Data Transmission Service

FAQ

Product Introduction

Contents

FAQ
Regular Expressions for Data Subscription

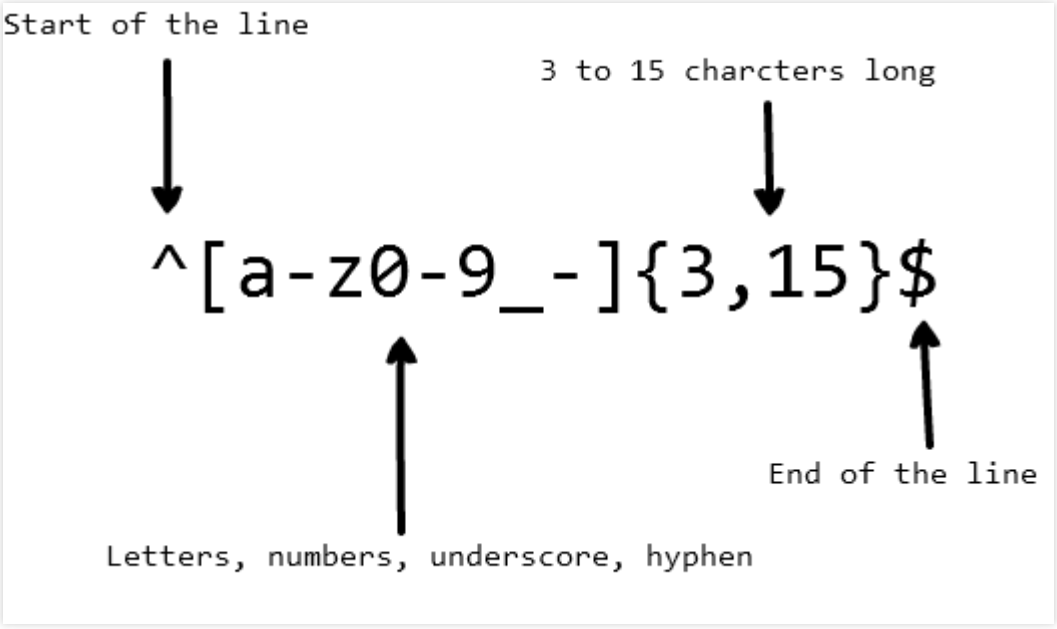
Last updated：2018-08-31 11:55:16

What is a regular expression?

A regular expression is used to retrieve text that meets a certain pattern from text.

It matches a string from left to right. We generally use "regex" or "regexp" for short.
A regular expression can be used to replace text in strings, validate forms, and extract strings from a string based on pattern matching.

Imagine that you are writing an application, and you want to set rules for users to select usernames. We want usernames to contain letters, numbers, underscores, and hyphens.
To make it look good, we also want to limit the number of characters for a username. We can use the following regular expression to verify usernames:



The above regular expression can match `john_doe`, `jo-hn\_doe`, and `john12\_as`. However, it cannot match `Jo` which contains an uppercase character and is too short.

Contents

Basic Match

Regular expressions are patterns we use to retrieve letters and numbers in text. For example, the regular expression `cat` indicates: the letter `c` followed by letters `a` and `t`.

```
"cat" => The cat sat on the mat
```

The regular expression `123` matches the string "123". Regular matching is done by comparing each character in the regular expression with that in the string to be matched one by one.
Regular expressions are generally case sensitive, so the regular expression `Cat` does not match the string "cat".

```
”Cat” => The cat sat on the Cat
```

Metacharacter

Metacharacters are the basic elements of regular expressions. Metacharacters here are not the same as usual, but are interpreted in a special way. Some metacharacters in square brackets have special meaning. Here are the metacharacters:

Metacharacter	Description
.	Match any characters other than line breaks.
[ ]	Character class. Match any characters enclosed in square brackets.
[^]	Negative character class. Match any characters not enclosed in square brackets.
*	Match the preceding subexpression zero or more times
+	Match the preceding subexpression one or more times
?	Match the previous subexpression zero or one time, or specify a non-greedy qualifier.
{n,m}	Curly bracket. Match the preceding character at least n times, but not more than m times.
(xyz)	Character group. Match the character xyz in an exact order.
|	Branch structure. Match characters before or after the symbol.
\	Escape character. It can restore the original meaning of metacharacters, allowing you to match reserved characters `[ ] ( ) { } . * + ? ^ $ \ |`
^	Match the start of the line
$	Match the end of the line

Period

The period `.` is the simplest example of a metacharacter. The metacharacter `.` can match any single character. It does not match a line break or a new line character. For example, the regular expression `.ar` indicates: any characters followed by letters `a` and `r`.

```
".ar" => The car parked in the garage.
```

Character set

A character set is also called character class. Square brackets are used to specify the character set. Specify the character range using the hyphen within the character set. The order of the character ranges in square brackets can be ignored.
For example, the regular expression `[Tt]he` indicates: uppercase `T` or lowercase `t` followed by the letters `h` and `e`.

```
"[Tt]he" => The car parked in the garage.
```

However, the period in the character set indicates its literal meaning. The regular expression `ar[.]` indicates the lowercase letter `a` followed by the letter `r` and a period `.` character.

```
"ar[.]" => A garage is a good place to park a car.
```

Negative character set

In general, the insertion character `^` indicates the start of a string. However, if it appears in square brackets, it cancels the character set. For example, the regular expression `[^c]ar` indicates: any characters other than the letter `c` followed by the character `a` and letter `r`.

```
"[^c]ar" => The car parked in the garage.
```

Repetition

The following metacharacters `+`, `*` or `?` are used to specify how many times the sub-pattern can appear. These metacharacters work differently in different situations.

Asterisk

The symbol `*` indicates matching the previous matching rule zero or more times. The regular expression `a*` indicates that the lowercase `a` can be repeated zero or more times. But if it appears after a character

set or character class, it indicates the repetition of the entire character set.
For example, the regular expression `[a-z]*` indicates: a line containing any number of lowercase letters.

```
"[a-z]*" => The car parked in the garage #21.
```

The symbol `*` can be used with the meta symbol `.` to match any string `.*`. The symbol `*` can be used with the space character `\s` to match a string of space characters.
For example, the regular expression `\s*cat\s*` indicates: zero or more spaces followed by a lowercase letter `c`, a lowercase letter `a`, a lowercase letter `t`, and zero or more spaces.

```
"¥s*cat¥s*" => The fat cat sat on the cat.
```

Plus sign

The symbol `+` matches the previous character one or more times. For example, the regular expression `c.+t` indicates: a lowercase letter `c` followed by any number of characters and a lowercase letter `t`.

```
"c.+t" => The fat cat sat on the mat.
```

Question mark

In regular expressions, the metacharacter `?` is used to indicate that the previous character is optional. This symbol matches the previous character zero or one time.
For example, the regular expression `[T]?he` indicates: the optional uppercase letter `T` followed by a lowercase letter `h` and a lowercase letter `e`.

```
"[T]he" => The car is parked in the garage.
```

```
"[T]?he" => The car is parked in the garage.
```

Curly bracket

Curly brackets (also called quantifier ?) are used in regular expressions to specify the number of times a character or a group of characters can be repeated. For example, the regular expression `[0-9]{2,3}` indicates: matching at least 2 numbers but no more than 3 numbers (characters ranging from 0 to 9).

```
"[0-9]{2,3}" => The number was 9.9997 but we rounded it off to 10.0.
```

We can omit the second number. For example, the regular expression `[0-9]{2,}` indicates: matching 2 or more numbers. If we delete the comma, the regular expression `[0-9]{2}` indicates: matching exactly two-digit numbers.

```
"[0-9]{2,}" => The number was 9.9997 but we rounded it off to 10.0.

"[0-9]{2}" => The number was 9.9997 but we rounded it off to 10.0.
```

Character group

A character group is a set of sub-patterns written in parentheses `(...)`. As we discussed in regular expressions, if we put a quantifier after a character, the previous character is repeated.
However, if we put a quantifier after a character group, the entire character group is repeated.
For example, the regular expression `(ab)*` indicates matching zero or more strings "ab". We can also use the metacharacter `|` in a character group. For example, the regular expression `(c|g|p)ar` indicates: the lowercase letter `c`, `g` or `p` followed by letters `a` and `r`.

```
"(c|g|p)ar" => The car is parked in the garage.
```

Branch structure

The vertical bar `|` is used to define the branch structure in a regular expression. The branch structure is like the condition between multiple expressions. Now you may think that this character set works in the same way as the branch structure.
But the difference is that the character set is only used at the character level, while the branch structure can be used at the expression level.
For example, the regular expression `(T|t)he|car` indicates: the uppercase letter `T` or lowercase letter `t` is followed by a lowercase letter `h`, a lowercase letter `e` or lowercase letter `c`, then a lowercase letter `a`, and a lowercase letter `r`.

```
"(T|t)he|car" => The car is parked in the garage.
```

Escape special character

Use the backslash `\` in the regular expression to escape the next character. This allows you to use reserved characters as the matching characters `{ } [ ] / \ + * . $ ^ | ?`. You can use it as a matching character by adding a `\` before a special character.
For example, the regular expression `.` is used to match any characters other than line breaks. To match the `.` character in the input string, the regular expression `(f|c|m)at\.?` indicates: the lowercase letter `f`, `c`, or `m` followed by a lowercase letter `a`, a lowercase letter `t`, and an optional `.` character.

```
"(f|c|m)at¥.?" => The fat cat sat on the mat.
```

Locator

In regular expressions, we use locators to check whether the matching symbol is a start or end symbol. There are two types of locators: `^`, which checks if the matching character is the start character, and `$`, which checks if the matching character is the end character of an input string.

Caret

The caret `^` is used to check if the matching character is the first character of an input string. If we use the regular expression `^a` (if "a" is the start symbol) to match the string `abc`, it matches `a`.
But if we use the regular expression `^b`, it matches nothing, because "b" in the string `abc` is not the start character.
Take a look at another regular expression `^(T|t)he`, which indicates: the uppercase letter `T` or lowercase letter `t` is the start symbol of the input string, followed by a lowercase letter `h` and a lowercase letter `e`.

```
"(T|t)he" => The car is parked in the garage.

"^(T|t)he" => The car is parked in the garage.
```

Dollar sign

Dollar sign `$` is used to check if the matching character is the last character of an input string. For example, the regular expression `(at\.)$` indicates: the lowercase letter `a` followed by the lowercase letter `t` and character `.`, and this matcher must be the end of the string.

```
"(at¥.)" => The fat cat. sat. on the mat.

"(at¥.)$" => The fat cat sat on the mat.
```

Abbreviated Character Set

Regular expressions provide abbreviations for common character sets and regular expressions. The abbreviated character set is as follows:

Abbreviation	Description
.	Match any characters other than line breaks
\w	Match all alphanumeric characters: `[a-zA-Z0-9_]`
\W	Match non-alphanumeric characters: `[^\w]`

Abbreviation	Description
\d	Match numeric characters: `[0-9]`
\D	Match non-numeric characters: `[^\d]`
\s	Match space characters: `[\t\n\f\r\p{Z}]`
\S	Match non-space characters: `[^\s]`

Assertion

Lookbehind assertions and lookahead assertions are sometimes referred to as assertions, which are special types of ***non-capturing groups*** (used for matching pattern, but not included in the matching list). When we use this pattern before or after a particular pattern, we use assertions first.
For example, we want to obtain all the numbers before the character `$` in the input string `$4.44 and $10.88`. We can use this regular expression `(?<=\$)[0-9\.]*` to indicate: get all numbers before the character `$` with the character `.` included.
The followings are the assertions used in regular expressions:

Symbol	Description
?=	Positive lookahead assertion
?!	Negative lookahead assertion
?<=	Positive lookbehind assertion
?<!	Negative lookbehind assertion

Positive lookahead assertion

For positive lookahead assertions, the first part of the expression must be a lookahead assertion expression. The returned matching result only contains the text that matches the first part of the expression.
To define a positive lookahead assertion in brackets, the question mark and equal sign in brackets are expressed as `(?=...)`. The lookahead assertion expression is put after the equal sign in brackets.
For example, the regular expression `(T|t)he(?=\sfat)` indicates: matching uppercase letter `T` or lowercase letter `t`, which is followed by letters `h` and `e`.
In brackets, we define a positive lookahead assertion that leads the regular expression engine to match `The` or `the` which is followed by `fat`.

```
"(T|t)he(?=¥sfat)" => The fat cat sat on the mat.
```

Negative lookahead assertion

When we need to obtain the content mismatching the expression from an input string, we use a negative lookahead assertion. Negative lookahead assertion is defined in the same way as positive lookahead assertion.
The only difference is that we use negation symbol `!` instead of equal sign `=`, such as `(?!...)`.
Take a look at the following regular expression `(T|t)he(?!\sfat)`, which indicates: get all `The` or `the` mismatching `fat` from the input string, with a space character added before `fat`.

```
"(T|t)he(?!¥sfat)" => The fat cat sat on the mat.
```

Positive lookbehind assertion

Positive lookbehind assertions are used to obtain all matching content before a particular pattern. The positive lookbehind assertion is expressed as `(?<=...)`. For example, the regular expression `(?<=(T|t)he\s)(fat|mat)` indicates: get all the `fat` and `mat` behind the word `The` or `the` from the input string.

```
"(?<=(T|t)he¥s)(fat|mat)" => The fat cat sat on the mat.
```

Negative lookbehind assertion

Negative lookbehind assertions are used to obtain all matching content that are not before a particular pattern. Negative lookbehind assertions are expressed as `(?<!...)`. For example, the regular expression `(?<!(T|t)he\s)(cat)` indicates: get all the `cat` that are not behind `The` or `the` in the input characters.

```
"(?<!(T|t)he¥s)(cat)" => The cat sat on cat.
```

Label

Label modifies the output of the regular expression, which is also called modifier. The following labels can be used in any order or combination, and are part of a regular expression.

Label	Description
i	Case insensitive: Set the matching rule as case insensitive.
g	Global search: Search the entire input string for all matching content.

Label	Description
m	Multiline match: Match each line of the input string.

Case insensitive

The modifier `i` is used to perform case-insensitive matching. For example, the regular expression `/The/gi` indicates: the uppercase letter `T` followed by a lowercase letter `h` and a letter `e`.
But at the end of regular matching, the label `i` informs the regular expression engine to ignore it. As you can see, we also use the label `g` because we want to search the entire input string for matching content.

```
"The" => The fat cat sat on the mat.
```

```
"/The/gi" => The fat cat sat on the mat.
```

Global search

The modifier `g` is used to perform a global match (it finds all matching items, and will not stop until the first one is found).
For example, the regular expression `/.(at)/g` indicates: any characters other than line breaks followed by a lowercase letter `a` and a lowercase letter `t`.
Because we use the label `g` at the end of the regular expression, it finds each matching item from the entire input string.

```
".(at)" => The fat cat sat on the mat.
```

```
"/.(at)/g" => The fat cat sat on the mat.
```

Multiline match

The modifier `m` is used to perform a multiline match. As we discussed earlier about `(^, $)`, use a locator to check whether the matching character is the start or the end of an input string. However, we want to use a locator for each line, so we use the modifier `m`.
For example, the regular expression `/at(.)?$/gm` indicates: the lowercase letter `a`, followed by the lowercase letter `t` matching any character other than line breaks zero or one time. And because of the label `m`, the regular expression engine matches the end of each line in the string.

```
"/.at(.)?$/" => The fat
                cat sat
```

```
        on the mat.

"/.at(.)?$/gm" => The fat
                  cat sat
                  on the mat.
```

Common Regular Expression

Type	Expression
Positive integer	^-\d+$
Negative integer	^-\d+$
Phone number	^+?[\d\s]{3,}$
Phone code	^+?[\d\s]+(?[\d\s]{10,}$
Integer	^-?\d+$
User name	^[\w\d_.]{4,16}$
Alphanumeric characters	^[a-zA-Z0-9]*$
Alphanumeric characters with spaces	^[a-zA-Z0-9]*$
Password	^(?=^.{6,}$)((?=.*[A-Za-z0-9])(?=.*[A-Z])(?=.*[a-z]))^.*$
Email	^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4})*$
IPv4 address	^((?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))*$`
Lowercase letter	^([a-z])*$
Uppercase letter	^([A-Z])*$
User name	^[\w\d_.]{4,16}$
Website	^(((http|https|ftp):\/\/)?([[a-zA-Z0-9]\-\.])+(\.)([[a-zA-Z0-9]]){2,4}([[a-zA-Z0-9]\/+=%&_\.~?\-]*))*$
VISA credit card number	^(4[0-9]{12}(?:[0-9]{3})?)*$

Date (MM/DD/YYYY)	^(0?[1-9]|1[012])[- /.](0?[1-9]|[12][0-9]|3[01])[- /.](19|20)?[0-9]{2}$
Date (YYYY/MM/DD)	^(19|20)?[0-9]{2}[- /.](0?[1-9]|1[012])[- /.](0?[1-9]|[12][0-9]|3[01])$
MasterCard credit card number	^(5[1-5][0-9]{14})*$

Common Issues for Data Subscription

Last updated：2018-08-31 11:55:22

Does the data subscription allow multiple SDKs to be connected to and consume one channel at a time?

No. A channel can only be connected with and consumed by one SDK. If you have multiple downstream SDKs to subscribe to the same database table, you can build multiple channels.

Does the data subscription support connecting one SDK to multiple channels?

Yes. An SDK can consume multiple channels at a time.

An error occurred while starting SDK: already has sdk on this channel.

A channel can only be connected with and consumed by one SDK. If you add a new connection, this error will be reported. In this case, confirm whether the program completely exits. If the error occurs again during reconnection after confirmation, the interval between restarts may be slightly longer, such as 20 seconds.

When the data subscription subscribes to the real-time incremental data, is the new data only the added data or does it include modified data?

Data subscription can subscribe to the following incremental data: all additions, deletions and modifications (DML), and structure changes (DDL).

A TencentDB instance and a local database have the same table structure, but different indexes. Does the data subscription support real time synchronization?

Yes. If the data subscription only subscribes to data changes, consumption will not be affected by different indexes. If it subscribes to structure changes, and indexes will change on the TencentDB instance, the structure changes may fail to be consumed locally due to different indexes.

Why can't I modify the consumption time point of a data subscription channel?

When an error occurred while modifying a consumption time point, a prompt will appear on the interface. It is generally because the subscription channel is consumed by a downstream SDK. You can check the consumption source IP on the DTS console to see if there is a downstream SDK consuming data. If yes, stop the consumption and then modify the consumption time point.

How can I determine whether the data is consumed normally?

When data is written into a channel (or some data is not consumed), if the data is consumed normally, the consumption time point on the console will be migrated normally.

If a record at the consumer end is not acknowledged by the data subscription, why SDK receives duplicate data after restart?

When SDK has messages that have not been acknowledged, the SDK will continue pulling messages until its cache is full, and no new message will be received. At this point, the consumption time point saved on the server is that of the message not acknowledged.
When SDK is restarted, the server will push data again from the consumption time point of the message not acknowledged to avoid message loss. Therefore, the SDK will receive some repeated messages.

When SDK quits and restarts after a few days, why does it fail to subscribe to data? Error: Maybe checkpint is too old?

The time range for data retention in a data subscription channel is 1 day, i.e. from [the current time - the current time of the next day]. The subscription channel will delete expired data. Therefore, if the data corresponding to the time point of the last consumption data when SDK quits is not within the valid data range of the current subscription channel, the data corresponding to this consumption time point cannot be subscribed to. To fix this problem, modify the consumption time point before starting SDK, to ensure it is within the valid data range.

When SDK pulls data, it suddenly stops and cannot subscribe to any data. But after restarting, it consumes some data before stopping again.

In this case, it is more likely that the API ackAsConsumed is not called in the SDK code to report the consumption time point. Because the SDK has a limited cache space, if ackAsConsumed is not called to report the consumption time point, the data in the cache space will not be deleted. When the cache is full, new data cannot be pulled, and the SDK will stop and fail to subscribe to any data. Note: All messages here, including BEGIN and COMMIT messages, must be acknowledged for consumption. Messages unrelated to business logic are also included.

How to ensure that the data subscribed to by SDK is a complete transaction, and will the record in the middle of the transaction be pulled based on the provided consumption time point?

No. Based on the user-specified consumption time point or the time point of the last acknowledged consumption, the server will search for the start point of the complete transaction corresponding to this consumption time point. Data is sent to the downstream SDK from the beginning of the entire transaction. So the full transaction content can be received.

Is there any problem with the data subscription during the TencentDB master/slave switch or when the master database is restarted? Will the data be lost?

No. When a switching between master and slave occurs or when TencentDB instance is restarted, the data subscription will automatically perform switching. This process is transparent to the SDK.

An error occurred while starting SDK: Do DTS authentication fail, caused by: get channel info from msg failed.

Confirm whether the input parameters are matched, including ip, port, secretId, secretKey, and channelId.

When SDK is started, why does the system report that secretId has no permission?

Sub-account has no permission by default. It must be given the access to the operation "name/dts:AuthenticateSubscribeSDK", or the access to all DTS operations "QcloudDTSFullAccess" by the root account.

Will the data subscription receive duplicate data?

No if data is consumed normally. If SDK quits abnormally, the information of the last acknowledgement time point is not reported timely, and duplicate data may be received when the SDK is started next time. But the probability is very low.
If a complete transaction is not acknowledged, the data is pulled again from the beginning of the transaction when the SDK is started next time. In this case, the data cannot be regarded as duplicate data. The core logic of the SDK guarantees the integrity of the transaction.

Can a data subscription instance subscribe to multiple TencentDB instances?

No. A data subscription channel can subscribe to only one TencentDB instance.

What if OOM occurs while SDK is running?

Choose a host with better configurations. When a single SDK runs smoothly at high speed, it consumes less than 1-core CPU and less than 1.5 GB of memory.

Common Issues

Last updated：2018-08-31 11:55:27

What is DTS?

TencentDB Service for Transmission (DTS) provides database data transfer service integrated with data migration, data synchronization and data subscription features, helping you achieve database migration without downtime. It also allows you to use a real-time synchronization channel to easily build a highly available database architecture which supports remote disaster recovery.

How to use Tencent Cloud DTS?

You can use Tencent Cloud DTS to migrate all your data to the Tencent Cloud-based database at a time, or you can perform continuous data replication. Tencent Cloud DTS captures changes to the source database, and applies them to the destination database in a transaction-consistent manner. For information on how to perform continuous data replication, please see Data Subscription.

Can DTS support data migration between TencentDB instances under two different Tencent Cloud accounts?

Yes. For the migration between TencentDB instances cross Tencent Cloud accounts, log in to the DTS with the Tencent Cloud account to which the destination TencentDB instance belongs. And for the source instance type, select the self-built database with public IP.

Which sources and destinations does Tencent Cloud DTS support?

Tencent Cloud DTS supports MySQL with public IP, MySQL built on CVM, MySQL with Direct Connect, MySQL with VPN access, source database for TencentDB for MySQL, and destination database for TencentDB for MySQL.

Can the progress of database migration tasks be monitored?

Yes. You can see the progress of the migration tasks on the Tencent Cloud DTS console management page.

When I use the Tencent Cloud DTS for data migration, will the data of source database be deleted after migration?

No. When DTS performs data migration, it actually copies the data of the source database, without any effect on it.

Does Tencent Cloud DTS support timed automatic migration?

Yes. After you create Tencent Cloud DTS, you can select the option of timed execution when modifying the configuration, and select the time for timed migration.

What version of CRS is applicable to migration?

Source instances of version 3.2 or above are not supported for migration.

What version of MySQL is applicable to migration?

MySQL 5.1/5.5/5.6 is supported for the migration of data to the cloud. Since MySQL 5.1 is no longer supported by Tencent Cloud TencentDB, we recommended that you update MySQL 5.1 to MySQL 5.5 before migrating data to TencentDB for MySQL 5.5. You can also use the DTS data migration tool to directly migrate data from local MySQL 5.1 to Tencent Cloud TencentDB for MySQL 5.5.

How to check the cause for connectivity failure?

Click **Click to View Details** to view the solution.

Why is the destination instance unavailable?

The destination instance cannot be used when:

1. The destination instance is not initialized.
2. The destination instance is locked by other tasks.
3. The destination instance has data.
4. The size of data used by the source instance is greater than the capacity of destination instance.

Why does a warning occur when I verify a task?

You can click **View Details** on the right of the warning item to view the cause and solution.

What causes the error during migration that results in the failure of the migration task?

1. Failed to bgsave the source instance during the migration process.
2. During the migration process, the volume of data written into the source instance is too large and exceeds the configured sync BUFFER, which causes the sync connection to reconnect. The migration task will continue to retry the connection and generate RDB.