标签页 1

StarRocks Redis catalog design doc

Reminder: related issue is issue#59842 and pull request is pr#58994 .

Background

This Redis catalog feature is inspired by Trino's Redis catalog. Users of the Trino Redis catalog can easily migrate to the SR Redis catalog. Users can conduct joint queries across multiple data sources based on multi-catalog, such as hive/iceberg/Redis catalog.

Goals

The goal of this design is:

1. To enable users to explore Redis data using SQL queries in StarRocks.
2. To allow users to perform federated queries between the Redis catalog and other data sources (e.g., Iceberg catalog).
3. To ensure compatibility between StarRocks' Redis catalog usage and Trino's Redis catalog implementation.

Non Goals

1. Writing data to Redis via StarRocks.
2. Accelerating Redis queries through StarRocks.

Prototype Design

When initially designing, the Java client of Redis was considered to be more mature and stable. Therefore, it was decided to use the JNI method at the BE layer to read Redis data, while the FE layer was responsible for managing Redis metadata, generating execution plans, etc.

Design reference

This solution is similar to the current JDBC catalog of SR. Especially for the JNI reading process
on the BE side, the CPP JNI code part of the Redis catalog referred to the CPP JNI implementation of JDBC.

However, Redis does not have the concept of structured metadata like JDBC datasource, so the metadata of Redis can only be customized by users in the form of a JSON file, and then the FE reads the JSON-defined Redis metadata information. **The JSON definition process of the metadata of the Redis catalog referred to the implementation of Trino Redis catalog**.

Precondition

Because Redis does not have structured metadata, **if we want to query Redis data from SR, we assume that the key/value pairs in Redis must follow a certain specific format so that SR can use this format to parse the Redis data** and finally return it in the form of SR table data to the SR client.

To be specific, the key definition in Redis must follow the format of `dbName:tblName:xx`. StarRocks will retrieve all keys and their values that match `dbName:tblName:*`, and then return these keys/values in the form of StarRocks table data. For example, if a user sets Redis keys as `testdb:test:aa` and `testdb:test:bb`, these two key/value pairs belong to the same database `testdb` and the `test` table. Users can query the Redis data by executing `select * from testdb.test.`
You can check the example to know more redis table definition info.

FE side Redis catalog design

Redis MetaData Parse:

The implementation of FE mainly focuses on how to parse the user-defined JSON files to construct a structured Redis metadata structure, and send the constructed table information (database table name, column name) to BE. Other parts of the code, such as PhysicalRedisScanOperator.java, are basically consistent with the existing other catalog code implementations.

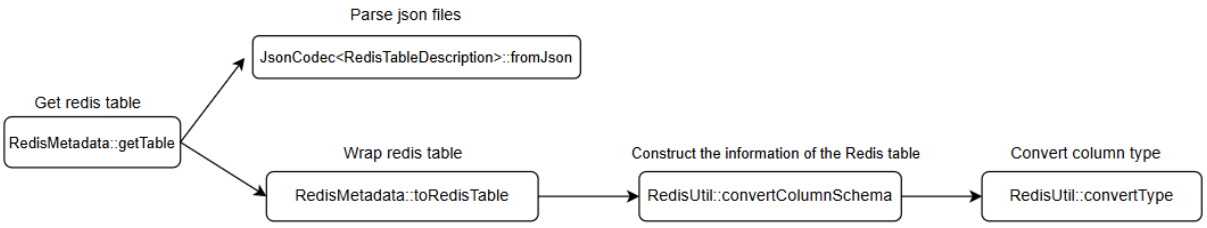

Figure 1. FE Side Redis Metadata Parsing Process

Figure 1. The definition and parsing process of Redis metadata mainly refer to the implementation on the Trino side. That is, the user defines the metadata structure of Redis using a JSON file, and the SR parses the JSON file and constructs the table metadata structure of the SR.

BE side Redis catalog design

On the BE side, Redis data is read using the JNI method. The JNI implementation section refers to the JNI implementation of JDBC catalog.

The following is an illustration of the code flow for reading Redis data from the Redis catalog on the BE side:

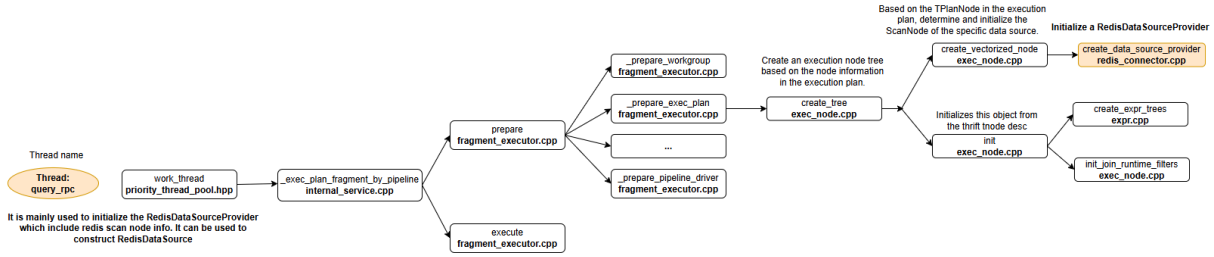

Figure 2. query_rpc thread, initializing RedisDataSourceProvider

Figure 2. The Redis-related content in the query_rpc thread mainly involves initializing RedisDataSourceProvider, which contains the scan node information of Redis and is used for subsequent initialization of RedisDataSource. RedisDataSource is the main class for performing Redis read operations.

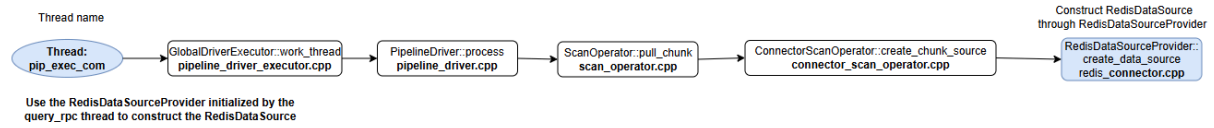

Figure 3. pip_exec_com thread, constructing the RedisDataSource object

Figure 3. The pip_exec_com thread mainly accomplishes the construction of the RedisDataSource object, as well as the subsequent construction of RedisScanner for the RedisDataSource user.

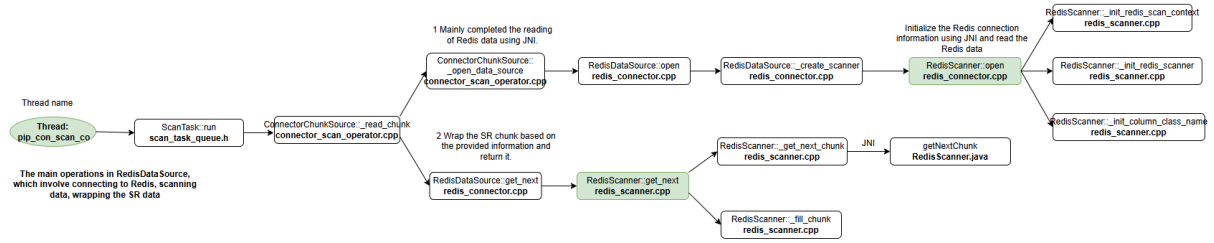

Figure 4. pip_con_scan_co thread, which uses RedisDataSource to scan and read Redis data via JNI methods.

Figure 4. The pip_con_scan_co thread mainly initializes the RedisScanner using the RedisDataSource object. Within the RedisScanner, it uses JNI to interact with the Redis service, scans the Redis data, and encapsulates it into StarRocks data blocks before returning them to the client. The execution process of JNI is generally similar to that of the SR JDBC catalog.

Example for discussion:

You can refer to the doc in the PR Add Redis Catalog to view the more detailed usage examples.

Here is a simple example that can be used for discussion:

- You can use the following command to create the Redis catalog:

```
CREATE EXTERNAL CATALOG redis_catalog
PROPERTIES
(
    "type"="redis",
    "password"="pawssd",
    "redis_uri"="127.0.0.1:6379",
    "redis.table-description-dir"="/path/redis"
);
```

- Write two lines of String type data in JSON format using the Redis CLI client:

```
SET testdb:testjson:bb "{\"id\":123,\"name\":\"Alice\",\"email\":\"alice@example.com\"}"
SET testdb:testjson:aa "{\"id\":456,\"name\":\"Lily\",\"email\":\"lily@example.com\"}"
```

You can see that the keys/values of Redis are all written in a specific predefined format. This makes it convenient for SR to parse/decode the Redis data and return it to the SR client in the form of an SR table.

- Create the table definition json file `/path/redis/test_json.json`:

```
{
  "tableName": "testjson",
  "schemaName": "testdb",
  "key": {
     "dataFormat": "raw",
     "fields": [
          {
              "name":"redis_key",
              "type":"VARCHAR"
          }
     ]
  },
  "value": {
     "dataFormat": "json",
     "fields": [
          {
              "name":"id",
```

```
            "type":"BIGINT"
        },
        {
            "name":"name",
            "type":"VARCHAR"
        },
        {
            "name":"email",
            "type":"VARCHAR"
        }
    ]
  }
}
```

The json file is used for SR to construct redis table metadata. **The attributes in the JSON file must be strictly corresponding to the keys/values data written in Redis. Only in this way can SR read the data from Redis.**

- Query the Redis table `redis_catalog.testdb.testjson` data through StarRocks:

```
mysql> select * from redis_catalog.testdb.testjson;
+--------------------+------+-------+-------------------+
| redis_key          | id   | name  | email             |
+--------------------+------+-------+-------------------+
| testdb:testjson:aa |  456 | Lily  | lily@example.com  |
| testdb:testjson:bb |  123 | Alice | alice@example.com |
+--------------------+------+-------+-------------------+
```