表格存储

常见问题

常见问题

一般性问题

表格存储是数据库吗？和传统数据库（比如mysql、SQL Server）有什么区别？

表格存储是 NoSQL 的数据存储服务，是基于云计算技术构建的一个分布式结构化和半结构化数据的存储和管理服务，与传统关系型数据库软件（RDBMS，例如 mysql、SQL Server）在数据模型和技术实现上都有较大的区别。

表格存储的数据模型以二维表为中心。表有行和列的概念，但是与传统数据库不一样，表格存储的表是稀疏的，每一行可以有不同的列，可以动态增加或者减少属性列，建表时不需要为表的属性列定义严格的 schema。

相对于传统数据库的丰富功能（视图、索引、事务、丰富的 SQL 语句支持），表格存储提供较为基础的功能，但是具有更好的规模扩展性，能够较容易地支持更大的数据规模（百 TB 级别）和并发访问（单表 10 万 QPS）。

在编程方面，表格存储提供统一的 HTTP Restful API，不支持传统的 SQL 语句标准。用户使用表格存储要按照实际占用的资源（存储和读/写吞吐量）进行付费。

如果问题还未能解决，请联系售后技术支持。

表格存储Table Store-建表时的注意事项

建表时需要指定属性列吗？

不需要。Table Store支持半结构化的表，即建表时只需要指定主键列（1至4列），不需要在创建表的时候指定

属性列。

Table Store表中包含的属性列个数无限制，且每一行数据可以拥有不同数量不同类型的属性列。在应用程序写入数据时，Table Store需要应用程序指定数据所有列（主键列及属性列）的列名和列值。

建表时主键（Primary Key）的第一列为分区键（Partition Key）怎么理解？

主键的第一列为分区键，可以理解为当表的数据量达到一个设定值时，Table Store会根据分区键列值的范围来进行分区的操作，通过分区来达到数据访问负载均衡的目的。

建表时，表内的数据默认拥有一个分区，即该表的所有数据在一个数据分区上。当表拥有多个分区时，每个分区所存储的数据对应的是该表分区键列值某个范围内所有的数据。所有的分区键列值范围是按照其列值自然序切分的，即按照Integer或String（主键列数据类型）的自然序切分。

除了会影响到数据访问的性能，数据的分区也会影响到您预留读/写吞吐量的使用率。当表拥有多个分区时，该表的预留读/写吞吐量会按照一定的比例预分配到各个分区上。

如何设定一个好的分区键？

建表时，分区键的选择是很重要的，会影响当表数据量很大时访问的性能。应用程序在选择分区键时，应该遵循以下基本原则：

不要使用拥有固定值或取值范围较小的属性，如客户性别（Male/Female）。

尽量避免使用按自然序排序后会有明显访问热点的属性，如在查询最新数据场景中使用时间戳（TimeStamp）作为分区键。

尽量使用按自然序排序后访问热点比较分散的属性，如UserID。

如果无法预估访问热点，该怎么做？

建议在写入分区键之前，根据应用程序的特点进行一次哈希（Hash）。如在写入一行数据时，将UserID通过简单的哈希算法生成一个哈希值，然后在哈希值后拼接UserID作为分区键的值存入Table Store的表。通过这种轻量级的操作可以有效地解决部分访问热点问题。但是需要特别注意的是，由于分区键的值是由哈希值和实际值拼接的，应用程序将无法使用分区键进行范围读取的操作（getRange）。

为什么一个账户下会有表个数的限制？

每个Table Store用户可以创建10个实例，每个实例可以创建64个表，即，Table Store限制在一个账户下最多可以创建640个表。

放开表个数的需求一般有以下几种情况：

数据量大、访问性能要求高

不同于传统的SQL数据库（如mySQL） 解决海量数据访问需求的方法是分库分表，Table Store作为分布式实现方式很好地解决了数据量及访问延迟的瓶颈。

您可以将结构化或半结构化的数据存在一张稀疏的大表中，不用担忧数据量过大后的访问的性能问题。

应用的快速增长

除了数据本身及访问量的增长，您可能使用Table Store为您的客户（如第三方伙伴、供应商等）提供服务。以为供应商提供服务为例，您有了一套基于Table Store的解决方案后，每加一个供应商就部署一组Table Store的表。这样，表的个数很快达到上限。如果您不断提高表个数的上限，会造成运维成本的不可控，也增加了后续全局数据分析的难度。

建议在使用Table Store时打破传统思想，使用大表的概念将同类型海量结构化及半结构化数据存在一张表上。

Table Store服务本身的考虑

基于Table Store分布式的实现，表的个数也成为了Table Store本身的一个资源属性。可以理解为在Table Store集群规模一定的情况下，表的个数是有一个最大值的。当然，Table Store的扩展能力可以有效地解决表个数的限制，但从Table Store服务本身资源可控性角度来看，Table Store设定了单个账户表个数的限制。

如果您仍然需要提高一个帐号下表个数的限制，请提交工单。

Table Store支持的数据类型

Table Store支持的数据类型包括INTEGER、STRING、DOUBLE和BOOLEAN。

说明：

- 无论是表的主键列，还是视图的主键列，都不能使用DOUBLE类型。
- 数据分区键，即主键的第一列不能使用DOUBLE和BOOLEAN类型。

INTEGER表示64位带符号整形，例如：23，-908，+87

范围从 -9,223,372,036,854,775,808 到 +9,223,372,036,854,775,807。

STRING表示UTF-8编码的字符串，传入时需加单引号（’ ），例如：’ Test’ 、’ Open Table Service’ 、’ Table Store用户指南’ 。字符串长度为0到512K字节，建议不超过1024字节。

BOOLEAN表示布尔类型，只能有2个值：TRUE和FALSE。

DOUBLE表示浮点数，采用IEEE 754标准的64位表示法，例如：1.23，-34.0，+102.23，1.0e-10，2.006E+3

表和实例的命名规范

表名称

表、表组、视图、列的命名规范如下：

只能由英文字符、数字和下划线（_）组成

大小写敏感

数字不能作为名称的首字符

名称长度为1−255个字符

实例名称

实例的名称在一个地域内必须唯一，不同的地域内实例名称可以相同。具体命名规范如下：

必须由英文字母、数字或连字符（-）组成

首字符必须为英文字母

末尾字符不能为连字符（-）

大小写不敏感

名称长度为3-16Byte

如何理解主键、数据分区和数据分区键

主键

表中的每一行由主键（PK）唯一确定。您在创建表的时候必须指定组成主键的列，这些列称为主键列。主键列必须有值。您必须确保主键列的值的组合能够唯一地确定一行。在后续使用的过程中，主键列的类型不能改变。

数据分区和分区键

Table Store会自动把表分成不同的数据分区，以达到对其存储数据的负载均衡。数据分区的划分粒度为主键的第一列，该列即为数据分区键。

拥有相同数据分区键的行必然在同一个数据分区中。Table Store能够保证对具有同一数据分区键的数据进行更改操作的一致性。

下图是一个电子邮件系统的邮件表的一部分。该表的主键和分区键信息如下：

列UserID、ReceiveTime、FromAddr分别表示邮件用户的ID、接收时间、发送人，这些列为主键列，唯一确定一封邮件。其中UserID列为数据分区键。

列ToAddr、MailSize、Subject、Read分别表示收件人、邮件大小、邮件主题和邮件是否已读，这些为普通的列，存储邮件的相关信息。

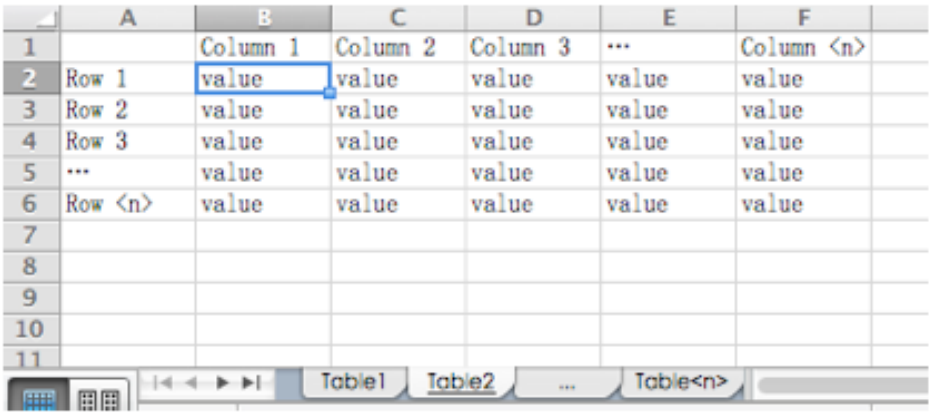
图中Table Store把UserID为U0001和U0002的用户信息划在一个数据分区中，而把UserID为U0003和U0004的用户信息划分在另一个数据分区中。

UserID	ReceiveTime	FromAddr	ToAddr	MailSize	Subject	Read
U0001	1998-1-1	eric@demo.com	bob@demo.com	10000	Hello	Y
U0001	2011-10-20	alice@demo.com	bob@demo.com; vivian@demo.com	15000	Fw: Greetings	Y
U0001	2011-10-21	alice@demo.com	team1@demo.com	8900	Review	N
U0001	2011-10-24	lucy@demo.com	vteam@demo.com	500	Meeting Request	N
U0001	2011-11-9	alice@demo.com	bob@demo.com; team@demo.com; robin@demo.com	1250	Re: Review	N
U0001	2011-11-11	alice@demo.com	team@demo.com	3300	Re: Review	Y
U0002	1999-12-1	bill@demo.com	tom@demo.com; windy@demo.co; bill@demo.com	4300	Re: Hello	Y
U0002	2010-3-18	windy@demo.com	tom@demo.com	500	Meeting Request	Y
U0003	2010-11-11	robin@demo.com	vteam@demo.com	21000	Have a nice day	Y
U0003	2010-12-10	bob@demo.com	vteam@demo.co; alice@demo.com	10000	Re: help	N
U0004	1999-6-29	vivian@demo.com	vivian@demo.com	50	Test	N

数据分片键	主键	列	数据分片

Tablestore中表、行、列、值和电子表格的类比

Tablestore的表存储着用户的结构化数据。用户可以在表中查询、插入、修改和删除数据。一个用户可以拥有多个表。数据在表中以行、列、值的形式来组织。



上图展示了Tablestore中表及其它概念与电子表格的类比：

表：类似电子表格中底端的标签，不同的标签对应到不同的表。

行：类似电子表格中的行。每一行包含一组由列到值的组。

列：类似电子表格中的列。位于同一列的数据具有相同的类别属性。

值：类似电子表格中的单元格。表示某一行在特定列上的值。与电子表格不同的是，ablestore允许某些列没有值，如果某些列没有值，则不占用存储空间。值的类型可以为STRING、INTEGER、BOOLEAN、DOUBLE、BINARY，如果该列为主键列，则值的类型只能是STRING、INTEGER或BINARY。

用户验证

Table Store通过对称签名的方法来验证某个请求是其拥有者发送，以及应答是Table Store所发送。

当用户在阿里云注册账号之后，可以在AccessKey控制台上获取AccessKeyID和AccessKeySecret。AccessKeyID用来标识用户，AccessKeySecret是用来对请求和响应进行签名和验证的密钥。AccessKeySecret需要保密。

用户以个人身份向Table Store发送请求时，需要包括：请求明文、AccessKeyID、和使用AccessKeySecret对请求明文中的信息部分签名产生的验证码。

Table Store收到请求后，会通过AccessKeyID找到相应的AccessKeySecret，以同样的方式签名明文中的信息。如果计算出来的验证码和提供的验证码相同，则认为是有效的用户发送的请求。

验证Table Store的应答时需要用户使用相同的方式进行计算，若计算的验证码和提供的验证码一致，则用户可以认为应答是有效的Table Store应答。

如何获取AccessKeyId和AccessKeySecret

访问密钥AccessKey（AK）相当于登录密码，只是使用场景不同。AccessKey用于程序方式调用云服务API，而登录密码用于登录控制台。如果您不需要调用API，那么就不需要创建AccessKey。

操作步骤

云账号登录RAM控制台。

在左侧导航栏的**人员管理**菜单下，单击**用户**。

在**用户登录名称/显示名称**列表下，单击目标RAM用户名称。

在**用户AccessKey**区域下，单击**创建新的AccessKey**。

首次创建时需填写手机验证码。

单击**确认**。

- AccessKeySecret只在创建时显示，不提供查询，请妥善保管。
- 若AccessKey泄露或丢失，则需要创建新的AccessKey，最多可以创建2个AccessKey。

AK权限控制

Table Store目前仅支持通过AccessKeyID和AccessKeySecret对用户身份进行认证，认证通过的用户对Table Store内的资源访问不受限。

一个用户的帐户下可以有多组不同的AccessKeyID和AccessKeySecret对，同一个帐户使用不同AccessKeyID和AccessKeySecret看到的Table Store资源是相同的。

如果您使用子账号，则子帐号需要被主账号授予权限，且子帐号只能访问被授权的资源。有关子帐号访问的详细内容，请参见授权管理。

存储数据量对查询速度有影响吗？

对于单行查询和范围查询，查询的速度不在于数据量有多少。

表格存储作为NoSQL数据库，其数据量可以随集群的规模线性扩展，并且对单行和范围查询的速度不会有任何

影响。即使数据规模达到亿级或者百亿级，查询速度都不会变。

在高性能实例（底层是SSD）上，单行查询的速度是毫秒级别，如果单行数据量比较小，查询速度一般在10毫秒以内。

关于查询API的详细信息，请参考GetRow、GetRange和BatchGetRow。

两表关联怎么查询

表格存储不支持两表关联查询，但您可以将表格存储连通DLA服务，然后使用通用的SQL语言（兼容MySQL5.7绝大部分查询语法）对表格存储进行灵活的数据分析。具体参见DLA文档。

如何突破批量写200条的限制

您可以使用TableStoreWriter异步接口进行批量写入，详情参见使用TableStoreWriter进行高并发、高吞吐的数据写入。

如何查看表的数据存储量

您可以通过以下两个方式在表格存储管理控制台查看表的数据存储量：

- 登录表格存储管理控制台，进入数据表详情页，查看**表格大小**。

- 登录表格存储管理控制台，进入数据表详情页。在左侧导航栏，单击**数据监控**，选择**表大小**查看数据存储量。

报错OTSErrorMsg: Disallow read index table in building base state

问题现象

读正在构造存量数据的索引表时出现如下报错。

OTSErrorMsg: Disallow read index table in building base state

问题分析

全局二级索引的存量构造需要对表中的存量数据进行读取，然后同步到索引表。在存量数据同步完成之前，不允许读索引表，当存量数据同步完成，可以正常读取索引表。存量数据同步时间与主表的数据量大小相关。

如何读取全表数据

您可以使用范围读（GetRange）接口读取全表数据。

GetRange接口用于读取一个范围内的数据。表格存储表中的行都是按照主键排序的，主键是由全部的主键列按照顺序组成的。GetRange接口支持按照指定的范围进行正序和反序读取，如需读取表中全部数据，可以通过设置主键的最小值（PrimaryKeyValue.INF_MIN）和最大值（PrimaryKeyValue.INF_MAX）进行读取，通过判断NextStartPrimaryKey是否为null，读取全部数据。

示例代码如下：

```
private static void getRange(SyncClient client){
RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(TABLE_NAME);

//设置起始主键
PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1,PrimaryKeyValue.INF_MIN);
primaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2,PrimaryKeyValue.INF_MIN);
rangeRowQueryCriteria.setInclusiveStartPrimaryKey(primaryKeyBuilder.build());

//设置结束主键
primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1,PrimaryKeyValue.INF_MAX);
primaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2,PrimaryKeyValue.INF_MAX);
rangeRowQueryCriteria.setExclusiveEndPrimaryKey(primaryKeyBuilder.build());

rangeRowQueryCriteria.setMaxVersions(1);
System.out.println("GetRange的结果为:");
while(true){
GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQueryCriteria));
for (Row row : getRangeResponse.getRows()){
System.out.println(row);
}

//若nextStartPrimaryKey不为null，则继续读取
if (getRangeResponse.getNextStartPrimaryKey() != null){
rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextStartPrimaryKey());
}else{
break;
```

```
}

}
}
```

表格存储是否支持类似关系数据库的in和between...and查询

您可以根据业务需求，选择不同的方法进行类似in的查询：

- 如果是能确定完整主键的in操作，可以使用BatchGetRow接口。
- 如果是不能确定完整主键列或者基于属性列的in操作，可以使用多元索引的TermsQuery接口。

如需类似between...and的查询，您可以：

- 如果是能确定完整主键列的操作，使用GetRange接口。
- 如果是不能确定完整主键列或者基于属性列的操作，使用多元索引的RangeQuery接口。

如何查看表的总行数

您可以通过查询表或多元索引的方法获取表格存储中某个表的总行数：

调用表格存储的GetRange API：

调用GetRange API后对表中的行数进行计数，由于并发及性能较低，因此计数较慢。

使用表格存储的SQL分析引擎：

计数是表格存储SQL 分析引擎的一项基本功能，SQL可以让您更方便地对表格存储中的数据进行统计分析，减少多个数据源之间相互同步的麻烦并节省成本。具体方法是开通SQL分析后，使用 select count(*) from table 即可。在执行SQL语句时，分析引擎会启动多个任务并发处理，所以计数较快。典型场景下（受行大小和表设计模式是否合理等影响），1000万行数据计数时间为10秒。

使用表格存储多元索引：

- 使用多元索引的MatchAllQuery接口，毫秒级返回表中总行数。

- 使用多元索引的统计聚合接口（Count），毫秒级返回表中总行数。

如果没有多元索引，需要先创建多元索引。

计量计费

预留读/写吞吐量

预留读/写吞吐量是表的一项属性。系统会在后台根据表的预留读/写吞吐量配置预留资源，保证您对该表的吞吐量需求。

创建表（CreateTable）时需要指定表的预留读/写吞吐量。在表创建成功后，还可以使用 UpdateTable 操作更新表的预留读/写吞吐量配置。

单表的预留读/写吞吐量均可以设置为 0 或者是大于 0 的值，默认不超过 5000（读和写分别不超过 5000）。如果用户有单表预留读/写吞吐量需要超出 5000 的需求，可以通过人工服务提高预留读/写吞吐量。当预留读吞吐量或者预留写吞吐量不为 0 时，无论是否有读/写请求，均会产生费用。

预留读/写吞吐量的计量单位为写服务能力单元和读服务能力单元，应用程序通过 API 进行表格存储读/写操作时均会消耗对应的写服务能力单元和读服务能力单元。

表格存储对实例中所有表的预留读/写吞吐量之和按小时计费。用户配置的预留读/写吞吐量可能会动态变化，表格存储以固定的时间间隔统计表的预留读/写吞吐量、计算每个小时的预留读/写吞吐量的平均值，将平均值乘以单价进行计费。预留读/写吞吐量单价可能发生变化，请参见阿里云官网信息。

如果问题还未能解决，请联系售后技术支持。

表格存储的数据存储

表格存储对实例的数据总量按小时计费。由于用户的数据总量会动态变化，因此表格存储以固定的时间间隔统计表的数据总量大小，计算每小时数据总量的平均值，将平均值乘以单价进行计费。单价可能发生变化，请参见阿里云官网信息。

实例中所有表的数据大小之和是该实例的数据总量，表的数据大小是表中的所有行数据大小之和，下面举例说

明表的数据大小的计算。

假设存在下表，id 是主键列，其他均为属性列：

id	name	length	comments
Integer(1)	String(10byte)	Integer	String(32Byte)
Integer(2)	String(20byte)	Integer	String(999Byte)
Integer(3)	String(43byte)	Integer	空

对于 id=1 的行，其数据大小为：len（‘id’）+ len（‘name’）+ len（‘length’）+ len（‘comments’）= 8 Byte + 10 Byte + 8 Byte + 32 Byte = 78 Byte。

对于 id=2 的行，其数据大小为：len（‘id’）+ len（‘name’）+ len（‘length’）+ len（‘comments’）= 8 Byte + 20 Byte + 8 Byte+ 999 Byte = 1055 Byte。

对于 id=3 的行，其数据大小为：len（‘id’）+ len（‘name’）+ len（‘length’）= 8 Byte + 43 Byte + 8 Byte = 71 Byte。

因此，表的数据大小之和为： 78 + 1055 + 71 =1204 Byte。

假设 1 小时内表的数据大小之未发生变化，将会按 1204 Byte 进行计费。表格存储对单表数据大小没有限制，用户可以根据自己的实际需求使用，按需付费。

如果问题还未能解决，请联系**售后技术支持**。

表格存储收费方式详解

表格存储是后付费产品，会根据一个账单周期内各个资源的实际使用量进行计量计费，所以不需要额外购买实例费用。

现在表格存储一共有 4 个维度的计量项：

数据存储量

预留读/写吞吐量

按量读/写吞吐量

外网下行流量

表格存储根据账单周期（小时）中的实际使用量来进行计量计费。

详解一：计算一个小时的使用费用

场景

在一个计费周期中，数据量为 50 GB，外网下行流量为 10 GB，在这个计费周期的一个小时内，在第 20 分钟的时候该实例下的所有表的预留读/写吞吐量从（1000,1500）调整到了（1200,800），这一个计费周期共使用了 50000 的按量读 CU 和 10000 的按量写 CU。

计费公式

该实例这一个小时的计费公式为：

存储费用： 50 GB * 每 GB 每小时单价。

流量费用： 10 GB * 外网下行流量单价。

预留 CU 费用：

这一个小时的平均预留读 CU 为：(1000 * 20 + 1200 * 40) / 60 = 1133.3

这一个小时的平均预留写 CU 为：(1500 * 20 + 800 * 40) / 60 = 1033

这一个小时的费用为： 1133.3 * 每小时读 CU 单价 + 1033.3 * 每小时写 CU 单价。

按量 CU 费用：50000/10000 * 每万按量读 CU 单价 + 10000/10000 * 每万按量写 CU 单价。

总费用为上述 4 项费用之和。

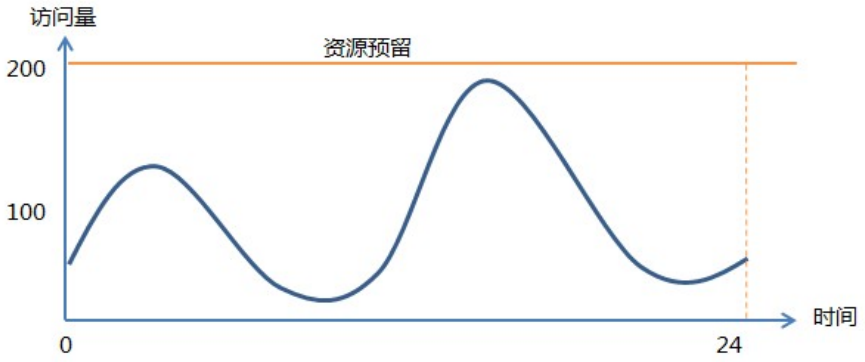
数据存储量和预留读/写吞吐量都是精确到分钟级别，在计费周期结束的时候会对这个计费周期中的数据存储量和预留读/写吞吐量取平均值，来作为这个计费周期该资源的实际使用量。按量读/写吞吐量是精确到秒级别，统计此次计费周期中每秒使用的按量 CU，再做一次聚合。

假如在前二十分钟中，预留读 CU 设置是 1000，在某一秒消耗了 2100 读 CU，那么这一秒按量读吞吐量就是 (2100 - 1000) = 1100。

详解二：计算一天的使用费用

场景 1

目前按传统方式购买资源的计费方式如下。



上图模拟了一个应用程序一天的访问情况，为方便说明，假设这个应用程序的读/写访问情况是一致的。那么该应用程序为了保证在波峰的时候能够有足够的资源来提供读/写服务，用户就需要按波峰的业务量进行购买资源，假如换算成表格存储的服务能力单位，就是购买 200 CU 的预留读/写吞吐量。

计费公式

该用户一天的费用就是： 200 * 每小时预留读 CU 单价 *24 + 200 * 每小时预留写 CU 单价 * 24 + 24 小时的数据存储费用 + 24 小时的外网下行流量费用。

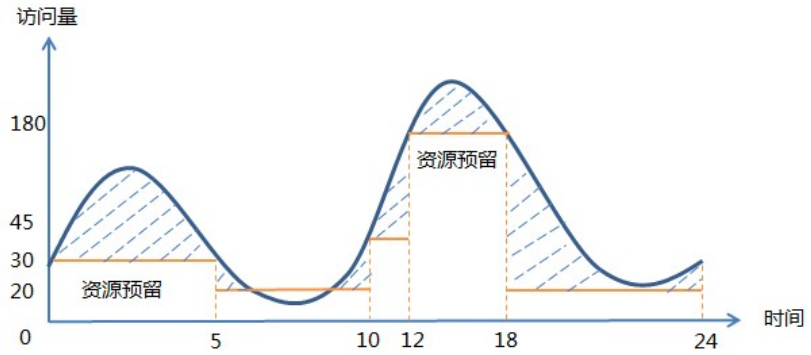
即：4800 * 每小时预留读 CU 单价 + 4800 * 每小时预留写 CU 单价 + 24 小时的数据存储费用 + 24 小时的外网下行流量费用。

表格存储提供了调整表预留读/写吞吐量的 API，**用户可以随时调整每张表的预留读/写吞吐量大小**，调整之后会在一分钟内生效，更关键但是调整表的预留吞吐量不会影响对业务造成任何影响。

基于这种方式，用户可以在波峰的时候调大以适应业务增长需要，波谷的时候再降低预留 CU 以节省费用。

场景 2

假如用户当天使用表格存储的方式为：



24 个小时的周期中：

0~5 点：设置读/写 CU 为 30。5 个小时中，读、写各消耗的超出预留部分的 CU 为 100000 个。

5~10 点：业务访问下降，调整读/写 CU 为 20。5 个小时中，读、写各消耗的超出预留部分的 CU 为 5000 个。

10~12 点：业务访问开始上升，调整读/写 CU 到 45。2 个小时中，读、写各消耗的超出预留部分的 CU 为 10000 个。

12~18 点：业务高峰到来，将读/写 CU 提高到 180。6 个小时中，读、写各消耗的超出预留部分的 CU 为 30000 个。

18~24 点：访问高峰已经过去，将读/写 CU 下调到 20。6 个小时中，读、写各消耗的超出预留部分的 CU 为 50000 个。

计费公式

为方便计算，我们假设读写比例为 1:1，读/写预留 CU 调整方式一样，那么这一整天在 CU 上面的费用就是：

读 CU 费用：

(30 * 5 小时 + 20 * 5 小时 + 45 * 2 小时 + 180 * 6 小时 + 20 * 6 小时) * 每小时预留读 CU 单价 + (100000 + 5000+ 10000 + 30000 + 50000) * 按量读 CU 单价

即 1540 * 每小时预留读 CU 单价 + 195000 * 按量读 CU 单价

写 CU 费用：

(30 * 5 小时 + 20 * 5 小时 + 45 * 2 小时 + 180 * 6 小时 + 20 * 6 小时) * 每小时预留写 CU 单价 + (100000 + 5000 + 10000 + 30000 + 50000) * 按量写 CU 单价

即 1540 * 每小时预留写 CU 单价 + 195000 * 按量写 CU 单价

相比**场景1**中传统购买资源的方式，一天中节省费用为：

4800 * 每小时预留读 CU 单价 + 4800 * 每小时预留写 CU 单价 - 1540 * 每小时预留读 CU 单价 - 19.5 * 每万按量读 CU 单价 - 1540 * 每小时预留写 CU 单价 - 19.5 * 每万按量写 CU 单价

注意事项

上述费用计算均不包括到 2019 年 12 月份之前每用户每自然月的**免费额度**。

按量读/写吞吐量的单价略贵于预留吞吐量的单价，所以建议用户根据业务动态合理的调整预留值，这样能够有效的降低成本。

可以用过 SDK 将表的预留读/写吞吐量设置在比较低的水平，这样就会优先使用到免费额度。

API/SDK

Java SDK报错：Invalid date format

现象

执行环境：Java 8

使用表格存储Java SDK时抛出以下异常：

[Error Code]:OTSParameterInvalid, [Message]:Invalid date format: Wed, 18 May 2016 08:32:51 +00:00.

原因

Classpath中依赖的Joda-time版本过低，joda-time的低版本在Java 8上会出现此类错误。

解决方案

可以更新到ots-public的最新版本2.2.4来解决这个问题。如果您也依赖了joda-time库，需要提升到2.9。

Java SDK报错：SocketTimeoutException

现象

当“SDK接收到数据的时间”减去“SDK发送数据的时间”超过SocketTimeout时，SDK会抛出

SocketTimeoutException。这段时间内包含了“应用发送请求（包含网络传输）”、“服务端处理”和“应用接收响应（包含网络传输）”。SocketTimeout可以在创建OTSClient时自定义，如果没有设置，默认是15s。

解决方案

出现SocketTimeoutException，可能是以下几种原因：

网络不通

如果全部请求都是SocketTimeoutException，那么首先可能是网络不通，可以通过ping或者curl命令测试是否为网络问题：

```
ping aaaa.cn-hangzhou.ots.aliyuncs.com
curl aaaa.cn-hangzhou.ots.aliyuncs.com
```

正常情况下，curl会返回类似下面的结果：

```
<?xml version="1.0" encoding="UTF-8"?>
<Error><Code>OTSUnsupportOperation</Code><Message>Unsupported operation:
''.</Message><RequestID>00054ec5-822c-8964-adaf-
990a07a4d0c9</RequestID><HostID>MTAuMTUzLjE3NS4xNzM=</HostID></Error>
```

如果发现是网络不通，可能是在非ECS环境使用了内网的endpoint。

服务端处理慢，超过了SDK设置的SocketTimeout值

一个请求在表格存储服务端的处理时间几乎是不会超过15秒的，因为服务端也会有一个超时时间，大概是10秒，超过这个时间会给客户端返回OTSTimeout错误。

但是，假设在SDK端自定义了SocketTimeout，例如2秒，那么当服务端执行时间超过2秒时，SDK就会抛出SocketTimeoutException错误。

网络传输慢

如果服务端处理时间并不长，但是网络传输慢，导致整体延迟长，也会导致SocketTimeoutException。检查是否存在流量过高、带宽吃紧、网络重传率高等情况。

Java进程的GC频繁，经常FullGC，导致SocketTimeoutException

这种情况也会经常遇到，程序负载高，GC频繁时出现SocketTimeoutException。原因是当发生FullGC的时候，请求发不出去，或者收不到响应，超过了SDK端设置的SocketTimeout，就会抛出SocketTimeoutException。

这种情况下需要用工具分析进程的GC情况，解决进程频繁GC的问题。

Java SDK日志库相关问题

表格存储Java SDK使用的是哪个日志库？

表格存储 Java SDK 依赖的是 slf4j，在依赖中默认依赖了 **log4j2** 作为日志实现库。

如何替换日志库？

您只需要在 Java SDK 的依赖中把 **log4j2** 的依赖声明移除即可，slf4j 就会自动在您的应用中寻找依赖的其他实现 slf4j 接口的日志库。

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>ots-public</artifactId>
<version>2.2.4</version>
<exclusions>
<exclusion>
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-api</artifactId>
</exclusion>
<exclusion>
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-core</artifactId>
</exclusion>
<exclusion>
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-slf4j-impl</artifactId>
</exclusion>
</exclusions>
</dependency>
```

主键类型报错

现象

Caused by: [ErrorCode]:OTSInvalidPK, [Message]:Validate PK type fail. Input: VT_STRING, Meta: VT_BLOB., [RequestId]:00055f43-3d31-012b-62c3-980a3eefe39e, [TraceId]:02822839-3b5b-af35-409a-cf68841239fa, [HttpStatus:]400

原因

建表时设置的主键类型为binary，写入数据时主键填了string类型的值。

解决方案

写入数据的类型与建表时设置的主键类型保持一致。

Java SDK报错：java.lang.IllegalStateException: Request cannot be executed; I/O reactor status: STOPPED

现象

使用Java SDK时出现如下异常：

java.lang.IllegalStateException: Request cannot be executed; I/O reactor status: STOPPED

原因

一般是因为OTSClient被调用了shutDown，其内部的I/O reactor均已被关闭。如果此时再调用OTSClient进行读写，则会抛出这个错误。

解决方案

检查OTSClient是否处于shutdown状态。

使用Java SDK时遇到Protobuf或HttpClient库冲突

现象

表格存储的Java SDK依赖了2.4.1版本的Protobuf库以及4.0.2版本的httpasyncclient，容易与您的应用程序中自带的相同库冲突。

解决方案

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>tablestore</artifactId>
<version>替换为您当前使用的版本</version>
<classifier>jar-with-dependencies</classifier>
<exclusions>
<exclusion>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
</exclusion>
<exclusion>
<groupId>org.apache.httpcomponents</groupId>
<artifactId>httpasyncclient</artifactId>
</exclusion>
</exclusions>
</dependency>
```

说明：classifier为jar-with-dependencies，它将依赖的HttpClient和Protobuf这两个库都通过rename package的方式打包进去，去除了对HttpClient和Protobuf的依赖。

使用SDK出现OTSUnsupportOperation异常

现象

调用syncClient.createTable(request)时出现如下错误：

Caused by: [ErrorCode]:OTSUnsupportOperation, [Message]:Unsupported operation: ‘CreateTable’.

原因

使用4.0.0之后版本的SDK访问旧版本的表。

解决方案

可以使用2.x.x版本的SDK：

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>ots-public</artifactId>
<version>2.2.5</version>
</dependency>
```

使用BatchWriteRow一次提交100条数据的时候报OTSParameterInvalid错误

现象

使用BatchWriteRow一次提交100条数据时出现以下错误：

ErrorCode: OTSParameterInvalid, ErrorMessage: The input parameter is invalid.,

原因

因为一次batch操作不能有重复行，如果有重复行，则会报错。

解决方案

将100条改为1条提交一次，其他代码不变，即可提交成功。

为什么使用表格存储的过程中会有少量的500错

误

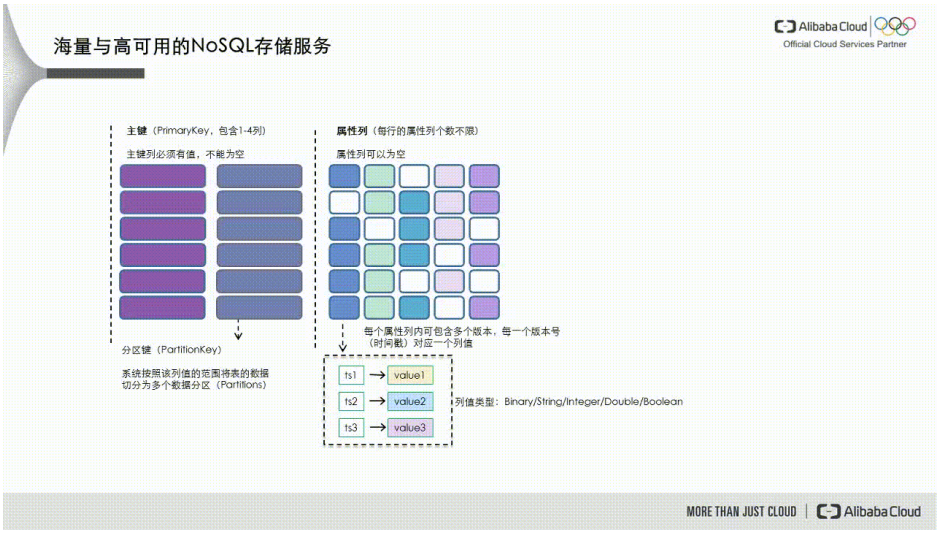
现象

不少用户在使用表格存储的过程中偶尔会接到一些500错误，主要错误码如下：

HTTPStatus	ErrorCode	ErrorMsg
503	OTSPartitionUnavailable	The partition is not available.
503	OTSServerUnavailable	Server is not available.
503	OTSServerBusy	Server is busy.
503	OTSTimeout	Operation timeout.

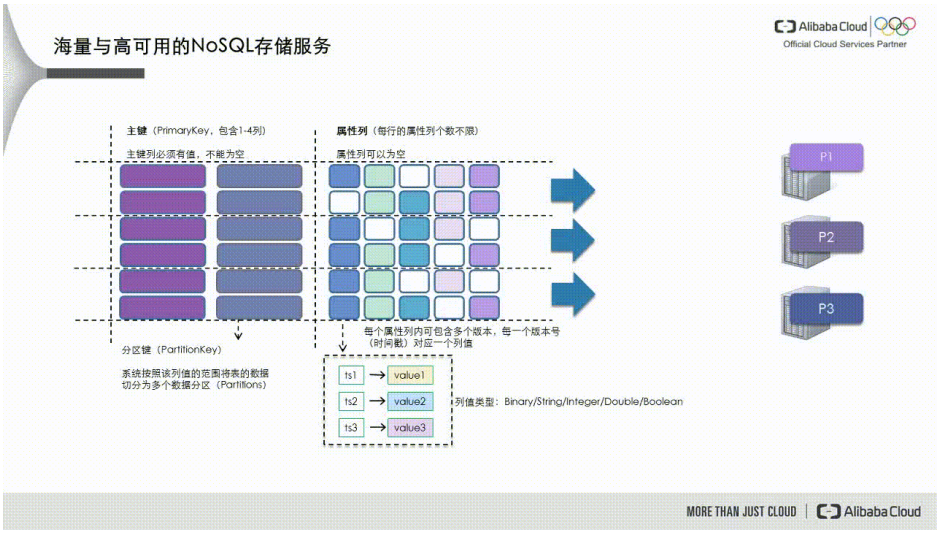
这是由于表格存储是一个纯分布式的NoSQL服务，服务端会根据数据分区的数据量、访问情况做自动的负载均衡，这样就突破了单机服务能力的限制，实现了数据规模和访问并发的无缝扩展。

如下图所示，表格存储会按照第一个主键的顺序，将实际数据划分到不同的数据分区中，不同的数据分区会调度到不同的服务节点提供读写服务。



当某个数据分区的数据量过大，或者访问过热，如下图的数据分区P1，表格存储的动态负载均衡机制能够检测到这种情况的发生，并将数据分区分裂成两个数据分区P1’ 和P5，并将该两个数据分区调度到负载较低的服务节点上。

表格存储使用上述的自动负载均衡机制实现表数据规模和访问并发的自动扩展，全程无需人工介入， 当然在数据表新建立时，只有一个数据分区，该表上能够提供的读写并发有限，自动负载均衡机制也有一定的延时性，所以可以直接联系到我们的工程师，预先将数据表划分成多个数据分区。



表格存储使用共享存储的机制，数据分区为逻辑单位，所以在负载均衡的过程中，不会有实际数据的迁移，仅仅是数据表元信息的变更，在元信息变更的过程中，为了保证数据的一致性，涉及到的数据分区会有短暂的不可用时间， **正常情况下影响时间为百毫秒级别，在数据分区负载较大时，可能会持续到秒级别，** 在这个时间内对该分区的读写操作就有可能接到上述的错误，一般重试即可解决。在官方的SDK中默认提供了一些重试策略，在初始化Client端时就可以指定重试策略。

同时，表格存储提供的也是标准Restful API协议，由于网络环境的不可控，所有的读写操作也都建议增加重试策略，能够对网络错误等有一定的容错能力。

> **说明：** 批量读写操作BatchWriteRow及BatchGetRow读写的数据可能属于多张表或者一张表的多个数据分区，有可能某一个分区正好在分裂，所以整个操作是非原子性的，只能够保证每个单行操作的原子性，该操作返回码为200时仍然需要检查response中的getFailedRows() 是否有失败的单行操作。

只设置一个主键，如何获取多行数据？

关于如何查询多行数据，可以使用GetRange接口，具体的代码示例请参考Github。

如何实现分页查询

表格存储是一个分布式存储系统，对于查询请求的翻页（分页），有多种方式。本文详细为您介绍如何实现分页查询。

表

如果只有表，没有多元索引，可以通过以下办法翻页：

- 使用next_token翻页：每次GetRange请求的Response中会有一个next_token，将这个next_token设置到下一次请求的Request中即可，这样就能实现连续翻页。
- 使用GetRangeIterator迭代器，通过iterator.next()方法持续获取下一条数据。
- 不支持offset跳页查询，如果业务需要，可以在客户端通过next_token或Iterator模拟。
- 不支持获取整个范围的行数和总页数。

多元索引

如果创建了多元索引，则可以通过以下办法翻页：

- 使用offset + limit方式：可以跳页，但是offset + limit最大值不能超过10000。如果超过，请使用第二种方法。
- 使用next_token翻页，每次Search请求的Response中会有下一次的next_token，将这个next_token设置到下一次请求的Request中即可，这样就能实现连续翻页。
- 使用SearchIterator迭代器，通过iterator.next()方法持续获取下一条数据。
- 支持获取总行数，总行数除以limit就是总页数，需要在Request中设置getTotalCount为true，该选项打开后会增大资源消耗，所以性能会有所下降。

表示例

下面是一个实现分页读接口的示例代码，提供offset过滤以及读取指定页数的数据。

```
/**
* 范围查询指定范围内的数据，返回指定页数大小的数据，并能根据offset跳过部分行。
*/
private static Pair<List<Row>, RowPrimaryKey> readByPage(OTSClient client, String tableName,
RowPrimaryKey startKey, RowPrimaryKey endKey, int offset, int pageSize) {
Preconditions.checkArgument(offset >= 0, "Offset should not be negative.");
Preconditions.checkArgument(pageSize > 0, "Page size should be greater than 0.");
List<Row> rows = new ArrayList<Row>(pageSize);
int limit = pageSize;
int skip = offset;
RowPrimaryKey nextStart = startKey;
// 若查询的数据量很大，则一次请求有可能不会返回所有的数据，需要流式查询所有需要的数据。
while (limit > 0 && nextStart != null) {
// 构造GetRange的查询参数。
// 注意：startPrimaryKey需要设置为上一次读到的位点，从上一次未读完的地方继续往下读，实现流式的范围查询
。
RangeRowQueryCriteria criteria = new RangeRowQueryCriteria(tableName);
criteria.setInclusiveStartPrimaryKey(nextStart);
criteria.setExclusiveEndPrimaryKey(endKey);
// 需要设置正确的limit，这里期望读出的数据行数最多为完整的一页数据以及需要过滤(offset)的数据
criteria.setLimit(skip + limit);
```

```
GetRangeRequest request = new GetRangeRequest();
request.setRangeRowQueryCriteria(criteria);
GetRangeResult response = client.getRange(request);
for (Row row : response.getRows()) {
if (skip > 0) {
skip--; // 对于offset之前的数据，需要过滤掉，采用的策略是读出来后在客户端进行过滤。
} else {
rows.add(row);
limit--;
}
}
// 设置下一次查询的起始位点
nextStart = response.getNextStartPrimaryKey();
}
return new Pair<List<Row>, RowPrimaryKey>(rows, nextStart);
}
```

下面是使用以上接口，顺序地按页读取某个指定范围内的所有数据。

```
private static void readByPage(OTSClient client, String tableName) {
int pageSize = 8;
int offset = 33;
RowPrimaryKey startKey = new RowPrimaryKey();
startKey.addPrimaryKeyColumn(COLUMN_GID_NAME, PrimaryKeyValue.INF_MIN);
startKey.addPrimaryKeyColumn(COLUMN_UID_NAME, PrimaryKeyValue.INF_MIN);
RowPrimaryKey endKey = new RowPrimaryKey();
endKey.addPrimaryKeyColumn(COLUMN_GID_NAME, PrimaryKeyValue.INF_MAX);
endKey.addPrimaryKeyColumn(COLUMN_UID_NAME, PrimaryKeyValue.INF_MAX);
// 读第一页，从范围的offset=33的行开始读起
Pair<List<Row>, RowPrimaryKey> result = readByPage(client, tableName, startKey, endKey, offset,
pageSize);
for (Row row : result.getKey()) {
System.out.println(row.getColumns());
}
System.out.println("Total rows count: " + result.getKey().size());
// 顺序翻页，读完范围内的所有数据
startKey = result.getValue();
while (startKey != null) {
System.out.println("============== start read next page ==============");
result = readByPage(client, tableName, startKey, endKey, 0, pageSize);
for (Row row : result.getKey()) {
System.out.println(row.getColumns());
}
startKey = result.getValue();
System.out.println("Total rows count: " + result.getKey().size());
}
}
```

多元索引示例

详见翻页与排序。

如何实现对特定列加一操作

您可以采取以下方式实现加一操作：

```
row = getRow(primary_key, 'col') // 先将该列的值读出来
old_value = row['col'] // 记录该列的旧的值
row['col'] = old_value + 1 // 计算新的值
updateRow(row, condition: row['col'] == old_value) // 写入新的值，写入时必须带条件检查，期望在写入时，当前列还是旧的值，即还没有其他人同时修改这一列
```

Python SDK ListTable示例

ListTable代码示例

```
# -- coding: utf8 --

import time

import logging

import unittest

from ots2 import *

ENDPOINT = “https://xxx.cn-hangzhou.ots.aliyuncs.com“;

ACCESSID = “xxx”；

ACCESSKEY = “xxx”；

INSTANCENAME = “xxx”；

ots_client = OTSClient(ENDPOINT, ACCESSID, ACCESSKEY, INSTANCENAME)

list_response = ots_client.list_table()

print u’ instance table:‘

for table_name in list_response:
```

```
print table_name
```

说明： Python SDK的安装和操作，请参见Python SDK文档。

Python SDK文档中没有import的提示，如果不加import的话会出现如下提示：

```
Traceback (most recent call last):
File “listtable.py”, line 6, in
ots_client = OTSClient(ENDPOINT, ACCESSID, ACCESSKEY, INSTANCENAME)
NameError: name ‘OTSClient’ is not defined
```

添加import运行即可：

```
[root@ example]# cat list.py
# -*- coding: utf8 -*-

import time
import logging
import unittest

from ots2 import *

ENDPOINT = "http://.cn-hangzhou.ots.aliyuncs.com"
ACCESSID = "Tb";
ACCESSKEY = "Y";
INSTANCENAME = "b ng";

ots_client = OTSClient(ENDPOINT, ACCESSID, ACCESSKEY, INSTANCENAME)

list_response = ots_client.list_table()
print u'instance table:'
for table_name in list_response:
    print table_name
[root@ # python list.py
instance table:
simple
test
```

索引问题

去重（collapse）使用

使用collapse去重后，返回结果可以按照设定的字段折叠，示例如下：

```
private static void UseCollapse(SyncClient client){
SearchQuery searchQuery = new SearchQuery();
MatchQuery matchQuery = new MatchQuery();
matchQuery.setFieldName("user_id");
matchQuery.setText("00002");

searchQuery.setQuery(matchQuery);
Collapse collapse = new Collapse("product_name"); //根据产品名去重

searchQuery.setGetTotalCount(true);
searchQuery.setCollapse(collapse);
SearchRequest searchRequest = new SearchRequest("order","order_index",searchQuery);

SearchResponse response = client.search(searchRequest);
System.out.println(response.getTotalCount());
System.out.println(response.getRows().size()); //有几种产品会返回几个产品名
}
```

全局二级索引和多元索引的选择

本文主要对原生Tablestore查询、全局二级索引（Global Secondary Index）和多元索引（Search Index）三种查询场景进行详细分析。

详细分析参见Tablestore存储和索引引擎详解。

原生Tablestore

数据查询依赖主键，主要是通过主键点查询（GetRow），主键范围查询（GetRange）。如需对属性列进行查询，需要使用Filter功能，在数据量很大的时候效率不高，甚至变成全表扫描。在实际业务中，主键查询也常常不能满足需求，而使用Filter在大数据量时效率很低。Tablestore推出了全局二级索引和多元索引，这两个功能弥补了原生Tablestore查询方式单一的缺点，本文主要为您分析全局二级索引以及多元索引的区别及选择。

全局二级索引

主表建立全局二级索引后，相当于多了一张Tablestore表，所以索引表的模型与Tablestore表一致。索引表相当于给主表提供了另外一种排序方式，即对查询条件预先设计了一种数据分布，加快数据查询的效率，索引表的查询方式仍然是基于主键点查、主键范围查、主键前缀范围查询。为了确保主键的唯一性，全局二级索引会将主表的主键列也放到索引表中。

多元索引

多元索引相比以上两种，底层增加了倒排索引，多维空间索引等，支持多字段自由组合查询、模糊查询、地理位置查询、全文检索等，相比功能较为单纯的二级索引更加丰富，而且一个索引可以满足多种维度的查询，支持多种查询条件，因此命名为多元索引。

索引选择

不一定需要索引

- 如果基于主键和主键范围查询的功能已经可以满足业务需求，那么不需要建立索引。
- 如果对某个范围内进行筛选，范围内数据量不大或者查询频率不高，可以使用Filter，不需要建立索引。
- 如果是某种复杂查询，执行频率较低，对延迟不敏感，可以考虑通过DLA（数据湖分析）服务访问Tablestore，使用SQL进行查询。

全局索引or多元索引

一个全局二级索引是一个索引表，类似于主表，其提供了另一种数据分布方式，或者认为是另一种主键排序方式。一个索引对应一种查询条件，预先将符合查询条件的数据排列在一起，查询效率很高。索引表可支撑的数据规模与主表相同，此外，全局二级索引的主键设计也同样需要考虑散列问题。

一个多元索引是一系列数据结构的组合，其中的每一列都支持建立倒排索引等结构，查询时可以按照其中任意一列进行排序。一个多元索引可以支持多种查询条件，不需要对不同查询条件建立多个多元索引。相比全局二级索引，也支持多条件组合查询、模糊查询、全文索引、地理位置查询等。多元索引本质上是通过各种数据结构加快了数据的筛选过程，功能丰富，但在数据按照某种固定顺序读取这种场景上，效率不如全局二级索引。多元索引的查询效率与倒排链长度等因素相关，即查询性能与整个表的全量数据规模有关，在数据规模达到百亿行以上时，建议使用RoutingKey对数据进行分片，查询时也通过指定RoutingKey查询来减少查询涉及到的数据量。简而言之，查询灵活度和数据规模不可兼得。

多元索引路由字段的使用

在创建多元索引时可以选择部分主键列作为路由字段，在进行索引数据写入时，会根据路由字段的值计算索引数据的分布位置，路由字段的值相同的记录会被索引到相同的数据分区中。

使用方法

在创建索引时，指定一个或多个路由字段。

您在创建多元索引时指定了路由字段后，索引数据的读写都会使用该路由字段进行定位，不能动态改变。如果想使用系统默认路由（即主键列路由）或者重新指定其他字段为路由字段，您需要重建索引。

路由字段只能是表格存储的主键列。

在索引查询时，在查询请求中指定路由字段值。

查询时使用路由，定向搜索指定数据分区，可以减少长尾对延迟的影响。对于自定义路由的查询请求，都要求用户提供路由字段。如不指定，虽然查询结果一样，但查询时会访问无关的数据分区，浪费系统资源，增加访问延迟。

示例代码

```
private static void testRoute(SyncClient client) throws InterruptedException {
//创建表
TableMeta meta = new TableMeta("order");
meta.addPrimaryKeyColumn("order_id",PrimaryKeyType.STRING);
meta.addPrimaryKeyColumn("user_id",PrimaryKeyType.STRING);
TableOptions options = new TableOptions();
options.setMaxVersions(1);
options.setTimeToLive(-1);
CreateTableRequest request = new CreateTableRequest(meta,options);
request.setReservedThroughput(new ReservedThroughput(new CapacityUnit(0, 0)));
CreateTableResponse response = client.createTable(request);

//创建多元索引并指定路由字段

CreateSearchIndexRequest searchIndexRequest = new CreateSearchIndexRequest();
searchIndexRequest.setTableName("order"); //订单表
searchIndexRequest.setIndexName("order_index"); //订单表索引名
IndexSchema indexSchema = new IndexSchema();
IndexSetting indexSetting = new IndexSetting();
indexSetting.setRoutingFields(Arrays.asList("user_id"));//设置商户id为路由字段
indexSchema.setIndexSetting(indexSetting);

//添加索引字段 这里只是给出示例，您可以根据业务需求添加索引字段
indexSchema.setFieldSchemas(Arrays.asList(
new FieldSchema("product_name",FieldType.KEYWORD).setStore(true).setIndex(true),
new FieldSchema("order_time",FieldType.LONG).setStore(true).setEnableSortAndAgg(true).setIndex(true),
new FieldSchema("user_id",FieldType.KEYWORD).setStore(true).setIndex(true)
));

searchIndexRequest.setIndexSchema(indexSchema);
client.createSearchIndex(searchIndexRequest);
```

```
Thread.sleep(6*1000); // 等待数据表加载

//插入一些测试数据

String[] productName = new String[]{"product a", "product b", "product c"};
String[] userId = new String[]{"00001", "00002", "00003", "00004", "00005"};
for (int i = 0; i < 100; i++){

PrimaryKeyBuilder primaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
primaryKeyBuilder.addPrimaryKeyColumn("order_id",PrimaryKeyValue.fromString(i+""));
primaryKeyBuilder.addPrimaryKeyColumn("user_id",PrimaryKeyValue.fromString(userId[i%(userId.length)]));
PrimaryKey primaryKey = primaryKeyBuilder.build();

RowPutChange rowPutChange = new RowPutChange("order",primaryKey);

//写入属性列
rowPutChange.addColumn("product_name",ColumnValue.fromString(productName[i%(productName.length)]));
rowPutChange.addColumn("order_time",ColumnValue.fromLong(System.currentTimeMillis()));
rowPutChange.setCondition(new Condition(RowExistenceExpectation.IGNORE));

client.putRow(new PutRowRequest(rowPutChange));

}

Thread.sleep(20*1000);//等待数据同步到多元索引

//带上路由字段的查询
SearchRequest searchRequest = new SearchRequest();
searchRequest.setTableName("order");
searchRequest.setIndexName("order_index");
MatchQuery matchQuery = new MatchQuery();
matchQuery.setFieldName("user_id");
matchQuery.setText("00002");
SearchQuery searchQuery = new SearchQuery();
searchQuery.setQuery(matchQuery);
searchQuery.setGetTotalCount(true);

SearchRequest.ColumnsToGet columnsToGet = new SearchRequest.ColumnsToGet();
columnsToGet.setReturnAll(true);
searchRequest.setColumnsToGet(columnsToGet);
searchRequest.setSearchQuery(searchQuery);

PrimaryKeyBuilder pkbuild = PrimaryKeyBuilder.createPrimaryKeyBuilder();
pkbuild.addPrimaryKeyColumn("user_id",PrimaryKeyValue.fromString("00002"));
PrimaryKey routingValue = pkbuild.build();
searchRequest.setRoutingValues(Arrays.asList(routingValue));
SearchResponse searchResponse = client.search(searchRequest);

System.out.println(searchResponse.isAllSuccess());
System.out.println("totalCount:"+ searchResponse.getTotalCount());
System.out.println("RowCount:"+searchResponse.getRows().size());

}
```

联系我们

如果您有任何反馈或者期望能够与工程师即时沟通，欢迎通过以下方式联系我们：

联系方式	地址
阿里云聆听	https://connect.aliyun.com/
云栖社区	https://yq.aliyun.com/teams/4/type_blog-cid_22-page_1
官方论坛	BBS论坛
钉钉用户群群号	11789671
钉钉用户群二维码