例混部的情况，就要把端口加上，比如实例一是 10.1.2.3:3306，实例二是 10.1.2.3:3307。

- 服务类型 job

比如所有的 MySQL 的监控数据，都统一打上 job=mysql 的标签，Redis 的监控数据，就打上 job=redis 的标签。如果是自研的模块，也可以使用 webserver backend frontend 这种分类方式。

- 地域可用区 zone

把地域信息放到标签里，有个巨大的好处，比如某个 zone 出问题了，就比较容易看出来，带有某个特定的 zone 的指标数据异常，快速执行切流止损即可。有了 zone 的信息，region 就可有可无了，zone 的前缀一般就是 region。

- 集群名称 cluster

有的时候一个可用区会部署多个集群，特别是一些中间件，比如 ElasticSearch，给每个重要的业务单独部署一个集群，一个大公司可能有几百套 ElasticSearch 集群，几千套 ZooKeeper 集群。

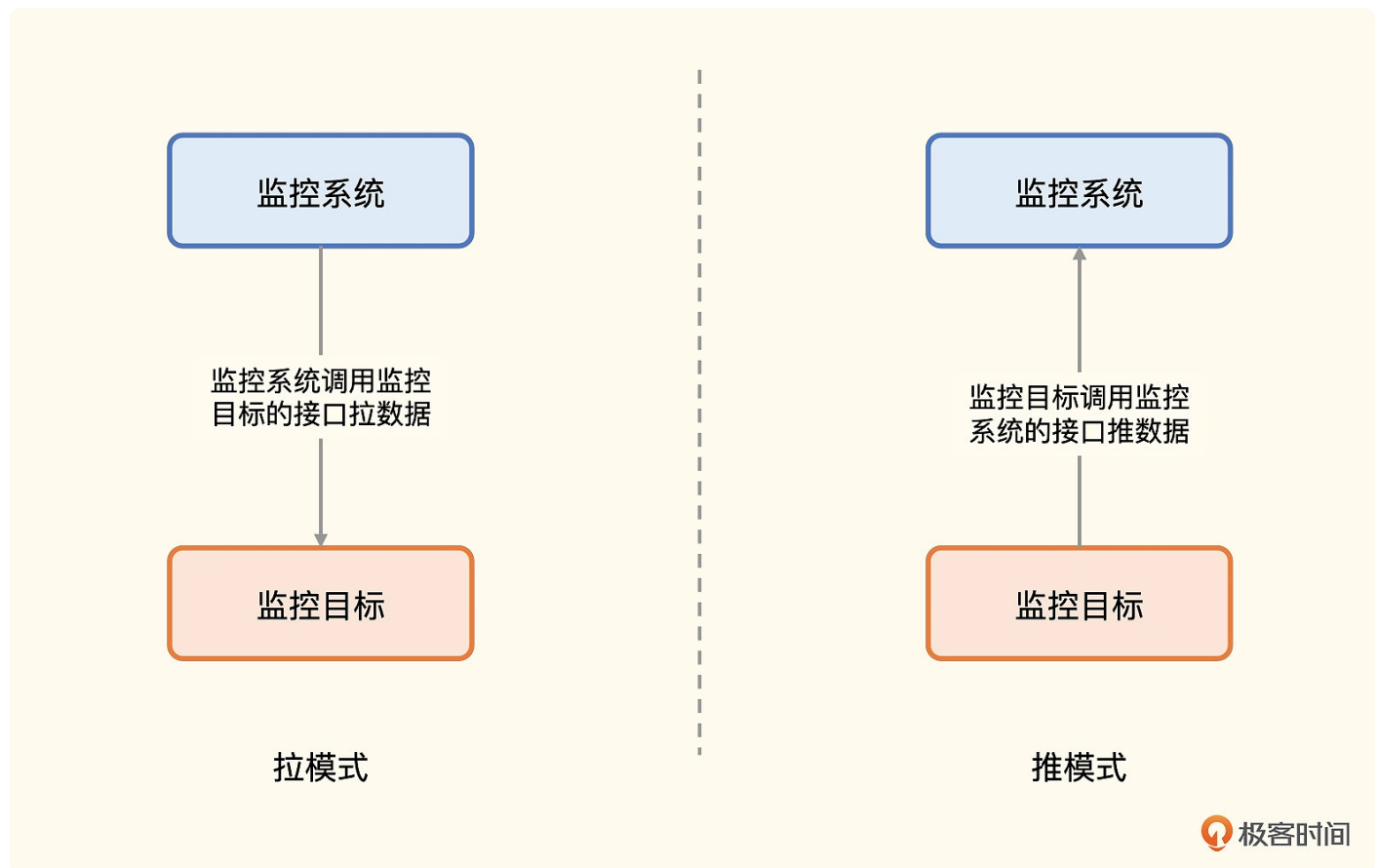
- 环境类型 env

环境类型 env 用来标识是生产环境还是测试环境。当然了，如果监控系统不复用（推荐这么做），生产用生产的监控系统，测试用测试的监控系统，就无需这个标签了。

指标的数据格式和传输协议制定好之后，各种 Exporter、各种支持 Remote Read/Write 的后端存储就可以接入进来了，而这些 Exporter、存储的丰富和繁荣，又反向推动了 Prometheus 的流行，形成正向循环。

主要使用拉模式，解耦

Prometheus 主要使用拉模式获取指标，辅以推模式（Pushgateway 的职能）。很多监控系统都是推模式，比如 Datadog、Open-Falcon、Telegraf+InfluxDB 组合。推拉两种方式，在监控领域讨论也比较多，它们各有优缺点和适用场景。



拉模式有个最重要的优势，就是解耦。你可能之前听很多人说过这个优势，但你未必能体会到解耦解在了哪里，我举一个例子你就懂了。

对于各类中间件，特别是非常基础的那些，很大概率是先于监控系统部署的。如果是拉模式，部署好监控系统之后，再来调用中间件的接口获取数据即可。如果是推模式，就需要在中间件里重新配置监控数据上报地址，然后重启中间件，这个代价就太高了。

但是拉模式需要有很好的服务发现机制，如果只有少量的几个目标要采集，怎么搞都可以，但是有几百个的时候，手工配置就比较麻烦了。所以 Prometheus 支持各种服务发现机制，尤其是基于 Kubernetes 的服务发现机制，是最常见的。

如果服务没有部署在 Kubernetes 中，而是部署在传统物理机或虚拟机上，这个时候就需要使用 Consul 之类的服务发现机制。但如果在监控体系建设之前，服务没有接入注册中心，为了满足监控需求而接入注册中心，用户会觉得成本太高。此时推模式就有了用武之地，这就是很多公司的自研服务都使用推模式发送监控数据的原因。

所以结论就来了：中间件类使用拉模式，自研的服务使用推模式，自研的服务如果都接入了注册中心，则也可以使用拉模式。



当然，推拉的选择还有一个点比较关键，就是**网络通路问题**，特别是网络 ACL 限制比较严格的环境，很多都是可出不可进，比如典型的 NAT 出网，这种情况下推模式的适配性更好，也就是说对 ACL 更友好一些。另一个关键点是**短周期任务或批处理任务**，通常不太可能监听 HTTP 端口，这种大概率也是推模式。

推拉模式的选择，还有一些其他影响因素，比如推模式服务端通常比较容易处理，因为数据接收是无状态的。但是拉模式在数据量大的时候要考虑分片的问题，还有就是失联告警问题，拉模式很容易感知到目标失联。推模式就比较复杂了，需要对数据缺失做告警，比如 Prometheus 的 `absent` 函数，`absent` 函数需要把指标的每个标签都写全，才能达到预期效果。而指标数量何止千万，几乎不可能完成。

最后就是可控性问题，拉模式，监控系统是主动的一方，可以控制频率；推模式，客户端是主动的一方，如果代码写“挫”了，就会给监控系统造成很大压力。

前面我们提到拉模式需要有很好的服务发现机制，Prometheus 采取的就是拉模式，因此它对监控目标动态发现机制的苛求度很高。

监控目标动态发现机制

监控目标动态发现机制，这个问题其实很早就有，老一些的监控系统，比如 Zabbix，是资产管理式，要监控的目标都需要在服务端注册、配置。这种方式在监控目标偏静态的场景还是比较好用的，但是云原生之后，基础设施动态化，监控目标的创建、销毁都比较频繁，就需要有一个更自动化的机制来获取监控目标列表。

Prometheus 内置了多种服务发现机制，最常见的有四种。

- **基于配置文件的发现机制**：这种方式看起来很低端，其实非常常用，因为可以配合配置管理工具一起使用，非常方便。使用配置管理工具批量更新配置，然后让监控系统重新加载一下就可以了，比较丝滑。
- **基于 Kubernetes 的发现机制**：Kubernetes 中有很多元信息，通过调用 kube-apiserver，可以轻易拿到 Pod、Node、Endpoint 等列表，Prometheus 内置支持了 Kubernetes 的服务发现机制，让这个过程变得更简单，Prometheus 基本成为了 Kubernetes 监控的标配。

- **基于公有云 API 的发现机制：**比如要监控公有云上所有的 RDS 服务，一条一条配置比较麻烦，这个时候就可以基于公有云的 OpenAPI 做一个服务发现机制，自动拉取相关账号下所有 RDS 实例列表，大幅降低管理成本。
- **基于注册中心的发现机制：**社区里最为常用的是 Consul，典型场景是 PING 监控和 HTTP 监控，把所有目标注册到 Consul 中，然后读取 Consul 生成监控对象列表即可。



这就是 Prometheus 支持的几个主要的服务发现机制，当然还有其他方式，但用得没那么多，这里就先不介绍了。

基于配置文件的管理方式

Prometheus 的告警规则管理、记录规则管理、抓取配置管理与发送策略管理，全部是基于配置文件的，这虽然不是一个关键设计，但确实是一个非常有特色的设计，我们也简单聊一下。

这个方式有两个好处，一个是简单，简单到令人发指，很多监控系统都是使用数据库来存储各类配置的，Prometheus 则直接使用 Yaml 文件，非常直观。第二个好处就是便于自动化，配合配置管理工具、Git、Kubernetes 等，与 Infrastructure as Code 的管理风潮非常契合。

当然，这样管理也有一个问题，就是不便于公司级协作，比如公司有 30 条业务线，数百个服务，上千个研发，大家都来管理一套配置，会非常混乱。所以很多公司的做法是由一个专职的运维团队来管理这套配置，其他业务线研发有需求了就给这个运维团队提工单，这种做法也凑合能用，只是苦了这个运维团队，团队成员比较容易产生焦躁情绪。

另外，Prometheus 默认是单机时序存储，容量有上限，基于配置管理的问题和容量问题，我个人非常建议那些推行 DevOps 的团队来使用，而且是每个团队自己有一套单独的 Prometheus，互不干扰，正所谓“You build it. You Run it. You monitor it. You own it.”

当然，这样也会带来问题，最典型的是数据孤岛问题，不过我们可以把各个 Prometheus 中的核心关键指标抽取到一个统一的地方来呈现，比如使用 Prometheus 联邦机制，**只共享核心指标**，其余指标不需要抽取到中心，自己团队消化就好。

当然，后面第 8 讲我们也会使用 Nightingale 来增强 Prometheus 的配置管理问题，算是另一个思路。

灵活的查询语言

PromQL (Prometheus Query Language) 是 Prometheus 的查询语言, 非常灵活。这也是 Prometheus 的一个关键设计。



很多老一代监控系统都只能对数据做简单过滤判断阈值, 没有 QL 的支持。当我们想要某个数据却发现没有的时候, 就天然倾向于在采集侧处理, 但是**采集侧是无法穷举所有计算场景的, 采集侧应该采集原始数据, 后续的二次计算还是应该放到中心来搞定。**

比如机器的内存指标, 我们可以从 `cat /proc/meminfo` 中看到很多内存相关的监控指标, 采集器可以轻易拿到 `MemAvailable` 和 `MemTotal` 这样的字段, 但是操作系统不会直接暴露内存可用率, 此时就需要使用 PromQL: `MemAvailable / MemTotal * 100` 做二次计算了。因为这个场景很常见, 有的采集器就直接自动做了计算, 输出 `mem_available_percent` 这样的指标。但是还有很多场景是不好穷举的, 下面我们再举个例子。

有一些监控数据的采集, 是完全由用户配置出来的, 比如 SNMP 数据采集, 采集哪些 OID 是用户配置的; JMX 数据采集, 采集哪些 MBean、哪些 Pattern 也是用户配置的。监控采集器压根就不知道这些数据的具体语义, 只有配置采集规则的人知道, 这种情况更不可能自动计算, 采集器只能采集原始数据, 如果有二次计算的需求, 最好是设计到服务端, 让服务端来做。

PromQL 为二次计算提供了能力支持, 多个指标的关联计算、多条件联合告警, 都可以用 PromQL 来实现, 作为现代监控系统, Query Language 已经是必备要求了。

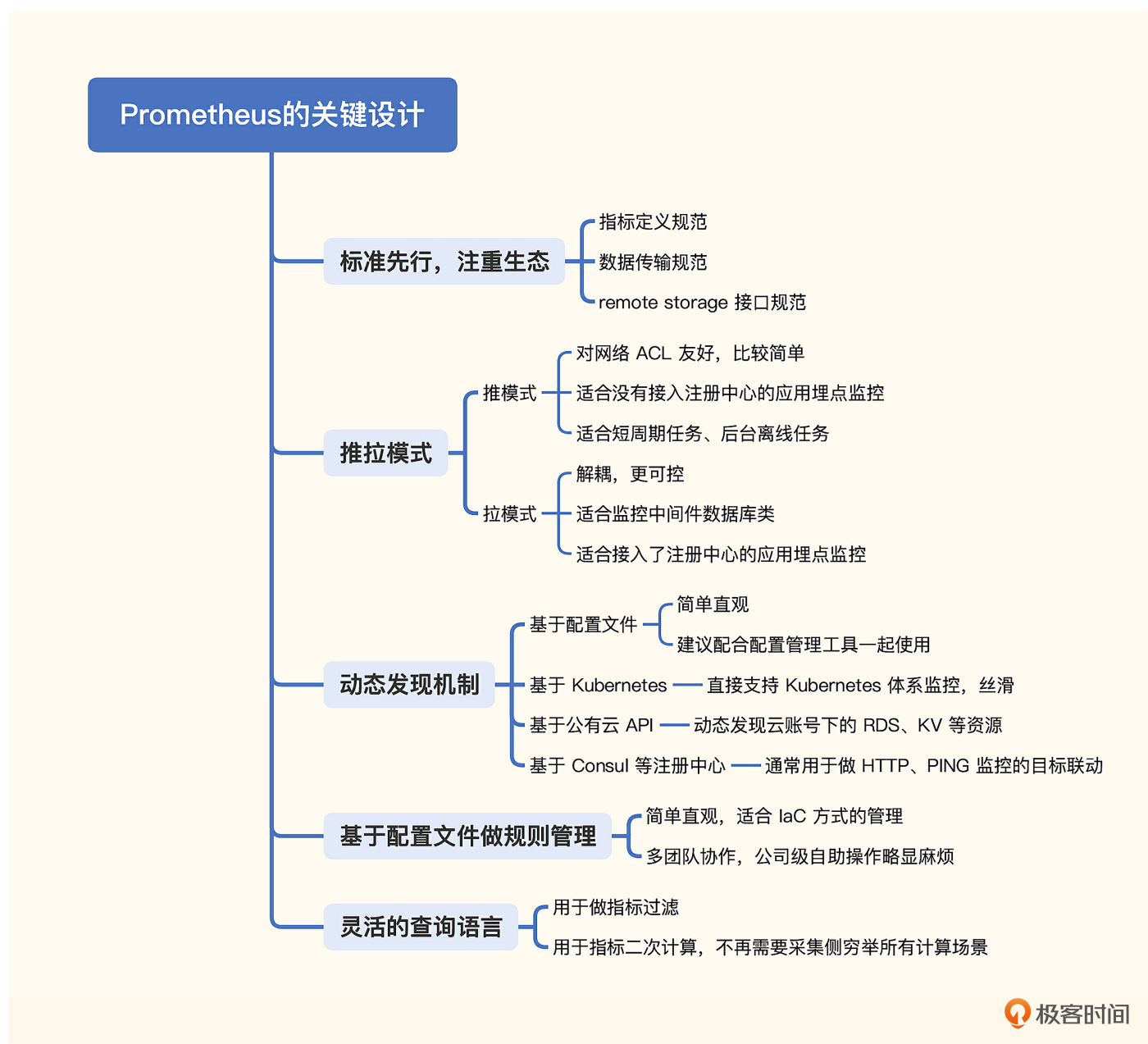
小结

这一讲, 我们介绍了一些 Prometheus 的关键设计, 这些设计成就了 Prometheus。下面我们对这一讲的内容做一个简单的回顾。

- Prometheus 非常注重标准制定和生态建设, 而且标准很稳定, 没有变来变去, 社区有很多人参与其中, 共建整套生态。Prometheus 不期望解决所有问题, 但是影响力巨大, 就是因为标准和生态的强大。
- Prometheus 主要使用拉模式, 辅以推模式。我们比较了推拉两种模式, 简单来看, 拉模式便于解耦, 推模式则更简单。中间件监控适合拉模式, 自研模块适合推模式。当然, 如果注册中心完备, 服务都接入了注册中心, 服务监控的数据采集也可以使用拉模式。

- 拉模式更需要监控目标动态发现机制，主要使用拉模型的 Prometheus 内置了多种发现机制，最常用的是基于配置文件、Kubernetes、公有云 API、Consul 四种发现机制。
- Prometheus 的配置管理非常简单，直接使用 Yaml 文件，很直观，便于推行 IaC 管理模式，只是在公司级大规模协同的时候会有些不方便。当然，国内的大部分企业没有践行 IaC，Yaml 配置的方式会相对难搞一些。
- Prometheus 查询语言就是 PromQL，这也是它最后一个关键设计，让采集侧专注采集，服务端提供灵活的计算能力，有些偏传统的企业难以接受这种做法，只想简单过滤指标来配置阈值，很难把 PromQL 的优势发挥出来，非常可惜。

最后，我整理了一张脑图，帮助你理解和记忆。




这一讲我们介绍了 **Prometheus** 的多个关键设计，弄懂这些内容非常重要，每个产品都有自己擅长的领域，我们要取各家所长为我所用。听我说了这么多，最后我也想听听你的看法和见解，你还知道 **Prometheus** 的哪些关键设计，哪些设计让你拍案叫绝或让你非常难受？

天下无鱼
<https://shikey.com/>

欢迎你留言分享，也欢迎你把今天的内容分享给身边的朋友，邀他一起学习。我们下一讲再见！

分享给需要的人，Ta购买本课程，你将得 **18** 元

 生成海报并分享

 赞 7  提建议

© 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。

上一篇 04 | 如何快速搭建Prometheus系统？

下一篇 06 | PromQL有哪些常见的使用场景？

精选留言 (10)

 写留言



叶夏

2023-01-19 来自江苏

可以把Prometheus的配置yaml和告警规则yaml单独放到一个代码仓库中，把这个仓库向有需要的人开放，如果他们想要添加某一个规则，就自己修改提交PR，提交PR的部署到测试环境，验证没有问题，prometheus owner 审核PR之后合并到master分支中，这个时候在自动部署到生产环境，我们是这么做的，感觉蛮好的

作者回复：

共 2 条评论 >

 10



Ishmael

2023-01-19 来自四川

我觉得最绝的还是限制值必须为float这点，限制死了之后保证了数据的高压缩度，进一步保证了计算性能和存储空间的节省。同时wal和分级存储也很绝，kv设计的思路也很不错。tag这个

设计在实际接入中帮了大忙。

作者回复:



天下无鱼

<https://shikey.com/>



👍 2



那一刻

2023-01-18 来自北京

Prometheus 数据的存储按冷热数据进行分离，最近的数据肯定是看的最多的，所以缓存在内存里面，为了防止宕机而导致数据丢失因而引入 wal 来做故障恢复。数据超过一定量之后会从内存里面剥离出来以 chunk 的形式存放在磁盘上这就是 head chunk。对于更早的数据会进行压缩持久化变成 block 存放到磁盘中。

对于 block 中的数据由于是不会变的，数据较为固定，所以每个 block 通过 index 来索引其中的数据，并且为了加快数据的查询引用倒排索引，便于快速定位到对应的 chunk。

作者回复:



👍 2



无名无姓

2023-01-18 来自北京

推和拉，有很大区别



👍 1



胡飞

2023-01-29 来自上海

在k8s模式下部署，配置采用yaml缺点就出来了，一套环境大家都有弄，改错了，甚至缩进写错了都会造成pod重启失败。后面采用servicemonitor/podmonitor 感觉就好多了

作者回复:



👍 1



Geek_a99361

2023-01-29 来自北京

Prometheus使用本地存储的方式有一些局限性，远端存储有什么推荐？比如 clickhouse或者influxDB

作者回复: 后面的章节会介绍如何扩展Prometheus的存储，个人推荐VictoriaMetrics



irving

2023-01-29 来自上海



天下无鱼

<https://shikey.com/>

telegraf是拉模式吗？telegraf中的各个plugin都要配置influxdb的地址，应该是push模式吧

作者回复: 是推模式，不是各个plugin都配置influxdb的地址，所有input的plugin都无需配置influxdb地址，只要配置一个output的plugin写入后端时序库就可以了



hshopeful

2023-01-19 来自湖北

基于gorilla的时序数据压缩算法很经典

作者回复:



peter

2023-01-18 来自北京

Q1: Prometheus是用什么开发的？JAVA吗？

Q2: Prometheus代码规模有多大？百万行代码？

作者回复: prometheus用go写的，我没做具体统计，整体代码量不大



Ecoder

2023-01-18 来自广东

Prometheus的数据模型和时序库也很经典

