

1. 基本介绍

2. PromQL 查询结果的数据类型

3. 查询基础语法

3.1 时间序列选择器 Time series Selectors

3.2. 即时向量 (Instant vector) 选择器

3.3 范围向量选择器

3.3.1. 语法

3.3.2 查询结果解析

3.3.3. 时间长度

3.3.4. 偏移修饰符(不常用)

3.3.5. @ 修饰符(不常用)

3.4. 标签匹配器

3.4.1 语法介绍

3.4.2. 标签匹配运算符

3.5. 避免慢速查询和过载

3.6. 子查询

4. 查询语法中的操作运算符 Operators

4.1. 二进制运算符

4.1.1. 算术运算符

4.1.2. 比较二进制运算符:

4.1.3. 逻辑二进制运算符

4.1.4. 二进制运算符优先级

4.2. 向量匹配运算符 Vector matching

4.2.1. 一对一 向量匹配 One-to-one vector matches

4.2.2. 组修饰符

4.2.2.1. 多对一 和 一对多 向量匹配 Many-to-one and one-to-many vector matches

4.3. 聚合运算符

1. 基本介绍

普罗米修斯提供了一种称为PromQL（普罗米修斯查询语言）的函数式查询语言，允许用户实时选择和聚合时间序列数据。

表达式的结果可以显示为图形，在 Prometheus 的表达式浏览器中显示为表格数据，也可以通过 HTTP API 由外部系统使用。

通常初学者可以在浏览器中，可以先通过 Prometheus 的查询框输入查询语句，从而获取到查询结果。

up

up 指标，几乎是每个 exporter 都会有一个 监控指标，是表示被监控对象是否存活的状态。1 表示存活，0 表示被监控对象不在线或者 Prometheus 和 被监控对象上的 exporter 无法通信的状态。

The screenshot shows the Prometheus web interface. At the top, there's a navigation bar with 'Prometheus', 'Alerts', 'Graph', 'Status', and 'Help'. Below it, there are checkboxes for 'Use local time', 'Enable query history', 'Enable autocomplete', 'Enable highlighting', and 'Enable linter'. The search bar contains the query 'up'. Below the search bar, there are tabs for 'Table' and 'Graph'. The 'Table' view is selected, showing a table with two rows of data. The first row is 'up{instance="192.168.146.138:9100", job="beijing-servers"}' with a value of 1. The second row is 'up{instance="localhost:9090", job="prometheus"}' with a value of 1. The 'Execute' button is visible on the right. At the bottom, there's an 'Add Panel' button.

Evaluation time	
up{instance="192.168.146.138:9100", job="beijing-servers"}	1
up{instance="localhost:9090", job="prometheus"}	1

2. PromQL 查询结果的数据类型

在普罗米修斯的表达语言中，表达式或子表达式计算结果可以是如下四种类型：

- **Instant vector** 即时向量 一组按照当前时间戳获取的监控指标时间序列数据，例如当前服务器内存可以大小，当前服务器的CPU使用率等。

示例：

假设目前监控了 3 台服务器，现在我希望获取 3 台服务器此刻每台服务器的内存剩余可用容量。可以输入语句：

```
node_memory_MemAvailable_bytes
```

应该返回如下样式的数据：

The screenshot shows the Prometheus web interface with the query 'node_memory_MemAvailable_bytes'. The 'Table' view is selected, showing a table with three rows of data. The first row is 'node_memory_MemAvailable_bytes{group="beijing", instance="10.10.40.200:9111", job="beijing"}' with a value of 17397915648. The second row is 'node_memory_MemAvailable_bytes{group="beijing", instance="10.10.40.62:9111", job="beijing"}' with a value of 28026298368. The third row is 'node_memory_MemAvailable_bytes{group="guangzhou", instance="13.200.255.111", job="hwcloud"}' with a value of 1622566784. The 'Execute' button is visible on the right. A red box highlights the values in the table, and a red text label '单位为 字节的内存容量' (Unit is bytes of memory capacity) points to the values.

Evaluation time	
node_memory_MemAvailable_bytes{group="beijing", instance="10.10.40.200:9111", job="beijing"}	17397915648
node_memory_MemAvailable_bytes{group="beijing", instance="10.10.40.62:9111", job="beijing"}	28026298368
node_memory_MemAvailable_bytes{group="guangzhou", instance="13.200.255.111", job="hwcloud"}	1622566784

上图中每一行都是一个服务器的内存当前可用容量指标数据，后面是具体的值，单位：字节。

- **Range vector** 范围向量 一组按照指定时间范围获取到的监控指标时间序列数据，包含了在这个时间范围内，随着时间变化而变化的指标值，例如最近 15 秒内的内存剩余可用值。可以输出查询语句：`node_memory_MemAvailable_bytes[15s]`

Q node_memory_MemAvailable_bytes[15s]		Execute
Table	Graph	Load time: 62ms Resolution: 14s Result series: 1
<div> <div><</div> <div>Evaluation time</div> <div>></div> </div>		
node_memory_MemAvailable_bytes{group="beijing", instance="10.10.40.200:9111", job="beijing"}		17051344896 @1694497278.455 17051881472 @1694497283.455 17048109056 @1694497288.455
node_memory_MemAvailable_bytes{group="beijing", instance="10.10.40.62:9111", job="beijing"}		28097605632 @1694497277.353 28082843648 @1694497282.353 28009291776 @1694497287.353
node_memory_MemAvailable_bytes{group="guangzhou", instance="10.10.40.38:9111", job="hwcloud"}		16236961792 @1694497275.256 16236937216 @1694497280.256 16236552192 @1694497285.256

从上图可以看到，每个服务器都有 3 组数据，这是因为在 prometheus.yml 配置中的 `scrape_interval` 的值是 5s, 表示每 5 秒中 Prometheus 会查询一次数据并保存到自己的数据库中。所以这里查询最近 15 秒的数据，就会返回 3 组数据。

17051344896 @1694497278.455 中 17051344896 是具体的数值，@1694497278.455 表示一个时间戳，这个时间戳就是从 Unix 元年开始到现在的秒数。

- Scalar 标量 - 一个简单的数字浮点值；通常是通过查询表达式计算出来的结果。例如，内存使用率 = 内存可用容量 / 内存总容量 = 0.23
- String 字符串 - 一个简单的字符串值；当前未使用

3. 查询基础语法

所有的查询都遵循如下的语法格式：

监控指标名称{标签名=标签值,...}[时间单位]

标签和时间单位都是可选的。

例如最简单的一个只有指标名称的查询: `up`

Prometheus
Alerts
Graph
Status
Help

☒ Use local time
☐ Enable query history
☒ Enable autocomplete
☒ Enable highlighting
☒ Enable linter

Q up
Execute

Table
Graph
Load time: 3ms Resolution: 14s Result series: 2

<

Evaluation time

>

up{instance="192.168.146.138:9100", job="beijing-servers"}	1
up{instance="localhost:9090", job="prometheus"}	1

Remove Panel

Add Panel

CSDN @shark_西瓜甜

3.1 时间序列选择器 Time series Selectors

Prometheus 的存储监控数据是用 时间序列方式保存的。

所以 PromQL 查询都是通过 时间序列的方式查询的。

时间序列选择器 用于对查询的数据进行条件过滤。

- 即时向量选择器
- 范围向量选择器

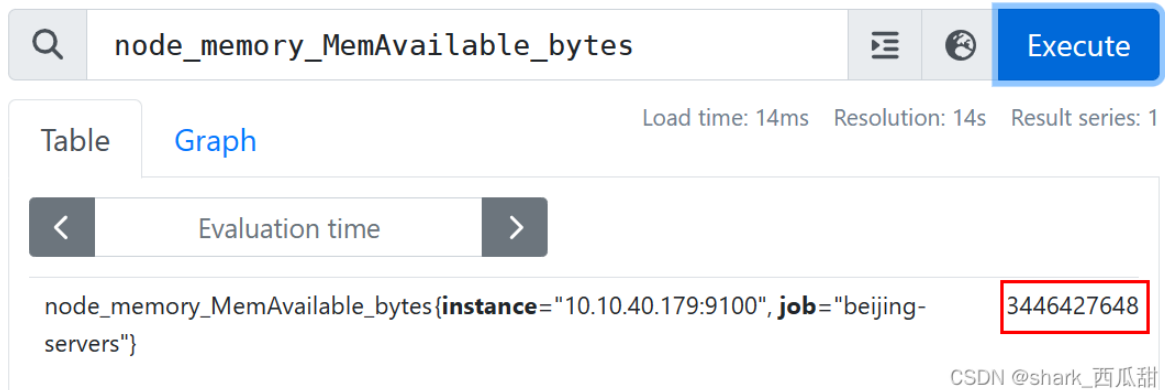
3.2. 即时向量 (Instant vector) 选择器

Instant vector 选择器允许在指定一个具体的时间戳，会返回此时间戳当时的指标值，不指定具体时间戳，查询的是当前时间。

在最简单的形式中，只指定一个指标名称。这将产生一个即时向量，其中包含具有此指标名称的所有时间序列的数据。

查询当前时间的值

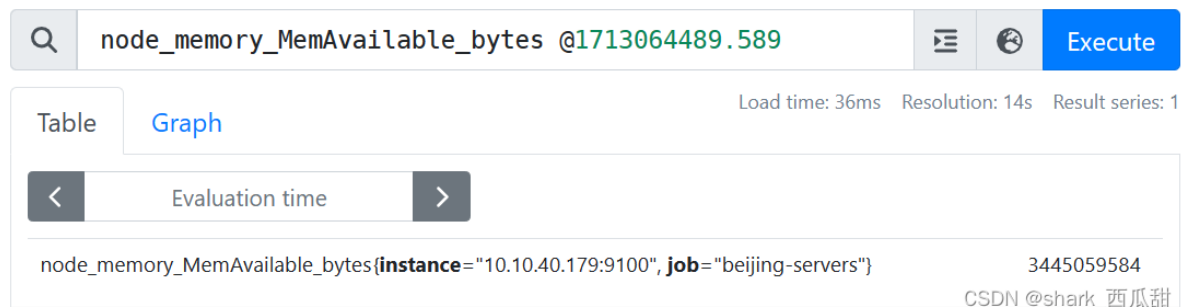
此示例查询服务器当前时间的剩余内存时间序列数据： `node_memory_MemAvailable_bytes`



查询指定时间戳的值

此示例查询服务器在指定时间戳 **1713064489.589** 时的剩余内存时间序列数据：

`node_memory_MemAvailable_bytes @1713064489.589`



时间戳转为年月日

```
[root@prome ~]# date -d @1713064489.589
2024年 04月 14日 星期日 11:14:49 CST
```

3.3 范围向量选择器

范围向量的工作原理与即时向量一样，只是它们是指定一个时间范围进行查询，会返回指定范围内的每个时间戳当时保存的时间序列数据。比如 5分钟内，1小时内，2天内等。

3.3.1. 语法

从语法上讲，时间长度附加在方括号（`[]`）中，在向量选择器的末尾。

在本例中，我们选择过去 1 分钟内为所有时间序列记录的所有值，这些值的指标名称为

`node_memory_MemAvailable_bytes`

```
node_memory_MemAvailable_bytes[1m]
```



3.3.2 查询结果解析

监控频率

时间范围查询，在返回结果中会有几个数据是根据在全局配置中或者具体的每个 `job` 中 `scrape_interval` 的值所决定的。表示Prometheus 每隔多长时间向被监控对象获取一次监控指标的数据。

如下配置是此文档实验环境中的配置值。

```
global:
  scrape_interval: 15s
```

返回结果的格式

返回结果的格式是以 `@` 符号为分隔，`@` 前面的是具体监控指标的值，`@` 后面是时间戳。

以上返回的结果表示服务器最近 1 分钟内4个时间节点的剩余内存的容量（单位: bytes 字节）。

3.3.3. 时间长度

时间长度指定为数字，紧接着是以下单位之一：

- `ms` 毫秒
- `s` 秒
- `m` 分钟
- `h` 小时
- `d` 天 - 假设一天总是24小时
- `w` 周 - 假设一周总是7d
- `y` 年 - 假设一年总是365d

时间长度可以通过串联来组合。单位必须从最长到最短给定，比如 1小时30分钟: `1h30m`。

给定的单位在一段时间内只能出现一次，比如不可以: `3m4m`。

以下是一些有效时间长度的示例：

```
1d
1h30m
5m
10s
```

3.3.4. 偏移修饰符(不常用)

`offset` 修饰符允许更改查询中单个即时向量和范围向量的时间偏移量。

例如，以下表达式返回 5 分钟之前 `node_memory_MemFree_bytes` 即时向量的值：

```
node_memory_MemAvailable_bytes offset 5m
```

就是 假设当前时间的 **3:50**, `offset 5m` 表示的时间就是 **3:45**, 那上面的查询语句, 表示查询服务器在 3 点 45 分内存的剩余可以大小。

请注意, `offset` 修饰符总是需要立即跟随选择器, 即以下内容是正确的:

```
sum(node_memory_MemAvailable_bytes offset 5m) // GOOD.
```

`sum` 是一个计算总和的函数, 后续会介绍
显然以下情况不正确:

```
sum(node_memory_MemAvailable_bytes) offset 5m // INVALID.
```

范围向量也是如此。这返回了 `node_memory_MemAvailable_bytes` 一周前的 5 分钟范围内的数据:

```
node_memory_MemAvailable_bytes[5m] offset 1w
```

3.3.5. @ 修饰符(不常用)

@ 修饰符允许对单个即时和范围向量的具体时间进行指定。提供给 @ 修饰符的时间是 **unix** 时间戳, 并用 **Float** 文字描述。

说白了就是可以指定某个时间点

例如, 以下表达式在 2024-04-14 12:30 返回 `node_memory_MemAvailable_bytes` 的值:

```
node_memory_MemAvailable_bytes @1713069000
```

时间转换: `date -d "20240414 12:30" +%s`

请注意, 同样 @ 修饰符总是需要立即跟随选择器。

范围向量也是如此。

@ 修饰符支持上述在 **int64** 范围内的所有浮点文字表示。它也可以与 `offset` 修饰符一起使用, 其中 `offset` 相对于 @ 修饰符时间应用, 无论先写入哪个修饰符, 这2个查询将产生相同的结果。

指定时间 **1713069000** 的 **5 m** 之前的数据:

```
# offset 在 @ 之后
node_memory_MemAvailable_bytes @1713069000 offset 5m
# offset 在 @ 之前
node_memory_MemAvailable_bytes offset 5m @1713069000
```

3.4. 标签匹配器

3.4.1 语法介绍

标签匹配器是在一个简单的指标名称后面添加大括号 (`{}`), 并在大括号中加入标签键值对, 可以实现对查询结果的过滤, 这有点像是 SQL 中的 **where** 条件语句。

PromQL:

```
up{instance="192.168.146.138:9100"}
```

转化成 SQL:

```
select * from up where instance="192.168.146.138:9100"
```

此示例表示查询标签 **instance** 的值是 **192.168.146.138:9100** 的 **up** 指标数据。

3.4.2. 标签匹配运算符

存在以下标签匹配运算符:

- **=** 选择与提供的字符串完全相等的标签。
- **!=** 选择不等于提供的字符串的标签。
- **=~** 选择与正则表达式匹配的标签。
- **!~** 选择与正则表达式不匹配的标签。

注意: 正则表达式匹配的完全锚定。 `env=~"foo"` 的匹配被视为 `env=~"^foo$"`

标签匹配器中使用逗号 **,** 来分隔多个标签, 并表示是 **and** 的关系。

如下示例: 返回标签 **instance** 的值以 **192.168.146** 开头, 后面跟着 1 到 3 个数字的任意组合, 并且标签 **job** 的值是 **beijing-servers** 的数据。

```
up{instance=~"192.168.146.[0-9]{1,3}.*", job="beijing-servers"}
```

加入指定了一个值为空的标签, 这样的匹配器也会匹配到没有设置此标签的数据。

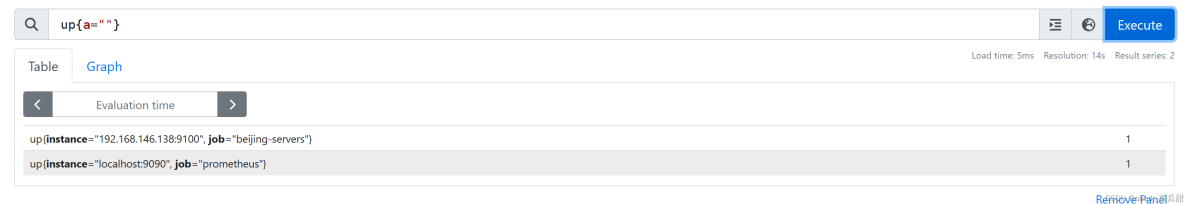


Table	Graph	Load time: 5ms	Resolution: 14s	Result series: 2
Evaluation time				
up{instance="192.168.146.138:9100", job="beijing-servers"}				1
up{instance="localhost:9090", job="prometheus"}				1

向量选择器必须指定一个名称或至少含有一个非空字符串 (相对来说, 空字符串是 `""`, 注意不是一个空格的字符串 `" "`。) 的标签匹配器。以下表达式是非法的:

```
{job=~".*"} # 错误!
```

错误的原因: 这个可以匹配到一个空字符串的标签, 比如 `{job=~""}`, 这种情况下, 它还没有指定一个具体的指标名称。

相比之下, 这些表达式是有效的, 因为它们都至少有一个非空的标签的选择器。

```
{job=~".+"} # Good!, 这个正则表示一个以及一个以上的字符 (包含空格, 虽然实际上并没有任何意义)。  
{job=~".*", method="get"} # Good!
```

以上两个表达式, 虽然都没有指定一个具体的指标名称, 但是它们都不会匹配到一个空标签。空标签就是 `{job=""}`

通过与内部 `__name__` 标签进行匹配, 标签匹配器也可以应用于指标名称。

例如, 表达式 `up` 等效于 `{__name__="up"}`。也可以使用除 `=` 之外的匹配运算符 (`!=`, `=~`, `!~`)。

以下表达式将会找到所有指标名称含有 `memoroy` 关键字的指标:

```
{__name__=~".*memory.*"}
```

指标名称不得是关键字 **bool**、**on**、**ignoring**、**group_left** 和 **group_right** 之一。以下表达方式是非法的：

```
on{}
```

此限制的变通办法是使用**name**标签：

```
{__name__="on"} # Good!
```

3.5. 避免慢速查询和过载

如果查询需要对大量数据进行操作，则绘制查询可能会超时或使服务器或浏览器过载。

因此，在对未知数据构建查询时，请始终在普罗米修斯表达式浏览器的**Table**视图中开始构建查询，直到结果集看起来合理（最多是数百个，而不是数千个时间序列）。

只有在充分筛选或汇总数据后，才能切换到 **Graph** 模式。如果表达式仍然需要很长时间来进行特别绘图，请通过[记录规则（recording rules）](#) 对其进行预记录。

这与普罗米修斯的查询语言尤其相关，在普罗米修斯中，像 `api_http_requests_total` 这样的裸度量名称选择器可以扩展到数千个具有不同标签的时间序列。还要记住，即使输出只是少量的时间序列，在许多时间序列上聚合的表达式也会在服务器上产生负载。这类似于对关系数据库中一列的所有值求和的速度很慢，即使输出值只有一个数字。

3.6. 子查询

子查询允许您针对给定的时间范围(range) 和获取频率(resolution)进行即时查询(instant query)。

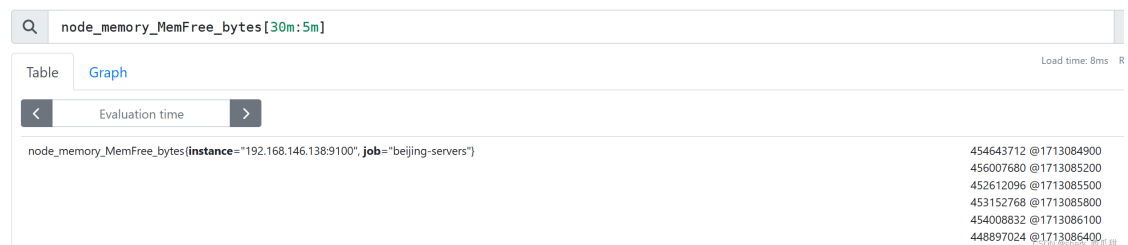
子查询的结果是一个范围向量。

语法： `<instant_query> '[' <range> ':' [<resolution>] ']' ['@ <float_literal>] [offset <duration>]`

`<resolution>` 是可选的。默认值是全局的监控指标获取间隔 `scrape_interval`。

- 返回过去30分钟剩余内存容量，并且按照 每 5 分钟的频率获取数据。

```
node_memory_MemFree_bytes[30m:5m]
```



4. 查询语法中的操作运算符 Operators

Prometheus支持许多二进制和聚合运算符。

4.1. 二进制运算符

PromQL 支持基本的逻辑和算术运算符。用于对两个即时向量之间的操作结果，进行算术运算和逻辑运算。

4.1.1. 算术运算符

普罗米修斯中存在以下二进制算术运算符：

- `+` 加法
- `-` 减法
- `*` 乘法
- `/` 除法
- `%` 取模运算(不常用)，计算两数相除后的余数，比如 `10 % 3` ,结果就是 1，因为 `10 / 3` 余数是 1
- `^` 幂运算(不常用), 比如 2 的 3 次方写成: `2^3`

计算根分区剩余可用率

```
node_filesystem_avail_bytes{mountpoint="/"} /  
node_filesystem_size_bytes{mountpoint="/"} * 100
```

4.1.2. 比较二进制运算符：

- `==` 等于
- `!=` 不等于
- `>` 大于
- `<` 小于
- `>=` 大于等于
- `<=` 小于等于

计算根分区剩余可用率 低于 20%

```
node_filesystem_avail_bytes{mountpoint="/"} /  
node_filesystem_size_bytes{mountpoint="/"} * 100 < 20
```

node_filesystem_avail_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint="/"} * 100 < 20

Table Graph Load time: 17ms Resolution: 14s Result series: 0

Empty query result 不符合条件，返回值为空

node_filesystem_avail_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint="/"} * 100 < 90

Table Graph Load time: 17ms Resolution: 14s Result series: 1

(device="/dev/mapper/centos_prometheus-root", fstype="xfs", instance="10.10.40.179:9100", job="beijing-servers", mountpoint="/") 88.70308907443749

Add Panel

CSDN @shark_西瓜甜

4.1.3. 逻辑二进制运算符

`and` 逻辑与

`unless` 排除某个

`or` 逻辑或

and

`vector1 and vector2` 返回的结果是由 `vector1` 的元素组成的向量，但是结果集的每条数据，都必须和其中 `vector2` 中的数据具有完全匹配的标签集（即标签名和其对应的标签值）。`vector1` 中不匹配的数据将删除。指标名称和值是从左侧向量中继承的。

如有如下数据：

```
uphost{hostname="server-a", location="beijing"}      1
uphost{hostname="server-b", location="nanjing"}      0
uphost{hostname="server-c", location="shanghai"}     0

webserver{hostname="server-a", location="beijing"}   156
webserver{hostname="server-c", location="wuhan"}     157
```

`uphost and webserver` 将会得到如下结果集

The image displays three screenshots of the Prometheus query interface, showing the results of different queries. Each screenshot includes a search bar, a 'Table' tab, and a 'Graph' tab. The 'Table' tab is selected, and the results are shown in a table format. The first screenshot shows the query 'uphost' with three rows. The second screenshot shows the query 'webserver' with two rows. The third screenshot shows the query 'uphost and webserver' with one row, which is the first row of the 'uphost' query.

Query	Result
uphost	uphost{hostname="server-a", instance="10.10.40.179:9100", job="beijing-servers", location="beijing"} 1
uphost	uphost{hostname="server-b", instance="10.10.40.179:9100", job="beijing-servers", location="nanjing"} 0
uphost	uphost{hostname="server-c", instance="10.10.40.179:9100", job="beijing-servers", location="shanghai"} 0
webserver	webserver{hostname="server-a", instance="10.10.40.179:9100", job="beijing-servers", location="beijing"} 156
webserver	webserver{hostname="server-c", instance="10.10.40.179:9100", job="beijing-servers", location="wuhan"} 157
uphost and webserver	uphost{hostname="server-a", instance="10.10.40.179:9100", job="beijing-servers", location="beijing"} 1

还可以使用 `on` 子句，控制匹配的标签限制在 `on` 指定的一个标签列表中，语法如下：

`vector1 and on (label-a, label-b) vector2`

如下示例是在右侧的数据集中找到和左侧的数据标签 `hostname` 值相同的，之后再把左侧的这个数据返回。

`webserver and on(hostname) uphost`

Q

uphost

Execute

Table

Graph

Load time: 15ms Resolution: 14s Result series: 3

<

Evaluation time

>

uphost(hosname="servera", instance="10.10.40.179:9100", job="beijing-servers", location="beijing")

1

uphost(hosname="serverb", instance="10.10.40.179:9100", job="beijing-servers", location="nanjing")

0

uphost(hosname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="shanghai")

0

Remove Panel

Q

webserver

Execute

Table

Graph

Load time: 17ms Resolution: 14s Result series: 2

<

Evaluation time

>

webserver(hosname="servera", instance="10.10.40.179:9100", job="beijing-servers", location="beijing")

156

webserver(hosname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="wuhan")

157

Remove Panel

Q

uphost and on(hosname) webserver

Execute

Table

Graph

Load time: 27ms Resolution: 14s Result series: 2

<

Evaluation time

>

uphost(hosname="servera", instance="10.10.40.179:9100", job="beijing-servers", location="beijing")

1

uphost(hosname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="shanghai")

0

CSDN @shark_西瓜甜

Remove Panel

or

vector1 or vector2

包含 vector1 的所有原始元素（标签集+值）以及另外包含 vector2 的在 vector1 中不具有匹配标签集的所有元素的向量。

Q uphost

Table Graph Load time: 15ms Resolution: 14s Result series: 3

Evaluation time

uphost(hostname="servera", instance="10.10.40.179:9100", job="beijing-servers", location="beijing")	1
uphost(hostname="serverb", instance="10.10.40.179:9100", job="beijing-servers", location="nanjing")	0
uphost(hostname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="shanghai")	0

Remove Panel

Q webserver

Table Graph Load time: 17ms Resolution: 14s Result series: 2

Evaluation time

webserver(hostname="servera", instance="10.10.40.179:9100", job="beijing-servers", location="beijing")	156
webserver(hostname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="wuhan")	157

Remove Panel

Q uphost or webserver

Table Graph Load time: 10ms Resolution: 14s Result series: 4

Evaluation time

uphost(hostname="servera", instance="10.10.40.179:9100", job="beijing-servers", location="beijing")	1
uphost(hostname="serverb", instance="10.10.40.179:9100", job="beijing-servers", location="nanjing")	0
uphost(hostname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="shanghai")	0
webserver(hostname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="wuhan")	157

Remove Panel

CSDN @shark_西瓜甜

假如只关注 hostname 标签:

Q uphost

Table Graph Load time: 15ms Resolution: 14s Result series: 3

Evaluation time

uphost(hostname="servera", [REDACTED])	1
uphost(hostname="serverb", [REDACTED])	0
uphost(hostname="serverc", [REDACTED])	0

Remove Panel

Q webserver

Table Graph Load time: 10ms Resolution: 14s Result series: 2

Evaluation time

webserver(hostname="servera", [REDACTED])	156
webserver(hostname="serverc", [REDACTED])	157

Remove Panel

Q uphost or on(hostname) webserver

Table Graph Load time: 23ms Resolution: 14s Result series: 3

Evaluation time

uphost(hostname="servera", [REDACTED])	1
uphost(hostname="serverb", [REDACTED])	0
uphost(hostname="serverc", [REDACTED])	0

Remove Panel

CSDN @shark_西瓜甜

unless

vector1 unless vector2

从 `vector1` 返回的结果中排除和 `vector2` 中标签及其值完全相同的数据。
示例:

```
uphost unless webserver
```

从查询的结果集中排除 `instance` 的值为 `localhost:9090` 的数据

The first screenshot shows the query `uphost` with three results. The first result is highlighted with a pink box, and the second and third are highlighted with a red box.

Query	Result
<code>uphost{hostname="servera", instance="10.10.40.179:9100", job="beijing-servers", location="beijing"}</code>	1
<code>uphost{hostname="serverb", instance="10.10.40.179:9100", job="beijing-servers", location="nanjing"}</code>	0
<code>uphost{hostname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="shanghai"}</code>	0

The second screenshot shows the query `webserver` with two results. The first result is highlighted with a pink box, and the second is highlighted with a red box.

Query	Result
<code>webserver{hostname="servera", instance="10.10.40.179:9100", job="beijing-servers", location="beijing"}</code>	156
<code>webserver{hostname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="wuhan"}</code>	157

The third screenshot shows the query `uphost unless webserver` with two results. The first result is highlighted with a pink box, and the second is highlighted with a red box.

Query	Result
<code>uphost{hostname="serverb", instance="10.10.40.179:9100", job="beijing-servers", location="nanjing"}</code>	0
<code>uphost{hostname="serverc", instance="10.10.40.179:9100", job="beijing-servers", location="shanghai"}</code>	0

4.1.4. 二进制运算符优先级

下面的列表显示了普罗米修斯中二进制运算符的优先级，从高到低。

1. `^`
2. `*`, `/`, `%`
3. `+`, `-`
4. `==`, `!=`, `<=`, `<`, `>=`, `>`
5. `and`, `unless`
6. `or`

具有相同优先级的运算符保持关联。例如，`2 * 3 % 2` 相当于 `(2 * 3) % 2`。但是 `^` 是右关联的，所以 `2 ^ 3 ^ 2` 等于 `2 ^ (3 ^ 2)`

4.2. 向量匹配运算符 Vector matching

向量之间的运算试图在右侧向量中为左侧的每个条目找到完全匹配的数据条目，这个匹配的条件两个数据具有完全相同的标签集和相应的值。

有两种基本类型的匹配行为：

- 一对一
- 多对一 或 一对多

4.2.1. 一对一 向量匹配 One-to-one vector matches

一对一 从左侧结果集中拿出每一条数据，到右侧结果集寻找一条标签名和相应标签的值完全相同的数据，之后用这两个数据进行二进制操作符（加、减、乘、除等）的运算。

在默认情况下，这是一个遵循 `vector1<operator>vector2` 格式的操作。

如果两个条目具有完全相同的标签集和相应的值，则它们匹配。我们可以利用如下的两个关键字中的一个，对标签的匹配进行影响：

- `ignoring` 关键字允许在匹配时忽略某些标签，就是在匹配的时候不需要考虑 `ignoring` 指定的这些标签，是一个列表，用逗号分隔每个标签。
- `on` 关键字允许将所需要匹配的标签集限定在所提供的列表中，就是在匹配的时候仅限于 `on` 所指定的这些标签。

语法：

```
<vector expr> <bin-op> ignoring(<label list>) <vector expr>

<vector expr> <bin-op> on(<label list>) <vector expr>
```

`<vector expr>` 是 PromQL 查询表达式

`<bin-op>` 就是二进制操作符

`<label list>` 就是 标签列表

假设有如下数据

```
method_code:http_errors:rate5m{method="get", code="500"} 24
method_code:http_errors:rate5m{method="get", code="404"} 30
method_code:http_errors:rate5m{method="put", code="501"} 3
method_code:http_errors:rate5m{method="post", code="500"} 6
method_code:http_errors:rate5m{method="post", code="404"} 21

method:http_requests:rate5m{method="get"} 600
method:http_requests:rate5m{method="del"} 34
method:http_requests:rate5m{method="post"} 120
```

查询示例：

```
method_code:http_errors:rate5m{code="500"} / ignoring(code)
method:http_requests:rate5m
```

这将返回一个结果向量，其中包含在过去5分钟内测量到的每个 HTTP 请求方法的响应状态代码为500的 HTTP请求占对应的HTTP请求方法的多少。

如果不忽略（code），就不会有匹配，因为指标不共享同一组标签。方法为 `put` 和 `del` 的条目不匹配，不会显示在结果中：

```
{method="get"} 0.04 // 24 / 600
{method="post"} 0.05 // 6 / 120
```

4.2.2. 组修饰符

这些组修饰符可以实现多对一/一对多矢量匹配：

- `group_left`
- `group_right`

多对一和一对多匹配是高级用例，应该仔细考虑。通常，正确使用 `ignoring(<label list>)` 可以提供所需的结果。

分组修饰符只能用于 **比较** 和 **算术**。操作为 **and**，**unless** 和 **or** 操作默认情况下与右向量中的所有可能条目匹配。

4.2.2.1. 多对一 和 一对多 向量匹配 Many-to-one and one-to-many vector matches

多对一和一对多匹配指的是“一”侧的每个向量元素可以与“多”侧的多个元素匹配的情况。这必须使用 `group_left` 或 `group_right` 修饰符明确请求，其中 `left/right` 决定哪个向量具有更高的基数。

匹配之后，永远会拿左侧的数据去和右侧匹配的数据进行二进制运算并返回运算的结果。

语法：

```
<vector expr> <bin-op> ignoring(<label list>) group_left(<label list>) <vector expr>
<vector expr> <bin-op> ignoring(<label list>) group_right(<label list>) <vector expr>
<vector expr> <bin-op> on(<label list>) group_left(<label list>) <vector expr>
<vector expr> <bin-op> on(<label list>) group_right(<label list>) <vector expr>
```

- `ignoring(<label list>)` 在匹配的时候不用考虑这些标签。
- `on(<label list>)` 在匹配的时候只考虑指定的这些标签
- `group_left(<label list>)` 标签列表可以提供给组修饰符，其中包含要包含在结果度量中的“一”侧的标签。

示例数据：

```
method_code:http_errors:rate5m{method="get", code="500"} 24
method_code:http_errors:rate5m{method="get", code="404"} 30
method_code:http_errors:rate5m{method="put", code="501"} 3
method_code:http_errors:rate5m{method="post", code="500"} 6
method_code:http_errors:rate5m{method="post", code="404"} 21

method:http_requests:rate5m{method="get", hostname="a", server="10"} 600
method:http_requests:rate5m{method="del", hostname="b", server="10"} 34
method:http_requests:rate5m{method="post", hostname="c", server="30"} 120
```

查询目的：

当前希望查询 每个请求方法(get,post,put)的响应码（500,404,501）在每个请求方法总数中的占比。

计算公式: 每种方法的状态码的指标数据 除以对应每种方法的请求总数

可使用查询示例：

```
method_code:http_errors:rate5m / on(method) group_left
method:http_requests:rate5m
```

在这种情况下，左侧表达式中每个 `method` 标签值包含了多个条目（比如 `get` 就有 2 条）。因此，我们使用 `group_left` 来表示这一点。

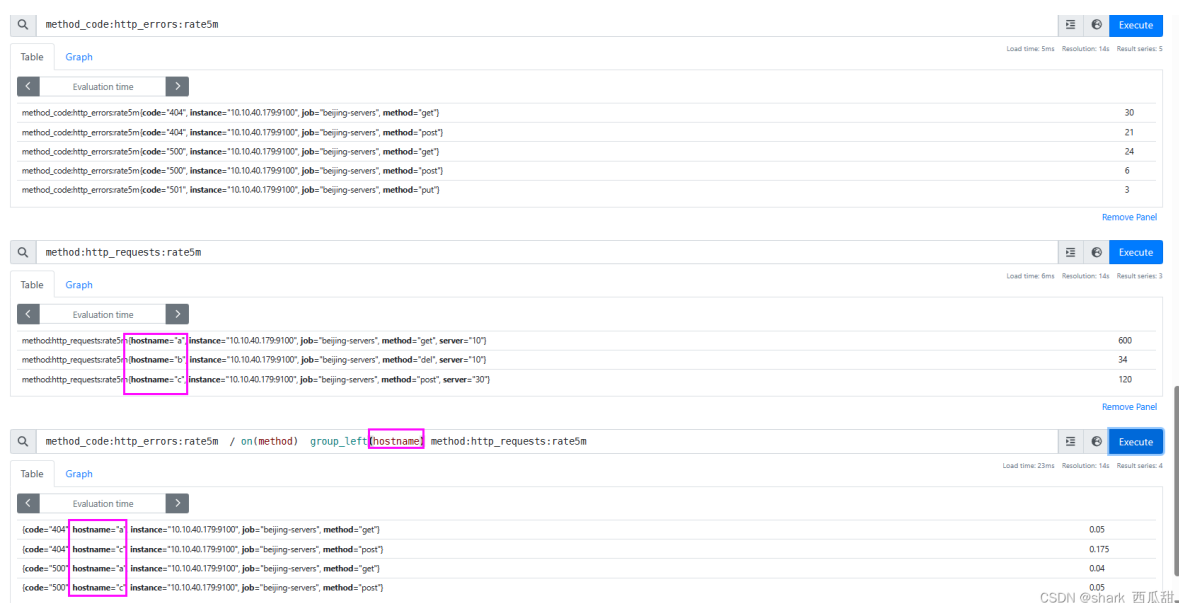
右侧表达式中的元素的标签，在左侧表达式具有相同 `method` 标签的数据有多个：

```
{code="404", instance="10.10.40.179:9100", job="beijing-servers", method="get"}
0.05      // 24 / 600
{code="404", instance="10.10.40.179:9100", job="beijing-servers", method="post"}
0.175     // 30 / 600
{code="500", instance="10.10.40.179:9100", job="beijing-servers", method="get"}
0.04      // 6 / 120
{code="500", instance="10.10.40.179:9100", job="beijing-servers", method="post"}
0.05      // 21 / 120
```

`put` 和 `del` 没有同时在两侧出现，所以无法匹配成功，结果中不包含 `put` 和 `del` 方法的条目。

`group_left(<label list>)` 的应用效果，将会把 "—" 那测被指定的标签也出现在结果集中。例如下图中指定了标签 `hostname`：

`group_left(hostname)`



4.3. 聚合运算符

Prometheus支持以下内置聚合运算符，这些运算符可用于对 `单个即时向量` 的数据进行集合运算，从而产生具有聚合值的更少数据的新向量：

- `sum`（对查询结果集中所有数据的值求和）
- `min`（获取最小值）
- `max`（获取最大值）
- `avg`（计算平均值）

- `group` (结果向量中的所有值都是1)

Search: `group(up)` Execute

Table Graph Load time: 4ms Resolution: 14s Result series: 1

< Evaluation time >

{}	1
----	---

CSDN @shark_西瓜甜

- `count` (计算查询的结果集中有多少条数据)
统计目前监控了多少台服务器

```
count(up)
```

- `count_values` (计算具有相同值的数据出现的次数, 需要传递一个英文字符串的参数, 这个参数会作为一个新的 key 给返回值使用, 这个有点像 SQL 语句中的 `group by`)
分别统计当前在线和非在线服务器的数量, 并且打上新的标签 `count`

```
count_values("server_st", up)
```

Search: `count_values("server_st", up)` Execute

Table Graph Load time: 32ms Resolution: 14s Result series: 2

< Evaluation time >

{server_st="1"}	37
{server_st="0"}	16

[Remove Panel](#)

[Add Panel](#)

CSDN @shark_西瓜甜

- `bottomk` (找到所有结果集中值最小的前几个)
- `topk` (找到所有结果集中值最大的前几个)
`topk(3,prometheus_http_requests_total)`

Search: `topk(3,prometheus_http_requests_total)` Execute

Table Graph Load time: 6ms Resolution: 14s Result series: 3

< Evaluation time >

prometheus_http_requests_total{code="200", handler="/metrics", instance="localhost:9090", job="prometheus"}	265
prometheus_http_requests_total{code="200", handler="/api/v1/query", instance="localhost:9090", job="prometheus"}	74
prometheus_http_requests_total{code="200", handler="/static/*filepath", instance="localhost:9090", job="prometheus"}	41

[Remove Panel](#)

- `quantile` (分位数计算 φ -quantile ($0 \leq \varphi \leq 1$), φ 的取值范围 0-1)

```
quantile(0.5, http_request_total)
```

分位数 (quantile)：在一组从小到大排列的一组数据中，体现数据的分布情况。

比如有一组数据 3 1 7 2 7 5 29,按照升序排列后就是：1 2 3 5 7 7 29.

排序后一共 7 个数，如果取处于 50%(0.5)分位的数据，就是在他们中间的数，就是 4 这个数字。

那这个数怎么计算得来的呢？

计算公式： $Ly = (n+1) * y\%$

Ly 是在要获取的数具体在这组数据的第几 (Ly) 个位置。

n 就是这组数据总共几个数字。

y 就是百分数的分子，百分之y,比如 25%，y 就是 25。

比如刚才的这组数：1 2 3 4 7 7 29，总共 7 个数字。想知道处于这组数据 50% 位置的数字在这组数据的第几个位置，计算公式如下：

$(7 + 1) * 50\% = 4$

第 4 个位置的数字 5 就是处在这组数据的第 50% 的分位数。50% 的数字也称为中位数。

以上介绍的聚合运算符运算时候，都可以通过使用 `without` 或 `by` 函数的子句来计算不同的维度。

`without` 和 `by` 函数接收一个包含标签名的列表，聚合函数会对按照标签值相同的数据进行计算，有点像 MySQL 中的 `group by` 语句。

`without` 或 `by` 可以在表达式之前或之后使用。

语法：

```
<aggr-op> [without|by (<label list>)] ([parameter,] <vector expression>)
```

或

```
<aggr-op>([parameter,] <vector expression>) [without|by (<label list>)]
```

- `aggr-op` 聚合函数
- `parameter` 聚合函数的参数(如果需要)，仅对 `count_values`, `quantile`, `topk` and `bottomk` 是必须的
- `label list` 标签列表，也可以包括用逗号隔开的一个或多个标签名，即 `(label1)`, `(label1, label2)` 和 `(label1, label2,)` 都是有效语法。

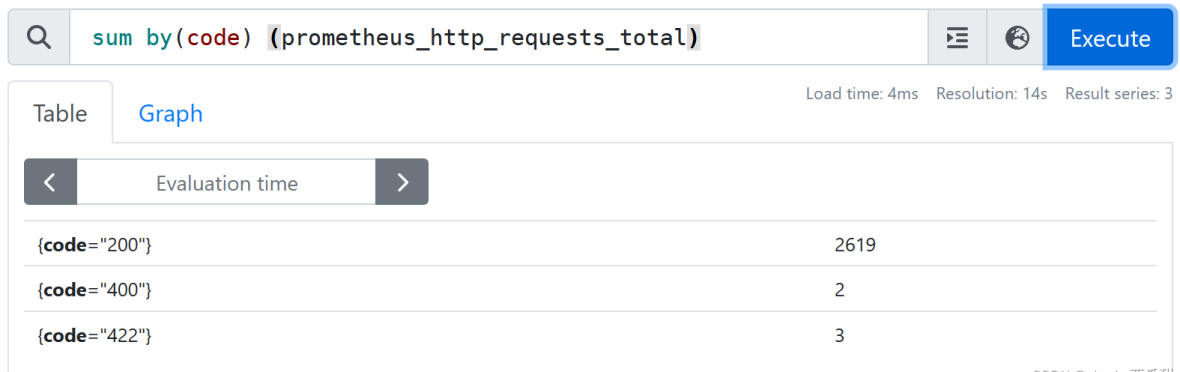
`by(label1)` 进行聚合函数计算的时候，只考虑 `label1` 这个标签，只会按照 `label1` 的值相同的数据进行分组计算。

例如有如下数据：

prometheus_http_requests_total	code="200"	handler="/-/ready", instance="localhost:9090", job="prometheus"	26
prometheus_http_requests_total	code="200"	handler="/api/v1/label/name/values", instance="localhost:9090", job="prometheus"	28
prometheus_http_requests_total	code="200"	handler="/api/v1/labels", instance="localhost:9090", job="prometheus"	2
prometheus_http_requests_total	code="200"	handler="/api/v1/metadata", instance="localhost:9090", job="prometheus"	8
prometheus_http_requests_total	code="200"	handler="/api/v1/query", instance="localhost:9090", job="prometheus"	100
prometheus_http_requests_total	code="200"	handler="/api/v1/series", instance="localhost:9090", job="prometheus"	1
prometheus_http_requests_total	code="200"	handler="/api/v1/status/buildinfo", instance="localhost:9090", job="prometheus"	1
prometheus_http_requests_total	code="200"	handler="/favicon.ico", instance="localhost:9090", job="prometheus"	26
prometheus_http_requests_total	code="200"	handler="/graph", instance="localhost:9090", job="prometheus"	26
prometheus_http_requests_total	code="200"	handler="/metrics", instance="localhost:9090", job="prometheus"	2328
prometheus_http_requests_total	code="200"	handler="/static/filepath", instance="localhost:9090", job="prometheus"	54
prometheus_http_requests_total	code="400"	handler="/api/v1/query", instance="localhost:9090", job="prometheus"	2
prometheus_http_requests_total	code="422"	handler="/api/v1/query", instance="localhost:9090", job="prometheus"	3

从上图数据中可以看出，code 标签值为 200 的是一组，code 值是 400 的是一组，code 值是 422 的是一组。如果我们希望按照 code 标签分组统计每个响应码多少数据，可以使用如下表达式：

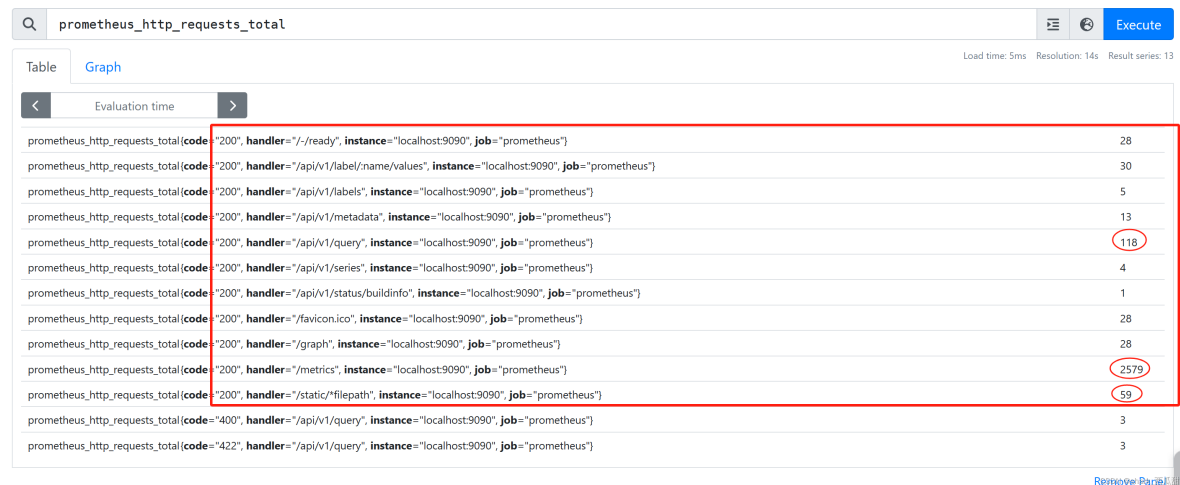
```
sum by(code) (prometheus_http_requests_total)
```



CSDN @shark_西瓜甜

再看一个 `topk` 的例子，它需要一个参数，用于从结果中筛选出值最高的前几名。`by` 的作用，在这里会让 `by` 指定的标签值相同的数据进行排列比较。

下图的结果可以看出，`code` 标签的值是 200 的前 3 名数据是 **2579, 118, 59**，`code` 的值是 400 和 422 的分别只有一条数据。所以前 3 名也是这一条数据会出现。



如果我们线下希望统计每个返回码的请求次数前 3 名，可以使用如下表达式。

```
topk by(code) (3,prometheus_http_requests_total)
```



`without(label1)` 进行聚合函数计算的时候，不考虑 `label1` 这个标签，而只会按照 `label1` 之外的其他值相同的标签进行分组计算。

还是使用上面图片中的数据进行计算。使用 `without` 需要使用如下表达式，会得到一样的结果。

```
sum without(handler,instance,job) (prometheus_http_requests_total)
```

