Table Store

FAQ

FAQ

Common issues

How does Table Store differ from conventional RDBMSs?

Table Store is a 2D, NoSQL data storage table system that uses rows and columns. Table Store features attribute columns that can be dynamically added or reduced without requiring a strict schema during table creation.

Although Table Store does not support the rich functions of traditional RDBMS (such as views, indexes, transactions, and innumerous SQL statements), it provides better scalability, easy-to-use functions, and supports extremely large volumes of data (hundreds of TB) in addition to concurrent queries at 100,000 QPS for each table.

Table Store also offers a uniform HTTP RESTful API for easier programming, and you only pay according your actual usage of the Table Store resources (storage and read/write throughput) which means it has no hidden costs.

What do I need to pay attention to when creating tables?

Is the attribute column required for table creation?

No. Table Store supports semi-structured tables. Only the primary key columns (columns 1-4) are required during table creation.

The number of attribute columns of a table is not limited in Table Store. Each row of data can have different quantities and types of attribute columns. When the application is writing data, Table Store needs the application to specify the names and values of all the data columns (primary key columns and attribute columns).

What is the partition key in the first column of the primary key at table creation?

When the table data size reaches a set value, Table Store partitions the table based on the range of values in the partition key column to achieve load balancing.

At table creation, the table has one partition by default and all table data is in this partition. When the table has multiple partitions, the data stored in each partition is all the data within a certain range of values in the partition key column. All the ranges of values in the partition key column are segmented by the natural order of the column values based on the data type Integer or String in the primary key column.

In addition to the data query performance, data partitioning also affects the usage of your reserved read/write throughput. When a table has multiple partitions, the reserved read/write throughput of the table is proportioned to each partition.

How do I set a proper partition key?

Selection of the partition key is important at table creation because it affects the query performance of the table when massive data exists. When setting the partition key for applications, observe the following basic principles:

Do not use attributes with a fixed value or a small value range, such as user gender (Male/Female).

Avoid attributes that have obvious query hotspots after sorting by the natural order, for example, using TimeStamp as the partition key for querying the latest data.

Use attributes with scattered query hotspots after sorting by the natural order, such as UserID.

What can I do if I cannot predict the query hotspots?

We recommend that you hash the data according to the application features before writing the partition key. For example, when writing a row of data, generate a hash value for the UserID using the simple hash algorithm, splice the hash value and the UserID, and save them as the partition key value

to the Table Store table. This lightweight operation can effectively solve some query hotspot issues. Note that because the partition key value is a result of the spliced hash value and actual value, the application cannot read the range (getRange) using the partition key.

What is the limit on the number of tables under an account? Can I increase the limit?

Each Table Store user can create up to 10 instances and each instance can have up to 64 tables, amounting to a maximum of 100 tables under one account.

The limit on the number of tables may be increased in the following scenarios:

Huge data volumes and high query performance requirement

Different from conventional SQL databases (for example, MySQL) which address massive data access demands by database sharding and table partitioning, Table Store adopts the distributed design and cracks the bottleneck of huge data volumes and access latency.

You can save the structured or semi-structured data in a sparse table, without worrying about the compromised query performance resulting from huge data volumes.

Rapid growth in applications

In addition to the increase in data and access volumes, you may use Table Store to offer services to your customers (such as third-party partners and vendors). If you provide services for vendors and have a solution based on Table Store, you must deploy a set of Table Store tables for each vendor. In this case, the number of tables quickly reaches the upper limit. If you constantly increase the upper limit on table quantities, it can cause uncontrollable costs of operation and maintenance and increase the complexity of global data analysis.

We recommend that you use Table Store in a new way and save the same type of massive structured/semi-structured data to a large table.

Considerations of Table Store

The number of tables is a resource attribute in the distributed architecture of Table Store. It can be understood that the number of tables has a maximum value given a fixed scale of a Table Store cluster. The scalability of Table Store can effectively address the limit of table quantities, but for resource controllability, Table Store sets a limit on the maximum number of tables under one account.

To increase the maximum number of tables under your account, you can open a ticket.

Data types supported by Table Store

Table Store supports the following data types: Integer, String, Boolean, and Double.

Integer indicates a 64-bit signed integer, such as 23, -908, and +87. The values of the Integer type range from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.

String indicates a UTF-8 string, which is enclosed by a pair of single quotation marks (') at input, for example, 'Test', 'Open Table Service', and 'Table Store User Guide'. The length of a string ranges from 0 KB to 512 KB. We recommend that the length not exceed 1,024 bytes.

Boolean indicates data of the Boolean type, which can be set to either TRUE or FALSE.

Double indicates a floating-point number in the 64-bit format of the IEEE 754 standard, for example, 1.23, -34.0. +102.23, 1.0e-10, and 2.006E+3.

Note:

- The Double type is not applicable to the primary key columns of a table or view.
- The Double and Boolean types are not applicable to the data partition key, that is, the first column of the primary key.

Naming conventions for tables and instances

Table name conventions

The following conventions are used for tables, table groups, views, and columns:

Each name can be 1 to 255 bytes in length.

Each name can contain uppercase and lowercase letters, numbers, and underscores (_).

Each name must begin with a letter or underscore.

Instance name conventions

Each instance name must be unique in a region (the same name may be used for instances across different regions).

Each name can be 3 to 16 bytes in length.

Each name can contain uppercase and lowercase letters, numbers, and hyphens (-).

Each name must begin with a letter.

No name can end with a hyphen.

What are primary keys, data partitions, and partition keys

Primary key

A primary key uniquely identifies a row of record in a table. When creating a table, you must specify the columns that generate the primary key. These specified columns are called primary key columns. A primary key column cannot be set to null. The combination of values of a primary key column must be able to uniquely identify a row. The type of a primary key column cannot be changed once created.

Data partition and partition key

To balance the load of stored data, Table Store automatically divides a table into data partitions. Data is partitioned by the first column of the primary key, which is called data partition key.

Rows that have the same data partition key must belong to the same data partition. This makes sure that Table Store can make consistent changes to data with the same data partition key.

The following figure shows a part of an email table in an email system. Information about the primary

key and partition key of the table is as follows:

The UserID, ReceiveTime, and FromAddr columns indicate the ID of the email user, receipt time, and sender, respectively. These columns are the primary key columns and can uniquely identify an email. The UserID column is the data partition key.

The ToAddr, MailSize, Subject, and Read columns indicate the recipient, email size, email subject, and whether the email is read, respectively. These columns are common columns that store information about the email.

In the following figure, Table Store stores information about users with UserIDs U0001 and U0002 in the same data partition, and information about users with UserIDs U0003 and U0004 in another data partition.

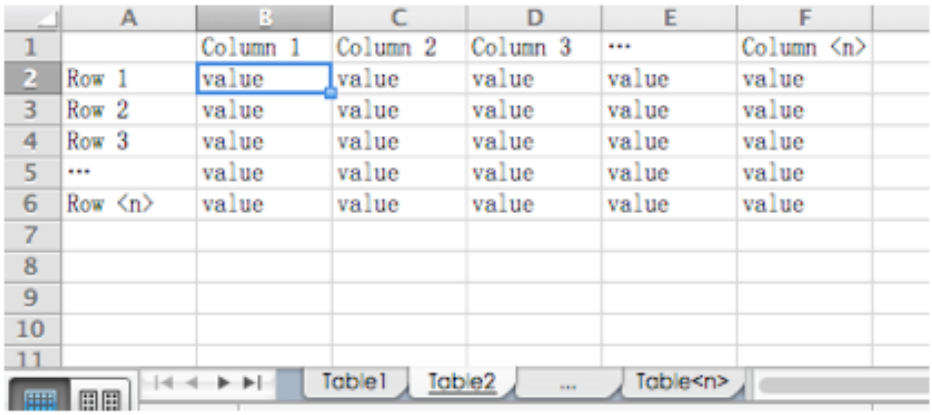
UserID	ReceiveTime	FromAddr	ToAddr	MailSize	Subject	Read
U0001	1998-1-1	eric@demo.com	bob@demo.com	10000	Hello	Y
U0001	2011-10-20	alice@demo.com	bob@demo.com; vivian@demo.com	15000	Fw: Greetings	Y
U0001	2011-10-21	alice@demo.com	team1@demo.com	8900	Review	N
U0001	2011-10-24	lucy@demo.com	vteam@demo.com	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

Partition key | Primary key | Column | Partition

Comparison of Table Store and a workbook

in terms of table, row, column, and value

Table Store tables enable you to manage structured data in tables by allowing you to query, insert, modify, and delete data in Table Store tables. Data in a Table Store table is organized by row, column, and value.



The preceding figure details basic Table Store concepts, which are described as follows:

Table: A table is referenced as a label that is featured at the bottom of the page. Different labels correspond to different tables.

Row: Each row contains a tuple mapping columns to values.

Column: Data in the same column has the same category attributes (that is, data types).

Value: Available value types are: String, Integer, Boolean, Double, or Binary. Table Store allows some columns to have no values. If a column does not have a value, the column does not use storage space.

> **Note:** If the column is a primary key column, the value type can be only String, Integer, or Binary.

User authentication

Table Store uses the symmetric signature to verify that a request is sent by its owner and a response

is returned by Table Store.

After you register an account, you can obtain an AccessKeyID and AccessKeySecret. The AccessKeyID represents your identity, and the AccessKeySecret is used to sign and verify requests and responses. The AccessKeySecret must be kept confidential.

When you send a request to Table Store, the request must contain the request plaintext, your AccessKeyID, and the verification code that is generated when the AccessKeySecret is used to sign information in the request plaintext.

After receiving the request, Table Store finds the AccessKeySecret corresponding to the AccessKeyID and signs the information in the plaintext. A request is considered valid if the calculated verification code is the same as the provided one.

To verify a response from Table Store, you must perform the same calculation. A response is valid if the calculated verification code is the same as the provided one.

How do I obtain AccessKeyID and AccessKeySecret

Procedure

Log on to the Alibaba Cloud console.

At the upper-right corner, hover the cursor over the username, and select **accesskeys** from the shortcut menu that appears.

In the security prompt box, click **Continue to manage AccessKey**.

The AccessKeyID and AccessKeySecret are displayed.

AccessKey permission control

Currently, Table Store only supports authentication based on your AccessKeyID and AccessKeySecret.

After authentication, you can access resources in Table Store without any access restraints.

If an Alibaba Cloud account has multiple pairs of AccessKeyID and AccessKeySecret, the resources in Table Store of that account are displayed regardless of which AccessKeyID and AccessKeySecret are used for access.

If you use a RAM user account to access Table Store resources, make sure that the RAM user has been granted permissions for accessing Table Store by the primary Alibaba Cloud account. A RAM user can only access resources whose permissions have been authorized. For more information about authorization, see Authorization management.

Is the query speed affected by the storage data volume

For single-row query and range query, the query speed is independent of the data volume.

As a NoSQL database, Table Store allows data volumes to extend linearly alongside cluster scaling without affecting the speeds of single-row query and range query. The query speed does not change even if the data volume reaches the hundred-million-level or ten-billion-level.

On high-performance instances that use SSDs, the querying speed of a single row is at the millisecond-level. If the data volume of a single row is small, the query speed is generally within 10 ms.

For more information about query APIs, see GetRow, GetRange, and BatchGetRow.

Measuring & Billing

What are the requirements for reserved read/write throughput?

Reserved read/write throughput is an attribute of a table. The system pre-allocates resources in the background based on the table’ s reserved read/write throughput configuration. This guarantees

your throughput needs for a specified table.

When using CreateTable, you must specify the table’s reserved read/write throughput at its creation. Once a table has been created, you can use UpdateTable to update the table’s reserved read/write throughput configuration.

Table Store requires each tables’ reserved read throughput and write throughput to be greater than 0, but less than 5,000. If a table requires a reserved read throughput, or write throughput, to be greater than 5,000, you must open a ticket.

The measurement units for reserved read/write throughput are capacity units (CU). When applications perform Table Store read/write operations through the API, this consumes a corresponding amount of read and write CUs.

Table Store charges an hourly fee based on the sum of reserved read/write throughput for all tables in an instance. As a users’ reserved read/write throughput configuration can dynamically change, Table Store uses statistics about a table’s reserved read/write throughput generated at set time intervals to calculate the average reserved read/write throughput per hour. This value is then multiplied by the unit price to obtain the actual fee. The reserved read/write throughput unit price may be updated by Alibaba Cloud. For the latest billing details, see Pricing.

How does Table Store manage data storage?

Table Store charges fees per hour based on the total volume of instance data. Because a user’s data size can change dynamically, Table Store calculates a table’s data size at set time intervals, and then calculates the hourly average data size. The fee is calculated by multiplying the average data size by the unit price. The unit price may be updated by Alibaba Cloud. For the latest billing information, see Pricing.

An instance’s total data volume is the sum of the data from each table in the instance, and a table’s data volume is the sum of the data from all rows in the table. The following example explains how a table’s data volume is calculated.

In the following table, the ID is the primary key column, and all other columns are attribute columns:

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

For the row id=1, its data volume is: len（‘id’）+ len（‘name’）+ len（‘length’）+ len

(‘comments’) + 8 Byte + 10 Byte + 8 Byte + 32 Byte = 78 Byte.

For the row id=2, its data volume is: len (‘id’) + len (‘name’) + len (‘length’) + len (‘comments’) + 8 Byte + 20 Byte + 8 Byte + 999 Byte = 1055 Byte.

For the row id=3, its data size is: len (‘id’) + len (‘name’) + len (‘length’) + 8 Byte + 43 Byte + 8 Byte = 71 Byte.

The table’ s data volume is 78 Byte + 1055 Byte + 71 Byte = 1204 Byte. If the table’ s data volume does not change within the measured hour, the user is billed for 1,204 Bytes. Table Store does not limit the data volume of individual tables, nor does it charge for unused services, so it has no hidden costs.

Billing of Table Store

Table Store is a Pay-As-You-Go service, which means it charges fees based on the actual usage of each resource within a billing cycle. You do not need to buy an additional volume for instances.

Currently, Table Store bills an instance in the following four dimensions:

Stored data volume

Reserved read/write throughput

Additional read/write throughput

Downstream Internet traffic

Table Store bills an instance based on the actual usage of resources in a billing cycle (per hour).

How is Table Store billing calculated per hour?

Scenario

In a billing cycle, assume that the data volume and Internet downstream traffic are 50 GB and 10 GB, respectively. At the 20th minute within the one hour, the reserved read/write throughput of all tables under the instance is adjusted from (1000,1500) to (1200,800). 50,000 volume-based read CUs and

10,000 volume-based write CUs are consumed.

Billing formula

The instance of this hour is billed using the following formulas:

Storage fee: 50 GB * Unit price per hour per GB

Traffic fee: 10 GB * Unit price of the Internet downstream traffic

Reserved CU fee:

Average reserved read CUs of this hour: (1000*20 + 1200*40)/60 = 1133.3

Average reserved write CUs of this hour: (1500*20 + 800*40)/60 = 1033

Fee of this hour: 1133.3 * Unit price of the read CUs per hour + 1033.3 * Unit price of the write CUs per hour

Fee of the volume-based CUs: 50,000/10,000 * Unit price of the volume-based read CUs per 10,000 + 10,000/10,000 * Unit price of the volume-based write CUs per 10,000

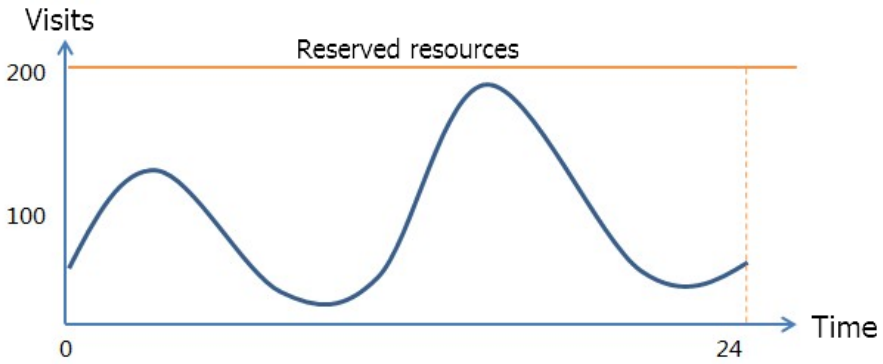
The total amount is the sum of the preceding fees.

The stored data volume and reserved read/write throughput are precise to the minute. At the end of a billing cycle, the average values of the stored data volume and reserved read/write throughput are calculated as the actual usage of the resource in the billing cycle. The volume-based read/write throughput is precise to the second. The volume-based CU used per second in the billing cycle is collected, and all values are aggregated.

Assume that 1000 read CUs are reserved in the first 20 minutes. If 2100 read CUs are consumed in one second, the volume-based read throughput at this second is 1100 (2100–1000). For more information about the metered items of Table Store, see Pricing.

How is Table Store billing calculated per day?

Scenario 1



The preceding figure shows access per day of an app. In this example, assume that the read and write volumes of the app are the same. To make sure that the app contains enough resources to provide read/write services during peak hour, you must buy resources based on the traffic volume in the peak hour. To convert the volume to the CUs of Table Store, you must buy 200 reserved read/write CUs.

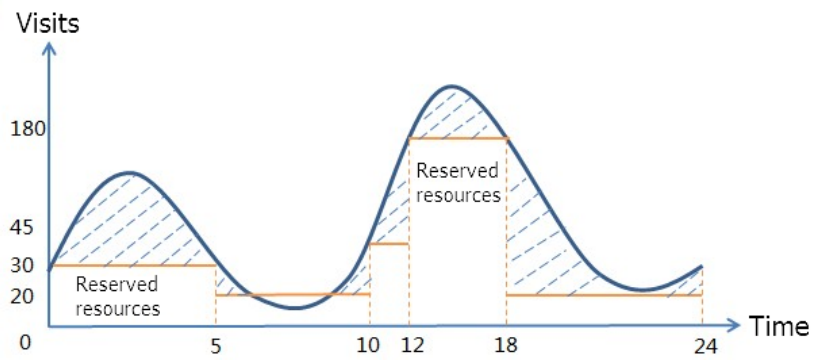
Billing formula

Fee per day = 200 * Unit price of the reserved read CUs per hour * 24 + 200 * Unit price of the reserved write CUs per hour * 24 + data storage fee per day + Internet downstream traffic fee per day

Table Store provides APIs that enable you to adjust the reserved read/write throughput of each table at any time. An adjusted throughput takes effect within one minute. However, you must make sure that the reserved throughput adjustment does not affect the services.

As a result, you can increase the number of reserved CUs during peak hours to meet service growth, and reduce the number of reserved CUs during non-peak hours to save costs.

Scenario 2



In one 24 hour cycle:

> 00:00–05:00: Set the read/write throughput to 30 CUs. During the five hours, the read and write operations each consume 100,000 CUs more than the reserved throughput.

05:00–10:00: The access traffic of the service is reduced. The number of read/write CUs is adjusted to 20. During the five hours, the read and write operations each consume 5000 CUs more than the reserved throughput.

10:00–12:00: The access traffic of the service increases. The number of read/write CUs is adjusted to 45. During the two hours, the read and write operations each consume 10,000 CUs more than the reserved throughput.

12:00–18:00: The service peak is met. The number of read/write CUs is increased to 180. During the six hours, the read and write operations each consume 30,000 CUs more than the reserved throughput.

18:00–24:00: The access peak lowers. The number of read/write CUs is reduced to 20. During the six hours, the read and write operations each consume 50,000 CUs more than the reserved throughput.

Billing formula

In this example, assume that the ratio of the read and write operations is 1:1, and the reserved read and write CUs are adjusted to the same values. The fees of the CUs per day are calculated as follows:

Read CU fee:

(30 * 5 hours + 20 * 5 hours + 45 * 2 hours + 180 * 6 hours + 20 * 6 hours) * Unit price of the reserved read CUs per hour + (100,000 + 5000+ 10,000 + 30,000 + 50,000) * Unit price of the volume-based read CUs

That is, 1540 * Unit price of the reserved read CUs per hour + 195000 * Unit price of the volume-based read CUs

Write CU fee:

(30 * 5 hours + 20 * 5 hours + 45 * 2 hours + 180 * 6 hours + 20 * 6 hours) * Unit price of the reserved write CUs per hour + (100,000 + 5000+ 10,000 + 30,000 + 50,000) * Unit price of the volume-based write CUs

That is, 1540 * Unit price of the reserved write CUs per hour + 195000 * Unit price of the volume-based write CUs

Compared with the billing method in Scenario 1, scenario 2 can save the following cost:

4800 * Unit price of the reserved read CUs per hour + 4800 * Unit price of the reserved write CUs per hour - 1540 * Unit price of the reserved read CUs per hour - 19.5 * Unit price of the volume-based

read CUs - 1540 * Unit price of the reserved write CUs per hour - 19.5 * Unit price of the volume-based write CUs

NOTES

The preceding calculation result does not include the free quota that each registered account is allocated.

The unit price of the volume-based read/write throughput is slightly higher than that of the reserved throughput. We recommend that you properly adjust the reserved values based on your service conditions to effectively reduce the cost.

You can use the SDK to set the reserved read/write throughput of a table to a lower value, so that you can preferentially use your free quota resources.

API/SDK

Java SDK error: Invalid date format

Symptom

In a Java 8 environment, if the Table Store Java SDK is used, the following exception is thrown:

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

Cause

Joda-Time in Classpath, which is SDK-dependent, does not meet the version requirements for Java 8.

Resolution

Update to ots-public 2.2.4. If the SDK depends on Joda-Time, update the Joda-Time version to 2.9.

Java SDK error: SocketTimeoutException

Symptom

The SDK throws SocketTimeoutException during application requests.

Cause

If the difference between the **Receive time** (the time when the SDK receives data) and the **Send time** (the time when the SDK sends data) is greater than the value of SocketTimeout, the SDK throws SocketTimeoutException. In this period, the application sends a request (including network transmission), the server processes the request, and the application receives a response (including network transmission). You can set the value of SocketTimeout when creating the OTSClient. The default value of SocketTimeout is 15s.

Resolution

Depending on the issue, SocketTimeoutException may be resolved as follows:

The network is disconnected.

Run the ping or curl command to check the network connection, especially if SocketTimeoutException is thrown for all requests.

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

If the network is connected and the curl command is run, a result similar to the following is returned:

```
<?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>
```

If the network is disconnected, check whether an intranet endpoint is being used in

a non-ECS environment.

The server processing time exceeds the time the SDK specified for SocketTimeout.

The Table Store server generally processes a request within 15s. If the processing time exceeds the time-out period on the server (typically 10s), the server returns OTSTimeout to the client.

SocketTimeoutException is thrown in the event that a smaller value is set for SocketTimeout, for example, 2s.

The network transmission speed is slow.

SocketTimeoutException may be thrown if latency is present due to low network transmission speeds, even if the server processing time is unaffected. In this case, check whether there is high traffic volume, insufficient bandwidth, or high network retransmission rates.

Garbage collections (GCs), including full GCs, are frequently occurring in Java.

SocketTimeoutException may be thrown when the program load is high and GCs occur frequently. When a full GC occurs, the application fails to send requests or receive responses. As a result, the time specified in the SDK for SocketTimeout expires, and SocketTimeoutException is thrown.

In this case, analyze the cause of GCs and resolve them.

Java SDK log library

Which log library is used by default in Table Store Java SDK?

Table Store Java SDNK depends on SLF4J, and **log4j2** is used as the default log implementation library.

How do I replace the default log library?

Delete **log4j2** from your Java SDK dependencies. Once deleted, SLF4J automatically finds another dependent log library that can implement the SLF4J API in your application.

```
<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>
```

Primary key type error

Symptom

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

Cause

The primary key type is set to **binary** during table creation, while the primary key type is set to **string** during data writing.

Resolution

Set the primary key type for data writing to be the same as that for table creation.

The expiration time of URL signature set in Java SDK is too long and does not take effect

Symptom

The URL signature expiration time set in the Java SDK cannot take effect.

Cause

The default value type for the URL signature expiration time is **int**. If the value is set to a long period of time, such as 100 years, the calculation formula is 3600 x 1000 x 24 x 365 x 100, which exceeds the calculation range of the **int** type.

Resolution

Java SDK needs to call the generatePresignedUrl function to obtain the URL signature link and the expiration time must be set in the function.

Convert the parameter value to **long** type. That is, set the **expiration** parameter to 3600 x 1000 x 24 x 365 x 100.

Java SDK error: java.lang.IllegalStateException

Symptom

The following exception occurs when Java SDK is used:

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

Cause

shutDown is called for OTSClient and all the I/O reactors in OTSClient are disabled. This error is

returned if you call OTSClient for read/write again.

Resolution

Check whether OTSClient is in **shutdown** state.

Conflict with Protobuf or HttpClient when Java SDK is used

Symptom

Table Store Java SDK depends on Protobuf 2.4.1 and HttpAsyncClient 4.0.2, which are known to create conflicts with the same libraries inherent in your application.

Resolution

You can use the following versions provided by Table Store SDK.

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>ots-public</artifactId>
<version>2.2.5</version>
<classifier>jar-with-dependencies</classifier>
<exclusions>
<exclusion>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
</exclusion>
</exclusions>
</dependency>
```

Or:

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>tablestore</artifactId>
<version>4.2.3</version>
<classifier>jar-with-dependencies</classifier>
<exclusions>
<exclusion>
<groupId>com.google.protobuf</groupId>
```

```
<artifactId>protobuf-java</artifactId>
</exclusion>
</exclusions>
</dependency>
```

Note: classifier is used to package HttpClient and Protobuf by using jar-with-dependencies and rename the package to remove dependencies on HttpClient and Protobuf.

Exception OTSUnsupportOperation occurred when SDK is used

Symptom

When syncClient.createTable(request) is called, the following error is returned:

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

Cause

An SDK of version 4.0.0 or later is used to access a table of an earlier version.

Resolution

Use an earlier SDK version (recommended version: 2.x.x).

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

How do I set the request signature?

Table Store transmits requests and responses through HTTP. Request signatures must be formatted

as Uniform Resource Identifiers (URIs). A request must contain APIVersion, Date, OTSAccessKeyId, SignatureMethod, SignatureVersion, and Signature.

APIVersion indicates the version of the current Table Store API. The value is **1**.

Date has the following UTC format only options:

%a, %d %b %Y %H:%M:%S GMT, for example, Mon, 3 Jan 2010 08:33:47 GMT

%A, %d-%b-%y %H:%M:%S GMT, for example, Monday, 3-Jan-10 08:33:47 GMT

%a %b %d %H:%M:%S %Y, for example, Mon Jan 3 08:33:47 2010

The maximum difference between the time indicated by Date and the time when the server receives the request is 15 minutes. If the time difference exceeds 15 minutes, the server returns the error “OTSAuthFailed”.

OTSAccessKeyId indicates the AccessKeyID provided by Table Store.

SignatureMethod indicates the signature algorithm. Currently, only **HMAC-SHA1** is supported.

SignatureVersion indicates the version of the signature algorithm. Currently, only the version whose value is **1** is supported.

Signature indicates the string generated after the whole URL is signed. The string used for signature consists of the resource names and parameters of the user request. After the parameter names and values are URL encoded, an equal sign (=) is used between the names and values of these parameters to form a string. Then, the parameter name-value pairs are alphabetically sorted and connected with the & symbol to form a string. The signature calculation method is as follows:

Base64(hmac-sha1(VERB + " " + "KEY1=VALUE1" + "&" + "KEY2=VALUE2", AccessKey

VERB indicates the requested resource name, which is followed by the parameter name and value.

An example user request is as follows:

http://service.ots.aliyun.com/CreateTable?TableName=CapTable&PK.1.Name=PrimaryKey1&PK.1.Type=STRING&PK.2.Name=PrimaryKey2&PK.2.Type=INTEGER&View.1.Name=View1&View.1.PK.1.Name=PrimaryKey1&View.1.PK.1.Type=STRING&View.1.PK.2.Name=Column1&Vi

ew.1.PK.2.Type=BOOLEAN&View.1.Column.1.Name=Column2&View.1.Column.1.Type=STRING&View.1.Column.2.Name=Column3&View.1.Column.2.Type=DOUBLE&APIVersion=1&Date=Fri%2C%2016%20Sep%202006%2018%3A07%3A20%20GMT&OTSAccessKeyId=ID1&SignatureMethod=HmacSHA1&SignatureVersion=1

Then, the string for signature is:

"/CreateTable" + "" +"APIVersion=1"
+"Date=Fri%2C%2016%20Sep%202006%2018%3A07%3A20%20GMT"
+"&OTSAccessKeyId=ID1" +"&PK.1.Name=PrimaryKey1" +"&PK.1.Type=STRING"
+"&PK.2.Name=PrimaryKey2" +"&PK.2.Type=INTEGER" +"&SignatureMethod=HmacSHA1"
+"&SignatureVersion=1" +"&TableName=CapTable" +"&View.1.Column.1.Name=Column2"
+"&View.1.Column.1.Type=STRING" +"&View.1.Column.2.Name=Column3"
+"&View.1.Column.2.Type=DOUBLE" +"&View.1.Name=View1"
+"&View.1.PK.1.Name=PrimaryKey1" +"&View.1.PK.1.Type=STRING"
+"&View.1.PK.2.Name=Column1" +"&View.1.PK.2.Type=BOOLEAN"

The final input format is (assume that the AccessKey of ID1 is KEY 1, and the following signature string is calculated using KEY1):

http://service.ots.aliyun.com/CreateTable?TableName=CapTable&PK.1.Name=PrimaryKey1&PK.1.Type=STRING&PK.2.Name=PrimaryKey2&PK.2.Type=INTEGER&View.1.Name=View1&View.1.PK.1.Name=PrimaryKey1&View.1.PK.1.Type=STRING&View.1.PK.2.Name=Column1&View.1.PK.2.Type=BOOLEAN&View.1.Column.1.Name=Column2&View.1.Column.1.Type=STRING&View.1.Column.2.Name=Column3&View.1.Column.2.Type=DOUBLE&APIVersion=1&Date=Fri%2C%2016%20Sep%202006%2018%3A07%3A20%20GMT&OTSAccessKeyId=ID1&SignatureMethod=HmacSHA1&SignatureVersion=1&Signature=%2FL034xFZBPO%2BNpxA%2BSufMiOt%2BKQ%3D

Return result:

If the AccessKeyID does not exist, Error 403 Forbidden is returned.

If the signature verification fails, error 403 Forbidden is returned.

If required parameters are not input, Error 400 Bad Request is returned.

The request must be input within 15 minutes before or after the current time of the Table Store server; otherwise, Error 403 Forbidden is returned.

How do I set the response signature?

The response signature is contained in Authorization of the HTTP header, indicating the response is authorized.

Sample header: Authorization:OTS44CF9590006BF252F707:jZNOcbfWmD/A/f3hSvVzXZjM2HU=

The values separated by the colon are the AccessKeyID (left) and signature (right). The verification code is calculated as follows.

```
"Authorization:   OTS " + AccessID + ":" + base64(hmac-sha1(
+ CONTENT-MD5 + "\n"
+ CONTENT-TYPE + "\n"
+ CanonicalizedOTSHeaders
+ CanonicalizedResource
, AccessKey))
```

As shown in the preceding code, the headers of a Table Store response signature include Content-Md5, Content-Type, CanonicalizedOTSHeaders (canonicalized Table Store header), and CanonicalizedResource (canonicalized Table Store resource address). Content-Typeand Content-MD5 are required. A carriage return must be added after each header, with the exception of if CanonicalizedOTSHeaders does not exist. A canonicalized Table Store header is a header prefixed with x-ots-.

The signature of CanonicalizedOTSHeaders must observe the following rules:

Headers are in lowercase.

The headers are sorted in ascending lexicographic order by name.

There cannot be a space before or after the colon that separates the header name and value.

A newline character \n must be used to separate headers.

If CanonicalizedOTSHeaders does not exist, the parameter can remain blank.

The current version has a unique CanonicalizedOTSHeaders, which is x-ots-date. It indicates the time that Table Store returns the response. The format is %a, %d %b %Y %H:%M:%S GMT, for example, Mon, 3 Jan 2010 08:33:47 GMT.

You can calculate the signature in the same way, and check whether the response is valid by

comparing the calculated signature and the one provided by Table Store in terms of consistency.

Error OTSParameterInvalid returned when BatchWriteRow is used to commit 100 data records at a time

Symptom

When BatchWriteRow is used to commit 100 data records at a time, the following error is returned.

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

Cause

Duplicate rows are detected in the batch operation.

Resolution

Commit one data record once, without modifying other code.

Why is Error 500 returned occasionally when Table Store is used?

Symptom

When using Table Store, some common 4xx and 5xx errors that may be reported include:

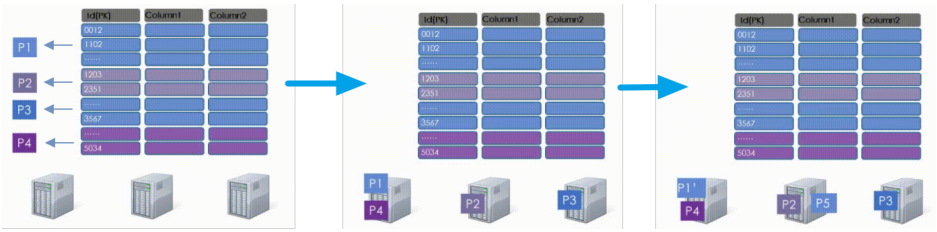
HTTPStatus	ErrorCode	ErrorMsg
503	OTSPartitionUnavailable	The partition is not available.
503	OTSServerUnavailable	Server is not available.
503	OTSServerBusy	Server is busy.

503	OTSTimeout	Operation timeout.

As Table Store is an exclusively distributed NoSQL service, the server automatically balances the load based on the data volume and access condition in the data partitions, enabling seamless scaling of the data size and access concurrency typically restricted by single-host services

Table Store divides data into different data partitions based on the sequence of the first primary key, and data partitions are then scheduled to different service nodes for read and write services.

If the data or access volume of a data partition is too large, Table Store first detects this condition, splits the data partition to two data partitions, and then uses its dynamic load balancing capability to schedule the two data partitions to a service node under a lighter load. In the following figure, P1 is partitioned by Table Store into P1' and P5 as follows:



As shown in the preceding example, Table Store enables automatic scaling of the table data size and access concurrency throughout the entire process, with **no manual intervention required**. However, when a data table is created, only one data partition exists, which provides limited read/write concurrency capabilities. As a result, latency exists even with automatic load balancing. You can open a ticket to request capabilities to divide a data table into multiple data partitions in advance.

Table Store uses shared storage, and data partitions are logic units. During load balancing, data table metadata changes without migration of data. When metadata changes, involved data partitions may be invalid for a short period for the purpose of data consistency. **This period lasts as short as hundreds of milliseconds normally and may last several seconds in the case of a large load of data partitions.**During this period, a 500 error may be reported if the data partition is read or written. You can try again to solve this problem. The official SDK provides some retry policies by default. You can specify a retry policy when initializing the client.

Table Store provides standard RESTful APIs. Due to the uncontrollable network environment, we recommend that you add retry policies for all read/write operations to tolerate network errors to a certain extend.

> **Note:** A data partition may be splitting while BatchWriteRow or BatchGetRow is used to write or read data in batches to or from multiple tables or multiple data partitions of a table, so the write or read operation is non-atomic. The operation is atomic to a row only. You have to check getFailedRows() in the response for failed rows even if a 200 message is returned.

How do I obtain multiple rows of data by setting one primary key

You can use GetRange to query multiple rows. For specific sample code, see Github.

How do I perform paging query?

Constraints

Compared with the paging query supported by conventional relational databases, the pagination feature within NoSQL data models and APIs has the following constraints:

- The number of rows in the whole range cannot be obtained, which means the total number of pages cannot be calculated.
- We recommend that you do not set offset for page jump, because the offset filter runs on the client, and data scanned and read by the server is fixed. If the offset value is too large, the query may take too long to resolve.
- Only sequential paging is supported.

Sample code

The following sample code implements the paginated reading API, which contains the offset filter to read certain data from specified page numbers.

```
    /**
* Query data in the specified range, return data on pages of specified numbers, and skip rows according to the
offset value.
*/
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;
```

```
// If a large amount of data needs to be queried, you must perform stream query to obtain the data.
while (limit > 0 && nextStart != null) {
// Construct a query parameter of GetRange.
// NOTE: Set the startPrimaryKey to the most recently read point in time for the query to continue from the proper start point.
RangeRowQueryCriteria criteria = new RangeRowQueryCriteria(tableName);
criteria.setInclusiveStartPrimaryKey(nextStart);
criteria.setExclusiveEndPrimaryKey(endKey);

// Set a proper limit so that the number of data rows required to be read is a complete page of data, including data that needs to be filtered (offset).
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--; // Data before offset must be filtered out. The policy will read the data first and then filter the data on the client.
} else {
rows.add(row);
limit--;
}
}

// Set the start point for the next query.
nextStart = response.getNextStartPrimaryKey();
}

return new Pair<List<Row>, RowPrimaryKey>(rows, nextStart);
}
```

The following sample code uses data within a specified range that is read page by page using the preceding API.

```
    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);
// Start reading data of the row with offset = 33 in the range on the first page.
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());
```

```
// Read data on the following pages in sequence until all the data in the range is read.
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());
}
}
```

Note: To download the complete example, click here.

How do I add a value to a specified column?

To increase the value of a column by 1 increment, follow these steps.

```
row = getRow(primary_key, 'col') // Read the column value.
old_value = row['col'] // Record the old value of this column.
row['col'] = old_value + 1 // Calculate the new value.
updateRow(row, condition: row['col'] == old_value) // Write the new value to the column. You must set the condition check to make sure that this column retains the old value when writing the new value; that is, the column is not modified by others simultaneously.
```

Example of Python SDK ListTable

ListTable sample code

```
# -- 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
```

Note: For more information about Python SDK, see Python SDK.

The Python SDK documentation does not include information about the import prompt. If import is not added, the following prompt appears.

```
Traceback (most recent call last):

File "listtable.py", line 6, in

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

NameError: name 'OTSClient' is not defined
```

To resolve this issue, you must add import execution as follows.

```
[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
```