Apache Cassandra 数据存储模型

我们在《Apache Cassandra 简介》文章中介绍了 Cassandra 的数据模型类似于 Google 的 Bigtable，对应的开源实现为 Apache HBase，而且我们在《HBase基本知识介绍及典型案例分析》文章中简单介绍了 Apache HBase 的数据模型。按照这个思路，Apache Cassandra 的数据模型应该和 Apache HBase 的数据模型很类似，那么这两者的数据存储模型是不是一样的呢？本文将为大家解答这些问题。我们从 KeySpace -> Table -> Partition -> Row -> Cell 顺序介绍。本文基于 Apache Cassandra 3.11.4 源码进行介绍的，不同版本可能有些不一样。

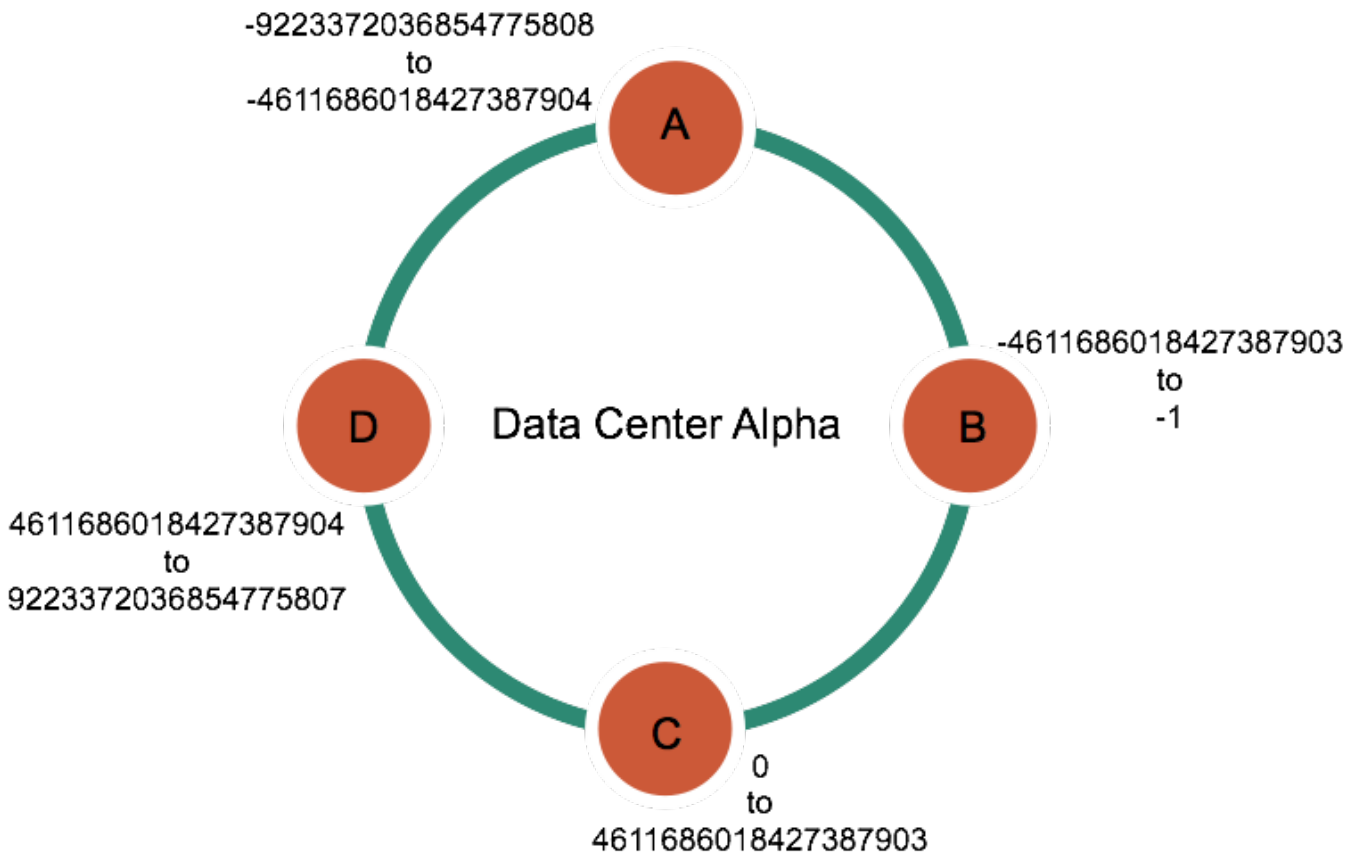
Table & KeySpace

Cassandra 中的 KeySpace 概念和 RDBMS 里面的 DataBase 概念很类似，一个 KeySpace 包含多张表，一般将有关联的数据表放到同一个 KeySpace 下面。KeySpace 创建的时候可以指定副本策略，副本因子以及是否启用 CommitLog 机制（类似 HBase 中的 WAL）。

Cassandra 中表的概念和 RDBMS 很类似。不同的是在 Cassandra 中属于同一张表的数据在物理上是分布在不同节点上存储的，同一张表由多个 Partition 组成。

Partitions

Cassandra 一般是由多台节点组成的，每台节点负责一定范围的，如果使用 Murmur3hash 的时候，每个节点负责的 Token 类似于下面那样：



所以 Token 范围为 -9223372036854775808 ~ -4611686018427387904 的数据存储在 A 节点；同理，Token 范围为 -4611686018427387903 ~ -1 之间的数据存储在 B节点，其他类似；每个 Token 范围由多个 Partition 构成，每个 Partition 由一行或多行数据组成，Partition 类似下面的：

username	type	email	age	tel
iteblog	1	wyphao.2007@163.com	100	13888888888
iteblog	3	xxx@163.com		
cassandra	1	cassandra@163.com		18888888888
spark	5	spark@apache.org	99	

其中，username 为 Partition key；type 为 Clustering key。那么在这种情况下，username = iteblog 的两条数据构成一个 Partition；另外两条构成分别构成两个 Partitions。在底层存储每个 Partition 格式如下：

PartitionHeader	N Bytes
Row	N Bytes
.....	
Row	N Bytes
EndPartition	1 Byte

从上图可以看出，一个 Partition 是由 PartitionHeader、零个或多个 Row（格式在后面介绍）以及 EndPartition 三部分组成的。EndPartition 这个用于标记 Partition 结束，只占用一个字节，并使用 0x00000001 标记。PartitionHeader 的格式如下：

Partition Key Length	2 Bytes
Partition Key	0 - N Bytes
Local Deletion Time	4 Bytes
Marked For Delete At	8 Bytes
Static Row	0 – N Bytes

- Partition Key 就是我们建表的时候指定的，由于 Partition Key 长度使用两字节表示，所以 Cassandra 中 Partition Key 长度必须小于等于 65535 字节。
- Local Delete Time 是删除发生时的服务器时间（以秒为单位），与 gc_grace_seconds 进行比较以确定何时可以清除它。当与 TTL 一起使用时，localDeletionTime 是数据到期的时间。共占四个字节；
- Marked For Delete At 记录删除的时间戳，时间戳小于此值的数据被视为已删除，共占用八字节。
- Static Row：如果我们建表的时候有 Static 字段，那么标记为 Static 的列会在这里存储。从这里也可以看出，partition key 相同的数据 Static 列只会保存一份数据。

在底层存储中，多个 Partition 组成一个 SSTable（Sorted-String Table）文件。那么同一个

SSTable 文件中的数据数据是如何组织的呢？答案是按照 Partition Key 计算得到的 Token 升序排序的。

如果 Partition Key 由多个字段构成那么是将这多个字段拼在一起再计算拼成后字符串的哈希值。假设我们的建表语句如下：

```
CREATE TABLE iteblog (
  username text,
  action text,
  type text,
  email text,
  age text,
  tel text,
  PRIMARY KEY((user_id, action), type)
);
```

那么 Cassandra 会将 username 和 action 字段拼在一起，每个字符串首先使用一定的算法计算字节数组，并且每个字符串计算完最后都会加上 0 bit 位；然后多个字符串拼接到一起，计算拼接后的哈希值。

Row

上面看出，Partition 里面包含了零个或多个 Row，这些 Row 对应的 Partition Key 是一样的。非 Static 的 Row 在磁盘存储的格式如下：

flags	1 Byte
extendedFlags	1 Byte
Clustering info	N Bytes
.....	
Clustering info	N Bytes
Row Body Size	1 – N Bytes
Previous Row Body Size	1 – N Bytes
Primary Key Liveness Timestamp	1 – N Bytes
Primary Key Liveness TTL	1 – N Bytes
Primary Key Liveness LocalExpirationTime	1 – N Bytes
Row Marked For Delete At	1 – N Bytes
Row Local Deletion Time	1 – N Bytes
Columns Bitmap	1 – N Bytes
Cell	1 – N Bytes
.....	
Cell	1 – N Bytes

上面除了 flags 、Row Body Size 、 Previous Row Body Size 以及 Primary Key Liveness Timestamp
这四个字段一定会存在，其他字段需要满足条件才会存储。下面对上面字段进行介绍：

- flags：Row 的标记信息，主要用于标记当前 Row 是否存在时间戳、TTL、被删除、是否包含所有的列等信息。flag 字段占用一个字节，
- hasExtendedFlags：当前 Row 是否含有 Static 列，存在才会有数据；
- Clustering info：每个 Row 包含零个或多个 Clustering 相关的信息。Clustering 信息就是我们创建表的时候指定的 Clustering key 信息。每个 Clustering Info 在持久化的时候会先存储头部信息，标记当前 Clustering key 是否为空、是否为 null 以及是否有值等信息；然后根据数据类型将值存下来，如果当前 Clustering key 的值占用字节非固定，还需要存储当前 Clustering key 值的字节数。
- Row Body Size：当前 Row Body 的大小，Row Body 包含 primary key 的 liveness 信息、Row 是否删除等信息以及 Cell 的信息。
- Previous Row Body Size：前一个 Row Body 的大小，这个主要用于加速反向查询的，不过当前并没有使用；
- Primary Key Liveness Timestamp：primary key 的 Liveness 用于确定行是否还活着或已经死了（没有 live cells 并且 liveness 为空）。这个字段主要用于存储当前 Row 的 Liveness 时间戳。注意，持久化到磁盘的时间戳是相对于当前 Memtable 最小时间戳的值。
- Primary Key Liveness TTL：这个字段主要用于存储当前 Row 的 Liveness TTL 信息。也是相对于当前 Memtable 最小 TTL 的值
- Primary Key Liveness LocalExpirationTime：当前 Liveness 的 ExpirationTime，也是相对时间；
- Row Marked For Delete At：当前 Row 的删除时间，也是相对时间，精确到毫秒；
- Row Local Deletion Time：当前被标记为 tombstone 时服务器的时间，也是相对时间，精确到秒；
- Columns Bitmap：从 Cassandra 3.x 开始，列的信息已经不保存到数据文件里面了，列的信息是保存在对应 SSTable 的 md-X-big-Statistics.db 文件中。这个字段是用于标记当前行哪些列存在，哪些列不存在。如果列存在则标记为0；如果列不存在则标记为1；如果列全部存在，直接标记为0。当表的字段数小于64个的时候，直接使用一个 long 类型的数据来存储这个 bitmap。如果大于等于64个，处理方案稍微复杂一些：
    - 先保存一个标记位，标记当前表拥有的字段个数大于等于64；
    - 如果存在的列没有占总列数的一半，则按照全部列的顺序保存存在的列在排序后列的索引位置；
    - 如果存在的列占总列数超过一半，则按照全部列的顺序保存不存在的列在排序后列的索引位置。

  可见，Cassandra 通过将列的信息（包括列的名称、类型、表名、keySpace等信息）保存到对应 SSTable 的 md-X-big-Statistics.db 文件中，相应的行只保存列是否存在的标记信息，这个可以节省存储空间的占用。注意，HBase 存储数据的时候每个 Cell 都需要保存列名称和列族名称的。

非 Static Row 的底层存储格式已经在前面描述过，对于 Static Row 除了没有上图的 Clustering info 信息，其余都一样，所以这里就不介绍了。

上图中最后有 N 个 Cell，那多个 Cell 之间的顺序是如何保证的呢？答案是

按照列的名称字典顺序升序排序的。比如我们表的定义如下：

```
CREATE TABLE iteblog (
  user_id text,
  type text,
  action text,
  username text,
  age text,
  email text,
  PRIMARY KEY(user_id)
);
```

那么 Cell 的顺序排列如下：

action -> age -> email -> type -> username

这个排序是通过 BTree 实现的，Row 的实现类为 BTreeRow。

Cell

Cell 就是每列数据的底层实现，Cell
里面包含了列的定义信息，比如是否被删除、是否过期、是否设置了时间戳等。在 Cassandra 里面，Column 有 Simple 和 Complex（CASSANDRA-8099引入的）之分。non-frozen collection 或 UDT（用户自定义类型）的列是 ComplexColumn（Complex Cell）。

Simple Cell（Simple Column）的底层格式

我们正常使用的列就是属于这种类型的，它的底层存储格式如下：

flags	1 Byte
Cell timestamp	1 – N Bytes
Cell Local Deletion Time	1 – N Bytes
Cell TTL	1 – N Bytes
Cell Value	1 – N Bytes

- flags：这个 Cell 的 flag 标记，主要用于标记当前 Cell 是否有值、是否被删除、是否过期、是否使用 Row 时间戳、是否使用 Row TTL 等信息。flag 字段占用一个字节，每位的含义代表如下：

保留位	保留位	保留位	TTL标记	时间戳标记	Cell无值	是否过期	是否删除

- timestamp：当前 Cell 的时间戳，Cassandra 中我们可以对每列设置时间戳；
- deletion time：当前 Cell 的删除时间；
- ttl：当前 Cell 的 TTL，Cassandra 中我们可以对每列设置 TTL，代表这个 Cell 保留多长时间；
- value：当前 Cell 的值；

注意：上面字段只有 flags 是一定会存在的，其他字段得看条件是否满足。在 Cassandra 中，Simple Cell 的实现类是 BufferCell。

Complex Cell（Complex Column）的底层格式

如果列属于 non-frozen collection 或 UDT（用户自定义类型），那么这个属于 Complex Cell，它的底层存储格式如下：

Complex Cell Marked For Delete At	1 Byte
Complex Cell Local Deletion Time	1 – N Bytes
Complex Cell Counts	1 – N Bytes
flags	1 Byte
Cell timestamp	1 – N Bytes
Cell Local Deletion Time	1 – N Bytes
Cell TTL	1 – N Bytes
Cell Path	1 – N Bytes
Cell Value	1 – N Bytes

可以看出，Complex Cell 和 Simple Cell 大部分很类似，下面只介绍不一样的地方：

- Complex Cell Marked For Delete At & Complex Cell Local Deletion Time：这两个属性和前面的类似，只不过针对 Complex Cell 而言的。
- Complex Cell Counts：Complex Cell 的个数；
- path：当前 Cell 的路径。

在 Cassandra 中， Complex Cell 的实现类是 ComplexColumnData。