数据模型

Prometheus会将所有采集到的样本数据以时间序列（time-series）的方式保存在内存数据库中。time-series是按照时间戳和值的序列顺序存放的，我们称之为向量(vector). 每条time-series通过指标名称(metrics name)和一组标签集(labelset)命名。如下所示，可以将time-series理解为一个以时间为Y轴的数字矩阵：

在time-series中的每一个点称为一个样本（sample），样本由以下三部分组成：

- **指标(metric)**：metric name和描述当前样本特征的labelsets;
- **时间戳(timestamp)**：一个精确到毫秒的时间戳;
- **样本值(value)**： 一个float64的浮点型数据表示当前样本的值。

```
<--------------- metric ---------------------><-timestamp -><-value->
http_request_total{status="200", method="GET"}@1434417560938 => 94355
http_request_total{status="200", method="GET"}@1434417561287 => 94334

http_request_total{status="404", method="GET"}@1434417560938 => 38473
http_request_total{status="404", method="GET"}@1434417561287 => 38544

http_request_total{status="200", method="POST"}@1434417560938 => 4748
http_request_total{status="200", method="POST"}@1434417561287 => 4785
```

在形式上，所有的指标(Metric)都通过如下格式标示：

```
<metric name>{<label name>=<label value>, ...}
```

Metrics类型

Prometheus定义了4种不同的指标类型(metric type)：**Counter（计数器）、Gauge（仪表盘）、Histogram（直方图）、Summary（摘要）。**

- Counter类型的指标其工作方式和计数器一样，只增不减（除非系统发生重置）。常见的监控指标，如prometheus_http_requests_total是Counter类型的监控指标。 一般在定义Counter类型指标的名称时推荐使用_total作为后缀。
- Gauge类型的指标侧重于反应系统的当前状态。因此这类指标的样本数据可增可减。常见指标如：node_memory_MemFree_bytes（主机当前空闲的内容大小）、node_memory_MemAvailable_bytes（可用内存大小）都是Gauge类型的监控指标。
- Histogram和Summary主用用于统计和分析样本的分布情况。

PromQL基础

Prometheus内置了一个强大的数据查询语言PromQL。 通过PromQL可以实现对监控数据的查询、聚合。

查询时间序列

当Prometheus通过Exporter采集到相应的监控指标样本数据后，我们就可以通过PromQL对监控样本数据进行查询。当我们直接使用监控指标名称查询时，可以查询该指标下的所有时间序列。

```
prometheus_http_requests_total
```

等价于：

```
prometheus_http_requests_total{}
```

会返回指标名称为prometheus_http_requests_total的全部时间序列。

prometheus_http_requests_total{code="200", container="prometheus", endpoint="web", handler="/", instance="192.168.225.49:9090", job="prometheus-k8s", namespace="monitoring", pod="prometheus-k8s-0", service="prometheus-k8s"}	**0**
prometheus_http_requests_total{**code**="200", **container**="prometheus", **endpoint**="web", **handler**="/-/healthy", **instance**="192.168.225.49:9090", **job**="prometheus-k8s", **namespace**="monitoring", **pod**="prometheus-k8s-0", **service**="prometheus-k8s"}	785
prometheus_http_requests_total{**code**="200", **container**="prometheus", **endpoint**="web", **handler**="/-/quit", **instance**="192.168.225.49:9090", **job**="prometheus-k8s", **namespace**="monitoring", **pod**="prometheus-k8s-0", **service**="prometheus-k8s"}	0
prometheus_http_requests_total{**code**="200", **container**="prometheus", **endpoint**="web", **handler**="/-/ready", **instance**="192.168.225.49:9090", **job**="prometheus-k8s", **namespace**="monitoring", **pod**="prometheus-k8s-0", **service**="prometheus-k8s"}	787

PromQL还支持用户根据时间序列的标签匹配模式来对时间序列进行过滤，目前主要支持两种匹配模式：完全匹配和正则匹配。

PromQL支持使用`=`和`!=`两种完全匹配模式：

- 通过使用`label=value`可以选择那些标签满足表达式定义的时间序列；
- 反之使用`label!=value`则可以根据标签匹配排除时间序列。

例如，如果我们只需要查询所有`prometheus_http_requests_total`时间序列中满足标签instance为192.168.225.49:9090的时间序列，则可以使用如下表达式：

```
prometheus_http_requests_total{instance="192.168.225.49:9090"}
```

PromQL还可以支持使用正则表达式作为匹配条件，多个表达式之间使用`|`进行分离：

- 使用`label=~regx`表示选择那些标签符合正则表达式定义的时间序列；
- 反之使用`label!~regx`进行排除;

例如，如果想查询多个环节下的时间序列序列可以使用如下表达式：

```
prometheus_http_requests_total{pod=~"prometheus-k8s-
0|prometheus",handler!="/alerts"}
```

范围查询

直接通过类似于PromQL表达式`prometheus_http_requests_total`查询时间序列时，返回值中只会包含该时间序列中的最新的一个样本值，这样的返回结果我们称之为**瞬时向量。**而相应的这样的表达式称之为**瞬时向量表达式。**

而如果我们想过去一段时间范围内的样本数据时，我们则需要使用**区间向量表达式。**区间向量表达式和瞬时向量表达式之间的差异在于在区间向量表达式中我们需要定义时间选择的范围，时间范围通过时间范围选择器`[]`进行定义。例如，通过以下表达式可以选择最近5分钟内的所有样本数据：

```
prometheus_http_requests_total{}[5m]
```

该表达式将会返回查询到的时间序列中最近5分钟的所有样本数据：

prometheus_http_requests_total{code="200", container="prometheus", endpoint="web", handler="/", instance="192.168.225.49:9090", job="prometheus-k8s", namespace="monitoring", pod="prometheus-k8s-0", service="prometheus-k8s"}	**0 @1714124094.065 0 @1714124124.065 0 @1714124154.065 0 @1714124184.065 0 @1714124214.065 0 @1714124244.065 0 @1714124274.065 0 @1714124304.065 0 @1714124334.065 0 @1714124364.065**

时间位移操作

如果我们想查询，5分钟前的瞬时样本数据，或昨天一天的区间内的样本数据，我们就可以使用位移操作，位移操作的关键字为**offset。**

可以使用offset时间位移操作：

```
prometheus_http_requests_total{} offset 5m
prometheus_http_requests_total{}[1d] offset 1d
```

聚合操作

描述样本特征的标签(label)在并非唯一的情况下，通过PromQL查询数据，会返回多条满足这些特征维度的时间序列。而PromQL提供的聚合操作可以用来对这些时间序列进行处理，形成一条新的时间序列：

```
# 查询系统所有普罗米修斯http请求的总量
sum(prometheus_http_requests_total)

# 按照mode计算主机CPU的平均使用时间
avg(node_cpu_seconds_total) by (mode)

# 按照主机查询各个主机的CPU使用率
sum(sum(irate(node_cpu_seconds_total{mode!='idle'}[5m]))  /
sum(irate(node_cpu_seconds_total[5m]))) by (instance)
```

标量和字符串

- 标量（Scalar）：一个浮点型的数字值，标量只有一个数字，没有时序。例如：`10`
- 字符串（String）：一个简单的字符串值，直接使用字符串，作为PromQL表达式，则会直接返回字符串。

合法的PromQL表达式

所有的PromQL表达式都必须至少包含一个指标名称(例如http_request_total)，或者一个不会匹配到空字符串的标签过滤器(例如{code="200"})。同时，除了使用`<metric name>{label=value}`的形式以外，我们还可以使用内置的`__name__`标签来指定监控指标名称：

```
{__name__=~"prometheus_http_request_total"} # 合法
{__name__=~"node_disk_read_bytes_total|node_disk_written_bytes_total"} # 合法
```

PromQL操作符

数学运算

例如，我们可以通过指标node_memory_MemFree_bytes获取当前主机可用的内存空间大小，其样本单位为Bytes。这是如果客户端要求使用MB作为单位响应数据，那只需要将查询到的时间序列的样本值进行单位换算即可：

```
node_memory_MemFree_bytes / (1024 * 1024)
```

node_memory_free_bytes_total表达式会查询出所有满足表达式条件的时间序列，称该表达式为瞬时向量表达式，而返回的结果成为瞬时向量。

当瞬时向量与标量之间进行数学运算时，数学运算符会依次作用域瞬时向量中的每一个样本值，从而得到一组新的时间序列。

瞬时向量与瞬时向量之间进行数学运算时，过程会相对复杂一点。 例如，如果我们想根据node_disk_read_bytes_total和node_disk_written_bytes_total获取主机磁盘IO的总量，可以使用如下表达式：

```
node_disk_read_bytes_total + node_disk_written_bytes_total
```

依次找到与左边向量元素匹配（标签完全一致）的右边向量元素进行运算，如果没找到匹配元素，则直接丢弃。同时新的时间序列将不会包含指标名称。 该表达式返回结果的示例如下所示：

{container="kube-rbac-proxy", device="mmcblk0", endpoint="https", instance="raspberrypi2", job="node-exporter", namespace="monitoring", pod="node-exporter-8st4n", service="node-exporter"}	**36676265984**
{**container**="kube-rbac-proxy", **device**="mmcblk0p1", **endpoint**="https", **instance**="raspberrypi2", **job**="node-exporter", **namespace**="monitoring", **pod**="node-exporter-8st4n", **service**="node-exporter"}	5895168
{**container**="kube-rbac-proxy", **device**="mmcblk0p2", **endpoint**="https", **instance**="raspberrypi2", **job**="node-exporter", **namespace**="monitoring", **pod**="node-exporter-8st4n", **service**="node-exporter"}	36669764608
{**container**="kube-rbac-proxy", **device**="mmcblk0", **endpoint**="https", **instance**="node", **job**="node-exporter", **namespace**="monitoring", **pod**="node-exporter-bd8ns", **service**="node-exporter"}	41763198976
{**container**="kube-rbac-proxy", **device**="mmcblk0p1", **endpoint**="https", **instance**="node", **job**="node-exporter", **namespace**="monitoring", **pod**="node-exporter-bd8ns", **service**="node-exporter"}	5895168
{**container**="kube-rbac-proxy", **device**="mmcblk0p2", **endpoint**="https", **instance**="node", **job**="node-exporter", **namespace**="monitoring", **pod**="node-exporter-bd8ns", **service**="node-exporter"}	41756697600

PromQL支持的所有数学运算符如下所示：

`+` **(加法)** `-` **(减法)** `*` **(乘法)** `/` **(除法)** `%` **(求余)** `^` **(幂运算)**

布尔运算过滤时间序列

在PromQL通过标签匹配模式，用户可以根据时间序列的特征维度对其进行查询。而布尔运算则支持用户根据时间序列中样本的值，对时间序列进行过滤。

例如，通过数学运算符我们可以很方便的计算出，当前所有主机节点的内存使用率：

```
(node_memory_MemTotal_bytes-node_memory_MemFree_bytes)/node_memory_MemTotal_bytes
```

而系统管理员在排查问题的时候可能只想知道当前内存使用率超过70%的主机呢？通过使用布尔运算符可以方便的获取到该结果：

```
(node_memory_MemTotal_bytes-node_memory_MemFree_bytes)/node_memory_MemTotal_bytes
> 0.7
```

瞬时向量与标量进行布尔运算时，PromQL依次比较向量中的所有时间序列样本的值，如果比较结果为true则保留，反之丢弃。

瞬时向量与瞬时向量直接进行布尔运算时，同样遵循默认的匹配模式：依次找到与左边向量元素匹配（标签完全一致）的右边向量元素进行相应的操作，如果没找到匹配元素，则直接丢弃。

目前，Prometheus支持以下布尔运算符如下：

`==` **(相等)** `!=` **(不相等)** `>` **(大于)** `<` **(小于)** `>=` **(大于等于)** `<=` **(小于等于)**

bool修饰符改变布尔运算符的行为

布尔运算符的默认行为是对时序数据进行过滤。而在其它的情况下我们可能需要的是真正的布尔结果。例如，只需要知道当前模块的HTTP请求量是否>=1000，如果大于等于1000则返回1（true）否则返回0（false）。这时可以使用bool修饰符改变布尔运算的默认行为。 例如：

```
prometheus_http_requests_total > bool 1000
```

如果是在两个标量之间使用布尔运算，则必须使用bool修饰符

```
2 == bool 2 # 结果为1
```

集合运算符

通过集合运算，可以在两个瞬时向量与瞬时向量之间进行相应的集合操作。目前，Prometheus支持以下集合运算符：

`and` **(并且)** `or` **(或者)** `unless` **(排除)**

vector1 and vector2 会产生一个由vector1的元素组成的新的向量。该向量包含vector1中完全匹配vector2中的元素组成。

vector1 or vector2 会产生一个新的向量，该向量包含vector1中所有的样本数据，以及vector2中没有与vector1匹配到的样本数据。

vector1 unless vector2 会产生一个新的向量，新向量中的元素由vector1中没有与vector2匹配的元素组成。

操作符优先级

在PromQL操作符中优先级由高到低依次为：

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

匹配模式

向量与向量之间进行运算操作时会基于默认的匹配规则：依次找到与左边向量元素匹配（标签完全一致）的右边向量元素进行运算，如果没找到匹配元素，则直接丢弃。

接下来将介绍在PromQL中有两种典型的匹配模式：**一对一（one-to-one）,多对一（many-to-one）或一对多（one-to-many）。**

一对一匹配:

一对一匹配模式会从操作符两边表达式获取的瞬时向量依次比较并找到唯一匹配(标签完全一致)的样本值。默认情况下，使用表达式：

```
vector1 <operator> vector2
```

在操作符两边表达式标签不一致的情况下，可以使用on(label list)或者ignoring(label list）来修改便签的匹配行为。使用ignoreing可以在匹配时忽略某些便签。而on则用于将匹配行为限定在某些便签之内。

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

多对一和一对多

多对一和一对多两种匹配模式指的是“一”侧的每一个向量元素可以与"多"侧的多个元素匹配的情况。在这种情况下，必须使用group修饰符：group_left或者group_right来确定哪一个向量具有更高的基数（充当“多”的角色）。

```
<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>
```

多对一和一对多两种模式一定是出现在操作符两侧表达式返回的向量标签不一致的情况。因此需要使用ignoring和on修饰符来排除或者限定匹配的标签列表。

PromQL聚合操作

Prometheus还提供了下列内置的聚合操作符，这些操作符作用域瞬时向量。可以将瞬时表达式返回的样本数据进行聚合，形成一个新的时间序列。

`sum` **(求和)** `min` **(最小值)** `max` **(最大值)** `avg` **(平均值)** `stddev` **(标准差)** `stdvar` **(标准方差)** `count` **(计数)** `count_values` **(对value进行计数)** `bottomk` **(后n条时序)** `topk` **(前n条时序)** `quantile` **(分位数)**

使用聚合操作的语法如下：

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

其中只有`count_values`, `quantile`, `topk`, `bottomk`支持参数(parameter)。

without用于从计算结果中移除列举的标签，而保留其它标签。by则正好相反，结果向量中只保留列出的标签，其余标签则移除。通过without和by可以按照样本的问题对数据进行聚合。

例如：

```
sum(prometheus_http_requests_total) without (instance)
```

等价于

```
sum(prometheus_http_requests_total) by
(code,container,endpoint,handler,job,namespace,pod,service)
```

count_values用于时间序列中每一个样本值出现的次数。count_values会为每一个唯一的样本值输出一个时间序列，并且每一个时间序列包含一个额外的标签。

例如：

```
count_values("count", prometheus_http_requests_total)
```

topk和bottomk则用于对样本值进行排序，返回当前样本值前n位，或者后n位的时间序列。

获取HTTP请求数前5位的时序样本数据，可以使用表达式：

```
topk(5, prometheus_http_requests_total)
```

quantile用于计算当前样本数据值的分布情况quantile(φ, express)其中0 ≤ φ ≤ 1。

例如，当φ为0.5时，即表示找到当前样本数据中的中位数：

```
quantile(0.5, prometheus_http_requests_total)
```

PromQL内置函数

计算Counter指标增长率

`increase(v range-vector)`函数是PromQL中提供的众多内置函数之一。其中参数v是一个区间向量，increase函数获取区间向量中的第一个后最后一个样本并返回其增长量。因此，可以通过以下表达式Counter类型指标的增长率：

```
increase(node_cpu_seconds_total[2m]) / 120
```

这里通过node_cpu_seconds_total[2m]获取时间序列最近两分钟的所有样本，increase计算出最近两分钟的增长量，最后除以时间120秒得到node_cpu_seconds_total样本在最近两分钟的平均增长率。并且这个值也近似于主机节点最近两分钟内的平均CPU使用率。

`rate(v range-vector)`函数，rate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。因此，通过以下表达式可以得到与increase函数相同的结果：

```
rate(node_cpu_seconds_total[2m])
```

irate同样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样本数据来计算区间向量的增长速率。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。

```
irate(node_cpu_seconds_total[2m])
```

irate函数相比于rate函数提供了更高的灵敏度，不过当需要分析长期趋势或者在告警规则中，irate的这种灵敏度反而容易造成干扰。因此在长期趋势分析或者告警中更推荐使用rate函数。

预测Gauge指标变化趋势

PromQL中内置的`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。

统计Histogram指标的分位数

Histogram和Summary都可以用于统计和分析数据的分布情况。区别在于Summary是直接在客户端计算了数据分布的分位数情况。而Histogram的分位数计算需要通过`histogram_quantile(φ float, b instant-vector)`函数进行计算。其中φ(0<φ<1)表示需要计算的分位数，如果需要计算中位数φ取值为0.5，以此类推即可。

使用如下表达式：

```
histogram_quantile(0.5, http_request_duration_seconds_bucket)
```

除了这些内置函数以外，PromQL还提供了大量的其它内置函数。这些内置函数包括一些常用的数学计算、日期等等。可以通过阅读Prometheus的官方文档，了解这些函数的使用方式。