OceanBase通信协议

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **编号** | **文档版本** | **修订章节** | **修订原因** | **修订日期** | **修订人** |
|  | 1.0 | 全文 | 新建 | 2010-10-22 | 若海 |
|  | 1.1 | 全文 | 根据review修改，修改序列化格式 | 2010-11-3 | 若海 |
|  | 2.0 | ObScanParam | 1. 新建简单属性过滤 2. 增加groupby协议 | 2011-04-18 | 无施 |
|  | 2.1 | ObScanParam | 1. 增加count(\*)协议描述 2. 增加select(\*)协议描述 |  |  |

本文档描述了客户端和服务器端完成请求的通信协议。

# 包结构

## 通用的包结构

|  |
| --- |
| Magic (4 Bytes) |
| Packet Sequence Id (4 Bytes) |
| Packet code (4 Bytes) |
| Total Body Length (4 Bytes) |
| API Version (4 Bytes) |
| Source Id (4 Bytes) |
| Target Id (4 bytes) |
| Timeout (4 Bytes) |
| ObRecordHeader (32 Bytes) |
| Body (As needed) |

通用包的头共64个字节，前16个字节为tbnet框架的定义，中间16个字节包括应用接口版本号以及调用目标，剩余的是ObRecordHeader（32字节），各字段的含义如下：

|  |  |  |
| --- | --- | --- |
| 字段名称 | 含义 | 值 |
| Magic | 通信包的Magic Number | 0x416e4574 |
| Packet Sequence Id | 通信包的序列号，每个通信包的序列号应该不同，服务器端返回时，会将该值复制到响应包的对应字段 |  |
| Packet Code | 包的标识代码 | 见通信包代码定义表 |
| Total Body Length | 包的长度（除去Magic, Packet Sequence Id, Packet Code和Total Body Length） |  |
| API Version | 该API接口的版本号 |  |
| Source Id | 发送方的身份，高2-byte为系统代码（OceanBase使用0x1），低2-byte为子系统 | 1. clieint 2. Rootserver 3. Mergeserver 4. Updateserver 5. chunkserver |
| Timeout | 发送方的超时时间 | 单位为 |
| ObHeader | 记录的头信息，包括校验码等 |  |

通信包代码定义

|  |  |
| --- | --- |
| 通信包类型 | 代码 |
| Get Request | 101 |
| Get Response | 102 |
| Scan Request | 122 |
| Scan Response | 123 |
| Get UpdateServer Info request | 211 |
| Get UpdateServer Info response | 212 |
| Write request | 301 |
| Write Response | 302 |
| Get latest frozen version Request | 303 |
| Get latest frozen version response | 304 |

## ObHeader的结构

|  |  |
| --- | --- |
| Magic (2 Bytes) | Header length (2 Bytes) |
| Version (2 Bytes) | Header Checksum (2 Bytes) |
| Data Original Length (4 Bytes) | |
| Data Compressed Length (4 Bytes) | |
| Data checksum (8 Bytes) | |
| Reserved (8 Bytes) | |

ObHeader共32个字节，各字段的含义如下：

|  |  |
| --- | --- |
| 字段名称 | 含义 |
| Magic | ObHeader(0XB0CC) |
| Header length | 此header的长度 |
| Version | 此header的版本号 |
| Header Checksum | Header元信息的校验码，奇偶校验 |
| Data Original Length | 数据原长度 |
| Data Compressed Length | 数据压缩后的长度<=数据的原长度，如果相等则表示没有压缩。压缩算法由应用另行定义。 |
| Data checksum | 数据的校验码，采用crc64校验算法 |
| Reserved | 保留位 |

# 基本数据类型和数据结构序列化描述

本节描述在通信协议中需要使用的基本数据类型和数据结构的序列化方式

## 基本数据类型

OceanBase基本类型主要包括数字和变长的字节流两种类型，针对数字，分为定常和变长两种序列化方式。

本文档的类型和说明定义如下：

|  |  |
| --- | --- |
| 类型 | 说明 |
| Int32 | 定长的4字节整型 |
| Int64 | 定长的8字节整型 |
| Vint32 | 变长的4字节整型 |
| Vint64 | 变长的8字节整型 |
| varchar | 字节流类型 |
| byte | 字节 |

### 数字类型定长序列化

数字类型的定长序列化是按网络序（高字节在前低字节在后）进行。比如一个32位的整数256，序列化后为：

|  |  |  |  |
| --- | --- | --- | --- |
| 00000000 | 00000000 | 00000001 | 00000000 |

### 数字类型的变长序列化

OceanBase的协议中支持对数字类型的变长传输，基本的格式为：

|  |  |  |
| --- | --- | --- |
| 标志位 | 值 | ………… |

每个字节的第一位为标识位，标识位用于表示该字节是否是数字的末字节，其它的7位表示真实的数据，标识为的定义为：

|  |  |
| --- | --- |
| 标识位 | 含义 |
| 1 | 该字节不是数字的末尾字节 |
| 0 | 该字节是数字的末尾字节 |

由于每个字节的第一位用于表示标识，用于表示真实数据只剩下7位，如果一个4字节的整型，其值超过28 bit能表示的范围，则序列化后需要5个字节。

小于4个字节的整型不支持变长序列化，布尔型用1表示true，0表示false

序列化例子1：

4字节的整型，值为5，二进制的完整表示为：

|  |  |  |  |
| --- | --- | --- | --- |
| 00000000 | 00000000 | 00000000 | 00000101 |

序列化后为

|  |  |
| --- | --- |
| 0 | 0000101 |

0表示该字节为数值的最后一个字节，其值为 0000101，序列化后可得知值为5

序列化例子1：

4字节的整形，值为536870912，其二进制完整表示为：

|  |  |  |  |
| --- | --- | --- | --- |
| 00100000 | 00000000 | 00000000 | 00000000 |

序列化后为

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 0000000 | 1 0000000 | 1 0000000 | 1 0000000 | 0 0000010 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000010 | 0000000 | 0000000 | 0000000 | 0000000 |

反序列化后得出结果为100000000000000000000000000000，即536870912

### 浮点型的表示

浮点型在序列化时按网络序（高字节在前低字节在后）进行，例如：

1. Float序列化为4字节
2. Double序列化为8字节

### 变长字节流的表示

变长字节流(varchar)的序列化格式为：

|  |  |
| --- | --- |
| 长度(vint64) | 字节流 |

长度为8字节的整形，按照变长的形式序列化，字节流为待序列化的字节流的拷贝。

比如testing，序列化后为

07 74 65 73 74 69 6e 67

其中07为长度（数字类型的变长序列化），后面为testing的utf-8形式

## 数据结构序列化描述

OceanBase通信中常用的数据结构有：

1. ObResultCode
2. ObServer
3. ObObj
4. ObGetParam
5. ObScanParam
6. ObMutator
7. ObScanner

### ObResultCode

ObResultCode用于表示一个操作的结果状态，由两部分组成：

|  |  |
| --- | --- |
| Code (vint32) | Message(varchar) |

message的长度不超过256字节

### ObServer

ObServer用于表示一个服务器的地址信息，结构为：

|  |  |  |
| --- | --- | --- |
| Version (vint32) | Port (vint32) | address |

version表示地址的协议版本，当前的值为：

|  |  |
| --- | --- |
| 4 | IPV4 |
| 6 | IPV6 |

如果为IPV4，则address格式为：

|  |
| --- |
| Address (vint32) |

如果是IPV6，则address格式为：

|  |  |  |  |
| --- | --- | --- | --- |
| Vint32 | Vint32 | Vint32 | Vint32 |

### ObObj

ObObj为Oceanbase中的通用类型，其可以表示多种数据类型，定义和序列化形式请参见《ObObj序列化格式与扩展字段语义》

### ObGetParam

ObGetParam用于指定查询相关列的数据，包括表名、rowkey和列名等，由如下的ObObj对象组成：

1. 基本参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj) +
2. 表行列参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj对象) +
3. 数据结束标识(一个ObObj扩展对象)

各部分具体序列化后的组成结构如下图所示：

*注：下划线对象为类型标识ObObj扩展，空白的格子表示没有ObObj对象，其余每个格子代表一个ObObj的对象，序列化结果是非空白区域的ObObj连续存储,以下各个图的注释与此相同。*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 基本参数标识 | 是否cache | version Border flag | begin version | | end version |
| 表行列参数标识 | 表名指令标识 | 表名或者ID | | | |
|  |  | Rowkey指令标识 | | Rowkey | |
|  | | 列名或者ID | |
| 列名或者ID | |
| … | |
| Rowkey指令标识 | | Rowkey | |
|  | | 列名或者ID | |
| 列名或者ID | |
| … | |
| … | | … | |
| 表名指令标识 | 表名或者ID | | | |
|  | Rowkey指令标识 | | Rowkey | |
|  | | 列名或者ID | |
| … | |
| 数据结束标识 | | | | | |

所有ID和名字相关的数据，客户端用名字(忽略ID)，内部调用用ID(忽略名字)，序列化结构不区分是ObObj(int)还是ObObj(varchar)，序列化时根据如果varchar类型长度为0，则序列化ObObj(int)类型，反序列化时根据ObObj的类型，设置不同的数据，由应用方根据协议约定获取ID或者Name。

**基本参数：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 基本参数标识 | 是否cache | version Border flag | begin version | end version |

1. 基本参数标识( ObObj(ext) ): 扩展obj标识类型 **BASIC\_PARAM\_FIELD**
2. 是否cache ( ObObj(int) ): 是否cache：0为否，1为是；
3. version borderflag ( ObObj(int) ) : 与ObRange中的BorderFlag类似；

version BorderFlag用于辅助表达后续的begin version和end version的含义，即0000EBCX，其中位E表示end是否最大，位B表示begin是否最小，位C表示end是否包含边界，位X表示begin是否包含边界；

1. begin version ( ObObj(int) ) :获取数据的起始版本；
2. end version (ObObj(int) ) :获取数据的截止版本；

begin和end两个version为update server frozen的version，是否包含边界由borderflag定义。客户端通常不需要关心这两个version，分别设置它们为最小和最大，merge server则需要根据chunk server返回的数据版本A，设置新的ObGetParam的begin version为A；如果是在做dump操作，则客户端需要从update server获取最近的forzen版本B，将结果设置到end version，merge server则需要根据chunk server返回的数据version A，设置新的get param的begin version为A, end version设置为B。

**表行列参数：**

|  |  |  |  |
| --- | --- | --- | --- |
| 表行列参数标识 | 表名指令 | 表名或者ID | |
|  |  | Rowkey指令 | Rowkey |
|  | 列名或者ID |
| 列名或者ID |
| … |
| Rowkey指令 | Rowkey |
|  | 列名或者ID |
| …. |

1. 表行列参数标识(ObObj(ext)): 扩展obj标识类型 **TABLE\_PARAM\_FIELD**
2. 表名指令(ObObj(ext)) + 表名(ObObj(varchar)) 或者表ID ObObj (int) **TABLE\_NAME\_FIELD**
3. rowkey指令(ObObj(ext)) + rowkey(ObObj(varchar)) **ROW\_KEY\_FIELD**
4. 列名(ObObj(varchar)) 或列ID(ObObj(int)) :读取的各个列名

由于通常请求时是查询一个表同一个rowkey的多列，所以GetParam中的ObCell大多具有相同的表名、rowkey，为了减小序列化的大小，相邻的Cell的表名相同，则除了第一个Cell序列化时写入表名，后续的ObCell都不写入表名，即减少两个ObObj(表名指令+ 表名或者表ID)，rowkey亦按照此规则进行序列化；反序列化时，如果迭代获取ObObj过程中没有遇到新的表名指令和表名或者ID，则采用上一个cell的表名或者ID，rowkey类似。

### ObScanParam

ObScanParam用于表示查询的数据所在的范围，范围由startkey和endkey两个rowkey指定，由如下的ObObj对象组成：

1. 基本参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj) +
2. 列参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj对象) +
3. 属性过滤参数标示（一个ObObj扩展对象）＋若干参数（若干ObObj对象）＋
4. groupby参数标示（一个ObObj扩展对象）＋若干参数（若干ObObj对象）
5. 排序参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj对象) +
6. 分页参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj对象)+
7. 过滤参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj对象) +
8. 数据结束标识(一个ObObj扩展对象)

各部分具体序列化后的组成结构如下表所示：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 基本参数标识 | 是否缓存 | Scan version Border flag | begin version | | end version | | 表名或表ID | Scan Range Border Flag | | Begin rowkey | | end rowkey | | 扫描顺序 | 大小限制 |
| 列参数标识 | 列名或者ID | | | | | | | | | | | | | | |
|  | 列名或者ID | | | | | | | | | | | | | | |
| … | | | | | | | | | | | | | | |
| 简单属性过滤参数标示 | left\_operand  列名或者ID | | | | | LogicOpType | | | | | right\_oprand  const Obj | | | | |
|  | left\_operand  列名或者ID | | | | | LogicOpType | | | | | right\_oprand  const Obj | | | | |
| … | | | | | | | | | | | | | | |
| GroupBy参数标示 |  | | | | | | | | | | | | | | |
|  | GroupBy列参数标示 | | | groupby column1: obj(column\_id/column\_name) | | | | | | | | | | | |
|  | | | groupby column2: obj(column\_id/column\_name) | | | | | | | | | | | |
| … | | | | | | | | | | | |
| Group返回列标示 | | | groupby return column1: obj(column\_id/column\_name) | | | | | | | | | | | |
|  | | | groupby return column2: obj(column\_id/column\_name) | | | | | | | | | | | |
|  | | | … | | | | | | | | | | | |
| aggregate function参数标示 | | | aggregate function type | | | | | as column name | | | | Aggregate function 作用的列（column\_id / column\_name) | | |
|  | | | aggregate function type | | | | | as column name | | | | Aggregate function 作用的列（column\_id / column\_name) | | |
| … | | | | | | | | | | | |
| 排序参数标识 | 列名或者ID | | | | | | 排序顺序 | | | | | | | | |
|  | 列名或者ID | | | | | | 排序顺序 | | | | | | | | |
| … | | | | | | … | | | | | | | | |
| 分页参数标识 | 起始偏移 | | | | | | 返回结果条数 | | | | | | | | |
| 数据结束标识 | | | | | | | | | | | | | | | |

所有ID和名字相关的数据，客户端用名字(忽略ID)，内部调用用ID(忽略名字)，序列化结构不区分是ObObj(int)还是ObObj(varchar)，反序列化时根据ObObj的类型，设置不同的数据，由应用方根据协议约定获取ID或者Name。

**基本参数：**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 基本参数标识 | 是否cache | Scan version Border flag | begin version | end version | 表名或表ID | Scan Range Border Flag | Begin rowkey | end rowkey | 扫描顺序 | 大小限制 |

1. 基本参数标识( ObObj(ext) ): 扩展obj标识类型 **BASIC\_PARAM\_FIELD**
2. 是否cache ( ObObj(int) ): 是否cache：0为否，1为是；
3. Scan version BorderFlag(ObObj(int))：

即后续的begin和end的含义，即0000EBCX，其中位E表示end是否最大，位B表示begin是否最小，位C表示end是否包含边界，位X表示begin是否包含边界，与ObRange中的BorderFlag类似

1. data begin version(ObObj(int)): start version
2. data end version(ObObj(int)): end version
3. 表名(ObObj(varchar))或者表ID(ObObj(int)) : 客户端用名字(忽略ID)，内部调用用ID(忽略名字)
4. Scan range borderflag ( ObObj(int) ) : range 范围标识
5. range BorderFlag用于辅助表达扫描rowkey的范围，含义同 version borderflag。begin rowkey(ObObj(varchar)): start row key
6. end rowkey( ObObj(varchar) ) : end row key
7. 扫描顺序(ObObj(int)) : 0升序，1降序(目前只支持升序)
8. 扫描大小限制(ObObj(int)) :

**列参数：**

|  |  |  |  |
| --- | --- | --- | --- |
| 列参数标识 | 列名或者ID | 列名或者ID | … |

1. 列参数标识( ObObj(ext) ): 扩展obj标识类型 **COLUMN\_PARAM\_FIELD**
2. 列名(ObObj(varchar)) 或者列ID(ObObj(int)) :读取的各个列名或
   1. 如果列参数中列数组为空，表示select(\*)，即select table中的所有列；如果同一个列在多个column group中出现，结果中只返回一次
   2. 如果列参数数组中仅有一个列，该列名为\*，表示select table中任意列，在count(\*)中使用

**简单属性过滤参数：**

|  |  |  |  |
| --- | --- | --- | --- |
| 简单属性过滤参数标示 | left\_operand  列名或者ID | LogicOpType | right\_oprand  const Obj |
|  | left\_operand  列名或者ID | LogicOpType | right\_oprand  const Obj |

1. **简单属性过滤参数标示(ObObj(ext))：扩展obj标示类型FILTER\_PARAM\_FIELD**
2. **列名(ObObj(varchar))者列id(ObObj(int))：属性过滤需要过滤的列**
3. **LogicOpType(ObObj(int)):比较类型**
4. 者列ID
5. **比较的常量(ObObj(取决于列本身的类型))**

**groupby参数：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GroupBy参数标示 |  | | | |
|  | GroupBy列参数标示 | groupby column1: obj(column\_id/column\_name) | | |
|  | groupby column2: obj(column\_id/column\_name) | | |
| … | | |
| Group返回列标示 | groupby return column1: obj(column\_id/column\_name) | | |
|  | groupby return column2: obj(column\_id/column\_name) | | |
|  | … | | |
| aggregate function参数标示 | aggregate function type | as column name | Aggregate function 作用的列（column\_id / column\_name) |
|  | aggregate function type | as column name | Aggregate function 作用的列（column\_id / column\_name) |
| … | | |

1. **groupby参数标示(ObObj(ext)): 扩展obj标示类型GROUPBY\_PARAM\_FIELD**
2. **groupby列参数标示(ObObj(ext)): 扩展obj标示类型 GROUPBY\_GRO\_COLUMN\_FIELD**
3. 列名(ObObj(varchar)) 或者列ID(ObObj(int)) :按照这些列进行group划分
4. **groupby 返回列参数标示(ObObj(ext)): 扩展obj标示类型GROUPBY\_RET\_COLUMN\_FIELD**
5. 列名(ObObj(varchar)) 或者列ID(ObObj(int)) :在groupby之后的结果中需要返回这些列
6. **groupby aggregate列参数标示(ObObj(ext)): 扩展obj标示类型GROUPBY\_AGG\_COLUMN\_FIELD**
7. **aggregate function type (ObObj(int)): aggregate函数类型**
8. **列名(ObObj(varchar)): 结果的列名字 select sum(a) as b**
9. 列名(ObObj(varchar)) 或者列ID(ObObj(int)) :aggregate function 作用的列
   1. 如果作用的列名为\*，表示作用于整行，只有在aggregate function type为count的时候有效

**排序参数：**

|  |  |  |
| --- | --- | --- |
| 排序参数标识 | 列名或者ID | 排序顺序 |
| 列名或者ID | 排序顺序 |

1. 排序参数标识( ObObj(ext) ): 扩展obj标识类型 **SORT\_PARMA\_FIELD**
2. 列名(ObObj(varchar)) 或者列ID(ObObj(int)) + 排序顺序(ObObj(int)): 排序的各个列以及规则，即升序还是降序，其中0表示升序，1表示降序；

**分页参数：**

|  |  |  |
| --- | --- | --- |
| 分页参数标识 | 起始偏移 | 返回结果条数 |

1. 分页参数标识( ObObj(ext) ): 扩展obj标识类型 **LIMIT\_PARAM\_FIELD**
2. 起始偏移(ObObj(int)) + 数量(ObObj(int)): 用于限制返回结果基于的偏移和本次返回的结果条数；

**过滤参数 (目前暂未实现)：**

1. 过滤参数标识( ObObj(ext) ): 扩展obj标识类型 **FILTER\_PARAM\_FIELD**
2. 列名(varchar)或列ID(int) + ObObj(int)(BorderFlag) + ObObj(begin) + ObObj(end) +

ObObj(int)(0为OR，1为AND) + ……

其中begin，end与该列数据类型一致或者为NullType

### ObMutator

ObMutator结构封装了对数据的修改信息，由如下的ObObj对象组成：

1. 操作指令参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj对象)
2. 数据结束标识(一个ObObj扩展对象)

各部分具体序列化后的组成结构如下表所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 操作参数指令标识 | 表名指令标识 | 表名或者表ID | | | |
|  |  | rowkey指令标识 | Rowkey | | |
|  | 列名或列ID | 操作符 | 操作值 |
| 列名或列ID | 操作符 | 操作值 |
| … | … | … |
| rowkey指令标识 | Rowkey | | |
|  | 列名或列ID | 操作符 | 操作值 |
| 列名或列ID | 操作符 | 操作值 |
| 表名指令标识 | 表名或者表ID | | | |
|  | rowkey指令标识 | Rowkey | | |
|  | Delete Row扩展标识 | | |
| 列名或列ID | 操作符 | 操作值 |
| OB/DB语意标识 | DB 或者OB | | | |
| 数据结束标识 | | | | | |

在操作参数指令中的几类扩展指令可以各种顺序出现，同样客户端用名字(忽略ID)，内部调用用ID(忽略名字)。

**操作参数：**

1. 操作参数指令标识(ObObj(ext)) **MUTATOR\_PARAM\_FIELD**
2. 表名指令标识(ObObj(ext)) + 表名(ObObj(varchar))或者表ID(ObObj(int)) **TABLE\_NAME\_FILED**
3. Row key 指令标识(ObObj(ext)) + rowkey (ObObj(varchar)) **ROW\_KEY\_FIELD**
4. 列名(ObObj(varchar))或列ID(ObObj(int)) + 操作符(ObObj(int)) + 操作值(ObObj(value)…) 操作符目前包括Update/insert/delete/add),其中delete列操作后面\无需value, 或者Delete Row扩展标识(ObObj(ext)) **DELETE\_ROW\_FIELD**
5. DB/OB语义扩展标识(ObObj(ext))+ DB或者OB (ObObj(int))，0表示OB, 1表示DB, 默认是OB语义，**DB\_SEMANTIC\_FIELD**
6. 由于通常更新请求是更新一个表同一个rowkey的多列，所以Mutator中的ObCell大多具有相同的表名、rowkey，为了减小序列化的大小，相邻的Cell的表名相同，则除了第一个Cell序列化时写入表名，后续的ObCell都不写入表名，即减少两个ObObj(表名指令+ 表名或者表ID)，rowkey亦按照此规则进行序列化；反序列化时，如果迭代获取ObObj过程中没有遇到新的表名指令和表名或者ID，则采用上一个cell的表名或者ID，rowkey类似。

### ObScanner

ObScanner封装了查询请求，包括Get和Scan接口返回的数据，由如下的ObObj对象组成：

1. 基本参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj) +
2. 表行列参数标识(一个ObObj扩展对象) + 若干参数(若干ObObj对象) +
3. 数据结束标识(一个ObObj扩展对象)

各部分具体序列化后的组成结构如下表所示：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 基本参数标识 | fullfill结束 | 处理到GetParam位置或者扫描row cout | ObScanner 数据的最大version | ObRange指令标识 | Range BorderFlag | start row | end row |
| 表行列参数标识 | 表名指令标识 | 表名或者表ID | | | | | |
|  |  | rowkey指令标识 | | | Rowkey | | |
|  | | | 列名或者列ID | 列值 | |
| … | … | |
| rowkey指令标识 | | | Rowkey | | |
|  | | | 列名或者列ID | 列值 | |
| 列名或者列ID | 列值 | |
| … | … | |
| 表名指令标识 | 表名或者表ID | | | | | |
|  | rowkey指令标识 | | | Rowkey | | |
|  | | | row not exist指令标识 | | |
| 数据结束标识 | | | | | | | |

在表行列参数指令中的几类扩展指令可以各种顺序出现，同样客户端用名字(忽略ID)，内部调用用ID(忽略名字)。

**基本参数：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 基本参数标识 | fullfill结束 | 处理到GetParam位置或者扫描row cout | 数据的最大version | ObRange指令标识 | Range BorderFlag | Start rowkey | End rowkey |

1. 基本参数标识(ObObj(ext)) **BASIC\_PARAM\_FIELD**
2. 是否fullfill结束(ObObj(int)): 0是没有结束，1是结束

对于get()，在ChunkServer上如果表改变或者tablet改变了，则get()停止并返回结束标记以及处理到的ObGetParam所在位置(一个整数)；在ChunkServer/UpdateServer上，如果结果超出缓冲区则返回未结束标识并以及处理到的ObGetParam所在位置(一个整数)；

对于scan()，在ChunkServer上如果当前tablet没有更多数据则scan()停止并返回结束标识和扫描的行数以及tablet的range；在ChunkServer/UpdateServer上，如果结果超出缓冲区则返回未结束标识；

1. 完成的param位置或者扫描的行数(ObObj(int))

get()返回一个整数指示处理到的GetParam位置，scan()时返回本次扫描的行数；

1. 数据的最大version (ObObj(int))

Cs返回当前扫描tablet的version, ups返回数据的当前tablet version

1. ObRange指令标识 (ObObj(ext)) + Range BorderFlag (ObObj(int)) + start row(ObObj(varchar)) + end row(ObObj(varchar)) **TABLET\_RANGE\_FIELD**

get()不用返回此标识，scan需要返回当前处理的**tablet的范围**；

**表行列参数：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 表行列参数标识 | 表名指令标识 | 表名或者表ID | | |
|  |  | rowkey指令标识 | Rowkey | |
|  | 列名或者列ID | 列value |
| … | … |
| 表名指令标识 | 表名或者表ID | | |
|  | rowkey指令标识 | Rowkey | |
|  | row not exist指令标识 | |

1. 操作参数指令标识(ObObj(ext)) **MUTATOR\_PARAM\_FIELD**
2. 表名指令标识(ObObj(ext)) + 表名(ObObj(varchar))或者表ID(ObObj(int)) **TABLE\_NAME\_FILED**
3. Row key 指令标识(ObObj(ext)) + rowkey (ObObj(varchar)) **ROW\_KEY\_FIELD**
4. 列名(ObObj(varchar))或列ID(ObObj(int)) + 列值(ObObj(value)…) 或者Row not exist扩展标识(ObObj(ext)) **ROW\_NOT\_EXIST**

同ObMutator以及ObGetParam类似，为了减小序列化的大小，相邻的Cell的表名相同，则除了第一个Cell序列化时写入表名，后续的ObCell都不写入表名，即减少两个ObObj(表名指令+ 表名或者表ID)，rowkey亦按照此规则进行序列化；反序列化时，如果迭代获取ObObj过程中没有遇到新的表名指令和表名或者ID，则采用上一个cell的表名或者ID，rowkey类似。

# 请求类型描述

## 获取UpdateServer信息的包

客户端在执行更新操作前，需要从rootserver那里获取UpdateServer的地址信息，该接口的格式描述如下：

### Request

获取UpdateServer的请求包不需要额外的信息

### Response

Response的Body格式如下：

|  |  |
| --- | --- |
| ObResultCode | ObServer |

ObResultCode的值：

|  |  |
| --- | --- |
| OB\_SUCCESS | 成功 |
| OB\_ERROR\_FUNC\_VERSION | API版本号不匹配，当前版本1 |

## rt\_get

客户端在发起查询请求前，需要根据待查询的数据或范围先向rootserver查询，以获得负责待查询的数据的MergeServer地址列表，然后根据MergeServer列表发起查询请求。

### Request

Request的Body只包含一个ObGetParam

|  |
| --- |
| ObGetParam |

ObGetParam应该包含一个ObCell，该ObCell中需要设置tablename和rowkey属性。

### Response

Response的Body格式如下：

|  |  |
| --- | --- |
| ObResultCode | ObScanner |

ObResultCode的值：

|  |  |
| --- | --- |
| OB\_SUCCESS | 成功 |
| OB\_ERROR\_FUNC\_VERSION | API版本号不匹配，当前版本1 |
| OB\_INVALID\_ARGUMENT | 非法参数，比如请求的表名不存在 |
| OB\_NOT\_INIT | Rootserver未初始化成功 |
| OB\_ERROR\_OUT\_OF\_RANGE | 超过Rootserver包含的范围 |
| OB\_NOT\_SUPPORTED | 不支持的IP协议，当前仅支持IPV4 |
| OB\_ERROR | 未知错误 |

Rootserver返回客户端的信息格式也使用表的格式，所以请求包含在一个ObScanner中，Scanner中每行数据客户端需要关心的字段及含义为：

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 含义 |
| x\_ms\_port | int32 | Mergeserver的端口 |
| x\_ipv6\_1 | int32 | Mergeserver的IPV6地址（当前未支持） |
| x\_ipv6\_2 | int32 |
| x\_ipv6\_3 | int32 |
| x\_ipv6\_4 | int32 |
| x\_ipv4 | int32 | Mergeserver的IPV4地址 |
| x\_tablet\_version | int64 | Tablet的版本号 |
| occupy\_size | int32 | Tablet的大小 |
| record\_count | int32 | Tablet记录的条数 |

说明1： 在Oceanbase中，数据有多个备份（当前为3），所以与数据对应的会有多个Tablet，Rootserver会返回所有备份的Tablet信息，字段中的X表示第几个备份（当前包括1, 2, 3）。

说明2：rootserver不保证每个tablet的3个备份信息都返回

说明3：rootserver返回的数据行数可能超过请求的行数，比如请求的数据对应一个tablet，rootserver可能会返回和该tablet邻近的tablet信息，但最多不超过8行

## rt\_scan

Rootserver也支持查询一个范围相关的Tablet信息。

### Request

Request的body只包含一个ObScanParam

|  |
| --- |
| ObScanParam |

在ObScanParam中，需要指定Table名和查询的范围，查询的范围需要指定startkey和endkey。

### Response

RootServer的Scan接口返回的格式和Get接口一样。

## ms\_get

客户端通过Mergeserver的ms\_get接口获取指定rowkey的数据，一个get请求可以获取多个rowkey的数据。

### Request

Request的Body只包含一个ObGetParam

|  |
| --- |
| ObGetParam |

ObGetParam应该包含ObCell列表，每个ObCell可以指定独立的表名、rowkey和列名。

### Response

Response的Body格式如下：

|  |  |
| --- | --- |
| ObResultCode | ObScanner |

ObResultCode的值：

|  |  |
| --- | --- |
| OB\_SUCCESS | 成功 |
| OB\_ERROR\_FUNC\_VERSION | API版本号不匹配，当前版本1 |
| OB\_ERR\_UNEXPECTED | 获取Schema未知错误 |
| OB\_SCHEMA\_ERROR | 请求的数据和schema不符 |
| OB\_ERROR | 未知错误 |

## ms\_scan

Mergeserver支持客户端指定一个范围（由startkey和endkey指定），返回该范围对应的数据。

### Request

Request的body只包含一个ObScanParam

|  |
| --- |
| ObScanParam |

在ObScanParam中，需要指定Table名和查询的范围，查询的范围需要指定startkey和endkey。

### Response

RootServer的Scan接口返回的格式和Get接口一样。

## get\_latest\_forzen\_version

updateserver会将内存中的数据按照一定的策略冻结为SSTable文件，如果客户端需要获取指定时间之前的数据（比如Dump），则需要关心updateserver最近一次冻结的版本号。

### request

request的body格式如下：

|  |
| --- |
| Dump\_time (int64\_t) |

该参数用于建议updateserver在此时间范围内，不将冻结的新SSTable文件发送给Chunkserver，以免Chunkserver上的数据的版本高于客户端请求的版本，导致dump失败。

说明：该时间仅是建议性的，updateserver不保证严格按照此时间中断发送新冻结的SSTable文件到chunkserver。

### response

response的body格式如下：

|  |
| --- |
| Version (int64\_t) |

Version即为updateserver截至请求时最新一个冻结内存表的版本号。

## apply

客户端对数据的修改操作，包括新增，更新和删除都通过apply接口完成。

### request

apply的request body格式如下

|  |
| --- |
| ObMutator |

### response

apply的response 包body仅包含一个返回值，即：

|  |
| --- |
| ObResultCode |

ObResultCode的值：

|  |  |
| --- | --- |
| OB\_SUCCESS | 成功 |
| OB\_ERROR\_FUNC\_VERSION | API版本号不匹配，当前版本1 |
| OB\_ERROR | 未知错误 |
| OB\_NOT\_SUPPORTED | 语义不支持，当前仅支持OB语义 |