1. 函数语法解释

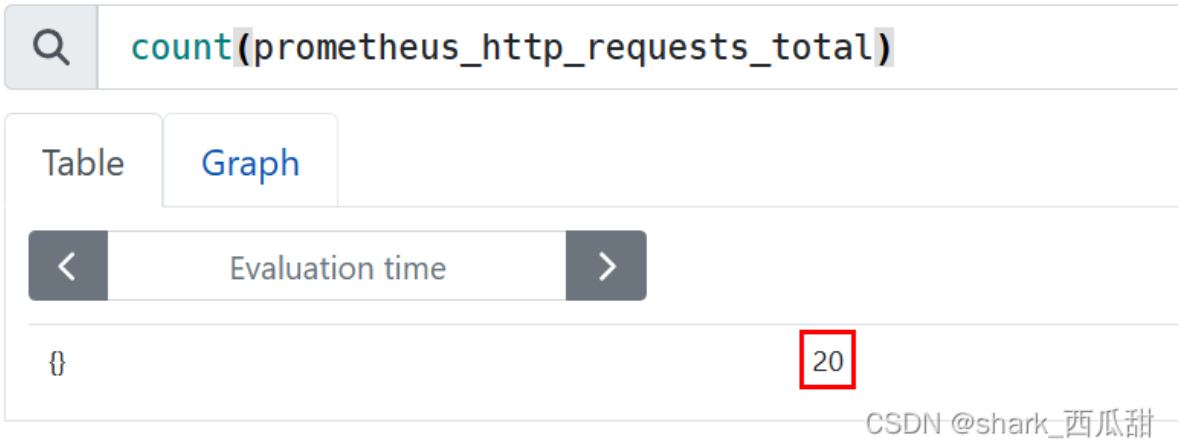
Promethues 的函数作用于 及时向量和范围向量。

举例说明, 函数 `increase` 的语法为： `increase(v range-vector)`，其中 `increase` 为函数， `v` 为函数的参数，就是一个查询表达式获取到的具体数值， `range-vector` 是参数 `v` 的类型， 基本上类型有:

- `instant-vector` 即时向量，就是当前时间的数据，例如 `up`
- `range-vector` 范围向量, 就是一个指定范围内的数据，例如 `up[5m]`

2. count(v instant-vector)

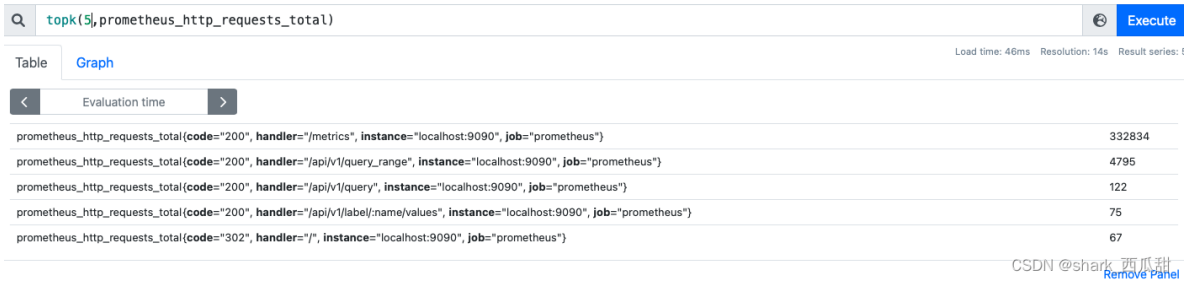
统计获取到结果的条目数



3. topk(n, v instant-vector)

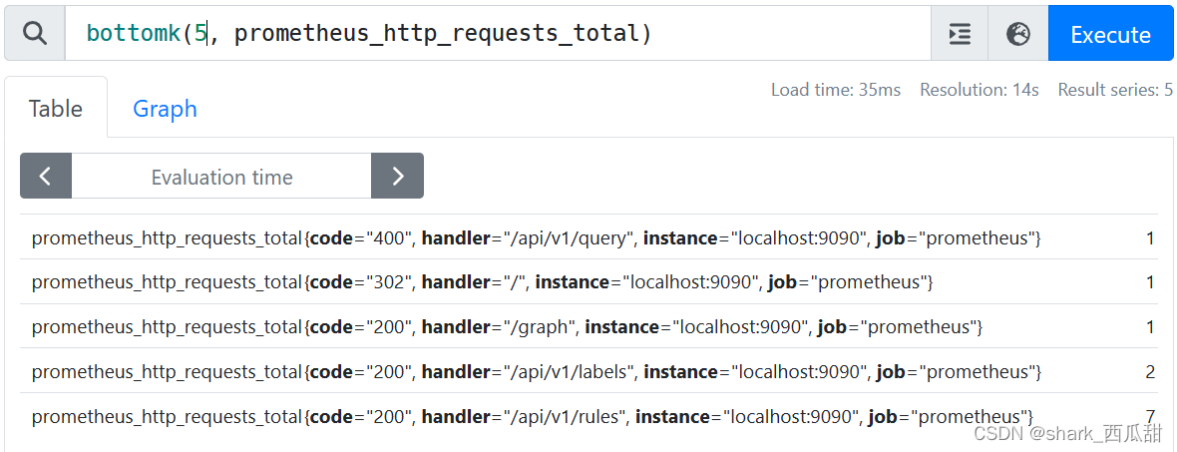
从给定的一组数值中取出最高的前 n 位, v 应该是聚合表达式中的 即时向量。

常用于瞬时报警，不用于观察曲线图



4. bottomk(n, v instant-vector)

从给定的一组数值中取出最低的后 n 位, v 应该是聚合表达式中的 即时向量。



5. increase(v range-vector)

计算范围向量 `v` 中时间序列的增长量。就是计算在一定时间范围内，某个监控指标的值总共增长了多少。

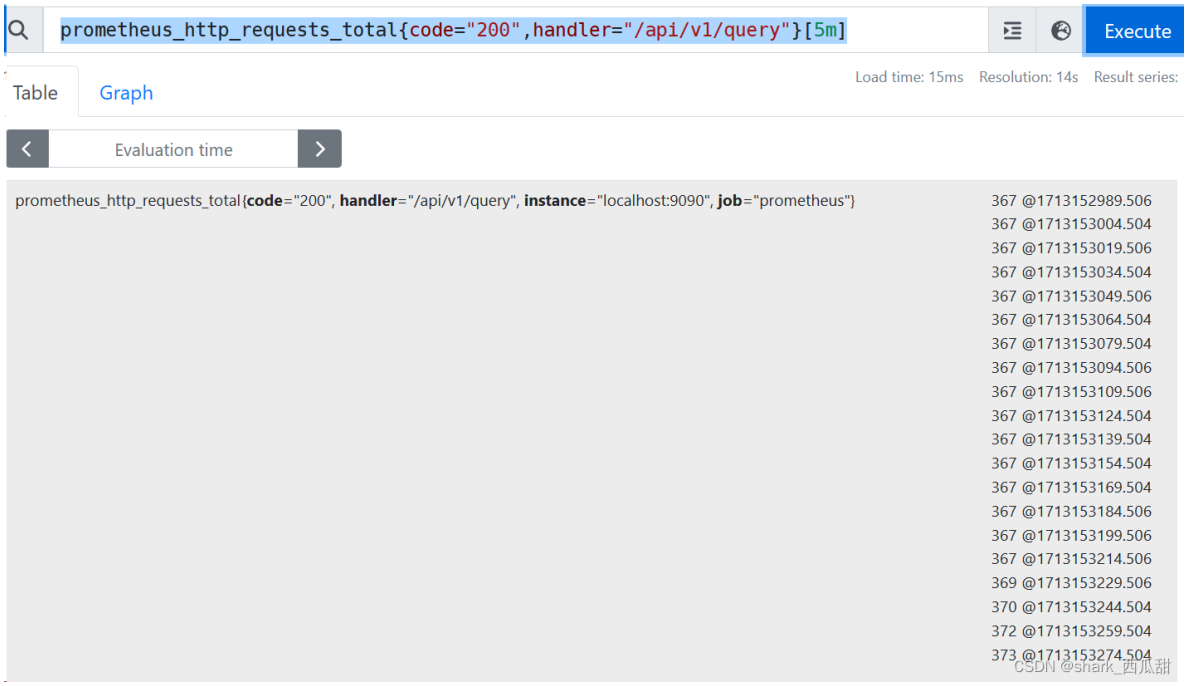
比如我在 13:20 统计 5分钟内某个指标的增加量是 10，表示 5分钟之前 13:15 到 13:20，这个指标的值增加了 5 个，如果一开始指标的值是 3，那个 13:20 的时候这个值应该是 8。

函数 **increase**只能与随时间而不断变化的数据一起使用。

示例：希望获取 最近 5分钟内，发送给 Prometheus URL "/api/v1/query" 的 HTTP 请求且返回成功的 请求，总共增长了多少个。

为了对比，首先查询 Prometheus 最近 5 分钟的数据。

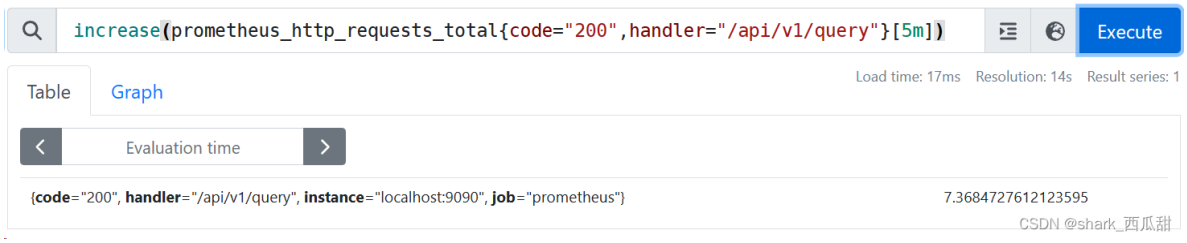
```
prometheus_http_requests_total{code="200",handler="/api/v1/query"}[5m]
```



之后再真正计算请求增长量：

```
increase(prometheus_http_requests_total{code="200",handler="/api/v1/query"}[5m])
```

367 到 373 ， 共有 7个



6. rate(v range-vector)

计算范围向量中时间序列的每秒平均增长率。

以下示例表达式返回范围向量中的每个时间序列在过去 1 分钟内的每秒请求监控指标数据的速率：

```
rate(promhttp_metric_handler_requests_total[1m])
```

> (60s / 15s) / 60s = 0.066/s
> 每秒增加 0.066 次请求
> 15 s 是全局设置中查询指标的间隔时间 scrape_interval 的值

函数 **rate** 最适合于报警，比如请求错误增长率，磁盘使用增长率等。

请注意，当将 `rate()` 与聚合运算符（例如 `sum()`）或随时间聚合的函数（任何以 `_over_time` 结尾的函数）组合时，始终先使用 `rate()`，然后再聚合。否则，当目标重新启动时，`rate()` 无法检测计数器重置。

它的语法糖是使用如下公式计算的：

`最近指定时间范围内比之前增加了多少量 / 时间范围内的总秒数 = 增加了多少量/秒`

7. rate 和 increase

- `rate` 适合用于数据采集频率高的数据，比如 CPU、内存、磁盘 IO，网络 IO
- `increase` 适合数据采集频率低的数据，比如每 5 分钟采集一次数据。
  这两个函数比较适合使用在监控上，比较重要。
- `rate` 计算的是一定时间内每秒的增长量
- `increase` 计算的是一定时间内增量的总和

`rate` 和 `increase` 都有首尾效应，如果希望更精确，可以使用 `irate`，但 `irate` 不适合分析长期趋势的数据，也不适合在告警规则中使用，告警规则而应该使用 `rate`。

8. irate(v range-vector)

计算的是取在一个指定时间范围内开头和结尾的两个数据进行计算每秒增长率，不适合分析长期趋势，也不适合用作告警规则中使用。
`irate(v range-vector)` 计算范围向量中时间序列的每秒即时增长率。这是基于最后两个数据点。单调的中断（例如目标重新启动导致的计数器重置）会自动调整。例如：

时间序列（转换成具体时间）	**数据**
12:00	0
12:05	50
12:10	中断
12: 15	150

上面表格中 4 个时间点，其中 1 个时间点服务中断了，没有数据，使用 `irate` 计算的话，会取 0 和 150 进行计算，总共增长了 150 个，在用 150 除以 15分钟 * 60 秒= 0.5个 ，结果增长率就是每秒增长 0.5 个。

示例：

```
irate(promhttp_metric_handler_requests_total[1m])
```

`irate()` 只能在绘制易波动、快速移动的 **counter** 时使用。将速率用于警报和缓慢移动的 **counter**，因为完全由罕见峰值组成的图形很难读取。

请注意，将 `irate()` 与聚合运算符（例如 `sum()`）或随时间聚合的函数（任何以 `_over_time` 结尾的函数）组合时，始终先使用 `irate`），然后再聚合。否则，`irate()` 无法在目标重新启动时检测计数器重置。

9. predict_linear(v range-vector, t scalar) 预测统计【重要】

可以通过一个范围向量 `v` 返回的数据样本，预测指标在 `t` 秒后的值。它基于简单线性回归的方式，对时间窗口内的样本数据进行统计，从而可以对时间序列的变化趋势做出预测。该函数的返回结果不带有指标名称，只有标签列表。

`通常用于对服务器磁盘使用率的预测。`

根据最近 1 天的根分区剩余容量的数据变化，预测 4 个小时之后 根分区的剩余容量是否会是10m。

```
predict_linear(node_filesystem_avail_bytes{mountpoint="/"}[1d], 4 * 3600) <
10485760
```

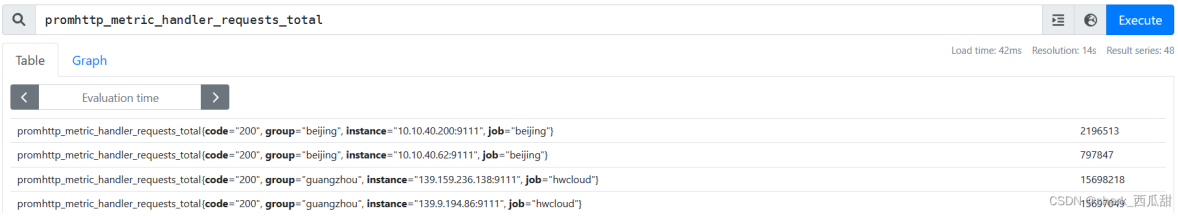
10. absent(v instant-vector)

用于判断写的语句是否有错误,或者是否返回了数据，就是编写的表达式如果能获取到数据，就返回空，表示表达式正确。否则就返回 1 ，就是表达式存在问题，比如表达式书写错误，或者要统计的数据没有匹配到，或者不符合判断条件。若果被采集数据的服务器挂了，那也会返回 1。

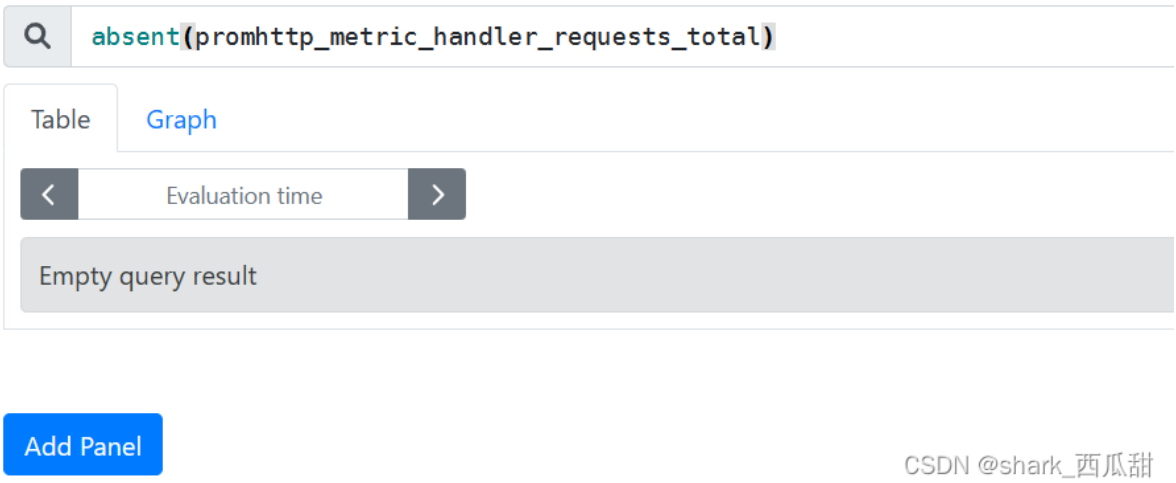
一般用于判断数据是否在正常采集中。

当监控度量指标时，如果获取到的样本数据是空的， 使用 `absent` 方法对告警是非常有用的。例如：

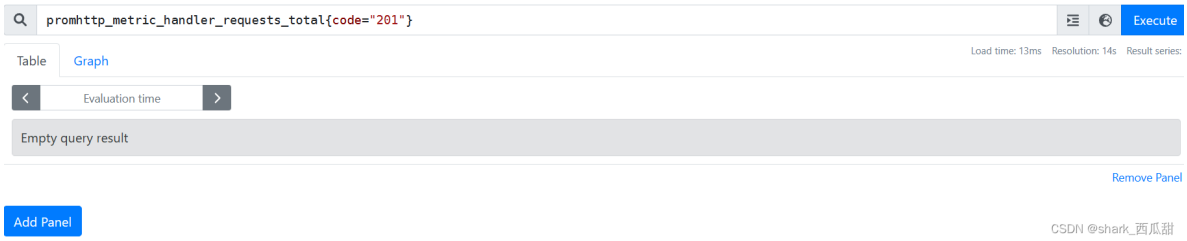
原始数据



使用 `absent` 后返回空数据



在原始数据中并没有标签 `code` 为 `201` 的数据



使用 `absent` 后返回值 1

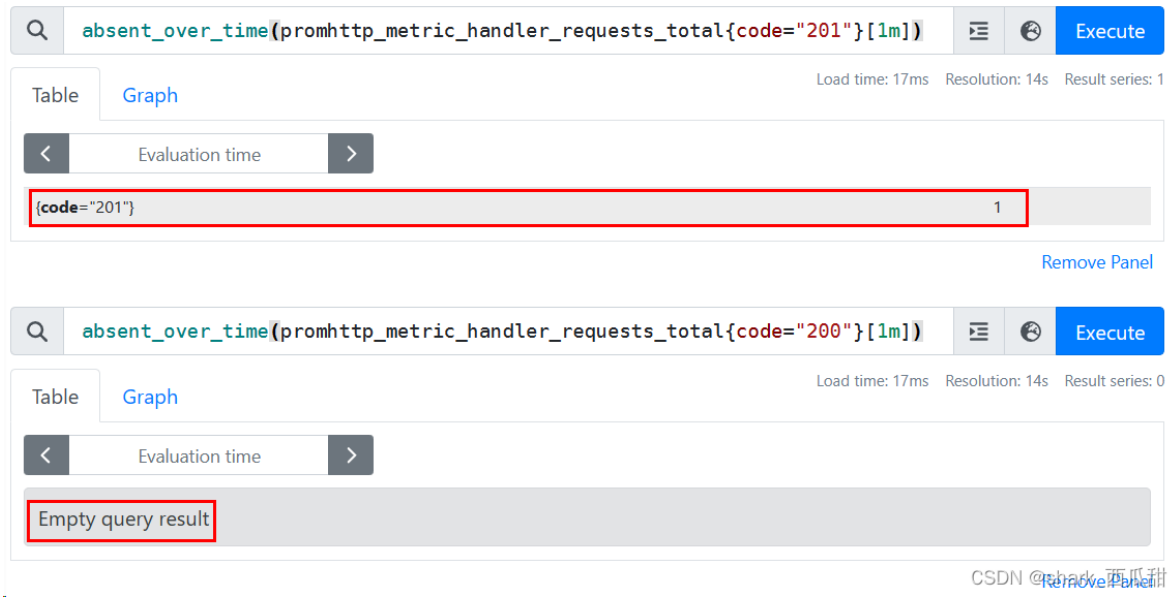
11. absent_over_time(v range-vector)

`absent_over_time(v range-vector)` 如果传递给它的范围向量具有任何数据，则返回一个空向量，如果传递给它的范围向量没有数据，则返回值为 1。

这对于在给定指标名称和标签组合在一段时间内不存在时发出警报非常有用。

```
absent_over_time(promhttp_metric_handler_requests_total{code="201"}[1m])
# => {code="201"}  1

absent_over_time(promhttp_metric_handler_requests_total{code="200"}[1m])
# => 空
```



12. histogram_quantile(φ scalar, b instant-vector)

根据经典直方图或原生直方图计算φ-分位数（0≤φ≤1）

`b` 中的浮点样本被认为是一个或多个经典直方图的每个桶中的观测计数。每个浮点样本必须有一个标签 `le`，其中标签值表示 `bucket` 的包含上限。（没有此类标签的浮点样本将被忽略。）其他标签和度量名称用于识别属于每个经典直方图的桶。直方图度量类型自动提供具有 `_bucket` 后缀和适当标签的时间序列。
`b` 中的原生直方图样本被单独处理为单独的直方图，以从中计算分位数。

只要不出现命名冲突，`b` 就可以包含经典直方图和原生直方图的混合。

使用 `rate()` 该函数可以控制分位数计算的时间在一定范围内。

示例：直方图度量称为 `http_request_duration_seconds`。要计算过去10分钟内请求持续时间的第90个百分点，请使用以下表达式：

```
histogram_quantile(0.9, rate(http_request_duration_seconds_bucket[10m]))
```

以 `http_request_duration_seconds` 为单位计算每个标签组合的分位数。要聚合，请在 `rate()` 函数周围使用 `sum()` 聚合器。由于 `histogram_quantile()` 需要 `le` 标签，因此它必须包含在 `by` 子句中。以下表达式按 `job` 聚合第 ***90*** 个百分位。

```
histogram_quantile(0.9, sum by (job, le)
(rate(http_request_duration_seconds_bucket[10m])))
```

要聚合所有内容，请仅指定标签：`le`

```
histogram_quantile(0.9, sum by (le)
(rate(http_request_duration_seconds_bucket[10m])))
```

`histogram_quantile()` 函数通过假设桶内的线性分布来插值分位数值。最高存储桶的上限必须为 `+Inf`。（否则，返回 `NaN`。）如果分位数位于最高存储桶中，则返回第二高存储桶的上限。如果最低桶的上限大于 `0`，则假定最低桶的下限为 `0`。在这种情况下，在该桶内应用通常的线性插值。否则，对于位于最低存储桶中的分位数，将返回最低存储桶的上限。

如果 `b` 有 `0` 个观察值，则返回 `NaN`。如果 `b` 包含少于两个桶，则返回 `NaN`。对于 `ϕ<0`，将返回 `-Inf`。对于 `ϕ>1`，返回 `+Inf`。对于 `ϕ=NaN`，返回 `NaN`。

13. changes()

`changes(v range-vector)`

对于每个输入时间序列，以即时向量的形式返回其值在提供的时间范围内更改的次数。

14. 其他

14.1 label_join()

```
label_join(v instant vector, dst_label string, separator string, src_label_1
string,src_label_2 string, ...)
```

对于 `v` 中每个时间序列中含有的标签 `src_label`，使用逗号连接所有 `src_label` 的所有值，使用 `separator string` 指定的符号拼接起来， 并将拼接后的值赋值给标签 `dst_label`。此函数中可以有任意数量的 `src_labels`，最终返回的指标数据中 `src_label` 和 `dst_label` 都会在结果中。

示例：
原数据中含有标签 `instance`，`job`，`group`

```
up{job="beijing"}
=>
up{group="beijing", instance="10.10.40.200:9111", job="beijing"}    1
up{group="beijing", instance="10.10.40.61:9111", job="beijing"}     0
up{group="beijing", instance="10.10.40.62:9111", job="beijing"}     1
```

下面表达式的意思是 将返回的每条数据中标签 `"instance", "job"` 的值，使用引文逗号 `,` 进行拼接，之后将拼接好的值赋值给新的标签 `msg`

```
label_join(up{job="beijing"},"msg", ",", "instance", "job")
```

返回的结果:

```
up{group="beijing", instance="10.10.40.200:9111", job="beijing",
msg="10.10.40.200:9111,beijing"}    1
up{group="beijing", instance="10.10.40.61:9111", job="beijing",
msg="10.10.40.61:9111,beijing"}     0
up{group="beijing", instance="10.10.40.62:9111", job="beijing",
msg="10.10.40.62:9111,beijing"}     1
```

14.2 label_replace()

```
label_replace(v instant-vector, dst_label string, replacement string, src_label string, regex string)
```

对于 `v` 中的每个时间序列，将正则表达式 `regex` 与 `src_label` 的值匹配。
如果匹配，则返回的时间序列中标签 `dst_label` 的值将是 `replacement` 的扩展，以及输入中的原始标签。
在正则表达式中捕获组可以用 `$1`、`$2` 等引用。如果正则表达式不匹配，则返回时间序列不变。

示例：

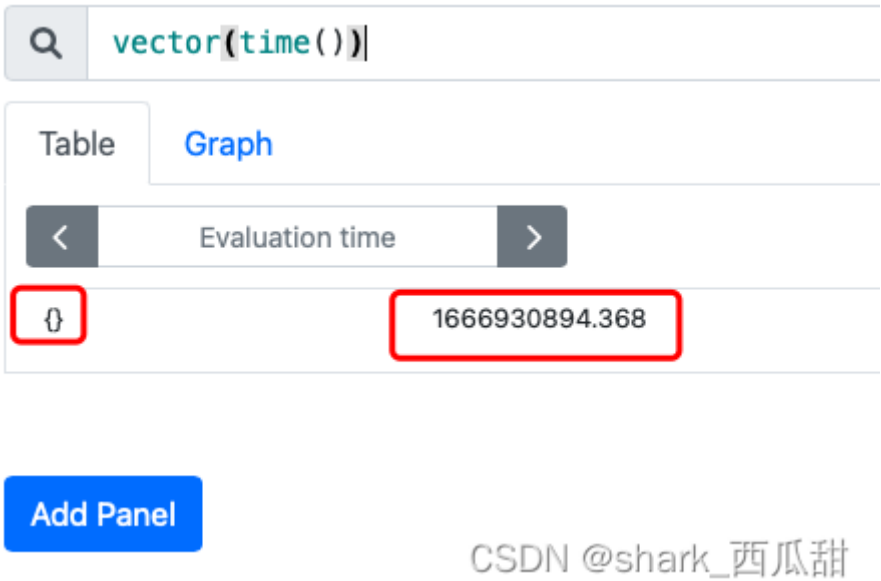
```
label_replace(up{job="beijing"},"msg", "服务器:$1", "instance", "(10.10.*)")
```

14.3 time()

返回自1970年1月1日以来的秒数。

14.4 vector(s scalar)

将标量s作为没有标签的向量返回。

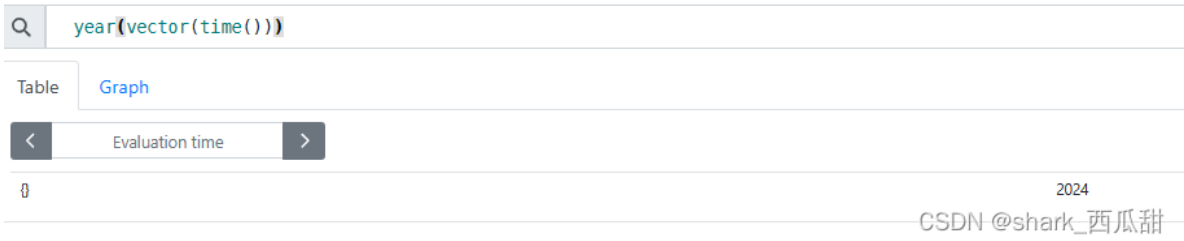


14.5 timestamp()

timestamp(v instant-vector)返回给定即时向量的每个样本的当前时间戳，自世界时1970年1月1日以来的秒数。

14.6 year(v=vector(time()) instant-vector)

以UTC为单位返回给定时间的年份。



14.7 day_of_week()

一周的星期几

`day_of_week(v=vector(time()) instant-vector)`

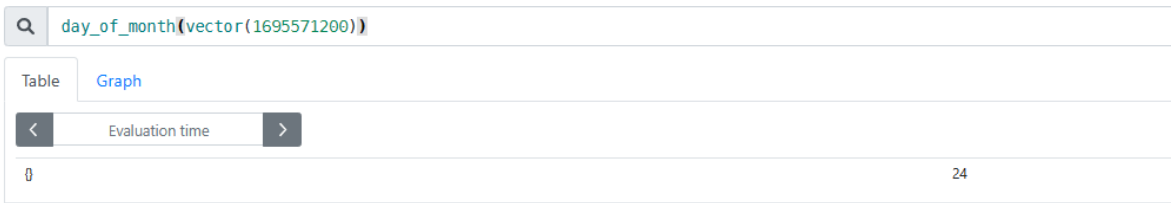
以 UTC 格式返回每个给定时间的星期几。返回值从 0 到 6，其中 0 表示星期日等。

14.8 day_of_month()

一个月中的哪一天

`day_of_month(v=vector(time()) instant-vector)`

以 UTC 格式返回每个给定时间的月份中的某一天。返回值介于 1 到 31 之间。



Add Panel

14.9 day_of_year(v=vector(time()) instant-vector)

一年中的哪一天

以 UTC 格式返回每个给定时间的一年中的日期。非闰年的返回值介于 1 到 365 之间，在闰年中，返回值为 1 到 366。

14.10 days_in_month(v=vector(time()) instant-vector)

返回某个月有多少天
返回当月中每个给定时间的 UTC 天数。返回值介于 28 到 31 之间。

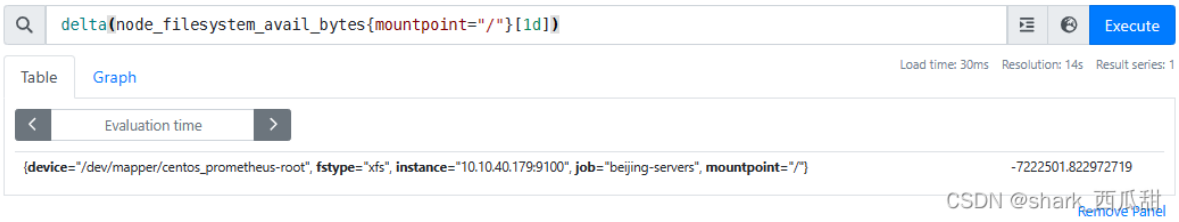
14.11 delta(v range-vector) 计算差值

计算范围向量中每个时间序列元素的第一个值和最后一个值之间的差值，返回一个具有给定增量和等效标签的即时向量。
增量被外推以覆盖范围向量选择器中指定的全部时间范围，因此即使样本值都是整数，也可以获得非整数结果。v

以下示例表达式返回从现在到 24 小时前的磁盘根分区的可用容量差：

```
delta(node_filesystem_avail_bytes{mountpoint="/"}[1d])
```

计算公式: `当前时间的值 - 过去时间的值`
可用容量少了 0.68MB左右。



14.12 idelta(v range-vector)

计算范围向量v中最后两个样本之间的差值，返回具有给定delta和等效标签的即时向量。

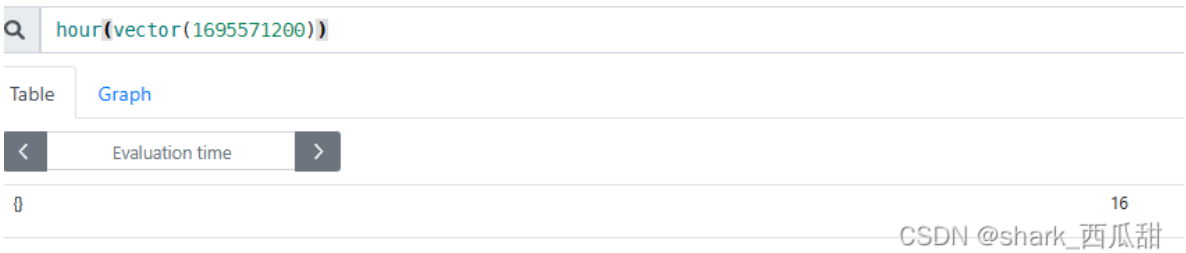
idelta只能与仪表一起使用

14.13 minute(v=vector(time()) instant-vector)

以UTC为单位返回每个给定时间的一小时内的几分钟。返回的值从0到59。

14.14 hour(v=vector(time()) instant-vector)

返回 UTC 中每个给定时间的一天中的小时数。返回值介于 0 到 23 之间。



14.15 month(v=vector(time()) instant-vector)

返回UTC中每个给定时间的月份。返回值从1到12，其中1表示1月等。

14.16 sum(v instant-vector)

把所有返回的指标的值加起来

14.17 sgn(v instant-vector)

返回一个向量，所有采样值都转换为其符号，定义如下：如果v为正，则为1；如果v为负，则为-1；如果v等于零，则为0。
可以用来判断指标的值是正数、负数或零。

14.18 sort(v instant-vector)

按照从小到大对返回的即时向量值排序。升序

14.19 sort_desc(v instant-vector)

按照从大到小对返回的即时向量值排序。降序

14.20 ceil(v instant-vector) 不常用

将返回的所有值向上舍入到最接近的整数。
2.77 =》 3

14.21 floor(v instant-vector) 不常用

将 v 所有元素的样本值向下舍入为最接近的整数。
2.77 =》 2

15. `<aggregation>_over_time()`

以下函数允许随着时间的推移聚合给定范围向量的每个系列，并返回具有每个序列聚合结果的即时向量：

- `avg_over_time(range-vector)`：指定区间内所有点的平均值。
- `min_over_time(range-vector)`：指定区间内所有点的最小值。
- `max_over_time(range-vector)`：指定区间内所有点的最大值。
- `sum_over_time(range-vector)`：指定区间中所有值的总和。
- `count_over_time(range-vector)`：指定区间内所有值的计数。
- `quantile_over_time(scalar, range-vector)`：指定区间中值的φ-quantile（0 ≤ φ ≤ 1）。
- `stddev_over_time(range-vector)`：指定区间内值的总体标准差。
- `stdvar_over_time(range-vector)`：指定区间中值的总体标准方差。
- `last_over_time(range-vector)`：指定区间内最近时间的值，一般就是最后一个值距离当前时间最近。