通过例子剖析 OpenTSDB 的 Rowkey 及列名设计

通过《OpenTSDB 底层 HBase 的 Rowkey 是如何设计的》文章我们已经了解 OpenTSDB 底层的 HBase Rowkey 是如何设计的了。我们现在来测试一下 OpenTSDB
导入的时序数据到底长什么样子。

在 OpenTSDB 里面默认存时序数据的表为
tsdb。前面说了，每个指标名称、标签名称以及标签值都有唯一的编码，这些编码数据是存放在 tsdb-uid 表里面。为了更加深入的了解 OpenTSDB 的 Rowkey 设计，下面我们通过往 OpenTSDB 里面插入一些测试数据，来加深理解。我们准备插入 OpenTSDB 的数据如下：

```
put sys.cpu.user 1541946115 42.5 host=iteblog cpu=0
put sys.cpu.user 1541946135 53.2 host=iteblog cpu=0
put sys.cpu.user 1542206107124 55 host=iteblog cpu=0
```

我们通过命令行往 OpenTSDB 插入上面的数据。执行完上面三个语句，我们来看下 tsdb 和 tsdb-uid 两张表的数据长什么样子：

```
hbase(main):020:0> scan 'tsdb'
ROW                                   COLUMN+CELL
 \x00\x00\x01[\xE85\xE0\x00\x00\x01\x00\x column=t:R;, timestamp=1542294109207, value=B*\x00\x00
 00\x01\x00\x00\x02\x00\x00\x02
 \x00\x00\x01[\xE85\xE0\x00\x00\x01\x00\x column=t:S{, timestamp=1542294118705, value=BT\xCC\xCD
 00\x01\x00\x00\x02\x00\x00\x02
 \x00\x00\x01[\xEC*`\x00\x00\x01\x00\x00\ column=t:\xF8\x09\xBD\x00, timestamp=1542294134705, value=7
 x01\x00\x00\x02\x00\x00\x02
2 row(s)
Took 0.0158 seconds
hbase(main):021:0> scan 'tsdb-uid'
ROW                                   COLUMN+CELL
 \x00                                 column=id:metrics, timestamp=0, value=\x00\x00\x00\x00\x00\x00\x00\x01
 \x00                                 column=id:tagk, timestamp=1542294106863, value=\x00\x00\x00\x00\x00\x00\x00\x02
 \x00                                 column=id:tagv, timestamp=1542294106881, value=\x00\x00\x00\x00\x00\x00\x00\x02
 \x00\x00\x01                         column=name:metrics, timestamp=1542294087695, value=sys.cpu.user
 \x00\x00\x01                         column=name:tagk, timestamp=1542294106836, value=host
 \x00\x00\x01                         column=name:tagv, timestamp=1542294106852, value=iteblog
 \x00\x00\x02                         column=name:tagk, timestamp=1542294106871, value=cpu
 \x00\x00\x02                         column=name:tagv, timestamp=1542294106886, value=0
 0                                    column=id:tagv, timestamp=1542294106889, value=\x00\x00\x02
 cpu                                  column=id:tagk, timestamp=1542294106874, value=\x00\x00\x02
 host                                 column=id:tagk, timestamp=1542294106839, value=\x00\x00\x01
 iteblog                              column=id:tagv, timestamp=1542294106856, value=\x00\x00\x01
 sys.cpu.user                         column=id:metrics, timestamp=1542294087699, value=\x00\x00\x01
8 row(s)
Took 0.0471 seconds
https://www.iteblog.com
```

如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

从上面的输出也可以看出，tsdb-uid 表就是存储的是标签名字、标签值以及指标名字的编码，以及编码和标签名字、标签值、指标名字的关系。我们可以看到指标名字 sys.cpu.user、标签名字 host 和标签值 iteblog 的编码均为 ₩x00₩x00₩x01；而标签名字 cpu 和其对于的标签值 0 编码均为 ₩x00₩x00₩x02。从这里可以说明以下几个问题：

- 同一类数据的编码只能唯一。比如指标名字的编码必须唯一，不存在一个指标编码对应多个指标名称；
- 不同类的数据编码可以相同。比如前面 sys.cpu.user、标签 host 和标签值 iteblog 的编码均为 ₩x00₩x00₩x01，因为它们之间是无影响的。

tsdb-uid 表中 Rowkey 为 ₩x00 的三行中的 id:metrics、id:tagk 以及 id:tagv 分别表示对应的指标名字、标签名字、标签值当前最大的编码。

tsdb 表里面有三行数据，这就是我们存储的上面插入的数据的。我们可以看到第一行和第二行的 Rowkey 是一样的，只是列名不一样，这两行对应的是上面 put 的前两行数据。现在我们来分解这个 Rowkey：
₩x00₩x00₩x01[₩xE85₩xE0
₩x00₩x00₩x01₩x00₩x00₩x01₩x00₩x00₩x02₩x00₩x00₩x02
我们主要来分析时间戳，见上面加粗的部分。为什么会显示 [呢？其实这里显示的是 ASCII 对应的字符，所以这四个字节（[₩xE85₩xE0）对应的十六进制是 5BE835E0，转换成十进制是 1541944800，这个正式我们前面分析的 2018-11-11 22:00:00 对应的时间戳。

接着我们来分析列名column=t:R; 这个其实也是 ASCII 对应的字符，十六进制显示为 523B，二进制表示为 101001000111011。我们前面说过，其实列名里面除了相对的秒数，其实还有当前列值得类型、值长度的信息。

在 介绍如何解析列名之前，我们先来了解一些前提知识。在 OpenTSDB 里面，如果 Rowkey 里面的时间戳只到秒级别的，那么列名称占用2个字节。
其中后面三位代表当前列值占用的字节数。

- 如果后三位为100，代表当前列值占了8字节；
- 如果后三位为011，代表当前列值占了4字节；
- 如果后三位为010，代表当前列值占了2字节；
- 如果后三位为000，代表当前列值占了1字节；
- 其他情况就是异常的。

倒数第四为代表当前列值的类型，

- 如果为0，则代表整型数据；
- 如果为1，则代表四字节浮点型数据。

前面12位才表示相对于 Rowkey 的秒数。

如果 Rowkey 里面的时间戳只到毫秒级别的，那么列名称占用4个字节。其中

- 最前四位固定为全1；

- 最后面四位的含义和前面一样；
- 倒数第5-6位为预留位；
- 其他的位才是真正的相对于 Rowkey 里面毫秒数。

所以列 column=t:R（二进制 101001000111011）可以按照上面的规则进行解析：

- 前12位二进制为 10100100011，十进制为 1315，这个就是我们前面分析的秒数！
- 倒数第四位为1，也印证了我们的数据类型为浮点型。
- 后三位为011，说明我们的数据值占用了4字节。

现在我们来解析 column=t:R 列对应的值。如上图所示，这列对应的值为 B*₩x00₩x00，这个其实也是 ASCII 对应的字符，对应的浮点型十六进制是 422A0000，由于其实一个浮点数，所以可以将这个十六进制转成十进制，数据为 42.5，这就是我们 put 进去的数据啊。

现在我们来解析上图中的第三行数据。我们可以看到，上面第三个 put 我们的时间戳精确到毫秒级别，所以我们需要按照上面的规则拆分 column=t:₩xF8₩x09₩xBD₩x00，我们可以将 ₩xF8₩x09₩xBD₩x00 使用二进制表示：11111000000010011011110100000000，我们对这个二进制解析如下：

- 这个二进制的前4位均为1；
- 倒数3位为 000，说明当前列对应的值占用了一个字节；
- 倒数第4位为0，说明当前列对应值的类型为整型数据；
- 除了前4位和后六位的二进制，剩下的就是相对于 Rowkey 的毫秒数，剩下的二进制为 10000000100111011110100，十进制为 2107124。这个值正是 1542206107124（2018-11-14 22:35:07）相对于 1542204000000（2018011014 22:00:00）多出来的毫秒数。

现在我们来解析 column=t:₩xF8₩x09₩xBD₩x00 列对应的值。从上图可以看出，这列的值为 7，其实这是 ASCII 字符，7对应的 ASCII 编码十六进制为 37，十进制是 55，这不就是前面第三个 put 插入的值嘛。

正是因为 OpenTSDB 的这种 Rowkey 设计，才使得 OpenTSDB 能够提供最高毫秒级精度的时间序列数据存储，能够长久保存原始数据并且不失精度。它拥有很强的数据写入能力，支持大并发的数据写入，并且拥有可无限水平扩展的存储容量。

本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接: 【】（）