# PromQL内置函数

Prometheus 提供了其它大量的内置函数，可以对时序数据进行丰富的处理。某些函数有默认的参数，例如：`year(v=vector(time()) instant-vector)`。其中参数 `v` 是一个瞬时向量，如果不提供该参数，将使用默认值 `vector(time())`。instant-vector 表示参数类型。

## 1. abs()

`abs(v instant-vector)` 返回输入向量的所有样本的绝对值。

## 2. absent()

`absent(v instant-vector)`，如果传递给它的向量参数具有样本数据，则返回空向量；如果传递的向量参数没有样本数据，则返回不带度量指标名称且带有标签的时间序列，且样本值为1。

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：

```
# 这里提供的向量有样本数据
absent(http_requests_total{method="get"})  => no data
absent(sum(http_requests_total{method="get"}))  => no data

# 由于不存在度量指标 nonexistent，所以 返回不带度量指标名称且带有标签的时间序列，且样本值为1
absent(nonexistent{job="myjob"})  => {job="myjob"}  1
# 正则匹配的 instance 不作为返回 labels 中的一部分
absent(nonexistent{job="myjob",instance=~".*"})  => {job="myjob"}  1

# sum 函数返回的时间序列不带有标签，且没有样本数据
absent(sum(nonexistent{job="myjob"}))  => {}  1
```

## 3. ceil()

`ceil(v instant-vector)` 将 v 中所有元素的样本值向上四舍五入到最接近的整数。例如：

```
node_load5{instance="192.168.1.75:9100"} # 结果为 2.79
ceil(node_load5{instance="192.168.1.75:9100"}) # 结果为 3
```

## 4. changes()

`changes(v range-vector)` 输入一个区间向量， 返回这个区间向量内每个样本数据值变化的次数（瞬时向量）。例如：

```
# 如果样本数据值没有发生变化，则返回结果为 1
changes(node_load5{instance="192.168.1.75:9100"}[1m]) # 结果为 1
```

## 5. clamp_max()

`clamp_max(v instant-vector, max scalar)` 函数，输入一个瞬时向量和最大值，样本数据值若大于 max，则改为 max，否则不变。例如：

```
node_load5{instance="192.168.1.75:9100"} # 结果为 2.79
clamp_max(node_load5{instance="192.168.1.75:9100"}, 2) # 结果为 2
```

# 6. clamp_min()

`clamp_min(v instant-vector, min scalar)` 函数，输入一个瞬时向量和最小值，样本数据值若小于 min，则改为 min，否则不变。例如：

```
node_load5{instance="192.168.1.75:9100"} # 结果为 2.79
clamp_min(node_load5{instance="192.168.1.75:9100"}, 3) # 结果为 3
```

# 7. day_of_month()

`day_of_month(v=vector(time()) instant-vector)` 函数，返回被给定 UTC 时间所在月的第几天。返回值范围：1~31。

# 8. day_of_week()

`day_of_week(v=vector(time()) instant-vector)` 函数，返回被给定 UTC 时间所在周的第几天。返回值范围：0~6，0 表示星期天。

# 9. days_in_month()

`days_in_month(v=vector(time()) instant-vector)` 函数，返回当月一共有多少天。返回值范围：28~31。

# 10. delta()

`delta(v range-vector)` 的参数是一个区间向量，返回一个瞬时向量。它计算一个区间向量 v 的第一个元素和最后一个元素之间的差值。由于这个值被外推到指定的整个时间范围，所以即使样本值都是整数，你仍然可能会得到一个非整数值。

例如，下面的例子返回过去两小时的 CPU 温度差：

```
delta(cpu_temp_celsius{host="zeus"}[2h])
```

这个函数一般只用在 Gauge 类型的时间序列上。

# 11. deriv()

`deriv(v range-vector)` 的参数是一个区间向量,返回一个瞬时向量。它使用简单的线性回归计算区间向量 v 中各个时间序列的导数。

这个函数一般只用在 Gauge 类型的时间序列上。

# 12. exp()

`exp(v instant-vector)` 函数，输入一个瞬时向量，返回各个样本值的 `e` 的指数值，即 e 的 N 次方。当 N 的值足够大时会返回 `+Inf`。特殊情况为：

- `Exp(+Inf) = +Inf`

- `Exp(NaN) = NaN`

# 13. floor()

`floor(v instant-vector)` 函数与 ceil() 函数相反，将 v 中所有元素的样本值向下四舍五入到最接近的整数。

# 14. histogram_quantile()

`histogram_quantile(φ float, b instant-vector)` 从 bucket 类型的向量 `b` 中计算 φ (0 ≤ φ ≤ 1) 分位数（百分位数的一般形式）的样本的最大值。（有关 φ 分位数的详细说明以及直方图指标类型的使用，请参阅直方图和摘要）。向量 `b` 中的样本是每个 bucket 的采样点数量。每个样本的 labels 中必须要有 `le` 这个 label 来表示每个 bucket 的上边界，没有 `le` 标签的样本会被忽略。直方图指标类型自动提供带有 `_bucket` 后缀和相应标签的时间序列。

可以使用 `rate()` 函数来指定分位数计算的时间窗口。

例如，一个直方图指标名称为 `employee_age_bucket_bucket`，要计算过去 10 分钟内 第 90 个百分位数，请使用以下表达式：

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

返回：

```
{instance="10.0.86.71:8080",job="prometheus"} 35.714285714285715
```

这表示最近 10 分钟之内 90% 的样本的最大值为 35.714285714285715。

这个计算结果是每组标签组合成一个时间序列。我们可能不会对所有这些维度（如 `job`、`instance` 和 `method`）感兴趣，并希望将其中的一些维度进行聚合，则可以使用 sum() 函数。例如，以下表达式根据 `job` 标签来对第 90 个百分位数进行聚合：

```
# histogram_quantile() 函数必须包含 le 标签
histogram_quantile(0.9, sum(rate(employee_age_bucket_bucket[10m])) by (job, le))
```

如果要聚合所有的标签，则使用如下表达式：

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

> **注意**
>
> `histogram_quantile` 这个函数是根据假定每个区间内的样本分布是线性分布来计算结果值的(也就是说它的结果未必准确)，最高的 bucket 必须是 le="+Inf" (否则就返回 NaN)。
>
> 如果分位数位于最高的 bucket（+Inf） 中，则返回第二个最高的 bucket 的上边界。如果该 bucket 的上边界大于 0，则假设最低的 bucket 的的下边界为 0，这种情况下在该 bucket 内使用常规的线性插值。
>
> 如果分位数位于最低的 bucket 中，则返回最低 bucket 的上边界。

如果 b 含有少于 2 个 buckets，那么会返回 `NaN`，如果 φ < 0 会返回 `-Inf`，如果 φ > 1 会返回 `+Inf`。

# 15. holt_winters()

`holt_winters(v range-vector, sf scalar, tf scalar)` 函数基于区间向量 v，生成时间序列数据平滑值。平滑因子 `sf` 越低, 对旧数据的重视程度越高。趋势因子 `tf` 越高，对数据的趋势的考虑就越多。其中，`0< sf, tf <=1`。

holt_winters 仅适用于 Gauge 类型的时间序列。

# 16. hour()

`hour(v=vector(time()) instant-vector)` 函数返回被给定 UTC 时间的当前第几个小时，时间范围：0~23。

# 17. idelta()

idelta(v range-vector) 的参数是一个区间向量, 返回一个瞬时向量。它计算最新的 2 个样本值之间的差值。

这个函数一般只用在 Gauge 类型的时间序列上。

# 18. increase()

`increase(v range-vector)` 函数获取区间向量中的第一个和最后一个样本并返回其增长量, 它会在单调性发生变化时(如由于采样目标重启引起的计数器复位)自动中断。由于这个值被外推到指定的整个时间范围，所以即使样本值都是整数，你仍然可能会得到一个非整数值。

例如，以下表达式返回区间向量中每个时间序列过去 5 分钟内 HTTP 请求数的增长数：

```
increase(http_requests_total{job="apiserver"}[5m])
```

`increase` 的返回值类型只能是计数器类型，主要作用是增加图表和数据的可读性。使用 `rate` 函数记录规则的使用率，以便持续跟踪数据样本值的变化。

# 19. irate()

`irate(v range-vector)` 函数用于计算区间向量的增长率，但是其反应出的是瞬时增长率。irate 函数是通过区间向量中最后两个两本数据来计算区间向量的增长速率，它会在单调性发生变化时(如由于采样目标重启引起的计数器复位)自动中断。这种方式可以避免在时间窗口范围内的“长尾问题”，并且体现出更好的灵敏度，通过irate函数绘制的图标能够更好的反应样本数据的瞬时变化状态。

例如，以下表达式返回区间向量中每个时间序列过去 5 分钟内最后两个样本数据的 HTTP 请求数的增长率：

```
irate(http_requests_total{job="api-server"}[5m])
```

irate 只能用于绘制快速变化的计数器，在长期趋势分析或者告警中更推荐使用 rate 函数。因为使用 irate 函数时，速率的简短变化会重置 `FOR` 语句，形成的图形有很多波峰，难以阅读。

> **注意**
>
> 当将 `irate()` 函数与聚合运算符（例如 `sum()`）或随时间聚合的函数（任何以 `_over_time` 结尾的函数）一起使用时，必须先执行 irate 函数，然后再进行聚合操作，否则当采样目标重新启动时 irate() 无法检测到计数器是否被重置。

## 20. label_join()

`label_join(v instant-vector, dst_label string, separator string, src_label_1 string, src_label_2 string, ...)` 函数可以将时间序列 v 中多个标签 `src_label` 的值，通过 `separator` 作为连接符写入到一个新的标签 `dst_label` 中。可以有多个 src_label 标签。

例如，以下表达式返回的时间序列多了一个 `foo` 标签，标签值为 `etcd,etcd-k8s`：

```
up{endpoint="api",instance="192.168.123.248:2379",job="etcd",namespace="monitoring",service="etcd-k8s"}
=>
up{endpoint="api",instance="192.168.123.248:2379",job="etcd",namespace="monitoring",service="etcd-k8s"}  1

label_join(up{endpoint="api",instance="192.168.123.248:2379",job="etcd",namespace="monitoring",service="etcd-k8s"}, "foo", ",", "job", "service")
=> up{endpoint="api",foo="etcd,etcd-k8s",instance="192.168.123.248:2379",job="etcd",namespace="monitoring",service="etcd-k8s"}  1
```

## 21. label_replace()

为了能够让客户端的图标更具有可读性，可以通过 `label_replace` 函数为时间序列添加额外的标签。label_replace 的具体参数如下：

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

该函数会依次对 v 中的每一条时间序列进行处理，通过 `regex` 匹配 src_label 的值，并将匹配部分 `relacement` 写入到 dst_label 标签中。如下所示：

```
label_replace(up, "host", "$1", "instance",  "(.*):.*")
```

函数处理后，时间序列将包含一个 `host` 标签，host 标签的值为 Exporter 实例的 IP 地址：

```
up{host="localhost",instance="localhost:8080",job="cadvisor"}   1
up{host="localhost",instance="localhost:9090",job="prometheus"}   1
up{host="localhost",instance="localhost:9100",job="node"}   1
```

## 22. ln()

`ln(v instant-vector)` 计算瞬时向量 v 中所有样本数据的自然对数。特殊情况：

- `ln(+Inf) = +Inf`
- `ln(0) = -Inf`
- `ln(x < 0) = NaN`
- `ln(NaN) = NaN`

## 23. log2()

`log2(v instant-vector)` 函数计算瞬时向量 v 中所有样本数据的二进制对数。特殊情况同上。

## 24. log10()

`log10(v instant-vector)` 计算瞬时向量 v 中所有样本数据的十进制对数。特殊情况同上。

## 25. minute()

`minute(v=vector(time()) instant-vector)` 函数返回给定 UTC 时间当前小时的第多少分钟。结果范围：0~59。

## 26. month()

`month(v=vector(time()) instant-vector)` 函数返回给定 UTC 时间当前属于第几个月，结果范围：0~12。

## 27. predict_linear()

`predict_linear(v range-vector, t scalar)` 函数可以预测时间序列 v 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**不带有度量指标**，只有标签列表。

例如，基于 2 小时的样本数据，来预测主机可用磁盘空间的是否在 4 个小时候被占满，可以使用如下表达式：

```
predict_linear(node_filesystem_free{job="node"}[2h], 4 * 3600) < 0
```

通过下面的例子来观察返回值：

```
predict_linear(http_requests_total{code="200",instance="120.77.65.193:9090",job="prometheus",method="get"}[5m], 5)
结果：
{code="200",handler="query_range",instance="120.77.65.193:9090",job="prometheus",method="get"}  1
{code="200",handler="prometheus",instance="120.77.65.193:9090",job="prometheus",method="get"}   4283.449995397104
{code="200",handler="static",instance="120.77.65.193:9090",job="prometheus",method="get"}   22.99999999999999
...
```

这个函数一般只用在 Gauge 类型的时间序列上。

## 28. rate()

`rate(v range-vector)` 函数可以直接计算区间向量 v 在时间窗口内平均增长速率，它会在单调性发生变化时(如由于采样目标重启引起的计数器复位)自动中断。该函数的返回结果**不带有度量指标**，只有标签列表。

例如，以下表达式返回区间向量中每个时间序列过去 5 分钟内 HTTP 请求数的每秒增长率：

```
rate(http_requests_total[5m])
结果:
{code="200",handler="label_values",instance="120.77.65.193:9090",job="prometheus
",method="get"} 0
{code="200",handler="query_range",instance="120.77.65.193:9090",job="prometheus"
,method="get"}  0
{code="200",handler="prometheus",instance="120.77.65.193:9090",job="prometheus",
method="get"}   0.2
...
```

rate() 函数返回值类型只能用计数器，在长期趋势分析或者告警中推荐使用这个函数。

> **注意**
>
> 当将 `rate()` 函数与聚合运算符（例如 `sum()`）或随时间聚合的函数（任何以 `_over_time` 结尾的函数）一起使用时，必须先执行 rate 函数，然后再进行聚合操作，否则当采样目标重新启动时 rate() 无法检测到计数器是否被重置。

# 29. resets()

`resets(v range-vector)` 的参数是一个区间向量。对于每个时间序列，它都返回一个计数器重置的次数。两个连续样本之间的值的减少被认为是一次计数器重置。

这个函数一般只用在计数器类型的时间序列上。

# 30. round()

`round(v instant-vector, to_nearest=1 scalar)` 函数与 `ceil` 和 `floor` 函数类似，返回向量中所有样本值的最接近的整数。`to_nearest` 参数是可选的,默认为 1,表示样本返回的是最接近 1 的整数倍的值。你也可以将该参数指定为任意值（也可以是小数），表示样本返回的是最接近它的整数倍的值。

# 31. scalar()

`scalar(v instant-vector)` 函数的参数是一个单元素的瞬时向量,它返回其唯一的时间序列的值作为一个标量。如果度量指标的样本数量大于 1 或者等于 0, 则返回 `NaN`。

# 32. sort()

`sort(v instant-vector)` 函数对向量按元素的值进行升序排序，返回结果：key: value = 度量指标: 样本值[升序排列]。

# 33. sort_desc()

`sort(v instant-vector)` 函数对向量按元素的值进行降序排序，返回结果：key: value = 度量指标: 样本值[降序排列]。

# 34. sqrt()

`sqrt(v instant-vector)` 函数计算向量 v 中所有元素的平方根。

# 35. time()

`time()` 函数返回从 1970-01-01 到现在的秒数。注意：它不是直接返回当前时间，而是时间戳

## 36. timestamp()

`timestamp(v instant-vector)` 函数返回向量 v 中的每个样本的时间戳（从 1970-01-01 到现在的秒数）。

该函数从 Prometheus 2.0 版本开始引入。

## 37. vector()

`vector(s scalar)` 函数将标量 s 作为没有标签的向量返回，即返回结果为：key: value= {}, s。

## 38. year()

`year(v=vector(time()) instant-vector)` 函数返回被给定 UTC 时间的当前年份。

## 39. _over_time()

下面的函数列表允许传入一个区间向量，它们会聚合每个时间序列的范围，并返回一个瞬时向量：

- `avg_over_time(range-vector)`：区间向量内每个度量指标的平均值。
- `min_over_time(range-vector)`：区间向量内每个度量指标的最小值。
- `max_over_time(range-vector)`：区间向量内每个度量指标的最大值。
- `sum_over_time(range-vector)`：区间向量内每个度量指标的求和。
- `count_over_time(range-vector)`：区间向量内每个度量指标的样本数据个数。
- `quantile_over_time(scalar, range-vector)`：区间向量内每个度量指标的样本数据值分位数，φ-quantile (0 ≤ φ ≤ 1)。
- `stddev_over_time(range-vector)`：区间向量内每个度量指标的总体标准差。
- `stdvar_over_time(range-vector)`：区间向量内每个度量指标的总体标准方差。

> **注意**
>
> 即使区间向量内的值分布不均匀，它们在聚合时的权重也是相同的。

```
# 一个指标的增长率
increase(http_request_total{endpoint="http",handler="/datasources/proxy/:id/*",instance="10.244.58.200:3000",job="grafana",method="get",namespace="monitoring",pod="grafana-86b55cb79f-fn4ss",service="grafana",statuscode="200"}[1h]) / 3600

rate(http_request_total{endpoint="http",handler="/datasources/proxy/:id/*",instance="10.244.58.200:3000",job="grafana",method="get",namespace="monitoring",pod="grafana-86b55cb79f-fn4ss",service="grafana",statuscode="200"}[1h])
# 长尾效应。
irate: 瞬时增长率，取最后两个数据进行计算
不适合做需要分期长期趋势或者在告警规则中使用。
rate

# 预测统计：
predict_linear(node_filesystem_files_free{mountpoint="/"}[1d], 4*3600) < 0
根据一天的数据，预测4个小时之后，磁盘分区的空间会不会小于0


absent()：如果样本数据不为空则返回no data，如果为空则返回1。判断数据是否在正常采集。
```

```
# 去除小数点：
    Ceil()：四舍五入，向上取最接近的整数，2.79 □ 3
    Floor：向下取， 2.79 □ 2

Delta()：差值

# 排序：
    Sort：正序
    Sort_desc：倒叙

Label_join：将数据中的一个或多个label的值赋值给一个新label
label_join(node_filesystem_files_free, "new_label", ",",  "instance",
"mountpoint")

label_replace：根据数据中的某个label值，进行正则匹配，然后赋值给新label并添加到数据中
label_replace(node_filesystem_files_free, "host","$2", "instance", "(.*)-(.*)")
```