**RocketMQ Connect JDBC Connector**

# **About Me**

* Name: Yuchen Li
* Mentor: Du Heng, Von Gosling
* Major: Computer Science and Technology
* University: Southern University of Science and Technology
* Email: [yuchenlichuck@gmail.com](mailto:yuchenlichuck@gmail.com)
* Github: <https://github.com/yuchenlichuck>
* HomePage: <http://liyc.xyz>
* Phone: (+86) 133-3914-5791
* Postal Address: No. 1088 Xueyuan Road, Nanshan District, Shenzhen, China
* Time Zone: UTC+08:00 (China)
* Hobby Activities: Open source, Programming, Tennis and Guitar

As for me, I am a sophomore in the computer science and engineering department of SUSTech. I love open source and I have taken part in plenty of projects, which makes me good at communicating with others and expressing my ideas properly.

I am familiar with Java, I have learned lots of knowledge about JDBC and SQL Databases. And This semester, I am learning Distributed System, which has some chapters about distributed Message Queue. I really enjoy the process of challenge myself with new things. I am a heavy user of GitHub and know the workflow in GitHub.

Besides, I have strong software engineering skills, once worked as an intern at Tencent analysis problem stack and also developed the software about image segmentation in my professor’s laboratory.

# **Synopsis**

# **Background**

Apache RocketMQ is an open source distributed messaging and streaming data platform.

In current version of RocketMQ project, different cluster can't exchange messages each other.

Although dLedger structure improves availability of rocketmq much more, it still can't implement backuping messages for different cluster.

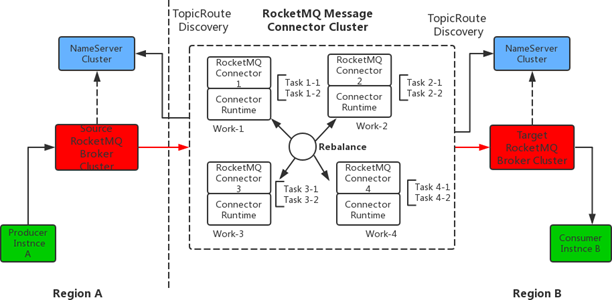
So, Message Connector is a mechanism to implement backuping messages for different cluster. It can improve availability of multiple RocketMQ cluster. Kafka Connectors to copy data between Kafka and other systems that you want to pull data from or push data to. Source Connectors import data from another system. Sink Connectors export data.

I will reuse the standard connect interface of OpenMessaging and the Connector Runtime environment

# **Design/description of Work**

The Design of RocketMQ Message Connector is mainly based on the standard connect interface of OpenMessaging and the Connector Runtime environment

**The overall architecture design of message connector**



- **RocketMQ Woker Example Model**

The Definition of Worker:

Connectors and tasks are logical units of work and must be scheduled to execute in a process. RocketMQ Connect calls these processes workers and has two types of workers: standalone and distributed. Standalone mode is the simplest mode, where a single process is responsible for executing all connectors and tasks. Distributed mode provides scalability and automatic fault tolerance for Kafka Connect.

- **The Case Design of RocketMQ Connector**

The Definition of Connector:

A connector instance is a logical concept that is responsible for maintenance, related configuration of a particular data system, such as a link address, what information needs to be synchronized, etc.

- **The Case Design of RocketMQ Routing Task**

The definition of Task:

Perform specific data parsing and dump tasks, and assign them to connect to specific source/sink data fragments.

a. SourceTask: From the source data system, the execution completion data parsing work is exposed to the connector through the poll interface.

b. SinkTask: The connector retrieves data from memory and parses it to the target data source system via the put interface method.

**JDBC Connector**

### RocketMQ JDBC Source Connector

The JDBC source connector is to import data from any relational database into RocketMQ topics using the JDBC driver. By using JDBC, this connector can support various databases without having to customize the code for each database.

Load data by periodically executing SQL queries and creating output records for each row in the result set. By default, all tables in the database are copied, each with its own output topic. Monitor the database to find new or deleted tables and adjust automatically. When replicating data from a table, the connector can only load new rows or modified rows by specifying which columns should be used to detect new data or modified data.

We can configure Java stream applications to deserialize and receive data in a variety of ways, including the RocketMQ console generator, the JDBC source connector, and the Java client generator.

Here are some features of RocketMQ JDBC Sourece Connector:

**Incremental Query Modes**

Each incremental query pattern tracks a set of columns for each row to track which rows have been processed and which rows are new or updated. I will write the mode setting to control this behavior and support the following options:

**Incrementing Column:** A single column containing a unique ID for each row, where newer rows are guaranteed to have larger IDs

**Timestamp Column:** A single column containing a modification timestamp is used to track the last time data was processed and to query only for rows that have been modified since that time.

**Timestamp and Incrementing Columns:** This is the most robust and accurate mode, combining an incrementing column with a timestamp column. By combining the two, as long as the timestamp is sufficiently granular, each (id, timestamp) tuple will uniquely identify an update to a row.

**Custom Query:** With a custom query, one of the other update automatic update modes can be used as long as the necessary WHERE clause can be correctly appended to the query. Alternatively, the specified query may handle filtering to new updates itself; however, note that no offset tracking will be performed

**Bulk:** This mode is unfiltered and therefore not incremental at all. It will load all rows from a table on each iteration.

**Mapping Column Types**

The source connector will have a few options for controlling how column types are mapped into Connect field types. By default, the connector maps SQL/JDBC types to the most accurate representation in Java, which is straightforward for many SQL types but maybe a bit unexpected for some types

**JDBC Drivers**

The source connector implements the data copying functionality on the generic JDBC APIs, but relies on JDBC drivers to handle the database-specific implementation of those APIs. Confluent Platform ships with a few JDBC drivers

**Message Keys**

Apache RocketMQ messages are key/value pairs. the value (payload) is the contents of the table row being ingested.

### JDBC Sink Connector

The JDBC sink connector is to export data from RocketMQ topics to any relational database using the JDBC driver. By using JDBC, this connector can support various databases without providing a dedicated connector for each database. Connectors poll data from RocketMQ for writing to the database based on subject subscriptions. Idempotent writing can be implemented with upserts. It also supports automatic table creation and limited automatic evolution.

Here are some features of RocketMQ JDBC Sink Connector:

**Data mapping**

data mapping is the process of creating data element mappings between two distinct data. In this feature, I will write primary key mode and fileds.

**Key handling**

Write different modes that enable to use fields from the RocketMQ record key, record value, or coordinates for the record.

**Idempotent writes**

Write the Upsert semantics to atomically adding a new row or updating the existing row if there is a primary key constraint violation, which provides idempotence.

If there are failures, the RocketMQ offset used for recovery may not be up-to-date with what was committed as of the time of the failure, which can lead to re-processing during recovery.

Aside from failure recovery, the source topic can also naturally contain multiple records over time with the same primary key, making upserts desirable.

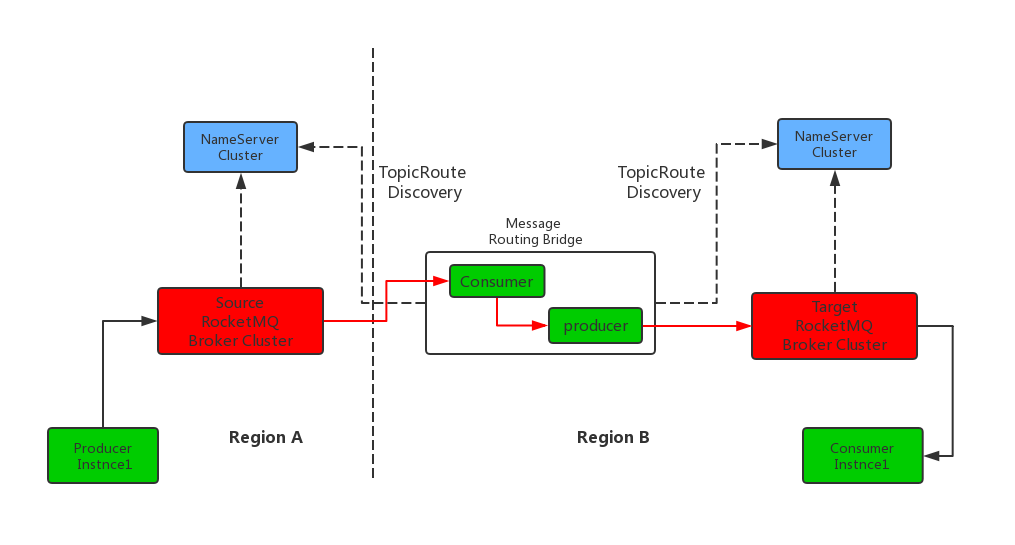
**Auto-creation and Auto-evoluton**

Auto.create: the connector can create the destination table if it is found to be missing. The creation takes place online with records being consumed from the topic, since the connector uses the record schema as a basis for the table definition.

Auto.evolve: the connector can perform limited auto-evolution by issuing alter on the destination table when it encounters a record for which a column is found to be missing.

**Dialect jdbc driver**

**Auto-creation and Auto-evoluton**



* **What I will do**

1. Standardize the code style, format the code, and complementary code structure and unit testing for the runtime of openmessaging
2. Optimize and modify some of the runtime‘s code so that runtime can be configured to load the source cluster URL address or the oms\_driver URL adaptively
3. Define and implement the functions of RmqConnector and RmqTask. This task includes code writing, unit testing and final integration.
4. Add the interface and implementation of the Metrics part of the message connect part in the runtime
5. The configuration information (mainly refers to topic and ACL) is synchronized between the cluster of source and sink of RocketMQ. Configure topic information, including TopicName, Offset, TopicPerm, QueueNum
6. Add RocketMQ’s message connect support for ACL message authentication
7. Finish a message connect connector for JDBC-Compatible databases(essential)
8. Finish a message connect connector for other databases, such as MongoDB and JMS(optional )

# **Deliverables**

Milestones:

1. Write a message connect connector to connect JDBC-Compatible databases and other databases
2. Standardize the code style and complementary code structure
3. Modify some of the runtime‘s code and write new function
4. Configure information to synchronize between the cluster of source
5. Add message connect support for ACL message authentication

documentation: Write the documentation about RocketMQ Connector and how to connect to JDBC and Other data source

tests: Unit test openmessaging runtime code and the Functions about RmqTask and RmqConnector

# **Results for the Apache Community**

Apache RocketMQ is a Top-Level Project in the Apache Software Foundation, also one of the most popular open source software in China. My work is mainly to build the new rocketmq message connector, which will help Apache RocketMQ users and developers to open up message exchange among multiple clusters and efficiently realize synchronous replication of messages in RocketMQ clusters distributed in different regions.

My work is a part of message connector for new futures which will be published in the new version of RocketMQ. It will make rocketmq more scalable and better support synchronization of multiple heterogeneous data sources.

# **Scheduling**

|  |  |  |
| --- | --- | --- |
| Date | Work | |
| Prior - May 7 | **·**  **·** | Get familiar with the RocketMQ and message connect refer to relevant information about connector, databases.  Keep diving into Rocket’s code by fixing issues. |
| May 7 - 27 | **·**  **·** | Time for community Bonding, learn about Apache Community, dive into the architecture of RocketMQ, set up a development environment for coding and debugging  Discuss with mentors about the implementation of the algorithm. Investigate the classic algorithms which we can refer to. |
| May 28 – June 9 | **·**  **·** | Standardize the code style, Format the code for runtime of Openmessaging  Complementary code structure and unit testing |
| June 10 – 23 | **·**  **·** | Optimize and modify some codes of runtime  Configure runtime to load source cluster URL address or the oms\_driver URL adaptively |
| June 24 - July 7 | **·**  **·** | Add interface and implementation of the Metrics part of the message connect in the runtime  Define and implement the functions about RocketMQ Connector and RocketMQ Task |
| July 8 - 21 | **·**  **·** | Synchronize the configuration information between the cluster of source and sink of RocketMQ. Configure topic information, including TopicName, Offset, TopicPerm, QueueNum and so on  Add RocketMQ’s message connect support for ACL message authentication |
| July 22 – August 4 | **·** | Finish a message connect connector for databases (mainly for JDBC, if time permits, do more databases’ connector) |
| August 5 - 11 | **·** | Do test for message connect connector and write the document for project |
| August 12 - 20 | **·** | There are 9 buffer days in case something didn’t go as planned in the weeks before. Fix bug and organize documents |

# **Other commitments**

My final exam for this semester will be held in early June. Then In my summer vacation, I will concentrate on the gsoc project and won’t have any part time work.

I am planning to dedicate approximately 42 hours per week to the project (7 hours per day, Monday to Saturday). On every Sunday, I will restructuring the code created throughout the week and submitting it in a pull request for evaluation.

Besides, I will update my progress on my GitHub every week..

Kafka Connectors to copy data between Kafka and other systems that you want to pull data from or push data to. Source Connectors import data from another system. Sink Connectors export data.

The who is who

• Producers write data to brokers.

• Consumers read data from

brokers.

• All this is distributed.

The data

• Data is stored in topics.

• Topics are split into partitions,

which are replicated.

OpenMessaging Specification

OpenMessaging is vendor-neutral and language-independent, provides industry guidelines for areas of finance, e-commerce, IoT and big-data, and aimed to develop messaging and streaming applications across heterogeneous systems and platforms.

# **Related Work**

[1] Kreps, J., Narkhede, N., & Rao, J. (2011, June). Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB* (pp. 1-7).

[2]kafka: Kafka® is used for building real-time data pipelines and streaming apps. It is horizontally scalable, fault-tolerant, wicked fast, and runs in production in thousands of companies.

[3] RocketMq: RocketMQ is a distributed messaging and streaming platform with low latency, high performance and reliability, trillion-level capacity and flexible scalability.

[4] OpenMessaging Specification: OpenMessaging is vendor-neutral and language-independent, provides industry guidelines for areas of finance, e-commerce, IoT and big-data, and aimed to develop messaging and streaming applications across heterogeneous systems and platforms.

[5] JDBC: Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database.