**OceanBase内部行存储格式**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **编号** | **文档版本** | **修订章节** | **修订原因** | **修订日期** | **修订人** |
| **1** | 0.1 |  | 新建文档 | 2012-4-6 | 正祥，郁白，  剑鸣 |
| **2** | 0.2.1 |  | 3.1节中TypeAttr的attr变为3 bit，添加min/max，修改cache对不存在行的说明(3.3) | 2012-4-11 | 竹翁，正祥 |
| **3** | 0.2.2 |  | 3.1节增加precise\_time、create\_time、modify\_time类型；增加了类型char(保留)；增加了类型number的说明；增加3.2稀疏格式和3.3稠密格式的说明；增加了使用紧凑格式进行orderby的说明； | 2012-5-31 | 剑鸣  正祥 |
| **4** |  |  |  |  |  |
| **5** |  |  |  |  |  |
| **6** |  |  |  |  |  |

[1 行存储格式目标 3](#_Toc326226767)

[2 名词解释 3](#_Toc326226768)

[3 行存储格式定义 3](#_Toc326226769)

[3.1 稀疏格式 4](#_Toc326226770)

[3.2 稠密格式 5](#_Toc326226771)

[3.3 带主键的行 5](#_Toc326226772)

[3.4 行的KV cache 6](#_Toc326226773)

[3.5 多行及外部索引 6](#_Toc326226774)

本文档对淘宝OceanBase( <http://oceanbase.taobao.org> )开源数据库的内部行存储格式进行了定义和说明。

# 行存储格式目标

作为一个数据库系统，表格的行是OceanBase的基本数据结构，内部的行存储格式需要满足以下需求：

1. 多行数据(每行若干<column\_id+column\_value>或<column\_value>)高效顺序追加；
2. 多行数据高效顺序读出(顺序迭代)；
3. 多行数据可在外部按列建索引并可按索引随机读取；
4. 传输及存储时无需序列化和反序列化(非char类型的2/4/8字节数据以X86处理器的little-endian表示)；
5. 支持稀疏和稠密格式；
6. 比较紧凑(减少内存以及磁盘消耗)；

# 名词解释

**Update Server**：OceanBase的增量更新子系统，记录用户的修改，通常由一主一备或一主多备构成；

**ChunkServer**：OceanBase的基线数据存储子系统，由多台机器构成，基线数据通常保存2~3副本并且保存在不同的ChunkServer上；

**MergeServer**：OceanBase的用户接口子系统，接收用户的查询请求，并从对应ChunkServer上获得基线数据以及从UpdateServer上获得更新的增量，然后融合二者后返回用户；

**RootServer**：OceanBase的管理服务器，用于管理UpdateServer、ChunkServer和MergeServer等;

**列：**一个列由列ID(column\_id)及其值(column\_value)组成；

**行：**一个行由若干列组成，有些时候其中的部分列构成主键(row key)并且整个表按主键顺序存储，有些表，如select指令的结果，可以不包含主键；

**schema**：表的列的类型、值范围等以及该表与其他表的join等关系称为表的schema；

**表**：一个表由若干列(在schema中定义)和任意行组成，有些表的每一行都存储了schema中定义的每一列，这样的表是稠密的，如静态(基线)数据的表一般是稠密的；另外一些表行只存储了部分的列，例如UpdateServer中的表的修改增量，这样的表是稀疏的。

# 行存储格式定义

此处分别说明：

* 稀疏格式
* 稠密格式
* 带主键的行
* 行的KV cache
* 多行以及外部索引

## 稀疏格式

稀疏格式的数据一般由UpdateServer产生，通常情况下，一行由若干<column\_id，column\_value>组成，紧凑型格式内按***column\_value在前column\_id在后***的方式存储，column\_id在column\_value之后，是2字节无符号整数(**需要注意INVALID\_ID的转换**)。**非char类型的2/4/8字节数据都以X86机器字节顺序存储(下同)**。

Column\_value由一个字节的TypeAttr开始，后面紧跟值的内容。其中，TypeAttr的位定义是：

TypeAttr = Type(5)Attr(3)

Type(5)为高5位，表示数据类型或转义：

|  |  |  |  |
| --- | --- | --- | --- |
| 取值 | 类型 | 后续字节数 | 备注 |
| 0 | null | 0 | *今后可能取消独立的null类型，改为在各个类型中记录单独的null值* |
| 1 | int8 | 1或0 | 8位有符号整数 |
| 2 | int16 | 2或0 | 16位有符号整数 |
| 3 | int32 | 4或0 | 32位有符号整数 |
| 4 | int64 | 8或0 | 64位有符号整数 |
| 5 | char |  | 暂时保留 |
| 6 | varchar | 2+strlen或0 | 紧跟2字节长度strlen，然后是strlen个字节(varchar内容) |
| 7 | number | 2+(w+1)\*4 | 2字节(16位)=p:7位(最高位)+s:6位+w:3位(最低位)，其中p是number的精度(十进制的小数点前+小数点后的位数)，s是number的十进制小数点后的位数，p和s当前的最大值是38(未来如果需要p可扩展到76，s可扩展到63)。(p,s)缺省是(18,0)。(w+1)对应于后续的int32的个数(1~4，未来如果需要可扩展到5~8)，这些int32就是把number去掉小数点的结果(例如(5,2)格式的数3.5就表示成350)，低int32在前，高int32在后，每个int32内部的字节用机器字节顺序。 |
| 8 | precise\_time | 8或0 | 时间类型，64位有符号整数(单位：微秒) |
| 9 | time | 8或0 | *时间类型，64位有符号整数(单位：秒)，此数据类型后续将淘汰* |
| 10 | create\_time | 8或0 | *时间类型，64位有符号整数(单位：微秒)，对象的创建时间。* |
| 11 | modify\_time | 8或0 | *时间类型，64位有符号整数(单位：微秒) ，对象的修改时间。* |
| 其他 | 保留 |  |  |
| 0x1f | 转义符 | 0 | Attr(3)=0：本行结束  Attr(3)=1：删除本行  Attr(3)=2：NOP(例如cache时标识本行不存在)  Attr(3)=其他值：保留 |
| 其他值 |  |  | 保留 |

当Type(5)为某种值类型时，Attr(3)的含义是：

|  |  |  |
| --- | --- | --- |
| 取值 | 含义 | 备注 |
| 0 | 0 | 正常，即本列设为该值 |
| 1 | add | 表示本对象需要add到基线值上 |
| 2 | null | 表示本对象的值是null，后续字节数是0 |
| 3 | min | 表示本对象的值是min，后续字节数是0 |
| 4 | Max | 表示本对象的值是max，后续字节数是0 |
| 5-7 |  | 保留 |

## 稠密格式

稠密格式的行包括从sstable中获得的行、稠密格式行与UpdateServer的修改增量合并后的行、select(orderby/groupby/view等)的结果数据等等。除了不保存2字节的列ID外，其他与稀疏格式一样。稠密行一般需要借助于表的schema来定位列。由于调试或鲁棒性等其他原因，ChunkServer/MergeServer内部也可以用稀疏格式来记录内部数据。

**稠密格式和稀疏格式用同一个类实现，类对象实例化后通过一个布尔标识来区别两种格式。**

## 带主键的行

尽管SQL select的结果不需要带上主键，但在内部处理中主键常常是需要的，例如从UpdateServer获得的修改增量必须有主键，修改增量与基线数据合并时基线数据也必须携带主键。目前主键有两类表示方式，即拼接后的字符串和拆分后的多个列，前者可以表示成column\_id=1的一个列，后者是多个<column\_id,column\_value>对，因此都可以表示成上述的格式。带主键的行可以是稀疏格式(例如UpdateServer产生的数据)，也可以是稠密格式(例如由sstable获得的行)。

带主键的行可以表示成一个行，此时主键是其中的部分列；也可以表示成两行，即主键部分单独构成一行，其余数据也构成一行。考虑到后续的行的KV cache等原有，表示成两行可能更合适。

## 行的KV cache

行的KV cache能够缩短查询响应时间，目前OceanBase有以下几个地方用到或即将用到：

1. UpdateServer上的冻结/转储后的memtable的行cache；
2. ChunkServer上的静态数据的行cache；
3. ChunkServer上的join cache(属于UpdateServer冻结后的memtable的行的cache)；
4. ChunkServer上UpdateServer冻结/转储后的memtable的行cache；

这些行有些是稀疏格式，有些是稠密格式，但它们都是带主键的行。

在KV cache中，其中的Key可能包括以下内部：

* table\_id
* 版本号(静态数据版本号或者动态数据的主版本号+子版本号)
* 其他信息(例如是本地sstable的行还是UpdateServer冻结/转储后的memtable的行)
* 主键

Value是对应的行内容(可能是空，表示对应行不存在)。

在ChunkServer上，静态数据的行的cache与获得的UpdateServer冻结/转储后的memtable的行cache可以统一存放和管理。

## 多行及外部索引

以下两个场景需要多行数据，其他场景也可能用到：

* 冻结/转储后的memtable分发到ChunkServer，ChunkServer以tablet为单位保存的多行的增量数据；
* MergeServer/ChunkServer对select的结果进行orderby，此时还需要进行外部索引

多行数据也有稀疏和稠密两种格式，即要么所有行都是稀疏格式，要么所有行都是稠密格式；可以带主键(所有行都带主键)，也可以不带主键(所有行都不带主键)。多行数据以行为单位逐行存储。

如果需要对其中的m个列做orderby，那就需要一个m\*n(n为行数)的对象数组以及一个长度为n的指针数组(指向每一行)。排序完成后，根据排序的结果，通过上述行指针获得排序后的数据。下面以SQL中的orderby的实现为例子来说明这种外部索引的使用：

1. 建立一个固定长度(例如256K，预设值)的指针数组P；
2. 从上游获得一行ObRow(展开的行)，没有数据则转到最后一步，否则执行下一步；
3. 计算ObRow转换成紧凑型格式ObCompactRow后的长度compact\_row\_len；
4. 如果排序缓冲区中的行数超过了指针数组的长度，则跳转到步骤6；
5. 如果已申请的排序缓冲区的空闲内存<(r个ObObj+compact\_row\_len），其中r是orderby的列数，且以下条件至少有一个成立，则执行步骤6，否则执行步骤7：
   1. 排序缓冲区大小超出了单个orderby能够使用的最大缓冲区尺寸(预设值，例如256MB)；
   2. 排序缓冲区大小达到了单个orderby使用的最小缓冲区尺寸(例如16MB，预设值)并且整个ChunkServer/mergeServer使用的内存达到了上限(例如16GB，预设值)；
6. 利用指针数组P对已经收集的数据进行内排序，然后把排序后的数据dump到磁盘，并在本次dump数据中记录上次dump数据的文件位置；更新内存中记录的最后一次dump数据的文件位置；释放排序缓冲区内存；设置bIsExternalSort为true；
7. 向排序缓冲区申请（r个ObObj+compact\_row\_len）字节长度的缓冲区(由于排序缓冲区内部是分块的，例如2MB大小的块，需要确保（r个ObObj+compact\_row\_len）在一个内存块中)；排序缓冲区的数据先存储orderby的列(r个ObObj，定长)，再存储紧凑型数据ObCompaction(变长)，并且确保这些ObObj中的指针(例如varchar/char/number)指向排序缓冲区中对应的ObCompaction；转到步骤2；
8. 对排序缓冲区中的数据(如果有的话)进行内排序(与步骤6相同)。如果bIsExternalSort为false，流程结束；否则，如同步骤6一样把内排后的数据dump到磁盘，并根据磁盘中每个dump的数据块记录的上一个dump数据块位置来建立多路归并数据源，后续的行迭代可以通过多路归并逐步输出排序后的行，流程结束；

一些后续处理可能需要知道行数以及首行和尾行等，这部分信息可以在先前的追加过程中记住并实时更新。