StarRocks Catalog

背景

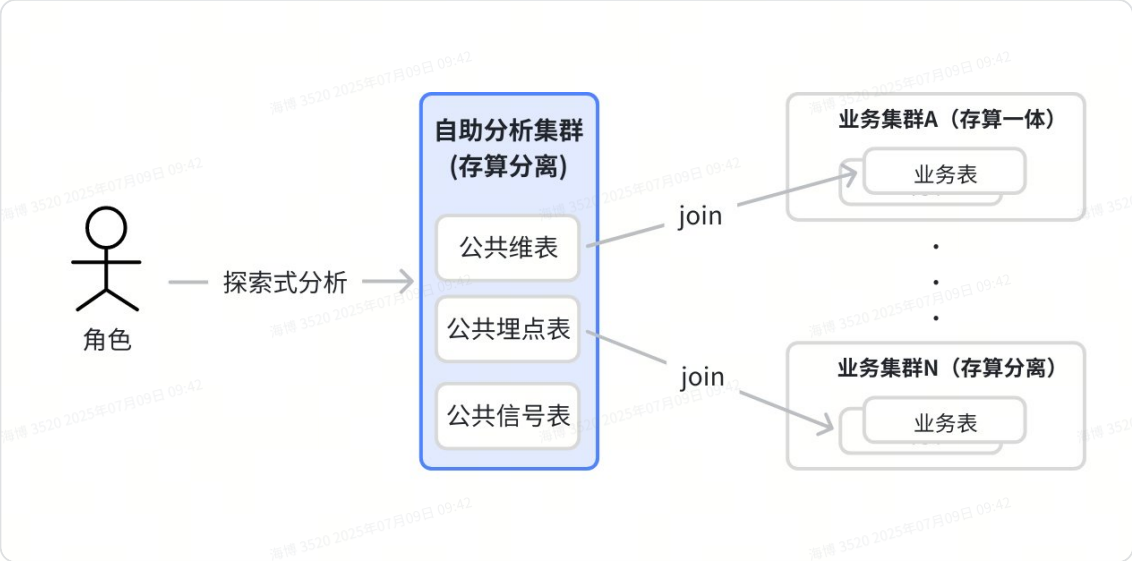
有2个业务场景目前不能被满足，本期主要解决场景一的问题

场景一

现状：如下所示目前共有10几个业务独立集群（存量集群为存算一体, 新增集群都为存算分离）。“**自助分析业务**”有独立的集群，其上有加工好的各种“公共表”，需要关联其他业务集群独有的“业务表”提供自助分析服务（需查询时长<=5s）。

痛点：

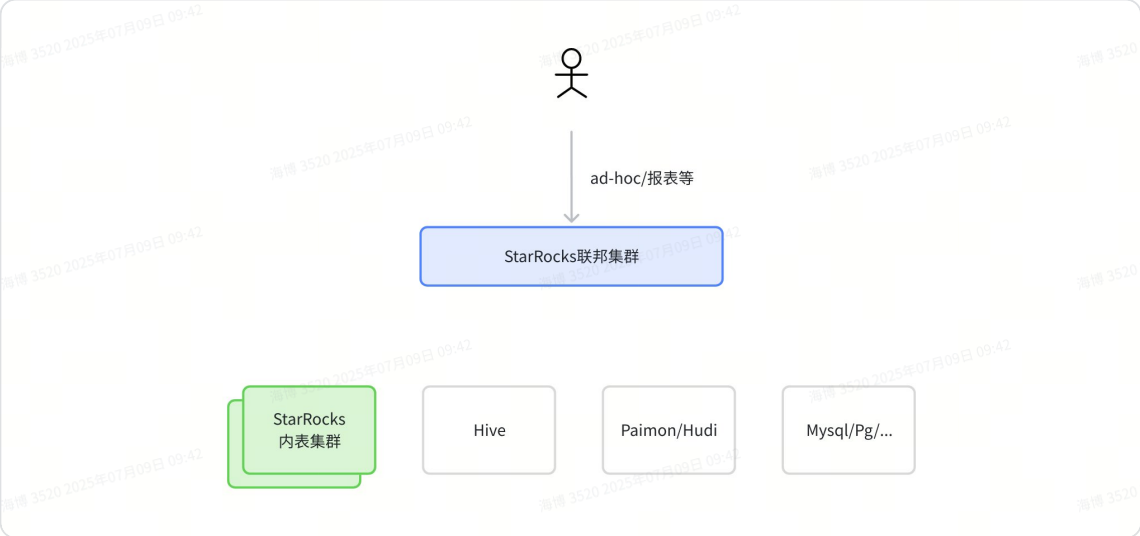
- 尝试通过JDBC Catalog实现跨StarRocks集群查询，由于不能切片、下推能力较弱等原因导致**查询性能不满足要求**
- 目前通过将业务集群表同步到自助分析集群解决，但是存在：**效率低、成本高、数据一致性**等问题



场景二

现状：使用一个“存算分离集群（没有内表）” 作为“联邦查询引擎”，服务ad-hoc等场景。

痛点：同场景一，用JdbcCatalog支持跨StarRocks集群，查询性能低



设计目标

- 功能：实现 StarRocks Catalog，支持高性能的跨StarRocks集群查询
  - 2025 H2：实现基于Tablet进行切片，调用外部StarRocks集群BE RPC集群获取数据的方式来支持
  - 2026 H1：实现直接读取外部StarRocks集群对象存储文件的方式来支持
- 性能：Scan性能达到 1GB/s（待定）

用户接口说明

创建Catalog

代码块

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "starrocks",
    CatalogParams,
    StorageCredentialParams,
)
```

catalog_name

StarRocks catalog 的名称。命名规则如下：

- 名称可以包含字母、数字 (0-9) 和下划线 (_)。必须以字母开头。
- 名称区分大小写，长度不能超过 1023 个字符。

comment

StarRocks catalog 的描述。此参数是可选的。

type

数据源的类型。将值设置为 `paimon` 。

CatalogParams

- 访问StarRocks集群元数据的配置
- 通过BE RPC端口获取数据的配置

参数	是否必须	默认值	描述
starrocks.run_mode	是	无	集群的运行模式（目前无用，预留用来支持直接从对象存储读取Segment文件）： • shared_data：存算分离 • shared_nothing：存算一体
starrocks.fetch.mode	否	rpc	从starrocks集群获取数据的方式： • rpc：从be的rpc端口获取 • s3: 直接从对象存储获取文件（预留）
starrocks.fe.http.url	是	无	FE的http地址
starrocks.fe.jdbc.url	是	无	FE的MySQL Server地址
starrocks.user	是	无	访问StarRocks集群的jdbc账号
starrocks.password	是	无	jdbc账号密码
starrocks.request.retries	否	3	向StarRocks发送一个读请求的重试次数
starrocks.request.connect.timeout.ms	否	30000	一个读请求读取StarRocks数据超时时间
starrocks.request.read.timeout.ms	否	30000	一次读请求读取StarRocks数据超时时间
starrocks.request.query.timeout.s	否	3600	从StarRocks查询数据的超时时间。默认1小时
starrocks.batch.size	否	4096	单次从BE读取的最大行数
starrocks.exec.mem.limit	否	2147483648	单个查询的内存限制，默认2GB

starrocks.enable_data_cache	否	true	是否开启data cache，开启cache将不会进行下推

StorageCredentialParams

TODO（本期不实现）：对象存储的配置，当StarRocks集群是存算分离集群时，可以直接读取对象存储中的文件。

技术内幕解析

调研了StarRocks-Spark-Connector和Trino-StarRocks-Connector的实现，他们的实现一致：

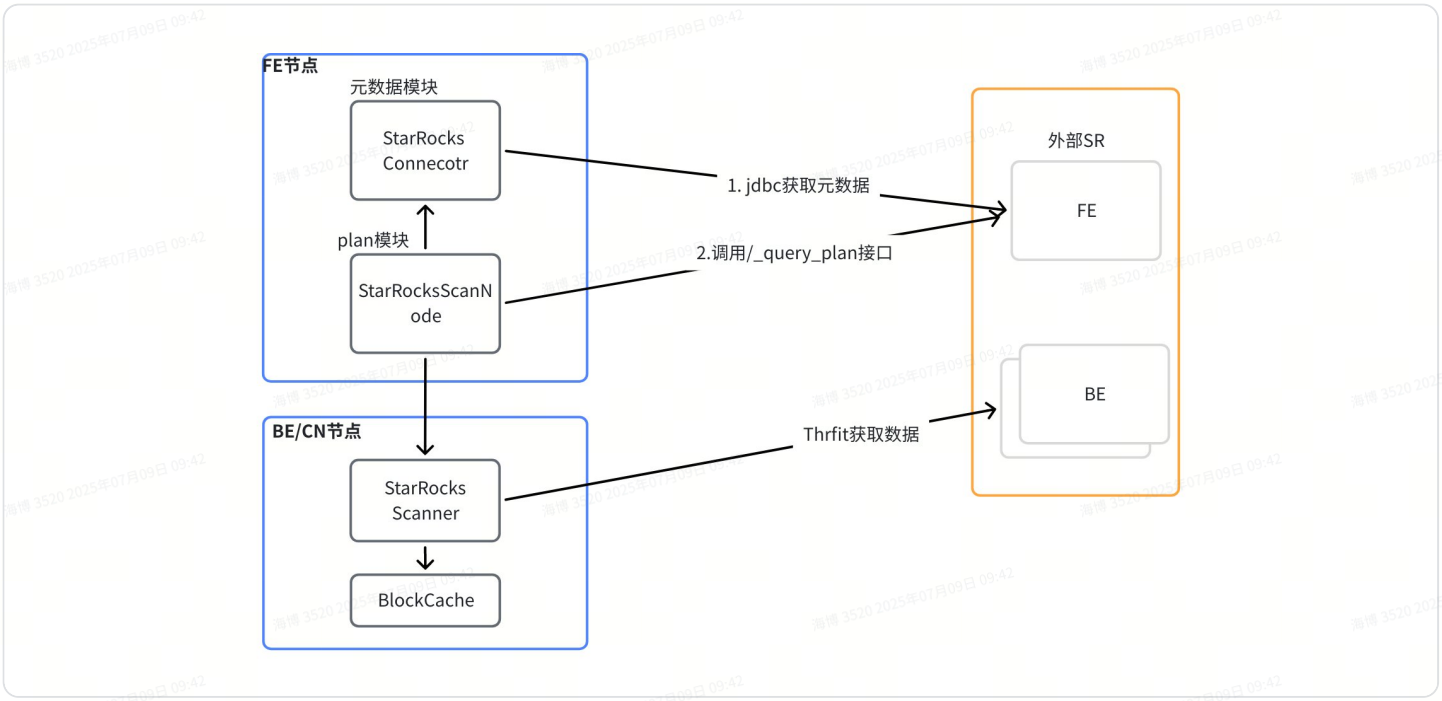
- **本期2025H2，参考其实现StarRocks Catalog**
- 下一期：StarRocks发展趋势是向存算分离收敛，类比于Paimon/Hudi等实现，可以将其他外部StarRocks集群作为一种湖格式，直接读取对象存储上对应的文件

本期实现执行流程如下：

1. 第一步：FE Connector模块增加StarRocks Connector，通过jdbc方式从外部StarRocks获取对应的库表等元数据
2. 第二步：执行SQL时，通过调用外部StarRocks的/_query_plan api结果获取要扫描的Tablet列表和

执行计划信息

3. 第三步：生成Fragment时，根据第二步的结果生成对应的StarRocksScanNode，根据Tablet进行切片（setUpScanRangeLocations）
4. 第四步：将StarRocksScanNode下发到BE/CN节点进行执行。CN/BE首先从BlockCache中读取数据，不存在时通过外部StarRocks BE节点Thrift接口分批获取对应Tablet的数据



FE：

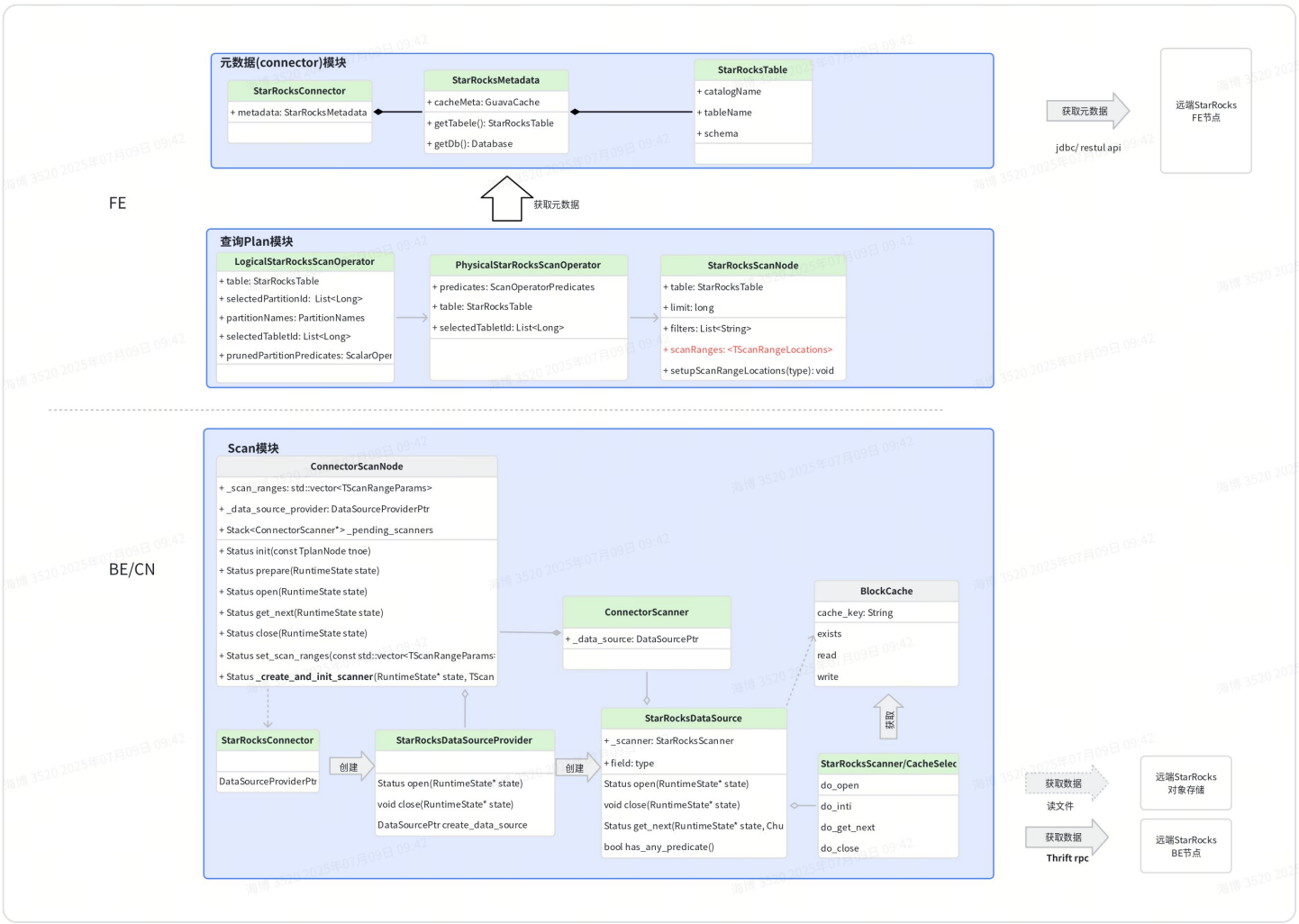
- 元数据模块：
  - 增加StarRocksConnector管理外部StarRocks元数据，通过JdbcURL获取外部StarRocks的库表等元数据
  - StarRocksMetadata提供元数据访问方法，在其中本地Cache外部StarRocks的元数据信息，并支持通过 "refresh external starrocks.catalog.db.tb" 进行元数据更新
- 查询Plan模块：
  - 增加逻辑执行计划和物理执行计划算子，在plan的对应阶段进行使用
  - 增加StarRocksScanNode，在生成Fragment时创建 ，实现 setupScanRangeLocations 实现对扫描数据根据Tablet进行切片，通过一致性Hash选择对应的节点下发执行

BE/CN：

Scan算子可以使用BlockCache，执行Scan时，StarRocksDataSource可以生成2种Scanner获取数据：

- 已有CacheSelectScanner：从BlockCache中获取数据，以TabletId + visableVersion 作为CacheKey
- 新增StarRocksScanner:
  - 本期：通过目标StarRocks集群Tablet所在BE节点的Thrift RPC端口，分批获取数据
  - TODO：对于存算分离集群，可以直接从对应的对象存储读取Segment文件

本期UML图如下：



设计折衷

无

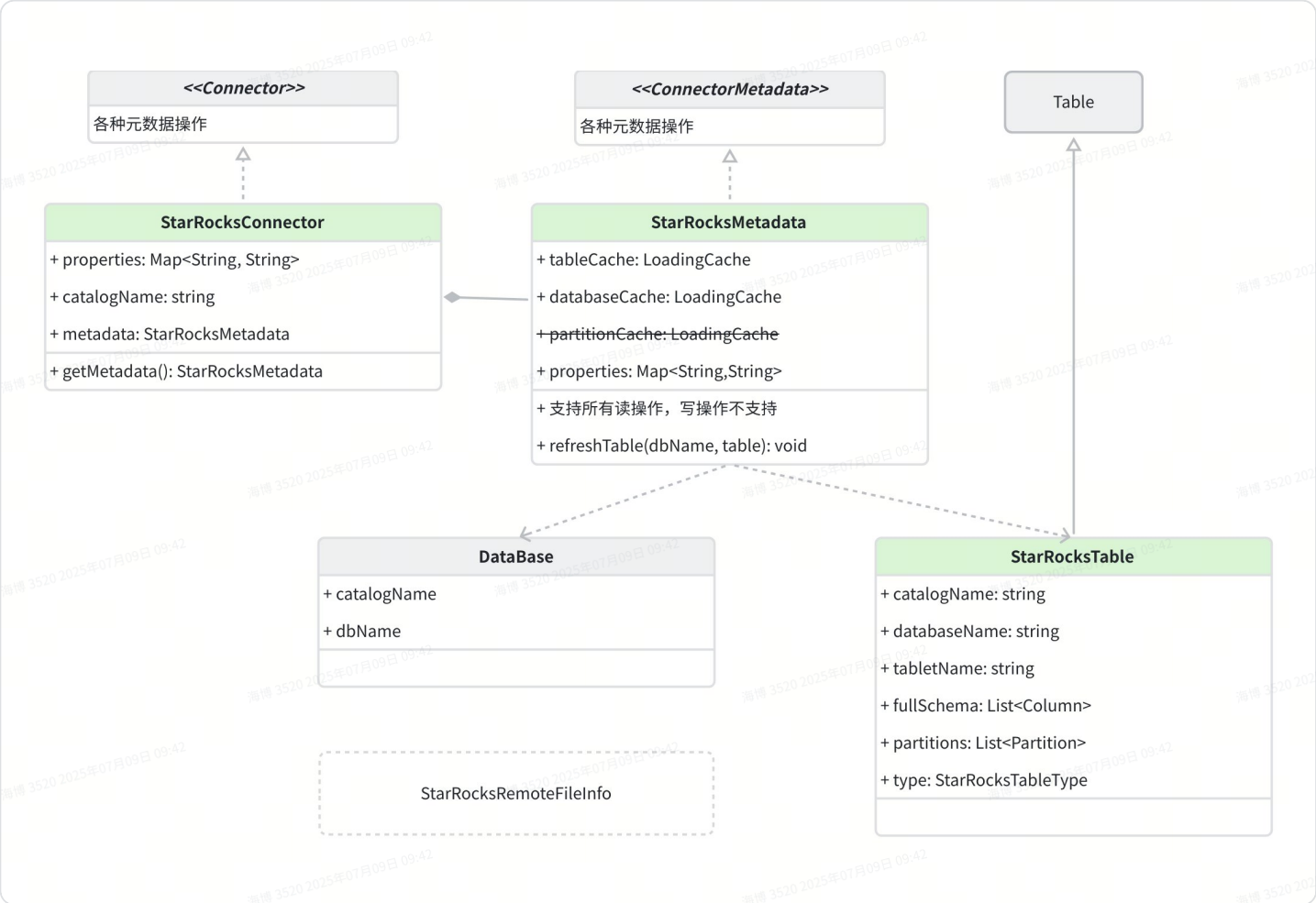
详细设计

元数据管理

元数据实现如下：

- 实现StarRocksMetadata：
  - 对元数据进行缓存，并提供refresh刷新元数据的方法
  - 和外部StarRocks通过JDBC交互，获取各类元数据信息，**只支持读元数据**，不支持修改元数据。
  - 备注：partitionCache不需要，原因：通过调用外部StarRocks集群的_query_plan接口实现分区、分桶裁剪
- 实现StarRocksTable，支持OlapTable和View（逻辑视图和物化视图）

- StarRockStarRocksemoteFileInfo本期不实现，为以后从对象存储获取文件预留



元数据获取实现：和JDBCMetadata实现一致

- 使用HikariDataSource构建JDBC连接
- 通过jdbc连接获取元数据信息，通过MysqlSchemaResolver进行转换，如获取Table信息的例子：

代码块

```
@Override
public Table getTable(ConnectContext context, String dbName, String tblName) {
    JDBCTableName jdbcTable = new JDBCTableName(null, dbName, tblName);
    return tableInstanceCache.get(jdbcTable,
            k -> {
                try (Connection connection = getConnection()) {
                    ResultSet columnSet =
schemaResolver.getColumns(connection, dbName, tblName);
                    List<Column> fullSchema =
schemaResolver.convertToStarRocksTable(columnSet);
                    List<Column> partitionColumns = Lists.newArrayList();
                    if (schemaResolver.isSupportPartitionInformation()) {
                        partitionColumns = listPartitionColumns(dbName,
tblName, fullSchema);
```

```
                    }
                    if (fullSchema.isEmpty()) {
                        return null;
                    }

                    Integer tableId =
tableIdCache.getPersistentCache(jdbcTable,
                            j ->
ConnectorTableId.CONNECTOR_ID_GENERATOR.getNextId().asInt());
                    return schemaResolver.getTable(tableId, tblName,
fullSchema,
                            partitionColumns, dbName, catalogName, properties);
                } catch (SQLException | DdlException e) {
                    LOG.warn("get table for JDBC catalog fail!", e);
                    return null;
                }
            });
}
```

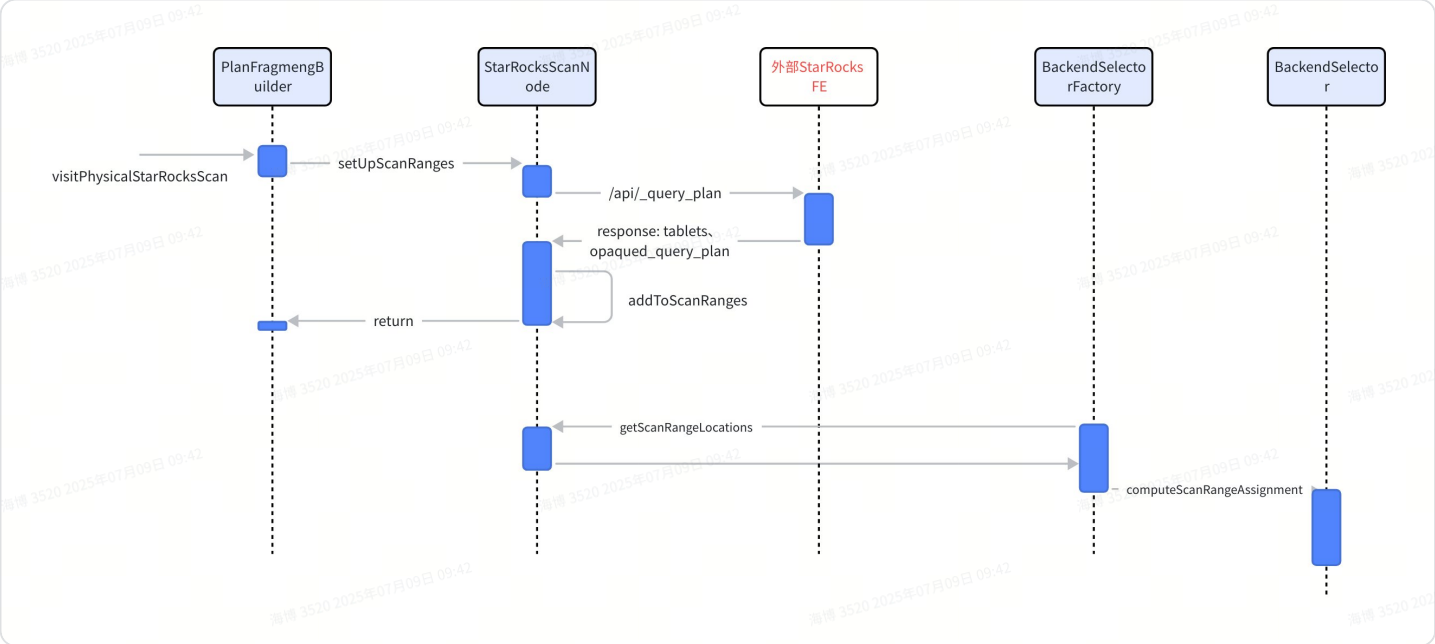
元数据刷新实现：参考JDBCMetadata，使对应表的缓存失效，下次查询直接从数据源获取

代码块

```
@Override
public void refreshTable(String StarRocksDbName, Table table, List<String>
partitionNames, boolean onlyCachedPartitions) {
    JDBCTable jdbcTable = (JDBCTable) table;
    JDBCTableName jdbcTableName = new JDBCTableName(null,
jdbcTable.getCatalogDBName(), jdbcTable.getName());
    if (!onlyCachedPartitions) {
        tableInstanceCache.invalidate(jdbcTableName);
    }
    partitionNamesCache.invalidate(jdbcTableName);
    partitionInfoCache.invalidate(jdbcTableName);
}
```

数据切片

在生成Fragment阶段，设置：StarRocksScanNode中的 List<TScanRangeLocations> scanRanges 的值。实现流程如下：



核心是调用setUpSanRanges方法，逻辑是：

- 根据查询的字段、库表、条件拼接SQL如`select a from x.y where dt = xx and a in (xx)`
- 拼接SQL作为参数调用表所在StarRocks集群FE的_query_plan接口，获取要扫描的Tablet信息（这一步可进行分区分桶的裁剪）和opaqued_query_plan，结果示例如下，可获得Tablet和其所在BE的ip信息

代码块

```
{
    "partitions": {
        "128651": {
            "routings": [
                "172.21.33.76:9060",
                "172.21.33.73:9060",
                "172.21.32.58:9060"
            ],
            "version": 2,
            "schemaHash": 2132871545
        },
        "128655": {
            "routings": [
                "172.21.32.57:9060",
                "172.21.33.75:9060",
                "172.21.33.73:9060"
            ],
            "version": 2,
            "schemaHash": 2132871545
        },
        "128659": {
            "routings": [
```

```
                "172.21.33.69:9060",
                "172.21.33.77:9060",
                "172.21.37.141:9060"
            ],
            "version": 2,
            "schemaHash": 2132871545
        },
        "128663": {
            "routings": [
                "172.21.33.76:9060",
                "172.21.32.58:9060",
                "172.21.33.70:9060"
            ],
            "version": 2,
            "schemaHash": 2132871545
        },
        "128667": {
            "routings": [
                "172.21.33.75:9060",
                "172.21.33.69:9060",
                "172.21.37.141:9060"
            ],
            "version": 2,
            "schemaHash": 2132871545
        },
        "128671": {
            "routings": [
                "172.21.33.69:9060",
                "172.21.33.70:9060",
                "172.21.32.57:9060"
            ],
            "version": 2,
            "schemaHash": 2132871545
        },
        "128675": {
            "routings": [
                "172.21.33.71:9060",
                "172.21.33.77:9060",
                "172.21.33.70:9060"
            ],
            "version": 2,
            "schemaHash": 2132871545
        },
        "128679": {
            "routings": [
                "172.21.33.71:9060",
                "172.21.33.75:9060",
```

```
                "172.21.33.76:9060"
            ],
            "version": 2,
            "schemaHash": 2132871545
        }
    },
    "opaqued_query_plan":
"DAABDAACDwABDAAAAAEIAAEAAAAACAACAAAAAAgAAwAAAAAKAAT/////////w8ABQgAAAABAAA
AAA8ABgIAAAABAAIACAAMABIIAAEAAAAADwACCwAAAAEAAAAHdXNlcl9pZA8AAwgAAAABAAAADwI
ABAELABQAAAAWdXNlcl9ncm91cF9ycWJmY2M0ZjNkOAIAFgANABcICAAAAAAPABgLAAAAAAIAGQA
PABsLAAAAAQAAAAd1c2VyX2lkCAAc/////wIAHgAPAB8MAAAAAQsAAQAAAAd1c2VyX2lkAgAEAQI
ABQECAAkACAAK/////wgAFAAAABQMABUPAAEMAAAAAQgAAQAAAAAMAAIIAAEAAAAPCAACAAAAQAA
AAAACACAAAgAhAQIAIAwECACQBCgAlAAAAAAAB9ooADgA5CAAAAAACADsAAAAPAAQMAAAAAQ8AAQw
AAAABCAABAAAAEAwAAg8AAQwAAAABCAABAAAAAAwAAggAAQAAAA8IAAL/////AAAACAAEAAAAAAw
ADwgAAQAAAAEIAAIAAAAAAAgAFP////8IABf/////AgAzAAIANAECADYBAgA5AAAADAAFCAABAAA
ABgwACAAADAAGCAABAAAAAQ8AAgwAAAAAAAANAAIKDAAAAAgAAAAAAAH2owoAAQAAAAAAAfajCgA
CAAAAAAAAAAIKAAMAAAAAAAAAAAgABH8hCXkAAAAAAAAB9pMKAAEAAAAAAAH2kwoAAgAAAAAAAAA
CCgADAAAAAAAAAAAIAAR/IQl5AAAAAAAAAfaXCgABAAAAAAAB9pcKAAIAAAAAAAAAgoAAwAAAAA
AAAAACAAEfyEJeQAAAAAAAAH2pwoAAQAAAAAAAfanCgACAAAAAAAAAAIKAAMAAAAAAAAAAgABH8
hCXkAAAAAAAAB9osKAAEAAAAAAAH2iwoAAgAAAAAAAACCgADAAAAAAAAAAAIAAR/IQl5AAAAAAA
AAfabCgABAAAAAAAB9psKAAIAAAAAAAAAAgoAAwAAAAAAAAACAAEfyEJeQAAAAAAAAH2nwoAAQA
AAAAAfafCgACAAAAAAAAAIKAAMAAAAAAAAAAgABH8hCXkAAAAAAAAB9o8KAAEAAAAAAAH2jwo
AAgAAAAAAAACCgADAAAAAAAAAAAIAAR/IQl5AAwAAw8AAQwAAAABCAABAAAAAQgAAgAAAAAMAAM
PAAEMAAAAAQgAAQAAAAAMAAIIAAEAAAAPCAACAAAAQAAAAAgABP////8IAAX/////CAAG/////wg
ABwAAAAELAAgAAAAHdXNlcl9pZAgACf////8CAAoBAgALAAIADAEADwACDAAAAAEIAAEAAAAACAA
C/////wgAA/////8KAAQAAAAAAAH2iQgABf////8ADwADDAAAAAEKAAEAAAAAAAH2iQgAAgAAAAE
IAAMAAAABCAAEAAAAAAsABwAAABZ1c2VyX2dyb3VwX3JxYmZjYzRmM2Q4CwAIAAAAAAwACwsAAQA
AABZ1c2VyX2dyb3VwX3JxYmZjYzRmM2Q4AAAADAAECgAB3XekgXU8RtwKAAKtBRzeR/ljvgAPAAU
LAAAAAQAAAAd1c2VyX2lkAA==",
    "status": 200
}
```


- 将Tablet信息生成List<TScanRangeLocations>
- 后续真正执行时BackendSelector可以根据一致性hash方法将tablet下发到不同的CN节点去执行：
  - 如果不使用data cache，则均匀下发
  - 如果使用data cache，则下发到cache所在BE节点

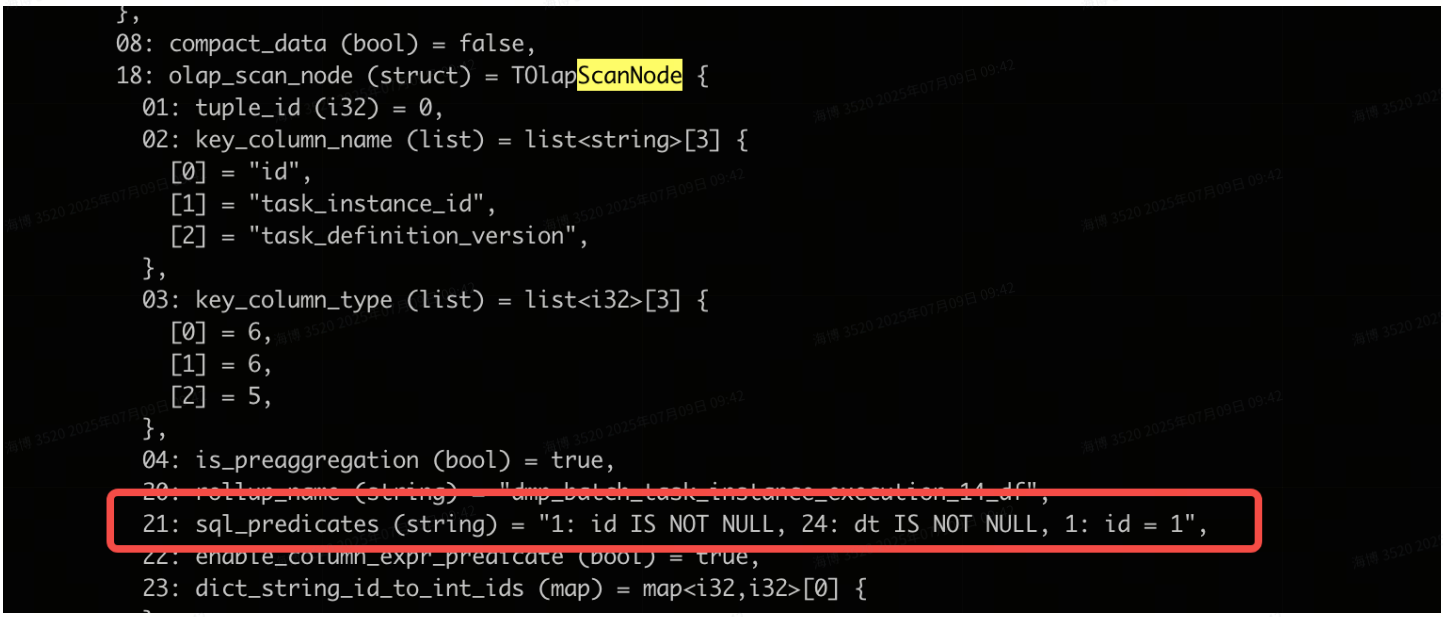
下推

opaqued_query_plan 是经过base64编码的执行计划，包含filter、limit、查询的列等信息。会作为参数传递给外部StarRocks，外部StarRocks可以利用opaqued_query_plan，从中获取对应的信息，执行Scan时可以执行：

- where条件下推

- limit下推
- 列裁剪

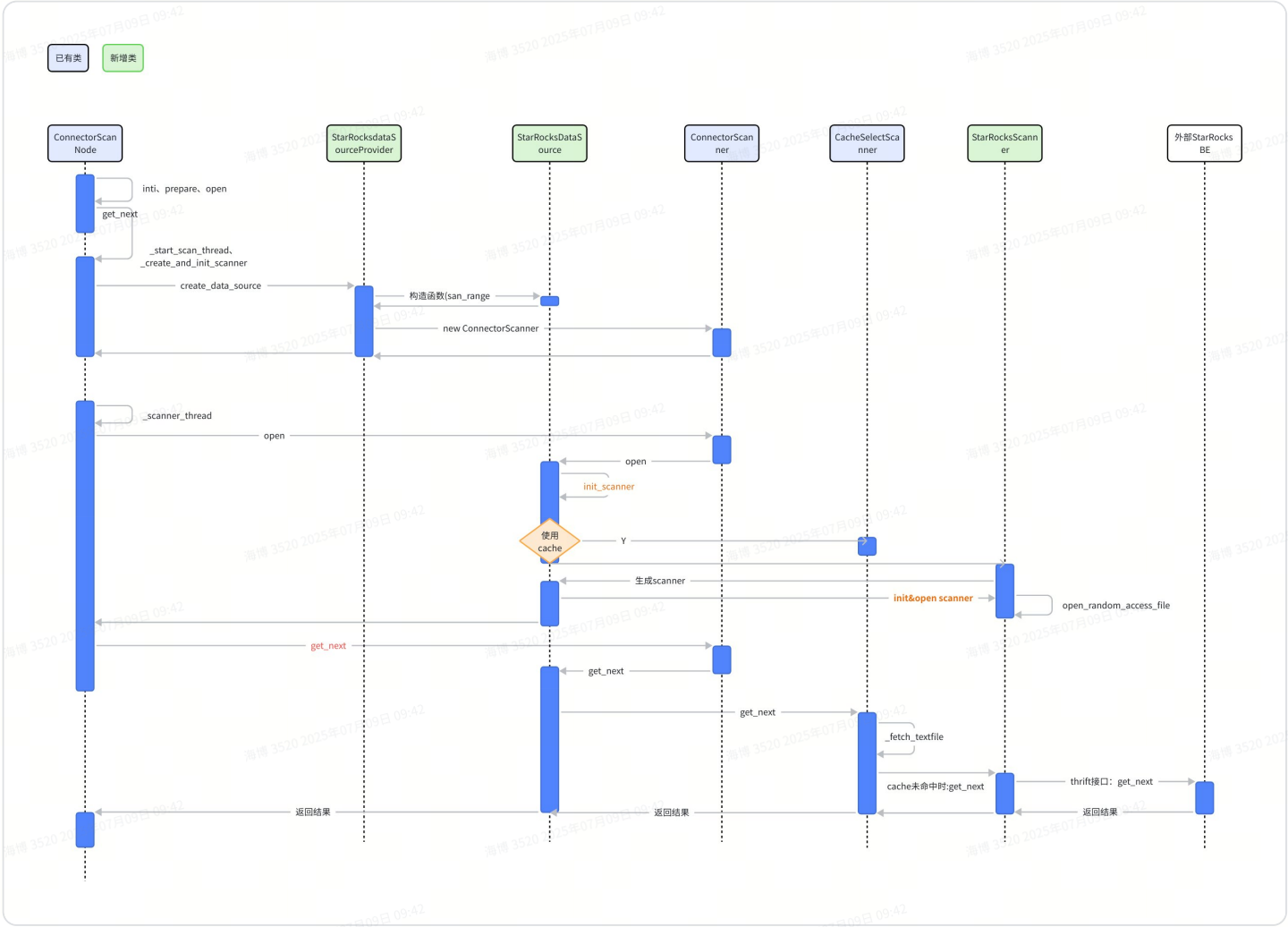
如：opaqued_query_plan 示例如下，携带predicates信息，可以利用其进行谓词下推



数据读取和缓存

流程如下所示：

- 首先调用CacheSelectScanner从BlockCache中获取数据
- 如果Cache中没有数据，调用StarRocksScanner从外部的StarRocks BE中获取数据



缓存设计：

- 利用已有的BlockCache
- CacheKey：tabletId+visiableVersion构成，用visiableVersion标识缓存的版本
- 读取和写入：携带版本读取，如果缓存数据版本落后，则从外部读取数据

StarRocksScanner读取数据：调用外部StarRocks BE的thrift接口分批获取数据，实现类似（starrocks-spark-connector中的实现）：

1. 首先调用openScanner方法
2. 不断调用get_next方法分批（默认一批4096行）获取数据
3. 数据获取完毕后调用closeScanner方法

代码块

```
/**
 * Open a scanner for reading StarRocks data.
 *
 * @param openParams thrift struct to required by request
```

```
     * @return scan open result
     * @throws ConnectedFailedException throw if cannot connect to StarRocks BE
     */
    public TScanOpenResult openScanner(TScanOpenParams openParams) throws
    ConnectedFailedException {
        logger.debug("OpenScanner to '{}', parameter is '{}'.", routing,
    openParams);
        if (!isConnected) {
            open();
        }
        TException ex = null;
        for (int attempt = 0; attempt < retries; ++attempt) {
            logger.debug("Attempt {} to openScanner {}.", attempt, routing);
            try {
                TScanOpenResult result = client.open_scanner(openParams);
                if (result == null) {
                    logger.warn("Open scanner result from {} is null.", routing);
                    continue;
                }
                if (!TStatusCode.OK.equals(result.getStatus().getStatus_code())) {
                    logger.warn("The status of open scanner result from {} is
    '{}', error message is: {}.",
                            routing, result.getStatus().getStatus_code(),
    result.getStatus().getError_msgs());
                    continue;
                }
                return result;
            } catch (TException e) {
                logger.warn("Open scanner from {} failed.", routing, e);
                ex = e;
            }
        }
        logger.error(ErrorMessages.CONNECT_FAILED_MESSAGE, routing);
        throw new ConnectedFailedException(routing.toString(), ex);
    }

    /**
     * get next row batch from StarRocks BE
     *
     * @param nextBatchParams thrift struct to required by request
     * @return scan batch result
     * @throws ConnectedFailedException throw if cannot connect to StarRocks BE
     */
    public TScanBatchResult getNext(TScanNextBatchParams nextBatchParams) throws
    StarrocksException {
        logger.debug("GetNext to '{}', parameter is '{}'.", routing,
    nextBatchParams);
```

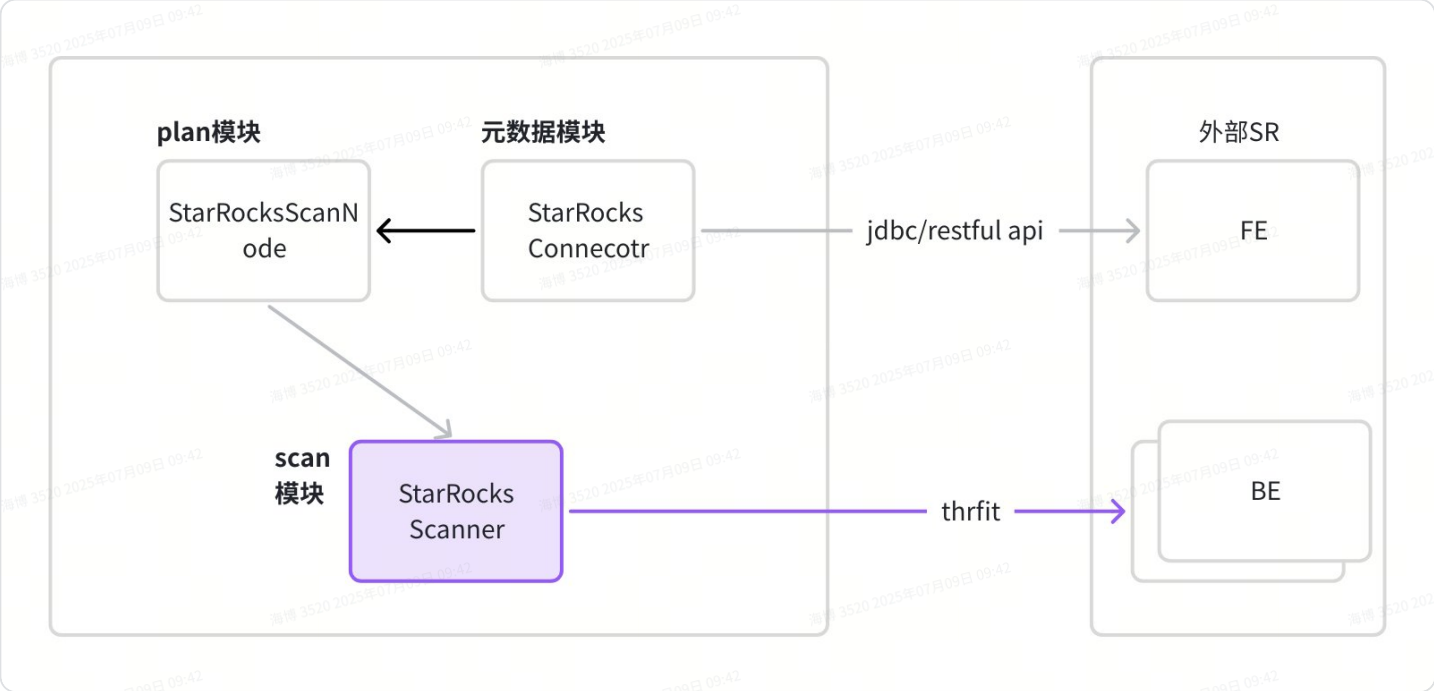
```
46          if (!isConnected) {
47              open();
48          }
49          TException ex = null;
50          TScanBatchResult result = null;
51          for (int attempt = 0; attempt < retries; ++attempt) {
52              logger.debug("Attempt {} to getNext {}.", attempt, routing);
53              try {
54                  result = client.get_next(nextBatchParams);
55                  if (result == null) {
56                      logger.warn("GetNext result from {} is null.", routing);
57                      continue;
58                  }
59                  if (!TStatusCode.OK.equals(result.getStatus().getStatus_code())) {
60                      logger.warn("The status of get next result from {} is '{}',
    error message is: {}.",
61                              routing, result.getStatus().getStatus_code(),
    result.getStatus().getError_msgs());
62                      continue;
63                  }
64                  return result;
65              } catch (TException e) {
66                  logger.warn("Get next from {} failed.", routing, e);
67                  ex = e;
68              }
69          }
70          if (result != null && (TStatusCode.OK !=
    (result.getStatus().getStatus_code()))) {
71              logger.error(ErrorMessages.STARROCKS_INTERNAL_FAIL_MESSAGE, routing,
    result.getStatus().getStatus_code(),
72                      result.getStatus().getError_msgs());
73              throw new StarrocksInternalException(routing.toString(),
    result.getStatus().getStatus_code(),
74                      result.getStatus().getError_msgs());
75          }
76          logger.error(ErrorMessages.CONNECT_FAILED_MESSAGE, routing);
77          throw new ConnectedFailedException(routing.toString(), ex);
78      }
79
80      /**
81       * close an scanner.
82       *
83       * @param closeParams thrift struct to required by request
84       */
85      public void closeScanner(TScanCloseParams closeParams) {
86          logger.debug("CloseScanner to '{}', parameter is '{}'.", routing,
    closeParams);
```

```
87          if (!isConnected) {
88              try {
89                  open();
90              } catch (ConnectedFailedException e) {
91                  logger.warn("Cannot connect to StarRocks BE {} when close
    scanner.", routing);
92                  return;
93              }
94          }
95          for (int attempt = 0; attempt < retries; ++attempt) {
96              logger.debug("Attempt {} to closeScanner {}.", attempt, routing);
97              try {
98                  TScanCloseResult result = client.close_scanner(closeParams);
99                  if (result == null) {
100                     logger.warn("CloseScanner result from {} is null.", routing);
101                     continue;
102                 }
103                 if (!TStatusCode.OK.equals(result.getStatus().getStatus_code())) {
104                     logger.warn("The status of get next result from {} is '{}',
    error message is: {}.",
105                             routing, result.getStatus().getStatus_code(),
    result.getStatus().getError_msgs());
106                     continue;
107                 }
108                 break;
109             } catch (TException e) {
110                 logger.warn("Close scanner from {} failed.", routing, e);
111             }
112         }
113         logger.info("CloseScanner to StarRocks BE '{}' success.", routing);
114         close();
115     }
```

稳定性设计

如下设计3个模块：

- 元数据模块通过jdbc和restful api访问目标StarRocks元数据，数据量小没有稳定性问题；plan模块和外部没有交互，没有稳定性问题
- 稳定性风险主要存在于Scan模块，需要通过thrift接口大量获取外部StarRocks的数据，可能面临网络、外部StarRocks异常等多种问题，这种问题可能会影响本集群。需要这些风险因素，隔离对本集群的影响。



对于Scan模块，为了应对潜在的稳定性风险：

- 分批获取数据，避免单次获取数据量太多占用网络和内存
- 设置超时时间
  - 设置Thrift连接超时属性connectTimeout，默认30s，避免外部StarRocks异常无响应的情况
  - 设置Thrift读取超时属性socketTimeout，默认30s，避免获取数据时网络异常或数据过大的情况
  - 透传query timeout到外部BE节点，熔断过大的查询
- 其他数据读取稳定性设计，对齐现有HdfsScanner

可观测性设计

- 元数据模块：打印和外部StarRocks的交互日志、异常日志
- scan模块：
  - 打印和外部StarRocks的交互日志（打印读取的tablet信息、请求的BE信息等）、异常日志
  - 延用现有Profile中的Metrics，对SQL进行问题定位，包括：
    - dataCache指标
    - RowStarRocksead等数据量读取指标
    - ScanRange信息等

缺点

无

未解决问题

统计信息采集：暂不采集

测试要点

基础功能、基础性能、异常测试等

功能测试用例	• 获取库表元数据：show tables、desc tables等SQL
	• 刷新元数据：元数据正常更新
	• 元数据只读：执行alter table报错
	• 元数据缓存功能
	• 读取外部StarRocks数据：关联本地StarRocks表和外部StarRocks表，能正常查询
	• 数据缓存：第二次查询从Cache中读取数据，通过profile查看
性能测试用例	外部StarRocks导入Tpch数据，在本StarRocks集群创建对应StarRocks Catalog，执行Tpch测试，和本地Tpch性能进行对比
异常测试用例	• 持续大量读取外部数据，观察系统状态，确保不影响集群正常服务和查询 • 故障演练：读取外部数据时，认为设置网络故障、外部StarRocks故障，预期本StarRocks集群不受影响

Roadmap

一个项目如果涉及周期比较长，比如超过多人一个月，那么尽量应该将一个周期长的工作拆分成多个Roadmap，这样能够保证项目在一个可持续交付的

模块	工作项	排期
元数据	Connector模块开发	7.14 - 7.25
plan模块	ScanNode和切片开发	
scan模块	StarRocks Scanner数据读取功能开发	7.25 - 8.15
	数据Cache能力开发	

测试	测试用例&自测	8.15 - 9.01