ALIBABA CLOUD

阿里云

表格存储Tablestore

API 参考

文档版本：20201126

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.API 概览

1.1. 操作汇总

本文介绍表格存储提供的相关API接口。

如需了解表格存储各场景的应用案例，参见适用场景。

单行数据操作：

- GetRow
- PutRow
- UpdateRow
- DeleteRow

多行数据操作：

- GetRange
- BatchGetRow
- BatchWriteRow

数据流Stream操作：

- ListStream
- DescribeStream
- GetShardIterator
- GetStreamRecord

表操作：

- CreateTable
- ListTable
- DeleteTable
- UpdateTable
- DescribeTable
- ComputeSplitPointsBySize

索引操作：

- CreateIndex
- DeleteIndex

1.2. GetRow

您可以使用此接口根据指定的主键读取单行数据。

请求结构

```
message GetRowRequest {
  required string table_name = 1;
  required bytes primary_key = 2;  // Plainbuffer编码为二进制
  repeated string columns_to_get = 3; // 不指定则读出所有的列
  optional TimeRange time_range = 4;
  optional int32 max_versions = 5;
  optional bytes filter = 7;
  optional string start_column = 8;
  optional string end_column = 9;
  optional bytes token = 10;
}
```

名称	类型	是否必选	描述
table_name	string	是	要读取的数据所在的表名。
primary_key	bytes	是	该行全部的主键列，包含主键名和主键值，由Plainbuffer。
columns_to_get	repeated string	否	• 需要返回的全部列的列名。若为空，则返回该行的所有列。 • 如果指定的列不存在，则不会返回该列的数据。 • 如果给出了重复的列名，返回结果只会包含一次该列。 • columns_to_get 中 string 的个数不应超过 128 个。
time_range	TimeRange	和max_versions必须至少存在一个	• 读取数据的版本时间戳范围。 • 时间戳的取值最小值为0，最大值为INT64.MAX。 • 若要查询一个范围，则指定start_time和end_time。 • 若要查询一个特定时间戳，则指定specific_time。 例子：如果指定的time_range为（100，200），则返回的列数据的时间戳必须位于（100，200）范围内，前闭后开区间。
max_versions	int32	和time_range只能存在一个	读取数据时，返回的最多版本个数。 例子：如果指定max_versions为2，则每一列最多返回2个版本的数据。
filter	bytes	否	过滤条件表达式。 Filter经过protobuf序列化后的二进制数据。

名称	类型	是否必选	描述
start_column	string	否	• 指定读取时的起始列，主要用于宽行读。 • 返回的结果中包含当前起始列。 • 列的顺序按照列名的字典序排序。 例子：如果一张表有a、b、c三列，读取时指定start_column为b，则会从b列开始读，返回b、c两列。
end_column	string	否	• 指定读取时的结束列，主要用于宽行读。 • 返回的结果中不包含当前结束列。 • 列的顺序按照列名的字典序排序。 例子：如果一张表有a、b、c三列，读取时指定end_column为b，则读到b列时会结束，返回a列。

响应消息结构

```
message GetRowResponse {
  required ConsumedCapacity consumed = 1;
  required bytes row = 2; // Plainbuffer编码为二进制
}
```

名称	类型	描述
consumed	CapacityUnit	本次操作消耗的服务能力单元。
row	bytes	读取到的数据，由Plainbuffer编码。 如果该行不存在，则数据为空。

服务能力单元消耗

- 如果请求的行不存在，消耗 1 读服务能力单元。
- 如果请求的行存在，消耗读服务能力单元的数值为这该行所有主键列的数据大小与实际读取的属性列数据大小之和除以 4 KB 向上取整。关于数据大小的计算请参见产品定价。
- 如果请求超时，结果未定义，服务能力单元有可能被消耗，也可能未被消耗。
- 如果返回内部错误（HTTP 状态码：5XX），则此次操作不消耗服务能力单元，其他错误情况消耗 1 读服务能力单元。

1.3. PutRow

您可以使用PutRow接口插入数据到指定的行。

> **说明**
> - 如果指定行不存在，则新增一行；若指定行存在，则覆盖原有行。
> - 返回结果中如没有出现报错表示本次操作成功。

请求消息结构

```
message PutRowRequest {
  required string table_name = 1;
  required bytes row = 2; // Plainbuffer编码为二进制
  required Condition condition = 3;
  optional ReturnContent return_content = 4;
}
```

名称	类型	是否必选	描述
table_name	string	是	请求写入数据的表名。
row	bytes	是	写入的行数据， 包括主键和属性列，Plainbuffer格式，编码详见Plainbuffer编码。
condition	Condition	是	在数据写入前是否进行行存在性检查，可以取下面三个值： • IGNORE 表示不做行存在性检查。 • EXPECT_EXIST 表示期望行存在。 • EXPECT_NOT_EXIST 表示期望行不存在。
return_content	ReturnContent	否	写入成功后返回的数据类型，目前仅支持返回主键，主要用于主键列自增功能中。

响应消息结构

```
message PutRowResponse {
  required ConsumedCapacity consumed = 1;
  optional bytes row = 2;
}
```

名称	类型	描述
consumed	ConsumedCapacity	本次操作消耗的服务能力单元。

名称	类型	描述
row	bytes	• 当设置了return_content后，返回的数据。 • 如果没设置return_content或者没返回数据，此处为NULL。 • Plainbuffer格式，编码详见Plainbuffer编码。

服务能力单元消耗

- 如果该行不存在：
  - 若指定条件检查为 IGNORE，消耗写服务能力单元的数值为本行的主键数据大小与要插入属性列数据大小之和除以 4 KB 向上取整。
  - 若指定条件检查为 EXPECT_NOT_EXIST，除了消耗本行的主键数据大小与要插入属性列数据大小之和除以 4 KB 向上取整的写CU，还需消耗该行主键数据大小除以 4 KB 向上取整的读 CU。
  - 若指定条件检查为 EXPECT_EXIST，本次插入失败并且消耗 1 单位写 CU 和 1 单位读 CU。
- 如果该行存在：
  - 若指定条件检查为 IGNORE，消耗写服务能力单元的数值为本行的主键数据大小与要插入属性列数据大小之和除以 4 KB 向上取整。
  - 若指定条件检查为 EXPECT_EXIST，除了消耗本行的主键数据大小与要插入属性列数据大小之和除以 4 KB 向上取整的写 CU，还需消耗该行主键数据大小除以 4 KB 向上取整的读 CU。
  - 若指定条件检查为 EXPECT_NOT_EXIST，本次插入失败并且消耗 1 单位写 CU 和 1 单位读 CU。
- 使用条件更新（Conditional Update）：
  - 操作成功，按照上述消耗服务能力单元方式进行计算。
  - 操作失败，则消耗 1 单位写 CU 和 1 单位读 CU。

  关于数据大小的计算请参见产品定价。
- 如果返回内部错误（HTTP 状态码：5XX），则此次操作不消耗服务能力单元；其他错误情况消耗 1 个写服务能力单元。
- 如果请求超时，结果未定义，服务能力单元有可能被消耗，也可能未被消耗。

1.4. UpdateRow

使用UpdateRow接口更新指定行的数据。

> **说明** 如果指定行不存在，则新增一行；若指定行存在，则根据请求的内容在该行中新增、修改或者删除指定列的值。

请求消息结构

```
message UpdateRowRequest {
  required string table_name = 1;
  required bytes row_change = 2;
  required Condition condition = 3;
  optional ReturnContent return_content = 4;
}
```

名称	类型	是否必选	描述
table_name	string	是	请求更新数据的表名。

row_change	bytes	是	• 更新的数据，包括主键和属性列，Plainbuffer格式，编码详情请参见Plainbuffer编码。
			• 该行本次需要更新的全部属性列，表格存储会根据row_change中UpdateType的内容在这一行中新增、修改或者删除指定列的值。
			• 该行已存在的且不在row_change中的列将不受影响。
			• UpdateType可以取如下值：
			  ○ PUT：此时value必须为有效的属性列值。语意为如果该列不存在，则新增一列；如果该列存在，则覆盖该列。
			  ○ DELETE：此时该value必须为空，需要指定timestamp。语意为删除该列特定版本的数据。
			  ○ DELETE_ALL：此时该value和timestamp都必须为空。语意为删除该列所有版本的数据。
			说明 删除本行的全部属性列不等同于删除本行，若需要删除本行，请使用DeleteRow操作。

condition	Condition	是	• 在数据更新前是否进行存在性检查，可以取如下值：
			  ○ IGNORE表示不做行存在性检查。
			  ○ EXPECT_EXIST表示期望行存在。
			• 若期待该行存在但该行不存在，则本次更新操作会失败，返回错误；若忽视该行是否存在，则无论该行是否存在，都不会因此导致本次操作失败。

return_content	ReturnContent	否	写入成功后返回的数据类型，目前仅支持返回主键，主要用于主键列自增功能中。

响应消息结构

```
message UpdateRowResponse {
  required ConsumedCapacity consumed = 1;
  optional bytes row = 2;
}
```

名称	类型	描述
consumed	ConsumedCapacity	本次操作消耗的服务能力单元。
row	bytes	• 当设置了return_content后，返回的数据。 • 如果未设置return_content或者没返回值，此处为NULL。 • Plainbuffer格式，编码详情请参见Plainbuffer编码。

服务能力单元消耗

- 如果该行不存在：
  - 若指定条件检查为IGNORE，消耗写服务能力单元的数值为本行的主键数据大小与要更新的属性列数据大小之和除以4 KB向上取整。如果UpdateRow中包含有需要删除的属性列，只有其列名长度计入该属性列数据大小。
  - 若指定条件检查为EXPECT_EXIST，本次插入失败并且消耗1单位写CU和1单位读CU。
- 如果该行存在：
  - 若指定条件检查为IGNORE，消耗写服务能力单元的数值为本行的主键数据大小与要更新的属性列数据大小之和除以4 KB向上取整。如果UpdateRow中包含有需要删除的属性列，只有其列名长度计入该属性列数据大小。
  - 若指定条件检查为EXPECT_EXIST，除了需要消耗在条件检查为IGNORE情况下的写CU，还需消耗该行主键数据大小除以4 KB向上取整的读CU。

  关于数据大小的计算请参见产品定价。
- 如果请求超时，结果未定义，服务能力单元有可能被消耗，也可能未被消耗。
- 如果返回内部错误（HTTP状态码：5XX），则此次操作不消耗服务能力单元；其他错误情况消耗1个写服务能力单元和1个读服务能力单元。

1.5. DeleteRow

您可以使用DeleteRow接口删除一行数据。

请求消息结构

```
message DeleteRowRequest {
  required string table_name = 1;
  required bytes primary_key = 2; // Plainbuffer编码为二进制
  required Condition condition = 3;
  optional ReturnContent return_content = 4;
}
```

名称	类型	是否必选	描述
table_name	string	是	请求更新数据的表名。
primary_key	bytes	是	删除行的主键，Plainbuffer格式，编码详见Plainbuffer编码。

condition	Condition	是	• 在数据更新前是否进行存在性检查，可以取下面两个值：
			  ○ IGNORE：表示不做行存在性检查。
			  ○ EXPECT_EXIST：表示期望行存在。
			• 若期待该行存在，但实际该行不存在，则本次删除操作会失败，返回错误；若忽视该行是否存在，则无论该行实际是否存在，都不会因此导致操作失败。

return_content	ReturnContent	否	写入成功后返回的数据类型。目前仅支持返回主键，主要用于主键列自增功能中。

响应消息结构

```
message DeleteRowResponse {
  required ConsumedCapacity consumed = 1;
  optional bytes row = 2;
}
```

名称	类型	描述
consumed	ConsumedCapacity	本次操作消耗的服务能力单元。
row	bytes	• 当设置了return_content后，返回的数据。 • 如果没有设置return_content或者没有返回值，此处为NULL。 • Plainbuffer格式，编码详见Plainbuffer编码。

服务能力单元消耗

- 如果该行不存在：

- 若指定条件检查为 IGNORE，消耗写服务能力单元的数值为该行主键数据大小除以 4 KB 向上取整。
- 若指定条件检查为 EXPECT_EXIST，删除该行失败，消耗 1 单位的写 CU 和 1 单位的读 CU。

- 如果该行存在：
  - 若指定条件检查为 IGNORE，消耗写服务能力单元的数值为该行主键数据大小除以 4 KB 向上取整。
  - 若指定条件检查为 EXPECT_EXIST，除了消耗该行主键数据大小除以 4 KB 向上取整的写 CU，还需消耗该行主键数据大小除以 4 KB 向上取整的读 CU。

  关于数据大小的计算请参见产品定价。
- 如果返回内部错误（HTTP 状态码：5XX），则此次操作不消耗服务能力单元；其他错误情况消耗 1 个写服务能力单元。
- 如果请求超时，结果未定义，服务能力单元有可能被消耗，也可能未被消耗。

1.6. GetRange

您可以使用GetRange接口读取指定主键范围内的数据。

请求结构

```
message GetRangeRequest {
  required string table_name = 1;
  required Direction direction = 2;
  repeated string columns_to_get = 3; // 不指定则读出所有的列
  optional TimeRange time_range = 4;
  optional int32 max_versions = 5;
  optional int32 limit = 6;
  required bytes inclusive_start_primary_key = 7; // Plainbuffer编码为二进制
  required bytes exclusive_end_primary_key = 8; // Plainbuffer编码为二进制
  optional bytes filter = 10;
  optional string start_column = 11;
  optional string end_column = 12;
}
```

名称	类型	是否必选	描述
table_name	string	是	要读取的数据所在的表名。

名称	类型	是否必选	描述
direction	Direction	是	本次查询的顺序。 • 若为正序，则 inclusive_start_primary 应小于 exclusive_end_primary，响应中各行按照主键由小到大的顺序进行排列。 • 若为逆序，则 inclusive_start_primary 应大于 exclusive_end_primary，响应中各行按照主键由大到小的顺序进行排列。
columns_to_get	repeated string	否	• 需要返回的全部列的列名。若为空，则返回读取结果中每行的所有列。 • 如果给出了重复的列名，返回结果只会包含一次该列。 • columns_to_get 中 string 的个数不应超过 128 个。
time_range	TimeRange	和 max_versions 只能存在一个	• 读取数据的版本时间戳范围。 • 时间戳的取值最小值为0，最大值为INT64.MAX。 • 若要查询一个范围，则指定start_time和end_time。 • 若要查询一个特定时间戳，则指定specific_time。 例子：如果指定的time_range为(100, 200)，则返回的列数据的时间戳必须位于[100, 200)范围内，前闭后开区间。
max_versions	int32	和time_range 只能存在一个	读取数据时，返回的最多版本个数。 例子：如果指定max_versions为2，则每一列最多返回2个版本的数据。
limit	int32	否	• 本次读取最多返回的行数，若查询到的行数超过此值，则通过响应中包含的断点记录本次读取到的位置，以便下一次读取。此值必须大于 0。 • 无论是否设置此值，表格存储最多返回行数为 5000 且总数据大小不超过 4 MB。
inclusive_start_primary_key	bytes	是	本次范围读取的起始主键，若该行存在，则响应中一定会包含此行。由Plainbuffer编码，详见Plainbuffer编码。

名称	类型	是否必选	描述
exclusive_end_primary_key	bytes	是	• 本次范围读取的终止主键，无论该行是否存在，响应中都不会包含此行。由Plainbuffer编码，详见Plainbuffer编码。 • 在 GetRange 中，inclusive_start_primary_key 和 exclusive_end_primary_key 中的 Column 的 type 可以使用本操作专用的两个类型 INF_MIN 和 INF_MAX。类型为 INF_MIN 的 Column 小于其它 Column；类型为 INF_MAX 的 Column 大于其它 Column。
filter	bytes	否	• 过滤条件表达式。 • Filter经过protobuf序列化后的二进制数据。
start_column	string	否	• 指定读取时的起始列，主要用于宽行读。 • 返回的结果中包含当前起始列。 • 列的顺序按照列名的字典序排序。 例子：如果一张表有"a"，"b"，"c"三列，读取时指定start_column为“b”，则会从"b"列开始读，返回"b"，"c"两列。
end_column	string	否	• 指定读取时的结束列，主要用于宽行读。 • 返回的结果中不包含当前结束列。 • 列的顺序按照列名的字典序排序。 例子：如果一张表有"a"，"b"，"c"三列，读取时指定end_column为“b”，则读到"b"列时会结束，返回"a"列。

响应消息结构

```
message GetRangeResponse {
  required ConsumedCapacity consumed = 1;
  required bytes rows = 2;
  optional bytes next_start_primary_key = 3;
}
```

名称	类型	描述
consumed	ConsumedCapacity	本次操作消耗的服务能力单元。

名称	类型	描述
rows	bytes	• 由Plainbuffer编码，详见Plainbuffer编码。 • 读取到的所有数据，若请求中 direction 为 FORWARD，则所有行按照主键由小到大进行排序；若请求中 direction 为 BACKWARD，则所有行按照主键由大到小进行排序。 • 其中每行的 primary_key_columns 和 attribute_columns 均只包含在 columns_to_get 中指定的列，其顺序不保证与 request 中的 columns_to_get 一致；primary_key_columns 的顺序亦不保证与建表时指定的顺序一致。 • 如果请求中指定的 columns_to_get 不含有任何主键列，那么其主键在查询范围内。但没有任何一个属性列在 columns_to_get 中的行将不会出现在响应消息里。
next_start_primary_key	bytes	• 本次 GetRange 操作的断点信息。由Plainbuffer编码，详见Plainbuffer编码。 • 若为空，则本次 GetRange 的响应消息中已包含了请求范围内的所有数据。 • 若不为空，则表示本次 GetRange 的响应消息中只包含了 [inclusive_start_primary_key, next_start_primary_key) 间的数据，若需要剩下的数据，需要将 next_start_primary_key 作为 inclusive_start_primary_key，原始请求中的 exclusive_end_primary_key 作为 exclusive_end_primary_key 继续执行 GetRange 操作。 **说明** 表格存储系统中限制了 GetRange 操作的响应消息中数据不超过 5000 行，大小不超过 4 MB。即使在 GetRange 请求中未设定 limit，在响应中仍可能出现 next_start_primary_key。因此在使用 GetRange 时一定要对响应中是否有 next_start_primary_key 进行处理。

服务能力单元消耗

- GetRange 操作消耗读服务能力单元的数值为查询范围内所有行主键数据大小与实际读取的属性列数据大小之和除以 4 KB 向上取整。关于数据大小的计算请参见产品定价。
- 如果请求超时，结果未定义，服务能力单元有可能被消耗，也可能未被消耗。
- 如果返回内部错误（HTTP 状态码：5XX），则此次操作不消耗服务能力单元，其他错误情况消耗 1 个读服务能力单元。

1.7. BatchGetRow

批量读取一个或多个表中的若干行数据。

BatchGetRow 操作可视为多个 GetRow 操作的集合，各个操作独立执行，独立返回结果，独立计算服务能力单元。

与执行大量的 GetRow 操作相比，使用 BatchGetRow 操作可以有效减少请求的响应时间，提高数据的读取速率。

请求结构

```
message BatchGetRowRequest {
  repeated TableInBatchGetRowRequest tables = 1;
}
```

请求参数

参数	类型	是否必需	描述
tables	repeated TableInBatchGetRowRequest	是	指定了需要要读取的行信息。 若 tables 中出现了下述情况，则操作整体失败，返回错误。 • tables 中任一表不存在。 • tables 中任一表名不符合命名规则和数据类型。 • tables 中任一行未指定主键、主键名称不符合规范或者主键类型不正确。 • tables 中任一表的 columns_to_get 内的列名不符合命名规则和数据类型。 • tables 中包含同名的表。 • tables 中任一表内包含主键完全相同的行。 • 所有 tables 中 RowInBatchGetRowRequest 的总个数超过 100 个。 • tables 中任一表内不包含任何 RowInBatchGetRowRequest。 • tables 中任一表的 columns_to_get 超过 128 列。

响应消息结构

```
message BatchGetRowResponse {
  repeated TableInBatchGetRowResponse tables = 1;
}
```

> **说明** BatchGetRow 操作可能会在行级别部分失败，此时返回的 HTTP 状态码仍为 200。应用程序必须对 RowInBatchGetRowResponse 中的 error 进行检查确认每一行的执行结果，并进行相应的处理。

响应参数

参数	类型	描述

参数	类型	描述
tables	repeated TableInBatchGetRowResponse	• 对应了每个 table 下读取到的数据。 • 响应消息中 TableInBatchGetRowResponse 对象的顺序与 BatchGetRowRequest 中的 TableInBatchGetRowRequest 对象的顺序相同；每个 TableInBatchGetRowResponse 下面的 RowInBatchGetRowResponse 对象的顺序与 TableInBatchGetRowRequest 下面的 RowInBatchGetRowRequest 相同。 • 如果某行不存在或者某行在指定的 columns_to_get 下没有数据，仍然会在 TableInBatchGetRowResponse 中有一条对应的 RowInBatchGetRowResponse，但其 row 下面的 primary_key_columns 和 attribute_columns 将为空。 • 若某行读取失败，该行所对应的 RowInBatchGetRowResponse 中 is_ok 将为 false，此时 row 将为空。

服务能力单元消耗

- 如果本次操作整体失败，不消耗任何服务能力单元。
- 如果请求超时，结果未定义，服务能力单元有可能被消耗，也可能未被消耗。
- 其他情况将每个 RowInBatchGetRowRequest 视为一个 GetRow 操作独立计算写服务能力单元，具体请参考 GetRow 服务能力单元消耗。

1.8. BatchWriteRow

批量插入、修改或删除一个或多个表中的若干行数据。

BatchWriteRow 操作可视为多个 PutRow、UpdateRow、DeleteRow 操作的集合。各个操作独立执行，独立返回结果，独立计算服务能力单元。

与执行大量的单行写操作相比，使用 BatchWriteRow 操作可以有效减少请求的响应时间，提高数据的写入速率。

请求结构

```
message BatchWriteRowRequest {
  repeated TableInBatchWriteRowRequest tables = 1;
}
```

请求参数

参数	类型	是否必需	描述

参数	类型	是否必需	描述
tables	repeated TableInBatchWriteRowRequest	是	指定了需要要执行写操作的行信息。 以下情况都会返回整体错误： • tables 中任一表不存在。 • tables 中包含同名的表。 • tables 中任一表名不符合命名规则和数据类型。 • tables 中任一行操作未指定主键、主键列名称不符合规范或者主键列类型不正确。 • tables 中任一属性列名称不符合命名规则和数据类型。 • tables 中任一行操作存在与主键列同名的属性列。 • tables 中任一主键列或者属性列的值大小超过通用限制。 • tables 中任一表中存在主键完全相同的请求。 • tables 中所有表总的行操作个数超过 200 个，或者其含有的总数据大小超过 4 M。 • tables 中任一表内没有包含行操作，则返回 OTSParameterInvalidException 的错误。 • tables 中任一 PutRowInBatchWriteRowRequest 包含的 Column 个数超过 1024 个。 • tables 中任一 UpdateRowInBatchWriteRowRequest 包含的 ColumnUpdate 个数超过 1024 个。

响应消息结构

```
message BatchWriteRowResponse {
  repeated TableInBatchWriteRowResponse tables = 1;
}
```

响应参数

参数	类型	描述

参数	类型	描述
tables	TableInBatchWriteRowResponse	对应了每个 table 下各操作的响应信息，包括是否成功执行，错误码和消耗的服务能力单元。 响应消息中 TableInBatchWriteRowResponse 对象的顺序与 BatchWriteRowRequest 中的 TableInBatchWriteRowRequest 对象的顺序相同；每个 TableInBatchWriteRowRequest 中 put_rows、update_rows、delete_rows 包含的 RowInBatchWriteRowResponse 对象的顺序分别与 TableInBatchWriteRowRequest 中 put_rows、update_rows、delete_rows 包含的 PutRowInBatchWriteRowRequest、UpdateRowInBatchWriteRowRequest 和 DeleteRowInBatchWriteRowRequest 对象的顺序相同。 若某行读取失败，该行所对应的 RowInBatchWriteRowResponse 中 is_ok 将为 false。

说明 BatchWriteRow 操作可能会在行级别部分失败，此时返回的 HTTP 状态码仍为 200。应用程序必须对 RowInBatchWriteRowResponse 中的 error 进行检查，确认每一行的执行结果并进行相应的处理。

服务能力单元消耗

- 如果本次操作整体失败，不消耗任何服务能力单元。
- 如果请求超时，结果未定义，服务能力单元有可能被消耗，也可能未被消耗。
- 其他情况将每个 PutRowInBatchWriteRowRequest、UpdateRowInBatchWriteRowRequestDelete、RowInBatchWriteRowRequest 依次视作相对应的写操作独立计算读服务能力单元。请参考PutRow 服务能力单元消耗，UpdateRow 服务能力单元消耗和DeleteRow 服务能力单元消耗。

1.9. CreateTable

根据给定的表结构信息创建相应的表。

请求结构

```
message CreateTableRequest {
  required TableMeta table_meta = 1;
  required ReservedThroughput reserved_throughput = 2;
  optional TableOptions table_options = 3;
  optional StreamSpecification stream_spec = 5;
}
```

请求参数

参数	类型	是否必需	描述
table_meta	TableMeta	是	将要创建的表的结构信息，其中 table_name 应在本实例范围内唯一；primary_key 的中 ColumnSchema 的个数应在 1~4 个范围内；primary_key 中的 ColumnSchema 的 name 应符合命名规则和数据类型，type 取值只能为 STRING、 INTEGER 或BINARY。 建表成功后，表的 Schema 将不能修改。
reserved_throughput	ReservedThroughput	是	将要创建的表的初始预留读/写吞吐量设定，任何表的预留读吞吐量与预留写吞吐量均不能超过 5000。 表的预留读/写吞吐量设定可以通过 UpdateTable 进行动态更改。
table_options	TableOptions	是	主要设置TimeToLive和最大版本数。
stream_spec	StreamSpecification	否	描述是否打开Stream相关的属性。

响应消息结构

```
message CreateTableResponse {
}
```

注意事项

- 创建成功的表并不能立刻提供读写服务。一般来讲，在建表成功后一分钟左右，即可对新创建的表进行读写操作。
- 单个实例下不能超过 64 个表，如果需要提高单实例下表数目的上限，请提交工单。

1.10. ListTable

获取当前实例下已创建的所有表的表名。

请求结构：

```
message ListTableRequest {
}
```

响应消息结构：

```
message ListTableResponse {
  repeated string table_names = 1;
}
```

table_names:

- 类型：repeated string
- 当前实例下所有表的表名

> **说明** 若一个表刚刚创建成功，其表名会出现在 ListTableResponse 里，但此时该表不一定能够进行读写。

1.11. DeleteTable

删除本实例下指定的表。

请求结构

```
message DeleteTableRequest {
  required string table_name = 1;
}
```

table_name

- 类型：string
- 是否必要参数：是
- 需要删除的数据表的表名称

响应消息结构

```
message DeleteTableResponse {
}
```

DeleteTable 的响应中没有任何错误即表示表已经成功删除。

1.12. UpdateTable

更新指定表的预留读吞吐量或预留写吞吐量设置，新设定将于更新成功一分钟内生效。

请求结构

```
message UpdateTableRequest {
  required string table_name = 1;
  optional ReservedThroughput reserved_throughput = 2;
  optional TableOptions table_options = 3;
  optional StreamSpecification stream_spec = 4;
}
```

table_name

- 类型：string
- 是否必要参数：是
- 需要更改预留读写吞吐量设置的数据表对应的表名称。

reserved_throughput

- 类型：ReservedThroughput
- 是否必要参数：是
- 将要更改的表的预留读/写吞吐量设定，该设定将于一分钟后生效。
- 可以只更改表的预留读吞吐量的设置或只更改表的预留写吞吐量的设置，也可以一并更改。
- capacity_unit 中 read 和 write 应至少有一个非空，否则请求失败，返回错误。

table_options:

- 类型：TableOptions
- 是否必要参数：是
- 主要设置TimeToLive和最大版本数。

StreamSpecification

- 类型：StreamSpecification
- 是否必要参数：否
- 描述是否打开Stream等Stream相关的属性。

响应消息结构

```
message UpdateTableResponse {
  required ReservedThroughputDetails reserved_throughput_details = 1;
  required TableOptions table_options = 2;
}
```

capacity_unit_details

- 类型：ReservedThroughputDetails
- 更新后，该表的预留读/写吞吐量设置信息除了包含当前的预留读/写吞吐量设置值之外，还包含了最近一次更新该表的预留读/写吞吐量设置的时间和当日已下调预留读/写吞吐量的次数。

> 注意

- 调整每个表预留读/写吞吐量的最小时间间隔为 2 分钟，如果本次 UpdateTable 操作距上次不到 2 分钟将被拒绝。
- 每个自然日（UTC 时间 00:00:00 到第二天的 00:00:00）内每个表上调和下调预留读写吞吐量次数不限。

table_options

- 类型：TableOptions
- 修改后，最新的table_options参数值。

1.13. DescribeTable

查询指定表的结构信息和预留读/写吞吐量设置信息。

请求结构：

```
message DescribeTableRequest {
  required string table_name = 1;
}
```

table_name:

- 类型：string
- 是否必要参数：是
- 需要查询的表名。

响应消息结构：

```
message DescribeTableResponse {
  required TableMeta table_meta = 1;
  required ReservedThroughputDetails reserved_throughput_details = 2;
  required TableOptions table_options = 3;
  optional StreamDetails stream_details = 5;
  repeated bytes shard_splits = 6;
}
```

table_meta:

- 类型：TableMeta
- 该表的Schema，与建表时给出的Schema相同。

reserved_throughput_details:

- 类型：ReservedThroughputDetails

- 该表的预留读/写吞吐设置信息除了包含当前的预留读/写吞吐设置值之外，还包含了最近一次更新该表的预留读/写吞吐设置的时间和当日已下调预留读/写吞吐的次数。

table_options:

- 类型：TableOptions
- 当前最新的table_options参数值。

StreamSpecification:

- 类型：StreamSpecification
- 是否必要参数：否
- 描述是否打开Stream相关的属性。

shard_splits:

- 类型：bytes
- 当前表所有分区的分裂点。

1.14. ComputeSplitPointsBySize

将全表的数据在逻辑上划分成接近指定大小的若干分片，返回这些分片之间的分割点以及分片所在机器的提示。一般用于计算引擎规划并发度等执行计划。

请求结构

```
message ComputeSplitPointsBySizeRequest {
  required string table_name = 1;
  required int64 split_size = 2; // in 100MB
}
```

table_name:

- 类型：string。
- 是否必要参数：是
- 要切分的数据所在的表名。

split_size:

- 类型：int64
- 是否必要参数：是
- 每个分片的近似大小，以百兆为单位。

响应消息结构

```
message ComputeSplitPointsBySizeResponse {
  required ConsumedCapacity consumed = 1;
  repeated PrimaryKeySchema schema = 2;
  /**
   * Split points between splits, in the increasing order
   *
   * A split is a consecutive range of primary keys,
   * whose data size is about split_size specified in the request.
   * The size could be hard to be precise.
   *
   * A split point is an array of primary-key column w.r.t. table schema,
   * which is never longer than that of table schema.
   * Tailing -inf will be omitted to reduce transmission payloads.
   */
  repeated bytes split_points = 3;
  /**
   * Locations where splits lies in.
   *
   * By the managed nature of TableStore, these locations are no more than hints.
   * If a location is not suitable to be seen, an empty string will be placed.
   */
   message SplitLocation {
     required string location = 1;
     required sint64 repeat = 2;
   }
   repeated SplitLocation locations = 4;
}
```

consumed:

- 类型：ConsumedCapacity
- 本次请求消耗的服务能力单元。

schema:

- 类型：PrimaryKeySchema
- 该表的 Schema，与建表时给出的 Schema 相同。

split_points:

- 类型：repeated bytes
- 分片之间的分割点。分割点之间保证单调增。每个分割点都是以Plainbuffer编码的行，并且只有primary-key项。为了减少传输的数据量，分割点最后的-inf不会传输。

locations:

- 类型：repeated SplitLocation
- 分割点所在机器的提示。可以为空。

举个例子，如果有一张表有三列主键，其中首列主键类型为string。调用这个API后得到5个分片，分别为 (-inf,-inf,-inf) 到 ("a",-inf,-inf) 、 ("a",-inf,-inf) 到 ("b",-inf,-inf) 、 ("b",-inf,-inf) 到 ("c",-inf,-inf) 、 ("c",-inf,-inf) 到 ("d",-inf,-inf) 和 ("d",-inf,-inf) 到 (+inf,+inf,+inf) 。前三个落在"machine-A"，后两个落在"machine-B"。那么，split_points为（示意） [("a"),("b"),("c"),("d")] ，而locations为（示意） "machine-A"*3, "machine-B"*2 。

服务能力单元消耗

消耗的读服务能力单元数量与分片的数量相同。不消耗写服务能力单元。

1.15. ListStream

获取当前实例下所有表的stream信息。

请求结构：

```
message ListStreamRequest {
 optional string table_name = 1;
}
```

table_name:

- 类型：optional string
- 当前stream所属的表名

响应消息结构：

```
message ListStreamResponse {
  repeated Stream streams = 1;
}
message Stream {
  required string stream_id = 1;
  required string table_name = 2;
  required int64 creation_time = 3;
}
```

stream_id:

- 类型：required string
- 当前stream的id

table_name:

- 类型：required string
- 当前stream所属的表名

creation_time:

- 类型：required int64
- 当前stream enable的时间

1.16. DescribeStream

获取当前stream的shard信息。

请求结构：

```
message DescribeStreamRequest {
  required string stream_id = 1;
  optional string inclusive_start_shard_id = 2;
  optional int32 shard_limit = 3;
}
```

stream_id:

- 类型：required string
- 当前stream的id

inclusive_start_shard_id:

- 类型：required string
- 查询起始shard的id

shard_limit:

- 类型：required string
- 单次查询返回shard数目的上限

响应消息结构：

```
message DescribeStreamResponse {
  required string stream_id = 1;
  required int32 expiration_time = 2;
  required string table_name = 3;
  required int64 creation_time = 4;
  required StreamStatus stream_status = 5;
  repeated StreamShard shards = 6;
  optional string next_shard_id = 7;
}
message StreamShard {
  required string shard_id = 1;
  optional string parent_id = 2;
  optional string parent_sibling_id = 3;
}
```

stream_id:

- 类型：required string
- 当前stream的id

expiration_time:

- 类型：required int32
- Stream的过期时间

table_name:

- 类型：required string
- 当前stream 所属的table名字

creation_time:

- 类型：required int32
- 当前stream创建的时间

stream_status:

- 类型：required StreamStatus
- 当前stream的状态，包括enabling和active

shards:

- 类型：required StreamShard
- streamShard的信息，包括shard的id，父shard的id，父shard的邻居shard信息（适用于父shard发生merge）

next_shard_id:

- 类型：optional string
- 分页查询下一个shard的起始id

注意事项：

读取当前shard的数据时需要确保父shard的数据已经全部读取完毕。

1.17. GetShardIterator

获取读取当前shard记录的起始iterator。

请求结构：

```
message GetShardIteratorRequest {
  required string stream_id = 1;
  required string shard_id = 2;
}
```

stream_id:

- 类型：required string
- 当前stream的id

shard_id:

- 类型：required string
- 当前shard的id

响应消息结构：

```
message GetShardIteratorResponse {
  required string shard_iterator = 1;
}
```

shard_iterator:

- 类型：required string
- 读取当前shard记录的起始iterator

1.18. GetStreamRecord

读取当前shard的增量内容。

请求结构：

```
message GetStreamRecordRequest {
  required string shard_iterator = 1;
  optional int32 limit = 2;
}
```

shard_iterator:

- 类型：required string
- 当前shard读取的iterator

响应消息结构：

```
message GetStreamRecordResponse {
  message StreamRecord {
    required ActionType action_type = 1;
    required bytes record = 2;
  }
  repeated StreamRecord stream_records = 1;
  optional raw_string next_shard_iterator = 2;
  optional ConsumedCapacity consumed = 3;
}
```

StreamRecord:

- 类型：repeated StreamRecord
- 读取当前shard记录的record entry

shard_iterator:

- 类型：required string
- 下次读取此shard的iterator

consumed:

- 类型：ConsumedCapacity
- 本次操作消耗的服务能力单元。
- 读取Stream数据时CU的计算是根据读取所有行总大小除以4KB向上取整。行的数据大小计算方式请参见数据存储。

1.19. 表格存储 ProtocolBuffer 消息定义

下面是table_store.proto和table_store_filter.proto的详细定义。

table_store.proto

```
package com.alicloud.openservices.tablestore.core.protocol;
message Error {
  required string code = 1;
  optional string message = 2;
}
enum PrimaryKeyType {
  INTEGER = 1;
  STRING = 2;
  BINARY = 3;
```

```
}
enum PrimaryKeyOption {
  AUTO_INCREMENT = 1;
}
message PrimaryKeySchema {
  required string name = 1;
  required PrimaryKeyType type = 2;
  optional PrimaryKeyOption option = 3;
}
message TableOptions {
  optional int32 time_to_live = 1; // 可以动态更改
  optional int32 max_versions = 2; // 可以动态更改
 optional int64 deviation_cell_version_in_sec = 5; // 可以动态修改
}
message TableMeta {
  required string table_name = 1;
  repeated PrimaryKeySchema primary_key = 2;
}
enum RowExistenceExpectation {
  IGNORE = 0;
  EXPECT_EXIST = 1;
  EXPECT_NOT_EXIST = 2;
}
message Condition {
  required RowExistenceExpectation row_existence = 1;
  optional bytes column_condition     = 2;
}
message CapacityUnit {
  optional int32 read = 1;
  optional int32 write = 2;
}
message ReservedThroughputDetails {
  required CapacityUnit capacity_unit = 1; // 表当前的预留吞吐量的值。
  required int64 last_increase_time = 2; // 最后一次上调预留吞吐量的时间。
  optional int64 last_decrease_time = 3; // 最后一次下调预留吞吐量的时间。
}
message ReservedThroughput {
  required CapacityUnit capacity_unit = 1;
}
message ConsumedCapacity {
  required CapacityUnit capacity_unit = 1;
```

```
  required CapacityUnit capacity_unit = 1;
}
/* ################################################ CreateTable ################################
############## */
/**
 * table_meta用于存储表中不可更改的schema属性，可以更改的ReservedThroughput和TableOptions独立出来，
作为UpdateTable的参数。
 * message CreateTableRequest {
 *      required TableMeta table_meta = 1;
 *      required ReservedThroughput reserved_throughput = 2;
 *      required TableOptions table_options = 3;
 * }
 */
message CreateTableRequest {
  required TableMeta table_meta = 1;
  required ReservedThroughput reserved_throughput = 2;
  optional TableOptions table_options = 3;
}
message CreateTableResponse {
}
/* ##########################################################################################
################ */
/* ################################################ UpdateTable ################################
############## */
message UpdateTableRequest {
  required string table_name = 1;
  optional ReservedThroughput reserved_throughput = 2;
  optional TableOptions table_options = 3;
}
message UpdateTableResponse {
  required ReservedThroughputDetails reserved_throughput_details = 1;
  required TableOptions table_options = 2;
}
/* ##########################################################################################
################ */
/* ################################################ DescribeTable ################################
############## */
message DescribeTableRequest {
  required string table_name = 1;
}
message DescribeTableResponse {
```

```
  required TableMeta table_meta = 1;
  required ReservedThroughputDetails reserved_throughput_details = 2;
  required TableOptions table_options = 3;
  repeated bytes shard_splits = 6;
}
/* ##########################################################################################
################## */
/* ############################################ ListTable ###################################
########## */
message ListTableRequest {
}
/**
 * 当前只返回一个简单的名称列表
 */
message ListTableResponse {
  repeated string table_names = 1;
}
/* ##########################################################################################
############## */
/* ############################################ DeleteTable #################################
############ */
message DeleteTableRequest {
  required string table_name = 1;
}
message DeleteTableResponse {
}
/* ############################################ UnloadTable #################################
############ */
message UnloadTableRequest {
  required string table_name = 1;
}
message UnloadTableResponse {
}
/* ##########################################################################################
################# */
/**
 * 时间戳的取值最小值为0，最大值为INT64.MAX
 * 1. 若要查询一个范围，则指定start_time和end_time
 * 2. 若要查询一个特定时间戳，则指定specific_time
 */
message TimeRange {
```

```
  optional int64 start_time = 1;
  optional int64 end_time = 2;
  optional int64 specific_time = 3;
}
/* ############################################## GetRow ###################################
######### */
enum ReturnType {
  RT_NONE = 0;
  RT_PK = 1;
}
message ReturnContent {
  optional ReturnType return_type = 1;
}
/**
 * 1. 支持用户指定版本时间戳范围或者特定的版本时间来读取指定版本的列
 * 2. 目前暂不支持行内的断点
 */
message GetRowRequest {
  required string table_name = 1;
  required bytes primary_key = 2; // encoded as InplaceRowChangeSet, but only has primary key
  repeated string columns_to_get = 3; // 不指定则读出所有的列
  optional TimeRange time_range = 4;
  optional int32 max_versions = 5;
  optional bytes filter = 7;
  optional string start_column = 8;
  optional string end_column = 9;
  optional bytes token = 10;
}
message GetRowResponse {
  required ConsumedCapacity consumed = 1;
  required bytes row = 2; // encoded as InplaceRowChangeSet
  optional bytes next_token = 3;
}
/* ##########################################################################################
########### */
/* ############################################## UpdateRow ###################################
############ */
message UpdateRowRequest {
  required string table_name = 1;
  required bytes row_change = 2;
  required Condition condition = 3;
```

```
  required Condition condition = 3;
  optional ReturnContent return_content = 4;
}
message UpdateRowResponse {
  required ConsumedCapacity consumed = 1;
  optional bytes row = 2;
}
/* ##########################################################################################
############## */
/* ############################################## PutRow ###################################
######### */
/**
 * 这里允许用户为每列单独设置timestamp，而不是强制整行统一一个timestamp。
 */
message PutRowRequest {
  required string table_name = 1;
  required bytes row = 2; // encoded as InplaceRowChangeSet
  required Condition condition = 3;
  optional ReturnContent return_content = 4;
}
message PutRowResponse {
  required ConsumedCapacity consumed = 1;
  optional bytes row = 2;
}
/* ##########################################################################################
########### */
/* ############################################## DeleteRow ################################
########### */
/**
 * OTS只支持删除指定行的所有列的所有版本。
 */
message DeleteRowRequest {
  required string table_name = 1;
  required bytes primary_key = 2; // encoded as InplaceRowChangeSet, but only has primary key
  required Condition condition = 3;
  optional ReturnContent return_content = 4;
}
message DeleteRowResponse {
  required ConsumedCapacity consumed = 1;
  optional bytes row = 2;
}
```

```
/* ##########################################################################################################
############## */
/* ############################################## BatchGetRow ##############################
############## */
message TableInBatchGetRowRequest {
  required string table_name = 1;
  repeated bytes primary_key = 2; // encoded as InplaceRowChangeSet, but only has primary key
  repeated bytes token = 3;
  repeated string columns_to_get = 4;  // 不指定则读出所有的列
  optional TimeRange time_range = 5;
  optional int32 max_versions = 6;
  optional bytes filter = 8;
  optional string start_column = 9;
  optional string end_column = 10;
}
message BatchGetRowRequest {
  repeated TableInBatchGetRowRequest tables = 1;
}
message RowInBatchGetRowResponse {
  required bool is_ok = 1;
  optional Error error = 2;
  optional ConsumedCapacity consumed = 3;
  optional bytes row = 4; // encoded as InplaceRowChangeSet
  optional bytes next_token = 5;
}
message TableInBatchGetRowResponse {
  required string table_name = 1;
  repeated RowInBatchGetRowResponse rows = 2;
}
message BatchGetRowResponse {
  repeated TableInBatchGetRowResponse tables = 1;
}
/* ##########################################################################################################
################ */
/* ############################################## BatchWriteRow ##############################
############### */
enum OperationType {
  PUT = 1;
  UPDATE = 2;
  DELETE = 3;
}
```

```
message RowInBatchWriteRowRequest {
  required OperationType type = 1;
  required bytes row_change = 2; // encoded as InplaceRowChangeSet
  required Condition condition = 3;
  optional ReturnContent return_content = 4;
}
message TableInBatchWriteRowRequest {
  required string table_name = 1;
  repeated RowInBatchWriteRowRequest rows = 2;
}
message BatchWriteRowRequest {
  repeated TableInBatchWriteRowRequest tables = 1;
}
message RowInBatchWriteRowResponse {
  required bool is_ok = 1;
  optional Error error = 2;
  optional ConsumedCapacity consumed = 3;
  optional bytes row = 4;
}
message TableInBatchWriteRowResponse {
  required string table_name = 1;
  repeated RowInBatchWriteRowResponse rows = 2;
}
message BatchWriteRowResponse {
  repeated TableInBatchWriteRowResponse tables = 1;
}
/* ##########################################################################################
################## */
/* ############################################ GetRange ###################################
########## */
enum Direction {
  FORWARD = 0;
  BACKWARD = 1;
}
message GetRangeRequest {
  required string table_name = 1;
  required Direction direction = 2;
  repeated string columns_to_get = 3; // 不指定则读出所有的列
  optional TimeRange time_range = 4;
  optional int32 max_versions = 5;
  optional int32 limit = 6;
```

```
  optional int32 limit = 6;
  required bytes inclusive_start_primary_key = 7; // encoded as InplaceRowChangeSet, but only has primary key
  required bytes exclusive_end_primary_key = 8; // encoded as InplaceRowChangeSet, but only has primary key
  optional bytes filter = 10;
  optional string start_column = 11;
  optional string end_column = 12;
  optional bytes token = 13;
}
message GetRangeResponse {
  required ConsumedCapacity consumed = 1;
  required bytes rows = 2; // encoded as InplaceRowChangeSet
  optional bytes next_start_primary_key = 3; // 若为空，则代表数据全部读取完毕. encoded as InplaceRowChangeSet, but only has primary key
  optional bytes next_token = 4;
}
/* ##################### ComputeSplitPointsBySize ##################### */
message ComputeSplitPointsBySizeRequest {
  required string table_name = 1;
  required int64 split_size = 2; // in 100MB
}
message ComputeSplitPointsBySizeResponse {
  required ConsumedCapacity consumed = 1;
  repeated PrimaryKeySchema schema = 2;
  /**
   * Split points between splits, in the increasing order
   *
   * A split is a consecutive range of primary keys,
   * whose data size is about split_size specified in the request.
   * The size could be hard to be precise.
   *
   * A split point is an array of primary-key column w.r.t. table schema,
   * which is never longer than that of table schema.
   * Tailing -inf will be omitted to reduce transmission payloads.
   */
  repeated bytes split_points = 3;
  /**
   * Locations where splits lies in.
   *
   * By the managed nature of TableStore, these locations are no more than hints.
```

```
    * If a location is not suitable to be seen, an empty string will be placed.
    */
    message SplitLocation {
      required string location = 1;
      required sint64 repeat = 2;
    }
    repeated SplitLocation locations = 4;
}
```

table_store_filter.proto

```
package com.alicloud.openservices.tablestore.core.protocol;
enum FilterType {
  FT_SINGLE_COLUMN_VALUE = 1;
  FT_COMPOSITE_COLUMN_VALUE = 2;
  FT_COLUMN_PAGINATION = 3;
}
enum ComparatorType {
  CT_EQUAL = 1;
  CT_NOT_EQUAL = 2;
  CT_GREATER_THAN = 3;
  CT_GREATER_EQUAL = 4;
  CT_LESS_THAN = 5;
  CT_LESS_EQUAL = 6;
}
message SingleColumnValueFilter {
  required ComparatorType comparator = 1;
  required string column_name = 2;
  required bytes column_value = 3;
  required bool filter_if_missing = 4;
  required bool latest_version_only = 5;
}
enum LogicalOperator {
  LO_NOT = 1;
  LO_AND = 2;
  LO_OR = 3;
}
message CompositeColumnValueFilter {
  required LogicalOperator combinator = 1;
  repeated Filter sub_filters = 2;
}
message ColumnPaginationFilter {
  required int32 offset = 1;
  required int32 limit = 2;
}
message Filter {
  required FilterType type = 1;
  required bytes filter = 2;  // Serialized string of filter of the type
}
```

1.20. CreateIndex

在指定的主表上创建索引表。

请求结构：

```
message CreateIndexRequest {
  required string main_table_name = 1;
  required IndexMeta index_meta = 2;
  optional bool include_base_data = 3;
}
```

main_table_name:

- 类型：string
- 是否必要参数：是
- 索引表所在主表的名字

index_meta:

- 类型：IndexMeta (链接)
- 是否必要参数：是
- 索引表的schema

include_base_data:

- 类型：bool
- 是否必要参数：否
- 索引表中，是否包含在创建索引表前，主表的存量数据。

响应消息结构

```
message CreateIndexResponse {
}
```

1.21. DeleteIndex

在指定的主表上删除索引表。

请求结构：

```
message DropIndexRequest {
  required string main_table_name = 1;
  required string index_name = 2;
}
```

main_table_name:

- 类型：string

- 类型：string
- 是否必要参数：是
- 要删除的索引表所在的主表

index_name：

- 类型：string
- 是否必要参数：是
- 要删除的索引表名称

响应消息结构

```
message DropIndexResponse {
}
```

2.DataType

2.1. ActionType

在 GetStreamRecord 的响应消息中，表示操作类型。

- PUT_ROW 表示此次操作类型是PutRow
- UPDATE_ROW 表示此次操作类型是UpdateRow
- DELETE_ROW 表示此次操作类型是DeleteRow

枚举取值列表

```
enum ActionType {
  PUT_ROW = 1;
  UPDATE_ROW = 2;
  DELETE_ROW = 3;
}
```

相关操作

GetStreamRecord

2.2. CapacityUnit

表示一次操作消耗服务能力单元的值或是一个表的预留读/写吞吐量的值。

数据结构

```
message CapacityUnit {
  optional int32 read = 1;
  optional int32 write = 2;
}
```

read:

- 类型：int32
- 描述：本次操作消耗的读服务能力单元或该表的预留读吞吐量。

write:

- 类型：int32
- 描述：本次操作消耗的写服务能力单元或该表的预留写吞吐量。

相关操作

UpdateRow

BatchWriteRow

2.3. ColumnPaginationFilter

宽行读取过滤条件，适用于filter。

数据结构

```
message ColumnPaginationFilter {
  required int32 offset = 1;
  required int32 limit = 2;
}
```

offset：

- 类型：int32
- 描述：起始列的位置，表示从第几列开始读。

limit：

- 类型：int32
- 描述：读取的列的个数。

相关操作

Filter

GetRow

GetRange

BatchGetRow

2.4. ComparatorType

关系运算符，被定义成枚举类型。

- CT_EQUAL 表示相等。
- CT_NOT_EQUAL 表示不相等。
- CT_GREATER_THAN 表示大于。
- CT_GREATER_EQUAL 表示大于等于。
- CT_LESS_THAN 表示小于。
- CT_LESS_EQUAL 表示小于等于。

枚举取值列表

```
enum ComparatorType {
  CT_EQUAL             = 1;
  CT_NOT_EQUAL         = 2;
  CT_GREATER_THAN      = 3;
  CT_GREATER_EQUAL     = 4;
  CT_LESS_THAN         = 5;
  CT_LESS_EQUAL        = 6;
}
```

2.5. CompositeColumnValueFilter

多个组合条件，比如 column_a > 5 AND column_b = 10 等。适用于 ConditionUpdate 和 Filter 功能。

数据结构

```
message CompositeColumnValueFilter {
  required LogicalOperator combinator = 1;
  repeated Filter sub_filters = 2;
}
```

combinator:

- 类型：LogicalOperator
- 描述：逻辑操作符。

sub_filter:

- 类型：Filter
- 描述：子条件表达式。

相关操作

- ConditionUpdate
  - PutRow
  - UpdateRow
  - DeleteRow
  - BatchWriteRow
- Filter
  - GetRow
  - GetRange
  - BatchGetRow

2.6. PrimaryKeyType

主键的类型。

枚举数据类型

- INTEGER：整数
- STRING：字符串
- BINARY：二进制

```
enum PrimaryKeyType {
  INTEGER = 1;
  STRING = 2;
  BINARY = 3;
}
```

2.7. Condition

在 PutRow、UpdateRow 和 DeleteRow 中使用的行判断条件，目前含有 row_existence 和 column_condition 两项。

数据结构

```
message Condition {
  required RowExistenceExpectation row_existence = 1;
  optional bytes column_condition    = 2;
}
```

row_existence:

- 类型：RowExistenceExpectation
- 描述：对该行进行行存在性检查的设置。

column_condition:

- 类型：bytes
- 描述：对列条件的设置。Filter经过protobuf序列化后的bytes。

相关操作

PutRow

UpdateRow

DeleteRow

BatchWriteRow

2.8. ConsumedCapacity

表示一次操作消耗的服务能力单元。

数据结构

```
message ConsumedCapacity {
  required CapacityUnit capacity_unit = 1;
}
```

capacity_unit：

- 类型：CapacityUnit
- 描述：本次操作消耗的服务能力单元的值。

2.9. Direction

在 GetRange 操作中，查询数据的顺序。

- FORWARD 表示此次查询按照主键由小到大的顺序进行。
- BACKWARD 表示此次查询按照主键由大到小的顺序进行。

枚举取值列表

```
enum Direction {
  FORWARD = 0;
  BACKWARD = 1;
}
```

相关操作

GetRange

2.10. Error

用于在操作失败时的响应消息中表示错误信息，以及在 BatchGetRow 和 BatchWriteRow 操作的响应消息中表示单行请求的错误。

数据结构

```
Error {
  required string code = 1;
  optional string message = 2;
}
```

code：

- 类型：string
- 描述：当前单行操作的错误码。

message:

- 类型：string
- 描述：当前单行操作的错误信息，具体含义可参考错误码。

相关操作

BatchGetRow

BatchWriteRow

2.11. Filter

列判断条件，适用于条件更新（Conditional Update）和过滤器（Filter）功能。

数据结构

```
message Filter {
  required FilterType type = 1;
  required bytes filter = 2;
}
```

type:

- 类型：FilterType
- 描述：列条件类型。

filter:

- 类型：bytes
- 描述：CompositeColumnValueFilter 、ColumnPaginationFilter或者SingleColumnValueFilter类型的条件语句通过 Protobuf 序列化后的二进制数据。

相关操作

- ConditionUpdate
  - PutRow
  - UpdateRow
  - DeleteRow
  - BatchWriteRow
- Filter
  - GetRow
  - GetRange
  - BatchGetRow

2.12. FilterType

条件更新或过滤的类型。

- FT_SINGLE_COLUMN_VALUE：单列条件。

- FT_COMPOSITE_COLUMN_VALUE：多列组合条件。
- FT_COLUMN_PAGINATION：宽行读取条件。

枚举取值列表

```
enum FilterType {
  FT_SINGLE_COLUMN_VALUE = 1;
  FT_COMPOSITE_COLUMN_VALUE = 2;
  FT_COLUMN_PAGINATION = 3;
}
```

2.13. LogicalOperator

逻辑操作符，枚举类型。

- LO_NOT 表示非。
- LO_AND 表示并。
- LO_OR 表示或。

枚举取值列表

```
enum LogicalOperator {
  LO_NOT            = 1;
  LO_AND            = 2;
  LO_OR             = 3;
}
```

2.14. OperationType

在 UpdateRow 中，表示对一列的修改方式。

- PUT 表示插入一列或覆盖该列的数据。
- UPDATE 表示更新一列数据。
- DELETE 表示删除该列。

枚举取值列表

```
enum OperationType {
  PUT = 1;
  UPDATE = 2;
  DELETE = 3;
}
```

2.15. PartitionRange

分区的范围信息。

```
message PartitionRange {
  required bytes begin = 1; // encoded as SQLVariant
  required bytes end = 2; // encoded as SQLVariant
}
```

begin:

- 类型：bytes
- 描述：分区的起始键，分区的区间为前闭后开，包含起始键。Plainbuffer格式，编码详见PlainBuffer。

end:

- 类型：bytes
- 描述：分区的结束键，分区的区间为前闭后开，不包含结束键。Plainbuffer格式，编码详见PlainBuffer。

2.16. PlainBuffer

因为 Protocol Buffer 序列化和解析小对象的性能很差，所以表格存储自定义了 PlainBuffer 数据格式用来表示行数据。

格式定义

```
plainbuffer = tag_header row1  [row2]  [row3]
row = ( pk [attr] | [pk] attr | pk attr ) [tag_delete_marker] row_checksum;
pk = tag_pk cell_1 [cell_2] [cell_3]
attr  = tag_attr cell1 [cell_2] [cell_3]
cell = tag_cell cell_name [cell_value] [cell_op] [cell_ts] cell_checksum
cell_name = tag_cell_name  formated_value
cell_value = tag_cell_value formated_value
cell_op = tag_cell_op  cell_op_value
cell_ts = tag_cell_ts cell_ts_value
row_checksum = tag_row_checksum row_crc8
cell_checksum = tag_cell_checksum row_crc8
formated_value = value_type value_len value_data
value_type = int8
value_len = int32
cell_op_value = delete_all_version | delete_one_version
cell_ts_value = int64
delete_all_version = 0x01 (1byte)
delete_one_version = 0x03 (1byte)
```

Tag取值

```
tag_header = 0x75 (4byte)
tag_pk = 0x01 (1byte)
tag_attr = 0x02 (1byte)
tag_cell = 0x03 (1byte)
tag_cell_name = 0x04 (1byte)
tag_cell_value = 0x05 (1byte)
tag_cell_op = 0x06 (1byte)
tag_cell_ts = 0x07 (1byte)
tag_delete_marker = 0x08 (1byte)
tag_row_checksum = 0x09 (1byte)
tag_cell_checksum = 0x0A (1byte)
```

ValueType 取值

formated_value 中 value_type 的取值如下：

```
VT_INTEGER = 0x0
VT_DOUBLE = 0x1
VT_BOOLEAN = 0x2
VT_STRING = 0x3
VT_NULL = 0x6
VT_BLOB = 0x7
VT_INF_MIN = 0x9
VT_INF_MAX = 0xa
VT_AUTO_INCREMENT = 0xb
```

计算 Checksum

计算 checksum 的基本逻辑是：

- 针对每个 cell 的 name/value/type/timestamp 计算。
- 针对 row 里面的 delete 计算，其中有 delete mark 补单字节1；若没有，补单字节0。
- 因为对每个 cell 计算了 checksum，所以计算 row 的 checksum 的时候直接利用 cell 的checksum 来计算，即 row 的 crc 只对 cell 的 checksum 做 crc，不对数据做 crc。

Java 实现：

> **说明** 以下代码摘自 `$tablestore-4.2.1-sources/com/alicloud/openservices/tablestore/core/protocol/PlainBufferCrc8.java`，具体参见Java SDK。

```
public static byte getChecksum(byte crc, PlainBufferCell cell) throws IOException {
  if (cell.hasCellName()) {
    crc = crc8(crc, cell.getNameRawData());
  }
  if (cell.hasCellValue()) {
    if (cell.isPk()) {
      crc = cell.getPkCellValue().getChecksum(crc);
    } else {
      crc = cell.getCellValue().getChecksum(crc);
    }
  }
  if (cell.hasCellTimestamp()) {
    crc = crc8(crc, cell.getCellTimestamp());
  }
  if (cell.hasCellType()) {
    crc = crc8(crc, cell.getCellType());
  }
  return crc;
}
public static byte getChecksum(byte crc, PlainBufferRow row) throws IOException {
  for (PlainBufferCell cell : row.getPrimaryKey()) {
    crc = crc8(crc, cell.getChecksum());
  }
  for (PlainBufferCell cell : row.getCells()) {
    crc = crc8(crc, cell.getChecksum());
  }
  byte del = 0;
  if (row.hasDeleteMarker()) {
    del = (byte)0x1;
  }
  crc = crc8(crc, del);
  return crc;
}
```

举例

假设一行数据有两列主键，4 列数据。其中主键类型为 string 和 integer，3 列数据中分别是 string、integer 和 double，版本分别是 1001、1002 和 1003，还有一列是删除所有版本。

- 主键列：
  - [pk1:string:iampk]
  - [pk2:integer:100]

- 属性列：
  - [column1:string:bad:1001]
  - [column2:integer:128:1002]
  - [column3:double:34.2:1003]
  - [column4:del_all_versions]

编码：

```
<Header开始>[0x75]
<主键列开始>[0x1]
 <Cell1>[0x3][0x4][0x3][3][pk1][0x5][0x3][5][iampk][_cell_checksum]
 <Cell2>[0x3][0x4][0x3][3][pk2][0x5][0x0][8][100][_cell_checksum]
<属性列开始>[0x2]
 <Cell1>[0x3][0x4][0x3][7][column1][0x5][0x3][3][bad][0x7][1001][_cell_checksum]
 <Cell2>[0x3][0x4][0x3][7][column2][0x5][0x0][8][128][0x7][1002][_cell_checksum]
 <Cell3>[0x3][0x4][0x3][7][column3][0x5][0x1][8][34.2][0x7][1003][_cell_checksum]
 <Cell4>[0x3][0x4][0x3][7][column4][0x6][1][_cell_checksum]
[_row_check_sum]
```

2.17. PrimaryKeyOption

主键的属性值，目前仅支持AUTO_INCREMENT。

枚举取值列表

```
enum PrimaryKeyOption {
  AUTO_INCREMENT = 1;
}
```

2.18. RowInBatchGetRowResponse

在 BatchGetRow 操作的返回消息中，表示一行数据。

数据结构

```
message RowInBatchGetRowResponse {
  required bool is_ok = 1 [default = true];
  optional Error error = 2;
  optional ConsumedCapacity consumed = 3;
  optional bytes row = 4;
 optional bytes next_token = 5;
}
```

is_ok:

- 类型：bool
- 描述：该行操作是否成功。若为 true，则该行读取成功，error 无效；若为 false，则该行读取失败，row 无效。

error:

- 类型：Error
- 描述：该行操作的错误信息。

consumed:

- 类型：ConsumedCapacity
- 描述：该行操作消耗的服务能力单元。

row:

- 类型：bytes
- 读取到的数据，由Plainbuffer编码，详见Plainbuffer编码。
- 如果该行不存在，则数据为空。

next_token:

- 类型：bytes
- 宽行读取时，下一次读取的起始位置，暂不可用。

相关操作

BatchGetRow

2.19. PrimaryKeySchema

主键的属性值。

数据结构

```
message PrimaryKeySchema {
  required string name = 1;
  required PrimaryKeyType type = 2;
  optional PrimaryKeyOption option = 3;
}
```

name:

- 类型：string
- 描述：该列的列名。

type:

- 类型：PrimaryKeyType
- 描述：该列的类型。

option:

- 类型：PrimaryKeyOption
- 描述：该列的附加属性值。

2.20. ReservedThroughput

表示一个表设置的预留读/写吞吐量数值。

数据结构

```
message ReservedThroughput {
  required CapacityUnit capacity_unit = 1;
}
```

capacity_unit:

- 类型：CapacityUnit
- 描述：表当前的预留读/写吞吐量数值。

相关操作

CreateTable

UpdateRow

DescribeTable

2.21. ReservedThroughputDetails

表示一个表的预留读/写吞吐量信息。

数据结构

```
message ReservedThroughputDetails {
  required CapacityUnit capacity_unit = 1;
  required int64 last_increase_time = 2;
  optional int64 last_decrease_time = 3;
  required int32 number_of_decreases_today = 4;
}
```

capacity_unit:

- 类型：CapacityUnit
- 描述：该表的预留读写吞吐量的数值。

last_increase_time:

- 类型：int64
- 描述：最近一次上调该表的预留读/写吞吐量设置的时间，使用 UTC 秒数表示。

last_decrease_time:

- 类型：int64
- 描述：最近一次下调该表的预留读/写吞吐量设置的时间，使用 UTC 秒数表示。

number_of_decreases_today:

- 类型：int32
- 描述：本个自然日内已下调该表的预留读/写吞吐量设置的次数。

相关操作

UpdateTable

DescribeTable

2.22. ReturnContent

返回的数据内容。

数据结构

```
message ReturnContent {
  optional ReturnType return_type = 1;
}
```

return_type:

- 类型：ReturnType
- 描述：返回数据的类型。

2.23. ReturnType

返回数据的类型。

枚举数据类型

```
enum ReturnType {
  RT_NONE = 0;
  RT_PK = 1;
}
```

- RT_NONE：默认值，不返回任何值。
- RT_PK：返回主键列。

2.24. RowExistenceExpectation

行存在性判断条件，枚举类型。

- IGNORE 表示不做行存在性检查。

- EXPECT_EXIST 表示期待该行存在。
- EXPECT_NOT_EXIST 表示期待该行不存在。

枚举取值列表

```
enum RowExistenceExpectation {
  IGNORE = 0;
  EXPECT_EXIST = 1;
  EXPECT_NOT_EXIST = 2;
}
```

相关操作

PutRow

UpdateRow

DeleteRow

BatchWriteRow

2.25. RowInBatchWriteRowRequest

在 BatchWriteRow 操作中，表示要插入、更新和删除的一行信息。

数据结构

```
message RowInBatchWriteRowRequest {
  required OperationType type = 1;
  required bytes row_change = 2; // encoded as InplaceRowChangeSet
  required Condition condition = 3;
  optional ReturnContent return_content = 4;
}
```

type:

- 类型：OperationType
- 描述：操作类型。

row_change:

- 类型：bytes
- 描述：行数据，包括主键和属性列，由Plainbuffer编码，详见Plainbuffer编码。

condition:

- 类型：Condition
- 描述：条件更新的值，包括行条件检测和属性列检测。

return_content:

- 类型：ReturnContent

- 是否必要参数：否
- 写入成功后返回的数据类型，目前仅支持返回主键，主要用于主键列自增功能中。

相关操作

BatchWriteRow

2.26. RowInBatchWriteRowResponse

在 BatchWriteRow 操作的返回消息中，表示一行写入操作的结果。

数据结构

```
message RowInBatchWriteRowResponse {
  required bool is_ok = 1 [default = true];
  optional Error error = 2;
  optional ConsumedCapacity consumed = 3;
}
```

is_ok:

- 类型：bool
- 描述：该行操作是否成功。若为 true，则该行写入成功，error 无效；若为 false，则该行写入失败。

error:

- 类型：Error
- 描述：该行操作的错误信息。

consumed:

- 类型：ConsumedCapacity
- 描述：该行操作消耗的服务能力单元。

相关操作

BatchWriteRow

2.27. SingleColumnValueFilter

单个条件，比如 column_a > 5 等。适用于 ConditionUpdate 和 Filter 功能。

数据结构

```
message SingleColumnValueFilter {
  required ComparatorType comparator = 1;
  required string column_name = 2;
  required bytes column_value = 3;
  required bool filter_if_missing = 4;
  required bool latest_version_only = 5;
}
```

comparator:

- 类型：ComparatorType
- 描述：比较类型。

column_name:

- 类型：string
- 描述：列名称。

column_value:

- 类型：bytes
- 描述：列值经过Plainbuffer编码后的值。

filter_if_missing:

- 类型：bool
- 描述：当某行的这一列不存在时，设置条件是否过滤。比如条件是 column_a>0，filter_if_missing 是 true，当某一行没有列 column_a 时，这一行的条件判断就会通过。

latest_version_only:

- 类型：bool
- 描述：是否只对最新版本有效。如果为true，则表示只检测最新版本的值是否满足条件；如果是false，则会检测所有版本的值是否满足条件。

相关操作

- ConditionUpdate

  PutRow

  UpdateRow

  DeleteRow

  BatchWriteRow

- Filter

  GetRow

  GetRange

  BatchGetRow

2.28. StreamDetails

表示一个表的stream信息。

数据结构

```
message StreamDetails {
  required bool enable_stream = 1;
  optional string stream_id = 2;
  optional int32 expiration_time = 3;
  optional int64 last_enable_time = 4;
}
```

enable_stream:

- 类型：required bool
- 描述：该表是否打开stream

stream_id:

- 类型：optional string
- 描述：该表的stream的id

expiration_time:

- 类型：optional int32
- 描述：该表的stream的过期时间

last_enable_time:

- 类型：optional int64
- 描述：该stream的打开的时间

相关操作

DescribeTable

2.29. StreamRecord

在 GetStreamRecord 的响应消息中，表示一行数据。

数据结构

```
message StreamRecord {
    required ActionType action_type = 1;
    required bytes record = 2;
}
```

action_type:

- 类型：required ActionType
- 描述：表示该行的操作类型。

record:

- 类型：required bytes
- 描述：表示该行的数据内容。

相关操作

GetStreamRecord

2.30. StreamSpecification

表示一个表的stream信息。

数据结构

```
message StreamSpecification {
  required bool enable_stream = 1;
  optional int32 expiration_time = 2;
}
```

enable_stream:

- 类型：bool
- 描述：该表是否打开stream。

expiration_time:

- 类型：int32
- 描述：该表的stream过期时间。

相关操作

CreateTable

DescribeTable

UpdateTable

2.31. TableInBatchGetRowRequest

在 BatchGetRow 操作中，表示要读取的一个表的请求信息。

数据结构

```
message TableInBatchGetRowRequest {
  required string table_name = 1;
  repeated bytes primary_key = 2; //Plainbuffer编码
  repeated bytes token = 3;
  repeated string columns_to_get = 4; // 不指定则读出所有的列
  optional TimeRange time_range = 5;
  optional int32 max_versions = 6;
  optional bool cache_blocks = 7 [default = true]; // 本次读出的数据是否进入BlockCache
  optional bytes filter = 8;
  optional string start_column = 9;
  optional string end_column = 10;
}
```

table_name:

- 类型：string
- 描述：该表的表名。

primary_key:

- 类型：repeated bytes
- 是否必要参数：是
- 该行全部的主键列，包含主键名和主键值，由Plainbuffer编码，详见Plainbuffer编码。

token:

- 类型：repeated bytes
- 是否必要参数：否
- 宽行读取时指定下一次读取的起始位置，暂不可用。

columns_to_get:

- 类型：repeated string
- 描述：该表中需要返回的全部列的列名。

time_range:

- 类型：TimeRange
- 是否必要参数：和max_versions必须至少存在一个。
- 读取数据的版本时间戳范围。
- 时间戳的单位是毫秒，取值最小值为0，最大值为INT64.MAX。
- 若要查询一个范围，则指定start_time和end_time。
- 若要查询一个特定时间戳，则指定specific_time。
- 例子：如果指定的time_range为(100, 200)，则返回的列数据的时间戳必须位于[100, 200)范围内，前闭后开区间。

max_versions:

- 类型：int32
- 是否必要参数：和time_range必须至少存在一个。
- 读取数据时，返回的最多版本个数。
- 例子：如果指定max_versions为2，则每一列最多返回2个版本的数据。

cache_blocks:

- 类型：bool
- 是否必要参数：否
- 本次读出的数据是否进入BlockCache。
- 默认值：true
- 当前暂不支持设置为false。

filter:

- 类型：bytes
- 是否必要参数：否
- 过滤条件表达式。
- Filter经过protobuf序列化后的二进制数据。

start_column:

- 类型：string
- 是否必要参数：否
- 指定读取时的起始列，主要用于宽行读。
- 返回的结果中包含当前起始列。
- 列的顺序按照列名的字典序排序。
- 例子：如果一张表有"a"，"b"，"c"三列，读取时指定start_column为“b”，则会从"b"列开始读，返回"b"，"c"两列。

end_column:

- 类型：string
- 是否必要参数：否
- 指定读取时的结束列，主要用于宽行读。
- 返回的结果中不包含当前结束列。
- 列的顺序按照列名的字典序排序。
- 例子：如果一张表有"a"，"b"，"c"三列，读取时指定end_column为“b”，则读到"b"列时会结束，返回"a"列。

相关操作

BatchGetRow

2.32. TableInBatchGetRowResponse

在 BatchGetRow 操作的返回消息中，表示一个表的数据。

数据结构

```
message TableInBatchGetRowResponse {
  required string table_name = 1;
  repeated RowInBatchGetRowResponse rows = 2;
}
```

table_name:

- 类型：string
- 描述：该表的表名。

rows:

- 类型：repeated RowInBatchGetRowResponse
- 描述：该表中读取到的全部行数据。

相关操作

BatchGetRow

2.33. TableInBatchWriteRowRequest

在 BatchWriteRow 操作中，表示要写入的一个表的请求信息。

数据结构

```
message TableInBatchWriteRowRequest {
  required string table_name = 1;
  repeated RowInBatchWriteRowRequest rows = 2;
}
```

table_name:

- 类型：string
- 描述：该表的表名。

rows:

- 类型：repeated RowInBatchWriteRowRequest
- 描述：该表中请求插入、更新和删除的行信息。

相关操作

BatchWriteRow

2.34. TableInBatchWriteRowResponse

在 BatchWriteRow 操作中，表示对一个表进行写入的结果。

数据结构

```
message TableInBatchWriteRowResponse {
  required string table_name = 1;
  repeated RowInBatchWriteRowResponse put_rows = 2;
  repeated RowInBatchWriteRowResponse update_rows = 3;
  repeated RowInBatchWriteRowResponse delete_rows = 4;
}
```

table_name:

- 类型：string
- 描述：该表的表名。

put_rows:

- 类型：RowInBatchWriteRowResponse
- 描述：该表中 PutRow 操作的结果。

update_rows:

- 类型：RowInBatchWriteRowResponse
- 描述：该表中 UpdateRow 操作的结果。

delete_rows:

- 类型：RowInBatchWriteRowResponse
- 描述：该表中 DeleteRow 操作的结果。

相关操作

BatchWriteRow

2.35. TableMeta

表示一个表的结构信息。

数据结构

```
message TableMeta {
  required string table_name = 1;
  repeated PrimaryKeySchema primary_key = 2;
}
```

table_name:

- 类型：string
- 描述：该表的表名。

primary_key:

- 类型：repeated PrimaryKeySchema

- 描述：该表全部的主键列。

相关操作

CreateTable

DescribeTable

2.36. TableOptions

表的参数值，包括TimeToLive，最大版本数等。

数据结构

```
message TableOptions {
  optional int32 time_to_live = 1; // 可以动态更改
  optional int32 max_versions = 2; // 可以动态更改
  optional int64 deviation_cell_version_in_sec = 5; // 可以动态修改
}
```

time_to_live:

- 类型：int32
- 描述：本张表中保存的数据的存活时间，单位秒。

max_versions:

- 类型：int32
- 描述：本张表保留的最大版本数。

deviation_cell_version_in_sec:

- 类型：int64
- 描述：最大版本偏差。目的主要是为了禁止写入与预期较大的数据，比如设置deviation_cell_version_in_sec为1000，当前timestamp如果为10000，那么允许写入的timestamp范围为[10000 - 1000, 10000 + 1000]。

2.37. TimeRange

查询数据时指定的时间戳范围或特定时间戳值。

数据结构

```
message TimeRange {
  optional int64 start_time = 1;
  optional int64 end_time = 2;
  optional int64 specific_time = 3;
}
```

start_time:

- 类型：int64
- 描述：起始时间戳。单位是毫秒。时间戳的取值最小值为0，最大值为INT64.MAX。

end_time:

- 类型：int64
- 描述：结束时间戳。单位是毫秒。时间戳的取值最小值为0，最大值为INT64.MAX。

specific_time:

- 类型：int64
- 描述：特定的时间戳值。specific_time和[start_time, end_time) 两个中设置一个即可。单位是毫秒。时间戳的取值最小值为0，最大值为INT64.MAX。

> **说明** [start_time, end_time) 表示大于等于start_time，小于end_time。