Data Transfer Service

SDK Documentation

Product Documentation

Contents

SDK Documentation
SDK Release Logs

Last updated：2023-11-21 10:33:03

Note

This SDK is used to consume data subscribed from the legacy data subscription feature, which is no longer unavailable for sale. If you need to create a subscription task for data consumption, use the data subscription feature (Kafka edition). For more information, see Data Subscription (Kafka Edition) and Data Subscription Guide (Kafka Edition).

Version 2.9.1

1. Fixed bugs.

Version 2.9.0

1. Supported MySQL 5.7.
2. Supported other encodings in addition to UTF-8.
3. Supported distinction between NULL and empty string.
4. Supported the BLOB field.

Version 2.8.2

1. Optimized SDK memory usage.
2. Supported the VPC authentication, allowing you to use the SDK over both private and public networks.

Version 2.8.0

1. Optimized the internal authentication logic.
2. Reduced the number of user parameters that need to be configured.

Version 2.7.2

1. Filled the `db` value in the DDL statement result.

Version 2.7.0

1. Supported seconds-level HA switch at the data subscription channel backend.
2. Added the SDK monitoring metric report feature.

Version 2.6.0

1. Supported subscribing to multiple channels via a single SDK.

2. Supported subscribing to "stop", "start" and other operations on the client.
3. Supported the serialization of `DataMessage.Record` .
4. Optimized the SDK performance and reduced the resource consumption.

Version 2.5.0

1. Fixed bugs occurred with a small probability in high-concurrency scenarios.
2. Supported the globally unique auto-increment ID recorded in transactions.

Version 2.4.0

1. Optimized the subscription logic by working with the backend to accurately display SDK's current consumption time point.
2. Fixed the issue that occurred while encoding a few special characters at the backend.
3. Fixed multiple compatibility issues. We recommend that you upgrade the SDK to this version as soon as possible if you're still using the earlier versions.

Data Subscription SDK

Last updated：2023-11-21 10:35:50

Note

The data subscription feature of the legacy version has been deactivated. If you need to create a subscription task and consume data, use the data subscription feature (Kafka edition). For more information, see Data Subscription (Kafka Edition) and Data Subscription Guide (Kafka Edition).

Downloading the Data Subscription SDK

Download the data subscription SDK v2.9.1 here.

How It Works

Pull

Message pull and acknowledgment in the SDK are performed by two async threads simultaneously. Messages are pulled in sequence. The two threads are executed independently and strictly in order, but they are async.
The registered `notify` function will be called for the pulled messages in sequence, and the SDK ensures that each message will be pushed once and only once. If the `m.ackAsConsumed()` function is not called, `notify` will still be called for messages, as pull and acknowledgment are async.

Acknowledgment mechanism

The SDK adopts the incremental acknowledgment mechanism, where all messages (including `BEGIN` and `COMMIT` messages) must be acknowledged but can be acknowledged repeatedly.
For example, if the client receives `1`, `2`, `3`, `4`, and `5` messages but calls the `m.ackAsConsumed()` function only for `1`, `2`, and `5`, the SDK will acknowledge the consumption of `1` and `2` to the server. If the client fails at this point, the SDK will obtain messages starting from message `3`.
As message pull and acknowledgment are async, the SDK will keep pulling new messages and notifying the client even if some messages are not acknowledged. However, if the number of unacknowledged messages exceeds a threshold (currently `8,000`), the SDK will stop pulling new messages.
Each message has a unique `record_id` and `checkpoint`, and the SDK actually acknowledges a message by acknowledging its `checkpoint`.

Runtime Environment Requirements

Java environment: JRE 1.6 or later.
The SDK needs to run on a CVM instance in the same VPC in the same region as the subscribed instance; otherwise, interconnection should be configured.
To access the SDK over the public network, the CVM instance can be used for port forwarding, but **the bandwidth and performance cannot be guaranteed** due to heavy dependence on the public network bandwidth.

Sample Code

Note
We recommend that you call the SDK by using the sub-account key and environment variables to improve the SDK security. For more information, see Access Key. When authorizing sub-accounts, follow the principle of least permission to prevent the leakage of resources beyond the destination storage bucket or objects.
If you must use a permanent key, we recommend that you follow the principle of least permission to limit the permission scope of the permanent key.
Below is the sample code for Tencent Cloud binlog subscription:

```
package com.qcloud.biz;
import com.qcloud.dts.context.NetworkEnv;
import com.qcloud.dts.context.SubscribeContext;
import com.qcloud.dts.message.ClusterMessage;
import com.qcloud.dts.message.DataMessage;
import com.qcloud.dts.subscribe.ClusterListener;
import com.qcloud.dts.subscribe.DefaultSubscribeClient;
import com.qcloud.dts.subscribe.SubscribeClient;
import java.util.List;
public class Main {
    public static void main(String[] args) throws Exception {
```

```
        // Create a context
        SubscribeContext context=new SubscribeContext();
        // User `secretId` and `secretKey`. We recommend that you use a sub-account
        context.setSecretId("AKID-522dabxxxxxxxxxxxxxxxxxx");
        context.setSecretKey("AKEY-0ff4cxxxxxxxxxxxxxxxxxxx");
        // Specify the channel region, which is required for SDKs on version 2.8.0
        // For more information on region values, visit https://cloud.tencent.com/d
        context.setRegion("ap-chongqing");
        // Subscribed `serviceIp` and `servicePort`
        // Note that the `IP` and `Port` parameters are required for SDKs on versio
        // context.setServiceIp("10.xx.xx.24");
        // context.setServicePort(50120);

        // Set the network environment to private network if the CVM instance where
        context.setNetworkEnv(NetworkEnv.LAN);
        // Create a client
        SubscribeClient client=new DefaultSubscribeClient(context);
        // Create a subscription listener
        ClusterListener listener= new ClusterListener() {
            @Override
            public void notify(List<ClusterMessage> messages) throws Exception {
                // Consume the subscribed data
                for(ClusterMessage m:messages){
                  for(DataMessage.Record.Field f:m.getRecord().getFieldList()){

                      if (f.getType().equals(DataMessage.Record.Field.Type.BLOB)){
                          System.out.println("["+f.getType()+"]["+f.getFieldname()+
                          byte[] theRawBytesValue = f.getValueAsBytes();
                      }if(f.getType().equals(DataMessage.Record.Field.Type.INT8)){
                          // If this value is null, `f.getValueAsInteger()` will re
                          System.out.println(f.getValueAsInteger());
                      }if(f.getType().equals(DataMessage.Record.Field.Type.JSON)){
                          // JSON data can be returned only if the source instance
                          System.out.println(f.getValueAsString());
                      }if(f.getType().equals(DataMessage.Record.Field.Type.STRING))
                          // If this value is null, `f.getValueAsString()` will ret
                          System.out.println(f.getValueAsString());
                          // The original encoding of the field
                          System.out.println(f.getFieldEnc());
                      }
                      else{
                          // The `f.getValue()` method will be disused soon.
                          String value = f.getValue() == null ? "Null": f.getValue(
                          String msg = "["+f.getType()+"]"+f.getFieldname()+"[encod
                          System.out.println(msg);
                      }
```

```
                    }
                    // Acknowledge the consumption
                    m.ackAsConsumed();
                }
            }
            @Override
            public void onException(Exception e){
                System.out.println("listen exception"+e);
            }};
        // Add a listener
        client.addClusterListener(listener);
        // Configure the requested subscription channel
        client.askForGUID("dts-channel-r0M8kKsSyRZmSxQt");
        // Start the client
        client.start();
    }
}
```

The entire process is an intuitive, typical producer-consumer pattern. As a consumer, the SDK constantly pulls the subscribed binlog data from the server, consumes the data, and then acknowledges data consumption.

1. Configure parameters and create a consumer client `SubscribeClient`.
2. Create a listener `ClusterListener`, consume the received binlog data, and return an acknowledgment after consumption.
3. Start the client to start the process.

The `ClusterListener` listener allows you to manipulate the received binlog data as needed and filter it by type; for example, you can filter out all `drop` statements.

In the sample code, you need to provide five parameters.

`secretId` and `secretKey` are values of keys associated with your Tencent Cloud account, which can be viewed in **Access Key**> **API Key Management** in the **CAM** console. The SDK uses these two parameters to authenticate your operations.

Note

The data subscription SDK has been connected to CAM. By default, a root account has all permissions, and you can use its TencentCloud API key to access the SDK. A sub-account has no permission, and it must be granted access to the `name/dts:AuthenticateSubscribeSDK` operation or all DTS operations through the `QcloudDTSFullAccess` policy by the root account.

`serviceIp`, `servicePort`, and `channelId` parameters are related to binlog subscription and can be viewed on the **Data Subscription** page in the DTS console after the subscription is configured in the TencentDB for MySQL console.

Note

`serviceIp` is the IP in the **service address**, `servicePort` is the port number in the **service address**, and `channelId` is the **channel ID** in the DTS console.

SDK Description

`SubscribeContext` class

Class description

This class is used to set your SDK configuration information, including security credentials (`secretId` and `secretKey`) and the IP address and port of the subscription service.

Construction method

public SubscribeContext()

Class method

Setting the security credential `secretId`

Function prototype

public void setSecretId(String secretId)

Input parameters

Parameter	Type	Description
secretId	String	Security credential `secretId` , which can be viewed in **Access Key** > **API Key Management** in the **CAM** console.

Returned result

N/A

Reported exception

N/A

Setting the security credential `secretKey`

Function prototype

public void setSecretKey(String secretKey)

Input parameters

Parameter	Type	Description
secretKey	String	Security credential `secretKey` , which can be viewed in **Access Key** > **API Key Management** in the **CAM** console.

Returned result

N/A

Reported exception

N/A

Setting the subscription service IP address

Function prototype

public void setServiceIp(String serviceIp)

Input parameters

Parameter	Type	Description
serviceIp	String	Subscription service IP address, which can be viewed on the subscription channel configuration page in the console.

Returned result

N/A

Reported exception

N/A

Setting the subscription service port

Function prototype

public void setServicePort(String servicePort)

Input parameters

Parameter	Type	Description
servicePort	String	The port number of the subscription service, which can be viewed on the subscription channel configuration page in the console.

Returned result

N/A

Reported exception

N/A

`SubscribeClient` and `DefaultSubscribeClient` APIs

Class description

The `DefaultSubscribeClient` class implements the `SubscribeClient` API.

This class is used to construct the subscription SDK client, that is, the consumer of binlog messages.

`DefaultSubscribeClient` provides sync and async acknowledgment implementations for different user needs.

In sync mode, an acknowledgment is synchronously returned each time the client consumes a binlog message, ensuring that message consumption acknowledgments can be received by the server as soon as possible. However,

the overall SDK performance is not as good as in async mode. In async mode, the consumer acknowledges messages asynchronously, that is, messages are pulled and acknowledged asynchronously. You can select either mode as needed.

Construction method

Constructing `DefaultSubscribeClient`

Function prototype

public DefaultSubscribeClient(SubscribeContext context, boolean isSync) throws Exception

Input parameters

Parameter	Type	Description
context	SubscribeContext	Your SDK configuration information
isSynce	boolean	Whether the sync consumption mode is used for the SDK

Returned result

`DefaultSubscribeClient` instance.

Reported exception

`llegalArgumentException` : This exception will be reported if any parameter is invalid in the parameter context you submitted, such as missing or incorrectly formatted security credentials, service IP, or port.

`Exception` : This exception will be reported in case of an internal error during SDK initialization.

Constructing `DefaultSubscribeClient`

Function prototype

public DefaultSubscribeClient(SubscribeContext context) throws Exception

Input parameters

Parameter	Type	Description
context	SubscribeContext	Your SDK configuration information

Returned result

`DefaultSubscribeClient` instance, which uses the async message acknowledgment mode by default.

Reported exception

`llegalArgumentException` : This exception will be reported if any parameter is invalid in the parameter context you submitted, such as missing or incorrectly formatted security credentials, service IP, or port.

`Exception` : This exception will be reported in case of an internal error during SDK initialization.

Class method

Adding a listener to the SDK consumer client

Function description

This function is used to add the `ClusterListener` listener to `SubscribeClient` to subscribe to the incremental data in the channel.

Function prototype

public void addClusterListener(ClusterListener listener) throws Exception

Input parameters

Parameter	Type	Description
listener	ClusterListener	The listener used for the consumer client. The main process for binlog data consumption should be implemented in `ClusterListener` .

Returned result

N/A

Reported exception

`IllegalArgumentException` : This exception will be reported if the submitted `listener` parameter is empty.

`Exception` : This exception will be reported if more than one listeners are added to the SDK.

Requesting the incremental data in a subscription channel

Function prototype

public void askForGUID(String channelId)

Input parameters

Parameter	Type	Description
channelId	String	Subscription channel ID, which can be viewed on the subscription channel configuration page in the console.

Returned result

N/A

Reported exception

N/A

Starting the SDK client

Function prototype

public void start() throws Exception

Input parameters

N/A

Returned result

N/A

Reported exception

`Exception` : This exception will be reported in case of an internal error during SDK startup.

Stopping the SDK client

Function prototype

public void stop(int waitSeconds) throws Exception

public void stop() throws Exception

Input parameters

Parameter	Type	Description
waitSeconds	int	Wait time in seconds before the SDK is forced to stop.

Here, the `stop` function without the timeout parameter will wait until the thread stops, which may take a long time subject to system scheduling; therefore, we recommend that you use the `stop` function with the timeout parameter in scenarios where the specific restart time is required.

Returned result

N/A

Reported exception

`Exception` : This exception will be reported in case of an internal error during SDK stop.

`ClusterListener` API

API description

This is a callback API. You need to implement the `notify` function of this API to consume the subscribed data and implement the `onException` function to handle any possible exceptions during consumption.

API function

Notifying the SDK of consuming messages subscribed by the client

Function description

This function is used to consume incremental data. However, the SDK will notify the `ClusterListener` of data consumption via the `notify` function when the data is received.

Function prototype

public abstract void notify(List<ClusterMessage> messages) throws Exception

Input parameters

Parameter	Type	Description

messages	List<ClusterMessage>	Subscribed data array. For more information on `ClusterMessage` implementation, see its definition.

Returned result

N/A

Reported exception

Any exception during the consumption of the subscribed data will be reported to the implemented `onException` function for custom handling as needed.

Handling exceptions while consuming the subscribed data

Function description

This function is used to handle exceptions during subscribed data consumption. You can implement your safe exit policy in `onException` .

Function prototype

public abstract void onException(Exception exception)

Input parameters

Parameter	Type	Description
exception	Exception	`Exception` class in the Java standard library

Returned result

N/A

Reported exception

N/A

`ClusterMessage` class

Class description

The `ClusterMessage` class delivers the consumed data through the `notify` function. Each `ClusterMessage` saves the data records of a **transaction** in TencentDB for MySQL, and each record in the transaction is saved via `Record` .

Class method

Obtaining records from `ClusterMessage`

Function prototype

public Record getRecord()

Input parameters

N/A

Returned result

Type	Description
Record	Change record, which corresponds to a specific record in a transaction, such as `begin` , `commit` , `update` , and `insert` .

Reported exception

N/A

Acknowledging consumed data

Function description

This function is used to send an acknowledgment to the subscription server about the consumed data synchronously or asynchronously based on the value set in `SubscribeClient` . This function must be called after consumption; otherwise, the SDK will receive duplicate data due to incorrect logic.

Note

The SDK must call `ackAsConsumed` for all received messages, including those the business logic may not care about; otherwise, the SDK will stop pulling new data after a certain number of messages remain unacknowledged.

Function prototype

public void ackAsConsumed() throws Exception

Input parameters

N/A

Returned result

N/A

Reported exception

`Exception` : This exception will be reported in case of an internal error during acknowledgment.

`Record` class

Class description

It indicates a certain record in the subscribed binlog data, which is usually a member of a transaction's `ClusterMessage` . A record can be a `begin` , `commit` , `update` , or .

Class method

Obtaining the attribute value of a record

Function prototype

public String getAttribute(String key)

Input parameters

Parameter	Type	Description

key	String	Attribute value name

Below are possible attribute key values:

Attribute Key Value	Description
record_id	Record ID, which is an auto-increment string in a channel but is not necessarily continuous.
source_type	The engine type of the database instance of the record, which is `mysql` currently.
source_category	Record type, which is `full_recorded` currently.
timestamp	The time when the record is stored into binlog, which is also the time when the SQL statement is executed in TencentDB.
sdkInfo	The offset of the binlog file of the record in the format of `file_offset@file_name`, where `file_name` is the numeric suffix of the binlog file.
record_type	The operation type of the record, mainly including `insert`, `update`, `delete`, `replace`, `ddl`, `begin`, `commit`, and `heartbeat`.
db	The database name of the record update table, which is empty for a DDL record.
table_name	The name of the record update table, which is empty for a DDL record.
record_encoding	Record encoding
primary	The name of the primary key column of the record update table
fields_enc	The encoding of each field value of the record, which is empty if the type is not character. Multiple fields are separated by comma.
gtid	The GTID of the transaction of the record

Returned result

Type	Description
String	Attribute value

Reported exception

N/A

Obtaining the change type of a record

Function prototype

public DataMessage.Record.Type getOpt()

Input parameters

N/A

Returned result

Type	Description
DataMessage.Record.Type	Record type, which can be `insert`, `delete`, `update`, `replace`, `ddl`, `begin`, `commit`, or `heartbeat`. `heartbeat` is the heartbeat table internally defined by DTS to check whether the subscription channel is healthy. In theory, a heartbeat is generated per second.

Reported exception

N/A

Obtaining the checkpoint recorded in the binlog

Function prototype

public String getCheckpoint()

Input parameters

N/A

Returned result

Type	Description
String	The checkpoint of a record in the binlog in the format of `binlog_offset@binlog_fid`. Here, `binlog_offset` is the change record offset in the binlog file, and `binlog_fid` is the binlog filename.

Reported exception

N/A

Obtaining the timestamp of a record in the binlog

Function prototype

public String getTimestamp()

Input parameters

N/A

Returned result

Type	Description
String	Timestamp string

Reported exception

N/A

Obtaining the database name of a record

Function prototype

public String getDbname()

Input parameters

N/A

Returned result

Type	Description
String	Database name string

Reported exception

N/A

Obtaining the data table name of a record

Function prototype

public String getTableName()

Input parameters

N/A

Returned result

Type	Description
String	Data table name string

Reported exception

N/A

Obtaining the primary key column name of a record

Function prototype

public String getPrimaryKeys()

Input parameters

N/A

Returned result

Type	Description
String	Primary key column name. Separate multiple names by semicolon for composite primary keys.

Reported exception

N/A

Obtaining the database type of a subscribed instance

Function prototype

public DBType getDbType()

Input parameters

N/A

Returned result

Type	Description
DBType	Currently, DTS only supports `DBType.MYSQL` , that is, TencentDB for MySQL.

Reported exception

N/A

Obtaining the number of fields in a record

Function prototype

public int getFieldCount()

Input parameters

N/A

Returned result

Type	Description
int	The number of fields in the record, which is the same as or twice (for update record) that of columns in the corresponding table.

Reported exception

N/A

Checking whether a record is the first record in the transaction

Function prototype

public Boolean isFirstInLogevent()

Input parameters

N/A

Returned result

Type	Description
Boolean	If it is the first record in the transaction, `True` is returned; otherwise, `False` is returned.

Reported exception

N/A

Obtaining the field definition list in the table of a record

Function prototype

public List<Field> getFieldList()

Input parameters

N/A

Returned result

Type	Description
List<Field>	Field array. For more information, see the definition of the `Field` class.

Note

For `INSERT` records, the `Field` values in `List` follow the sequence defined by the subscribed table. Values of records in `Field` are inserted values, that is, post-images.

For `DELETE` records, the `Field` values in `List` follow the sequence defined by the subscribed table. Values of records in `Field` are values before deletion, that is, pre-images.

For `UPDATE` records, `List` contains values before and after modification, that is, pre-images and post-images. Pre-images (values before modification) are in even positions in the `List` , while post-images are in odd positions. The list of pre-images and post-images also follows the sequence defined by the subscribed table. Therefore, the number of `Field` values in `List` is twice that of columns in the corresponding subscribed table.

Reported exception

N/A

`Field` class

Class description

The `Field` class defines the attributes of a field such as encoding, type, name, value, and primary key status.

Class method

Obtaining the encoding format of a field

Function prototype

public String getFieldEnc()

Input parameters

N/A

Returned result

Type	Description

String	Field encoding, which is a string.

Reported exception

N/A

Obtaining the field name

Function prototype

public String getFieldname()

Input parameters

N/A

Returned result

Type	Description
String	Field name, which is a string.

Reported exception

N/A

Obtaining the data type of a field

Function prototype

public Field.Type getType()

Input parameters

N/A

Returned result

Type	Description
Field.Type	`Field.Type` is an enumeration type corresponding to the data types supported by MySQL, including `INT8` , `INT16` , `INT24` , `INT32` , `INT64` , `DECIMAL` , `FLOAT` , `DOUBLE` , `NULL` , `TIMESTAMP` , `DATE` , `TIME` , `DATETIME` , `YEAR` , `BIT` , `ENUM` , `SET` , `BLOB` , `GEOMETRY` , `STRING` , and `UNKNOWN` .

Reported exception

N/A

Obtaining the field value

Function prototype

public ByteString getFieldname()

Input parameters

N/A

Returned result

Type	Description
ByteString	Field value, which is NULL if empty.

Reported exception

N/A

Checking whether a field is a primary key

Function prototype

public Boolean isPrimary()

Input parameters

N/A

Returned result

Type	Description
Boolean	If the field is a primary key, `True` is returned; otherwise, `False` is returned.

Reported exception

N/A

SDK Upgrade

Last updated：2023-11-21 10:40:36

Note

The data subscription feature of the legacy version has been deactivated. If you need to create a subscription task for data consumption, use the data subscription feature (Kafka edition). For more information, see Data Subscription (Kafka Edition) and Data Subscription Guide (MySQL).

Data subscription APIs on v2.0 will become unavailable. Data subscription SDK 2.8.0 and earlier versions use API 2.0 for authentication, and thus will be affected after API 2.0 is deprecated.

Follow the steps below to upgrade the applications you have consumed through the SDK in time to avoid the impact of API 2.0 deprecation.

Note

We recommend that you call the SDK by using the sub-account key and environment variables to improve the SDK security. For more information, see Access Key. When authorizing sub-accounts, follow the principle of minimum permission to avoid leaking resources other than the destination buckets and objects.

If you must use a permanent key, we recommend that you follow the principle of minimum permission to limit the permission scope of the permanent key.

Step 1. Check the SDK version

Check the version of the used SDK you have downloaded from Data Subscription SDK.

Check the SDK version as follows:

Check the package name of the SDK you have used, which contains the version information, such as `binlogsdk-2.8.0-jar-with-dependencies.jar`.

Check the value of `SDK_VERSION`. If the SDK package has been renamed, run the `unzip` command to decompress the package and check the value of `SDK_VERSION` in the `tencentSubscribe.properties` file.

If the data subscription SDK version is 2.8.0 or earlier, proceed to step 2.

Step 2. Upgrade the SDK

1. Download the latest SDK from Data Subscription SDK and replace SDK 2.8.0 and earlier versions with it.
2. After the replacement, refer to the following code modifications to add a line of code to set `region` for the subscription channel.

```
package com.qcloud.biz;
import com.qcloud.dts.context.NetworkEnv;
import com.qcloud.dts.context.SubscribeContext;
import com.qcloud.dts.message.ClusterMessage;
import com.qcloud.dts.message.DataMessage;
import com.qcloud.dts.subscribe.ClusterListener;
import com.qcloud.dts.subscribe.DefaultSubscribeClient;
import com.qcloud.dts.subscribe.SubscribeClient;
import java.util.List;
public class Main {
    public static void main(String[] args) throws Exception {
```

```
        SubscribeContext context=new SubscribeContext();
        // User `secretId` and `secretKey`. We recommend that you use a sub-account
        context.setSecretId("AKID-522dabxxxxxxxxxxxxxxxxx");
        context.setSecretKey("AKEY-0ff4cxxxxxxxxxxxxxxxxxxx");
        /*****************Code modification starts****************/
        // If you do not set the `region` parameter, the API 2.0 will still be used
        // However, the authentication via API 2.0 will fail then, so the SDK will
        // Set `region` for subscription channel.
        context.setRegion("ap-chongqing");
        /*****************Code modification ends******************/


        // Create a client
        SubscribeClient client=new DefaultSubscribeClient(context);
        // Create a subscription listener
        ClusterListener listener= new ClusterListener() {
            @Override
            public void notify(List<ClusterMessage> messages) throws Exception {
                // Consume the subscribed data
                for(ClusterMessage m:messages){
                    for(DataMessage.Record.Field f:m.getRecord().getFieldList()){
                        if(f.getFieldname().equals("id")){
                            System.out.println("seq:"+f.getValue());
                        }
                        DataMessage.Record record  = m.getRecord();
                    }
                    // Acknowledge the consumption
                    m.ackAsConsumed();
                }
            }
            @Override
            public void onException(Exception e){
                System.out.println("listen exception"+e);
            }};
        // Add a listener
        client.addClusterListener(listener);
        // Configure the requested subscription channel
        client.askForGUID("dts-channel-r0M8kKsSyRZmSxQt");
        // Start the client
        client.start();
    }
}
```