**OceanBase接口文档**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **编号** | **文档版本** | **修订章节** | **修订原因** | **修订日期** | **修订人** |
| **1** | 0.1 | new | 新建文档 | 2010-07-28 | 若海 |
| **2** | 0.2 | 全文 | 设计更新 | 2010-08-06 | 若海 |
| **3** | 0.3 | 全文 | 接口更新 | 2010-10-21 | 若海 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

[公用对象 3](#_Toc275437324)

[查询操作的基本对象 4](#_Toc275437325)

[更新操作的基本对象 8](#_Toc275437326)

[查询接口 9](#_Toc275437327)

[更新接口 9](#_Toc275437328)

[Rootserver提供的内部服务 10](#_Toc275437329)

[基本对象 10](#_Toc275437330)

[给client的接口 11](#_Toc275437331)

[汇报与注册接口 11](#_Toc275437332)

[Schema服务 12](#_Toc275437333)

[UpdateServer提供的内部服务 12](#_Toc275437334)

[Chunkserver提供的内部服务 13](#_Toc275437335)

本文档定义了OceanBase各个模块（包括rootserver，updateserver，mergeserver和chunkserver）对外提供的接口。

# 公用对象

**struct** ObResultCode

**{**

**int32\_t** result\_code\_; // 结果代码

charmessage\_[OB\_ERROR\_MESSAGE\_MAX\_LENGTH]; // 描述信息

};

**enum** ObObjType {

*ObNullType* = 0,

*ObIntType*,

*ObFloatType*,

*ObDoubleType*,

*ObDateTimeType*,

*ObPreciseDateTimeType*,

*ObVarcharType*,

*ObSeqType*,

};

**class** ObObj {

**struct** ObObjMeta {

int32\_t type\_ :8;

int32\_t extend\_field\_ :8;

int32\_t op\_flag\_ :8;

int32\_t reserved\_ :8;

};

ObObjMeta meta\_;

int32\_t varchar\_len\_;

**union** {

int64\_t int\_val;

**float** float\_val;

**double** double\_val;

ObDateTime time\_val;

ObPreciseDateTime precisetime\_val;

**const** **char** \*varchar\_val;

} value\_;

}

# 查询操作的基本对象

**class** ObBorderFlag {

**public**:

**static** **const** int8\_t *INCLUSIVE\_START* = 0x1;

**static** **const** int8\_t *INCLUSIVE\_END* = 0x2;

**static** **const** int8\_t *MIN\_VALUE* = 0x4;

**static** **const** int8\_t *MAX\_VALUE* = 0x8;

**private**:

int8\_t data\_;

};

**struct** ObRange {

uint64\_t table\_id\_;

ObBorderFlag border\_flag\_;

ObString start\_key\_;

ObString end\_key\_;

};

**struct** ObCellInfo {

ObString table\_name\_; // 客户端发起请求时使用

uint64\_t table\_id\_; // 服务器个模块之间使用

ObString row\_key\_;

uint64\_t column\_id\_; // 0表示查询一行所有的列

ObString column\_name\_; // NULL表示查询一行所有的列

ObObj value\_;

};

**class** ObReadParam {

**public**:

**ObReadParam**();

**virtual** **~ObReadParam**();

**void** **set\_begin\_timestamp**(**const** int64\_t timestamp);

int64\_t **get\_begin\_timestamp**() **const**;

/// @fn get data whose timestamp is not newer than the given timestamp

/// -# when reading cs, if not setted, the result is decided by the server;

/// -# when reading ups, this parameter must be setted

**void** **set\_end\_timestamp**(**const** int64\_t timestamp);

int64\_t **get\_end\_timestamp**() **const**;

/// @fn when reading cs, indicating whether the result (including intermediate result,

/// like sstable block readed from sstable) of this operation should be cached.

///

/// ups just ignores this parameter

**void** **set\_is\_result\_cached**(**const** **bool** cached);

**bool** **get\_is\_result\_cached**() **const**;

NEED\_SERIALIZE\_AND\_DESERIALIZE;

**private**:

int64\_t begin\_timestamp;

int64\_t end\_timestamp\_;

**struct** ExtraReadInfo {

uint8\_t is\_result\_cached\_ :1;

uint8\_t reserved\_ :7;

};

**union** {

uint8\_t info\_value\_;

ExtraReadInfo info\_;

} extra\_read\_info\_;

};

**class** ObGetParam: **public** ObReadParam {

**public**:

**struct** ObRowIndex {

int32\_t offset\_;

int32\_t size\_;

};

**public**:

**ObGetParam**();

**virtual** **~ObGetParam**();

**int** **init\_row\_index**();

**int** **add\_cell**(**const** ObCellInfo& cell\_info);

**inline** int64\_t **get\_cell\_size**() **const** {

**return** cell\_info\_list\_.get\_array\_index();

}

**inline** **const** ObCellInfo\* **const** **get\_cell**() **const** {

**return** cell\_list\_;

}

**inline** int64\_t **get\_row\_size**() **const** {

**return** row\_size\_;

}

**inline** **const** ObRowIndex\* **get\_row\_index**() **const** {

**return** row\_index\_;

}

**inline** **const** **bool** **is\_inited\_row\_index**() **const** {

**return** row\_index\_inited\_;

}

NEED\_SERIALIZE\_AND\_DESERIALIZE;

**private**:

ObCellInfo cell\_list\_[OB\_MAX\_GET\_COLUMN\_NUMBER];

ObArrayHelper<ObCellInfo> cell\_info\_list\_;

ObRowIndex row\_index\_[OB\_MAX\_GET\_COLUMN\_NUMBER];

int32\_t row\_size\_;

int32\_t max\_column\_num\_;

**bool** row\_index\_inited\_;

};

**class** ObScanParam: **public** ObReadParam {

**public**:

**enum** Order {

*ASC* = 1, *DESC*

};

**ObScanParam**();

**virtual** **~ObScanParam**();

**void** **set**(**const** uint64\_t& table\_id, **const** ObString& table\_name,

**const** ObRange& range);

**int** **add\_column**(**const** ObString& column\_name);

**int** **add\_column**(**const** uint64\_t& column\_id);

**inline** **void** **set\_scan\_size**(**const** int64\_t scan\_size) {

scan\_size\_ = scan\_size;

}

**inline** uint64\_t **get\_table\_id**() **const** {

**return** table\_id\_;

}

**inline** **const** ObString **get\_table\_name**() **const** {

**return** table\_name\_;

}

**inline** **const** ObRange\* **const** **get\_range**() **const** {

**return** &range\_;

}

**inline** int64\_t **get\_scan\_size**() **const** {

**return** scan\_size\_;

}

**inline** int64\_t **get\_column\_name\_size**() **const** {

**return** column\_list\_.get\_array\_index();

}

**inline** int64\_t **get\_column\_id\_size**() **const** {

**return** column\_id\_list\_.get\_array\_index();

}

**inline** **const** ObString\* **const** **get\_column\_name**() **const** {

**return** column\_names\_;

}

**inline** **const** uint64\_t\* **const** **get\_column\_id**() **const** {

**return** column\_ids\_;

}

/// set and get order by information

**int** **add\_orderby\_column**(**const** ObString & column\_name, Order order = *ASC*);

**int** **add\_orderby\_column**(**const** uint64\_t column\_id, Order order = *ASC*);

int64\_t **get\_orderby\_column\_size**() **const**;

**void** **get\_orderby\_column**(ObString **const**\* & names, uint8\_t **const**\* & orders,

int64\_t &column\_size) **const**;

**void** **get\_orderby\_column**(uint64\_t **const**\* & column\_ids,

uint8\_t **const** \* & orders, int64\_t &column\_size) **const**;

/// set and get limit information

**int** **set\_limit\_info**(**const** int64\_t offset, **const** int64\_t count);

**void** **get\_limit\_info**(int64\_t &offset, int64\_t &count) **const**;

NEED\_SERIALIZE\_AND\_DESERIALIZE;

**private**:

uint64\_t table\_id\_;

ObString table\_name\_;

ObRange range\_;

ObString column\_names\_[OB\_MAX\_COLUMN\_NUMBER];

ObArrayHelper<ObString> column\_list\_;

uint64\_t column\_ids\_[OB\_MAX\_COLUMN\_NUMBER];

ObArrayHelper<uint64\_t> column\_id\_list\_;

int64\_t scan\_size\_;

ObString orderby\_column\_names\_[OB\_MAX\_COLUMN\_NUMBER];

ObArrayHelper<ObString> orderby\_column\_name\_list\_;

uint64\_t orderby\_column\_ids\_[OB\_MAX\_COLUMN\_NUMBER];

ObArrayHelper<uint64\_t> orderby\_column\_id\_list\_;

uint8\_t orderby\_orders\_[OB\_MAX\_COLUMN\_NUMBER];

ObArrayHelper<uint8\_t> orderby\_order\_list\_;

int64\_t limit\_offset\_;

/// @property 0 means not limit

int64\_t limit\_count\_;

};

**class** ObScanner {

MemBlock \*head\_memblock\_;

MemBlock \*cur\_memblock\_;

MemBlock \*rollback\_memblock\_;

int64\_t cur\_size\_counter\_;

int64\_t rollback\_size\_counter\_;

ObString cur\_table\_name\_;

uint64\_t cur\_table\_id\_;

ObString cur\_row\_key\_;

int64\_t mem\_size\_limit\_;

Iterator iter\_;

**bool** first\_next\_;

ObString ext\_info\_;

**bool** have\_ext\_info\_;

ObStringBuf string\_buf\_;

int64\_t timestamp\_;

/// @property fullfilled item number

/// -# get, this means cell number fullfilled

/// -# scan, this means row number fullfilled

int64\_t fullfilled\_item\_num\_;

**struct** ExtraResultInfo {

uint8\_t is\_timestamp\_setted\_ :1;

uint8\_t is\_request\_fullfilled\_ :1;

uint8\_t reserved\_ :6;

};

**union** {

uint8\_t info\_value\_;

ExtraResultInfo info\_;

} extra\_result\_info\_;

};

# 更新操作的基本对象

**class** ObMutator: **public** ObIterator {

**public**:

// Adds cell mutation to list

**int** **add\_cell**(**const** ObCellInfo& cell);

**public**:

**virtual** **void** **reset\_iter**();

**virtual** **int** **next\_cell**();

**virtual** **int** **get\_cell**(ObCellInfo\*\* cell);

**private**:

CellInfoNode\* list\_head\_;

CellInfoNode\* list\_tail\_;

PageArena<CellInfoNode> page\_arena\_;

ObStringBuf str\_buf\_;

ObString last\_row\_key\_;

ObString last\_table\_name\_;

int64\_t store\_size\_;

CellInfoNode\* cur\_iter\_node\_;

**bool** has\_begin\_;

};

# 查询接口

ObResultCode **ms\_get**(**const** ObGetParam& get\_param, ObScanner& scanner);

ObResultCode **up\_get**(**const** ObGetParam& get\_param, ObScanner& scanner);

ObResultCode **cs\_get**(**const** ObGetParam& get\_param, ObScanner& scanner);

ObResultCode **rt\_get**(**const** ObGetParam& get\_param, ObScanner& scanner);

ObResultCode **ms\_scan**(**const** ObScanParam& scan\_param, ObScanner& scanner);

ObResultCode **up\_scan**(**const** ObScanParam& scan\_param, ObScanner& scanner);

ObResultCode **cs\_scan**(**const** ObScanParam& scan\_param, ObScanner& scanner);

ObResultCode **cs\_scan**(**const** ObScanParam& scan\_param, ObScanner& scanner);

# 更新接口

ObResultCode **up\_apply**(**const** ObMutator& mutator);

# Rootserver提供的内部服务

## 基本对象

**struct** ObServer {

int32\_t version\_;

int32\_t port\_;

**struct** {

int32\_t v4\_;

int32\_t v6[4]\_;

} ip\_;

};

**struct** ObServerList {

int32\_t size;

ObServer\* servers;

};

**struct** ObTablet {

ObRange range;

int32\_t record\_count; // 记录条数

int32\_t occupy\_size; // 占用空间

};

**struct** ObTabletList {

int32\_t size;

ObTablet\* tablets;

};

**struct** ObTabletLocation {

uint64\_t tablet\_id;

ObServer chunkserver;

};

**struct** ObTabletLocationList {

**static** **const** int32\_t *MAX\_REPLICAS* = 3;

ObTabletLocation location[*MAX\_REPLICAS*];

};

**struct** ObRootMeta {

ObTablet tablet;

ObTabletLocationList location\_list;

};

**struct** ObRootMetaList {

int32\_t size;

ObRootMeta\* metas;

};

## 给client的接口

/\*

\* 获取updateserver地址

\* @param [out] updateserver地址信息

\*/

ObResultCode **rt\_get\_update\_server\_info**(ObServer& update\_server\_info);

## 汇报与注册接口

/\*

\* ChunkServer向RootServer汇报其管理的tablets;

\* @param [in] server 汇报的ChunkServer

\* @param [in] tablets 管理的所有tablets信息

\* @param [in] timestamp

\*/

ObResultCode **rt\_report\_tablets**(**const** ObServer& server,

**const** ObTabletList& tablets, uint64\_t time\_stamp);

/\*

\* 向RootServer注册，可以是Mergeserver和Chunkserver

\* @param [in] server 需要注册的server信息

\* @param [in] is\_merge\_server 是否为mergeserver

\* @param [in] status 0表示不开始汇报，1表示开始汇报

\*/

ObResultCode **rt\_register**(**const** ObServer& server, **const** **bool** is\_merge\_server,

int32\_t& status);

/\*

\* 迁移完成信息

\* @param [in] range 迁移完成的range

\* @param [in] src\_server 迁出的源服务器

\* @param [in] dest\_server 迁入的目标服务器

\* @keep\_src [in] 是否保留源数据

\*/

ObResultCode **rt\_migrate\_over**(**const** ObRange& range, **const** ObServer& src\_server,

**const** ObServer& dest\_server, **const** **bool** keep\_src);

/\*

\* Chunkserver容量汇报

\* @param [in] server Chunkserver信息

\* @param [in] capacity 总容量

\* @param [in] used 已用容量

\*/

ObResultCode **rt\_report\_capacity\_info**(**const** ObServer& server,

**const** int64\_t capacity, **const** int64\_t used);

## Schema服务

/\*

\* 从Root Server网络获取table schema信息

\* 这个接口只有ChunkServer和UpdateServer在每天合并之前进行调用，对于RootServer

\* 来说，必须在调用之前准备好新的schema，这个时间窗口比较重要。

\* @param [in] timestamp

\* @param [out] schema 获取的schema信息

\*/

ObResultCode **rt\_fetch\_schema**(**const** int64\_t timestamp, ObSchemaManager & schema);

/\*

\* schema切换完成，当合并时，rootserver发送命令让updateserver和chunkserver开始切换shcema，

× 当切换完成后，updateserver和chunkserver通过该接口通知rootserver切换完成

\* @param [in] timestamp

\* @param [in] server 完成切换的server信息

\*/

ObResultCode **rt\_schmea\_changed**(**const** int64\_t timestamp, **const** ObServer server);

# UpdateServer提供的内部服务

/\*

\* 开始冻结内存表

\* @param [in] timestamp 新schema的timestamp

\*/

ObResultCode **ups\_freeze\_memtable**(**const** int64\_t timestamp);

/\*

\* 卸载Memtable，等到所有的ChunkServer合并完成以后，Root Server发送卸载Memtable指令

\*/

ObResultCode **drop\_memtable**();

/\*

\* slave updateserver注册

\* @param [in] slave slave信息

\* @param [out] 同步相关的信息

\*/

ObResultCode **ups\_slave\_register**(**const** ObServer& slave,

**const** ObFetchParam& fetch\_param);

/\*

 \* 获取最近一次冻结内存表的版本号

 \*

 \* @param [out] version 最近一次冻结内存表的版本号

 \* @param [in] dump\_time 建议updateserver在这段时间内（单位为微秒）

\*                        不将冻结后的SSTable发送给chunkserver

\*                        该参数为0时，updateserver将忽略

\*/

ObResultCode **ups\_get\_latest\_frozen\_version**(int64\_t& version**,**

**const** int64\_t dump\_time);

# Chunkserver提供的内部服务

/\*

\* Chunkserver开始新schema

\* @param [in] timestamp 新schema的timetamp

\* @param [in] init\_flag 切换/汇报

\*/

ObResultCode **cs\_start\_schema**(**const** int64\_t timestamp, **const** int32\_t init\_flag);

/\*

\* Chunkserver释放老的tablet

\* @param [in] timestamp drop基于的timestamp

\*/

ObResultCode **cs\_drop\_old\_tablets**(**const** int64\_t timestamp);

/\*

\* Chunkserver用于接收RootServer的心跳请求

\* @param [in] lease\_duration lease的有效期

\*/

ObResultCode **cs\_heart\_beat**(**const** int64\_t lease\_duration);

/\*

\* ChunkServer迁移Tablet

\* @param [in] range 需要迁移的range

\* @param [in] dest\_server 迁移的目标服务器

\* @param [in] keep\_src 是否保留源数据

\*/

ObResultCode **cs\_migrate\_tablet**(**const** ObRange& range,

**const** ObServer& dest\_server, **const** **bool** keep\_src);

/\*

\* 迁移目标ChunkServer load本次迁移所获取的新tablet

\* @param [in] range 新的range

\* @param [in] size SSTable文件个数

\* @param [in] path SSTable文件列表

\*/

ObResultCode **cs\_load\_tablet**(**const** ObRange& range, **const** int64\_t size,

**const** **char**(\*path)[OB\_MAX\_FILE\_NAME\_LENGTH]);