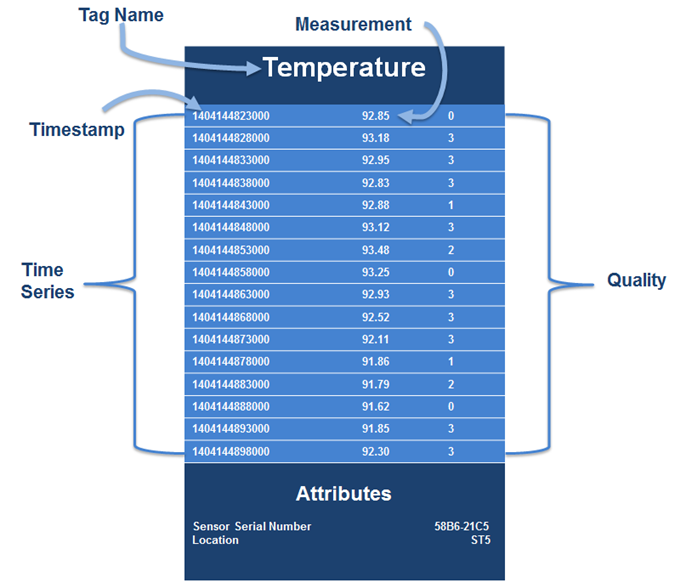
时间序列数据灌注与消费

**数据灌注(Ingestion)**

时间序列使用标签，所述标签通常用于表示传感器（例如，温度传感器）。标签包括一个标签名（传感器）、一个时间戳（时间）和一个测量值（数值）。作为附加选项，时间序列标签也可以包括质量以及一个或多个属性（键/值对）。

**图1时间序列标签格式**

****

| 表1时间序列标签结构 | |
| --- | --- |
| **术语** | **定义** |
| 标签名 | （必填）标签的名称（例如，“温度”）。标签名可以包含字母数字字符、句点（.）、斜杠（/）、破折号（-）、下划线（\_）和空格。  请勿使用标签名存储属性。您可以使用属性选项存储关于数据点的详细信息。 |
| 测量值 | （必填）读数（值）数字（例如：98'C）。 |
| 时间戳 | （必填）UNIX新纪元时间的日期和时间，精度为毫秒。 |
| 质量 | （选填）数据的质量，用数值0、1、2、3表示。关于数据质量数值的说明，请参见下面的*数据质量*。 |
| 属性 | （选填）属性是用于存储标签相关数据的键/值对（例如： "ManufacturerGE":"AircraftID230"）。  对于筛选数据非常有用。  ***注：属性允许您存储关于特定数据点的其他相关详细信息——不得用于存储大量数据。***  ***属性只能包含字母数字字符、句点（.）、斜杠（/）、破折号（-）和下划线（\_）。属性不能包含冒号（:），分号(;)，或等号（=）作为其数值。请勿使用空值。*** |

**数据质量**

您可以通过表示数据质量的值为数据贴标签，这样当您只想检索或处理具有特定质量的数据时，您就可以按照质量进行查询。使用数据质量有助于确保您在查询结果中获得的数据与您的目的最相关。

| 表2数据质量值 | |
| --- | --- |
| **数值** | **说明** |
| 0 | 质量较差。 |
| 1 | 质量不确定。 |
| 2 | 不适用。 |
| 3 | 质量良好。如果您未指定质量，默认值为良好（3）。 |

**数据消费(Consumption)**

时间序列服务提供用于查询与聚合时间序列数据的REST API。通过该API您可以：

* 查询指定标签（传感器等）与时间窗口的时间序列数据。
* 按照属性值筛选。
* 从时间序列数据库中检索所有标签和属性键。
* 在给定时间窗口内添加聚合和内插数据点。
* 查询最新值的时间序列数据库。

***注：当您使用GET最新API时，查询后退15天。***

* 删除标签和所有相关数据点。

关于这些API的更多信息，请参见API文档。

数据灌注的提示

* 请勿将属性用作标签名。
* 在您的应用程序中请勿将时间戳用作数据。
* 将Ingestion请求分散至多个连接，因为向同一连接发送大量负载会增加查询服务中数据可用性的等待时间。

***注：使用多个连接时请注意您的数据计划。***

推送时间序列数据

将命令行界面用作与时间序列服务网关交互的简单方法。时间序列网关使用WebSocket协议进行流式Ingestion。Ingestion端点格式为：wss://ingestion\_url。

使用WebSocket协议而不是HTTP，因为可以获取更大量数据，这样可以提高性能。

您可以在[https://github.com/predixdev/timeseries-bootstrap](https://github.com/predixdev/timeseries-bootstrap" \t "_blank)页面查看用WebSocket实现时间序列Ingestion的示例。

***注：***

***当您将您的应用程序与您的时间序列服务实例绑定时，IngestionURI和<Predix-Zone-Id>包含在您应用程序的环境变量中。如果要查看环境变量，可以在命令行中输入：***

***cf env <application\_name>***

对于所有Ingestion请求，使用您从UAA接收到的令牌在Authorization：中，HTTP标头部分的格式为Bearer <token from trusted issuer>。

***注：在以前的版本中，质量被当做属性处理，但是从当前版本开始，您必须明确为每个数据点提供质量，以及时间戳和测量值。***

**数据灌注请求示例**

下面显示了用于Ingestion请求的JSON有效载荷的一个示例：

URL: wss://ingestion\_urlHeaders:Authorization: Bearer <token from trusted issuer>Predix-Zone-Id: <Predix-Zone-Id>Origin: http://<origin-hostname>/Request Payload:{"messageId": "<MessageID>","body":[{"name":"<TagName>","datapoints":[[<EpochInMs>,<Measure>,<Quality>]],"attributes":{"<AttributeKey>":"<AttributeValue>","<AttributeKey2>":"<AttributeValue2>"}}]}

***注：***

***时间序列服务现在接受压缩的（GZIP）JSON有效载荷。无论Ingestion请求的格式是什么，实际JSON有效载荷的大小限值为512 KB。对于压缩的有效载荷来说，这就意味着解压后的有效载荷不会超过512 KB。***

如果您正在使用Java客户端，下面显示了数据Ingestion请求的一个示例。

IngestionRequestBuilder ingestionBuilder = IngestionRequestBuilder.createIngestionRequest().withMessageId("<MessageID>").addIngestionTag(IngestionTag.Builder.createIngestionTag().withTagName("TagName").addDataPoints(Arrays.asList(new DataPoint(new Date().getTime(), Math.random(), Quality.GOOD),new DataPoint(new Date().getTime(), "Bad Value", Quality.BAD),new DataPoint(new Date().getTime(), null, Quality.UNCERTAIN))).addAttribute("AttributeKey", "AttributeValue").addAttribute("AttributeKey2", "AttributeValue2").build());String json = ingestionBuilder.build().get(0);IngestionResponse response = ClientFactory.ingestionClientForTenant(tenant).ingest(json);String responseStr = response.getMessageId() + response.getStatusCode();

***注：<MessageID>可以是字符串或整型数，而且必须唯一。当使用整型数时，<MessageID>应在0到264（18446744073709551616）之间。***

关于时间序列标签结构的更多信息，请参见时间序列数据灌注与消费。

**确认消息**

对于每个消息包，一条确认消息被发送回客户端。以下示例显示了一条确认消息：

{"messageId": <MessageID>,"statusCode": <AcknowledgementStatusCode>}

确认状态代码包括：

| **代码** | **消息** |
| --- | --- |
| 202 | 成功接受 |
| 400 | 请求无效 |
| 401 | 未授权 |
| 413 | 请求实体过大  ***注：有效载荷不能超过512KB。*** |
| 503 | Ingest数据失败 |

**相关信息**

[时间序列服务查询故障排除](https://www.predix.io/docs/ts_troubleshooting_time_series.html)

**数据类型**

时间序列灌注服务接受混合数据类型。下表显示了混合变量类型（文本、数字和空）的输入和输出样例。

|  |  |  |  |
| --- | --- | --- | --- |
| **IngestionJSON输入** | **IngestionJSON输入类型** | **查询输出** | **数据点类型** |
| 27 | 长整型 | 27 | 数字 |
| “34” | 字符串 | 34 | 数字 |
| 3.345 | 浮点数 | 3.345 | 数字 |
| “1.123” | 字符串 | 1.123 | 数字 |
| “1.0E-2” | 字符串 | 0.01 | 数字 |
| “abc” | 字符串 | “abc” | 文本 |
| “true” | 字符串 | “true” | 文本 |
| true / false | 布尔量 | “true” / “false” | 文本 |
| “null” | 字符串 | “null” | 文本 |
| 空 | 空 | 空 | 空 |
| "" | 字符串 | "" | 空 |
| " " | 字符串 | " " | 空 |

查询时间序列数据

执行*时间序列服务入门*中的所有任务。

根据这些步骤创建您的时间序列查询。

1. 通过在命令行中输入以下内容查看您应用程序的环境变量：

cf env <application\_name>

您应用程序的环境变量中包括查询URI、可信发布者令牌和<Predix-Zone-Id>。您将在您的查询HTTP标头中使用到此信息。

1. 创建您的查询。

您的查询的标头必须是以下格式：

URL: https://query\_url/v1/datapoints Method: POST Headers:Authorization: Bearer <token from trusted issuer>Predix-Zone-Id: <Predix-Zone-Id>

***注：***

***您可以使用GET和POST两种方法查询时间序列数据。关于如何构建查询请求的更多信息，请参见***[***查询属性与示例***](https://www.predix.io/docs/?r=101547#JV5kqUXT)***。***

时间序列数据查询的提示

**属性**

* 属性的属性键或值均不得包含高基数值。
* 请勿将计数器存储为属性，因为这会降低查询性能。

**GET与POST方法**

* 对于不太复杂的查询使用GET方法。响应可以缓存在代理服务器和浏览器中。
* 当查询JSON很长，而且您不需要缓存请求时，使用POST方法更方便。

使用POST API时，以下规则适用：

* 仅支持筛选器运算，而且是可选的。
* 指定开始时间是可选项，但是，如果您指定了一个开始时间，那么也必须指定一个结束时间。
* 如果您未定义时间窗口，那么查询检索从当前时间（现在）开始的最新数据点。

**性能**

* 每个查询中含有少量标签的多个查询，其性能优于包含多个标签的少量查询。

**数据点限值**

查询结果中的数据点不得超过最大限值200,000。

查询属性与示例

**查询属性**

您的查询必须指定一个开始字段，可以是绝对的也可以是相对的。结束字段是可选项，可以是绝对的也可以是相对的。如果您未指定 end 时间，查询使用当前日期和时间。

* 绝对开始时间 – 表示为一个整型时间戳值，单位毫秒。
* 相对开始时间 – 相对于当前时间计算得出。例如，如果开始时间是15分钟，查询返回最近15分钟的所有匹配数据点。相对开始时间表示为 <value><time-unit>-ago格式的字符串，其中：
  + <value> 必须是一个大于零（0）的整型数。
  + <time-unit> – 必须表示为以下形式：

| **时间单位** | **代表** |
| --- | --- |
| ms | 毫秒 |
| s | 秒 |
| mi | 分钟 |
| h | 小时 |
| d | 天 |
| w | 周 |
| mm | 月 |
| y | 年 |

**一般查询示例**

您可以使用时间序列API列出标签和属性，也可以查询数据点。您可以使用开始时间、结束时间、标签名、时间范围、测量值和属性查询数据点。以下示例显示了使用POST方法的JSON数据查询，查询结果中返回数据点的数量限值为1000：

URL: https://*query\_url*/v1/datapoints Method: POST Headers:Authorization: Bearer <token from trusted issuer>Predix-Zone-Id: <Predix-Zone-Id>Payload:{    "start": 1427463525000,    "end": 1427483525000,    "tags": [       {            "name": [                "ALT\_SENSOR",                "TEMP\_SENSOR"            ],            "limit": 1000,            "aggregations": [{                "type": "avg",                "interval": "1h"              }],            "filters": {                "attributes": {                    "host": "<host>",                    "type": "<type>"                },                "measurements": {                    "condition": "ge",                    "values": "23.1"                },                "qualities": {                    "values": [                        "0",                        "3"                    ]                }            },            "groups": [                {                    "name": "attribute",                    "attributes": [                        "attributename1",                        "attributename2"                    ]                }            ]        }    ]}

如果您正在使用Java时间序列客户端库，下面显示了查询请求的一个示例：

QueryBuilder builder = QueryBuilder.createQuery().withStartAbs(1427463525000L).withEndAbs(1427483525000L).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("ALT\_SENSOR", "TEMP\_SENSOR")).withLimit(1000).addAggregation(Aggregation.Builder.averageWithInterval(1, TimeUnit.HOURS)).addFilters(FilterBuilder.getInstance().addAttributeFilter("host", Arrays.asList("<host>")).build()).addFilters(FilterBuilder.getInstance().addAttributeFilter("type", Arrays.asList("<type>")).build()).addFilters(FilterBuilder.getInstance().addMeasurementFilter(FilterBuilder.Condition.GREATER\_THAN\_OR\_EQUALS, Arrays.asList("23.1")).build()).addFilters(FilterBuilder.getInstance().withQualitiesFilter(Arrays.asList(Quality.BAD, Quality.GOOD)).build()).build());QueryResponse response = ClientFactory.queryClientForTenant(tenant).queryAll(builder.build());

将您从可信发布者处接收的令牌以'Bearer <token from trusted issuer>'的格式应用到所有查询请求的HTTP标头‘Authorization’中。

***注：***

***您可以使用GET和POST两种方法查询时间序列数据。使用GET HTTP方法调用查询API时，通过将查询JSON作为一个URL查询参数进行传递。。GET方法允许将请求和响应缓存在代理服务器和浏览器中。***

***当查询JSON很长且超过查询参数长度时（对于某些浏览器来说），使用方便的POST方法调用时间序列API。***

***POST请求对数据长度没有限制。然而，请求不会被缓存。时间序列服务API设计支持对多个标签及其属性进行复杂的嵌套时间序列查询。例如，更复杂的查询可能包括聚合与筛选。***

关于该API的更多信息，请参见API文档。

***注：标签名、测量值和属性区分大小写，只能包含字母数字字符、句点（.）、斜杠（/）、破折号（-）和下划线（\_）。***

**在一个查询中使用相对开始和结束时间**

以下示例显示了一个JSON查询，使用数值为“15”单位为“分钟”的相对开始时间，和单位为毫秒的绝对结束时间。

{    "start": "15mi-ago","end": 1427483525000,    "tags": [        {            "name": "ALT\_SENSOR",            "limit": 1000       }   ]}

以下示例显示了一个使用相对开始时间的查询，使用了时间序列客户端库：

QueryBuilder builder = QueryBuilder.createQuery().withStartRelative(15, TimeUnit.MINUTES, TimeRelation.AGO).withEndAbs(1427483525000L).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("ALT\_SENSOR")).withLimit(1000).build());QueryResponse queryResponse = ClientFactory.queryClientForTenant(tenant).queryAll(builder.build());

以下示例显示了一个JSON查询，使用单位为毫秒的绝对开始时间，和数值为“5”单位为“分钟”的相对结束时间。

{    "start": 1427463525000,    "end": "5mi-ago",    "tags": [        {            "name": "ALT\_SENSOR",            "limit": 1000       }   ]}

以下示例显示了一个使用绝对开始时间和相对结束时间的查询，使用了时间序列客户端库：

QueryBuilder builder = QueryBuilder.createQuery().withStartAbs(1427463525000L).withEndRelative(5, TimeUnit.MINUTES, TimeRelation.AGO).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("ALT\_SENSOR")).withLimit(1000).build());QueryResponse queryResponse = ClientFactory.queryClientForTenant(tenant).queryAll(builder.build());

**数据点限值**

以下限值适用于数据查询：

* 查询结果中的数据点个数不得超过最大限值200,000。

缩小您的查询条件（例如，时间窗口），以返回更少数量的数据点。

**限制一个查询返回的数据点**

当您创建一个查询时，您可以为查询结果返回的数据点数量指定一个限值。本示例显示了一个JSON查询，其"limit"属性设置为1000。

{    "start": 1427463525000,    "end": 1427483525000,    "tags": [        {            "name": "ALT\_SENSOR",            "limit": 1000       }   ]}

以下示例显示了使用时间序列客户端库的"limit"查询。

QueryBuilder builder = QueryBuilder.createQuery().withStartAbs(1427463525000L).withEndAbs(1427483525000L).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("ALT\_SENSOR")).withLimit(1000).build());QueryResponse queryResponse = ClientFactory.queryClientForTenant(tenant).queryAll(builder.build());

**指定一个返回查询数据点的顺序**

查询结果的默认顺序是升序，按照时间戳排序。您可以在您的查询请求中使用order属性指定您希望数据点返回的顺序。以下示例以降序返回数据点：

{    "start": 1427463525000,    "end": 1427483525000,    "tags": [        {            "name": "ALT\_SENSOR",            "order": "desc"       }   ]}

以下示例显示了使用时间序列客户端库的"order"查询：

QueryBuilder builder = QueryBuilder.createQuery().withStartAbs(1427463525000L).withEndAbs(1427483525000L).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("ALT\_SENSOR")).withOrder(QueryTag.Order.DESCENDING).build());QueryResponse queryResponse = ClientFactory.queryClientForTenant(tenant).queryAll(builder.build());

**相关概念**时间序列数据的灌注与消费

数据聚合器

聚合函数允许您对一系列数据点执行数学运算，以创建数量更少的数据点。聚合函数每次执行不同的数学运算，而且对取样期间包括的所有数据点执行运算。

***注：时间范围内属于字符串或空类型的数据点在数据聚合过程中会被忽略，在响应中不返回。***

您可以使用<query url>/v1/aggregations端点获取可用聚合的列表。

关于这些API的更多信息，请参见API文档。

聚合以查询中指定的顺序被处理。一个聚合的输出被传递到下一个作为输入，直到它们均被处理。

**聚合器参数**

unit与sampling聚合器是多个聚合器共有的。

单位是一个字符串，必须为以下其中一种：

* 毫秒
* 秒
* 分钟
* 小时
* 天
* 周

sampling聚合器是一个JSON对象，包含一个value (integer)和一个unit。

**范围聚合器**

您可以为任意范围聚合器设置以下参数：

* sampling– 包含一个单位和数值的JSON对象。
* align\_start\_time（布尔量） – 如果您想要使每个范围内聚合数据点的时间落在范围的开始，而不是范围内第一个数据点的数值处，将其设置为真。
* align\_sampling（布尔量） – 如果您想要使聚合范围与取样大小对齐，而且您想要在刷新数据做图时数据具有相同的形状，将其设置为真。
* start\_time（长整型） – 开始计算范围的开始时间（查询开始）。
* time\_zone（长时区格式） – 基于时间计算使用的时区。

**聚合器**

| **名称** | **类型** | **参数** | **说明** |
| --- | --- | --- | --- |
| avg | 平均值 | 扩展range聚合器。 | 返回数值的平均值。 |
| count | 计数 | 扩展range聚合器。 | 计算数据点的数量。 |
| dev | 标准差 | 扩展range聚合器。 | 返回时间序列的标准差。 |
| diff | 差值 | 扩展range聚合器。 | 计算连续数据点之间的差值。 |
| div | 除法 | divisor（双精度）参数为必填。这是对数据点做除法的除数。 | 返回可以被divisor属性的特定值除尽的每个数据点。 |
| gaps | 间隔 | 扩展range 聚合器。 | 根据取样率，通过空数据点在数据中标记间隔。 |
| interpolate | 内插 | 扩展range聚合器。 | 对选择的窗口执行线性内插。 |
| least\_squares | 最小二乘 | 扩展range聚合器。 | 返回代表范围内数据点集最佳拟合线的两个点。 |
| max | 最大值 | 扩展range聚合器。 | 继承自range聚合器。返回最近的最大值。 |
| min | 最小值 | 扩展range聚合器。 | 返回最近的最小值。 |
| percentile | 百分位数 | 扩展range聚合器。  percentile– 计数的百分位数。 | 计算概率分布，返回分布的特定百分位数。“百分位数”值定义为0 < 百分位数 <= 1，其中.5代表50%，1代表100%。 |
| rate | 比率 | * sampling– 设置取样，用于计算比率。 * unit – 必填。设置取样持续时间，单位为毫秒、秒、分钟、小时、天或周。例如，如果您将单位设置为小时，取样率为一小时。 * time\_zone – 必须使用长整型格式，例如： | 返回一对数据点之间的变化率。 |
| sampler | 取样器 | 可选参数包括：   * unit – 设置取样持续时间，单位为毫秒、秒、分钟、小时、天或周。例如，如果您将单位设置为小时，取样率为一小时。 * time\_zone– 必须使用长整型格式，例如： | 计算数据点的取样变化率。 |
| scale | 换算 | factor（双精度） – 换算值。 | 通过系数换算每个数据点。 |
| sum | 求和 | 扩展range聚合器。 | 返回所有值的和。 |
| trendmode | 趋势模式 | * sampling– 设置在每个间隔内计算最小值和最大值的取样。 | 返回每个间隔内的min与max值。  ***注：趋势模式将所有长整型转换为双精度型。对于非常大的长整型值这样会损失精度。*** |

聚合示例

**数据插值**

当数据丢失时您可以在原始数据样例之间内插数据，并根据一个时间间隔（例如，每一小时）查询结果。下面显示了为给定的时间窗口每小时内插数据的请求样例：

{"start":1432667547000,"end":1432667552000,"tags":[{"name":"phosphate.level","aggregations":[{"type":"interpolate","interval": "1h"}]}]}

**查询返回特定数量的数据点**

以下显示了一个使用插值方法查询数据并指定返回的数据点时间窗口和数量的示例：

{"start":1432667547000,"end":1432667552000,"tags":[{"name":"phosphate.level","aggregations":[{"type":"interpolate","sampling":{"datapoints":500}}]}]}

**查询平均值**

此处显示了一个以30秒为间隔查询平均值以便聚合数据的示例：

{"tags": [{"name": "Compressor-2015","aggregations": [{"type": "avg","sampling": {"unit": "seconds","value": "30"}}]}],"start": 1452112200000,"end": 1453458896222}

通过混合数据类型取平均

平均聚合器返回数值的平均值。

**Ingestion JSON示例**

[{"name": "test.average.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.average.mixedType","aggregations": [{"type": "avg","sampling": {"datapoints": 1}}]}]}

下面显示了使用时间序列客户端库的平均聚合器的一个示例：

QueryBuilder builder = QueryBuilder.createQuery().withStartAbs(1432671129000L).withEndAbs(1432672230500L).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("ALT\_SENSOR")).addAggregation(Aggregation.Builder.averageWithInterval(3600, TimeUnit.SECONDS)).build());QueryResponse queryResponse = ClientFactory.queryClientForTenant(tenant).queryForLatest(builder.build());

**结果示例**

{"tags": [{"name": "test.average.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776300000, 28.875, 3]]}],"stats": {"rawCount": 4}}]}

通过双精度数据类型取平均

平均聚合器返回数值的平均值。

**Ingestion JSON示例**

[{"name": "test.average.doubleType","datapoints": [[1435766300000, 2.0],[1435766400000, 2.5],[1435766500000, 3.0],[1435766550000, 3.5],[1435766600000, 4.0],[1435766700000, 50.0],[1435766800000, 100.0],[1435766900000, 150.5],[1435777000000, 175.0],[1435777100000, 175.5],[1435777200000, 200.0],[1435777300000, 225.0],[1435777400000, 225.5],[1435777500000, 250],[1435777600000, 300.0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435787600000,"tags": [{"name": "test.average.doubleType","aggregations": [{"type": "avg","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.average.doubleType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435766300000, 124.43333333333334, 3]]}],"stats": {"rawCount": 15}}]}

通过混合数据类型计数

计数聚合器返回最近的最大值。

**示例1**

**Ingestion JSON示例**

[{"name": "test.count.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.count.mixedType","filters": {"qualities": {"values": ["3"]}},"aggregations": [{"type": "trendmode","sampling": {"datapoints": 1}}, {"type": "count","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.count.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"filters": {"qualities": {"values": ["3"]}},"attributes": {},"values": [[1435776500000, 1, 3]]}],"stats": {"rawCount": 1}}]}

**示例2**

**Ingestion JSON示例**

[{"name": "test.count.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.count.mixedType","aggregations": [{"type": "trendmode","sampling": {"datapoints": 1}}, {"type": "count","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.count.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776300000, 2, 3]]}],"stats": {"rawCount": 4}}]}

**示例3**

**Ingestion JSON示例**

[{"name": "test.count.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.count.mixedType","aggregations": [{"type": "count","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.count.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776300000, 4, 3]]}],"stats": {"rawCount": 4}}]}

通过双精度数据类型计数

计数聚合器返回最近的最大值。

**Ingestion JSON示例**

[{"name": "test.count.mode.doubleType","datapoints": [[1435766300000, 2.0],[1435766400000, 2.5],[1435766500000, 3.0],[1435766550000, 3.5, 2],[1435766600000, 4.0],[1435766700000, 50.0],[1435766800000, 100.0],[1435766900000, 150.5],[1435777000000, 175.0],[1435777100000, 175.5],[1435777200000, 200.0],[1435777300000, 225.0],[1435777400000, 225.5],[1435777500000, 250],[1435777600000, 300.0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435787600000,"tags": [{"name": "test.count.mode.doubleType","aggregations": [{"type": "count","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.count.mode.doubleType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435766300000, 15, 3]]}],"stats": {"rawCount": 15}}]}

通过混合数据类型求差值

差值聚合器计算连续数据点之间的差值。

**Ingestion JSON示例**

[{"name": "test.diff.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.diff.mixedType","aggregations": [{"type": "diff"}]}]}

**结果示例**

{"tags": [{"name": "test.diff.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776500000, 8.5, 3],[1435776550000, 89.5, 3],[1435776900000, -97, 3]]}],"stats": {"rawCount": 4}}]}

通过混合数据类型做除法

除法聚合器返回可以被divisor属性的特定值除尽的每个数据点。

**Ingestion JSON示例**

[{"name": "test.div.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.div.mixedType","aggregations": [{"type": "div","divisor": 3}]}]}

**结果示例**

{"tags": [{"name": "test.div.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776300000, 0.6666666666666666, 3],[1435776500000, 3.5, 3],[1435776550000, 33.333333333333336, 3],[1435776900000, 1, 3]]}],"stats": {"rawCount": 4}}]}

通过双精度数据类型做除法

除法聚合器返回可以被divisor属性的特定值除尽的每个数据点。

**Ingestion JSON示例**

[{"name": "test.div.doubleType","datapoints": [[1435766300000, 2.0],[1435766400000, 2.5],[1435766500000, 3.0],[1435766550000, 3.5],[1435766600000, 4.0],[1435766700000, 50.0],[1435766800000, 100.0],[1435766900000, 150.5],[1435777000000, 175.0],[1435777100000, 175.5],[1435777200000, 200.0],[1435777300000, 225.0],[1435777400000, 225.5],[1435777500000, 250],[1435777600000, 300.0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435787600000,"tags": [{"name": "test.div.doubleType","aggregations": [{"type": "div","divisor": 3}]}]}

**结果示例**

{"tags": [{"name": "test.div.doubleType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435766300000, 0.6666666666666666, 3],[1435766400000, 0.8333333333333334, 3],[1435766500000, 1, 3],[1435766550000, 1.1666666666666667, 3],[1435766600000, 1.3333333333333333, 3],[1435766700000, 16.666666666666668, 3],[1435766800000, 33.333333333333336, 3],[1435766900000, 50.166666666666664, 3],[1435777000000, 58.333333333333336, 3],[1435777100000, 58.5, 3],[1435777200000, 66.66666666666667, 3],[1435777300000, 75, 3],[1435777400000, 75.16666666666667, 3],[1435777500000, 83.33333333333333, 3],[1435777600000, 100, 3]]}],"stats": {"rawCount": 15}}]}

通过混合数据类型求间隔

间隔聚合器根据取样率，通过空数据点在数据中标记间隔。

**Ingestion JSON示例**

[{"name": "test.gaps.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.gaps.mixedType","aggregations": [{"type": "gaps","interval": "100000ms"}]}]}

**结果示例**

{"tags": [{"name": "test.gaps.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776300000, 2, 1],[1435776400000, null, 3],[1435776500000, 10.5, 3],[1435776550000, 100, 2],[1435776600000, null, 3],[1435776700000, null, 3],[1435776800000, null, 3],[1435776900000, 3, 0]]}],"stats": {"rawCount": 4}}]}

通过双精度数据类型求间隔

间隔聚合器根据取样率，通过空数据点在数据中标记间隔。

**Ingestion JSON示例**

[{"name": "test.gaps.doubleType","datapoints": [[1435766300000, 2.0],[1435766400000, 2.5],[1435766500000, 3.0],[1435766550000, 3.5],[1435766600000, 4.0],[1435766700000, 50.0],[1435766800000, 100.0],[1435766900000, 150.5],[1435777000000, 175.0],[1435777100000, 175.5],[1435777200000, 200.0],[1435777300000, 225.0],[1435777400000, 225.5],[1435777500000, 250],[1435777600000, 300.0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435787600000,"tags": [{"name": "test.gaps.doubleType","aggregations": [{"type": "gaps","interval": "500000ms"}]}]}

**结果示例**

{"tags": [{"name": "test.gaps.doubleType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435766300000, 2, 3],[1435766400000, 2.5, 3],[1435766500000, 3, 3],[1435766550000, 3.5, 3],[1435766600000, 4, 3],[1435766700000, 50, 3],[1435766800000, 100, 3],[1435766900000, 150.5, 3],[1435767300000, null, 3],[1435767800000, null, 3],[1435768300000, null, 3],[1435768800000, null, 3],[1435769300000, null, 3],[1435769800000, null, 3],[1435770300000, null, 3],[1435770800000, null, 3],[1435771300000, null, 3],[1435771800000, null, 3],[1435772300000, null, 3],[1435772800000, null, 3],[1435773300000, null, 3],[1435773800000, null, 3],[1435774300000, null, 3],[1435774800000, null, 3],[1435775300000, null, 3],[1435775800000, null, 3],[1435776300000, null, 3],[1435777000000, 175, 3],[1435777100000, 175.5, 3],[1435777200000, 200, 3],[1435777300000, 225, 3],[1435777400000, 225.5, 3],[1435777500000, 250, 3],[1435777600000, 300, 3]]}],"stats": {"rawCount": 15}}]}

通过双精度数据类型求内插值

**Ingestion JSON示例**

[{"name": "test.interpolation.doubleType","datapoints": [[1435766300000, 2.0],[1435766400000, 2.5],[1435766500000, 3.0],[1435766550000, 3.5],[1435766600000, 4.0],[1435766700000, 50.0],[1435766800000, 100.0],[1435766900000, 150.5],[1435777000000, 175.0],[1435777100000, 175.5],[1435777200000, 200.0],[1435777300000, 225.0],[1435777400000, 225.5],[1435777500000, 250],[1435777600000, 300.0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435767300000,"tags": [{"name": "test.interpolation.doubleType","aggregations": [{"type": "interpolate","interval": "10000ms"}]}]}

**结果示例**

{"tags": [{"name": "test.interpolation.doubleType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435766310000, 2.05, 3],[1435766320000, 2.1, 3],[1435766330000, 2.15, 3],[1435766340000, 2.2, 3],[1435766350000, 2.25, 3],[1435766360000, 2.3, 3],[1435766370000, 2.35, 3],[1435766380000, 2.4, 3],[1435766390000, 2.45, 3],[1435766400000, 2.5, 3],[1435766410000, 2.55, 3],[1435766420000, 2.6, 3],[1435766430000, 2.65, 3],[1435766440000, 2.7, 3],[1435766450000, 2.75, 3],[1435766460000, 2.8, 3],[1435766470000, 2.85, 3],[1435766480000, 2.9, 3],[1435766490000, 2.95, 3],[1435766500000, 3, 3],[1435766510000, 3.1, 3],[1435766520000, 3.2, 3],[1435766530000, 3.3, 3],[1435766540000, 3.4, 3],[1435766550000, 3.5, 3],[1435766560000, 3.6, 3],[1435766570000, 3.7, 3],[1435766580000, 3.8, 3],[1435766590000, 3.9, 3],[1435766600000, 4, 3],[1435766610000, 8.600000000000001, 3],[1435766620000, 13.200000000000001, 3],[1435766630000, 17.8, 3],[1435766640000, 22.400000000000002, 3],[1435766650000, 27, 3],[1435766660000, 31.6, 3],[1435766670000, 36.2, 3],[1435766680000, 40.800000000000004, 3],[1435766690000, 45.4, 3],[1435766700000, 50, 3],[1435766710000, 55, 3],[1435766720000, 60, 3],[1435766730000, 65, 3],[1435766740000, 70, 3],[1435766750000, 75, 3],[1435766760000, 80, 3],[1435766770000, 85, 3],[1435766780000, 90, 3],[1435766790000, 95, 3],[1435766800000, 100, 3],[1435766810000, 105.05, 3],[1435766820000, 110.1, 3],[1435766830000, 115.15, 3],[1435766840000, 120.2, 3],[1435766850000, 125.25, 3],[1435766860000, 130.3, 3],[1435766870000, 135.35, 3],[1435766880000, 140.4, 3],[1435766890000, 145.45, 3],[1435766900000, 150.5, 3],[1435766910000, 150.5, 3],[1435766920000, 150.5, 3],[1435766930000, 150.5, 3],[1435766940000, 150.5, 3],[1435766950000, 150.5, 3],[1435766960000, 150.5, 3],[1435766970000, 150.5, 3],[1435766980000, 150.5, 3],[1435766990000, 150.5, 3],[1435767000000, 150.5, 3],[1435767010000, 150.5, 3],[1435767020000, 150.5, 3],[1435767030000, 150.5, 3],[1435767040000, 150.5, 3],[1435767050000, 150.5, 3],[1435767060000, 150.5, 3],[1435767070000, 150.5, 3],[1435767080000, 150.5, 3],[1435767090000, 150.5, 3],[1435767100000, 150.5, 3],[1435767110000, 150.5, 3],[1435767120000, 150.5, 3],[1435767130000, 150.5, 3],[1435767140000, 150.5, 3],[1435767150000, 150.5, 3],[1435767160000, 150.5, 3],[1435767170000, 150.5, 3],[1435767180000, 150.5, 3],[1435767190000, 150.5, 3],[1435767200000, 150.5, 3],[1435767210000, 150.5, 3],[1435767220000, 150.5, 3],[1435767230000, 150.5, 3],[1435767240000, 150.5, 3],[1435767250000, 150.5, 3],[1435767260000, 150.5, 3],[1435767270000, 150.5, 3],[1435767280000, 150.5, 3],[1435767290000, 150.5, 3],[1435767300000, 150.5, 3]]}],"stats": {"rawCount": 8}}]}

通过混合数据类型求内插值

**Ingestion JSON示例**

[{"name": "test.iterpolation.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.iterpolation.mixedType","aggregations": [{"type": "interpolate","interval": "100000ms"}]}]}

下面显示了使用时间序列客户端库通过间隔使用插值聚合器进行查询的一个示例：

QueryBuilder builder = QueryBuilder.createQuery().withStartAbs(1432671129000L).withEndAbs(1432672230500L).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("ALT\_SENSOR")).addAggregation(Aggregation.Builder.interpolateWithInterval(3600, TimeUnit.SECONDS)).build());QueryResponse queryResponse = ClientFactory.queryClientForTenant(tenant).queryForLatest(builder.build());

下面显示了使用时间序列客户端库通过数据点计数使用插值聚合器进行查询的一个示例：

QueryBuilder builder = QueryBuilder.createQuery().withStartAbs(1432671129000L).withEndAbs(1432672230500L).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("ALT\_SENSOR")).addAggregation(Aggregation.Builder.interpolateForDatapointCount(100)).build());QueryResponse queryResponse = ClientFactory.queryClientForTenant(tenant).queryForLatest(builder.build());

**结果示例**

{"tags": [{"name": "test.iterpolation.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435766400000, 0, 0],[1435766500000, 0, 0],[1435766600000, 0, 0],[1435766700000, 0, 0],[1435766800000, 0, 0],[1435766900000, 0, 0],[1435767000000, 0, 0],[1435767100000, 0, 0],[1435767200000, 0, 0],[1435767300000, 0, 0],[1435767400000, 0, 0],[1435767500000, 0, 0],[1435767600000, 0, 0],[1435767700000, 0, 0],[1435767800000, 0, 0],[1435767900000, 0, 0],[1435768000000, 0, 0],[1435768100000, 0, 0],[1435768200000, 0, 0],[1435768300000, 0, 0],[1435768400000, 0, 0],[1435768500000, 0, 0],[1435768600000, 0, 0],[1435768700000, 0, 0],[1435768800000, 0, 0],[1435768900000, 0, 0],[1435769000000, 0, 0],[1435769100000, 0, 0],[1435769200000, 0, 0],[1435769300000, 0, 0],[1435769400000, 0, 0],[1435769500000, 0, 0],[1435769600000, 0, 0],[1435769700000, 0, 0],[1435769800000, 0, 0],[1435769900000, 0, 0],[1435770000000, 0, 0],[1435770100000, 0, 0],[1435770200000, 0, 0],[1435770300000, 0, 0],[1435770400000, 0, 0],[1435770500000, 0, 0],[1435770600000, 0, 0],[1435770700000, 0, 0],[1435770800000, 0, 0],[1435770900000, 0, 0],[1435771000000, 0, 0],[1435771100000, 0, 0],[1435771200000, 0, 0],[1435771300000, 0, 0],[1435771400000, 0, 0],[1435771500000, 0, 0],[1435771600000, 0, 0],[1435771700000, 0, 0],[1435771800000, 0, 0],[1435771900000, 0, 0],[1435772000000, 0, 0],[1435772100000, 0, 0],[1435772200000, 0, 0],[1435772300000, 0, 0],[1435772400000, 0, 0],[1435772500000, 0, 0],[1435772600000, 0, 0],[1435772700000, 0, 0],[1435772800000, 0, 0],[1435772900000, 0, 0],[1435773000000, 0, 0],[1435773100000, 0, 0],[1435773200000, 0, 0],[1435773300000, 0, 0],[1435773400000, 0, 0],[1435773500000, 0, 0],[1435773600000, 0, 0],[1435773700000, 0, 0],[1435773800000, 0, 0],[1435773900000, 0, 0],[1435774000000, 0, 0],[1435774100000, 0, 0],[1435774200000, 0, 0],[1435774300000, 0, 0],[1435774400000, 0, 0],[1435774500000, 0, 0],[1435774600000, 0, 0],[1435774700000, 0, 0],[1435774800000, 0, 0],[1435774900000, 0, 0],[1435775000000, 0, 0],[1435775100000, 0, 0],[1435775200000, 0, 0],[1435775300000, 0, 0],[1435775400000, 0, 0],[1435775500000, 0, 0],[1435775600000, 0, 0],[1435775700000, 0, 0],[1435775800000, 0, 0],[1435775900000, 0, 0],[1435776000000, 0, 0],[1435776100000, 0, 0],[1435776200000, 0, 0],[1435776300000, 2, 1],[1435776400000, 2, 1],[1435776500000, 10, 3],[1435776600000, 100, 2],[1435776700000, 100, 2],[1435776800000, 100, 2],[1435776900000, 0, 0],[1435777000000, 0, 0]]}],"stats": {"rawCount": 4}}]}

通过混合数据类型取最小值

最小值聚合器返回最近的最小值。

**Ingestion JSON示例**

[{"name": "test.min.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.min.mixedType","aggregations": [{"type": "min","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.min.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776300000, 2, 1]]}],"stats": {"rawCount": 4}}]}

通过双精度数据类型取最小值

最小值聚合器返回最近的最小值。

**Ingestion JSON示例**

[{"name": "test.min.doubleType","datapoints": [[1435766300000, 2.0],[1435766400000, 2.5],[1435766500000, 3.0],[1435766550000, 3.5],[1435766600000, 4.0],[1435766700000, 50.0],[1435766800000, 100.0],[1435766900000, 150.5],[1435777000000, 175.0],[1435777100000, 175.5],[1435777200000, 200.0],[1435777300000, 225.0],[1435777400000, 225.5],[1435777500000, 250],[1435777600000, 300.0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435787600000,"tags": [{"name": "test.min.doubleType","aggregations": [{"type": "min","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.min.doubleType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435766300000, 2, 3]]}],"stats": {"rawCount": 15}}]}

通过混合数据类型取最大值

最大值聚合器返回最近的最大值。

**Ingestion JSON示例**

[{"name": "test.max.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435776300000,"end": 1435779600000,"tags": [{"name": "test.max.mixedType","aggregations": [{"type": "max","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.max.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776550000, 100, 2]]}],"stats": {"rawCount": 4}}]}

通过双精度类型取最大值

最大值聚合器返回最近的最大值。

**Ingestion JSON示例**

[{"name": "test.max.mode.doubleType","datapoints": [[1435766300000, 2.0],[1435766400000, 2.5],[1435766500000, 3.0],[1435766550000, 3.5, 2],[1435766600000, 4.0],[1435766700000, 50.0],[1435766800000, 100.0],[1435766900000, 150.5],[1435777000000, 175.0],[1435777100000, 175.5],[1435777200000, 200.0],[1435777300000, 225.0],[1435777400000, 225.5],[1435777500000, 250],[1435777600000, 300.0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777800000,"tags": [{"name": "test.max.mode.doubleType","aggregations": [{"type": "max","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.max.mode.doubleType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435777600000, 300, 3]]}],"stats": {"rawCount": 15}}]}

通过混合数据类型换算

换算聚合器通过系数换算每个数据点。

**Ingestion JSON示例**

[{"name": "test.scale.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.scale.mixedType","aggregations": [{ "type": "scale","factor": 3}]}]}

**结果示例**

{"tags": [{"name": "test.scale.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776300000, 6, 1],[1435776500000, 31.5, 3],[1435776550000, 300, 2],[1435776900000, 9, 0]]}],"stats": {"rawCount": 4}}]}

通过双精度数据类型换算

换算聚合器通过系数换算每个数据点。

**Ingestion JSON示例**

[{"name": "test.scale.doubleType","datapoints": [[1435766300000, 2.0],[1435766400000, 2.5],[1435766500000, 3.0],[1435766550000, 3.5, 2],[1435766600000, 4.0],[1435766700000, 50.0],[1435766800000, 100.0],[1435766900000, 150.5],[1435777000000, 175.0],[1435777100000, 175.5],[1435777200000, 200.0],[1435777300000, 225.0],[1435777400000, 225.5],[1435777500000, 250],[1435777600000, 300.0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435787600000,"tags": [{"name": "test.scale.doubleType","aggregations": [{"type": "scale","factor": 3}]}]}

下面显示了使用时间序列客户端库通过系数使用换算聚合器进行查询的一个示例：

QueryBuilder builder = QueryBuilder.createQuery().withStartAbs(1432671129000L).withEndAbs(1432672230500L).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("ALT\_SENSOR")).addAggregation(Aggregation.Builder.scaleWithFactor((double) 10)).build());QueryResponse queryResponse = ClientFactory.queryClientForTenant(tenant).queryForLatest(builder.build());

**结果示例**

{"tags": [{"name": "test.scale.doubleType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435766300000, 6, 3],[1435766400000, 7.5, 3],[1435766500000, 9, 3],[1435766550000, 10.5, 2],[1435766600000, 12, 3],[1435766700000, 150, 3],[1435766800000, 300, 3],[1435766900000, 451.5, 3],[1435777000000, 525, 3],[1435777100000, 526.5, 3],[1435777200000, 600, 3],[1435777300000, 675, 3],[1435777400000, 676.5, 3],[1435777500000, 750, 3],[1435777600000, 900, 3]]}],"stats": {"rawCount": 15}}]}

通过混合数据类型求和

求和聚合器返回所有值的和。

**Ingestion JSON示例**

[{"name": "test.sum.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777000000,"tags": [{"name": "test.sum.mixedType","aggregations": [{"type": "sum","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.sum.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776300000, 115.5, 3]]}],"stats": {"rawCount": 4}}]}

通过双精度数据类型求趋势模式

趋势模式聚合器返回特定时间范围内最小和最大值数据点。使用sampling参数指定范围。

**Ingestion JSON示例**

[{"name": "test.trend.mode.doubleType","datapoints": [[1435766300000, 2.0],[1435766400000, 2.5],[1435766500000, 3.0],[1435766550000, 3.5, 2],[1435766600000, 4.0],[1435766700000, 50.0],[1435766800000, 100.0],[1435766900000, 150.5],[1435777000000, 175.0],[1435777100000, 175.5],[1435777200000, 200.0],[1435777300000, 225.0],[1435777400000, 225.5],[1435777500000, 250],[1435777600000, 300.0]]}]

**查询示例**

{"start": 1435766300000,"end": 1435777800000,"tags": [{"name": "test.trend.mode.doubleType","aggregations": [{"type": "trendmode","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.trend.mode.doubleType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435766300000, 2, 3],[1435777600000, 300, 3]]}],"stats": {"rawCount": 15}}]}

通过混合数据类型求趋势模式

趋势模式聚合器返回特定时间范围内最小和最大值数据点。使用 sampling 参数指定范围。

**Ingestion JSON示例**

[{"name": "test.trend.mode.mixedType","datapoints": [[1435776300000, 2, 1],[1435776400000, null],[1435776500000, 10.5, 3],[1435776550000, "100", 2],[1435776600000, "string"],[1435776700000, "string36"],[1435776800000, true],[1435776900000, 3, 0]]}]

**查询示例**

{"start": 1435776300000,"end": 1435779600000,"tags": [{"name": "test.trend.mode.mixedType","aggregations": [{"type": "trendmode","sampling": {"datapoints": 1}}]}]}

**结果示例**

{"tags": [{"name": "test.trend.mode.mixedType","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1435776300000, 2, 1],[1435776550000, 100, 2]]}],"stats": {"rawCount": 4}}]}

筛选器

您可以使用一组比较运算按照属性、质量和多个标签名筛选数据点，包括以下内容：

| **运算** | **属性** |
| --- | --- |
| 大于（>）) | gt |
| 大于（ >）或等于（=）) | ge |
| 小于（<） | lt |
| 小于（<）或等于（=） | le |
| 等于（=） | eq |
| 不等于（≠） | ne |

**通过多个标签进行大于或等于运算**

以下代码示例显示了对时间序列服务实例的一个查询，包括用于大于（>）或等于（=）一个特定数据点值的数据点的多个标签、一个时间窗口和属性。JSON请求有效载荷示例：

{    "start": 1432671129000,    "end": 1432671130500,    "tags": [        {            "name": [                "tag119",                "tag121"            ],            "filters": {                "measurements": {                    "condition": "ge",                    "values": "2"                }            }        }    ]}

下面显示了使用时间序列客户端库通过多个标签进行查询筛选的一个示例：

QueryBuilder builder = QueryBuilder.createQuery().withStartAbs(1432671129000L).withEndAbs(1432671130500L).addTags(QueryTag.Builder.createQueryTag().withTagNames(Arrays.asList("tag119", "tag121")).addFilters(FilterBuilder.getInstance().addMeasurementFilter(FilterBuilder.Condition.GREATER\_THAN\_OR\_EQUALS, Arrays.asList("2")).build()).build());QueryResponse queryResponse = ClientFactory.queryClientForTenant(tenant).queryAll(builder.build());

**按照属性筛选**

以下示例筛选一个或多个标签的值：

{    "start": 1349109376000,    "end": 1349109381000,    "tags": [        {            "name": [                "<TagName1>",                "<TagName2>"            ],            "filters": {                "attributes": {                    "< AttributeKey1>": "<AttributeValue1>"                }            }        }    ]}

以下示例筛选每个标签的一个不同值：

{    "start": 1349109376000,    "end": 1349109381000,    "tags": [        {            "name": "<phosphate.level1>",            "filters": {                "attributes": {                    "< AttributeKey1>": "<AttributeValue1>"                }            }        },        {            "name": "<phosphate.level2>",            "filters": {                "attributes": {                    "< AttributeKey2>": "<AttributeValue2>"                }            }        }    ]}

**按照质量筛选**

以下示例筛选一个或多个标签的质量：

{    "start": 1349109376000,    "end": 1349109381000,    "tags": [        {            "name": [                "<TagName1>",                "<TagName2>"            ],            "filters": {                "qualities": {                    "values": ["0", "1"]                }            }        }    ]}

以下示例筛选每个标签的不同质量：

{    "start": 1349109376000,    "end": 1349109381000,    "tags": [        {            "name": "<TagName1>",            "filters": {                "qualities": {                    "values": ["0", "1"]                }            }        },        {            "name": "<TagName2>",            "filters": {                "qualities": {                    "values": ["0", "1"]                }            }        }    ]}

用户组

在查询中使用groups选项返回特定用户组内的数据点。

**按照属性分组**

您可以按照属性值对数据点进行分组，如以下示例所示：

{"start":1432671128000,"end":1432671129000,"tags":[{"name":["tagname1","tagname2"],"groups":[{"name":"attribute","attributes":["attributename1"]}]}]}

**按照质量分组**

您可以根据质量对数据点进行分组，如以下示例所示：

{"start": 1432671128000,"end": 1432671129000,"tags": [{"name": ["tagname1","tagname2"],"groups": [{"name": "quality"}]}]}

**按照测量值分组**

您可以根据测量值对数据点结果进行分组，如以下示例所示：

{"start": "15d-ago","end": "1mi-ago","tags": [{"name": "tagname","groups": [{"name": "measurement","rangeSize": 3}]}]}

rangeSize是一个用户组中数值的数量。例如，范围大小10将0-9之间的测量值放在一个用户组中，10-19放在另一个用户组中，以此类推。

**按照时间分组**

您可以按照一个时间范围将数据点结果分组，从查询的开始时间开始，如以下示例所示：

{"start": 1432671128000,"end": 1432671138000,"tags": [{"name": ["tagname1","tagname2"],"groups": [{"name": "time","rangeSize": "1h","groupCount": 24}]}]}

groupCount属性定义了要返回的用户组的最大数量。在上面的示例中，rangeSize为“1h”，groupCount为“24”，返回值是一天范围，按一小时分组。

通过良好和较差质量求数据插值

本部分显示了当一个特定标签（或整型类型）有质量差的数据时，按照时间范围内插数据点的示例。

| 表3数据质量值 | |
| --- | --- |
| **数值** | **说明** |
| 0 | 质量很差。 |
| 1 | 质量不确定。 |
| 2 | 不适用。 |
| 3 | 质量良好。如果您未指定质量，默认值为良好（3）。 |

**良好值和较差值数据点之间的插值**

下面显示了Ingest多个数据点的一个Ingestion请求，包含一个良好数据点和一个具有较晚时间戳的较差数据点。请求有效载荷示例：

| **标签名** | **时间戳** | **测量值** | **质量** |
| --- | --- | --- | --- |
| tagName | 1432671129000 | 541 | q=3 |
| tagName | 1432671130000 | 843 | q=0 |

下面显示了一个开始时间和结束时间覆盖上面Ingestion请求所创建的数据点的查询，其聚合间隔小于两个数据点之间的时间窗口。请求有效载荷示例：

{"start":1432671129000,"end":1432671130500,"tags":[{"name":"coolingtower.corrosion.level","aggregations":[{"type":"interpolate","interval": "500ms"}]}]}

结果显示良好和较差数据点之间的内插点具有良好数据点的数值。结果示例：

{"tags": [{"name": "coolingtower.corrosion.level","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1432671129500,541,3],[1432671130000,0,0],[1432671130500,0,0]]}],"stats": {"rawCount": 2}}]}

**较差值和良好值数据点之间的插值**

下面显示了Ingest多个数据点的一个Ingestion请求，包含一个较差数据点和一个具有较晚时间戳的良好数据点。请求有效载荷示例：

| **标签名** | **时间戳** | **测量值** | **质量** |
| --- | --- | --- | --- |
| tagName | 1432671129000 | 541 | q=0 |
| tagName | 1432671130000 | 843 | q=3 |

下面显示了开始时间和结束时间涵盖上面Ingestion请求所创建数据点的查询，其聚合间隔小于两个数据点之间的时间窗口。请求有效载荷示例：

{"start":1432671129000,"end":1432671130500,"tags":[{"name":"coolingtower.corrosion.level","aggregations":[{"type":"interpolate","interval": "500ms"}]}]}

预期结果是较差和良好数据点之间的内插点其数值为0（零）。结果示例：

{"tags": [{"name": "coolingtower.corrosion.level","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1432671129500,0,0],[1432671130000,843,3],[1432671130500,843,3]]}],"stats": {"rawCount": 2}}]}

**两个较差值之间的插值**

下面显示了一个Ingestion请求，包含一个较差数据点和另一个具有较晚时间戳的较差数据点：

| **标签名** | **时间戳** | **测量值** | **质量** |
| --- | --- | --- | --- |
| tagName | 1432671129000 | 541 | q=0 |
| tagName | 1432671130000 | 843 | q=0 |

下面显示了开始时间和结束时间完全覆盖之前示例中所创建两个数据点的查询，其聚合间隔小于数据点之间的时间窗口。请求有效载荷示例：

{"start":1432671129000,"end":1432671130500,"tags":[{"name":"coolingtower.corrosion.level","aggregations":[{"type":"interpolate","interval": "500ms"}]}]}

预期结果是较差数据点之间的内插点数值为0（零）。结果示例：

{"tags": [{"name": "coolingtower.corrosion.level","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1432671129500,0,0],[1432671130000,0,0],[1432671130500,0,0]]}],"stats": {"rawCount": 2}}]}

**一个较差数据点的插值**

下面显示了Ingest多个数据点的一个Ingestion请求，至少包含一个较差数据点。请求有效载荷示例：

| **标签名** | **时间戳** | **测量值** | **质量** |
| --- | --- | --- | --- |
| tagName | 1432671129000 | 541 | q=0 |
| tagName | 1432671130000 | 843 | q=3 |

下面显示了开始时间等于较差数据点的时间戳的一个数据查询。请求有效载荷示例：

{"start":1432671129000,"end":1432671130500,"tags":[{"name":"coolingtower.corrosion.level","aggregations":[{"type":"interpolate","interval": "500ms"}]}]}

预期结果是较差数据点处的内插点具有数值0（零）。结果示例：

{"tags": [{"name": "coolingtower.corrosion.level","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1432671129500,0],[1432671130000,843,3],[1432671130500,843,3]]}],"stats": {"rawCount": 2}}]}

通过良好和较差质量以及双精度数据类型求数据插值

本部分显示了当一个浮点数（双精度）类型的特定标签有质量差的数据时，按照时间范围内插数据点的示例。

| 表4数据质量值 | |
| --- | --- |
| **数值** | **说明** |
| 0 | 质量很差。 |
| 1 | 质量不确定。 |
| 2 | 不适用。 |
| 3 | 质量良好。如果您未指定质量，默认值为良好（3）。 |

**良好值和较差值数据点之间的插值**

下面显示了Ingest多个数据点的一个Ingestion请求，包含一个良好数据点和一个具有较晚时间戳的较差数据点。请求有效载荷示例：

| **标签名** | **时间戳** | **测量值** | **质量** |
| --- | --- | --- | --- |
| tagName | 1432671129000 | 541.26 | q=3 |
| tagName | 1432671130000 | 843.25 | q=0 |

下面显示了开始时间和结束时间覆盖上面Ingestion请求所创建数据点的查询，其聚合间隔小于两个数据点之间的时间窗口。请求有效载荷示例：

{"start":1432671129000,"end":1432671130500,"tags":[{"name":"coolingtower.corrosion.level","aggregations":[{"type":"interpolate","interval": "500ms"}]}]}

结果显示良好和较差数据点之间的内插点具有良好数据点的数值。结果示例：

{"tags": [{"name": "coolingtower.corrosion.level","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1432671129500,541.26,3],[1432671130000,0,0],[1432671130500,0,0]]}],"stats": {"rawCount": 2}}]}

**较差值和良好值数据点之间的插值**

下面显示了Ingest多个数据点的一个Ingestion请求，包含一个较差数据点和一个具有较晚时间戳的良好数据点。请求有效载荷示例：

| **标签名** | **时间戳** | **测量值** | **质量** |
| --- | --- | --- | --- |
| tagName | 1432671129000 | 541.26 | q=0 |
| tagName | 1432671130000 | 843.25 | q=3 |

下面显示了开始时间和结束时间覆盖上面Ingestion请求所创建数据点的查询，其聚合间隔小于两个数据点之间的时间窗口。请求有效载荷示例：

{"start":1432671129000,"end":1432671130500,"tags":[{"name":"coolingtower.corrosion.level","aggregations":[{"type":"interpolate","interval": "500ms"}]}]}

预期结果是较差和良好数据点之间的内插点具有数值0.0。结果示例：

{"tags": [{"name": "coolingtower.corrosion.level","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1432671129500,0,0],[1432671130000,843.25,3],[1432671130500,843.25,3]]}],"stats": {"rawCount": 2}}]}

**两个较差值之间的插值**

下面显示了一个Ingestion请求，包含一个较差数据点和另一个具有较晚时间戳的较差数据点：

| **标签名** | **时间戳** | **测量值** | **质量** |
| --- | --- | --- | --- |
| tagName | 1432671129000 | 541.26 | q=0 |
| tagName | 1432671130000 | 843.25 | q=0 |

下面显示了开始时间和结束时间完全覆盖之前示例中所创建两个数据点的查询，其聚合间隔小于数据点之间的时间窗口。请求有效载荷示例：

{"start":1432671129000,"end":1432671130500,"tags":[{"name":"coolingtower.corrosion.level","aggregations":[{"type":"interpolate","interval": "500ms"}]}]}

预期结果是较差数据点之间的内插点具有数值0.0。结果示例：

{"tags": [{"name": "coolingtower.corrosion.level","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1432671129500,0,0],[1432671130000,0,0],[1432671130500,0,0]]}],"stats": {"rawCount": 2}}]

**两个良好值之间的插值**

下面显示了Ingest多个数据点的一个Ingestion请求，包含一个良好数据点和另一个具有较晚时间戳的良好数据点：

| **标签名** | **时间戳** | **测量值** | **质量** |
| --- | --- | --- | --- |
| tagName | 1432671129000 | 541.26 | q=3 |
| tagName | 1432671130000 | 843.25 | q=3 |

下面显示了开始和结束时间覆盖两个良好数据点的一个数据查询。

{    "start": 1432671129000,    "end": 1432671130500,    "tags": [        {            "name": "coolingtower.corrosion.level",            "aggregations": [                {                    "type": "interpolate",                    "interval": "500ms"                }            ]        }    ]}

下面显示了预期结果：

{"tags": [{"name": "test.interpolation.mixed.quality.integer","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1432671129500,692.255,3],[1432671130000,843.25,3],[1432671130500,843.25,3]]}],"stats": {"rawCount": 2}}]}

**一个较差数据点的插值**

下面显示了Ingest多个数据点的一个Ingestion请求，至少包含一个较差数据点。请求有效载荷示例：

| **标签名** | **时间戳** | **测量值** | **质量** |
| --- | --- | --- | --- |
| tagName | 1432671129000 | 541.26 | q=0 |
| tagName | 1432671130000 | 843.25 | q=3 |

下面显示了开始时间等于较差数据点的时间戳的一个数据查询。请求有效载荷示例：

{"start":1432671129000,"end":1432671130500,"tags":[{"name":"coolingtower.corrosion.level","aggregations":[{"type":"interpolate","interval": "500ms"}]}]}

预期结果是较差数据点处的内插点具有数值0.0。结果示例：

{"tags": [{"name": "coolingtower.corrosion.level","results": [{"groups": [{"name": "type","type": "number"}],"attributes": {},"values": [[1432671129500,0,0],[1432671130000,843.25,3],[1432671130500,843.25,3]]}],"stats": {"rawCount": 2}}]}

解除一个应用程序与时间序列服务的绑定

解除一个服务实例的绑定：

cf unbind-service <application\_name> <my\_time\_series\_instance>

其中：

* *application\_name*是您的应用程序的名称。
* *my\_time\_series\_instance*是您的服务实例的名称。

删除一个时间序列服务实例

删除一个服务实例：

cf delete-service <service\_instance\_name>