InfluxDB 学习手册

作者：hardy

Table of Contents

InfluxDB简介

InfluxDB 是用Go语言编写的一个开源分布式时序、事件和指标数据库，无需外部依赖。
类似的数据库有Elasticsearch、Graphite等。

其主要特色功能

1. 基于时间序列，支持与时间有关的相关函数（如最大，最小，求和等）
2. 可度量性：你可以实时对大量数据进行计算
3. 基于事件：它支持任意的事件数据

InfluxDB的主要特点

1. 无结构（无模式）：可以是任意数量的列
2. 可拓展的
3. 支持min, max, sum, count, mean, median 等一系列函数，方便统计
4. 原生的HTTP支持，内置HTTP API
5. 强大的类SQL语法
6. 自带管理界面，方便使用

InfluxDB安装

本文以写这篇文章时的最新稳定版（Stable v0.13.0）为例，介绍下InfluxDB的安装。

OS X (via Homebrew)

```
brew update
brew install influxdb
MD5: 4f0aa76fee22cf4c18e2a0779ba4f462
```

Ubuntu & Debian (64-bit)

```
wget https://dl.influxdata.com/influxdb/releases/influxdb_0.13.0
sudo dpkg -i influxdb_0.13.0_amd64.deb
MD5: bcca4c91bbd8e7f60e4a8325be67a08a
```

Ubuntu & Debian (ARM)

```
wget https://dl.influxdata.com/influxdb/releases/influxdb_0.13.0
sudo dpkg -i influxdb_0.13.0_armhf.deb
MD5: b64ada82b6abf5d6382ed08dde1e8579
```

RedHat & CentOS (64-bit)

```
wget https://dl.influxdata.com/influxdb/releases/influxdb-0.13.0
sudo yum localinstall influxdb-0.13.0.x86_64.rpm
MD5: 286b6c18aa4ef37225ea6605a729b61d
```

RedHat & CentOS (ARM)

```
wget https://dl.influxdata.com/influxdb/releases/influxdb-0.13.0
sudo yum localinstall influxdb-0.13.0.armhf.rpm
MD5: 4cf99debb5315fbbb26166506807d965
```

Standalone Binaries (64-bit)

```
wget https://dl.influxdata.com/influxdb/releases/influxdb-0.13.0
tar xvfz influxdb-0.13.0_linux_amd64.tar.gz
MD5: 187854536393c67f7793ada1c096da8e
```

Standalone Binaries (ARM)

```
wget https://dl.influxdata.com/influxdb/releases/influxdb-0.13.0
tar xvfz influxdb-0.13.0_linux_armhf.tar.gz
```

Docker Image

```
docker pull influxdb
```

InfluxDB启动

服务端启动

如果是通过包安装的，可以使用如下语句启动：

```
sudo service influxdb start
```

如果直接下载的二进制包，则进入InfluxDB目录下的usr/bin文件夹，执行：

```
./influxd
```

即可。
这样就启动了服务端。

客户端

在usr/bin里使用influx即可登入Influx服务器。也可以将路径加入环境变量中，这样既可在任意地方使用influx。

InfluxDB自带web管理界面，在浏览器中输入 http://服务器IP:8083 即可进入web管理页面。

安装后web页面无法访问的解决方案

问题原因

InfluxDB在0.13版本以后，就默认关闭了web管理页面，而国内的文档大多都以旧版的InfluxDB为标准写的，所以下载安装好最新版本以后，就会出现8083端口的web管理页面访问不了的问题。

解决方案

新版的InfluxDB虽然默认关闭了web管理页面，但我们可以通过很简单的方式进行开启。

打开配置文件，找到如下几行：

```
187  # [admin]
188    # Determines whether the admin service is enabled.
189    # enabled = false
190
191    # The default bind address used by the admin service.
192    # bind-address = ":8083"
193
194    # Whether the admin service should use HTTPS.
195    # https-enabled = false
196
197    # The SSL certificate used when HTTPS is enabled.
198    # https-certificate = "/etc/ssl/influxdb.pem"
199
200  ###
```

将这几个配置之前的注释号去掉，将enabled改为true即可，[admin]这个的#号也要去掉。

https酌情开启。

另外，在配置文件中还写着，这个web管理界面将在1.1以后的版本中删除。

本人感觉这个web界面还是挺方便的，停用以后无疑会给我们的工作造成很大不便，对他们的这个决定不是很理解。

InfluxDB概念

InfluxDB与传统数据库在概念上有许多的不同，本文就给大家介绍下InfluxDB中的一些基本概念。

InfluxDB基本概念

与传统数据库中的名词做比较

influxDB中的名词	传统数据库中的概念
database	数据库
measurement	数据库中的表
points	表里面的一行数据

InfluxDB中独有的概念

1. tag--标签：在InfluxDB中，tag是一个非常重要的部分，表名+tag一起作为数据库的索引，是“key-value”的形式。
2. field--数据：field主要是用来存放数据的部分，也是“key-value”的形式。
3. timestamp--时间戳：作为时序型数据库，时间戳是InfluxDB中最重要的部分，在插入数据时可以自己指定也可留空让系统指定。 说明：在插入新数据时，tag、field和timestamp之间用空格分隔。
4. series--序列：所有在数据库中的数据，都需要通过图表来展示，而这个series表示这个表里面的数据，可以在图表上画成几条线。
5. Retention policy--数据保留策略：可以定义数据保留的时长，每个数据库可以有多个数据保留策略，但只能有一个默认策略。
6. Point--点：表示每个表里某个时刻的某个条件下的一个field的数据，因为体现在图表上就是一个点，于是将其称为point。

InfluxDB关键概念

InfluxDB特点

- 可以设置metric的保存时间。
- 支持通过条件过滤以及正则表达式删除数据。
- 支持类似 sql 的语法。
- 可以设置数据在集群中的副本数。
- 支持定期采样数据，写入另外的measurement，方便分粒度存储数据。

InfluxDB概念

数据格式 Line Protocol

在 InfluxDB 中，我们可以粗略的将要存入的一条数据看作一个虚拟的 key 和其对应的 value(field value)，格式如下：

cpu_usage,host=server01,region=us-west value=0.64 1434055562000000000

虚拟的 key 包括以下几个部分： database, retention policy, measurement, tag sets, field name, timestamp。 database 和 retention policy 在上面的数据中并没有体现，通常在插入数据时在 http 请求的相应字段中指定。

database: 数据库名，在 InfluxDB 中可以创建多个数据库，不同数据库中的数据文件是隔离存放的，存放在磁盘上的不同目录。

retention policy: 存储策略，用于设置数据保留的时间，每个数据库刚开始会自动创建一个默认的存储策略 autogen，数据保留时间为永久，之后用户可以自己设置，例如保留最近2小时的数据。插入和查询数据时如果不指定存储策略，则使用默认存储策略，且默认存储策略可以修改。InfluxDB 会定期清除过期的数据。

measurement: 测量指标名，例如 cpu_usage 表示 cpu 的使用率。

tag sets: tags 在 InfluxDB 中会按照字典序排序，不管是 tagk 还是 tagv，只要不一致就分别属于两个 key，例如 host=server01,region=us-west 和 host=server02,region=us-west 就是两个不同的 tag set。

field name: 例如上面数据中的 value 就是 fieldName，InfluxDB 中支持一条数据中插入多个 fieldName，这其实是一个语法上的优化，在实际的底层存储中，是当作多条数据来存储。

timestamp: 每一条数据都需要指定一个时间戳，在 TSM 存储引擎中会特殊对待，以为了优化后续的查询操作。

Point

InfluxDB 中单条插入语句的数据结构，series + timestamp 可以用于区别一个 point，也就是说一个 point 可以有多个 field name 和 field value。

Series

series 相当于是 InfluxDB 中一些数据的集合，在同一个 database 中，retention policy、measurement、tag sets 完全相同的数据同属于一个 series，同一个 series 的数据在物理上会按照时间顺序排列存储在一起。

series 的 key 为 measurement + 所有 tags 的序列化字符串，这个 key 在之后会经常用到。

代码中的结构如下：

```
type Series struct {
    mu          sync.RWMutex
    Key         string              // series key
    Tags        map[string]string   // tags
    id          uint64              // id
    measurement *Measurement        // measurement
}
```

Shard

shard 在 InfluxDB 中是一个比较重要的概念，它和 retention policy 相关联。每一个存储策略下会存在许多 shard，每一个 shard 存储一个指定时间段内的数据，并且不重复，例如 7点-8点 的数据落入 shard0 中，8点-9点的数据则落入 shard1 中。每一个 shard 都对应一个底层的 tsm 存储引擎，有独立的 cache、wal、tsm file。

创建数据库时会自动创建一个默认存储策略，永久保存数据，对应的在此存储策略下的 shard 所保存的数据的时间段为 7 天，计算的函数如下：

```
func shardGroupDuration(d time.Duration) time.Duration {
    if d >= 180*24*time.Hour || d == 0 { // 6 months or 0
        return 7 * 24 * time.Hour
    } else if d >= 2*24*time.Hour { // 2 days
        return 1 * 24 * time.Hour
    }
    return 1 * time.Hour
}
```

如果创建一个新的 retention policy 设置数据的保留时间为 1 天，则单个 shard 所存储数据的时间间隔为 1 小时，超过1个小时的数据会被存放到下一个 shard 中。

InfluxDB原理详解

感兴趣的可以访问地址： https://www.linuxdaxue.com/influxdb-principle.html

InfluxDB操作

InfluxDB提供三种操作方式：

1. 客户端命令行方式
2. HTTP API接口
3. 各语言API库

基本操作

如同MYSQL一样，InfluxDB提供多数据库支持，对数据库的操作也与MYSQL相同。

数据库操作

显示数据库：

```
> show databases
name: databases
---------------
name
telegraf
_internal
lir
testDB
testMyDb
```

新建数据库：

```
> create database test
> show databases
name: databases
---------------
name
telegraf
_internal
lir
testDB
testMyDb
xk_name
test
```

删除数据库

```
> drop database test
> show databases
name: databases
---------------
name
telegraf
_internal
lir
testDB
testMyDb
xk_name
```

使用某个数据库

```
> use xk_name
Using database xk_name
```

数据表操作

在InfluxDB当中，并没有表（table）这个概念，取而代之的是MEASUREMENTS，MEASUREMENTS的功能与传统数据库中的表一致，因此我们也可以将MEASUREMENTS称为InfluxDB中的表。

显示所有表

```
>SHOW MEASUREMENTS
name: measurements
------------------
name
weather
```

新建表

InfluxDB中没有显式的新建表的语句，只能通过insert数据的方式来建立新表。如下所示：

```
insert disk_free,hostname=server01 value=442221834240i 143536218
```

其中 disk_free 就是表名，hostname是索引，value=xx是记录值，记录值可以有多个，最后是指定的时间

执行后结果如下

```
> select * from disk_free
name: disk_free
---------------
time            hostname    value
1435362189575692182    server01    442221834240
```

删除表

```
> drop measurement disk_free
> show measurements
name: measurements
------------------
name
weather
```

数据操作

增加数据

增加数据采用insert的方式，要注意的是 InfluxDB的insert中，表名与数据之间用逗号（,）分隔，tag和field之间用 空格分隔，多个tag或者多个field之间用逗号（,）分隔。

```
> insert disk_free,hostname=server01 value=442221834240i 1435362
> select * from disk_free
name: disk_free
---------------
time            hostname    value
1435362189575692182    server01    442221834240
```

在这条语句中，disk_free是表名,hostname=server01是tag，属于索引，value=xx是field，这个可以随意写，随意定义。

查询数据

查询语句与SQL一样，在此不再赘述。

修改和删除数据

InfluxDB属于时序数据库，没有提供修改和删除数据的方法。

但是删除可以通过InfluxDB的数据保存策略（Retention Policies）来实现，这个会在以后的文章中讲到。

series操作

series表示这个表里面的数据，可以在图表上画成几条线，series主要通过tags排列组合算出来。

我们可以查询表的series，如下所示：

```
> show series from mem
key
mem,host=ResourcePool-0246-billing07
mem,host=billing07
```

界面操作

InfluxDB还提供了管理界面，大大降低了入门难度，在启动了InfluxDB服务之后，直接输入 `<IP>:8083` 即可访问界面。

数据保留策略

InfluxDB每秒可以处理成千上万条数据，要将这些数据全部保存下来会占用大量的存储空间，有时我们可能并不需要将所有历史数据进行存储，因此，InfluxDB推出了数据保留策略（Retention Policies），用来让我们自定义数据的保留时间。

说明

InfluxDB的数据保留策略（RP） 用来定义数据在InfluxDB中存放的时间，或者定义保存某个期间的数据。

一个数据库可以有多个保留策略，但每个策略必须是独一无二的。

目的

InfluxDB本身不提供数据的删除操作，因此用来控制数据量的方式就是定义数据保留策略。

因此定义数据保留策略的目的是让InfluxDB能够知道可以丢弃哪些数据，从而更高效的处理数据。

操作

查询策略

可以通过如下语句查看数据库的现有策略：

```
> SHOW RETENTION POLICIES ON telegraf
name    duration    shardGroupDuration    replicaN    default
default    0        168h0m0s        1        true
```

可以看到，telegraf只有一个策略，各字段的含义如下：

name--名称，此示例名称为 default

duration--持续时间，0代表无限制

shardGroupDuration--shardGroup的存储时间，shardGroup是InfluxDB的一个基本储存结构，应该大于这个时间的数据在查询效率上应该有所降低。

replicaN--全称是REPLICATION，副本个数

default--是否是默认策略

新建策略

```
> CREATE RETENTION POLICY "2_hours" ON "telegraf" DURATION 2h RE
> SHOW RETENTION POLICIES ON telegraf
name    duration    shardGroupDuration    replicaN    default
default    0        168h0m0s        1        false
2_hours    2h0m0s        1h0m0s            1        true
```

通过上面的语句可以添加策略，本例在 telegraf 库添加了一个2小时的策略，名字叫做 2_hours， duration为2小时，副本为1，设置为默认策略。

因为名为default的策略不再是默认策略，因此，在查询使用default策略的表时要显式的加上策略名 “default”。

```
> select * from "default".cpu limit 2
name: cpu
---------
time            cpu        host                host_id    usage_
1467884670000000000    cpu-total    ResourcePool-0246-billing07
1467884670000000000    cpu9        billing07                0
```

修改策略

修改策略使用如下语句修改

```
> ALTER RETENTION POLICY "2_hours" ON "telegraf" DURATION 4h DEF
> show retention POLICIES on telegraf
name    duration    shardGroupDuration    replicaN    default
default    0        168h0m0s        1        false
2_hours    4h0m0s        1h0m0s            1        true
```

可以看到，修改后的策略发生了变化。

删除策略

InfluxDB中策略的删除操作如下所示：

```
> drop retention POLICY "2_hours" ON "telegraf"
> show retention POLICIES on telegraf
name    duration    shardGroupDuration    replicaN    default
default    0        168h0m0s        1        false
```

可以看到，名为2_hours的策略已经被删除了。

其他说明

策略这个关键词“POLICY”在使用是应该大写，小写应该会出粗。

当一个表使用的策略不是默认策略时，在进行操作时一定要显式的指定策略名称，否则会出现错误。

连续查询

连续查询主要用在将数据归档，以降低系统空间的占用率，主要是以降低精度为代价。

定义

InfluxDB的连续查询是在数据库中自动定时启动的一组语句，语句中必须包含 SELECT 关键词和 GROUP BY time() 关键词。

InfluxDB会将查询结果放在指定的数据表中。

目的

使用连续查询是最优的降低采样率的方式，连续查询和存储策略搭配使用将会大大降低InfluxDB的系统占用量。

而且使用连续查询后，数据会存放到指定的数据表中，这样就为以后统计不同精度的数据提供了方便。

操作

只有管理员用户可以操作连续查询。

新建连续查询

新建连续查询的语法如下所示：

```
CREATE CONTINUOUS QUERY <cq_name> ON <database_name>
[RESAMPLE [EVERY <interval>] [FOR <interval>]]
BEGIN SELECT <function>(<stuff>)[,<function>(<stuff>)] INTO <dif
FROM <current_measurement> [WHERE <stuff>] GROUP BY time(<interv
END
```

查询部分被 CREATE CONTINUOUS QUERY [...] BEGIN 和 END 所包含，主要的逻辑代码也是在这一部分。

使用示例：

```
> CREATE CONTINUOUS QUERY cq_30m ON telegraf BEGIN SELECT mean(u
> SHOW CONTINUOUS QUERIES
name: telegraf
--------------
name    query
cq_30m    CREATE CONTINUOUS QUERY cq_30m ON telegraf BEGIN
SELECT mean(used) INTO telegraf."default".mem_used_30m FROM tele
GROUP BY time(30m) END


name: _internal
---------------
name    query
```

示例在telegraf库中新建了一个名为 cq_30m 的连续查询，每三十分钟取一个used字段的平均值，加入 mem_used_30m 表中。使用的数据保留策略都是 default。

显示所有已存在的连续查询

查询所有连续查询可以使用如下语句：

```
> SHOW CONTINUOUS QUERIES
name: telegraf
--------------
name    query
cq_30m    CREATE CONTINUOUS QUERY cq_30m ON telegraf
BEGIN SELECT mean(used) INTO telegraf."default".mem_used_30m FRO
GROUP BY time(30m) END


name: _internal
---------------
name    query
```

可以看到其连续查询的名称以及 语句等信息。

删除Continuous Queries

删除连续查询的语句如下：

```
DROP CONTINUOUS QUERY <cq_name> ON <database_name>
```

其他说明

在InfluxDB中，将连续查询与数据存储策略一起使用会达到最好的效果。

比如，将精度高的表的存储策略定为一个周，然后将精度底的表存储策略定的时间久一点，这要就可以实现高低搭配，以满足不同的工作需要。

HTTP API写入操作

说明

为了方便，本文主要使用curl来发起http请求，示例当中也是使用curl这个工具来模拟HTTP 请求。

在实际使用中，可以将请求写入代码中，通过其他编程语言来模拟HTTP请求。

InfluxDB通过HTTP API操作数据库

建立数据库

```
    curl -POST http://localhost:8086/query --data-urlencode "q=C
```

执行这个语句后，会在本地建立一个名为mydb的数据库。

删除数据库

```
    curl -POST http://localhost:8086/query --data-urlencode "q=D
```

其实使用HTTP API就是向 InfluxDB 接口发送相应的POST请求。

将语句通过POST方式发送到服务器。

InfluxDB通过HTTP API添加数据

InfluxDB通过HTTP API添加数据主要使用如下格式：

```
curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-bina
```

说明： db=mydb是指使用mydb这个数据库。

--data-binary后面是需插入数据。

cpu_load_short是表名（measurement），tag字段是host和region，值分别为：server01和us-west。

field key字段是value，值为0.64。

时间戳（timestamp）指定为1434055562000000000。

这样，就向mydb数据库的cpu_load_short表中插入了一条数据。

其中，db参数必须指定一个数据库中已经存在的数据库名，数据体的格式遵从InfluxDB规定格式，首先是表名，后面是tags，然后是field，最后是时间戳。tags、field和时间戳三者之间以空格相分隔。

InfluxDB通过HTTP API添加多条数据

InfluxDB通过HTTP API添加多条数据与添加单条数据相似，示例如下：

```
curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-bina
cpu_load_short,host=server02,region=us-west value=0.55 142256854
cpu_load_short,direction=in,host=server01,region=us-west value=2
```

这条语句向数据库mydb的表cpu_load_short中插入了三条数据。

第一条指定tag为host，值为server02，第二条指定tag为host和region，值分别为server02和us-west，第三条指定tag为direction，host，region，值分别为：in，server01，us-west。

InfluxDB 的HTTP API响应

在使用HTTP API时，InfluxDB的响应主要有以下几个：

1）2xx：204代表no content，200代表InfluxDB可以接收请求但是没有完成请求。一般会在body体中带有出错信息。

2）4xx：InfluxDB不能解析请求。

3）5xx：系统出现错误。

HTTP API查询操作

说明

官方文档上介绍说，使用HTTP API进行查询是比较初级的一种方式。推荐使用第三方语言库和客户端管理程序进行查询操作。

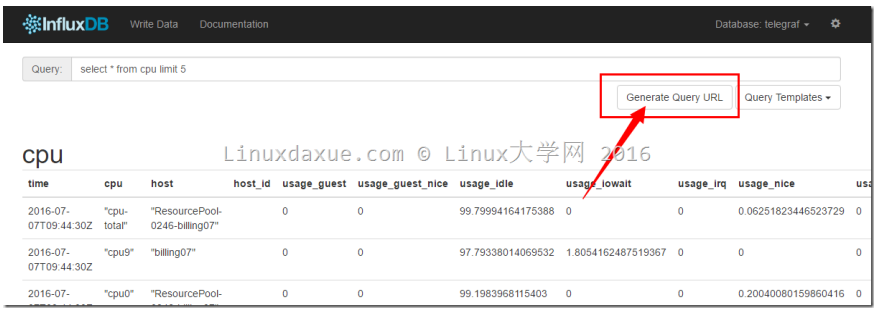
InfluxDB进行HTTP API查询方法

使用HTTP API在InfluxDB进行查询主要是发送 GET 请求到 InfluxDB的 /query 端，调用示例如下所示：

```
curl -GET 'http://localhost:8086/query?pretty=true' --data-urlen
--data-urlencode "q=SELECT value FROM cpu_load_short WHERE regio
```

参数db指定了需查询的数据库，q代表了需执行的查询语句。

在页面中也提供了生成HTTP 请求URL的方法，如下所示：



在界面上输入语句后，点击生成url就会生成http请求URL。

在浏览器执行后，会返回json格式的串。

如果查询出错的话，则会返回关键词“error”+错误信息。

InfluxDB进行HTTP API查询多条数据

我们可能需要用InfluxDB进行多条查询，HTTP API提供的多条查询的格式如下所示：

```
curl -G 'http://localhost:8086/query?pretty=true' --data-urlenco
SELECT count(value) FROM cpu_load_short WHERE region='us-west'"
```

格式与单条查询相同，只是在多条语句之间要用分号“；”分隔。

返回值也是包含结果的json串。

InfluxDB HTTP 查询的格式化输出

规定时间格式

在使用HTTP查询时可以使用 epoch 参数指定输出的时间格式。可选值有 epoch=[h,m,s,ms,u,ns]。

例如：

```
curl -G 'http://localhost:8086/query' --data-urlencode "db=mydb"
--data-urlencode "epoch=s" --data-urlencode "q=SELECT value FROM
```

这样会获取到以秒为单位的时间数据。

指定每次查询数据大小

可以使用 chunk_size 参数来指定每次结果的大小。比如，我要结果每次返回200个点的数据，则如下所示：

```
curl -G 'http://localhost:8086/query' --data-urlencode "db=mydb"
```

这样查询结果就会返回200个点的数据。

InfluxDB常用函数

InfluxDB提供了很多的有用的函数，本文及接下来几篇文章就来给大家介绍下这些常用的函数。

1. 聚合类函数
2. 选择类函数
3. 变换类函数

聚合类函数

count()函数

返回一个（field）字段中的非空值的数量。

语法：

```
SELECT COUNT(<field_key>) FROM <measurement_name> [WHERE <stuff>
```

示例：

```
>SELECT COUNT(water_level) FROM h2o_feet
name: h2o_feet
--------------
time                           count
1970-01-01T00:00:00Z     15258
```

说明 water_level这个字段在 h2o_feet表中共有15258条数据。

注意：InfluxDB中的函数如果没有指定时间的话，会默认以 epoch 0 (1970-01-01T00:00:00Z) 作为时间。

可以在where 中加入时间条件，如下：

```
> SELECT COUNT(water_level) FROM h2o_feet WHERE time >= '2015-08
name: h2o_feet
--------------
time                           count
2015-08-17T00:00:00Z     1440
2015-08-21T00:00:00Z     1920
2015-08-25T00:00:00Z     1920
2015-08-29T00:00:00Z     1920
2015-09-02T00:00:00Z     1915
2015-09-06T00:00:00Z     1920
2015-09-10T00:00:00Z     1920
2015-09-14T00:00:00Z     1920
2015-09-18T00:00:00Z     335
```

这样结果中会包含时间结果。

DISTINCT()函数

返回一个字段（field）的唯一值。

语法：

```
SELECT DISTINCT(<field_key>) FROM <measurement_name> [WHERE <stu
```

使用示例

```
> SELECT DISTINCT("level description") FROM h2o_feet
name: h2o_feet
--------------
time                            distinct
1970-01-01T00:00:00Z     between 6 and 9 feet
1970-01-01T00:00:00Z     below 3 feet
1970-01-01T00:00:00Z     between 3 and 6 feet
1970-01-01T00:00:00Z     at or greater than 9 feet
```

这个例子显示level description这个字段共有四个值，然后将其显示了出来，时间为默认时间。

MEAN() 函数

返回一个字段（field）中的值的算术平均值（平均值）。字段类型必须是长整型或float64。

语法格式：

```
SELECT MEAN(<field_key>) FROM <measurement_name> [WHERE <stuff>]
```

使用示例

```
> SELECT MEAN(water_level) FROM h2o_feet
name: h2o_feet
--------------
time                            mean
1970-01-01T00:00:00Z     4.286791371454075
```

说明water_level字段的平均值为4.286791371454075

时间为默认时间，当然，你也可以加入where条件。

MEDIAN()函数

从单个字段（field）中的排序值返回中间值（中位数）。字段值的类型必须是长整型或float64格式。

语法：

```
SELECT MEDIAN(<field_key>) FROM <measurement_name> [WHERE <stuff
```

使用示例

```
> SELECT MEDIAN(water_level) from h2o_feet
name: h2o_feet
--------------
time                            median
1970-01-01T00:00:00Z     4.124
```

说明表中 water_level字段的中位数是 4.124

SPREAD()函数

返回字段的最小值和最大值之间的差值。数据的类型必须是长整型或float64。

语法：

```
SELECT SPREAD(<field_key>) FROM <measurement_name> [WHERE <stuff
```

使用示例

```
> SELECT SPREAD(water_level) FROM h2o_feet
name: h2o_feet
--------------
time                            spread
1970-01-01T00:00:00Z      10.574
```

SUM()函数

返回一个字段中的所有值的和。字段的类型必须是长整型或float64。

语法：

```
SELECT SUM(<field_key>) FROM <measurement_name> [WHERE <stuff>]
```

使用示例：

```
> SELECT SUM(water_level) FROM h2o_feet
name: h2o_feet
--------------
time                            sum
1970-01-01T00:00:00Z     67777.66900000002
```

此语句计算出了 h2o_feet表中 所有 water_level 字段的和。

选择类函数

TOP()函数

作用：返回一个字段中最大的N个值，字段类型必须是长整型或float64类型。

语法：

```
SELECT TOP( <field_key>[,<tag_key(s)>],<N> )[,<tag_key(s)>|<fiel
```

使用示例

```
> SELECT TOP("water_level",3) FROM "h2o_feet"

name: h2o_feet
time                   top
----                   ---
2015-08-29T07:18:00Z   9.957
2015-08-29T07:24:00Z   9.964
2015-08-29T07:30:00Z   9.954
```

这个例子返回表中 water_level字段中最大的三个值。

BOTTOM()函数

作用：返回一个字段中最小的N个值。字段类型必须是长整型或float64类型。

语法：

```
SELECT BOTTOM(<field_key>[,<tag_keys>],<N>)[,<tag_keys>] FROM <m
```

使用示例

```
> SELECT BOTTOM(water_level,3) FROM h2o_feet
name: h2o_feet
--------------
time                           bottom
2015-08-29T14:30:00Z     -0.61
2015-08-29T14:36:00Z     -0.591
2015-08-30T15:18:00Z     -0.594
```

这个例子返回表中 water_level字段中最小的三个值。

也可将关联tag放在一起查询，但如果tag值少于N的值，则返回的值的个数只会取tag中字段值少的那个。

如下所示：

```
> SELECT BOTTOM(water_level,location,3) FROM h2o_feet
name: h2o_feet
--------------
time                           bottom     location
2015-08-29T10:36:00Z     -0.243     santa_monica
2015-08-29T14:30:00Z     -0.61      coyote_creek
```

语句取最小的三个值，然而结果只返回了2个值，因为 location 这个tag只有 两个取值。

FIRST()函数

作用：返回一个字段中最老的取值。

语法：

```
SELECT FIRST(<field_key>)[,<tag_key(s)>] FROM <measurement_name>
```

示例：

```
> SELECT FIRST(water_level) FROM h2o_feet WHERE location = 'sant
name: h2o_feet
--------------
time                           first
2015-08-18T00:00:00Z     2.064
```

这个语句返回了 在 location为santa_monica条件下，最旧的那个water_level字段的取值和时间。

LAST()函数

作用：返回一个字段中最新的取值。

语法：

```
SELECT LAST(<field_key>)[,<tag_key(s)>] FROM <measurement_name>
```

示例：

```
> SELECT LAST(water_level),location FROM h2o_feet WHERE time >=
name: h2o_feet
--------------
time                            last      location
2015-08-18T00:54:00Z     6.982     coyote_creek
```

MAX()函数

作用：返回一个字段中的最大值。该字段类型必须是长整型，float64，或布尔类型。

语法：

```
SELECT MAX(<field_key>)[,<tag_key(s)>] FROM <measurement_name> [
```

示例：

```
> SELECT MAX(water_level),location FROM h2o_feet
name: h2o_feet
--------------
time                            max       location
2015-08-29T07:24:00Z     9.964     coyote_creek
```

MIN()函数

作用：返回一个字段中的最小值。该字段类型必须是长整型，float64，或布尔类型。

语法：

```
SELECT MIN(<field_key>)[,<tag_key(s)>] FROM <measurement_name> [
```

示例：

```
> SELECT MIN(water_level),location FROM h2o_feet
name: h2o_feet
--------------
time                           min       location
2015-08-29T14:30:00Z    -0.61     coyote_creek
```

PERCENTILE()函数

作用：返回排序值排位为N的百分值。字段的类型必须是长整型或float64。

百分值是介于100到0之间的整数或浮点数，包括100。

语法：

```
SELECT PERCENTILE(<field_key>, <N>)[,<tag_key(s)>] FROM <measure
```

示例：

```
> SELECT PERCENTILE(water_level,5),location FROM h2o_feet
name: h2o_feet
--------------
time                        percentile     location
2015-08-28T12:06:00Z      1.122             santa_monica
```

就是将water_level字段按照不同的location求百分比，然后取第五位数据。

变换类函数

DERIVATIVE()函数

作用：返回一个字段在一个series中的变化率。

InfluxDB会计算按照时间进行排序的字段值之间的差异，并将这些结果转化为单位变化率。其中，单位可以指定，默认为1s。

语法：

```
SELECT DERIVATIVE(<field_key>, [<unit>]) FROM <measurement_name>
```

其中，unit取值可以为以下几种：

u --microseconds s --seconds m --minutes h --hours d --days w --weeks
DERIVATIVE()函数还可以在GROUP BY time()的条件下与聚合函数嵌套使用，格式如下：

```
SELECT DERIVATIVE(AGGREGATION_FUNCTION(<field_key>),[<unit>]) FR
```

示例：

假设location = santa_monica 条件下数据有以下几条：

```
name: h2o_feet
--------------
time                           water_level
2015-08-18T00:00:00Z     2.064
2015-08-18T00:06:00Z     2.116
2015-08-18T00:12:00Z     2.028
2015-08-18T00:18:00Z     2.126
2015-08-18T00:24:00Z     2.041
2015-08-18T00:30:00Z     2.051
```

计算每一秒的变化率：

```
> SELECT DERIVATIVE(water_level) FROM h2o_feet WHERE location =
name: h2o_feet
--------------
time                            derivative
2015-08-18T00:06:00Z     0.00014444444444444457
2015-08-18T00:12:00Z     -0.00024444444444444465
2015-08-18T00:18:00Z     0.0002722222222222218
2015-08-18T00:24:00Z     -0.00023611111111111
2015-08-18T00:30:00Z     2.777777777777842e-05
```

第一行数据的计算公式为(2.116 - 2.064) / (360s / 1s)

计算每六分钟的变化率

```
> SELECT DERIVATIVE(water_level,6m) FROM h2o_feet WHERE location
name: h2o_feet
--------------
time                            derivative
2015-08-18T00:06:00Z     0.052000000000000046
2015-08-18T00:12:00Z     -0.08800000000000008
2015-08-18T00:18:00Z     0.09799999999999986
2015-08-18T00:24:00Z     -0.08499999999999996
2015-08-18T00:30:00Z     0.010000000000000231
```

第一行数据的计算过程如下：(2.116 - 2.064) / (6m / 6m)

计算每12分钟的变化率：

```
> SELECT DERIVATIVE(water_level,12m) FROM h2o_feet WHERE locatio
name: h2o_feet
--------------
time                            derivative
2015-08-18T00:06:00Z     0.10400000000000009
2015-08-18T00:12:00Z     -0.17600000000000016
2015-08-18T00:18:00Z     0.19599999999999973
2015-08-18T00:24:00Z     -0.16999999999999993
2015-08-18T00:30:00Z     0.020000000000000462
```

第一行数据计算过程为：(2.116 - 2.064 / (6m / 12m)

计算每12分钟最大值的变化率

```
> SELECT DERIVATIVE(MAX(water_level)) FROM h2o_feet WHERE locati
name: h2o_feet
--------------
time                           derivative
2015-08-18T00:12:00Z     0.009999999999999787
2015-08-18T00:24:00Z     -0.07499999999999973
```

这个函数功能非常多，也非常复杂，更多对于此功能的详细解释请看官网：
https://docs.influxdata.com/influxdb/v0.13/query_language/functions/#derivative

DIFFERENCE()函数

作用：返回一个字段中连续的时间值之间的差异。字段类型必须是长整型或float64。

最基本的语法：

```
SELECT DIFFERENCE(<field_key>) FROM <measurement_name> [WHERE <s
```

与GROUP BY time()以及其他嵌套函数一起使用的语法格式：

```
SELECT DIFFERENCE(<function>(<field_key>)) FROM <measurement_nam
```

其中，函数可以包含以下几个：

COUNT(), MEAN(), MEDIAN(),SUM(), FIRST(), LAST(), MIN(), MAX(), 和 PERCENTILE()。

使用示例

例子中使用的源数据如下所示：

```
> SELECT water_level FROM h2o_feet WHERE location='santa_monica'
name: h2o_feet
--------------
time                           water_level
2015-08-18T00:00:00Z      2.064
2015-08-18T00:06:00Z      2.116
2015-08-18T00:12:00Z      2.028
2015-08-18T00:18:00Z      2.126
2015-08-18T00:24:00Z      2.041
2015-08-18T00:30:00Z      2.051
2015-08-18T00:36:00Z      2.067
```

计算water_level间的差异：

```
> SELECT DIFFERENCE(water_level) FROM h2o_feet WHERE location='s
name: h2o_feet
--------------
time                           difference
2015-08-18T00:06:00Z      0.052000000000000046
2015-08-18T00:12:00Z      -0.08800000000000008
2015-08-18T00:18:00Z      0.09799999999999986
2015-08-18T00:24:00Z      -0.08499999999999996
2015-08-18T00:30:00Z      0.010000000000000231
2015-08-18T00:36:00Z      0.016000000000000014
```

数据类型都为float类型。

ELAPSED()函数

作用：返回一个字段在连续的时间间隔间的差异，间隔单位可选，默认为1纳秒。

单位可选项如下图：

image

语法：

```
SELECT ELAPSED(<field_key>, <unit>) FROM <measurement_name> [WHE
```

示例：

计算h2o_feet字段在纳秒间隔下的差异。

```
> SELECT ELAPSED(water_level) FROM h2o_feet WHERE location = 'sa
name: h2o_feet
--------------
time                            elapsed
2015-08-18T00:06:00Z      360000000000
2015-08-18T00:12:00Z      360000000000
2015-08-18T00:18:00Z      360000000000
2015-08-18T00:24:00Z      360000000000
```

在一分钟间隔下的差异率：

```
> SELECT ELAPSED(water_level,1m) FROM h2o_feet WHERE location =
name: h2o_feet
--------------
time                            elapsed
2015-08-18T00:06:00Z      6
2015-08-18T00:12:00Z      6
2015-08-18T00:18:00Z      6
2015-08-18T00:24:00Z      6
```

注意：如果设置的时间间隔比字段数据间的时间间隔更大时，则函数会返回0，如下所示：

```
> SELECT ELAPSED(water_level,1h) FROM h2o_feet WHERE location =
name: h2o_feet
--------------
time                            elapsed
2015-08-18T00:06:00Z      0
2015-08-18T00:12:00Z      0
2015-08-18T00:18:00Z      0
2015-08-18T00:24:00Z      0
```

MOVING_AVERAGE()函数

作用：返回一个连续字段值的移动平均值，字段类型必须是长整形或者float64类型。

语法：

基本语法

```
SELECT MOVING_AVERAGE(<field_key>,<window>) FROM <measurement_na
```

与其他函数和GROUP BY time()语句一起使用时的语法

```
SELECT MOVING_AVERAGE(<function>(<field_key>),<window>) FROM <me
```

此函数可以和以下函数一起使用：

COUNT(), MEAN(),MEDIAN(), SUM(), FIRST(), LAST(), MIN(), MAX(), and PERCENTILE().

示例：

```
> SELECT water_level FROM h2o_feet WHERE location = 'santa_monic
name: h2o_feet
--------------
time                           water_level
2015-08-18T00:00:00Z      2.064
2015-08-18T00:06:00Z      2.116
2015-08-18T00:12:00Z      2.028
2015-08-18T00:18:00Z      2.126
2015-08-18T00:24:00Z      2.041
2015-08-18T00:30:00Z      2.051
2015-08-18T00:36:00Z      2.067
```

NON_NEGATIVE_DERIVATIVE()函数

作用：返回在一个series中的一个字段中值的变化的非负速率。

语法：

```
SELECT NON_NEGATIVE_DERIVATIVE(<field_key>, [<unit>]) FROM <meas
```

其中unit取值可以为以下几个：

image

与聚合类函数放在一起使用时的语法如下所示：

```
SELECT NON_NEGATIVE_DERIVATIVE(AGGREGATION_FUNCTION(<field_key>)
```

此函数示例请参阅：DERIVATIVE()函数

STDDEV()函数

作用：返回一个字段中的值的标准偏差。值的类型必须是长整型或float64类型。

语法：

```
SELECT STDDEV(<field_key>) FROM <measurement_name> [WHERE <stuff
```

示例：

```
> SELECT STDDEV(water_level) FROM h2o_feet
name: h2o_feet
--------------
time                            stddev
1970-01-01T00:00:00Z     2.279144584196145
```

示例2：

```
> SELECT STDDEV(water_level) FROM h2o_feet WHERE time >= '2015-0
name: h2o_feet
tags: location = coyote_creek
time                            stddev
----                            ------
2015-08-13T00:00:00Z     2.2437263080193985
2015-08-20T00:00:00Z     2.121276150144719
2015-08-27T00:00:00Z     3.0416122170786215
2015-09-03T00:00:00Z     2.5348065025435207
2015-09-10T00:00:00Z     2.584003954882673
2015-09-17T00:00:00Z     2.2587514836274414

name: h2o_feet
tags: location = santa_monica
time                            stddev
----                            ------
2015-08-13T00:00:00Z     1.11156344587553
2015-08-20T00:00:00Z     1.0909849279082366
2015-08-27T00:00:00Z     1.9870116180096962
2015-09-03T00:00:00Z     1.3516778450902067
2015-09-10T00:00:00Z     1.4960573811500588
2015-09-17T00:00:00Z     1.075701669442093
```

InfluxDB数据管理

InfluxDB提供了数据的备份和恢复方法，在实际工作中，可以通过这些方法来实现数据的高可用。

本地备份

备份元数据

influxDB本地备份元数据的语法如下，这只会备份InfluxDB的的internal库数据，包含那些最基本的系统信息、用户信息等。

```
influxd backup <path-to-backup>
```

示例：

```
$ influxd backup /tmp/backup
2016/02/01 17:15:03 backing up metastore to /tmp/backup/meta.00
2016/02/01 17:15:03 backup complete
```

备份数据库

可以通过 -database 参数来指定备份的数据库。

语法：

```
influxd backup -database <mydatabase> <path-to-backup>
```

其他可选参数：

```
-retention <retention policy name>
-shard <shard ID>
-since <date>
```

注：日期为RFC3339 格式，例如：2015-12-24T08:12:23Z

示例：

```
$ influxd backup -database telegraf -retention autogen -since 20
2016/02/01 18:02:36 backing up rp=default since 2016-02-01 00:00
2016/02/01 18:02:36 backing up metastore to /tmp/backup/meta.01
2016/02/01 18:02:36 backing up db=telegraf rp=default shard=2 to
2016/02/01 18:02:36 backup complete
```

远程备份

InfluxDB可以使用 -host 参数实现数据的远程备份，端口一般是8088

示例：

```
$ influxd backup -database mydatabase -host 10.0.0.1:8088 /tmp/m
```

注，上文所有参数在远程备份中均可使用。

数据恢复

语法：

```
influxd restore [ -metadir | -datadir ] <path-to-meta-or-data-di
```

必要参数：

```
-metadir <path-to-meta-directory>
或
-datadir <path-to-data-directory>
```

可选参数：

```
-database <database>
-retention <retention policy>
-shard <shard id>
```

示例，恢复数据库：

```
$ influxd restore -database telegraf -datadir /var/lib/influxdb/
Restoring from backup /tmp/backup/telegraf.*
unpacking /var/lib/influxdb/data/telegraf/default/2/000000004-00
unpacking /var/lib/influxdb/data/telegraf/default/2/000000005-00
```