尚硅谷大数据之电商实时数仓

之

DIM 层实现

（作者：尚硅谷研究院）

版本：V3.0

第 1 章 DIM 层设计要点

(1) DIM 层的设计依据是维度建模理论，该层存储维度模型的维度表。

(2) DIM 层的数据存储在 HBase 表中

DIM 层表是用于维度关联的，要通过主键去获取相关维度信息，这种场景下 K-V 类型数据库的效率较高。常见的 K-V 类型数据库有 Redis、HBase，而 Redis 的数据常驻内存，会给内存造成较大压力，因而选用 HBase 存储维度数据。

(3) DIM 层表名的命名规范为 dim_表名

第 2 章 配置表

本层的任务是将业务数据直接写入到不同的 HBase 表中。那么如何让程序知道流中的哪些数据是维度数据？维度数据又应该写到 HBase 的哪些表中？为了解决这个问题，我们选择在 MySQL 中构建一张配置表，通过 Flink CDC 将配置表信息读取到程序中。

2.1 Flink CDC

Flink CDC 的介绍及配置请参考如下文档。

尚硅谷大数据技术
之Flink CDC.docx

注意：部分系统使用 FlinkCDC2.1 连接 MySQL 报错 javax.net.ssl.SSLHandshakeException: No appropriate protocol 的异常。

方案 1：参考 https://blog.csdn.net/wuyu7448/article/details/121131352

修改 jre/lib/security/java.security 中的 disabledAlgorithms，删除 SSLv3,

TLSv1,TLSv1.1，然后重启应用即可

jdk.tls.disabledAlgorithms=RC4, DES,MD5withRSA, \DH keySize < 1024, EC keySize < 224, 3DES_EDE_CBC, anon, NULL, \include jdk.disabled.namedCurves

方案 2：把 flinkCDC 版本改低

https://ververica.github.io/flink-cdc-connectors/release-1.4/content/about.html#usage-for-datastream-api

2.2 配置表设计

2.2.1 字段解析

我们将为配置表设计五个字段

- source_table：作为数据源的业务数据表名
- sink_table：作为数据目的地的 Phoenix 表名
- sink_columns：Phoenix 表字段
- sink_pk：Phoenix 表主键
- sink_extend：Phoenix 建表扩展，即建表时一些额外的配置语句

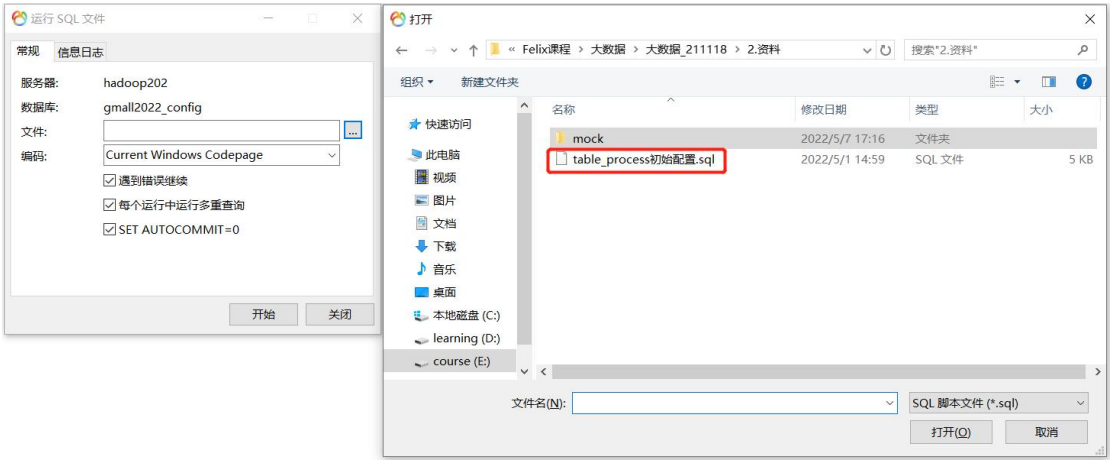
将 source_table 作为配置表的主键，可以通过它获取唯一的目标表名、字段、主键和建表扩展，从而得到完整的 Phoenix 建表语句。

2.2.2 Mysql 中创建数据库建表并开启 Binlog

(1) 创建数据库 gmall2022_config ，注意:和 gmall2022 业务库区分开

```
[atguigu@hadoop202 dblog]$ mysql -uroot -p123456 -e"create database gmall2022_config charset utf8 default collate utf8_general_ci"
```

(2) 执行建表语句以及数据导入 SQL 文件如下



(3) 建表语句如下

```
CREATE TABLE `table_process` (
  `source_table` varchar(200) NOT NULL COMMENT '来源表',
  `sink_table` varchar(200) DEFAULT NULL COMMENT '输出表',
  `sink_columns` varchar(2000) DEFAULT NULL COMMENT '输出字段',
  `sink_pk` varchar(200) DEFAULT NULL COMMENT '主键字段',
  `sink_extend` varchar(200) DEFAULT NULL COMMENT '建表扩展',
  PRIMARY KEY (`source_table`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

(4) 在 MySQL 配置文件中增加 gmall_config 开启 Binlog，

```
server-id= 1
log-bin=mysql-bin
binlog_format=row
binlog-do-db=gmall2022
binlog-do-db=gmall2022_config
```

(5) 重启 MySQL 服务

```
[atguigu@hadoop202 ~]$ sudo systemctl restart mysqld
```

第 3 章 主要任务

3.1 接收 Kafka 数据，过滤空值数据

对 Maxwell 抓取的数据进行 ETL，有用的部分保留，没用的过滤掉。

3.2 动态拆分维度表功能

由于 Maxwell 是把全部数据统一写入一个 Topic 中，这样显然不利于日后的数据处理。所以需要把各个维度表拆开处理。

在实时计算中一般把维度数据写入存储容器，一般是方便通过主键查询的数据库比如 HBase,Redis,MySQL 等。

这样的配置不适合写在配置文件中，因为这样的话，业务端随着需求变化每增加一张维度表表，就要修改配置重启计算程序。所以这里需要一种动态配置方案，把这种配置长期保存起来，一旦配置有变化，实时计算可以自动感知。这种可以有三个方案实现：

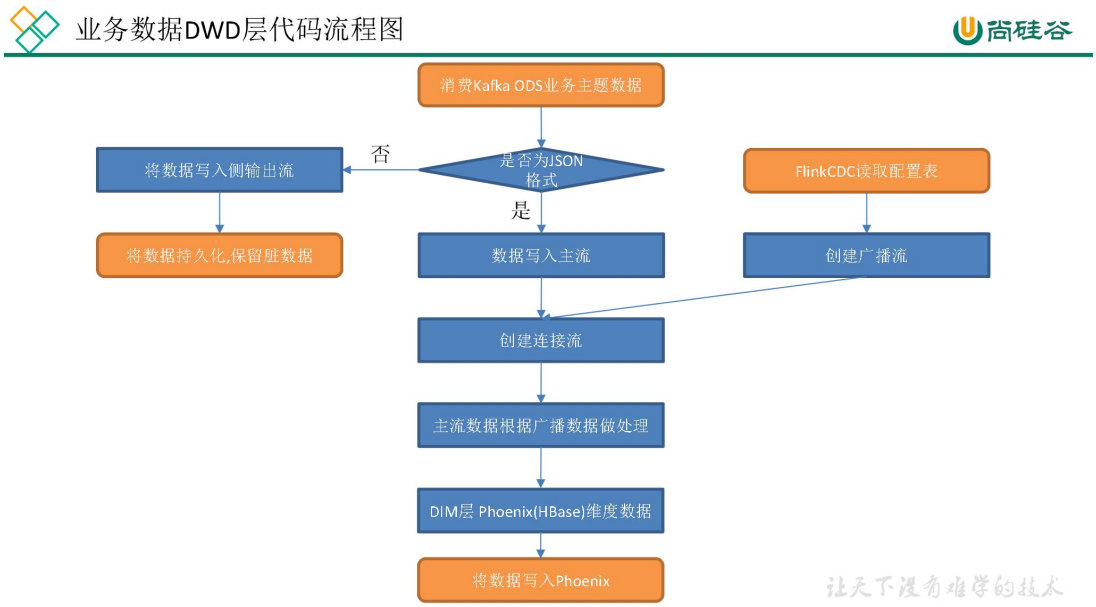
一种是用 Zookeeper 存储，通过 Watch 感知数据变化；

另一种是用 mysql 数据库存储，周期性的同步；

再一种是用 mysql 数据库存储，使用广播流。

这里选择第三种方案，主要是 MySQL 对于配置数据初始化和维护管理，使用 FlinkCDC 读取配置信息表，将配置流作为广播流与主流进行连接。

所以就有了如下图：



3.3 把流中的数据保存到对应的维度表

维度数据保存到 HBase 的表中。

第 4 章 代码实现

4.1 接收 Kafka 数据，过滤空值数据

4.1.1 创建 KafkaUtil 工具类

和 Kafka 交互要用到 Flink 提供的 FlinkKafkaConsumer、FlinkKafkaProducer 类，为了提高模板代码的复用性，将其封装到 KafkaUtil 工具类中。

此处从 Kafka 读取数据，创建 getKafkaConsumer(String topic, String groupId) 方法

```
package com.atguigu.gmall.realtime.util;

import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
import org.apache.flink.streaming.connectors.kafka.KafkaDeserializationSchema;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;

import java.util.Properties;

/**
 * Author: Felix
 * Date: 2022/5/9
 * Desc: 操作 kafka 的工具类
 */
public class MyKafkaUtil {
    private static String BOOTSTRAP_SERVERS = "hadoop202:9092, hadoop203:9092,
hadoop204:9092";
    private static String DEFAULT_TOPIC = "default_topic";

    public static FlinkKafkaConsumer<String>  getKafkaConsumer(String topic,
String groupId) {
        Properties prop = new Properties();
        prop.setProperty("bootstrap.servers", BOOTSTRAP_SERVERS);
        prop.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);

        FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>(topic,
            new KafkaDeserializationSchema<String>() {
                @Override
```

```
            public boolean isEndOfStream(String nextElement) {
                return false;
            }

            @Override
            public String deserialize(ConsumerRecord<byte[], byte[]> record)
throws Exception {
                if(record != null && record.value() != null) {
                    return new String(record.value());
                }
                return null;
            }

            @Override
            public TypeInformation<String> getProducedType() {
                return TypeInformation.of(String.class);
            }
        }, prop);
        return consumer;
    }
}
```

4.1.2 创建维度读取主程序 DimSinkApp

```
package com.atguigu.gmall.realtime.app.dim;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONException;
import com.alibaba.fastjson.JSONObject;
import com.atguigu.gmall.realtime.util.MyKafkaUtil;
import org.apache.flink.api.common.functions.FilterFunction;
import org.apache.flink.api.common.restartstrategy.RestartStrategies;
import org.apache.flink.api.common.time.Time;
import org.apache.flink.runtime.state.filesystem.FsStateBackend;
import org.apache.flink.runtime.state.hashmap.HashMapStateBackend;
import org.apache.flink.streaming.api.CheckpointingMode;
import org.apache.flink.streaming.api.datastream.DataStreamSource;
import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
import org.apache.flink.streaming.api.environment.CheckpointConfig;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;

import java.util.concurrent.TimeUnit;

/**
 * Author: Felix
 * Date: 2022/5/9
 * Desc: 维度处理类
 */
public class DimSinkApp {
    public static void main(String[] args) throws Exception {
        // TODO 1. 环境准备
        StreamExecutionEnvironment                    env                    =
StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(4);

        // TODO 2. 检查点相关设置
        env.enableCheckpointing(3000L, CheckpointingMode.EXACTLY_ONCE);
        env.getCheckpointConfig().setCheckpointTimeout(60 * 1000L);
        env.getCheckpointConfig().setMinPauseBetweenCheckpoints(3000L);
```

```
env.getCheckpointConfig().enableExternalizedCheckpoints(CheckpointConfig.Ext
ernalizedCheckpointCleanup.RETAIN_ON_CANCELLATION);
        env.setRestartStrategy(RestartStrategies.failureRateRestart(10,
Time.of(1L, TimeUnit.DAYS), Time.of(3L, TimeUnit.MINUTES)));
        env.setStateBackend(new HashMapStateBackend());

env.getCheckpointConfig().setCheckpointStorage("hdfs://hadoop202:8020/gmall/
ck");
        System.setProperty("HADOOP_USER_NAME", "atguigu");

        // TODO 3. 读取业务主流
        //3.1 声明消费主题以及消费者组
        String topic = "topic_db";
        String groupId = "dim_sink_app";
        //3.2 获取消费者对象
        FlinkKafkaConsumer<String>                kafkaConsumer                =
MyKafkaUtil.getKafkaConsumer(topic, groupId);
        //3.3 消费数据 得到流
        DataStreamSource<String> dbStrDS = env.addSource(kafkaConsumer);

        // TODO 4. 对读取的流中数据结构转换
        SingleOutputStreamOperator<JSONObject>             jsonDS             =
dbStrDS.map(JSON::parseObject);

        // TODO 5. 主流 ETL
        SingleOutputStreamOperator<JSONObject> filterDS = jsonDS.filter(
            new FilterFunction<JSONObject>() {
                @Override
                public boolean filter(JSONObject jsonObj) throws Exception {
                    try {
                        jsonObj.getJSONObject("data");
                        if (jsonObj.getString("type").equals("bootstrap-start")
                                ||
jsonObj.getString("type").equals("bootstrap-complete")) {
                            return false;
                        }
                        return true;
                    } catch (JSONException jsonException) {
                        return false;
                    }
                }
            });
        // 打印测试
        filterDS.print("filterDS >>> ");
        env.execute();
    }
}
```

4.2 根据 MySQL 的配置表处理维度数据

4.2.1 导入依赖

```
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-jdbc_${scala.version}</artifactId>
```

```
    <version>${flink.version}</version>
</dependency>

<dependency>
    <groupId>com.ververica</groupId>
    <artifactId>flink-connector-mysql-cdc</artifactId>
    <version>2.1.0</version>
</dependency>

<!-- 如果不引入 flink-table 相关依赖，则会报错:
Caused                                                                    by:
java.lang.ClassNotFoundException:org.apache.flink.connector.base.source.read
er.RecordEmitter
引入以下依赖可以解决这个问题（引入某些其它的 flink-table 相关依赖也可）
-->
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-table-api-java-bridge_${scala.version}</artifactId>
    <version>${flink.version}</version>
</dependency>

<dependency>
    <groupId>org.apache.phoenix</groupId>
    <artifactId>phoenix-spark</artifactId>
    <version>5.0.0-HBase-2.0</version>
    <exclusions>
        <exclusion>
            <groupId>org.glassfish</groupId>
            <artifactId>javax.el</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.2.8</version>
</dependency>
```

4.2.2 创建配置表实体类

```
package com.atguigu.gmall.realtime.bean;
import lombok.Data;
/**
 * Author: Felix
 * Date: 2022/5/10
 * Desc: 配置表对应实体类
 */
@Data
public class TableProcess {
    //来源表
    String sourceTable;
    //输出表
    String sinkTable;
    //输出字段
    String sinkColumns;
    //主键字段
    String sinkPk;
```

```
    //建表扩展
    String sinkExtend;
}
```

4.2.3 在 DimSinkApp 类中添加读取配置表代码并广播

```
// TODO 6. FlinkCDC 读取配置流并广播流
// 6.1 FlinkCDC 读取配置表信息
MySqlSource<String> mySqlSource = MySqlSource.<String>builder()
    .hostname("hadoop202")
    .port(3306)
    .databaseList("gmall2022_config") // set captured database
    .tableList("gmall2022_config.table_process") // set captured table
    .username("root")
    .password("123456")
    .deserializer(new JsonDebeziumDeserializationSchema())
    .startupOptions(StartupOptions.initial())
    .build();

// 6.2 封装为流
DataStreamSource<String> mysqlDS
    =       env.fromSource(mySqlSource,       WatermarkStrategy.noWatermarks(),
"MysqlSource");

// 6.3 广播配置流
MapStateDescriptor<String, TableProcess> tableConfigDescriptor
    =   new   MapStateDescriptor<String,   TableProcess>("table-process-state",
String.class, TableProcess.class);
BroadcastStream<String>                        broadcastDS                    =
mysqlDS.broadcast(tableConfigDescriptor);
```

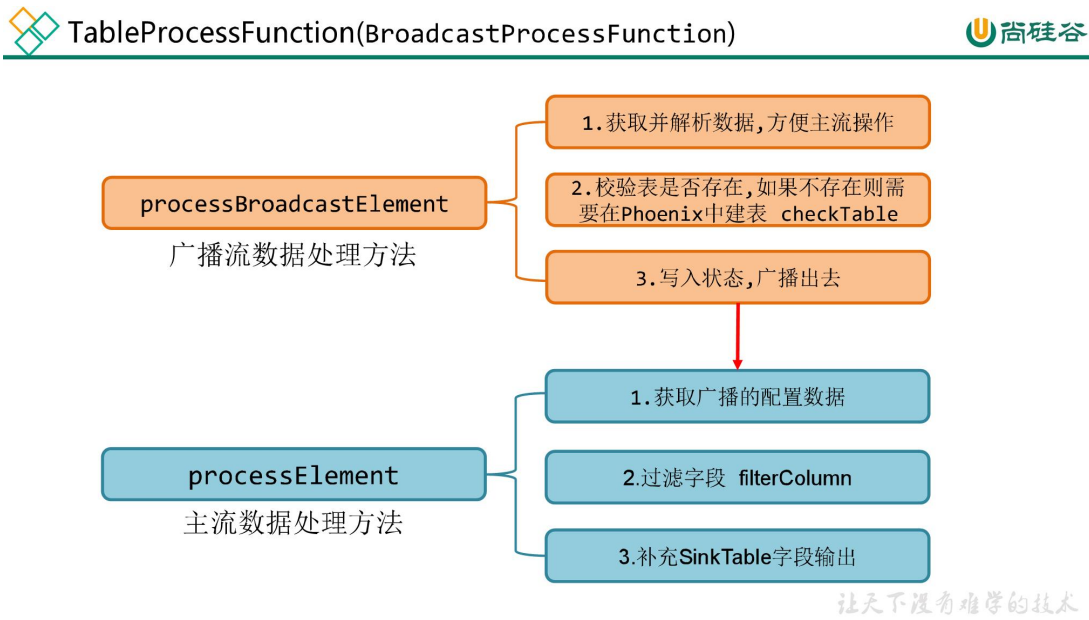
4.2.4 将主流和配置流进行关联

```
// TODO 7. 将主流和配置流进行关联
BroadcastConnectedStream<JSONObject,      String>      connectedStream      =
filterDS.connect(broadcastDS);
```

4.2.5 对关联之后的数据进行处理理

```
// TODO 8. 处理关联后数据
SingleOutputStreamOperator<JSONObject> dimDS = connectedStream.process(
    new TableProcessFunction(configTableDescriptor)
);
```

4.2.6 自定义函数 TableProcessFunction



具体代码

```
package com.atguigu.gmall.realtime.app.func;

import com.alibaba.druid.pool.DruidDataSource;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.atguigu.gmall.realtime.bean.TableProcess;
import com.atguigu.gmall.realtime.common.GmallConfig;
import com.atguigu.gmall.realtime.util.DruidDSUtil;
import org.apache.flink.api.common.state.BroadcastState;
import org.apache.flink.api.common.state.MapStateDescriptor;
import org.apache.flink.api.common.state.ReadOnlyBroadcastState;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.co.BroadcastProcessFunction;
import org.apache.flink.util.Collector;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.*;

/**
 * Author: Felix
 * Date: 2022/5/18
 * Desc: 从主流中过滤维度数据
 */
public class TableProcessFunction extends BroadcastProcessFunction<JSONObject,
String, JSONObject> {

    private DruidDataSource druidDataSource;

    private MapStateDescriptor<String, TableProcess> mapStateDescriptor;

    public    TableProcessFunction(MapStateDescriptor<String,    TableProcess>
mapStateDescriptor) {
```

```
        this.mapStateDescriptor = mapStateDescriptor;
    }

    @Override
    public void open(Configuration parameters) throws Exception {
        //获取连接池对象
        druidDataSource = DruidDSUtil.createDataSource();
    }

    //处理主流中的数据
    @Override
    public void processElement(JSONObject jsonObj, ReadOnlyContext ctx,
Collector<JSONObject> out) throws Exception {
        //获取广播状态
        ReadOnlyBroadcastState<String, TableProcess> broadcastState =
ctx.getBroadcastState(mapStateDescriptor);

        //根据处理业务流数据的表名，到广播中获取配置对象。
        String tableName = jsonObj.getString("table");

        TableProcess tableProcess = broadcastState.get(tableName);
        if(tableProcess != null){
            //获取当前操作影响的业务记录
            JSONObject dataJsonObj = jsonObj.getJSONObject("data");

            //在向下游传递数据前，将不需要传递的属性过滤掉
            String sinkColumns = tableProcess.getSinkColumns();
            filterColumn(dataJsonObj,sinkColumns);

            //在向下游传递数据前，应该补充维度发送的目的地 sink_table
            String sinkTable = tableProcess.getSinkTable();
            dataJsonObj.put("sink_table",sinkTable);

            //如果获取到了，说明当前处理的业务流数据是维度
            out.collect(dataJsonObj);
        }else{
            //从广播状态(配置表中)没有获取到配置信息，说明不是维度
            System.out.println("$$$不是维度，过滤掉" + jsonObj);
        }
    }


/*{"before":null,"after":{"source_table":"base_province","sink_table":"dim_b
ase_province",

"sink_columns":"id,name,region_id,area_code,iso_code,iso_3166_2","sink_pk":n
ull,"sink_extend":null},

"source":{"version":"1.5.4.Final","connector":"mysql","name":"mysql_binlog_s
ource","ts_ms":1652837005681,

"snapshot":"false","db":"gmall1118_config","sequence":null,"table":"table_pr
ocess","server_id":0,"gtid":null,

"file":"","pos":0,"row":0,"thread":null,"query":null},"op":"r","ts_ms":16528
37005681,"transaction":null}*/
    //处理广播流中数据
    @Override
    public void processBroadcastElement(String jsonStr, Context ctx,
```

```
Collector<JSONObject> out) throws Exception {
        //为了处理方便，可以将从配置表中读取的一条数据由 jsonstr 转换为 json 对象
        JSONObject jsonObj = JSON.parseObject(jsonStr);
        //获取广播状态
        BroadcastState<String,       TableProcess>       broadcastState       =
ctx.getBroadcastState(mapStateDescriptor);
        String op = jsonObj.getString("op");
        if("d".equals(op)){
            TableProcess before = jsonObj.getObject("before", TableProcess.class);
            String sourceTable = before.getSourceTable();
            tableConfigState.remove(sourceTable);
        }else {
            //获取配置表中的一条记录 ， 将读取这条配置信息直接封装为 TableProcess 对象
            TableProcess     tableProcess     =     jsonObj.getObject("after",
TableProcess.class);

            //获取业务数据库表名
            String sourceTable = tableProcess.getSourceTable();
            //获取数仓中维度表名
            String sinkTable = tableProcess.getSinkTable();
            //获取数仓中维度表中字段
            String sinkColumns = tableProcess.getSinkColumns();
            //获取数仓中维度表中主键字段
            String sinkPk = tableProcess.getSinkPk();
            //获取建表扩展语句
            String sinkExtend = tableProcess.getSinkExtend();

            //提前创建维度表
            checkTable(sinkTable, sinkColumns, sinkPk, sinkExtend);

            //将配置表中的配置放到广播状态中保存起来
            broadcastState.put(sourceTable,tableProcess);
        }
    }

    //创建维度表
    private void checkTable(String tableName, String columnStr, String pk, String
ext) {
        //对空值进行处理
        if (ext == null) {
            ext = "";
        }
        if (pk == null) {
            pk = "id";
        }
        //拼接建表语句
        StringBuilder createSql = new StringBuilder("create table if not exists
" + GmallConfig.PHOENIX_SCHEMA + "." + tableName + "(");

        String[] columnArr = columnStr.split(",");
        for (int i = 0; i < columnArr.length; i++) {
            String column = columnArr[i];
            if (column.equals(pk)) {
                createSql.append(column + " varchar primary key");
            } else {
                createSql.append(column + " varchar");
            }
```

```
            //除了最后一个字段后面不加逗号，其它的字段都需要在后面加一个逗号
            if (i < columnArr.length - 1) {
                createSql.append(",");
            }
        }
        createSql.append(") " + ext);
        System.out.println("在 phoenix 中执行的建表语句： " + createSql);

        Connection conn = null;
        PreparedStatement ps = null;
        try {
            //从连接池中获取连接对象
            conn = druidDataSource.getConnection();
            //创建数据库操作对象
            ps = conn.prepareStatement(createSql.toString());
            //执行 sql 语句
            ps.execute();
        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            //释放资源
            if(ps != null){
                try {
                    ps.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if(conn != null){
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    //过滤字段  dataJsonObj: {"tm_name":"xzls11","logo_url":"dfas","id":12}
    //sinkColumns:  id,tm_name
    private void filterColumn(JSONObject dataJsonObj, String sinkColumns) {
        String[] fieldArr = sinkColumns.split(",");
        List<String> fieldList = Arrays.asList(fieldArr);
        //获取 json 对象中所有的元素
        Set<Map.Entry<String, Object>> entrySet = dataJsonObj.entrySet();
        entrySet.removeIf(entry->!fieldList.contains(entry.getKey()));
    }
}
```

4.2.7 创建获取数据库连接池的工具类 DruidDSUtil

```
package com.atguigu.gmall.realtime.util;

import com.alibaba.druid.pool.DruidDataSource;
import com.atguigu.gmall.realtime.common.GmallConfig;

/**
 * Author: Felix
```

```
 * Date: 2022/5/10
 * Desc: 获取数据库连接池对象
 */
public class DruidDSUtil {
    private static DruidDataSource druidDataSource;

    public static DruidDataSource createDataSource() {
        if(druidDataSource == null){
            synchronized (DruidDSUtil.class){
                if(druidDataSource == null){
                    System.out.println("~~~获取 Druid 连接池对象~~~");
                    // 创建连接池
                    druidDataSource = new DruidDataSource();
                    // 设置驱动全类名

druidDataSource.setDriverClassName(GmallConfig.PHOENIX_DRIVER);
                    // 设置连接 url
                    druidDataSource.setUrl(GmallConfig.PHOENIX_URL);
                    // 设置初始化连接池时池中连接的数量
                    druidDataSource.setInitialSize(5);
                    // 设置同时活跃的最大连接数
                    druidDataSource.setMaxActive(20);
                    // 设置空闲时的最小连接数，必须介于 0 和最大连接数之间，默认为 0
                    druidDataSource.setMinIdle(5);
                    // 设置没有空余连接时的等待时间，超时抛出异常，-1 表示一直等待
                    druidDataSource.setMaxWait(-1);
                    // 验证连接是否可用使用的 SQL 语句
                    druidDataSource.setValidationQuery("select 1");
                    // 指明连接是否被空闲连接回收器（如果有）进行检验，如果检测失败，则连接
将被从池中去除
                    // 注意，默认值为 true，如果没有设置 validationQuery，则报错
                    // testWhileIdle is true, validationQuery not set
                    druidDataSource.setTestWhileIdle(true);
                    // 借出连接时，是否测试，设置为 false，不测试，否则很影响性能
                    druidDataSource.setTestOnBorrow(false);
                    // 归还连接时，是否测试
                    druidDataSource.setTestOnReturn(false);
                    // 设置空闲连接回收器每隔 30s 运行一次
                    druidDataSource.setTimeBetweenEvictionRunsMillis(30 * 1000L);
                    // 设置池中连接空闲 30min 被回收，默认值即为 30 min
                    druidDataSource.setMinEvictableIdleTimeMillis(30 * 60 * 1000L);
                }
            }
        }
        return druidDataSource;
    }
}
```

4.2.8 定义一个项目中常用的配置常量类 GmallConfig

```
package com.atguigu.gmall.realtime.common;
/**
 * Author: Felix
 * Date: 2022/5/10
```

```
 * Desc: 实时数仓系统常量类
 */
public class GmallConfig {
    // Phoenix 库名
    public static final String PHOENIX_SCHEMA = "GMALL2022_REALTIME";
    // Phoenix 驱动
    public static final String PHOENIX_DRIVER = "org.apache.phoenix.jdbc.PhoenixDriver";
    // Phoenix 连接参数
    public static final String PHOENIX_URL = "jdbc:phoenix:hadoop202,hadoop203,hadoop204:2181";
}
```

4.2.9 测试建表以及维度输出

(1) 在 resources 目录下创建 hbase-site.xml 文件，并在文件中添加如下内容

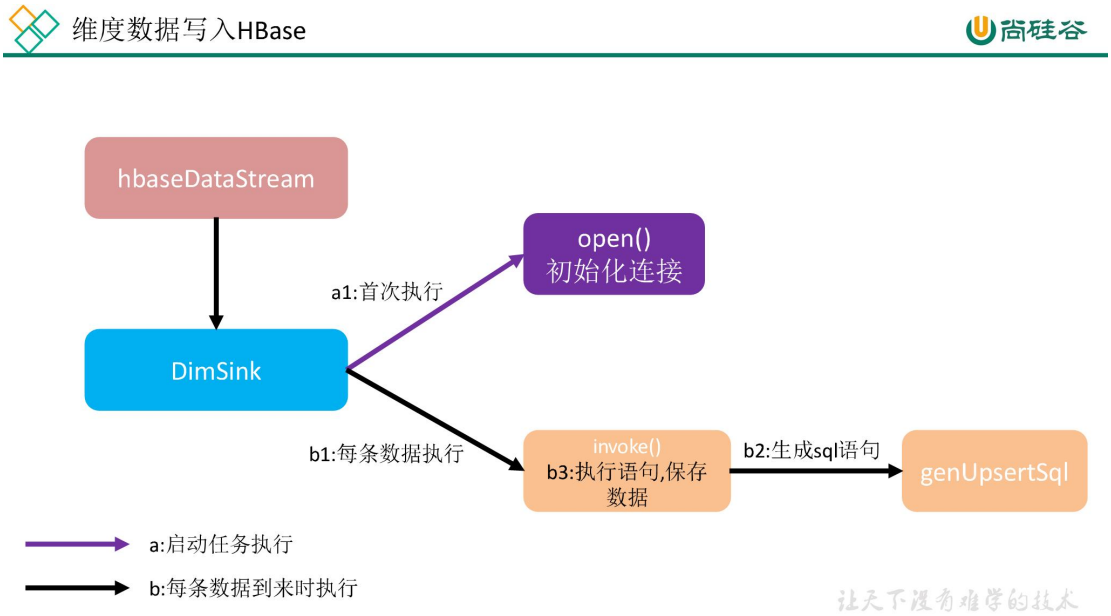
```
<configuration>
    <!-- 注意：为了开启 hbase 的 namespace 和 phoenix 的 schema 的映射，在程序中需要加这个配置文件，另外在 linux 服务上，也需要在 hbase 以及 phoenix 的 hbase-site.xml 配置文件中，加上以上两个配置，并使用 xsync 进行同步-->
    <property>
        <name>phoenix.schema.isNamespaceMappingEnabled</name>
        <value>true</value>
    </property>

    <property>
        <name>phoenix.schema.mapSystemTablesToNamespace</name>
        <value>true</value>
    </property>
</configuration>
```

(2) 在 phoenix 中创建 GMALL2022_REALTIM

```
0: jdbc:phoenix:> create schema GMALL2022_REALTIME;
```

4.3 保存维度到 HBase(Phoenix)

4.3.1 程序流程分析



DimSink 继承了 RickSinkFunction，这个 function 得分两条时间线：

一条是任务启动时执行 open 操作（图中紫线），我们可以把连接的初始化工作放在此处一次性执行；

另一条是随着每条数据的到达反复执行 invoke()（图中黑线）,在这里面我们要实现数据的保存，主要策略就是根据数据组合成 sql 提交给 hbase。

4.3.2 主程序 DimSinkApp 中调用 DimSinkFunction

```
// TODO 9. 将数据写入 Phoenix 表
dimDS.addSink(new DimSinkFunction());
```

4.3.3 封装 DimSinkFunction，将流中数据输出到 Phoenix 表中

```
package com.atguigu.gmall.realtime.app.func;

import com.alibaba.druid.pool.DruidDataSource;
import com.alibaba.fastjson.JSONObject;
```

```
import com.atguigu.gmall.realtime.common.GmallConfig;
import com.atguigu.gmall.realtime.util.DruidDSUtil;
import com.atguigu.gmall.realtime.util.PhoenixUtil;
import org.apache.commons.lang3.StringUtils;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.sink.RichSinkFunction;

import java.sql.Connection;

/**
 * Author: Felix
 * Date: 2022/5/20
 * Desc: 将维度流中的数据 写到phoenix表中
 */
public class DimSinkFunction extends RichSinkFunction<JSONObject> {

    private DruidDataSource druidDataSource;

    @Override
    public void open(Configuration parameters) throws Exception {
        druidDataSource = DruidDSUtil.createDataSource();
    }

    //将流中的数据，保存到phoenix不同的维度表中
    // jonObj:  {"tm_name":"xzls11","sink_table":"dim_base_trademark","id":12}
    @Override
    public void invoke(JSONObject jsonObj, Context context) throws Exception {

        //获取维度输出的目的地表名
        String tableName = jsonObj.getString("sink_table");
        //为了将jsonObj中的所有属性保存到phoenix表中，需要将输出目的地从jsonObj中删除
掉
        //===>{"tm_name":"xzls11","id":12}
        jsonObj.remove("sink_table");

        String upsertSQL = "upsert into " + GmallConfig.PHOENIX_SCHEMA + "." +
tableName
            + "(" + StringUtils.join(jsonObj.keySet(), ",") + ") " +
            " values" +
            " ('" + StringUtils.join(jsonObj.values(), "','") + "')";
        System.out.println("向phoenix表中插入数据的语句为: " + upsertSQL);

        //从连接池中获取连接对象
        Connection conn =  druidDataSource.getConnection();
        //调用向Phoenix表中插入数据的方法
        PhoenixUtil.executeSQL(upsertSQL,conn);
    }
}
```

4.3.4 封装操作 Phoenix 的工具类 PhoenixUtil

```
package com.atguigu.gmall.realtime.util;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

/**
```

```
 * Author: Felix
 * Date: 2022/5/20
 * Desc: 操作 phoenix 的工具类
 */
public class PhoenixUtil {
    //执行 DDL 以及 DML
    public static void executeSQL(String sql, Connection conn) {
        PreparedStatement ps = null;
        try {
            //获取数据库操作对象
            ps = conn.prepareStatement(sql);
            //执行 SQL 语句
            ps.execute();
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("执行操作 phoenix 语句发生了异常");
        }finally {
            close(ps,conn);
        }
    }
    public static void close(PreparedStatement ps, Connection conn){
        if(ps != null){
            try {
                ps.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if(conn != null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

4.3.5 优化 TableProcessFunction 建表代码

```
//创建维度表
private void checkTable(String tableName, String columnStr, String pk, String
ext) {
    //对空值进行处理
    if (ext == null) {
        ext = "";
    }
    if (pk == null) {
        pk = "id";
    }
    //拼接建表语句
    StringBuilder createSql = new StringBuilder("create table if not exists " +
GmallConfig.PHOENIX_SCHEMA + "." + tableName + "(");

    String[] columnArr = columnStr.split(",");
    for (int i = 0; i < columnArr.length; i++) {
        String column = columnArr[i];
        if (column.equals(pk)) {
```

```
            createSql.append(column + " varchar primary key");
        } else {
            createSql.append(column + " varchar");
        }
        //除了最后一个字段后面不加逗号，其它的字段都需要在后面加一个逗号
        if (i < columnArr.length - 1) {
            createSql.append(",");
        }
    }
    createSql.append(") " + ext);
    System.out.println("在 phoenix 中执行的建表语句： " + createSql);

    try {
        //从连接池中获取连接对象
        Connection conn = druidDataSource.getConnection();
        PhoenixUtil.executeSQL(createSql.toString(),conn);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

4.3.6 测试

(1) 启动 HDFS、ZK、Kafka、Maxwell、HBase

(2) 运行 IDEA 中的 DimSinkApp

(3) 执行 mysql_to_kafka_init.sh 脚本

```
mysql_to_kafka_init.sh all
```

(4) 通过 phoenix 查看 hbase 的 schema 以及表情况

```
0: jdbc:phoenix:> !tables
+------------+--------------------+--------------------+---------------+----------+------------+----------------------------+---+
| TABLE_CAT  |    TABLE_SCHEM     |     TABLE_NAME     |  TABLE_TYPE   | REMARKS  | TYPE_NAME  | SELF_REFERENCING_COL_NAME  |   |
+------------+--------------------+--------------------+---------------+----------+------------+----------------------------+---+
|            | SYSTEM             | CATALOG            | SYSTEM TABLE  |          |            |                            |   |
|            | SYSTEM             | FUNCTION           | SYSTEM TABLE  |          |            |                            |   |
|            | SYSTEM             | LOG                | SYSTEM TABLE  |          |            |                            |   |
|            | SYSTEM             | SEQUENCE           | SYSTEM TABLE  |          |            |                            |   |
|            | SYSTEM             | STATS              | SYSTEM TABLE  |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_ACTIVITY_INFO  | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_ACTIVITY_RULE  | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_ACTIVITY_SKU   | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_BASE_CATEGORY1 | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_BASE_CATEGORY2 | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_BASE_CATEGORY3 | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_BASE_DIC       | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_BASE_PROVINCE  | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_BASE_REGION    | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_BASE_TRADEMARK | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_COUPON_INFO    | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_COUPON_RANGE   | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_FINANCIAL_SKU_COST | TABLE     |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_SKU_INFO       | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_SPU_INFO       | TABLE         |          |            |                            |   |
|            | GMALL2022_REALTIME | DIM_USER_INFO      | TABLE         |          |            |                            |   |
+------------+--------------------+--------------------+---------------+----------+------------+----------------------------+---+
0: jdbc:phoenix:> select * from GMALL2022_REALTIME.DIM_ACTIVITY_INFO
```

```
0: jdbc:phoenix:> select * from GMALL2022_REALTIME.DIM_ACTIVITY_INFO;
+----+---------------+---------------+---------------+---------------------+---------------------+-------------+
| ID | ACTIVITY_NAME | ACTIVITY_TYPE | ACTIVITY_DESC |     START_TIME      |      END_TIME       | CREATE_TIME |
+----+---------------+---------------+---------------+---------------------+---------------------+-------------+
| 1  | 联想专场          | 3101          | 联想满减          | 2020-10-21 18:49:12 | 2020-10-31 18:49:15 |             |
| 2  | Apple品牌日      | 3101          | Apple品牌日      | 2020-06-10 00:00:00 | 2020-06-12 00:00:00 |             |
| 3  | 女神节           | 3102          | 满件打折          | 2020-03-08 00:00:00 | 2020-03-09 00:00:00 |             |
+----+---------------+---------------+---------------+---------------------+---------------------+-------------+
```