最前沿開源監控 Prometheus專題講座

第五讲 Prometheus 监控数据格式学习

第五讲内容

- prometheus metrics的概念
- k/v的数据形式
- prometheus exporter的使用（pull形式采集数据）
- prometheus pushgateway的入门介绍（push形式采集数据）

promethes监控中 对于采集过来的数据 统一称为metrics数据

Metrics，我们听到的太多了，熟悉大数据系统的不可能没听说过metrics，当我们需要为某个系统某个服务做监控、做统计，就需要用到Metrics。

metrics是一种对采样数据的总称 （metrics 并不代表某一种具体的数据格式 是一种对于度量计算单位的抽象 ）

咱们来介绍一下 metrics 的几种主要的类型

Gauges

最简单的度量指标，只有一个简单的返回值，或者叫瞬时状态，例如，我们想衡量一个待处理队列中任务的个数

用更简单的方式举个例子

例如： 如果我要监控硬盘容量或者内存的使用量，那么就应该使用Gauges的metrics格式来度量

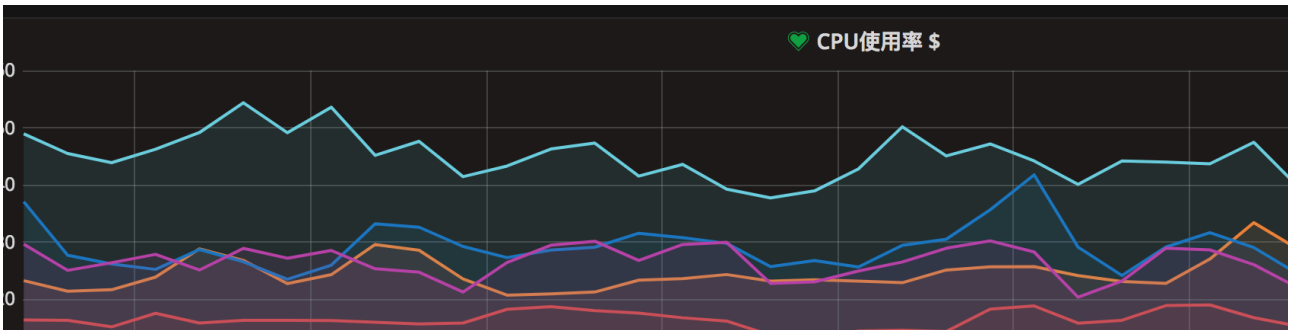
因为 硬盘的容量或者 内存的使用量 是随着时间的推移 不断的瞬时 没有规则 变化的

这种变化没有规律 当前是多少 采集回来的就是多少 5:00 21%。5:01 25% ， 5:02 17%

既不能肯定是 一只持续增长 也不能肯定是一直降低

是多少 就是多少 这种就是Gauges使用类型的代表

如图所示 CPU的上下浮动就是采集使用Gauge形式的 metrics数据 没有规律 是多少就得到多少 然后显示出来



Counters 类型metris

Counter 就是计数器，从数据量0开始累积计算 在理想状态下 只能是永远的增长 不会降低（一些特殊情况另说）

Counter => 数字 +++ 2:00 => 0 prometheus

举个例子来说

比如对用户访问量的采样数据

我们的产品被用户访问一次就是1 过了10分钟后 积累到 100

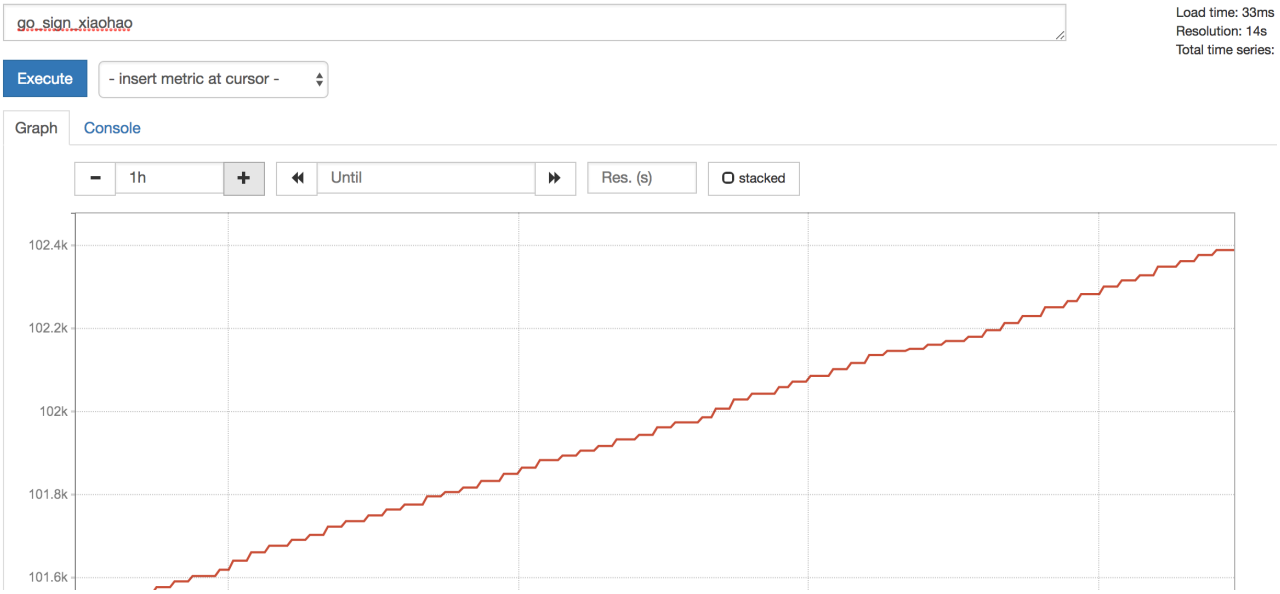
过一天后 积累到 20000

一周后 积累到 100000-150000

如下图展示的。Counter数据 是从0开始 一直不断的积累 累加下去的 所以理想状态下 不可能出现任何降低的状况

最多只可能是 一直保持不变（例如 用户不再访问了，那么当前累积的总访问量 会以一条水平线的状态保持下去 直到 再被访问）

如下图展示的 就是一个counter类型的metrics数据采集。 采集的是 用户的访问累积量



Histograms

Histogram统计数据的分布情况。比如最小值，最大值，中间值，还有中位数，75百分位, 90百分位, 95百分位, 98百分位, 99百分位, 和 99.9百分位的值(percentiles)。

这是一种特殊的 metrics数据类型， 代表的是一种
近似的百分比估算数值

这个metrics种类 是最难理解的一种（但是很实用）估计大多数学员看了上面这几行定义 头会很大
大米老师 用一种更简单的方式来给大家做解释一下 什么是 Histogram数据

比如我们在企业工作中 经常接触这种数据

Http_response_time HTTP响应时间

代表的是 一次用户HTTP请求 在系统传输和执行过程中 总共花费的时间

nginx中的 也会记录这一项数值 在日志中

那么问题来了

我们做一个假设

如果我们想通过监控的方式 抓取当天的nginx access_log， 并且想监控用户的访问时间
我们应该怎么做呢?

同学们肯定很容易想到

简单啊 把日志每行的 http_response_time 数值统统采集下来啊 然后计算一下总的平均值即可

那么大米要问大家一句了 假如我们采集到 今天一天的访问量 是100万次

然后把这100万次的 http_response_time 全都加一起 然后 除以100万 最后得出来一个avg值 0.05秒 = 50毫秒
这个数据的意义大么?

假如 今天中午1:00的时候 发生了一次线上故障 系统整体的访问变得非常缓慢 大部分的用户请求时间都达到了 0.5 ~ 1秒作用
但是这一段时间 只持续了5分钟， 总的一天的平均值并不能表现得出来 我们如何在1:00-1:05的时候 实现报警呢?

在举个例子:

就算我们一天下来 线上没有发生故障 大部分用户的响应时间 都在 0.05秒（通过 总时间/总次数得出）
但是我们不要忘了 任何系统中 都一定存在 慢请求 就是有一少部分的用户 请求时间会比总的平均值大很多 甚至接近 5秒 10秒的也有
（这种情况很普遍 因为各种因素 可能是软件本身的bug 也可能是系统的原因 更有可能是少部分用户的使用途径中出现了问题）

那么我们的监控需要 发现和报警 这种少部分的特殊状况，用总平均能获得吗?

如果采用总平均的方式，那不管发生什么特殊情况，因为大部分的用户响应都是正常的 你永远也发现不了少部分的问题

所以 Histogram的metrics类型 在这种时候就派上用场了

通过histogram类型（prometheus中 其实提供了一个 基于histogram算法的 函数 可以直接使用）
可以分别统计出 全部用户的响应时间中
~=0.05秒的 量有多少 0 ~ 0.05秒的有多少， > 2秒的有多少 >10秒的有多少 => 1%

我们就可以很清晰的看到 当前我们的系统中 处于基本正常状态的有多少百分比的用户（或者是请求）
多少处于速度极快的用户， 多少处于慢请求或者有问题的请求

metrics的类型其实还有另外的类型

但是在我们大米运维的课程中 我们最主要使用的 就是 counter ganga 和 histogram

2) k/v的数据形式

我们之前了解了metrics的概念，和 类型
prometheus 的数据类型 就是依赖于 这种 metris的类型来计算的

而对于采集回来的数据类型 再往细了说 必须要以一种具体的数据格式 供我们查看和使用

那么我们来看一下 一个exporter 给我们采集来的 服务器上的k/v形式 metrics数据

当一个exporter(node_exporter) 被安装和运行在 被监控的服务器上后

使用简单的 curl命令 就可以看到 exporer帮我们采集到 metrics数据的样子， 以 k / v 的形式展现和保存

curl localhost:9100/metrics

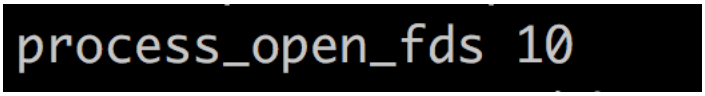
```
node_vmstat_unevictable_pgs_rescued 748
# HELP node_vmstat_unevictable_pgs_scanned /proc/vmstat information field unevictable_pgs_scanned
# TYPE node_vmstat_unevictable_pgs_scanned untyped
node_vmstat_unevictable_pgs_scanned 0
# HELP node_vmstat_unevictable_pgs_stranded /proc/vmstat information field unevictable_pgs_strand
# TYPE node_vmstat_unevictable_pgs_stranded untyped
node_vmstat_unevictable_pgs_stranded 0
# HELP node_vmstat_zone_reclaim_failed /proc/vmstat information field zone_reclaim_failed.
# TYPE node_vmstat_zone_reclaim_failed untyped
node_vmstat_zone_reclaim_failed 0
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 2369.74
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 65535
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 10
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 1.9734528e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.51600078926e+09
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 8.8592384e+08
```

如上图所示 curl之后的结果输出

prometheus_server

带# 的行 是注释行 用来解释下面这一项 k / v 数值 是什么东东的采样数据

而 我们真正关心的 是这样的 数据



看到了没有 就是用空格分开的 KEY / Value 数据

第一个代表的是 当前采集的 最大文件句柄数 是 65535
第二个代表的是 当前采集的 被打开的文件句柄数是： 10

这样就非常好理解了

另外 我们在看下这

里

```
process_max_fds 65535
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 10
```

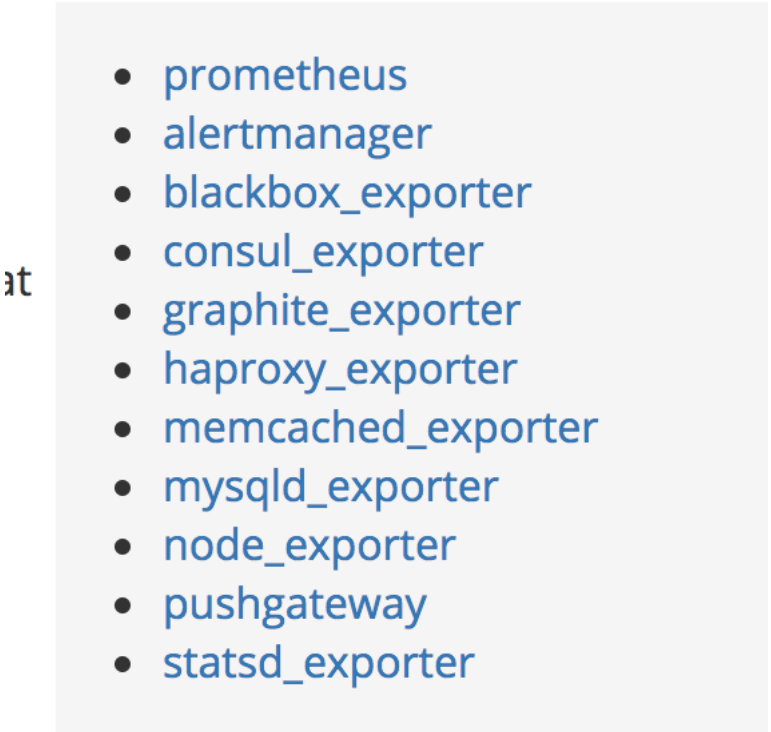
第二行的 # 告诉我们了 这一项数据的metrics类型 属于gauge

因为很简单， 文件句柄数的使用 是没有规律的瞬时采样数据 当前是多少就是多少

3） exporter的使用

官网提供了丰富的 成型 exportrs插件可以使用

举几个例子



下载首页 其实就已经提供了 很多 很常用的 exporters

这些exporters 分别使用不同的开发语言开发，有 go 有 Java 有python 有ruby 等等

我们不关心社区组织 用什么语言做的开发
我们只要关心 如何下载和正确安装使用 即可

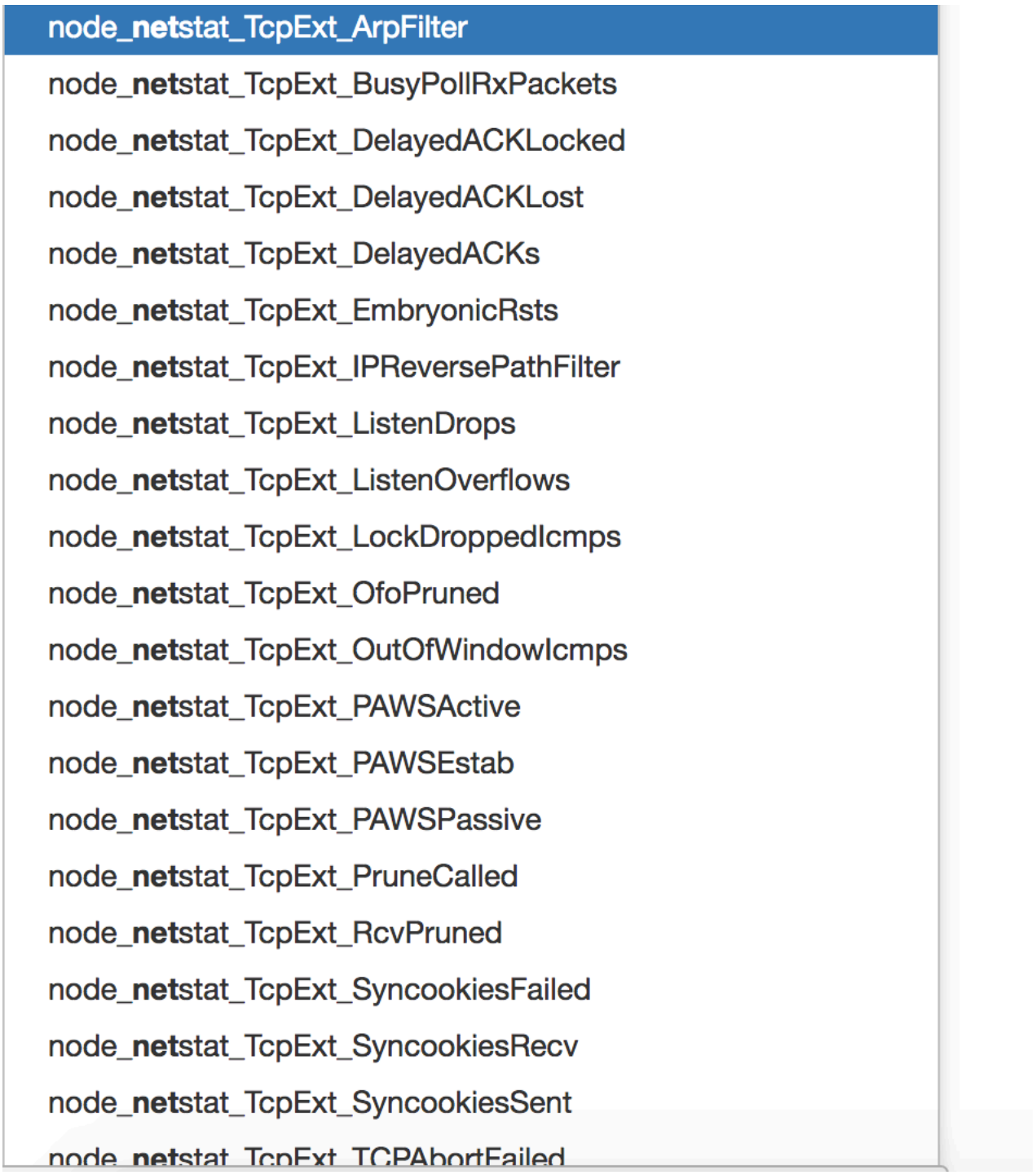
大多数exporters 下载之后，就提供了启动的命令 一般直接运行 带上一定的参数就可以了

比如 最常用的 node_exporter =》 这个exporter非常强大，几乎可以把 Linux系统中 和系统自身相关的监控数据 全抓出来了（很多参数 说真的 都没听说过 比你想象的 学过的 多的多）

这里只给大家一个截图 展示node_exporter一部分的 支持的监控数据采集

Enabled by default

Name	Description	OS
arp	Exposes ARP statistics from `/proc/net/arp`.	Linux
bcache	Exposes bcache statistics from `/sys/fs/bcache/`.	Linux
conntrack	Shows conntrack statistics (does nothing if no `/proc/sys/net/netfilter/` present).	Linux
cpu	Exposes CPU statistics	Darwin, Dragonfly, FreeBSD, Linux
diskstats	Exposes disk I/O statistics.	Darwin, Linux
edac	Exposes error detection and correction statistics.	Linux
entropy	Exposes available entropy.	Linux
exec	Exposes execution statistics.	Dragonfly, FreeBSD
filefd	Exposes file descriptor statistics from `/proc/sys/fs/file-nr`.	Linux
filesystem	Exposes filesystem statistics, such as disk space used.	Darwin, Dragonfly, FreeBSD, Linux, OpenBSD
hwmon	Expose hardware monitoring and sensor data from `/sys/class/hwmon/`.	Linux
infiniband	Exposes network statistics specific to InfiniBand and Intel OmniPath configurations.	Linux
	Exposes IPVS status from `/proc/net/ip_vs` and stats from	



每一项 其实还有N多的子项， 该有的数据都有了，不该有的 不重要的 基本也有了 应有尽有 咱们还需要 自己开发采集exporter吗？ 其实不怎么需要了 直接用就好了

4） pushgateway 的概念介绍

之前说的 exporter 是首先安装在 被监控服务器上 运行在后台

然后 自动采集系统数据， 本身又是一个 HTTP_server 可以被prometheus服务器 定时去HTTP GET取得数据
属于pull的形式

如果把这个过程反过来
push的形式是 把 pushgateway安装在 客户端或者服务端（其实装哪里都无俗谓）
pushgateway本身也是一个 http服务器

运维通过写自己的脚本程序 抓自己想要的监控数据 然后 推送到 pushgateway(HTTP) 再由pushgateway推送到 prometheus服务端

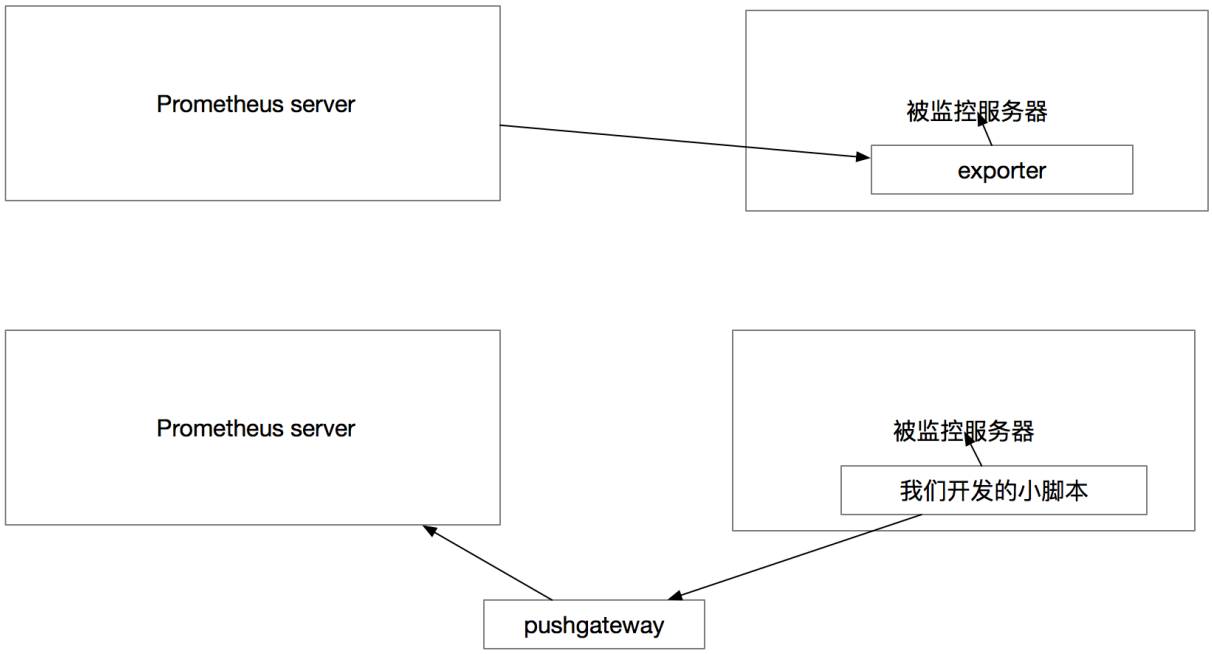
是一个反过来的被动模式

一个问题？为什么已经有了那么强大的 pull 形式的node_exporter采集
还需要一个pushgateway的形式呢？

其实对这个问题的回答是：

- exporter虽然采集类型已经很丰富了，但是我们依然需要很多自制的监控数据 非规则化的 自定制的
- exporter 由于数据类型采集量大 其实很多数据 或者说大部分数据 其实我们监控中 真的用不到 用pushgateway是定义一项数据 就采集着一种 节省资源
- 一个新的自定义的pushgateway脚本开发 远远比 开发一个全新的exporter 简单快速的多的多的多！（exporter的开发 需要使用真正的编程语言， shell这种快速脚本 是不行的 而且需要了解很多 prometheus自定的编程格式才能开始 制作 工作量很大）
- exporter虽然已经很丰富了，但是依然有很多的我们需要的采集形式，exporter无法提供，或者说 现有的expoter还不支持，但是如果使用pushgateway的形式 就可以任意灵活 想做什么都可以做到 而且极快

最后 用一张图 来立刻这两种不同的 采集形式


Prometheus server
被监控服务器
exporter
Prometheus server
被监控服务器
我们开发的小脚本
pushgateway