智慧物流大数据平台

1 项目背景

2015年，在国务院提出“互联网+”行动战略之后，以“互联网+物流”的智慧物流概念也被不断提及，但是，由于物流系统本身的复杂性和业务差异性，究竟如何打造智慧物流系统，未能够有大范围的落地实施。

一些中大型的电商企业比如京东、淘宝、苏宁现在都有自己的物流体系，运行也非常高效，在一些地域甚至可以实现上午下单下午送达，如此高效的物流体系实际上都依托于信息技术的发展，其中，大数据技术的作用不可忽视。

上述电商网站日处理订单数量可达数万、数十万甚至数百万，对各个环节都有很高的要求，那么基于大数据的智慧化物流自然是非常迫切的需求。

本课程内容转化自某智慧物流大数据平台，课程将原平台功能进行了裁剪脱敏，保留了部分进行讲解，包括：仓储预测、智能运单调度（车货匹配）、车辆监控、轨迹回放等功能。

2 项目介绍

第 1 节 仓储预测

为什么现在电商购物送货那么快？仓储预测是非常重要的一个原因，意思就是根据某一区域历史数据进行某一类产品的销量预测，对商品进行提前运输就近仓储，那么当用户购买商品的时候，对应商品已经提前配置在了离你最近的存货点。话说回来，电商企业怎么知道我要买什么东西呢？其实不是针对某一个用户个人，而是针对区域购买数据预测该区域用户群体未来的购买需求。

举个例子：针对一些品牌手机首发做的“未买先送”，对某个区域的具体手机进行建模预测其销量，根据预测结果在首发之前就把手机提前配送，用户就可以在下单之后很短时间内拿到货。

预测性分析是大数据应用的一个重点，通过利用历史消费、浏览数据和仓储数据建模，对销量进行预测并进行提前仓储存货，这是一个提前预测计算的过程。

概念

一级仓库：向供货商采购的商品会优先送往这里，一般设置在中心城市，覆盖范围大

二级仓库：覆盖一些中、小型城市及边远地区，通常会根据需求将商品从一级仓库调配过来

需求

1）预测二级仓库A在未来一个月或者一周的商品销量情况

第 2 节 智能运单调度（车货匹配）

需求

1）当二级仓库需要调配商品时，系统应选择尽可能少的车辆运输，降低成本。

第 3节 车辆监控管理

需求

货车均安装有采集传感器，采集车辆的经纬度、油耗等信息，这些信息每30s上传一次，数据经过网关最终存储在MySQL数据库，采集信息项如下

调配编号(行程唯一编号)

SIM卡号

道路运输证号

车牌号

采集时间

经度

纬度

速度

方向

里程

剩余油量

AD值

载重质量

ACC开关

是否定位

运营状态

车辆油路是否正常

车辆电路是否正常

1）平台应该能够支持实时轨迹查看

2）对车辆油耗进行实时监控，当油耗<=阈值（30%）时发出预警

3）支持货车车牌号、调配编号查询

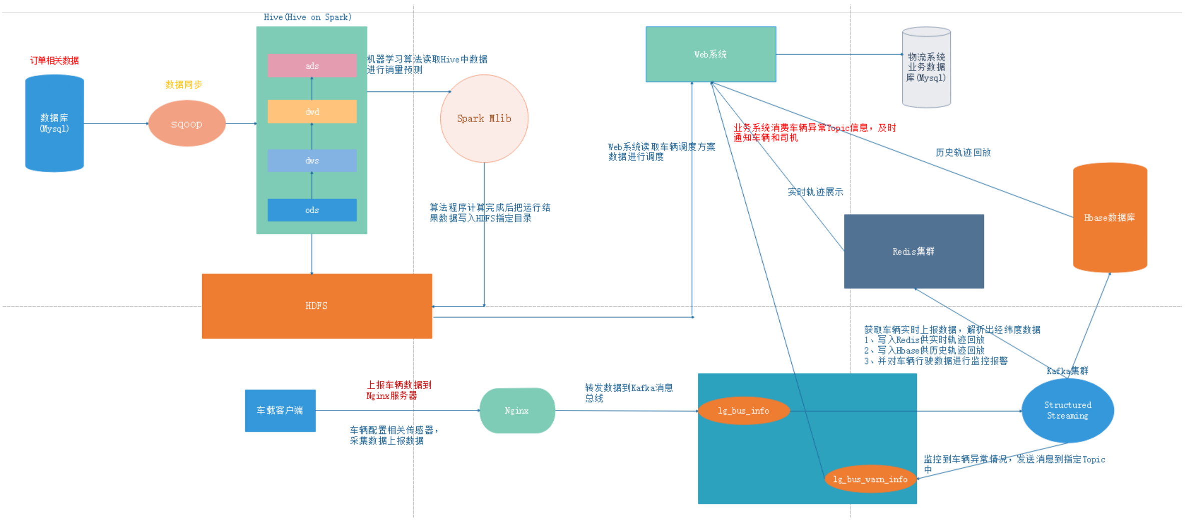
第 4 节 轨迹回放

需求

1）平台应支持货车轨迹回放，必须输入回放时间区间（时间区间<=1小时）

2）支持货车车牌号、调配编号，与查询

3 项目整体架构


Hive (Hive on Spark)
订单相关数据
数据库 (Mysql)
数据同步
sqoop
ads
dwd
dws
ods
机器学习算法读取Hive中数据进行销量预测
Spark Mlib
算法程序计算完成后把运行结果数据写入HDFS指定目录
HDFS
Web系统
物流系统业务数据库 (Mysql)
Web系统读取车辆调度方案数据进行调度
业务系统消费车辆异常Topic信息，及时通知车辆和司机
历史轨迹回放
实时轨迹展示
Redis集群
Hbase数据库
获取车辆实时上报数据，解析出经纬度数据
1、写入Redis供实时轨迹回放
2、写入Hbase供历史轨迹回放
3、并对车辆行驶数据进行监控报警
Kafka集群
车载客户端
上报车辆数据到Nginx服务器
车辆配置相关传感器，采集数据上报数据
Nginx
转发数据到Kafka消息总线
lg_bus_info
lg_bus_warn_info
Structured Streaming
监控到车辆异常情况，发送消息到指定Topic中

整个项目主要分为两部分

- 基于数仓历史数据仓储预测和车辆智能调度
- 车辆运行数据实时监控分析

技术框架选择：

CDH5.14.0

- 数据采集：Sqoop(1.4.6)
- 数仓：Hive(Hive on Spark):1.1.0-cdh5.14.0 ,Hadoop(HDFS,YARN):2.6.0-cdh5.14.0
- 机器学习库：Spark Mlib :默认1.6，升级到2.4
- 语言：Java，Scala
- 消息系统：kafka：1.0.1
- 缓存数据库：Redis：3.2
- 大数据数据库:Hbase：1.2.0-cdh5.14.0
- 实时处理引擎：Spark StructedStreaming:2.4.0
- Web系统：SpringCloud

4 CDH搭建

参考CDH搭建文档

5 数据采集

第 1 节 数据说明

智慧物流项目挑选出两个重要的场景，一个仓储预测，另一个是车货匹配。

其中仓储预测使用机器学习算法LightGBM;智能调度则使用动态规划算法。

算法开发的一般流程：

- 机器学习算法开发流程

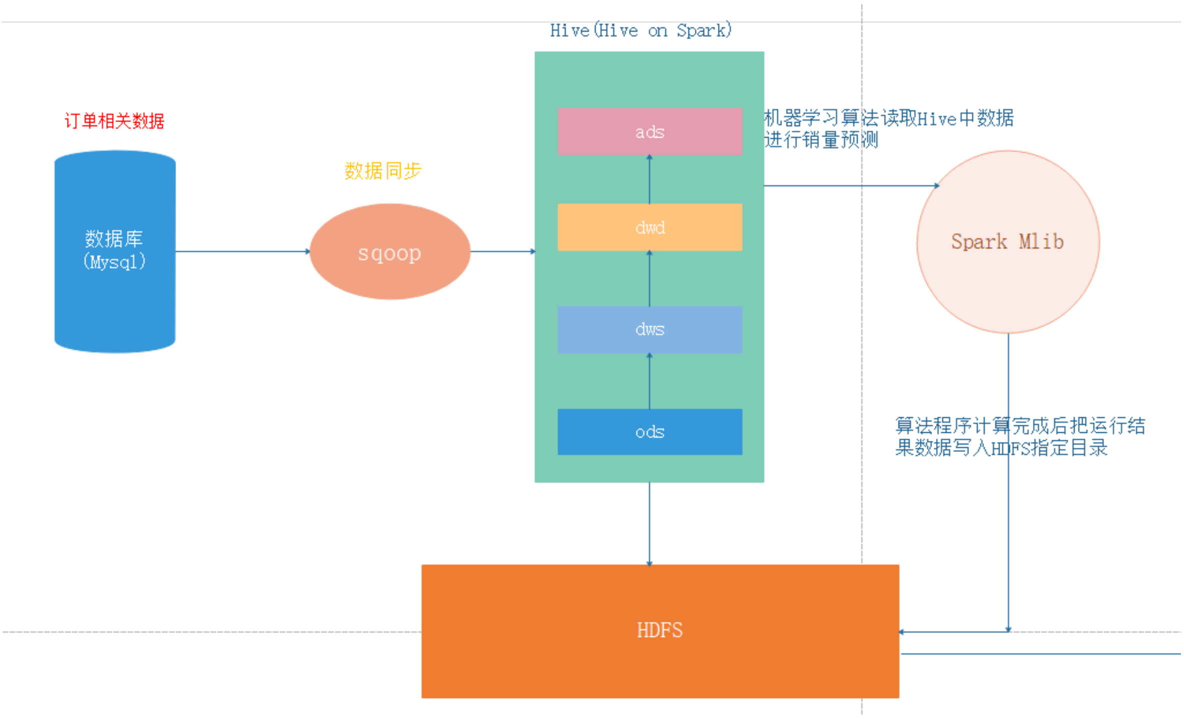
  1、准备数据
  　公司本身数据
  　购买数据
  2、明确问题
  　建立算法数据：根据数据类型划分应用种类
  3、数据基本处理
  　pd去处理数据(缺失值，合并表....)
  4、特征工程
  　特征进行处理(训练集，测试集、验证集)
  5、找寻合适的算法进行分析
  　1、估计器选择
  　2、调用fit(x_train,y_train)
  　3、调用　a)预测：y_predict=predict(x_test)　b)预测的准确率：score(x_test,y_test)
  6、模型的评估 ---->评估不合格，则考虑：1、换算法 2、调参数 3、特征工程再进一步处理
  7、模型实现预测，以API形式提供，或者直接提供计算结果数据

- 传统算法开发流程

  传统算法的开发流程相对于机器学习算法开发流程更简单，只保留前几个部分即可。

不管使用何种算法来解决问题，重要的都是我们能拿到的数据，只有充分利用现有的数据进行数据分析和处理，选择合适的算法才能获取到比较好的结果。在一个成熟的企业中对于算法的选择和实现以及评价调优都是由专门的算法工程师或者机器学习工程师来负责。我们大数据部门工程师主要负责提供满足算法部门要求的数据。我们以仓储预测场景来梳理一个算法的开发流程。

大体流程



Mysql数据库：lg_orders(订单表),lg_items(商品表)，lg_item_cats(商品分类表),lg_enterports(仓库表)

数据采集：使用Sqoop把数据同步到Hive数仓中

数仓ETL：对数据统一化，清洗，预处理

数据特征工程：Python/Scala

训练模型：SparkMlib

模型预测：SparkMlib

模型评价：损失函数

模型优化：调整

第 2 节 业务数据

根据算法部门的要求，仓储预测模型需要提供4类数据

sales_train;items;item_categories;entreports

- sales_train

  表示的是销售数据，有日期，月份,仓库，商品，价格和日销售量

统计出的是每个仓库每个商品的日销量数据，可以来自于**订单表(lg_orders);**同步到数仓中然后统计出指标数据；数据库是mysql；

- items

  表示的是商品信息，有商品名称，商品id,商品分类id

  可以来自商品表，数据库：mysql

- entreports

  仓库数据集，

  仓库id,仓库名称(地址信息)

  数据库：mysql
- item_category

  来源商品分类表，其中有商品分类id以及分类的名称

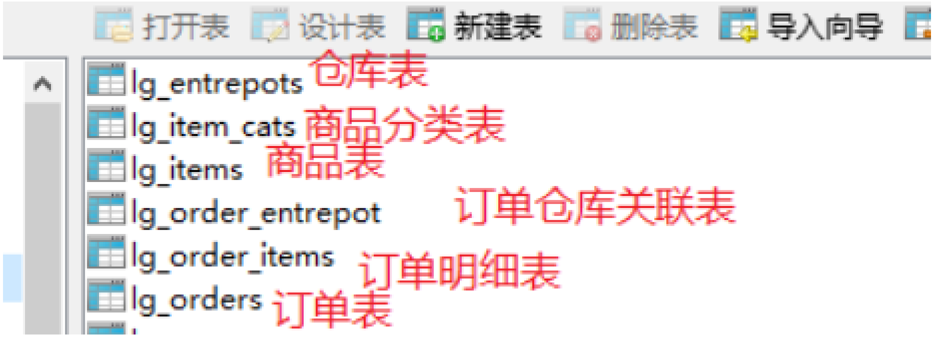
  数据来自表，来源是mysql.

选择hadoop1上的Mysql模拟业务数据库

创建lg_lgstic数据库

```
mysql> create database lg_logstic;
Query OK, 1 row affected (0.01 sec)

mysql> use lg_logstic;
Query OK, 1 row affected (0.01 sec)

#上传sql脚本文件到root/mysql_logstic，source执行
mysql> source /root/mysql_logstic/lg_logstic.sql;
Query OK, 1 row affected (0.01 sec)
```

数据库中表如下：



lg_entrepots 仓库表
lg_item_cats 商品分类表
lg_items 商品表
lg_order_entrepot 订单仓库关联表
lg_order_items 订单明细表
lg_orders 订单表

第 3 节 同步数据到Hive

数仓分层实现

数仓主要分为四层：

- ODS
- DWD
- DWS
- ADS

在Hive中分别创建四个数据库对应数仓的四层

```
create database lg_ods;
create database lg_dwd;
create database lg_dws;
create database lg_ads;
```

对于数据采集来说分为两部分，

- 一部分是离线数据采集，主要是Mysql中相关业务数据的采集，
- 一部分是实时数据采集，主要是车辆行驶相关数据采集。

3.1 离线数据采集

对于以上业务数据的采集来说，继续沿用离线数仓的数据采集方式，也就是针对不同的表类型选择不同的同步方式。

共有6张表

```
lg_orders,
lg_order_entrepot,
lg_order_items,
lg_items,
lg_entrepots,
lg_item_cats
```

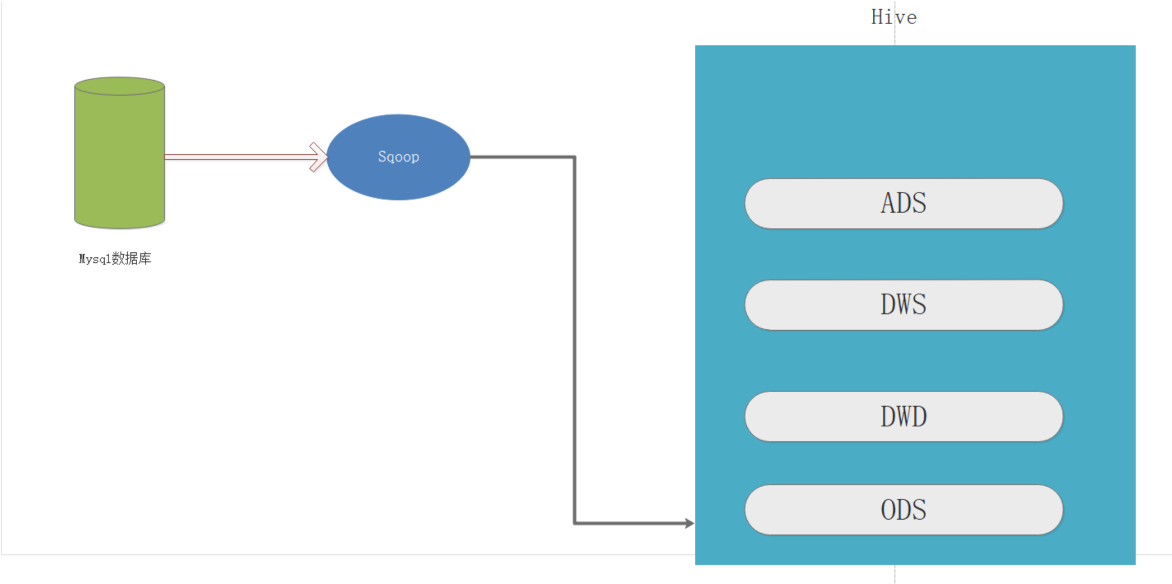
```

事实表：lg_orders,lg_order_items，lg_order_entrepot

维度表：lg_items,lg_entrepots,lg_item_cats

对于lg_orders,lg_items,lg_entrepots,lg_item_cats在数仓中主要以拉链表方式保存；

```

Hive创建ODS层表

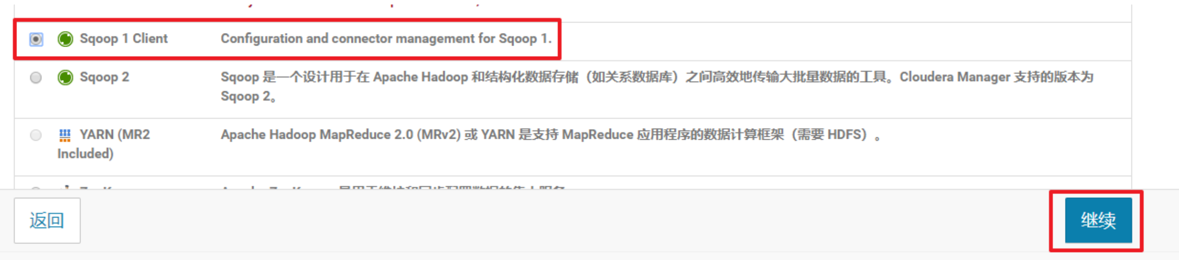
```
-- 创建ODS层商品分类表
drop table if exists `lg_ods`.`lg_item_cats`;
create  table `lg_ods`.`lg_item_cats`(
    catId                bigint,
    parentId             bigint,
    catName              string,
    isShow               bigint,
    isFloor              bigint,
    catSort              bigint,
    dataFlag             bigint,
    createTime           string,
    commissionRate       double,
    catImg               string,
    subTitle             string,
    simpleName           string,
    seoTitle             string,
    seoKeywords          string,
    seoDes               string,
    catListTheme         string,
    detailTheme          string,
    mobileCatListTheme   string,
    mobileDetailTheme    string,
    wechatCatListTheme   string,
    wechatDetailTheme    string,
    cat_level            bigint,
    modifyTime           string
)
```

```
partitioned by (dt string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

-- 创建ODS层商品表
drop table if exists `lg_ods`.`lg_items`;
create  table `lg_ods`.`lg_items`(
    itemsId              bigint,
    itemsSn              string,
    productNo            string,
    itemsName            string,
    itemsImg             string,
    entrepotId           bigint,
    itemsType            bigint,
    marketPrice          double,
    entrepotPrice        double,
    warnStock            bigint,
    itemsStock           bigint,
    itemsUnit            string,
    itemsTips            string,
    isSale               bigint,
    isBest               bigint,
    isHot                bigint,
    isNew                bigint,
    isRecom              bigint,
    itemsCatIdPath       string,
    itemsCatId           bigint,
    entrepotCatId1       bigint,
    entrepotCatId2       bigint,
    brandId              bigint,
    itemsDesc            string,
    itemsStatus          bigint,
    saleNum              bigint,
    saleTime             string,
    visitNum             bigint,
    appraiseNum          bigint,
    isSpec               bigint,
    gallery              string,
    itemsSeoKeywords     string,
    illegalRemarks       string,
    dataFlag             bigint,
    createTime           string,
    isFreeShipping       bigint,
    itemsSerachKeywords  string,
    modifyTime           string
)
partitioned by (dt string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

--创建ods层订单仓库关联表
drop table if exists `lg_ods`.`lg_order_entrepot`;
CREATE TABLE `lg_ods`.`lg_order_entrepot` (
  oeId  bigint,
  orderId bigint,
  itemId bigint,
  itemNums bigint,
  itemName string,
  entrepotId int,
  userName string,
  userAddress string,
  promotionJson string,
  createtime string,
  modifyTime string

) partitioned by (dt string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

-- 创建ODS层仓库表
drop table if exists `lg_ods`.`lg_entrepots`;
CREATE TABLE `lg_ods`.`lg_entrepots` (
  entrepotId  bigint,
  areaId bigint,
  entrepotName string,
  entrepotkeeper string,
  telephone string,
  entrepotImg string,
  entrepotTel string,
  entrepotQQ string,
  entrepotAddress string,
  invoiceRemarks string,
  serviceStartTime bigint,
  serviceEndTime bigint,
  freight bigint,
  entrepotAtive int,
  entrepotStatus int,
  statusDesc string,
  dataFlag int,
  createTime string ,
  modifyTime string

) partitioned by (dt string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

-- 创建ODS层订单表
drop table if exists `lg_ods`.`lg_orders`;
CREATE TABLE lg_ods.lg_orders (
  orderId        bigint,
  orderNo        string,
  userId         bigint,
  orderStatus    bigint,
  itemsMoney     double,
  deliverType    bigint,
  deliverMoney   double,
  totalMoney     double,
  realTotalMoney double,
  payType        bigint,
  isPay          bigint,
  areaId         bigint,
  userAddressId  bigint,
  areaIdPath     string,
  userName       string,
  userAddress    string,
  userPhone      string,
  orderScore     bigint,
  isInvoice      bigint,
  invoiceClient  string,
  orderRemarks   string,
  orderSrc       bigint,
  needPay        double,
  payRand        bigint,
  orderType      bigint,
  isRefund       bigint,
  isAppraise     bigint,
  cancelReason   bigint,
  rejectReason   bigint,
  rejectOtherReason string,
  isClosed          bigint,
  itemsSearchKeys   string,
  orderunique       string,
  isFromCart        string,
  receiveTime       string,
  deliveryTime      string,
  tradeNo           string,
  dataFlag          bigint,
  createTime        string,
  settlementId      bigint,
  commissionFee     double,
  scoreMoney        double,
  useScore          bigint,
  orderCode         string,
  extraJson         string,
  orderCodeTargetId bigint,
  noticeDeliver     bigint,
  invoiceJson       string,
  lockCashMoney     double,
  payTime           string,
  isBatch           bigint,
  totalPayFee       bigint,
  modifyTime        string
) partitioned by (dt string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;


-- 创建ODS层订单明细表
drop table if exists `lg_ods`.`lg_order_items`;
create  table `lg_ods`.`lg_order_items`(
    ogId            bigint,
    orderId         bigint,
    itemsId         bigint,
    itemsNum        bigint,
    itemsPrice      double,
    payPrice        double,
    itemsSpecId     bigint,
    itemsSpecNames  string,
    itemsName       string,
    itemsImg        string,
    extraJson       string,
    itemsType       bigint,
    commissionRate  double,
    itemsCode       string,
    promotionJson   string,
    createtime      string
)
partitioned by (dt string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED
AS TEXTFILE;
```

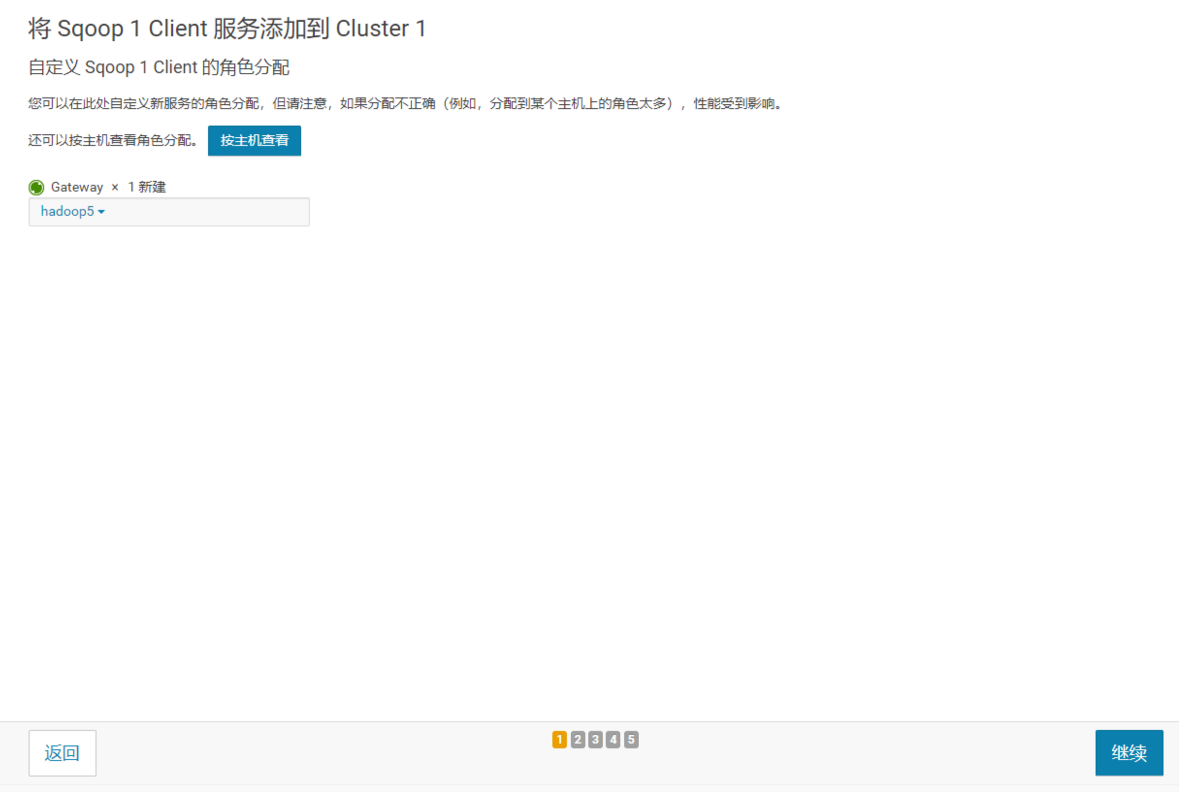
使用sqoop1实现采集；在CDH中其实建议使用sqoop1不建议使用sqoop2所以我们采用sqoop1实现数据的采集！



添加Sqoop服务



●	Sqoop 1 Client	Configuration and connector management for Sqoop 1.
○	Sqoop 2	Sqoop 是一个设计用于在 Apache Hadoop 和结构化数据存储（如关系数据库）之间高效地传输大批量数据的工具。Cloudera Manager 支持的版本为 Sqoop 2。
○	YARN (MR2 Included)	Apache Hadoop MapReduce 2.0 (MRv2) 或 YARN 是支持 MapReduce 应用程序的数据计算框架（需要 HDFS）。



将 Sqoop 1 Client 服务添加到 Cluster 1

自定义 Sqoop 1 Client 的角色分配

您可以在此处自定义新服务的角色分配，但请注意，如果分配不正确（例如，分配到某个主机上的角色太多），性能受到影响。

还可以按主机查看角色分配。

Gateway × 1 新建

hadoop5

sqoop的使用回顾

- --query

sqoop在导入数据时，可以使用--query搭配sql来指定查询条件，并且还需在sql中添加$CONDITIONS，来实现并行运行mr的功能。

如果使用--query但是不加$CONDITIONS会报错。如下

```
ERROR tool.ImportTool: Import failed: java.io.IOException: Query [select * from
Person where score>50] must contain '$CONDITIONS' in WHERE clause.
```

- 指定多个mapper导入数据；-m设置多个任务，比如 -m 10 注意必须加上-split-by参数用来指定如何划分数据

  补充：sqoop会向关系型数据库例如mysql发送一个命令：select max(id),min(id) from test ;会把max和min之间的区间平分10份，最后并行10个map去拉取数据。

数据源：hadoop1的Mysql数据库

目的地：Hive的ODS层表

编写sqoop导入任务，从mysql中抽取数据到Hive中

3.1.1 第一次全量导入

1、 导入lg_orders表

以lg_orders表为例：编写shell脚本

import_order_data.sh

```
#!/bin/bash
source /etc/profile
##如果第一个参数不为空，则作为工作日期使用
if [ -n "$1" ]
then
do_date=$1
else
##昨天日期，减一
do_date=`date -d "-1 day" +"%Y%m%d"`
fi

#定义sqoop命令位置，Hive命令位置，在hadoop2
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期
#do_date=20200827

#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
import_data(){
$sqoop import \
--connect jdbc:mysql://hadoop1:3306/lg_logstic \
--username root \
--password 123456 \
--target-dir /user/hive/warehouse/lg_ods.db/$1/dt=$do_date \
--delete-target-dir \
--query "$2 and  \$CONDITIONS" \
--num-mappers 1 \
--fields-terminated-by ',' \
--null-string '\\N' \
--null-non-string '\\N'
}


# 全量导入订单数据方法
import_lg_orders(){
  import_data lg_orders "select
                            *
                        from lg_orders
                        where 1=1"
}


#调用全量导入订单数据方法
import_lg_orders


#注意sqoop导入数据的方式，对于Hive分区表来说需要执行添加分区操作，数据才能被识别到
$Hive -e "alter table lg_ods.lg_orders add partition(dt='$do_date');"
```

对于其它表可以仿照上面的实现编写独立脚本，也可以全部综合到一起，如下

import_all_data.sh

```
#!/bin/bash
source /etc/profile
##如果第一个参数不为空，则作为工作日期使用
if [ -n "$1" ]
then
do_date=$1
else
##昨天日期，减一
do_date=`date -d "-1 day" +"%Y%m%d"`
fi
#定义sqoop命令位置，Hive命令位置，在hadoop2
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期


#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
import_data(){
$sqoop import \
--connect jdbc:mysql://hadoop1:3306/lg_logstic \
--username root \
--password 123456 \
--target-dir /user/hive/warehouse/lg_ods.db/$1/dt=$do_date \
--delete-target-dir \
--query "$2 and  \$CONDITIONS" \
--num-mappers 1 \
--fields-terminated-by ',' \
--null-string '\\N' \
--null-non-string '\\N'
}
```

```
# 全量导入订单数据方法
import_lg_orders(){
  import_data lg_orders "select
                            *
                        from lg_orders
                        where 1=1"
}

# 全量导入订单明细数据(包含商品)方法
import_lg_order_items(){
  import_data lg_order_items "select
                            *
                        from lg_order_items
                        where 1=1"
}

# 全量导入商品方法
import_lg_items(){
  import_data lg_items "select
                            *
                        from lg_items
                        where 1=1"
}

# 全量导入仓库方法
import_lg_entrepots(){
  import_data lg_entrepots "select
                            *
                        from lg_entrepots
                        where 1=1"
}

# 全量导入商品分类数据方法
import_lg_item_cats(){
  import_data lg_item_cats "select
                            *
                        from lg_item_cats
                        where 1=1"
}

# 全量导入订单仓库关联数据方法
import_lg_order_entrepot(){
  import_data lg_order_entrepot "select
                            *
                        from lg_order_entrepot
                        where 1=1"
}

#调用全量导入订单数据方法
import_lg_orders

#调用全量导入订单明细数据方法
import_lg_order_items

#调用全量导入商品数据方法
import_lg_items
```

```
#调用全量导入仓库数据方法
import_lg_entrepots

#调用全量导入商品分类数据方法
import_lg_item_cats
#调用全量导入订单仓库关联数据方法
import_lg_order_entrepot



#注意sqoop导入数据的方式，对于Hive分区表来说需要执行添加分区操作，数据才能被识别到
$Hive -e "alter table lg_ods.lg_orders add partition(dt='$do_date');
alter table lg_ods.lg_order_items add partition(dt='$do_date');
alter table lg_ods.lg_items add partition(dt='$do_date');
alter table lg_ods.lg_entrepots add partition(dt='$do_date');
alter table lg_ods.lg_item_cats add partition(dt='$do_date');
alter table lg_ods.lg_order_entrepot add partition(dt='$do_date');"
```

报错：缺少mysql驱动类

```
 cp mysql-connector-java.jar /opt/cloudera/parcels/CDH-5.14.0-1.cdh5.14.0.p0.24/lib/sqoop/lib/mysql-connector.jar
```

验证导入结果

```
select * from lg_ods.lg_orders limit 5;
select * from lg_ods.lg_order_items limit 5;
select * from lg_ods.lg_items limit 5;
select * from lg_ods.lg_entrepots limit 5;
select * from lg_ods.lg_item_cats limit 5;
select * from lg_ods.lg_order_entrepot limit 5;
```

3.1.2 增量导入

对于拉链表则ODS表需要每日拉取新增和更新的数据

```
--抽取每日新增和更新的订单数据
SELECT *
FROM lg_orders
WHERE DATE_FORMAT(modifyTime, '%Y%m%d') = '${do_date}';
```

shell脚本文件

import_incr_data.sh

```
#!/bin/bash

source /etc/profile
##如果第一个参数不为空，则作为工作日期使用
if [ -n "$1" ]
```

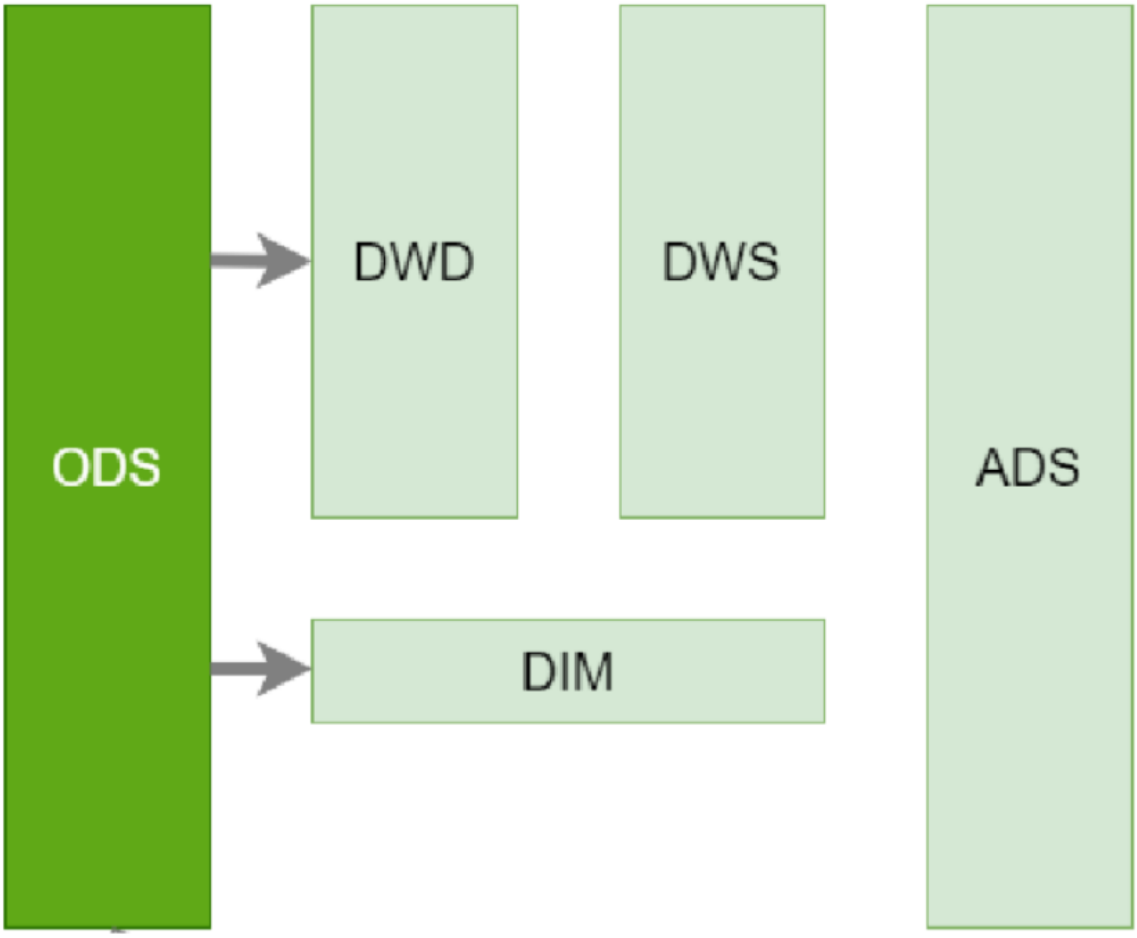
```
then
do_date=$1
else
##昨天日期，减一
do_date=`date -d "-1 day" +"%Y%m%d"`
fi

sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive

import_data(){
$sqoop import \
--connect jdbc:mysql://hadoop1:3306/lg_logstic \
--username root \
--password 123456 \
--target-dir /user/hive/warehouse/lg_ods.db/$1/dt=$do_date \
--delete-target-dir \
--query "$2 and  \$CONDITIONS" \
--num-mappers 1 \
--fields-terminated-by ',' \
--null-string '\\N' \
--null-non-string '\\N'
}

##导入新增订单数据的方法
import_lg_orders(){
  import_data lg_orders "SELECT *
                        FROM lg_orders
                    WHERE DATE_FORMAT(modifyTime, '%Y%m%d') = '${do_date}' "
}

# 导入新增订单明细数据(包含商品)方法
import_lg_order_items(){
  import_data lg_order_items "select
                            *
                        from lg_order_items
                        WHERE DATE_FORMAT(createTime, '%Y%m%d') = '${do_date}'"
}

# 导入新增和变化商品方法
import_lg_items(){
  import_data lg_items "select
                            *
                        from lg_items
                        WHERE DATE_FORMAT(modifyTime, '%Y%m%d') = '${do_date}'"
}

# 导入新增和变化仓库方法
import_lg_entrepots(){
  import_data lg_entrepots "select
                            *
                        from lg_entrepots
                         WHERE DATE_FORMAT(modifyTime, '%Y%m%d') = '${do_date}'"
}

# 导入新增商品分类数据方法
import_lg_item_cats(){
  import_data lg_item_cats "select
```

```
                            *
                        from lg_item_cats
                         WHERE DATE_FORMAT(modifyTime, '%Y%m%d') = '${do_date}'"
}

# 导入新增订单仓库关联数据方法
import_lg_order_entrepot(){
  import_data lg_order_entrepot "select
                            *
                        from lg_order_entrepot
                         WHERE DATE_FORMAT(modifyTime, '%Y%m%d') = '${do_date}'"
}

#导入新增订单明细数据方法
import_lg_order_items

#调用导入新增商品数据方法
import_lg_items

#调用导入新增仓库数据方法
import_lg_entrepots

#导入新增商品分类数据方法
import_lg_item_cats
#导入新增订单数据
import_lg_orders

#导入新增订单仓库关联数据
import_lg_order_entrepot

#执行Hive修复分区命令
$Hive -e "alter table lg_ods.lg_orders add partition(dt='$do_date');
alter table lg_ods.lg_order_items add partition(dt='$do_date');
alter table lg_ods.lg_items add partition(dt='$do_date');
alter table lg_ods.lg_entrepots add partition(dt='$do_date');
alter table lg_ods.lg_item_cats add partition(dt='$do_date');
alter table lg_ods.lg_order_entrepot add partition(dt='$do_date');"
```

验证导入结果

```
select * from lg_ods.lg_orders where dt='20200918' limit 5;
select * from lg_ods.lg_order_items where dt='20200918' limit 5;
select * from lg_ods.lg_items where dt='20200918' limit 5;
select * from lg_ods.lg_entrepots where dt='20200918' limit 5;
select * from lg_ods.lg_item_cats where dt='20200918' limit 5;
select * from lg_ods.lg_order_entrepot where dt='20200918' limit 5;
```

6 数据ETL



- ods-->dwd
  - lg_ods.lg_orders:订单表;
  - lg_ods.lg_order_items:订单明细表
  - lg_order_entrepot:订单仓库关联表

订单表是周期性事实表；为保留订单状态，使用拉链表进行处理；

订单明细表是普通的事实表，不涉及到状态变化和保留； 如果有数据清洗、数据转换的业务需求，使用每日增量同步到dwd层即可；

订单仓库关联表是普通的事实表，不涉及到状态变化和保留； 如果有数据清洗、数据转换的业务需求，使用每日增量同步到dwd层即可；

如果没有数据清洗、数据转换的业务需求，保留在ODS，不做任何变化。

- ods-->dim
  - lg_ods.lg_entrepots:仓库表
  - lg_ods.lg_items:商品表
  - lg_ods.lg_item_cats:商品分类表

仓库表，商品表，商品分类表都是作为维度表存在，使用拉链表进行处理；

创建对应数据库

```
--创建DIM层数据库
create database lg_dim;
```

备注:

- dwd,dws,dim层表底层均以parquet格式，因为底层使用spark引擎，spark程序对parquet格式的数据处理效率更好。
- 与维表不同，订单事实表的记录数非常多 订单有生命周期；
    - 订单的状态不可能永远处于变化之中（订单的生命周期一般在15天左 右）
    - 订单是一个拉链表，而且是分区表； 分区的目的：订单一旦终止，不会重复计算； 分区的条件：订单创建日期；保证相同笔订单存储在同一个分区

第 1 节 ODS->DWD

1.1 lg_dwd.fact_orders--订单拉链表

1.1.1 创建DWD层订单表

```
-- 创建DWD层订单表--拉链表
drop table if exists `lg_dwd`.`fact_orders`;
CREATE TABLE lg_dwd.fact_orders (
  orderId        bigint,
  orderNo        string,
  userId         bigint,
  orderStatus    bigint,
  itemsMoney     double,
  deliverType    bigint,
  deliverMoney   double,
  totalMoney     double,
  realTotalMoney double,
  payType        bigint,
  isPay          bigint,
  areaId         bigint,
  userAddressId  bigint,
  areaIdPath     string,
  userName       string,
  userAddress    string,
  userPhone      string,
  orderScore     bigint,
  isInvoice      bigint,
  invoiceClient  string,
  orderRemarks   string,
  orderSrc       bigint,
  needPay        double,
  payRand        bigint,
  orderType      bigint,
  isRefund       bigint,
  isAppraise     bigint,
  cancelReason   bigint,
  rejectReason   bigint,
  rejectOtherReason string,
  isClosed          bigint,
  itemsSearchKeys   string,
  orderunique       string,
  isFromCart        string,
  receiveTime       string,
  deliveryTime      string,
  tradeNo           string,
  dataFlag          bigint,
  createTime        string,
  settlementId      bigint,
  commissionFee     double,
  scoreMoney        double,
  useScore          bigint,
  orderCode         string,
  extraJson         string,
  orderCodeTargetId bigint,
  noticeDeliver     bigint,
  invoiceJson       string,
  lockCashMoney     double,
  payTime           string,
  isBatch           bigint,
  totalPayFee       bigint,
  modifytime        string,
  start_date       string,
  end_date         string
) partitioned by (dt string) STORED AS PARQUET ;


-- 创建DWD层临时订单表--拉链表
drop table if exists `lg_dwd`.`tmp_fact_orders`;
CREATE TABLE lg_dwd.tmp_fact_orders (
  orderId        bigint,
  orderNo        string,
  userId         bigint,
  orderStatus    bigint,
  itemsMoney     double,
  deliverType    bigint,
  deliverMoney   double,
  totalMoney     double,
  realTotalMoney double,
  payType        bigint,
  isPay          bigint,
  areaId         bigint,
  userAddressId  bigint,
  areaIdPath     string,
  userName       string,
  userAddress    string,
  userPhone      string,
  orderScore     bigint,
  isInvoice      bigint,
  invoiceClient  string,
  orderRemarks   string,
  orderSrc       bigint,
  needPay        double,
  payRand        bigint,
  orderType      bigint,
  isRefund       bigint,
  isAppraise     bigint,
  cancelReason   bigint,
  rejectReason   bigint,
  rejectOtherReason string,
  isClosed          bigint,
  itemsSearchKeys   string,
  orderunique       string,
  isFromCart        string,
  receiveTime       string,
  deliveryTime      string,
  tradeNo           string,
  dataFlag          bigint,
  createTime        string,
  settlementId      bigint,
  commissionFee     double,
  scoreMoney        double,
  useScore          bigint,
  orderCode         string,
  extraJson         string,
  orderCodeTargetId bigint,
  noticeDeliver     bigint,
  invoiceJson       string,
  lockCashMoney     double,
  payTime           string,
  isBatch           bigint,
  totalPayFee       bigint,
  modifytime        string,
  start_date       string,
  end_date         string
) partitioned by (dt string) STORED AS PARQUET ;
```

1.1.2 订单拉链操作

1. 第一次导入拉链表

开启Hive的动态分区，并根据数据的createtime字段进行分区划分，同一天创建的订单放在同一分区！！

```
#开启动态分区，默认是false
#开启允许所有分区都是动态的，否则必须要有静态分区才能使用
set hive.exec.dynamici.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
```

订单表数据：ODS层导入DWD层

(1) 之前全部历史数据进入拉链表

```
insert overwrite table lg_dwd.fact_orders partition(dt)
select
orderid          ,
orderno          ,
userid           ,
orderstatus      ,
itemsmoney       ,
delivertype      ,
delivermoney     ,
totalmoney       ,
realtotalmoney   ,
paytype          ,
ispay            ,
areaid           ,
useraddressid    ,
areaidpath       ,
username         ,
useraddress      ,
userphone        ,
orderscore       ,
isinvoice        ,
invoiceclient    ,
orderremarks     ,
ordersrc         ,
needpay          ,
payrand          ,
ordertype        ,
isrefund         ,
isappraise       ,
cancelreason     ,
rejectreason     ,
rejectotherreason,
isclosed         ,
itemssearchkeys  ,
orderunique      ,
isfromcart       ,
receivetime      ,
deliverytime     ,
tradeno          ,
dataflag         ,
createtime       ,
settlementid     ,
commissionfee    ,
scoremoney       ,
usescore         ,
ordercode        ,
extrajson        ,
ordercodetargetid,
noticedeliver    ,
invoicejson      ,
lockcashmoney    ,
paytime          ,
isbatch          ,
totalpayfee      ,
modifytime       ,
--增加开始时间
date_format(modifytime,'yyyy-MM-dd') as start_date,
--增加结束时间
'9999-12-31' as end_date,
--指定动态分区使用的字段，动态分区的用法：就是查询字段的最后一个字段Hive表进行解析然后存入指定分区
--此次数据分区按照订单的创建时间
date_format(createtime,'yyyyMMdd')
from lg_ods.lg_orders where dt="20200609";
```

2. 拉链表与每日合并

```
insert overwrite table lg_dwd.tmp_fact_orders partition(dt)
```

```
--新增数据的更新
select
orderid          ,
orderno          ,
userid           ,
orderstatus      ,
itemsmoney       ,
delivertype      ,
delivermoney     ,
totalmoney       ,
realtotalmoney   ,
paytype          ,
ispay            ,
areaid           ,
useraddressid    ,
areaidpath       ,
username         ,
useraddress      ,
userphone        ,
orderscore       ,
isinvoice        ,
invoiceclient    ,
orderremarks     ,
ordersrc         ,
needpay          ,
payrand          ,
ordertype        ,
isrefund         ,
isappraise       ,
cancelreason     ,
rejectreason     ,
rejectotherreason,
isclosed         ,
itemssearchkeys  ,
orderunique      ,
isfromcart       ,
receivetime      ,
deliverytime     ,
tradeno          ,
dataflag         ,
createtime       ,
settlementid     ,
commissionfee    ,
scoremoney       ,
usescore         ,
ordercode        ,
extrajson        ,
ordercodetargetid,
noticedeliver    ,
invoicejson      ,
lockcashmoney    ,
paytime          ,
isbatch          ,
totalpayfee      ,
modifytime       ,
--增加开始时间
date_format(modifyTime,'yyyy-MM-dd') as start_date,
--增加结束时间
```

```
'9999-12-31' as end_date,
--指定动态分区使用的字段，动态分区的用法：就是查询字段的最后一个字段Hive表进行解析然后存入指定分区
--此次数据分区按照订单的创建时间
date_format(createtime,'yyyyMMdd') as part
from lg_ods.lg_orders where dt="20200610"
union all
--历史拉链表更新数据
select
dw.orderid           as orderid              ,
dw.orderno           as orderno              ,
dw.userid            as userid               ,
dw.orderstatus       as orderstatus          ,
dw.itemsmoney        as itemsmoney           ,
dw.delivertype       as delivertype          ,
dw.delivermoney      as delivermoney         ,
dw.totalmoney        as totalmoney           ,
dw.realtotalmoney    as realtotalmoney       ,
dw.paytype           as paytype              ,
dw.ispay             as ispay                ,
dw.areaid            as areaid               ,
dw.useraddressid     as useraddressid        ,
dw.areaidpath        as areaidpath           ,
dw.username          as username             ,
dw.useraddress       as useraddress          ,
dw.userphone         as userphone            ,
dw.orderscore        as orderscore           ,
dw.isinvoice         as isinvoice            ,
dw.invoiceclient     as invoiceclient        ,
dw.orderremarks      as orderremarks         ,
dw.ordersrc          as ordersrc             ,
dw.needpay           as needpay              ,
dw.payrand           as payrand              ,
dw.ordertype         as ordertype            ,
dw.isrefund          as isrefund             ,
dw.isappraise        as isappraise           ,
dw.cancelreason      as cancelreason         ,
dw.rejectreason      as rejectreason         ,
dw.rejectotherreason as rejectotherreason    ,
dw.isclosed          as isclosed             ,
dw.itemssearchkeys   as itemssearchkeys      ,
dw.orderunique       as orderunique          ,
dw.isfromcart        as isfromcart           ,
dw.receivetime       as receivetime          ,
dw.deliverytime      as deliverytime         ,
dw.tradeno           as tradeno              ,
dw.dataflag          as dataflag             ,
dw.createtime        as createtime           ,
dw.settlementid      as settlementid         ,
dw.commissionfee     as commissionfee        ,
dw.scoremoney        as scoremoney           ,
dw.usescore          as usescore             ,
dw.ordercode         as ordercode            ,
dw.extrajson         as extrajson            ,
dw.ordercodetargetid as ordercodetargetid    ,
dw.noticedeliver     as noticedeliver        ,
dw.invoicejson       as invoicejson          ,
dw.lockcashmoney     as lockcashmoney        ,
dw.paytime           as paytime              ,
dw.isbatch           as isbatch              ,
dw.totalpayfee       as totalpayfee          ,
dw.modifytime        as modifytime           ,
dw.start_date        as start_date           ,
--修改end_date
case when ods.orderid is not null and dw.end_date ='9999-12-31'
then '2020-06-09'
else dw.end_date
end as end_date,
--动态分区需要的字段
dw.dt as part
from
lg_dwd.fact_orders  dw
left join
(select * from lg_ods.lg_orders where dt ='20200610') ods
on dw.orderid=ods.orderid  ;
```

注意：使用union all要保证两个子查询得到的字段名称一致！！

临时表中数据插入拉链表中

```
insert overwrite table lg_dwd.fact_orders partition(dt) select * from
lg_dwd.tmp_fact_orders ;
```

1.2 lg_order_items--订单明细表

对于订单明细表,订单仓库信息的ETL来说，由于该表中数据是作为订单表的补充不会变化，也不涉及到数据清洗，数据处理等需求，所以ods->dwd可以不做。直接从ods层获取数据即可。

第 2 节 ODS-->DIM

主要以下三张表需要从ODS转到DIM层，并且使用拉链表保存。

- lg_ods.lg_entrepots:仓库表
- lg_ods.lg_items:商品表
- lg_ods.lg_item_cats:商品分类表

仓库表，商品表，商品分类表都是作为维度表存在，使用拉链表进行处理;

2.1 lg_dim.lg_items 商品表

创建DIM层商品表

```
use lg_dim;

DROP TABLE IF EXISTS `lg_dim`.`lg_dim_items`;
CREATE TABLE `lg_dim`.`lg_dim_items`(
    goodsId bigint,
    goodsSn string,
    productNo string,
    goodsName string,
    goodsImg string,
```

```sql
    shopId bigint,
    goodsType bigint,
    marketPrice double,
    shopPrice double,
    warnStock bigint,
    goodsStock bigint,
    goodsUnit string,
    goodsTips string,
    isSale bigint,
    isBest bigint,
    isHot bigint,
    isNew bigint,
    isRecom bigint,
    goodsCatIdPath string,
    goodsCatId bigint,
    shopCatId1 bigint,
    shopCatId2 bigint,
    brandId bigint,
    goodsDesc string,
    goodsStatus bigint,
    saleNum bigint,
    saleTime string,
    visitNum bigint,
    appraiseNum bigint,
    isSpec bigint,
    gallery string,
    goodsSeoKeywords string,
    illegalRemarks string,
    dataFlag bigint,
    createTime string,
    isFreeShipping bigint,
    goodsSerachKeywords string,
    modifyTime string,
    start_date string,
    end_date string
)
STORED AS PARQUET;

DROP TABLE IF EXISTS `lg_dim`.`tmp_lg_dim_items`;
CREATE TABLE `lg_dim`.`tmp_lg_dim_items`(
    goodsId bigint,
    goodsSn string,
    productNo string,
    goodsName string,
    goodsImg string,
    shopId bigint,
    goodsType bigint,
    marketPrice double,
    shopPrice double,
    warnStock bigint,
    goodsStock bigint,
    goodsUnit string,
    goodsTips string,
    isSale bigint,
    isBest bigint,
    isHot bigint,
    isNew bigint,
    isRecom bigint,
    goodsCatIdPath string,
    goodsCatId bigint,
    shopCatId1 bigint,
    shopCatId2 bigint,
    brandId bigint,
    goodsDesc string,
    goodsStatus bigint,
    saleNum bigint,
    saleTime string,
    visitNum bigint,
    appraiseNum bigint,
    isSpec bigint,
    gallery string,
    goodsSeoKeywords string,
    illegalRemarks string,
    dataFlag bigint,
    createTime string,
    isFreeShipping bigint,
    goodsSerachKeywords string,
    modifyTime string,
    start_date string,
    end_date string
)
STORED AS PARQUET;
```

2.1.1 商品拉链表操作

1. 第一次导入拉链表

```
insert overwrite table `lg_dim`.`lg_dim_items`
select
itemsid            as  goodsId,
itemssn            as  goodsSn,
productno          as  productNo,
itemsname          as  goodsName,
itemsimg           as  goodsImg,
entrepotid         as  shopId,
itemstype          as  goodsType,
marketprice        as   marketPrice,
entrepotprice      as   shopPrice,
warnstock          as   warnStock,
itemsstock         as  goodsStock,
itemsunit          as   goodsUnit,
itemstips          as   goodsTips,
issale             as   isSale,
isbest             as   isBest,
ishot              as   isHot,
isnew              as   isNew,
isrecom            as   isRecom,
itemscatidpath     as   goodsCatIdPath,
itemscatid         as   goodsCatId,
entrepotcatid1     as   shopCatId1,
entrepotcatid2     as   shopCatId2,
brandid            as    brandId,
itemsdesc          as    goodsDesc,
itemsstatus        as    goodsStatus,
salenum            as    saleNum,
saletime           as    saleTime,
visitnum           as    visitNum,
appraisenum        as    appraiseNum,
isspec             as    isSpec,
gallery            as    gallery,
itemsseokeywords   as    goodsSeoKeywords,
illegalremarks     as    illegalRemarks,
dataflag           as    dataFlag,
createtime         as    createTime,
isfreeshipping     as    isFreeShipping,
itemsserachkeywords as    goodsSerachKeywords,
modifytime         as    modifyTime,
    case when modifyTime is not null
        then from_unixtime(unix_timestamp(modifyTime, 'yyyy-MM-dd
HH:mm:ss'),'yyyy-MM-dd')
        else from_unixtime(unix_timestamp(createTime, 'yyyy-MM-dd HH:mm:ss'),
'yyyy-MM-dd')
        end as start_date,
    '9999-12-31' as end_date
from
    `lg_ods`.`lg_items`
where dt = '20200609';
```

2. 每日合并拉链表

```
-- 将历史数据 当日数据合并加载到临时表
drop table if exists `lg_dim`.`tmp_lg_dim_items`;
create table `lg_dim`.`tmp_lg_dim_items`
as
select
    dim.goodsId,
    dim.goodsSn,
    dim.productNo,
    dim.goodsName,
    dim.goodsImg,
    dim.shopId,
    dim.goodsType,
    dim.marketPrice,
    dim.shopPrice,
    dim.warnStock,
    dim.goodsStock,
    dim.goodsUnit,
    dim.goodsTips,
    dim.isSale,
    dim.isBest,
    dim.isHot,
    dim.isNew,
    dim.isRecom,
    dim.goodsCatIdPath,
    dim.goodsCatId,
    dim.shopCatId1,
    dim.shopCatId2,
    dim.brandId,
    dim.goodsDesc,
    dim.goodsStatus,
    dim.saleNum,
    dim.saleTime,
    dim.visitNum,
    dim.appraiseNum,
    dim.isSpec,
    dim.gallery,
    dim.goodsSeoKeywords,
    dim.illegalRemarks,
    dim.dataFlag,
    dim.createTime,
    dim.isFreeShipping,
    dim.goodsSerachKeywords,
    dim.modifyTime,
    dim.start_date,
    case when dim.end_date >= '9999-12-31' and ods.itemsid is not null
        then '2020-06-09'
        else dim.end_date
        end as end_date
from
    `lg_dim`.`lg_dim_items` dim
    left join
    (select * from `lg_ods`.`lg_items` where dt='20200610') ods
    on dim.goodsId = ods.itemsid
union all
select
  itemsid            as  goodsId,
itemssn            as  goodsSn,
productno          as  productNo,
itemsname          as  goodsName,
itemsimg           as  goodsImg,
entrepotid         as  shopId,
itemstype          as  goodsType,
marketprice        as   marketPrice,
entrepotprice      as   shopPrice,
warnstock          as   warnStock,
itemsstock         as  goodsStock,
itemsunit          as   goodsUnit,
itemstips          as   goodsTips,
issale             as   isSale,
isbest             as   isBest,
ishot              as   isHot,
isnew              as   isNew,
isrecom            as   isRecom,
itemscatidpath     as   goodsCatIdPath,
itemscatid         as   goodsCatId,
entrepotcatid1     as   shopCatId1,
entrepotcatid2     as   shopCatId2,
brandid            as    brandId,
itemsdesc          as    goodsDesc,
itemsstatus        as    goodsStatus,
salenum            as    saleNum,
saletime           as    saleTime,
visitnum           as    visitNum,
appraisenum        as    appraiseNum,
isspec             as    isSpec,
gallery            as    gallery,
itemsseokeywords   as    goodsSeoKeywords,
illegalremarks     as    illegalRemarks,
dataflag           as    dataFlag,
createtime         as    createTime,
isfreeshipping     as    isFreeShipping,
itemsserachkeywords as    goodsSerachKeywords,
modifytime         as    modifyTime,
    case when modifyTime is not null
        then from_unixtime(unix_timestamp(modifyTime, 'yyyy-MM-dd
HH:mm:ss'),'yyyy-MM-dd')
        else from_unixtime(unix_timestamp(createTime, 'yyyy-MM-dd HH:mm:ss'),
'yyyy-MM-dd')
        end as start_date,
    '9999-12-31' as end_date
from
    `lg_ods`.`lg_items`
where dt = '20200610';
```

临时表中数据插入拉链表

```
insert overwrite table lg_dim.lg_dim_items select * from
lg_dim.tmp_lg_dim_items;
```

2.2 lg_dim.dim_product_cat 商品分类表

lg_dim.dim_product_cat表为商品分类数据，在数仓中使用拉链表存储。

2.2.1 创建商品分类拉链表

```
drop table if exists lg_dim.dim_product_cat;
create  table lg_dim.dim_product_cat(
    catId               bigint,
    parentId            bigint,
    catName             string,
    isShow              bigint,
    isFloor             bigint,
    catSort             bigint,
    dataFlag            bigint,
    createTime          string,
    commissionRate      double,
    catImg              string,
    subTitle            string,
    simpleName          string,
    seoTitle            string,
    seoKeywords         string,
    seoDes              string,
    catListTheme        string,
    detailTheme         string,
    mobileCatListTheme  string,
    mobileDetailTheme   string,
    wechatCatListTheme  string,
    wechatDetailTheme   string,
    cat_level           bigint,
    modifyTime          string,
    start_date          string,
    end_date            string
)
 STORED AS PARQUET;


 drop table if exists lg_dim.tmp_dim_product_cat;
create  table lg_dim.tmp_dim_product_cat(
    catId                   bigint,
    parentId                bigint,
    catName                 string,
    isShow                  bigint,
    isFloor                 bigint,
    catSort                 bigint,
    dataFlag                bigint,
    createTime              string,
    commissionRate          double,
    catImg                  string,
    subTitle                string,
    simpleName              string,
    seoTitle                string,
    seoKeywords             string,
    seoDes                  string,
    catListTheme            string,
    detailTheme             string,
    mobileCatListTheme      string,
    mobileDetailTheme       string,
    wechatCatListTheme      string,
    wechatDetailTheme       string,
    cat_level               bigint,
    modifyTime              string,
    start_date              string,
    end_date                string
)
 STORED AS PARQUET;
```

2.2.2 分类表拉链操作

1. 第一次导入拉链表

```
insert overwrite table lg_dim.dim_product_cat
select
catid                   ,
parentid                ,
catname                 ,
isshow                  ,
isfloor                 ,
catsort                 ,
dataflag                ,
createtime              ,
commissionrate          ,
catimg                  ,
subtitle                ,
simplename              ,
seotitle                ,
seokeywords             ,
seodes                  ,
catlisttheme            ,
detailtheme             ,
mobilecatlisttheme ,
mobiledetailtheme  ,
wechatcatlisttheme ,
wechatdetailtheme  ,
cat_level          ,

modifytime         ,
    case when modifyTime is not null
        then from_unixtime(unix_timestamp(modifyTime, 'yyyy-MM-dd
HH:mm:ss'),'yyyy-MM-dd')
        else from_unixtime(unix_timestamp(createTime, 'yyyy-MM-dd HH:mm:ss'),
'yyyy-MM-dd')
        end as start_date,
    '9999-12-31' as end_date
from
    `lg_ods`.`lg_item_cats`
where dt = '20200609';
```

2. 每日合并拉链表

```
drop table if exists lg_dim.tmp_dim_product_cat;
create table lg_dim.tmp_dim_product_cat as
select
dim.catid              ,
dim.parentid           ,
dim.catname            ,
dim.isshow             ,
dim.isfloor            ,
dim.catsort            ,
dim.dataflag           ,
dim.createtime         ,
dim.commissionrate     ,
dim.catimg             ,
dim.subtitle           ,
dim.simplename         ,
dim.seotitle           ,
dim.seokeywords        ,
dim.seodes             ,
dim.catlisttheme       ,
dim.detailtheme        ,
dim.mobilecatlisttheme ,
dim.mobiledetailtheme  ,
dim.wechatcatlisttheme ,
dim.wechatdetailtheme  ,
dim.cat_level          ,

dim.modifytime         ,
dim.start_date         ,
case when dim.end_date >= '9999-12-31' and ods.catid is not null
        then '2020-06-097'
        else dim.end_date
        end as end_date
from
    `lg_dim`.`dim_product_cat` dim
    left join
    (select * from `lg_ods`.`lg_item_cats` where dt='20200610') ods
    on dim.catid = ods.catid
union all
select
catid              ,
parentid           ,
catname            ,
isshow             ,
isfloor            ,
catsort            ,
dataflag           ,
createtime         ,
commissionrate     ,
catimg             ,
subtitle           ,
simplename         ,
seotitle           ,
seokeywords        ,
seodes             ,
catlisttheme       ,
detailtheme        ,
mobilecatlisttheme ,
mobiledetailtheme  ,
wechatcatlisttheme ,
wechatdetailtheme  ,
cat_level          ,

modifytime         ,
    case when modifyTime is not null
        then from_unixtime(unix_timestamp(modifyTime, 'yyyy-MM-dd
HH:mm:ss'),'yyyy-MM-dd')
        else from_unixtime(unix_timestamp(createTime, 'yyyy-MM-dd HH:mm:ss'),
'yyyy-MM-dd')
        end as start_date,
    '9999-12-31' as end_date
from
    `lg_ods`.`lg_item_cats`
where dt = '20200610';
```

导入拉链表

```
insert overwrite table lg_dim.dim_product_cat select * from
lg_dim.tmp_dim_product_cat ;
```

2.3、创建仓库拉链表

2.3.1 创建仓库拉链表

```

drop table if exists `lg_dim`.`dim_lg_entrepots`;
CREATE TABLE `lg_dim`.`dim_lg_entrepots` (
  entrepotId  bigint,
  areaId bigint,
  entrepotName string,
  entrepotkeeper string,
  telephone string,
  entrepotImg string,
  entrepotTel string,
  entrepotQQ string,
  entrepotAddress string,
  invoiceRemarks string,
  serviceStartTime bigint,
  serviceEndTime bigint,
  freight bigint,
  entrepotAtive int,
  entrepotStatus int,
  statusDesc string,
  dataFlag int,
  createTime string ,
  modifyTime string,
  start_date   string,
  end_date       string

)  STORED AS PARQUET;


drop table if exists `lg_dim`.`tmp_dim_lg_entrepots`;
CREATE TABLE `lg_dim`.`tmp_dim_lg_entrepots` (
  entrepotId  bigint,
  areaId bigint,
  entrepotName string,
  entrepotkeeper string,
  telephone string,
  entrepotImg string,
  entrepotTel string,
  entrepotQQ string,
  entrepotAddress string,
  invoiceRemarks string,
  serviceStartTime bigint,
  serviceEndTime bigint,
  freight bigint,
  entrepotAtive int,
  entrepotStatus int,
  statusDesc string,
  dataFlag int,
  createTime string ,
  modifyTime string,
  start_date   string,
  end_date       string

)  STORED AS PARQUET;
```

2.3.2 仓库表拉链操作

1. 第一次导入拉链表

```
insert overwrite table lg_dim.dim_lg_entrepots
select
entrepotId  ,
areaId       ,
entrepotName  ,
entrepotkeeper  ,
telephone      ,
entrepotImg  ,
entrepotTel  ,
entrepotQQ      ,
entrepotAddress ,
invoiceRemarks  ,
serviceStartTime,
serviceEndTime  ,
freight   ,
entrepotAtive   ,
entrepotStatus  ,
statusDesc      ,
dataFlag   ,
createTime      ,
modifyTime      ,
    case when modifyTime is not null
        then from_unixtime(unix_timestamp(modifyTime, 'yyyy-MM-dd
HH:mm:ss'),'yyyy-MM-dd')
        else from_unixtime(unix_timestamp(createTime, 'yyyy-MM-dd HH:mm:ss'),
'yyyy-MM-dd')
        end as start_date,
    '9999-12-31' as end_date
from
    `lg_ods`.`lg_entrepots`
where dt = '20200609';
```

2. 每日合并拉链表

```
drop table if exists lg_dim.tmp_dim_product_cat;
create table lg_dim.tmp_dim_product_cat as
select
dim.catid              ,
dim.parentid           ,
dim.catname            ,
dim.isshow             ,
dim.isfloor            ,
dim.catsort            ,
dim.dataflag           ,
dim.createtime         ,
dim.commissionrate     ,
dim.catimg             ,
dim.subtitle           ,
dim.simplename         ,
dim.seotitle           ,
dim.seokeywords        ,
dim.seodes             ,
dim.catlisttheme       ,
dim.detailtheme        ,
dim.mobilecatlisttheme ,
dim.mobiledetailtheme  ,
dim.wechatcatlisttheme ,
dim.wechatdetailtheme  ,
dim.cat_level          ,

dim.modifytime         ,
dim.start_date         ,
case when dim.end_date >= '9999-12-31' and ods.catid is not null
        then '2020-06-09'
        else dim.end_date
        end as end_date
from
    `lg_dim`.`dim_product_cat` dim
    left join
    (select * from `lg_ods`.`lg_item_cats` where dt='20200610') ods
    on dim.catid = ods.catid
union all
select
catid              ,
parentid           ,
catname            ,
isshow             ,
isfloor            ,
catsort            ,
dataflag           ,
createtime         ,
commissionrate     ,
catimg             ,
subtitle           ,
simplename         ,
seotitle           ,
seokeywords        ,
seodes             ,
catlisttheme       ,
detailtheme        ,
mobilecatlisttheme ,
mobiledetailtheme  ,
wechatcatlisttheme ,
wechatdetailtheme  ,
cat_level          ,

modifytime         ,
    case when modifyTime is not null
        then from_unixtime(unix_timestamp(modifyTime, 'yyyy-MM-dd
HH:mm:ss'),'yyyy-MM-dd')
        else from_unixtime(unix_timestamp(createTime, 'yyyy-MM-dd HH:mm:ss'),
'yyyy-MM-dd')
        end as start_date,
    '9999-12-31' as end_date
from
    `lg_ods`.`lg_item_cats`
where dt = '20200610';
```

导入拉链表

```
insert overwrite table lg_dim.dim_lg_entrepots select * from
lg_dim.tmp_dim_lg_entrepots ;
```

7 数据导出

针对仓储预测准备相关数据

根据算法部门的要求，仓储预测模型需要提供4个数据集

sales_train.csv;items.csv;item_categories.csv;entreports.csv

- sales_train

  历史销售数据，有日期字段，月份,仓库，商品，价格和日销售量

	date	block_num	entrepot_id	item_id	item_price	item_cnt_day
0	2017-01-02	0	59	22154	999.00	1.0
1	2017-01-02	0	25	2552	899.00	1.0
2	2017-01-02	0	25	2552	899.00	1.0
3	2017-01-02	0	25	2554	1709.05	1.0
4	2017-01-02	0	25	2555	1099.00	1.0

统计出的是每个仓库每个商品的日销量数据，

- items.csv

  表示的是商品信息，有商品名称，仓库id,商品分类id

- entrepots.csv

  仓库数据集，仓库id,仓库名称

- item_category

  来源商品分类表，其中有商品分类id以及分类的名称

第 一 节 导出销量数据

观察sales数据，其中有商品id,商品价格，商品销量数据，这些字段可以从订单明细表中获取，

此外还需要提供仓库id，这个字段需要从订单仓库关联表中获取

最后关于月份的编号需要额外进行处理获取；

```
#创建tmp层
create database lg_tmp;
# 订单明细表关联订单仓库表，基于订单id与商品id
use lg_tmp
drop table if exists lg_tmp.tmp_order_item_entrepot;

create table tmp_order_item_entrepot
as
select t1.itemsid as itemId,
t1.itemsname as itemName,
t2.entrepotId as entrepotId,
t1.itemsNum as itemNum,
t1.itemsPrice as itemPrice,
t1.dt as date
from
lg_ods.lg_order_items t1
```

```
join
lg_ods.lg_order_entrepot t2
on t1.orderId=t2.orderId
and
t1.itemsid =t2.itemid;
#对上面结果数据进行汇总，按照天统计每个商品的销量
use lg_tmp;
drop table if exists lg_tmp.tmp_order_item_entrepot_day;
create table tmp_order_item_entrepot_day
as
select
itemid,
entrepotid,
date,
sum(itemNum) as itemNum,
itemprice
from lg_tmp.tmp_order_item_entrepot
group by
itemId,
entrepotid,
date ,
itemprice;

#验证结果
select * from lg_tmp.tmp_order_item_entrepot_day limit 5;
#增加月份信息 使用排名函数，需要对数据按照月份排序打上序号
获取月份数据
drop table if exists tmp_order_item_entrepot_month;
create table tmp_order_item_entrepot_month as
select itemid,entrepotid,itemnum,itemprice,date,substr(date,0,6) as month from
lg_tmp.tmp_order_item_entrepot_day ;
对月份进行排序
create table lg_tmp.tmp_order_item_rank
as
select
itemid,
entrepotid,
itemnum,
itemprice,
date,
dense_rank() over(order by month ) rank
from lg_tmp.tmp_order_item_entrepot_month ;
#导出为csv文件
hive -e "set hive.cli.print.header=true; select  date,rank as block_num,
entrepotid as entrepot_id,itemid as item_id,itemprice as item_price, itemnum as
item_cnt_day from lg_tmp.tmp_order_item_rank ;" |grep -v "WARN" | sed 's/\t/,/g'
>> /sales.csv
```

第 二 节 导出商品数据

观察商品数据特征，有商品名称，仓库id,商品分类id

基于lg_tmp.tmp_order_item_entrepot获取

```
#lg_tmp.tmp_order_item_entrepot与商品表进行关联（获取全部数据不用考虑拉链表实现）
```

```
drop table if exists lg_tmp.tmp_items;
create table lg_tmp.tmp_items
as
select
t1.itemname as itemname,
t1.itemid as itemid,
t2.goodscatid as catid
from
lg_tmp.tmp_order_item_entrepot t1
join
lg_dim.lg_dim_items t2
on t1.itemid =t2.goodsid ;
#导出为csv文件
hive -e "set hive.cli.print.header=true; select  * from lg_tmp.tmp_items ;"
|grep -v "WARN" | sed 's/\t/,/g' >> /items.csv
```

第 三 节 导出仓库及商品分类数据

直接从相关表中导出即可。

```
#导出仓库数据
hive -e "set hive.cli.print.header=true; select  entrepotname,entrepotid from
lg_ods.lg_entrepots ;" |grep -v "WARN" | sed 's/\t/,/g' >> /entrepots.csv
```

```
#导出商品分类数据
hive -e "set hive.cli.print.header=true; select
t2.catname as type,
t1.catName as subtype,
t1.catId as item_category_id
from (select catId,parentId,catName from lg_ods.lg_item_cats where cat_level = 3
) t1 JOIN
(select catId,catName from lg_ods.lg_item_cats where cat_level = 2 ) t2
on t1.parentId =t2.catId ;" |grep -v "WARN" | sed 's/\t/,/g' >>
/item_categories.csv
```

8 仓储预测

调用算法部门开发完成的程序进行预测，关于机器学习部分的内容会在后续课程中介绍。

参考predict_sales工程

9 车货匹配

参考算法讲义

第一节 需求介绍

在物流系统中，为了保证提供给客户及时高效的送货服务，我们需要在就近仓库提前储备相关商品。在面对二级仓库的补货需求时一级仓库需要基于货物数量合理安排车辆送货。

运输车辆规格信息：

```
货车（厢式/板车）车型：xxx   实际载重量：25吨/60立方米
```

备注：实际装车过程中，由于大部分商品都是日常百货用品，所以往往会先达到容积的限制但是载重可能还有很大余量。所以在解决实际问题时我们只需考虑如何尽可能满足空间。

一级仓库与二级仓库之间的货物调配每天都会进行，二级仓库每天会向一级仓库发送很多补货订单，

一个二级仓库的补货订单信息示例如下：

```
 ID     pack_total_volume
10001   1.18041
10002   8.92088
10003   4.58139
10004   5.29266
10005   1.23892
10006   3.93005
10007   7.53638
10008   6.92084
10009   3.89288
10010   4.67208
10011   0.37816
10012   0.7633
10013   1.49049
10014   0.2945
10015   3.63577
10016   4.53125
10017   6.86774
10018   7.03136
10019   8.32038
10020   3.28645
10021   0.92165
10022   6.0227
10023   6.67724
10024   2.25087
10025   6.36347
```

备注：一级仓库在每天五点之后会收集汇总每个二级仓库的补货信息，然后相同品类商品打包汇总之后生成如上运单信息，然后调配车辆进行运输。

对于这个场景中我们需要合理组合货物，保证尽量充分利用每辆车的空间来减少发车数量，降低企业的成本支出！！

总结：基于上面的分析，发现该问题是一个求最优解的问题。所以选择使用动态规划算法实现！！

第二节 思路分析

2.1 定义状态

数据：货物订单数据

```
volumeArr=
[1.18041,8.92088,4.58139,5.29266,1.23892,3.93005,7.53638,6.92084,3.89288,4.67208
,0.37816,0.7633..]
```

求解的是最少空间

```
dp[i][j]
```

含义：前i件物品在不超过最大容积j的时的最大体积

2.2 确定状态转移方程

参照01背包问题

dp(i,j)=dp(i-1,j-1)，确定这两者之间的关系

如果只剩最后一件物品时，有两种情况

- 不选择该物品：dp(i,j)=dp(i-1,j)
- 选择该物品：dp(i,j)=dp(i-1,j-volumeArr[i-1])+volumeArr[i-1]

dp(i,j)返回的是最大体积

max(dp(i-1,j),dp(i-1,j-volumeArr[i-1])+volumeArr[i-1])

第三节 代码开发

3.1 代码实现

参考01背包问题代码实现

```java
package com.lg.transport_goods;

public class TransportGoods {
    /*
    使用动态规划求解货物组合的最大占用空间(每辆车)
     */

    //定义一个方法求解

    /**
     * 求取一辆车最大使用空间
     * @param volumeArr:货物的体积数组
     * @param capacity：车辆的容积
     * @return
     */
    public static int getMaxVolume(int[] volumeArr,int capacity){

        //定义状态数组
        int[][] dp=new int[volumeArr.length+1][capacity+1];
        //遍历数据
        for (int i = 1; i <= volumeArr.length; i++) {
            for (int j = 1; j <= capacity; j++) {
                //翻译状态转移方程
                if(j <volumeArr[i-1]){
                    dp[i][j]=dp[i-1][j];
                }else{
                    dp[i][j] = Math.max(dp[i - 1][j - volumeArr[i - 1]] +
volumeArr[i - 1], dp[i - 1][j]);
                }

            }
        }

        return dp[volumeArr.length][capacity];
    }

    public static void main(String[] args) {
        //准备体积数据
        int[] volumeArr={1,2,3,4,5,6};
        //容积数据
        int capacity=10;

        final int maxVolume = getMaxVolume(volumeArr, 10);
        System.out.println(maxVolume);
    }
}
```

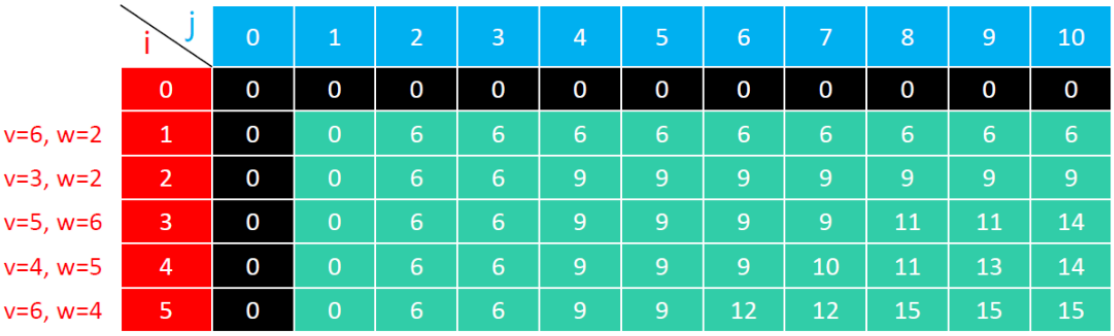
细节补充：

1、打印物品组合方案

实现01背包问题中物品组合方案的打印

```
package com.lagou.pack.teach;

import java.net.DatagramPacket;

public class PackSack {
    public static void main(String[] args) {
        int[] values = {6, 3, 5, 4, 6};
        int[] weights = {2, 2, 6, 5, 4};
        int capacity = 10;
        System.out.println(getMaxVal(values, weights, capacity));
    }

    static int getMaxVal(int[] values, int[] weights, int capacity) {
        //过滤掉不合理的值
        if (values == null || values.length == 0) {
            return 0;
        }
        if (weights == null || weights.length == 0) {
            return 0;
        }
        if (capacity < 0) {
            return 0;
        }
        //使用递推方式:dp(i,j):最大承重为j,有前i件物品可选时的最大总价值
        int[][] dp = new int[values.length + 1][capacity + 1]; //数组初始化默认值就
是0
        //遍历
        for (int i = 1; i <= values.length; i++) {
            for (int j = 1; j <= capacity; j++) {
                //翻译状态转移方程
                if (j < weights[i - 1]) {
                    dp[i][j] = dp[i - 1][j];
                } else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i - 1][j - weights[i -
1]] + values[i - 1]);
                }
            }
        }

        return dp[values.length][capacity];
    }


}
```

递推填表



	i j	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
v=6, w=2	1	0	0	6	6	6	6	6	6	6	6	6
v=3, w=2	2	0	0	6	6	9	9	9	9	9	9	9
v=5, w=6	3	0	0	6	6	9	9	9	9	11	11	14
v=4, w=5	4	0	0	6	6	9	9	9	10	11	13	14
v=6, w=4	5	0	0	6	6	9	9	12	12	15	15	15

打印货运组合方案信息

注意：数组索引越界异常

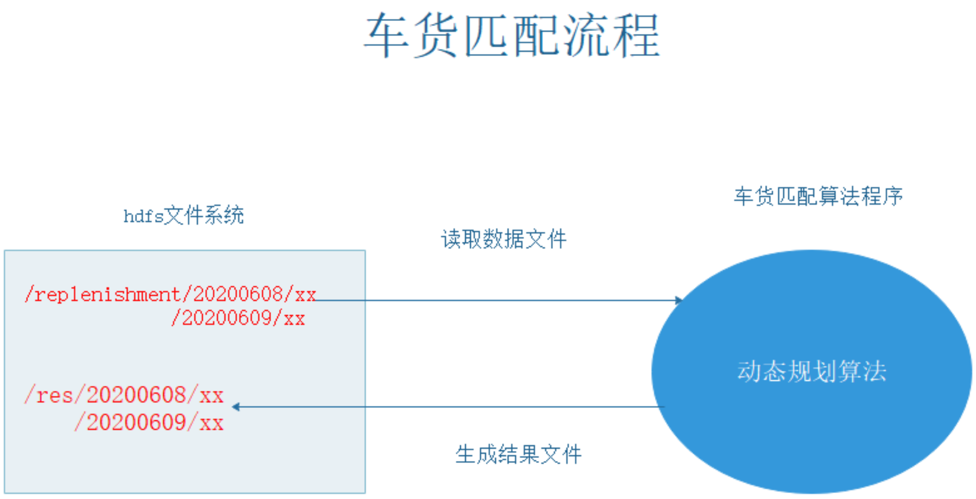
```
package com.lg.packsack;

public class PackSack {

    public static void main(String[] args) {
        int[] values = {6, 3, 5, 4, 6};
        int[] weights = {11,2, 2, 6, 5, 4};
        int capacity = 10;
        System.out.println(getMaxVal(values, weights, capacity));
    }

    static int getMaxVal(int[] values, int[] weights, int capacity) {
        //过滤掉不合理的值
        if (values == null || values.length == 0) {
            return 0;
        }
        if (weights == null || weights.length == 0) {
            return 0;
        }
        if (capacity < 0) {
            return 0;
        }
        //使用递推方式:dp(i,j):最大承重为j,有前i件物品可选时的最大总价值
        int[][] dp = new int[values.length + 1][capacity + 1]; //数组初始化默认值就
是0
        //遍历
        for (int i = 1; i <= values.length; i++) {
            for (int j = 1; j <= capacity; j++) {
                //翻译状态转移方程
                if (j < weights[i - 1]) {
                    dp[i][j] = dp[i - 1][j];
                } else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i - 1][j - weights[i -
1]] + values[i - 1]);
                }
            }
        }
        //dp[i][j]=dp[i-1][j-weights[i-1]]+values[i-1]  //说明选择了这件物品
        printGoodsInfo(dp, weights, values, capacity);
        System.out.println();
        System.out.println("------------------");
        return dp[values.length][capacity];
    }


    //打印出来商品组合信息
    public static void printGoodsInfo(int[][] dp, int[] weights, int[] values,
int capacity) {
        //从后往前遍历二维数组
        int i = weights.length;
        int j = capacity;
        //dp[i][j]=
        //循环操作
        while (i > 0 && j > 0) {
          try{
              if (dp[i][j] == dp[i - 1][j - weights[i - 1]] + values[i - 1]) {
                  //说明了weights[i-1]这件物品被选择了
                  System.out.print("选择了价值为" + values[i - 1] + "物品; ");
                  //j的值改变
                  j = j - weights[i - 1];
              }
          }catch (ArrayIndexOutOfBoundsException e){

          }
            //不管是否选择了物品，i都要移动到上一行
            i--;
        }

    }
}
```

循环遍历多辆车并且输出每辆车的物品组合

```
package com.lg.transport_goods;

import org.apache.commons.math3.analysis.function.Tan;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;

//参考01背包实现
public class TransportGoods2 {
    //货物体积数据 单位是立方米 转为立方分米
    static Double[] volumeArr = {
            1.18041, 8.92088, 4.58139, 5.29266, 1.23892, 3.93005, 7.53638,
6.92084, 3.89288, 4.67208, 0.37816, 0.7633, 1.49049, 0.2945, 3.63577, 4.53125,
6.86774, 7.03136,
            8.32038, 3.28645, 0.92165, 6.0227, 6.67724, 2.25087, 6.36347,
0.91234, 0.4131, 6.03253, 2.10624, 0.94981, 5.49145, 1.84597, 1.64853, 4.24115,
2.70332, 3.73644,
            5.50686, 4.85712, 5.72609, 3.07901, 5.43169, 7.88526, 2.37452,
2.87198, 6.30832, 3.03864, 3.19001, 3.50305, 0.28338, 9.95386, 3.94953, 6.72467,
3.82848, 3.03825,
            6.27153, 9.97295, 6.87071, 8.19529, 8.23215, 0.75555, 3.89223,
2.5766, 1.29116, 3.21259, 1.45661, 7.13919, 9.66758, 5.29364, 0.59208, 1.06844,
6.91517, 6.27877,
            1.50904, 6.50522, 1.77932, 6.30585, 1.19917, 4.87037, 1.07854,
7.28688, 3.50365, 2.16646, 0.5104, 1.01293, 4.91249, 6.23181, 2.64389, 3.06228,
5.61209, 3.58101
    };

    static Integer capacity = 40 * 1000;

    //如何判断货物是否装载完毕
    public static void getMaxVolumeMulti(Integer[] volumeArr, Integer capacity)
{
        int num = 0;
        //判断货物体积数组中是否有数据
        while (volumeArr.length != 0) {
            num++;
            Integer[] arr = getMaxVolume(volumeArr, capacity, num);
            volumeArr = arr;
        }
    }

    //对货物体积数据进行整数化处理
    public static Integer[] translate(Double[] volumeArr) {
        //准备一个整数数组
        Integer[] arr = new Integer[volumeArr.length];
        //保留小数点后三位的数据
        DecimalFormat decimalFormat = new DecimalFormat("######0.000");
        //对数据进行转换
        for (int i = 0; i < volumeArr.length; i++) {
            String text = decimalFormat.format(volumeArr[i]);
            double v = Double.parseDouble(text);
            arr[i] = (int) (v * 1000);
        }
        return arr;
    }

    /**
     * @param volumeArr
     * @param capacity
     * @return
     */
    public static Integer[] getMaxVolume(Integer[] volumeArr, int capacity, int
num) {
        //定义状态数组
        int[][] dp = new int[volumeArr.length + 1][capacity + 1];
        //遍历数据
        for (int i = 1; i <= volumeArr.length; i++) {
            for (int j = 1; j <= capacity; j++) {
                //状态方程
                if (j < volumeArr[i - 1]) {
                    //剩余容积不满足选择该物品
                    dp[i][j] = dp[i - 1][j];
                } else {
                    //选或者不选
//                    dp[i-1][j],
                    dp[i][j] = Math.max(dp[i - 1][j - volumeArr[i - 1]] +
volumeArr[i - 1], dp[i - 1][j]);
                }
            }
        }

        System.out.println();
        System.out.println("----------------------");
    //打印最大体积值
        System.out.println("第" + num + "辆车，最大组合体积为" +
dp[volumeArr.length][capacity]);
        //调用详细货物组合方案信息
        Integer[] newArr = printGoodsInfo(dp, volumeArr, capacity);
        return newArr;
    }


    //打印出来货物组合信息
    public static Integer[] printGoodsInfo(int[][] dp, Integer[] volumeArr, int
capacity) {
        //从后往前遍历二维数组
        int i = volumeArr.length;
        int j = capacity;
        //dp[i][j]=
        //循环操作
        while (i > 0 && j > 0) {
            try {
                if (dp[i][j] == dp[i - 1][j - volumeArr[i - 1]] + volumeArr[i -
1]) {
                    //说明了volumeArr[i - 1]这件物品被选择了
                    System.out.print("选择了体积为" + volumeArr[i - 1] + "货物; ");
                    //j的值改变
                    j = j - volumeArr[i - 1];

                    //标记被选择的物品
                    volumeArr[i - 1] = 0;
                }
            } catch (ArrayIndexOutOfBoundsException e) {

            }
            //不管是否选择了物品，i都要移动到上一行
            i--;
        }
//物品被选择之后，需要从原始数组中移除,所有数据为0的数据过滤掉
```

```
        return getNewArr(volumeArr);

    }

    //剔除选择的物品
    public static Integer[] getNewArr(Integer[] volumeArr) {
        //准备接收的list
        ArrayList<Integer> list = new ArrayList<>();
        for (int i = 0; i < volumeArr.length; i++) {
            if (volumeArr[i] != 0) {
                list.add(volumeArr[i]);
            }
        }
        //转成数组
        Integer[] arr = new Integer[list.size()];
        return list.toArray(arr);
    }

    public static void main(String[] args) {
        getMaxVolumeMulti(translate(volumeArr), capacity);
    }
}
```

3.3 车货匹配流程



CDH集群依赖

```
由于cdh版本的所有的软件涉及版权的问题，所以并没有将所有的jar包托管到maven仓库当中去，而是托管
在了CDH自己的服务器上面，所以我们默认去maven的仓库下载不到，需要自己手动的添加repository去
CDH仓库进行下载，以下两个地址是官方文档说明，请仔细查阅
https://www.cloudera.com/documentation/enterprise/release-
notes/topics/cdh_vd_cdh5_maven_repo.html
https://www.cloudera.com/documentation/enterprise/release-
notes/topics/cdh_vd_cdh5_maven_repo_514x.html
```

Apche版本

```
 <!-- hadoop-common ,hadoop-client,hadoop-hdfs-->
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.8.2</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common
-->
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>2.9.2</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-client
-->
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>2.9.2</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs --
>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-hdfs</artifactId>
            <version>2.9.2</version>
        </dependency>


    </dependencies>
```

CDH版本

```
<repositories>
        <repository>
            <id>cloudera</id>
            <url>https://repository.cloudera.com/artifactory/cloudera-
repos/</url>
```

```
        </repository>
    </repositories>


    <dependencies>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>2.6.0-mr1-cdh5.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>2.6.0-cdh5.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-hdfs</artifactId>
            <version>2.6.0-cdh5.14.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-mapreduce-client-core</artifactId>
            <version>2.6.0-cdh5.14.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/junit/junit -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>RELEASE</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                    <!--    <verbal>true</verbal>-->
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>2.4.3</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <minimizeJar>true</minimizeJar>
                        </configuration>
                    </execution>
                </executions>
            </plugin>

        </plugins>
    </build>
```

log4j.properties

```
log4j.rootCategory=debug, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n
```

完整代码

```
package com.lg.transport_goods;

import com.lg.util.DownloadFile;
import com.lg.util.LgFileUtil;
import com.lg.util.UploadFile;

import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

//参考01背包实现
public class TransportGoods3 {


    //如何判断货物是否装载完毕
    public static void getMaxVolumeMulti(Integer[] volumeArr, Integer capacity,String fileName) {
        int num = 0;
        //判断货物体积数组中是否有数据
        while (volumeArr.length != 0) {
            num++;
            Integer[] arr = getMaxVolume(volumeArr, capacity, num,fileName);
            volumeArr = arr;
        }
    }

    //对货物体积数据进行整数化处理
    public static Integer[] translate(Double[] volumeArr) {
        //准备一个整数数组
        Integer[] arr = new Integer[volumeArr.length];
        //保留小数点后三位的数据
        DecimalFormat decimalFormat = new DecimalFormat("######0.000");
        //对数据进行转换
        for (int i = 0; i < volumeArr.length; i++) {
            String text = decimalFormat.format(volumeArr[i]);
            double v = Double.parseDouble(text);
            arr[i] = (int) (v * 1000);
        }
        return arr;
    }

    /**
     * @param volumeArr
     * @param capacity
     * @return
     */
    public static Integer[] getMaxVolume(Integer[] volumeArr, int capacity, int
num,String fileName) {
        //定义状态数组
        int[][] dp = new int[volumeArr.length + 1][capacity + 1];
        //遍历数据
        for (int i = 1; i <= volumeArr.length; i++) {
            for (int j = 1; j <= capacity; j++) {
                //状态方程
                if (j < volumeArr[i - 1]) {
                    //剩余容积不满足选择该物品
                    dp[i][j] = dp[i - 1][j];
                } else {
                    //选或者不选
//                      dp[i-1][j],
                    dp[i][j] = Math.max(dp[i - 1][j - volumeArr[i - 1]] +
volumeArr[i - 1], dp[i - 1][j]);
                }
            }
        }

        System.out.println();
        System.out.println("----------------------");
        //打印最大体积值
        System.out.println("第" + num + "辆车,最大组合体积为" +
dp[volumeArr.length][capacity]);
        //输出结果到目标文件
        try {
            LgFileUtil.writeString2SimpleFile("第" + num + "辆车,最大组合体积为" +
dp[volumeArr.length][capacity], fileName, "utf-8");
        } catch (Exception e) {
            e.printStackTrace();
        }
        //调用详细货物组合方案信息
        Integer[] newArr = printGoodsInfo(dp, volumeArr, capacity,fileName);
        return newArr;
    }
    //打印出来货物组合信息
    public static Integer[] printGoodsInfo(int[][] dp, Integer[] volumeArr, int
capacity ,String fileName) {
        //从后往前遍历二维数组
        int i = volumeArr.length;
        int j = capacity;
        //dp[i][j]=
        //循环操作
        while (i > 0 && j > 0) {
            try {
                if (dp[i][j] == dp[i - 1][j - volumeArr[i - 1]] + volumeArr[i -
1]) {
                    //说明了volumeArr[i - 1]这件物品被选择了
                    System.out.print("选择了体积为" + volumeArr[i - 1] + "货物; ");
                    LgFileUtil.writeString2SimpleFile(volumeArr[i - 1]+"",
fileName, "utf-8");
                    //j的值改变
                    j = j - volumeArr[i - 1];

                    //标记被选择的物品
                    volumeArr[i - 1] = 0;
                }
            } catch (ArrayIndexOutOfBoundsException e) {

            } catch (Exception e) {
                e.printStackTrace();
            }
            //不管是否选择了物品，i都要移动到上一行
            i--;
        }
//物品被选择之后，需要从原始数组中移除,所有数据为0的数据过滤掉
        return getNewArr(volumeArr);

    }

    //剔除选择的物品
    public static Integer[] getNewArr(Integer[] volumeArr) {
        //准备接收的list
        ArrayList<Integer> list = new ArrayList<>();
        for (int i = 0; i < volumeArr.length; i++) {
            if (volumeArr[i] != 0) {
                list.add(volumeArr[i]);
            }
        }
        //转成数组
        Integer[] arr = new Integer[list.size()];
        return list.toArray(arr);
    }

    public static void main(String[] args) {
        //hdfs和本地相关目录信息
        String day = "20200608";
        String hdfsDir = "/replenishment";
        String hdfsResDir = "/res";

        String localDir = "e:/replenishment";
        String localResDir = "e:/res";
        Integer capacity = 40 * 1000;
        //下载hdfs的数据目录
        try {
            DownloadFile.init("hdfs://lgns");
            DownloadFile.download(hdfsDir + "/" + day, localDir + "/" + day);
            DownloadFile.destory();
        } catch (Exception e) {
            e.printStackTrace();
        }
        //读取下载好的本地的数据文件，解析数据得到物品体积数组，然后调用动态规划算法
        HashMap<String, Double[]> map=null;
        try {
            map = LgFileUtil.readFiles(localDir + "/" + day);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if(map==null){return;}
        //遍历map获取到每一个体积数组
        for (Map.Entry<String, Double[]> entry : map.entrySet()) {
            String key = entry.getKey();//仓库编号
            Double[] volumeArr = entry.getValue();//体积数组
            getMaxVolumeMulti(translate(volumeArr),
capacity,localResDir+"/"+day+"/"+key);
        }

        //上传结果文件到hdfs
        try {
            UploadFile.init("hdfs://lgns");
            UploadFile.uploadFolder(localResDir+"/"+day, hdfsResDir+"/"+day);
            UploadFile.destory();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        //保存调度信息到mysql(车辆调度信息)以及hbase（装载货物信息）
        try {
            LgFileUtil.readScheduleFiles(localResDir+"/"+day);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

保存调度数据

- 车辆调度信息存储Mysql
- 车辆货物信息存储Hbase

Mysql存储

lg_bus表

```
CREATE TABLE `lg_bus` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `plateNum` varchar(256) NOT NULL COMMENT '车牌号',
  `travle_distance` bigint(20) NOT NULL COMMENT '行驶里程',
  `status` tinyint(5) NOT NULL COMMENT '状态：0-->等待调度；1-->正在调度;',
  `sim` varchar(50) NOT NULL COMMENT 'sim卡号',
  `transportNum` varchar(50) NOT NULL COMMENT '道路运输证',
  `oilRemain` int(11) NOT NULL COMMENT '剩余油量',
  `weights` int(11) NOT NULL COMMENT '最大载重',
  `volume` int(11) NOT NULL COMMENT '最大容积',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8;


##插入数据
INSERT INTO `lg_bus` VALUES ('1', '京A-11111', '1000', '0', '1111', 'ysz11111', '50', '25', '40');
INSERT INTO `lg_bus` VALUES ('2', '京A-22222', '2000', '0', '2222', 'ysz22222', '60', '25', '40');
INSERT INTO `lg_bus` VALUES ('3', '京A-33333', '3000', '0', '3333', 'ysz333333', '70', '25', '40');
INSERT INTO `lg_bus` VALUES ('4', '京A-44444', '4000', '0', '4444', 'ysz44444', '80', '25', '40');
```

log_schedule

```
CREATE TABLE `log_schedule` (
  `id` varchar(256) NOT NULL COMMENT '主键',
  `travel_id` varchar(256) DEFAULT NULL COMMENT '调度编号',
  `bus_number` varchar(64) NOT NULL COMMENT '车牌号',
  `src_location` varchar(256) DEFAULT NULL COMMENT '行程开始时间',
  `dest_location` varchar(256) NOT NULL COMMENT '目的地',
  `creat_time` timestamp NULL DEFAULT NULL COMMENT '调度任务创建时间',
  `schedule_time` timestamp NULL DEFAULT NULL COMMENT '开始调度时间',
  `finish_time` timestamp NULL DEFAULT NULL COMMENT '调度完成时间',
  `status` tinyint(12) NOT NULL DEFAULT '0' COMMENT '0-等待调度，1-调度中，2-已完成',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Hbase存储

```
create 'lg_trans_info','goods'
```

代码

```
package com.lg.util;

import com.lg.bean.Bus;

import java.io.*;
import java.sql.Connection;
import java.sql.SQLException;
import java.text.NumberFormat;
import java.text.SimpleDateFormat;
import java.util.*;
```

```java
public class LgFileUtil {

    // 缓冲区大小
    private final static int buffer_size = 1024;

    // 日志格式工具
    private final static SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

    // 小数的格式化工具,设置最大小数位为10
    private final static NumberFormat numFormatter =
NumberFormat.getNumberInstance();

    static {
        numFormatter.setMaximumFractionDigits(10);
    }

    // 换行符

    private final static String lineSeparator = java.security.AccessController
            .doPrivileged(new
sun.security.action.GetPropertyAction("line.separator"));


    /**
     * 以指定编码格式写入字符串到CSV文件
     *
     * @param csvStr      要输出的字符串(有CSV格式)
     * @param filePath    CSV文件的具体路径
     * @param charsetName GB2312
     * @throws Exception
     */
    public static void writeCsvStr2CsvFile(String csvStr, String filePath,
String charsetName) throws Exception {
        LgFileUtil.writeString2SimpleFile(csvStr, filePath, charsetName);
    }

    /**
     * 以指定编码格式写入字符串到简单文件
     *
     * @param str         要输出的字符串
     * @param filePath    简单文件的具体路径
     * @param charsetName UTF-8 | GB2312
     * @throws Exception
     */
    public static void writeString2SimpleFile(String str, String filePath,
String charsetName) throws Exception {

        BufferedWriter out = null;
        try {
            File file = new File(filePath);

            createNewFileIfNotExists(file);

            OutputStreamWriter os = new OutputStreamWriter(new
FileOutputStream(file, true), charsetName);
```

```
            out = new BufferedWriter(os, LgFileUtil.buffer_size);

            out.write(str);
            out.newLine();
            out.flush();
        } finally {
            LgFileUtil.close(out);
        }

    }


    /**
     * 如果文件不存在,创建一个新文件
     */
    private static void createNewFileIfNotExists(File file) throws IOException {
        if (!file.exists()) {
            // 创建目录
            if (!file.getParentFile().exists()) {
                file.getParentFile().mkdirs();
            }

            // 创建文件
            file.createNewFile();
        }
    }

    /**
     * 关闭输出流
     */
    private static void close(Writer out) {
        if (null != out) {
            try {
                out.close();
            } catch (IOException e) {
                // e.printStackTrace();
            }
        }
    }

    //读取指定文件夹下的所有文件并把解析的体积数据存储到数组中
    public static HashMap<String, Double[]> readFiles(String strPath) throws
Exception {
        File dir = new File(strPath);
        File[] files = dir.listFiles();
        if (files == null)
            return null;
        //准备一个存放了所有文件内容的集合
        final HashMap<String, Double[]> map = new HashMap<>();
        for (int i = 0; i < files.length; i++) {
            if (files[i].isFile()) {
                String strFileName = files[i].getAbsolutePath().toLowerCase();
                final int i1 = strFileName.lastIndexOf('\\');
                String fileName = strFileName.substring(i1 + 1);
                System.out.println("--文件名--" + fileName);
                //存放体积数据的集合

                ArrayList<Double> list = new ArrayList<Double>();
                final BufferedReader br = new BufferedReader(new FileReader(new
File(strFileName)));
                String line;
                int r = 0;
                while ((line = br.readLine()) != null) {
                    if (r == 0) {
                        r++;
                        continue;
                    }

                    //解析文件，按照制表符切分
                    final String[] arr = line.split("\t");
                    list.add(Double.parseDouble(arr[1]));

                }
                br.close();
                Double[] arr = new Double[list.size()];
                list.toArray(arr);
                map.put(fileName, arr);

            }
        }
        return map;
    }


    //遍历调度文件夹结果数据保存到mysql以及hbase
    public static void readScheduleFiles(String strPath) throws Exception {
        File dir = new File(strPath);
        File[] files = dir.listFiles();
        if (files == null)
            return;
        for (int i = 0; i < files.length; i++) {
            if (files[i].isFile()) {
                final String fileName = files[i].getName();
                String entrepotNum = fileName.substring(0,
fileName.lastIndexOf("."));
                System.out.println("仓库编号：" + entrepotNum);
                final BufferedReader br = new BufferedReader(new
FileReader(files[i]));
                String line;
                int num = 0;
                StringBuffer sb = new StringBuffer();
                while ((line = br.readLine()) != null) {
                    final String[] arr = line.split(":");
                    if (arr.length < 2) {
                        if (num != 0) {
                            //说明是说明语句，不是货物信息，但是表明了需要一辆车来运送货
物,上一辆车的货物信息
                            saveToMysqlAndHbase(entrepotNum, sb.toString());
                            sb = new StringBuffer();
                        }
                        num++; //num代表的就是第几辆车
                    } else {
                        //说明是货物信息，拼接一辆车的货物信息称为一个字符串
                        sb.append(line).append("&");
                    }
                }
                //读完整个文件，把最后一辆车信息输出
                saveToMysqlAndHbase(entrepotNum, sb.toString());
                br.close();
            }
        }

    }

    /**
     * @param entrepotNum:仓库编号
     * @param str:一辆车的货物信息
     */
    public static void saveToMysqlAndHbase(String entrepotNum, String str)
throws SQLException, IOException, ClassNotFoundException {
        //1 获取一辆车
        final Connection connection = DBUtil.openConnection();
        String sql = "select * from lg_bus where status = 0 order by
travle_distance,id limit 1;";
        Bus bus = null;
        try {
            bus = DBUtil.queryBean(connection, sql, Bus.class);
        } catch (Exception e) {
            System.out.println("车辆不足。。。");
            throw new RuntimeException();
        }
        //2 对它进行调度，保存到mysql指定表中
        final String id = UUID.randomUUID().toString();
        String[] deployArr = {"316d5c75-e860-4cc9-a7de-ea2148c244a0",
                "32102c12-6a73-4e03-80ab-96175a8ee686",
                "a97f6c0d-9086-4c68-9d24-8a7e89f39e5a",
                "adfgfdewr-5463243546-4c68-9d24-8a7e8"};
        final Random rd = new Random();
        String travle_id = deployArr[rd.nextInt(deployArr.length)];
        String sqlStr1 = "insert into log_schedule
(id,travel_id,bus_number,src_location,dest_location,creat_time,status)" +
                "values(?,?,?,?,?,?,?);";
        DBUtil.execute(connection, sqlStr1,
                id,
                travle_id,
                bus.getPlateNum(),
                "一级仓库",
                entrepotNum+"号二级仓库",
                new Date(),
                1);
        //更新lg_bus中的状态
        String sqlStr2 = "update lg_bus set status =1 where plateNum =?";
        DBUtil.execute(connection, sqlStr2, bus.getPlateNum());

        //把货物信息存储到hbase中
        String rowkey = travle_id + bus.getPlateNum();
        DBUtil.closeConnection(connection);
        HbaseUtil.putRow("lg_trans_info", rowkey, "info", "goods", str);
    }


    public static void main(String[] args) throws Exception {
        readScheduleFiles("E:\\res\\20200608");
    }
```

```
}
```

10 指标统计

第一节 指标体系

- 快递单主题
    - 快递单量的统计主要是从多个不同的维度计算快递单量，从而监测公司快递业务运营情况。

指标列表	维度
最大快递单数	各类客户最大快递单数
	各渠道最大快递单数
	各网点最大快递单数
	各终端最大快递单数
最小快递单数	各类客户最小快递单数
	各渠道最小快递单数
	各网点最小快递单数
	各终端最小快递单数

- 运单主题
    - 运单统计根据区域、公司、网点、线路、运输工具等维度进行统计，可以对各个维度运单数量进行排行，如对网点运单进行统计可以反映该网点的运营情况

指标列表	维度
最大运单数	最大区域运单数
	各分公司最大运单数
	各网点最大运单数
	各线路最大运单数
	各运输工具最大运单数
	各类客户最大运单数
最小运单数	各区域最小运单数
	各分公司最小运单数
	各网点最小运单数
	各线路最小运单数
	各运输工具最小运单数
	各类客户最小运单数

第二节 业务数据

1.1 物流系统数据库表

揽件表（lg_collect_package）

字段名	字段描述
id	ID
empId	快递员ID
express_bill__id	快递单ID
package_id	包裹ID
state	揽件状态
collect_package_dt	揽件时间
cdt	创建时间
udt	修改时间
remark	备注

客户表（lg_customer）

字段名	字段描述
id	ID
customer_nnumber	客户编号
customer_name	客户姓名
customer_tel	客户电话
customer_mobile	客户手机
customer_email	客户邮箱
customer_type	客户类型ID
is_own_reg	是否自行注册
customer_reg_dt	注册时间
customer_reg_channel_id	注册渠道ID
customer_state_id	客户状态ID
cdt	创建时间
udt	修改时间
last_login_dt	最后登录时间
remark	备注

物流系统码表（lg_codes）

字段名	字段描述
id	ID
business_implications	业务含义
business_code	业务码
business_val	业务码值
business_val_type	业务码值类型
cdt	创建时间
udt	修改时间
remark	备注

快递单据表（lg_express_bill）

字段名	字段描述
id	ID
express_number	快递单号
customer_id	客户ID
employee_id	员工ID
order_channel_id	下单渠道ID
order_dt	下单时间
order_terminal_type	下单设备类型ID
order_terminal_os_type	下单设备操作系统ID
reserve_dt	预约取件时间
is_collect_package_timeout	是否取件超时
timeout_dt	超时时间
express_bill_type	快递单生成方式
cdt	创建时间
udt	修改时间
remark	备注

客户地址表（lg_customer_address）

字段名	字段描述
id	ID
name	用户名
tel	电话
mobile	手机
detail_addr	详细地址
area_id	区域ID
gis_addr	gis地址
addr_type	地址类型
cdt	创建时间
udt	修改时间
remark	备注

网点表（lg_dot）

字段名	字段描述
id	ID
dot_number	网点编号
dot_name	网点名称
dot_addr	网点地址
dot_gis_addr	网点GIS地址
dot_tel	网点电话
company_id	网点所属公司ID
manage_area_id	网点管理辖区ID
manage_area_gis	网点管理辖区地理围栏
state	网点状态
cdt	创建时间
udt	修改时间
remark	备注

公司表（lg_company）

字段名	字段描述
id	ID
company_name	公司名称
city_id	城市ID
company_number	公司编号
company_addr	公司地址
company_addr_gis	公司gis地址
company_tel	公司电话
is_sub_company	母公司ID
state	公司状态
cdt	创建时间
udt	修改时间
remark	备注

公司网点关联表（lg_company_dot_map）

字段名	字段描述
id	ID
company_id	公司ID
dot_id	网点ID
cdt	创建时间
udt	修改时间
remark	备注

运单表（lg_waybill）

字段名	字段描述
id	ID
waybill_number	运单号
customer_id	寄件信息ID
employee_id	收件员工ID
order_channel_id	下单渠道ID
order_dt	下单时间
order_terminal_type	下单设备类型ID
order_terminal_os_type	下单设备操作系统ID
reserve_dt	预约取件时间
is_collect_package_timeout	是否取件超时
pkg_id	订装ID
pkg_number	订装编号
timeout_dt	超时时间
transform_type	运输方式
delivery_customer_name	发货人
delivery_addr	发货地址
delivery_mobile	发货人手机
delivery_tel	发货人电话
receive_customer_name	收货人
receive_addr	收货地址
receive_mobile	收货人手机
receive_tel	收货人电话
cdt	创建时间
udt	修改时间
remark	运单备注

线路表（lg_route）

字段名	字段描述
id	ID
start_station	运输线路起点
start_warehouse_id	起点仓库
stop_station	运输线路终点
stop_warehouse_id	终点仓库
transport_kilometre_km	运输里程
transport_dt_hour	运输时长
transport_route_state	线路状态
cdt	创建时间
udt	修改时间
remark	备注

运输工具表（lg_transport_tool）

字段名	字段描述
id	ID
brand	运输工具品牌
model	运输工具型号
type	运输工具类型
given_load	额定载重
load_cn_unit	中文载重单位
load_en_unit	英文载重单位
buy_dt	购买时间
license_plate	牌照
state	运输工具状态
cdt	创建时间
udt	修改时间
remark	备注

转运记录表（lg_transport_record）

字段名	字段描述
id	ID
pw_id	入库表ID
pw_waybill_id	入库运单ID
pw_waybill_number	入库运单号
ow_id	出库表ID
ow_waybill_id	出库运单ID
ow_waybill_number	出库运单号
sw_id	起点仓库ID
ew_id	终点仓库ID
driver1_id	车辆驾驶员
driver2_id	车辆跟车员1
driver3_id	车辆跟车员2
route_id	运输路线ID
distance	运输里程
duration	运输耗时
start_vehicle_dt	发车时间
predict_arrivals_dt	预计到达时间
actual_arrivals_dt	实际达到时间
cdt	创建时间
udt	修改时间
remark	备注

包裹表（lg_pkg）

字段名	字段描述
id	ID
pw_bill	订装箱号
pw_dot_id	操作人ID
warehouse_id	所属仓库ID
cdt	创建时间
udt	修改时间
remark	备注

运单路线明细表（lg_waybill_line）

字段名	字段描述
id	ID
waybill_number	运单号
route_id	路线ID
serial_number	序号
transport_tool	运输工具ID
delivery_record_id	发车记录单ID
cdt	创建时间
udt	修改时间
remark	备注

快递员表（lg_courier）

字段名	字段描述
id	ID
job_num	快递员工号
name	快递员姓名
birathday	快递员生日
tel	快递员联系电话
pda_num	PDA编号
car_id	车辆ID
postal_standard_id	收派标准ID
work_time_id	收派时间ID
dot_id	网点ID
state	快递员状态
cdt	创建时间
udt	修改时间
remark	备注

区域表（lg_areas）

字段名	字段描述
id	ID
name	区域名称
type	区域类型
pid	父区域ID
zipcode	区域邮编
order_no	排序字段
name_en	区域英文首字母
is_zxs	是否直辖市
cdt	创建时间
udt	修改时间
remark	备注

第三节 快递单主题

3.1 lg_ods层建表

建表语句

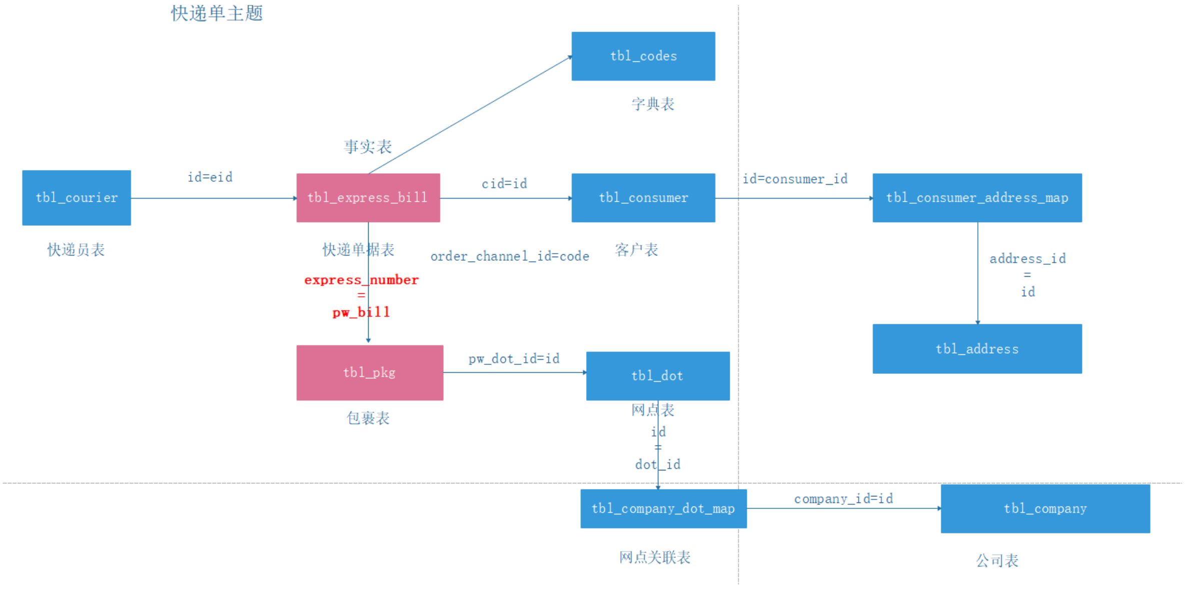
```
--客户地址表
DROP TABLE IF EXISTS `lg_ods.lg_address`;
CREATE TABLE `lg_ods.lg_address` (
  `id` string,
  `name` string,
  `tel` string,
  `mobile` string,
  `detail_addr` string,
  `area_id` string,
  `gis_addr` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '客户地址表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';
--客户表
DROP TABLE IF EXISTS `lg_ods.lg_customer`;
CREATE TABLE `lg_ods.lg_customer` (
  `id` string,
  `name` string,
  `tel` string,
  `mobile` string,
  `email` string,
  `type` string,
  `is_own_reg` string,
  `reg_dt` string,
  `reg_channel_id` string,
  `state` string,
  `cdt` string,
  `udt` string,
  `last_login_dt` string,
  `remark` string)
COMMENT '客户表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';

DROP TABLE IF EXISTS `lg_ods.lg_areas`;
CREATE TABLE `lg_ods.lg_areas` (
  `id` string,
  `name` string,
  `pid` string,
  `sname` string,
  `level` string,
  `citycode` string,
  `yzcode` string,
  `mername` string,
  `lng` string,
  `lat` string,
  `pinyin` string
) COMMENT '区域表'
```

```
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';


DROP TABLE IF EXISTS `lg_ods.lg_courier`;
CREATE TABLE `lg_ods.lg_courier` (
  `id` string,
  `job_num` string,
  `name` string,
  `birathday` string,
  `tel` string,
  `pda_num` string,
  `car_id` string,
  `postal_standard_id` string,
  `work_time_id` string,
  `dot_id` string,
  `state` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '快递员表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';



DROP TABLE IF EXISTS `lg_ods.lg_dot`;
CREATE TABLE `lg_ods.lg_dot` (
  `id` string,
  `dot_number` string,
  `dot_name` string,
  `dot_addr` string,
  `dot_gis_addr` string,
  `dot_tel` string,
  `company_id` string,
  `manage_area_id` string,
  `manage_area_gis` string,
  `state` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '网点表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';


DROP TABLE IF EXISTS `lg_ods.lg_company_dot_map`;
CREATE TABLE `lg_ods.lg_company_dot_map` (
  `id` string,
  `company_id` string,
  `dot_id` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '公司网点关联表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';


DROP TABLE IF EXISTS `lg_ods.lg_company`;
CREATE TABLE `lg_ods.lg_company` (
  `id` string,
  `company_name` string,
  `city_id` string,
  `company_number` string,
  `company_addr` string,
  `company_addr_gis` string,
  `company_tel` string,
  `is_sub_company` string,
  `state` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '公司表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';


DROP TABLE IF EXISTS `lg_ods.lg_customer_address_map`;
CREATE TABLE `lg_ods.lg_customer_address_map` (
  `id` string,
  `consumer_id` string,
  `address_id` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '客户地址关联表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';

DROP TABLE IF EXISTS `lg_ods.lg_codes`;
CREATE TABLE `lg_ods.lg_codes` (
  `id` string,
  `name` string,
  `type` string,
  `code` string,
  `code_desc` string,
  `code_type` string,
  `state` string,
  `cdt` string,
  `udt` string

) COMMENT '字典表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';

DROP TABLE IF EXISTS `lg_ods.lg_express_bill`;
CREATE TABLE `lg_ods.lg_express_bill` (
  `id` string,
  `express_number` string,
  `cid` string,
  `eid` string,
  `order_channel_id` string,
  `order_dt` string,
  `order_terminal_type` string,
  `order_terminal_os_type` string,
  `reserve_dt` string,
  `is_collect_package_timeout` string,
  `timeout_dt` string,
  `type` string,
  `cdt` string,
  `udt` string,
  `remark` string
```

```
) COMMENT '快递单据表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';

DROP TABLE IF EXISTS `lg_ods.lg_pkg`;
CREATE TABLE `lg_ods.lg_pkg` (
  `id` string,
  `pw_bill` string,
  `pw_dot_id` string,
  `warehouse_id` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '包裹表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';
```

3.2 数据采集

业务数据保存在MySQL中，每日凌晨导入上一天的表数据。



- 事实表
  - 全量导入

```
#!/bin/bash
source /etc/profile
##如果第一个参数不为空，则作为工作日期使用
if [ -n "$1" ]
then
do_date=$1
else
##昨天日期，减一
do_date=`date -d "-1 day" +"%Y%m%d"`
fi
#定义sqoop命令位置，Hive命令位置，在hadoop2
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期
```

```
#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
import_data(){
$sqoop import \
--connect jdbc:mysql://hadoop1:3306/lg_logistics \
--username root \
--password 123456 \
--target-dir /user/hive/warehouse/lg_ods.db/$1/dt=$do_date \
--delete-target-dir \
--query "$2 and  \$CONDITIONS" \
--num-mappers 1 \
--fields-terminated-by ',' \
--null-string '\\N' \
--null-non-string '\\N'
}


# 全量导入快递单据表数据
import_lg_express_bill(){
  import_data lg_express_bill "select
                            *
                        from lg_express_bill
                        where 1=1"
}

# 全量导入包裹表方法
import_lg_pkg(){
  import_data lg_pkg "select
                            *
                        from lg_pkg
                        where 1=1"
}



#调用全量导入数据方法
import_lg_express_bill
import_lg_pkg


#注意sqoop导入数据的方式，对于Hive分区表来说需要执行添加分区操作，数据才能被识别到
$Hive -e "alter table lg_ods.lg_express_bill add
partition(dt='$do_date');
alter table lg_ods.lg_pkg add partition(dt='$do_date');"
```

- 增量导入

```
#!/bin/bash
source /etc/profile
##如果第一个参数不为空，则作为工作日期使用
if [ -n "$1" ]
then
do_date=$1
else
##昨天日期，减一
do_date=`date -d "-1 day" +"%Y%m%d"`
fi
#定义sqoop命令位置，Hive命令位置，在hadoop2
```

```
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期


#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
import_data(){
$sqoop import \
--connect jdbc:mysql://hadoop1:3306/lg_logistics \
--username root \
--password 123456 \
--target-dir /user/hive/warehouse/lg_ods.db/$1/dt=$do_date \
--delete-target-dir \
--query "$2 and  \$CONDITIONS" \
--num-mappers 1 \
--fields-terminated-by ',' \
--null-string '\\N' \
--null-non-string '\\N'
}


# 增量导入快递单据表数据
import_lg_express_bill(){
  import_data lg_express_bill "select
                            *
                        from lg_express_bill
                        WHERE DATE_FORMAT(cdt, '%Y%m%d') = '${do_date}'
"
}

# 增量导入包裹表方法
import_lg_pkg(){
  import_data lg_pkg "select
                            *
                        from lg_pkg
                       WHERE DATE_FORMAT(cdt, '%Y%m%d') = '${do_date}'
"
}



#调用全量导入数据方法
import_lg_express_bill
import_lg_pkg


#注意sqoop导入数据的方式，对于Hive分区表来说需要执行添加分区操作，数据才能被识别到
$Hive -e "alter table lg_ods.lg_express_bill add
partition(dt='$do_date');
alter table lg_ods.lg_pkg add partition(dt='$do_date');"
```

- 维度表
  - 全量导入

```
#!/bin/bash
```

```
source /etc/profile
##如果第一个参数不为空，则作为工作日期使用
if [ -n "$1" ]
then
do_date=$1
else
##昨天日期，减一
do_date=`date -d "-1 day" +"%Y%m%d"`
fi
#定义sqoop命令位置，Hive命令位置，在hadoop2
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期


#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
import_data(){
$sqoop import \
--connect jdbc:mysql://hadoop1:3306/lg_logistics \
--username root \
--password 123456 \
--target-dir /user/hive/warehouse/lg_ods.db/$1/dt=$do_date \
--delete-target-dir \
--query "$2 and  \$CONDITIONS" \
--num-mappers 1 \
--fields-terminated-by ',' \
--null-string '\\N' \
--null-non-string '\\N'
}


# 全量导入客户地址表
import_lg_address(){
  import_data lg_address "select
                            *
                        from lg_address
                        where 1=1"
}

# 全量导入仓库方法
import_lg_entrepots(){
  import_data lg_entrepots "select
                            *
                        from lg_entrepots
                        where 1=1"
}

# 全量导入区域表
import_lg_areas(){
  import_data lg_areas "select
                            *
                        from lg_areas
                        where 1=1"
}

# 全量导入快递员表
import_lg_courier(){
  import_data lg_courier "select
```

```
                            *
                        from lg_courier
                        where 1=1"
}

# 全量导入网点表
import_lg_dot(){
  import_data lg_dot "select
                            *
                        from lg_dot
                        where 1=1"
}

# 全量导入公司网点关联表
import_lg_company_dot_map(){
  import_data lg_company_dot_map "select
                            *
                        from lg_company_dot_map
                        where 1=1"
}

# 全量导入公司网点关联表
import_lg_company(){
  import_data lg_company "select
                            *
                        from lg_company
                        where 1=1"
}

# 全量导入客户表
import_lg_customer(){
  import_data lg_customer "select
                            *
                        from lg_customer
                        where 1=1"
}

# 全量导入客户地址关联表
import_lg_customer_address_map(){
  import_data lg_customer_address_map "select
                            *
                        from lg_customer_address_map
                        where 1=1"
}

# 全量导入字典表
import_lg_codes(){
  import_data lg_codes "select
                            *
                        from lg_codes
                        where 1=1"
}


#调用全量导入数据方法
import_lg_address
import_lg_entrepots
import_lg_areas
```

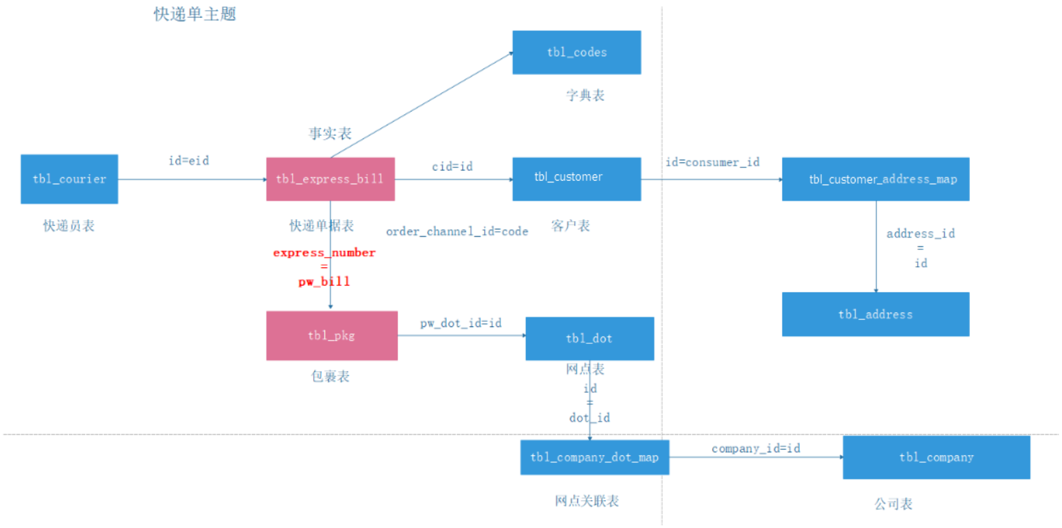
```
import_lg_courier
import_lg_dot
import_lg_company_dot_map
import_lg_company
import_lg_customer
import_lg_customer_address_map
import_lg_codes

#注意sqoop导入数据的方式，对于Hive分区表来说需要执行添加分区操作，数据才能被识别到
$Hive -e "alter table lg_ods.lg_address add partition(dt='$do_date');
alter table lg_ods.lg_areas add partition(dt='$do_date');
alter table lg_ods.lg_courier add partition(dt='$do_date');
alter table lg_ods.lg_dot add partition(dt='$do_date');
alter table lg_ods.lg_company_dot_map add partition(dt='$do_date');
alter table lg_ods.lg_company add partition(dt='$do_date');
alter table lg_ods.lg_customer add partition(dt='$do_date');
alter table lg_ods.lg_customer_address_map add
partition(dt='$do_date');
alter table lg_ods.lg_codes add partition(dt='$do_date');"
```

3.3 ETL

1 指标明细

指标列表	维度
最大快递单数	各类客户最大快递单数
	各渠道最大快递单数
	各网点最大快递单数
	各终端最大快递单数
最小快递单数	各类客户最小快递单数
	各渠道最小快递单数
	各网点最小快递单数
	各终端最小快递单数
平均快递单数	各类客户平均快递单数
	各渠道平均快递单数
	各网点平均快递单数
	各终端平均快递单数

2 表关系示意图



事实表

表名	描述
lg_express_bill	快递单据表
lg_pkg	包裹表

维度表

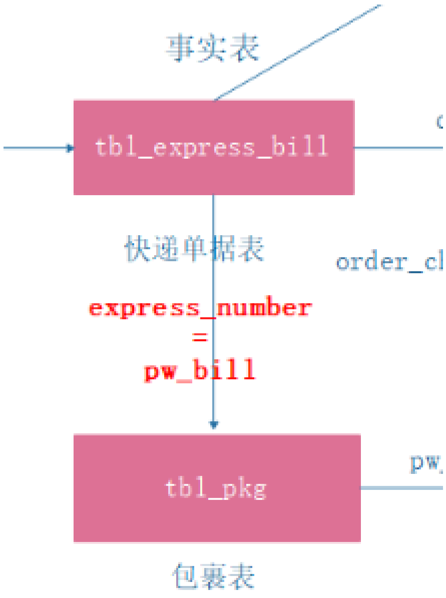
表名	描述
lg_consumer	客户表
lg_courier	快递员表
lg_areas	区域表
lg_dot	网点表
lg_company_dot_map	公司网点关联表
lg_company	公司表
lg_customer_address_map	客户地址关联表
lg_address	客户地址表
lg_codes	字典表

3 预处理

创建lg_dwd层表

3.1 事实表

事实表拉宽



创建lg_dwd层表

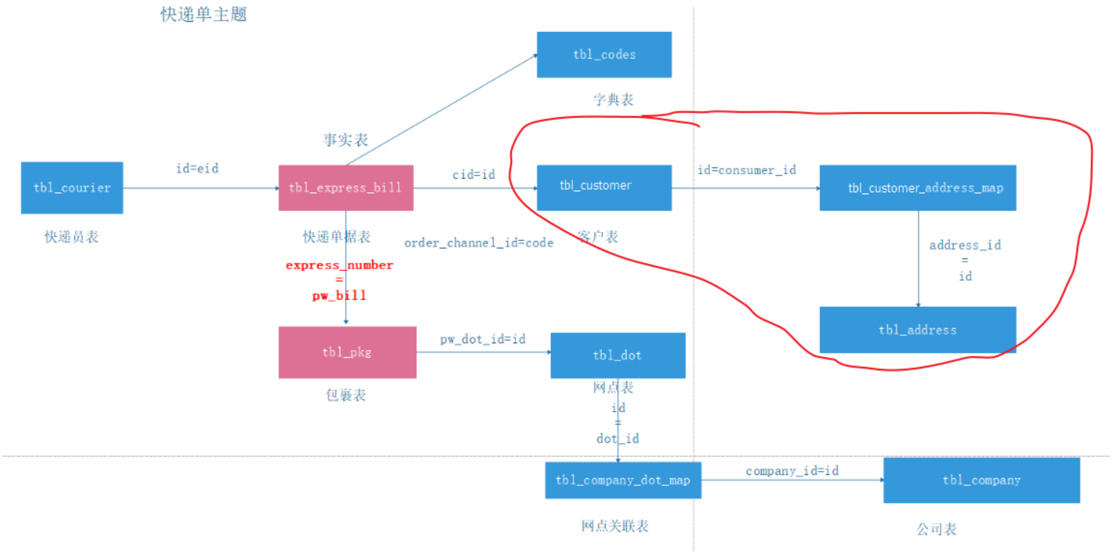
```
create database lg_dwd;
use lg_dwd;
DROP TABLE IF EXISTS `lg_dwd.expression_lg_express_pkg`;
CREATE TABLE `lg_dwd.expression_lg_express_pkg` (
id                        string,
express_number            string,
cid                       string,
eid                       string,
order_channel_id          string,
order_dt                  string,
order_terminal_type       string,
order_terminal_os_type    string,
reserve_dt                string,
is_collect_package_timeout       string,
timeout_dt                string,
cdt                       string,
udt                       string,
remark                    string,
pw_bill                   string,
pw_dot_id                 string
)COMMENT '快递单据表-包裹关联表'
partitioned by (dt string) STORED AS PARQUET ;
```

ETL-SQL

```
insert overwrite table lg_dwd.expression_lg_express_pkg partition(dt='2020-06-
02')
select
ebill.id ,
ebill.express_number ,
ebill.cid ,
ebill.eid ,
ebill.order_channel_id ,
ebill.order_dt,
ebill.order_terminal_type,
ebill.order_terminal_os_type ,
ebill.reserve_dt ,
ebill.is_collect_package_timeout ,
ebill.timeout_dt ,
ebill.cdt ,
ebill.udt ,
ebill.remark ,
pkg.pw_bill,
pkg.pw_dot_id
from
(select * from lg_ods.lg_express_bill where dt='2020-06-02') ebill
left join (select pw_bill,pw_dot_id from lg_ods.lg_pkg where dt ='2020-06-02')
pkg
on ebill.express_number =pkg.pw_bill ;
```

3.2 维度表

DIM



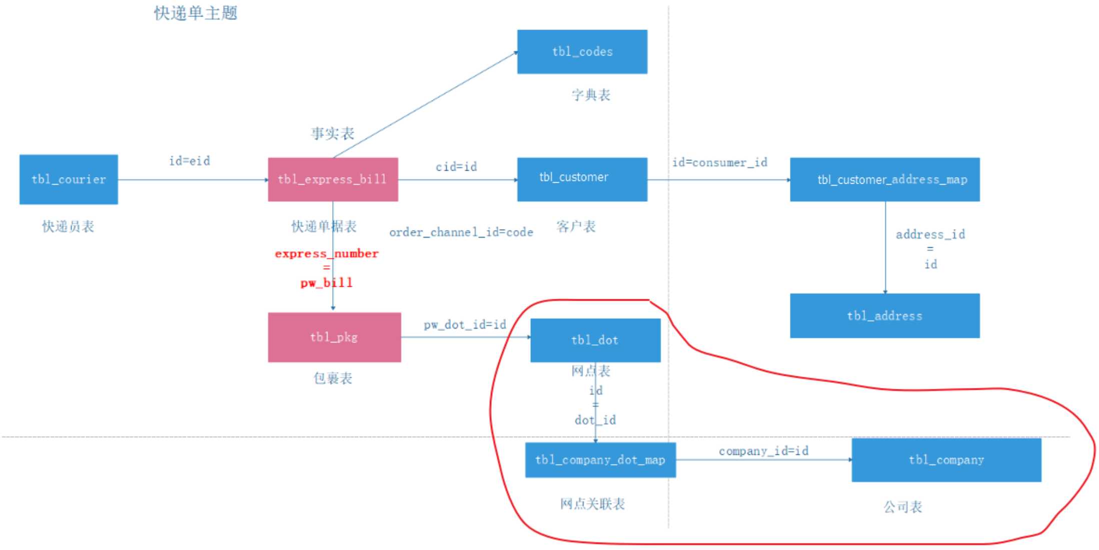
客户地址信息拉宽表

创建表

```
drop table if exists dim.express_customer_address;
create table dim.express_customer_address(
id                   string,
cname                   string,
caddress                string,
type                    string)COMMENT '快递单主题-客户地址信息'
partitioned by (dt string) STORED AS PARQUET ;

insert overwrite table dim.express_customer_address partition(dt='2020-06-02')
select
customer.id as id,
customer.name as cname,
address.detail_addr as caddress,
customer.type  as type
from
(select name,id,type from lg_ods.lg_customer where dt ='2020-06-02') customer
left join (select address_id,consumer_id from lg_ods.lg_customer_address_map
where dt='2020-06-02' ) address_map
```

```
on customer.id=address_map.consumer_id
left join (select id,detail_addr from lg_ods.lg_address where dt ='2020-06-02')
address
on address_map.address_id=address.id ;
```

ETL-SQL



网点公司信息拉宽表

```
drop table if exists dim.express_company_dot_addr;
create table dim.express_company_dot_addr(
id                  string,
dot_name                string,
company_name            string)COMMENT '快递单主题-公司网点地址信息'
partitioned by (dt string) STORED AS PARQUET ;

insert overwrite table dim.express_company_dot_addr partition(dt='2020-06-02')
select
dot.id as id,
dot.dot_name as dot_name,
company.company_name as company_name
from (select id,dot_name from lg_ods.lg_dot where dt ='2020-06-02') dot
left join (select dot_id,company_id from lg_ods.lg_company_dot_map where dt
='2020-06-02') companydot
on dot.id=companydot.dot_id
left join (select company_name,id from lg_ods.lg_company where dt ='2020-06-02')
company
on company.id=companydot.company_id ;
```

3.4 指标统计

计算的字段

指标列表	维度
快递单数	总快递单数
最大快递单数	各类客户最大快递单数
	各渠道最大快递单数
	各网点最大快递单数
	各终端最大快递单数
最小快递单数	各类客户最小快递单数
	各渠道最小快递单数
	各网点最小快递单数
	各终端最小快递单数
平均快递单数	各类客户平均快递单数
	各渠道平均快递单数
	各网点平均快递单数
	各终端平均快递单数

```sql
-- 创建lg_dws层数据库
drop table if exists lg_dws.express_base_agg;
create table lg_dws.express_base_agg(
express_count           bigint,
customer_type           string,
dot_name                string,
channel_id              string,
terminal_type           string)COMMENT '快递单主题-初步汇总表'
partitioned by (dt string) STORED AS PARQUET ;

insert overwrite table lg_dws.express_base_agg partition(dt='2020-06-02')
select
count(express_number) as express_count,
t2.type as customer_type,
t3.dot_name as dot_name,
t1.order_channel_id as channel_id,
t1.order_terminal_type as terminal_type
from
(select * from lg_dwd.expression_lg_express_pkg where dt ='2020-06-02') t1
left join
(select * from lg_dim.express_customer_address where dt ='2020-06-02') t2
on t1.cid =t2.id
left join
(select * from lg_dim.express_company_dot_addr where dt ='2020-06-02') t3
on t1.pw_dot_id =t3.id
group by
t2.type,
t3.dot_name,
t1.order_channel_id,
t1.order_terminal_type;
```

指标统计sql

```
--最大快递单数

--各类客户最大快递单数
select sum(express_count) as customer_type_count ,t.customer_type as
customer_type from (select * from lg_dws.express_base_agg where dt='2020-06-02')
t group by t.customer_type order by customer_type_max_count desc limit 1;
--各渠道最大快递单数
select sum(express_count) as channel_max_count ,t.channel_id as channel_id from
(select * from lg_dws.express_base_agg where dt='2020-06-02') t group by
t.channel_id order by channel_max_count desc limit 1;
--各终端最大快递单数
select sum(express_count) as terminal_max_count ,t.terminal_type as
terminal_type from (select * from lg_dws.express_base_agg where dt='2020-06-02')
t group by t.terminal_type order by terminal_max_count desc limit 1;
--各网点最大快递单数
select sum(express_count) as dot_max_count ,t.dot_name as dot_name from (select
* from lg_dws.express_base_agg where dt='2020-06-02') t group by t.dot_name
order by dot_max_count desc limit 1;



--最小快递单数

--各类客户最小快递单数
select sum(express_count) as customer_type_min_count ,t.customer_type as
customer_type from (select * from lg_dws.express_base_agg where dt='2020-06-02')
t group by t.customer_type order by customer_type_min_count asc limit 1;
--各渠道最小快递单数
select sum(express_count) as channel_min_count ,t.channel_id as channel_id from
(select * from lg_dws.express_base_agg where dt='2020-06-02') t group by
t.channel_id order by channel_min_count asc limit 1;
--各终端最小快递单数
select sum(express_count) as terminal_min_count ,t.terminal_type as
terminal_type from (select * from lg_dws.express_base_agg where dt='2020-06-02')
t group by t.terminal_type order by terminal_min_count asc limit 1;
--各网点最小快递单数
select sum(express_count) as dot_min_count ,t.dot_name as dot_name from (select
* from lg_dws.express_base_agg where dt='2020-06-02') t group by t.dot_name
order by dot_min_count asc limit 1;

--创建lg_ads层表
drop table if exists lg_ads.express_metrics;
create table lg_ads.express_metrics(
customer_type_max_count bigint,
customer_max_type       string,
channel_max_count       bigint,
channel_max_id          string,
terminal_max_count      bigint,
terminal_max_type       string,
dot_max_count           bigint,
dot_max_name            string,
customer_type_min_count bigint,
customer_min_type       string,
channel_min_count       bigint,
channel_min_id          string,
terminal_min_count      bigint,
terminal_min_type       string,
dot_min_count           bigint,
dot_min_name            string,
dt                      string
)COMMENT '快递单主题-指标表' row format delimited fields terminated by ',' STORED
AS textfile ;
--汇总sql
insert into table lg_ads.express_metrics
select
t1.customer_type_max_count,
t1.customer_type as customer_max_type,
t2.channel_max_count,
t2.channel_id as channel_max_id,
t3.terminal_max_count,
t3.terminal_type as terminal_max_type,
t4.dot_max_count,
t4.dot_name as dot_max_name,
t5.customer_type_min_count,
t5.customer_type as customer_min_type ,
t6.channel_min_count,
t6.channel_id as channel_min_id,
t7.terminal_min_count,
t7.terminal_type as terminal_min_type,
t8.dot_min_count,
t8.dot_name as dot_min_name,
'2020-06-02'
  from
  (select sum(express_count) as customer_type_max_count ,t.customer_type as
customer_type from (select * from lg_dws.express_base_agg where dt='2020-06-02')
t group by t.customer_type order by customer_type_max_count desc limit 1) t1
join
  (select sum(express_count) as channel_max_count ,t.channel_id as channel_id
from (select * from lg_dws.express_base_agg where dt='2020-06-02') t group by
t.channel_id order by channel_max_count desc limit 1 ) t2 join
  (select sum(express_count) as terminal_max_count ,t.terminal_type as
terminal_type from (select * from lg_dws.express_base_agg where dt='2020-06-02')
t group by t.terminal_type order by terminal_max_count desc limit 1) t3 join
  (select sum(express_count) as dot_max_count ,t.dot_name as dot_name from
(select * from lg_dws.express_base_agg where dt='2020-06-02') t group by
t.dot_name order by dot_max_count desc limit 1) t4 join
(select sum(express_count) as customer_type_min_count ,t.customer_type as
customer_type from (select * from lg_dws.express_base_agg where dt='2020-06-02')
t group by t.customer_type order by customer_type_min_count asc limit 1) t5 join
(select sum(express_count) as channel_min_count ,t.channel_id as channel_id from
(select * from lg_dws.express_base_agg where dt='2020-06-02') t group by
t.channel_id order by channel_min_count asc limit 1) t6 join
(select sum(express_count) as terminal_min_count ,t.terminal_type as
terminal_type from (select * from lg_dws.express_base_agg where dt='2020-06-02')
t group by t.terminal_type order by terminal_min_count asc limit 1) t7 join
(select sum(express_count) as dot_min_count ,t.dot_name as dot_name from (select
* from lg_dws.express_base_agg where dt='2020-06-02') t group by t.dot_name
order by dot_min_count asc limit 1) t8;
```

导出指标数据

创建Mysql指标表

```
CREATE TABLE `express_agg_metrics` (
  `customer_type_max_count` bigint(20) DEFAULT NULL,
  `customer_max_type` varchar(10) DEFAULT NULL,
  `channel_max_count` bigint(20) DEFAULT NULL,
  `channel_max_id` varchar(10) DEFAULT NULL,
  `terminal_max_count` bigint(20) DEFAULT NULL,
  `terminal_max_type` varchar(10) DEFAULT NULL,
  `dot_max_count` bigint(20) DEFAULT NULL,
  `dot_max_name` varchar(25) DEFAULT NULL,
  `customer_type_min_count` bigint(20) DEFAULT NULL,
  `customer_min_type` varchar(10) DEFAULT NULL,
  `channel_min_count` bigint(20) DEFAULT NULL,
  `channel_min_id` varchar(10) DEFAULT NULL,
  `terminal_min_count` bigint(20) DEFAULT NULL,
  `terminal_min_type` varchar(10) DEFAULT NULL,
  `dot_min_count` bigint(20) DEFAULT NULL,
  `dot_min_name` varchar(25) DEFAULT NULL,
  `dt` varchar(25) NOT NULL,
  PRIMARY KEY (`dt`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

使用Sqoop 工具导出

```
#!/bin/bash
source /etc/profile

#定义sqoop命令位置，Hive命令位置，在hadoop2
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期


#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
export_data(){
$sqoop export \
--connect "jdbc:mysql://hadoop1:3306/lg_metrics?useUnicode=true&characterEncoding=utf-8" \
--username root \
--password 123456 \
--table $1 \
--export-dir /user/hive/warehouse/lg_ads.db/$2/ \
--num-mappers 1 \
--input-fields-terminated-by ',' \
--update-mode allowinsert \
--update-key dt
}


# 导出快递单指标数据
export_lg_express_metrics(){
    export_data express_agg_metrics express_metrics
}

#导出数据方法
export_lg_express_metrics
```

第四节 运单主题

4.1 lg_ods层建表

建表语句

```
--运单表
DROP TABLE IF EXISTS `lg_ods.lg_waybill`;
CREATE TABLE `lg_ods.lg_waybill` (
  `id` string,
  `express_bill_number` string,
  `waybill_number` string,
  `cid` string,
  `eid` string,
  `order_channel_id` string,
  `order_dt` string,
  `order_terminal_type` string,
  `order_terminal_os_type` string,
  `reserve_dt` string,
  `is_collect_package_timeout` string,
  `pkg_id` string,
  `pkg_number` string,
  `timeout_dt` string,
  `transform_type` string,
  `delivery_customer_name` string,
  `delivery_addr` string,
  `delivery_mobile` string,
  `delivery_tel` string,
  `receive_customer_name` string,
  `receive_addr` string,
  `receive_mobile` string,
  `receive_tel` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '运单表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';


--转运记录表:lg_transport_record

DROP TABLE IF EXISTS `lg_ods.lg_transport_record`;
CREATE TABLE `lg_ods.lg_transport_record` (
  `id` string,
  `pw_id` string,
  `pw_waybill_id` string,
  `pw_waybill_number` string,
  `ow_id` string,
  `ow_waybill_id` string,
  `ow_waybill_number` string,
  `sw_id` string,
  `ew_id` string,
  `transport_tool_id` string,
  `pw_driver1_id` string,
  `pw_driver2_id` string,
  `pw_driver3_id` string,
  `ow_driver1_id` string,
  `ow_driver2_id` string,
  `ow_driver3_id` string,
  `route_id` string,
  `distance` string,
  `duration` string,
  `state` string,
  `start_vehicle_dt` string,
  `predict_arrivals_dt` string,
  `actual_arrivals_dt` string,
  `cdt` string,
  `udt` string,
  `remark` string
)COMMENT '转运记录表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';


--线路表:lg_route

DROP TABLE IF EXISTS `lg_ods.lg_route`;
CREATE TABLE `lg_ods.lg_route` (
  `id` string,
  `start_station` string,
  `start_station_area_id` string,
  `start_warehouse_id` string,
  `end_station` string,
  `end_station_area_id` string,
  `end_warehouse_id` string,
  `mileage_m` string,
  `time_consumer_minute` string,
  `state` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '线路表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';

--车辆表:lg_transport_tool

DROP TABLE IF EXISTS `lg_ods.lg_transport_tool`;
CREATE TABLE `lg_ods.lg_transport_tool` (
  `id` string,
  `brand` string,
  `model` string,
  `type` string,
  `given_load` string,
  `load_cn_unit` string,
  `load_en_unit` string,
  `buy_dt` string,
  `license_plate` string,
  `state` string,
  `cdt` string,
  `udt` string,
  `remark` string
) COMMENT '车辆表'
PARTITIONED BY (`dt` string) row format delimited fields terminated by ',';
```

4.2 数据采集

业务数据保存在MySQL中，每日凌晨导入上一天的表数据。

- 事实表
  - 全量导入

```
#!/bin/bash
source /etc/profile
##如果第一个参数不为空，则作为工作日期使用
if [ -n "$1" ]
then
do_date=$1
else
##昨天日期，减一
do_date=`date -d "-1 day" +"%Y%m%d"`
fi
#定义sqoop命令位置，Hive命令位置，在hadoop2
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期


#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
import_data(){
$sqoop import \
--connect jdbc:mysql://hadoop1:3306/lg_logistics \
--username root \
--password 123456 \
--target-dir /user/hive/warehouse/lg_ods.db/$1/dt=$do_date \
--delete-target-dir \
--query "$2 and  \$CONDITIONS" \
--num-mappers 1 \
--fields-terminated-by ',' \
--null-string '\\N' \
--null-non-string '\\N'
}


# 全量导入运单表数据
import_lg_waybill(){
  import_data lg_waybill "select
                            *
                        from lg_waybill
                        where 1=1"
}
```

```
# 全量导入转运记录表
import_lg_transport_record(){
  import_data lg_transport_record "select
                            *
                        from lg_transport_record
                        where 1=1"
}



#调用全量导入数据方法
import_lg_waybill
import_lg_transport_record

#注意sqoop导入数据的方式，对于Hive分区表来说需要执行添加分区操作，数据才能被识别到
$Hive -e "alter table lg_ods.lg_waybill add partition(dt='$do_date');
alter table lg_ods.lg_transport_record add partition(dt='$do_date');"
```

- 增量导入

```
#!/bin/bash
source /etc/profile
##如果第一个参数不为空，则作为工作日期使用
if [ -n "$1" ]
then
do_date=$1
else
##昨天日期，减一
do_date=`date -d "-1 day" +"%Y%m%d"`
fi
#定义sqoop命令位置，Hive命令位置，在hadoop2
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期


#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
import_data(){
$sqoop import \
--connect jdbc:mysql://hadoop1:3306/lg_logistics \
--username root \
--password 123456 \
--target-dir /user/hive/warehouse/lg_ods.db/$1/dt=$do_date \
--delete-target-dir \
--query "$2 and  \$CONDITIONS" \
--num-mappers 1 \
--fields-terminated-by ',' \
--null-string '\\N' \
--null-non-string '\\N'
}


# 增量导入运单数据
import_lg_waybill(){
  import_data lg_waybill "select
                            *
```

```
                        from lg_waybill
                        WHERE DATE_FORMAT(cdt, '%Y%m%d') = '${do_date}'
"
}

# 增量导入转运记录
import_lg_transport_record(){
  import_data lg_transport_record "select
                            *
                        from lg_transport_record
                       WHERE DATE_FORMAT(cdt, '%Y%m%d') = '${do_date}'
"
}



#调用全量导入数据方法
import_lg_waybill
import_lg_transport_record


#注意sqoop导入数据的方式，对于Hive分区表来说需要执行添加分区操作，数据才能被识别到
$Hive -e "alter table lg_ods.lg_express_bill add
partition(dt='$do_date');
alter table lg_ods.lg_pkg add partition(dt='$do_date');"
```

- 维度表
  - 全量导入

```
#!/bin/bash
source /etc/profile
##如果第一个参数不为空，则作为工作日期使用
if [ -n "$1" ]
then
do_date=$1
else
##昨天日期，减一
do_date=`date -d "-1 day" +"%Y%m%d"`
fi
#定义sqoop命令位置，Hive命令位置，在hadoop2
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期


#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
import_data(){
$sqoop import \
--connect jdbc:mysql://hadoop1:3306/lg_logistics \
--username root \
--password 123456 \
--target-dir /user/hive/warehouse/lg_ods.db/$1/dt=$do_date \
--delete-target-dir \
--query "$2 and  \$CONDITIONS" \
--num-mappers 1 \
--fields-terminated-by ',' \
--null-string '\\N' \
--null-non-string '\\N'
}


# 全量导入线路表
import_lg_route(){
  import_data lg_route "select
                            *
                        from lg_route
                        where 1=1"
}

# 全量导入车辆表
import_lg_transport_tool(){
  import_data lg_transport_tool "select
                            *
                        from lg_transport_tool
                        where 1=1"
}

#调用全量导入数据方法
import_lg_route
import_lg_transport_tool
#注意sqoop导入数据的方式，对于Hive分区表来说需要执行添加分区操作，数据才能被识别到
$Hive -e "alter table lg_ods.lg_route add partition(dt='$do_date');
alter table lg_ods.lg_transport_tool add partition(dt='$do_date');"
```

4.3 ETL

1 指标明细

指标列表	维度
运单数	总运单数
最大运单数	最大区域运单数
	各分公司最大运单数
	各网点最大运单数
	各线路最大运单数
	各运输工具最大运单数
	各类客户最大运单数
最小运单数	各区域最小运单数
	各分公司最小运单数
	各网点最小运单数
	各线路最小运单数
	各运输工具最小运单数
	各类客户最小运单数
平均运单数	各区域平均运单数
	各分公司平均运单数
	各网点平均运单数
	各线路平均运单数
	各运输工具平均运单数
	各类客户平均运单数

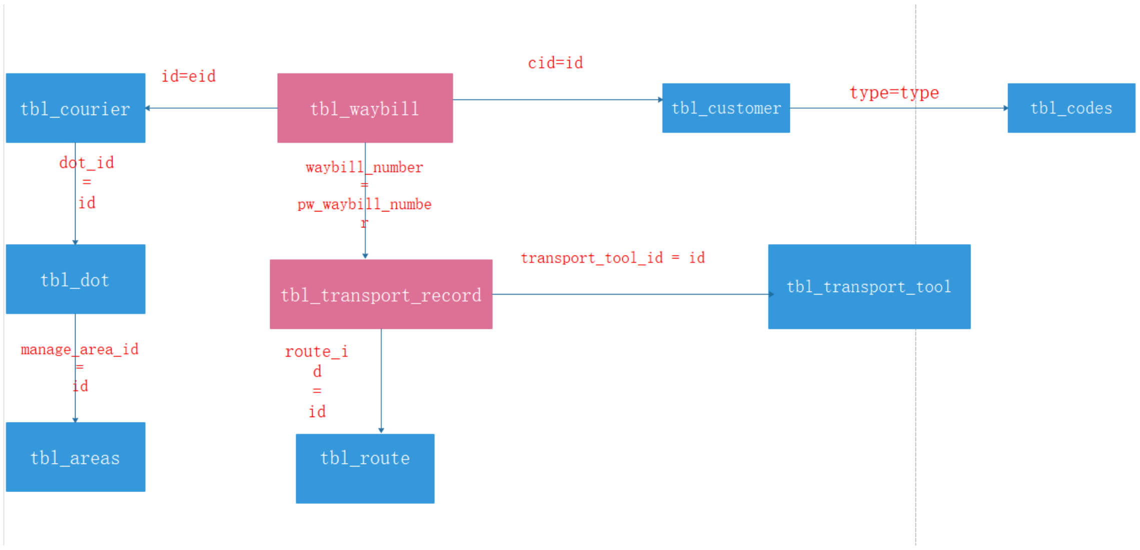
2 表关系示意图

事实表

表名	描述
lg_waybill	运单表
lg_transport_record	转运记录表

维度表

表名	描述
lg_courier	快递员表
lg_areas	区域表
lg_route	线路表
lg_dot	网点表
lg_company	公司表
lg_customer	客户表
lg_transport_tool	车辆表
lg_codes	物流系统码表



3 预处理

3.1 事实表

运单与转运记录宽表

```
drop table if exists lg_dwd.waybill_way_tran_record;
create table lg_dwd.waybill_way_tran_record(
express_bill_number      string,
waybill_number           string,
cid                      string,
eid                      string,
order_channel_id         string,
order_dt                 string,
order_terminal_type      string,
order_terminal_os_type   string,
reserve_dt               string,
is_collect_package_timeout      string,
pkg_id                   string,
pkg_number               string,
timeout_dt               string,
transform_type           string,
delivery_customer_name   string,
delivery_addr            string,
delivery_mobile          string,
delivery_tel            string,
receive_customer_name   string,
receive_addr            string,
receive_mobile          string,
receive_tel             string,
pw_id                   string,
pw_waybill_id           string,
pw_waybill_number       string,
ow_id                   string,
ow_waybill_id           string,
ow_waybill_number       string,
sw_id                   string,
ew_id                   string,
transport_tool_id       string,
pw_driver1_id           string,
pw_driver2_id           string,
pw_driver3_id           string,
ow_driver1_id           string,
ow_driver2_id           string,
ow_driver3_id           string,
route_id                string,
distance                string,
duration                string,
state                   string,
start_vehicle_dt        string,
predict_arrivals_dt     string,
actual_arrivals_dt      string
)COMMENT '运单主题-运单与转运记录事实表'
partitioned by (dt string) STORED AS PARQUET ;

--插入数据
insert overwrite   table lg_dwd.waybill_way_tran_record partition(dt='2020-06-02')
select
t1.express_bill_number       ,
t1.waybill_number            ,
t1.cid                       ,
t1.eid                       ,
t1.order_channel_id          ,
t1.order_dt                  ,
t1.order_terminal_type       ,
t1.order_terminal_os_type    ,
t1.reserve_dt                ,
t1.is_collect_package_timeout,
t1.pkg_id                  ,
t1.pkg_number              ,
t1.timeout_dt              ,
t1.transform_type          ,
t1.delivery_customer_name  ,
t1.delivery_addr           ,
t1.delivery_mobile         ,
t1.delivery_tel            ,
t1.receive_customer_name   ,
t1.receive_addr            ,
t1.receive_mobile          ,
t1.receive_tel             ,

t2.pw_id                   ,
t2.pw_waybill_id           ,
t2.pw_waybill_number       ,
t2.ow_id                   ,
t2.ow_waybill_id           ,
t2.ow_waybill_number       ,
t2.sw_id                   ,
t2.ew_id                   ,
t2.transport_tool_id       ,
t2.pw_driver1_id           ,
t2.pw_driver2_id           ,
t2.pw_driver3_id           ,
t2.ow_driver1_id           ,
t2.ow_driver2_id           ,
t2.ow_driver3_id           ,
t2.route_id                ,
t2.distance                ,
t2.duration                ,
t2.state                   ,
t2.start_vehicle_dt        ,
t2.predict_arrivals_dt     ,
t2.actual_arrivals_dt
from (select * from lg_ods.lg_waybill where dt ='2020-06-02') t1
left join (select * from lg_ods.lg_transport_record where dt ='2020-06-02') t2
on t1.waybill_number =t2.pw_waybill_number ;
```

3.2 维度表

创建客户字典宽表

```
drop table if exists dim.waybill_customer_codes;
create table dim.waybill_customer_codes (
id                      string,
name                    string,
tel                     string,
mobile                  string,
email                   string,
type                    string,
is_own_reg              string,
reg_dt                  string,
reg_channel_id          string,
state                   string,
last_login_dt           string,
code_name               string,
code_type               string,
code                    string,
code_desc               string,
code_cust_type          string,
codes_state             string
)COMMENT '运单主题-客户字典宽表'
partitioned by (dt string) STORED AS PARQUET ;

insert overwrite table dim.waybill_customer_codes partition(dt='2020-06-02')
select
t1.id            ,
t1.name           ,
t1.tel            ,
t1.mobile         ,
t1.email          ,
t1.type           ,
t1.is_own_reg     ,
t1.reg_dt         ,
t1.reg_channel_id ,
t1.state          ,

t1.last_login_dt  ,

t2.name  as code_name    ,
t2.type  as code_type    ,
t2.code      ,
t2.code_desc ,
t2.code_type as code_cust_type,
t2.state   as codes_state
from (select * from lg_ods.lg_customer where dt ='2020-06-02') t1
left join (select * from lg_ods.lg_codes where dt ='2020-06-02' and type ='1')
t2
on t1.type =t2.type ;
```

创建网点区域宽表

```
drop table if exists dim.waybill_courier_dot_area;
create table dim.waybill_courier_dot_area (
id                      string,
job_num                 string,
name                    string,
birathday               string,
tel                     string,
pda_num                 string,
car_id                  string,
postal_standard_id      string,
work_time_id            string,
dot_id                  string,
state                   string,
dot_number              string,
dot_name                string,
dot_addr                string,
dot_gis_addr            string,
dot_tel                 string,
company_id              string,
manage_area_id          string,
manage_area_gis         string,
dot_state               string,
area_name               string,
pid                     string,
sname                   string,
level                   string,
citycode                string,
yzcode                  string,
mername                 string,
lng                     string,
lat                     string,
pinyin                  string) COMMENT '运单主题-快递员网点区域宽表'
partitioned by (dt string) STORED AS PARQUET ;


insert overwrite table dim.waybill_courier_dot_area partition(dt='2020-06-02')
select
t1.id                ,
t1.job_num           ,
t1.name              ,
t1.birathday         ,
t1.tel               ,
t1.pda_num           ,
t1.car_id            ,
t1.postal_standard_id,
t1.work_time_id      ,
t1.dot_id            ,
t1.state ,
t2.dot_number     ,
t2.dot_name       ,
t2.dot_addr       ,
t2.dot_gis_addr   ,
t2.dot_tel        ,
t2.company_id     ,
t2.manage_area_id ,
t2.manage_area_gis,
t2.state   as dot_state       ,

t3.name     as area_name  ,
t3.pid        ,
t3.sname      ,
t3.level      ,
t3.citycode   ,
t3.yzcode     ,
t3.mername    ,
t3.lng        ,
t3.lat        ,
t3.pinyin

from (select * from lg_ods.lg_courier where dt ='2020-06-02' ) t1
left join (select * from lg_ods.lg_dot where dt ='2020-06-02' ) t2
on t1.dot_id=t2.id
left join (select * from lg_ods.lg_areas where dt ='2020-06-02' ) t3
on t2.manage_area_id =t3.id;
```

线路表

车辆表

4.4 指标统计

计算指标

指标描述
最大区域运单数
最小区域运单数
各网点最大运单数
各网点最小运单数
各线路最大运单数
各线路最小运单数
各运输工具最大运单数
各运输工具最小运单数
各类客户最大运单数
各类客户最小运单数

```
--创建lg_dws层表
drop table if exists lg_dws.waybill_base_agg;
create table lg_dws.waybill_base_agg
(waybill_count           bigint,
area_name               string,
dot_id                  string,
dot_name                string,
route_id                string,
tool_id                 string,
cus_type                string)COMMENT '运单主题-lg_dws基础汇总表'
partitioned by (dt string) STORED AS PARQUET ;

insert overwrite table lg_dws.waybill_base_agg partition(dt='2020-06-02')
select
count(record.waybill_number) as waybill_count,
courier_area.area_name as area_name,
courier_area.dot_id as dot_id,
courier_area.dot_name as dot_name,
route.id as route_id,
tran_tool.id as tool_id,
cus_codes.type as cus_type
from
(select * from lg_dwd.waybill_way_tran_record where dt ='2020-06-02') record
left join
(select * from dim.waybill_customer_codes where dt='2020-06-02') cus_codes
on record.cid= cus_codes.id
left join
(select * from dim.waybill_courier_dot_area where dt='2020-06-02') courier_area
on record.eid= courier_area.id
left join
(select * from lg_ods.lg_route where dt ='2020-06-02') route
on record.route_id =route.id
left join
(select * from lg_ods.lg_transport_tool where dt ='2020-06-02') tran_tool
on record.transport_tool_id =tran_tool.id
```

```
group by courier_area.area_name,courier_area.dot_id,courier_area.dot_name,route.id,tran_tool.id,cus_codes.type;
```

指标统计

```
--运单数

--各区域最大运单数
select sum(t.waybill_count) as area_max_count, t.area_name from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.area_name  order by area_max_count desc limit 1;
--各网点最大运单数
select sum(t.waybill_count) as dot_max_count, t.dot_name, t.dot_id from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.dot_id,t.dot_name  order by dot_max_count desc limit 1;
--各线路最大运单数
select sum(t.waybill_count) as route_max_count,  t.route_id from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.route_id  order by route_max_count desc limit 1;
--各运输工具最大运单数
select sum(t.waybill_count) as tool_max_count,  t.tool_id from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.tool_id  order by tool_max_count desc limit 1;
--各类客户最大运单数
select sum(t.waybill_count) as cus_max_count,  t.cus_type from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.cus_type  order by cus_max_count desc limit 1;




--各区域最小运单数
select sum(t.waybill_count) as area_min_count, t.area_name from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.area_name  order by area_min_count asc limit 1;
--各网点最小运单数
select sum(t.waybill_count) as dot_min_count, t.dot_name, t.dot_id from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.dot_id,t.dot_name  order by dot_min_count asc limit 1;
--各线路最小运单数
select sum(t.waybill_count) as route_min_count,  t.route_id from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.route_id  order by route_min_count asc limit 1;
--各运输工具最小运单数
select sum(t.waybill_count) as tool_min_count,  t.tool_id from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.tool_id  order by tool_min_count desc limit 1;
--各类客户最小运单数
select sum(t.waybill_count) as cus_min_count,  t.cus_type from (select * from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.cus_type  order by cus_min_count asc limit 1;




--创建lg_ads层表
drop table if exists lg_ads.waybill_metrics;
create table lg_ads.waybill_metrics(
area_max_count          bigint,
area_max_name           string,
dot_max_count           bigint,
dot_max_name            string,
route_max_count         bigint,
route_max_id            string,
tool_max_count          bigint,
tool_max_id             string,
cus_max_count           bigint,
cus_max_type            string,
area_min_count          bigint,
area_min_name           string,
dot_min_count           bigint,
dot_min_name            string,
route_min_count         bigint,
route_min_id            string,
tool_min_count          bigint,
tool_min_id             string,
cus_min_count           bigint,
cus_min_type            string ,
dt                      string
)COMMENT '运主题-指标表' row format delimited fields terminated by ',' STORED AS
textfile ;

--汇总sql
insert overwrite table lg_ads.waybill_metrics
select
t1.area_max_count,
t1.area_name as area_max_name,
t2.dot_max_count,
t2.dot_name as dot_max_name,
t3.route_max_count,
t3.route_id as route_max_id,
t4.tool_max_count,
t4.tool_id as tool_max_id,
t5.cus_max_count,
t5.cus_type as cus_max_type,
t6.area_min_count,
t6.area_name as area_min_name,
t7.dot_min_count,
t7.dot_name as dot_min_name,
t8.route_min_count,
t8.route_id as route_min_id,
t9.tool_min_count,
t9.tool_id as tool_min_id,
t10.cus_min_count,
t10.cus_type as cus_min_type,
'2020-06-02'
from
(select sum(t.waybill_count) as area_max_count, t.area_name from (select * from
lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.area_name  order by
area_max_count desc limit 1) t1
join
(select sum(t.waybill_count) as dot_max_count, t.dot_name, t.dot_id from (select
* from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by
t.dot_id,t.dot_name  order by dot_max_count desc limit 1) t2
join
```

```
(select sum(t.waybill_count) as route_max_count,  t.route_id from (select * from
lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.route_id  order by
route_max_count desc limit 1) t3
join
(select sum(t.waybill_count) as tool_max_count,  t.tool_id from (select * from
lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.tool_id  order by
tool_max_count desc limit 1) t4
join
(select sum(t.waybill_count) as cus_max_count,  t.cus_type from (select * from
lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.cus_type  order by
cus_max_count desc limit 1) t5
join
(select sum(t.waybill_count) as area_min_count, t.area_name from (select * from
lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.area_name  order by
area_min_count asc limit 1) t6
join
(select sum(t.waybill_count) as dot_min_count, t.dot_name, t.dot_id from (select
* from lg_dws.waybill_base_agg where dt ='2020-06-02') t group by
t.dot_id,t.dot_name  order by dot_min_count asc limit 1) t7
join
(select sum(t.waybill_count) as route_min_count,  t.route_id from (select * from
lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.route_id  order by
route_min_count asc limit 1) t8
join
(select sum(t.waybill_count) as tool_min_count,  t.tool_id from (select * from
lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.tool_id  order by
tool_min_count desc limit 1) t9
join
(select sum(t.waybill_count) as cus_min_count,  t.cus_type from (select * from
lg_dws.waybill_base_agg where dt ='2020-06-02') t group by t.cus_type  order by
cus_min_count asc limit 1) t10;
```

导出指标数据

创建Mysql指标表

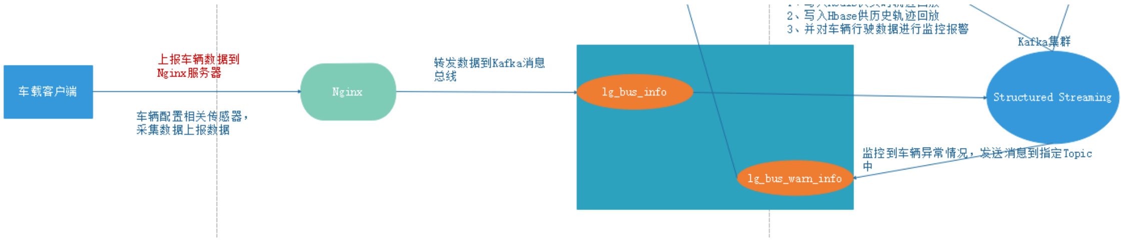
```
CREATE TABLE `waybill_agg_metrics` (
  `area_max_count` bigint(20) DEFAULT NULL,
  `area_max_name` varchar(50) DEFAULT NULL,
  `dot_max_count` bigint(20) DEFAULT NULL,
  `dot_max_name` varchar(50) DEFAULT NULL,
  `route_max_count` bigint(20) DEFAULT NULL,
  `route_max_id` varchar(25) DEFAULT NULL,
  `tool_max_count` bigint(20) DEFAULT NULL,
  `tool_max_id` varchar(25) DEFAULT NULL,
  `cus_max_count` bigint(20) DEFAULT NULL,
  `cus_max_type` varchar(25) DEFAULT NULL,
  `area_min_count` bigint(20) DEFAULT NULL,
  `area_min_name` varchar(50) DEFAULT NULL,
  `dot_min_count` bigint(20) DEFAULT NULL,
  `dot_min_name` varchar(50) DEFAULT NULL,
  `route_min_count` bigint(20) DEFAULT NULL,
  `route_min_id` varchar(25) DEFAULT NULL,
  `tool_min_count` bigint(20) DEFAULT NULL,
  `tool_min_id` varchar(25) DEFAULT NULL,
  `cus_min_count` bigint(20) DEFAULT NULL,
  `cus_min_type` varchar(25) DEFAULT NULL,
  `dt` varchar(25) NOT NULL,
  PRIMARY KEY (`dt`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

使用Sqoop 工具导出

```
#!/bin/bash
source /etc/profile

#定义sqoop命令位置，Hive命令位置，在hadoop2
sqoop=/opt/cloudera/parcels/CDH/bin/sqoop
Hive=/opt/cloudera/parcels/CDH/bin/hive
#定义工作日期


#编写导入数据通用方法 接收两个参数：第一个：表名，第二个：查询语句
export_data(){
$sqoop export \
--connect "jdbc:mysql://hadoop1:3306/lg_metrics?
useUnicode=true&characterEncoding=utf-8" \
--username root \
--password 123456 \
--table $1 \
--export-dir /user/hive/warehouse/lg_ads.db/$2/ \
--num-mappers 1 \
--input-fields-terminated-by ',' \
--update-mode allowinsert \
--update-key dt
}


# 导出快递单指标数据
export_lg_waybill_metrics(){
    export_data waybill_agg_metrics waybill_metrics

}

#导出数据方法
export_lg_waybill_metrics
```

11 实时数据采集

架构流程



车辆会装有各种传感器以及与后台通信的客户端，通过客户端把相关数据传送到后台。

第一节 配置Nginx

1、 安装git工具，安装wget下载工具

```
yum install wget git -y
yum install gcc-c++ -y
```

2、切换到/usr/local/src目录，然后将kafka的 客户端源码使用git clone到本地

```
cd /usr/local/src
git clone https://github.com/edenhill/librdkafka
```

注意：要保证下载成功！ ！ 不能出现failed字样

3、进入librdkafka目录，对kafka客户端源码进行编译

```
cd /usr/local/src/librdkafka
yum install -y gcc gcc-c++ pcre-devel zlib-devel
./configure
make && make install
```

注意：gcc编译的时候出现如下问题：

collect2 cannot find 'ld'

解决方式：yum reinstall binutils -y

4、安装nginx 整合kafka的插件，进入到/usr/local/src目录下，使用git clone nginx整合kafka的源码

```
cd /usr/local/src
git clone https://github.com/brg-liuwei/ngx_kafka_module
```

5、下载nginx源码包

```
cd /usr/local/src
wget http://nginx.org/download/nginx-1.17.8.tar.gz
tar -zxvf nginx-1.17.8.tar.gz
cd nginx-1.17.8
yum install gcc zlib zlib-devel openssl openssl-devel pcre pcre-devel -y
```

6、进入到nginx的源码目录下(编译nginx，包含与kafka整合的插件)

```
cd /usr/local/src/nginx-1.17.8
./configure --add-module=/usr/local/src/ngx_kafka_module/
make && make install

/usr/local/nginx/conf
```

7、修改nginx配置文件

设置一个location和kafka的topic信息

注意：

1、配置文件路径是：/usr/local/nginx/conf下的nginx.conf文件，不要与之前下载的nginx安装包目录混淆！！

2、内容都是添加到http标签内

完整文件参考

```
#user  nobody;
worker_processes  1;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

#pid        logs/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       mime.types;
    default_type  application/octet-stream;

    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    #access_log  logs/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;
    kafka;
    kafka_broker_list hadoop2:9092,hadoop3:9092,hadoop4:9092;
    #gzip  on;

    server {
        listen       80;
        server_name  localhost;
        #charset koi8-r;

        #access_log  logs/host.access.log  main;

        location / {
            root   html;
            index  index.html index.htm;
        }
        location = /log/lg_bus_info {
            kafka_topic lg_bus_info;
        }

        #error_page  404              /404.html;

        # redirect server error pages to the static page /50x.html
        #
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }

        # proxy the PHP scripts to Apache listening on 127.0.0.1:80
        #
        #location ~ \.php$ {
        #    proxy_pass   http://127.0.0.1;
        #}

        # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
        #
        #location ~ \.php$ {
        #    root           html;
        #    fastcgi_pass   127.0.0.1:9000;
        #    fastcgi_index  index.php;
        #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
        #    include        fastcgi_params;
        #}

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        #location ~ /\.ht {
        #    deny  all;
        #}
    }


    # another virtual host using mix of IP-, name-, and port-based configuration
    #
    #server {
    #    listen       8000;
    #    listen       somename:8080;
    #    server_name  somename  alias  another.alias;

    #    location / {
    #        root   html;
    #        index  index.html index.htm;
    #    }
    #}

    # HTTPS server
    #
    #server {
    #    listen       443 ssl;
    #    server_name  localhost;

    #    ssl_certificate      cert.pem;
    #    ssl_certificate_key  cert.key;

    #    ssl_session_cache    shared:SSL:1m;
    #    ssl_session_timeout  5m;

    #    ssl_ciphers  HIGH:!aNULL:!MD5;
    #    ssl_prefer_server_ciphers  on;

    #    location / {
    #        root   html;
    #        index  index.html index.htm;
    #    }
    #}

}
```

8、创建kafka topic

```
lg_bus_info
```

启动nginx

```
/usr/local/nginx/sbin/nginx
```

报错：

```
/usr/local/nginx/sbin/nginx: error while loading shared libraries:
librdkafka.so.1: cannot open shared object file: No such file or directory
解决方式
echo "/usr/local/lib" >> /etc/ld.so.conf
#手动加载
ldconfig
```

再次启动

```
/usr/local/nginx/sbin/nginx
#验证nginx启动
# ps -ef | grep nginx
root 104388 1 0 01:09 ? 00:00:00 nginx: master process
/usr/local/nginx/sbin/nginx
nobody 104389 104388 0 01:09 ? 00:00:00 nginx: worker process
root 104394 99171 0 01:09 pts/0 00:00:00 grep --color=auto nginx
```

9、测试

通过curl命令测试发送请求到Nginx

```
curl http://hadoop5/log/lagou_bus_info -d "this is a msg from nginx to kafka"
```

使用控制台消费者验证结果

```
kafka-console-consumer --bootstrap-server hadoop4:9092 --from-beginning --topic
lagou_bus_info
```

第二节 车载客户端

编写Java程序模拟车载客户端上传各种传感器采集的数据

```
package com.lg.collect;

import org.apache.http.client.ResponseHandler;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.BasicResponseHandler;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;

import java.io.IOException;
import java.util.Random;
import java.util.concurrent.TimeUnit;

public class DataClient {
    //调配编号
    static String[] deployArr = {"316d5c75-e860-4cc9-a7de-ea2148c244a0",
            "32102c12-6a73-4e03-80ab-96175a8ee686",
            "a97f6c0d-9086-4c68-9d24-8a7e89f39e5a",
            "adfgfdewr-5463243546-4c68-9d24-8a7e8",
            };
    //sim卡号
    static String[] simArr = {"1111", "2222", "3333", "4444"};
    //道路运输证
    static String[] transpotNumArr = {"ysz11111",
"ysz22222","ysz333333","ysz44444"};
    //车牌号
    static String[] plateNumArr = {"京A-11111", "京A-22222", "京A-33333", "京A-
44444"};
    //时间static
    static String[] timeStrArr = {"1594076827", "1594076527", "1594076327"};
    //经纬度
    static String[] lglatArr = {"116.437355_39.989739",
            "116.382306_39.960325",
            "116.623784_40.034688",
            "116.32139_39.81157",
            "116.45551_39.944381",};
    //速度
    static String[] speedArr = {"50", "60", "70", "80"};
    //方向
    static String[] directionArr = {"west", "east", "south", "north"};
    //里程
    static String[] mileageArr = {"6000", "7000", "8000", "9000"};
    //剩余油量
    static String[] oilRemainArr = {"20", "30", "70", "80"};
    //载重质量
    static String[] weightsArr = {"500", "1000", "2000", "3000"};
    //ACC开关
    static String[] accArr = {"0", "1"};
    //是否定位
    static String[] locateArr = {"0", "1"};
    //车辆油路是否正常
    static String[] oilWayArr = {"0", "1"};
    //车辆电路是否正常
    static String[] electricArr = {"0", "1"};


    /**
     * @param url
     * @param msg
     * @return
     */
    public static String httpPost(String url, String msg) {
        String returnValue = "这是默认返回值，接口调用失败";
        CloseableHttpClient httpClient = HttpClients.createDefault();
        ResponseHandler<String> responseHandler = new BasicResponseHandler();
        try {
            //第一步：创建HttpClient对象
            httpClient = HttpClients.createDefault();
            //第二步：创建httpPost对象
            HttpPost httpPost = new HttpPost(url);
            //第三步：给httpPost设置JSON格式的参数
            StringEntity requestEntity = new StringEntity(msg, "utf-8");
            requestEntity.setContentEncoding("UTF-8");
            httpPost.setEntity(requestEntity);
            //第四步：发送HttpPost请求，获取返回值
            httpClient.execute(httpPost, responseHandler);

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                httpClient.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        //第五步：处理返回值
        return returnValue;
    }

    public static void main(String[] args) throws InterruptedException {
        String url = "http://hadoop5/log/lagou_bus_info";
        int n = 100;
        final Random rd = new Random();
        while (n > 0) {
            //拼接信息
            final StringBuilder sb = new StringBuilder();
            sb.append(deployArr[rd.nextInt(deployArr.length)]).append(",");
            sb.append(simArr[rd.nextInt(simArr.length)]).append(",");

sb.append(transpotNumArr[rd.nextInt(transpotNumArr.length)]).append(",");
            sb.append(plateNumArr[rd.nextInt(plateNumArr.length)]).append(",");
            sb.append(lglatArr[rd.nextInt(lglatArr.length)]).append(",");
            sb.append(speedArr[rd.nextInt(speedArr.length)]).append(",");

sb.append(directionArr[rd.nextInt(directionArr.length)]).append(",");
            sb.append(mileageArr[rd.nextInt(mileageArr.length)]).append(",");
            sb.append(timeStrArr[rd.nextInt(timeStrArr.length)]).append(",");

sb.append(oilRemainArr[rd.nextInt(oilRemainArr.length)]).append(",");
            sb.append(weightsArr[rd.nextInt(weightsArr.length)]).append(",");
            sb.append(accArr[rd.nextInt(accArr.length)]).append(",");
            sb.append(locateArr[rd.nextInt(locateArr.length)]).append(",");
            sb.append(oilWayArr[rd.nextInt(oilWayArr.length)]).append(",");
            sb.append(electricArr[rd.nextInt(electricArr.length)]);

            httpPost(url, sb.toString());
            TimeUnit.SECONDS.sleep(1);
            n--;
        }


    }
}
```

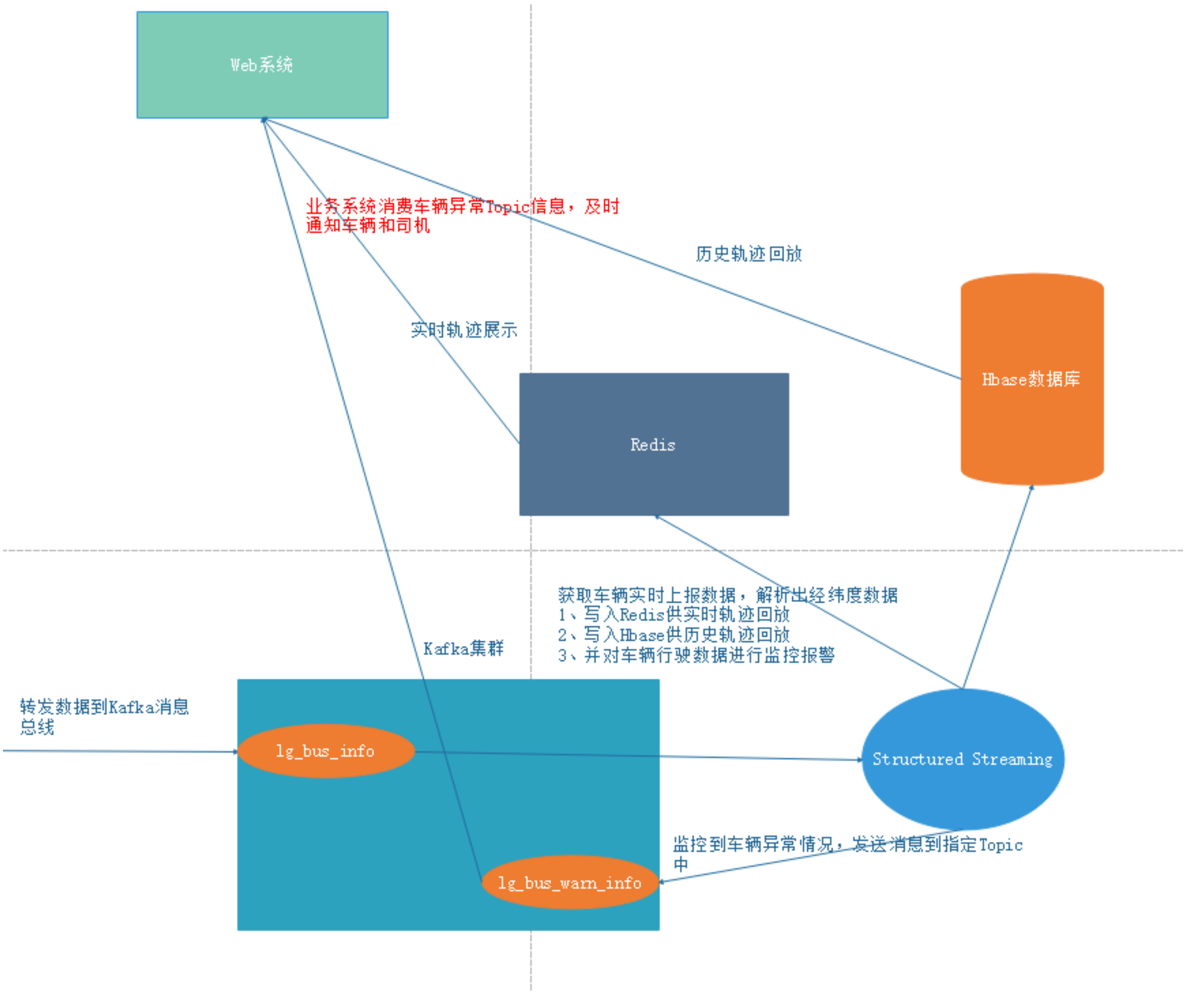
pom依赖

```
    <dependencies>
        <!--
https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->
        <dependency>
            <groupId>org.apache.httpcomponents</groupId>
            <artifactId>httpclient</artifactId>
            <version>4.5.10</version>
        </dependency>

    </dependencies>
```

验证Kafka端

```
kafka-console-consumer --bootstrap-server hadoop4:9092 --from-beginning --topic
bus_info
```

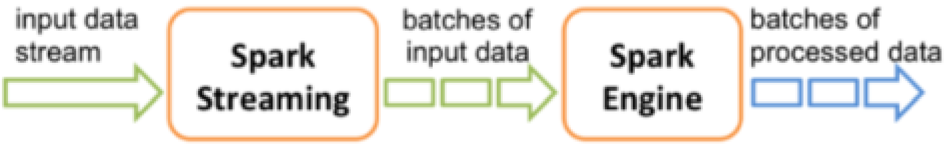
12 车辆监控

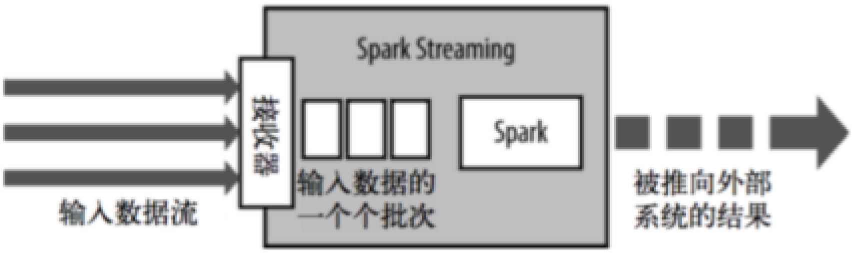

Web系统
业务系统消费车辆异常Topic信息，及时
通知车辆和司机
历史轨迹回放
实时轨迹展示
Redis
Hbase数据库
获取车辆实时上报数据，解析出经纬度数据
1、写入Redis供实时轨迹回放
2、写入Hbase供历史轨迹回放
3、并对车辆行驶数据进行监控报警
Kafka集群
转发数据到Kafka消息
总线
lg_bus_info
Structured Streaming
监控到车辆异常情况，发送消息到指定Topic
中
lg_bus_warn_info

第一节 Structured Streaming

1.1 Structured Streaming发展历史

1.1.1 Spark Streaming





在2.0之前，Spark Streaming作为核心API的扩展，针对实时数据流，提供了一套可扩展、高吞吐、可容错的流式计算模型。Spark Streaming会接收实时数据源的数据，并切分成很多小的batches，然后被Spark Engine执行，产出同样由很多小的batchs组成的结果流。本质上，这是一种**micro-batch（微批处理）的方式处理**，用批的思想去处理流数据.这种设计让Spark Streaming面对复杂的流式处理场景时捉襟见肘。

其实在流计算发展的初期，市面上主流的计算引擎本质上都只能处理特定的场景，

spark streaming这种构建在微批处理上的流计算引擎，**比较突出的问题就是处理延时较高（无法优化到秒以下的数量级），以及无法支持基于event_time的时间窗口做聚合逻辑。**

在这段时间，流式计算一直没有一套标准化、能应对各种场景的模型，直到2015年google发表了The Dataflow Model的论文。

https://yq.aliyun.com/articles/73255

1.1.2 Dataflow模型

在日常商业运营中，无边界、乱序、大规模数据集越来越普遍（例如，网站日志，手机应用统计，传感器网络）。同时，对这些数据的消费需求也越来越复杂，比如说按事件发生时间序列处理数据，按数据本身的特征进行窗口计算等等。同时人们也越来越苛求立刻得到数据分析结果。
作为数据工作者，**不能把无边界数据集（数据流）切分成有边界的数据，等待一个批次完整后处理。**

相反地，应该假设永远无法知道数据流是否终结，何时数据会变完整。唯一确信的是，新的数据会源源不断而来，老的数据可能会被撤销或更新。

由此，google工程师们提出了Dataflow模型，从根本上对从前的数据处理方法进行改进。

1、核心思想

对无边界，无序的数据源，允许按数据本身的特征进行窗口计算，得到基于事件发生时间的有序结果，并能在准确性、延迟程度和处理成本之间调整。

2、四个维度

抽象出四个相关的维度，通过灵活地组合来构建数据处理管道，以应对数据处理过程中的各种复杂的场景

what 需要计算什么

where 需要基于什么时间（事件发生时间）窗口做计算

when 在什么时间（系统处理时间）真正地触发计算

how 如何修正之前的计算结果

论文的大部分内容都是在说明如何通过这四个维度来应对各种数据处理场景。

3、相关概念

在现实场景中，从一个事件产生，到它被数据分析系统收集到，要经过非常复杂的链路，这本身就会存在一定的延时，还会因为一些特殊的情况加剧这种情况。比如基于移动端APP的用户行为数据，会因为手机信号较差、没有wifi等情况导致无法及时发送到服务端系统。面对这种时间上的偏移，数据处理模型如果只考虑处理时间，势必会降低最终结果的正确性。

●事件时间和处理时间

event_time，事件的实际发生时间
process_time，处理时间，是指一个事件被数据处理系统观察/接收到的时间

现在假设，你正在去往地下停车场的路上，并且打算用手机点一份外卖。选好了外卖后，你就用在线支付功能付款了，这个时候是12点05分。恰好这时，你走进了地下停车库，而这里并没有手机信号。因此外卖的在线支付并没有立刻成功，而支付系统一直在Retry重试“支付”这个操作。

当你找到自己的车并且开出地下停车场的时候，已经是12点15分了。这个时候手机重新有了信号，手机上的支付数据成功发到了外卖在线支付系统，支付完成。

在上面这个场景中你可以看到，支付数据的事件时间是12点05分，而支付数据的处理时间是12点15分

●窗口

除了一些无状态的计算逻辑（如过滤，映射等），经常需要把无边界的数据集切分成有限的数据片以便于后续聚合处理（比如统计最近5分钟的XX等），窗口就应用于这类逻辑中，常见的窗口包括：

sliding window，滑动窗口，除了窗口大小，还需要一个滑动周期，比如小时窗口，每5分钟滑动一次。

fixed window，固定窗口，按固定的窗口大小定义，比如每小时、天的统计逻辑。

固定窗口可以看做是滑动窗口的特例，即窗口大小和滑动周期相等。

sessions，会话窗口，以某一事件作为窗口起始，通常以时间定义窗口大小（也有可能是事件次数），发生在超时时间以内的事件都属于同一会话，比如统计用户启动APP之后一段时间的浏览信息等。

●总结

论文中远不止这些内容，还有很多编程模型的说明和举例，感兴趣的同学可以自行阅读。

https://yq.aliyun.com/articles/73255

除了论文，google还开源了Apache Beam项目，基本上就是对Dataflow模型的实现，目前已经成为Apache的顶级项目，但是在国内使用不多。国内使用的更多的是后面要学习的Flink，因为阿里大力推广Flink，甚至把花7亿元把Flink收购了.

1.2 Structured Streaming

1.2.1 介绍

官网地址

```
http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
```

也许是对Dataflow模型的借鉴，也许是英雄所见略同，spark在2.0版本中发布了新的流计算的API，Structured Streaming/结构化流。

- Structured Streaming是一个基于Spark SQL引擎的可扩展、容错的流处理引擎。**统一了流、批的编程模型**，你可以使用静态数据批处理一样的方式来编写流式计算操作。并且支持基于event_time的时间窗口的处理逻辑。
- Structured Streaming会以一种增量的方式来执行这些操作，并且持续更新结算结果。
    - 可以使用Scala、Java、Python或R中的DataSet / DataFrame API来表示流聚合、事件时间窗口、流到批连接等。此外，
    - Structured Streaming会通过checkpoint和预写日志等机制来实现Exactly-Once语义。

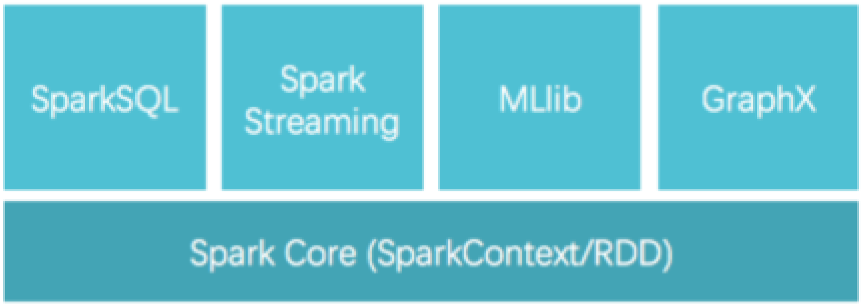
简单来说，对于开发人员来说，根本不用去考虑是流式计算，还是批处理，只要使用同样的方式来编写计算操作即可，Structured Streaming提供了快速、可扩展、容错、端到端的一次性流处理，而用户无需考虑更多细节

默认情况下，结构化流式查询使用微批处理引擎进行处理，该引擎将数据流作为一系列小批处理作业进行处理，从而实现端到端的延迟，最短可达100毫秒，并且完全可以保证一次容错。自Spark 2.3以来，引入了一种新的低延迟处理模式，称为连续处理，它可以在至少一次保证的情况下实现低至1毫秒的端到端延迟。也就是类似于 Flink 那样的实时流，而不是小批量处理。实际开发可以根据应用程序要求选择处理模式。

1.2.2 API

1.Spark Streaming 时代 -DStream-RDD

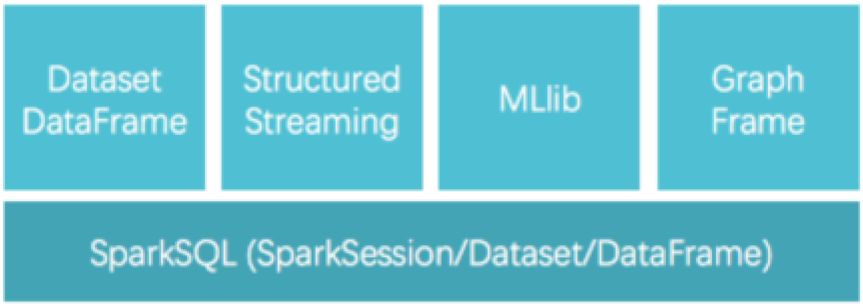
Spark Streaming 采用的数据抽象是DStream，而本质上就是时间上连续的RDD，对数据流的操作就是针对RDD的操作



2.Structured Streaming 时代 - DataSet/DataFrame -RDD

Structured Streaming是Spark2.0新增的可扩展和高容错性的实时计算框架，它构建于Spark SQL引擎，把流式计算也统一到DataFrame/Dataset里去了。

Structured Streaming 相比于 Spark Streaming 的进步就类似于 Dataset 相比于 RDD 的进步



1.2.3 编程模型

●编程模型概述

一个流的数据源从逻辑上来说就是一个不断增长的动态表格，随着时间的推移，新数据被持续不断地添加到表格的末尾。

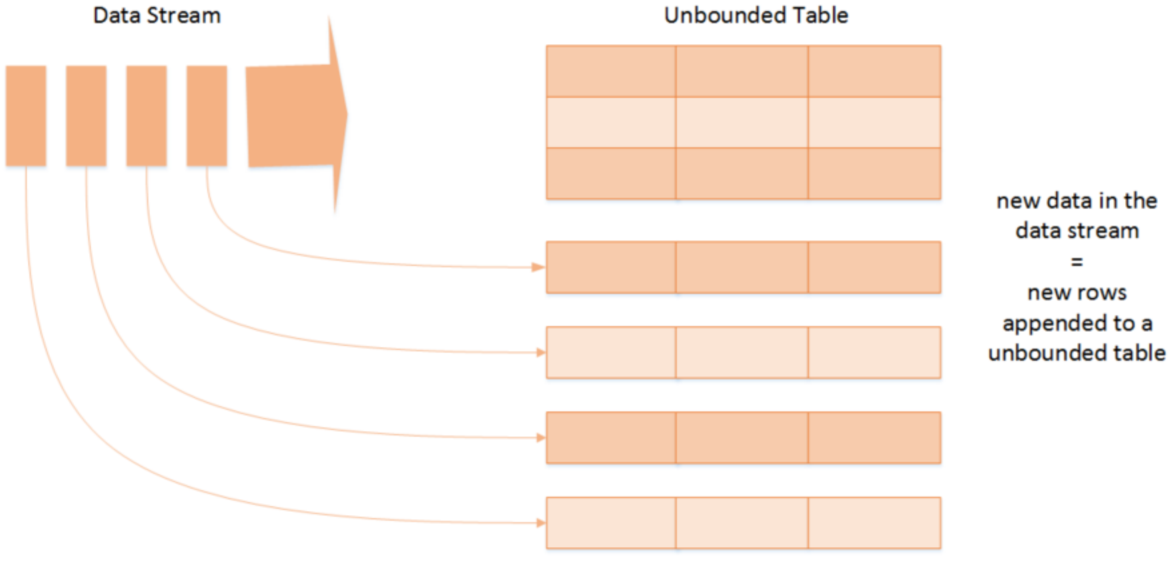
- 用户可以使用 Dataset/DataFrame 函数式API或者 SQL 来对这个动态数据源进行实时查询。每次查询在逻辑上就是对当前的表格内容执行一次 SQL 查询。
- 什么时候执行查询则是由用户通过触发器（Trigger）来设定时间(毫秒级)。用户既可以设定执行周期让查询尽可能快地执行，从而达到实时的效果也可以使用默认的触发。

一个流的输出有多种模式，

- 可以是基于整个输入执行查询后的完整结果，complete
- 也可以选择只输出与上次查询相比的差异，update
- 或者就是简单地追加最新的结果。append

这个模型对于熟悉 SQL 的用户来说很容易掌握，对流的查询跟查询一个表格几乎完全一样，十分简洁，易于理解

●核心思想


Data Stream
Unbounded Table
new data in the data stream = new rows appended to a unbounded table

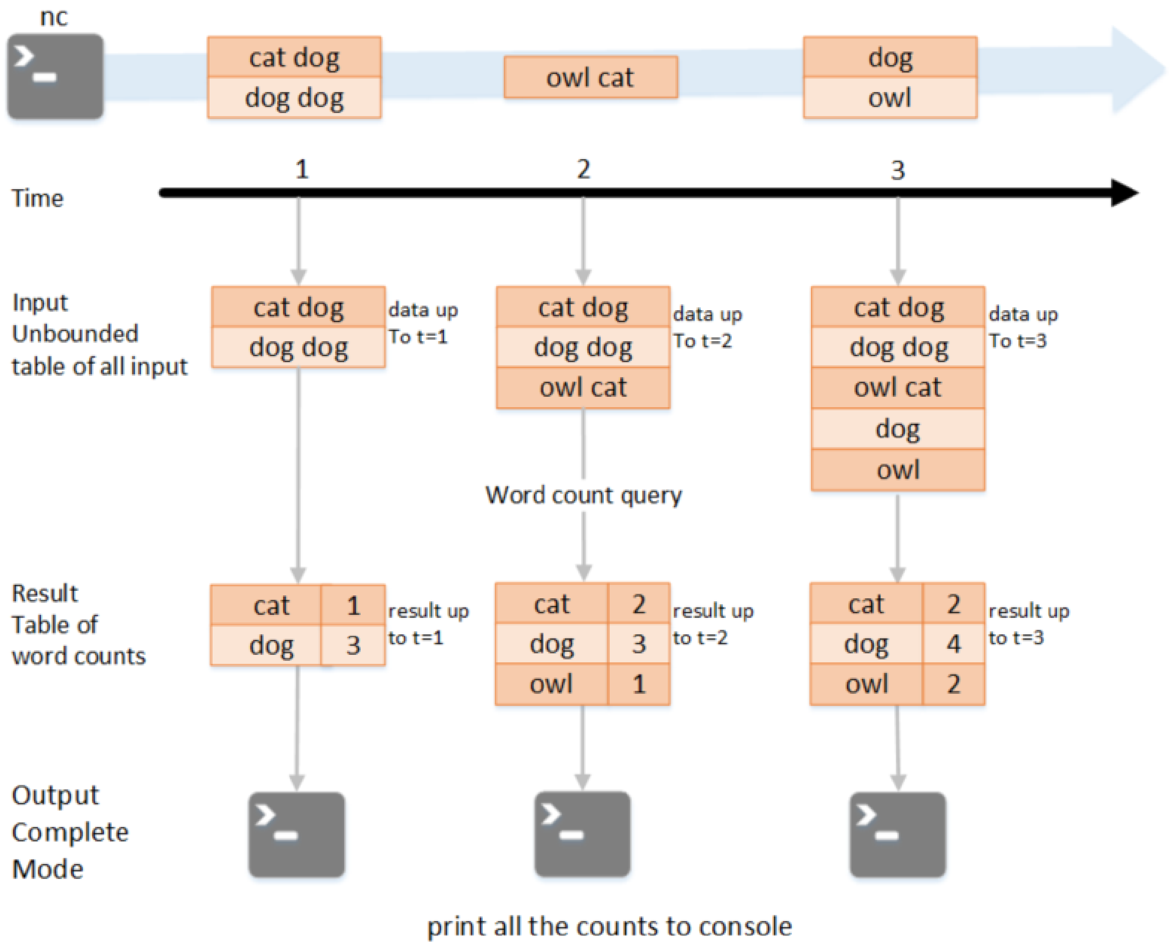
Data stream as an unbounded table

Structured Streaming最核心的思想就是将实时到达的数据看作是一个不断追加的unbound table无界表，到达流的每个数据项(RDD)就像是表中的一个新行被附加到无边界的表中.这样用户就可以用静态结构化数据的批处理查询方式进行流计算，如可以使用SQL对到来的每一行数据进行实时查询处理；

●应用场景

Structured Streaming将数据源映射为类似于关系数据库中的表，然后将经过计算得到的结果映射为另一张表，完全以结构化的方式去操作流式数据，这种编程模型非常有利于处理分析结构化的实时数据；

●WordCount图解



Model of the Quick Example

如图所示，

第一行表示从socket不断接收数据，

第二行是时间轴，表示每隔1秒进行一次数据处理，

第三行可以看成是之前提到的“unbound table",

第四行为最终的wordCounts是结果集。

当有新的数据到达时，Spark会执行“增量"查询，并更新结果集；

该示例设置为Complete Mode，因此每次都将所有数据输出到控制台；

1.在第1秒时，此时到达的数据为"cat dog"和"dog dog"，因此我们可以得到第1秒时的结果集cat=1 dog=3，并输出到控制台；
2.当第2秒时，到达的数据为"owl cat"，此时"unbound table"增加了一行数据"owl cat"，执行word count查询并更新结果集，可得第2秒时的结果集为cat=2 dog=3 owl=1，并输出到控制台；
3.当第3秒时，到达的数据为"dog"和"owl"，此时"unbound table"增加两行数据"dog"和"owl"，执行word count查询并更新结果集，可得第3秒时的结果集为cat=2 dog=4 owl=2；

1.3 Structured Streaming

1.3.1 Source

Socket source (for testing): 从socket连接中读取文本内容。

Kafka source: 从Kafka中拉取数据,与0.10或以上的版本兼容，后面单独整合Kafka

- Socket

```
yum install -y nc
nc -lk 9999
```

```
object WordCount {
  def main(args: Array[String]): Unit = {
    //1.创建SparkSession,因为StructuredStreaming的数据模型也是DataFrame/DataSet
    val spark: SparkSession =
SparkSession.builder().master("local[*]").appName("SparkSQL").getOrCreate()
    val sc: SparkContext = spark.sparkContext
    sc.setLogLevel("WARN")
    //2.接收数据
    val dataDF: DataFrame = spark.readStream
      .option("host", "node01")
      .option("port", 9999)
      .format("socket")
      .load()
    //3.处理数据
    import spark.implicits._
    val dataDS: Dataset[String] = dataDF.as[String]
    val wordDS: Dataset[String] = dataDS.flatMap(_.split(" "))
    val result: Dataset[Row] =
wordDS.groupBy("value").count().sort($"count".desc)
    //result.show()
    //Queries with streaming sources must be executed with
writeStream.start();
    result.writeStream
      .format("console")//往控制台写
      .outputMode("complete")//每次将所有的数据写出
    .trigger(Trigger.ProcessingTime(0))//触发时间间隔
      //.option("checkpointLocation","./ckp")//设置checkpoint目录,socket不支持
数据恢复,所以第二次启动会报错,需要注掉
    .start()//开启
      .awaitTermination()//等待停止
  }
}
```

整合Kafka单独讲解。

```
<!-- 指定仓库位置，依次为aliyun、cloudera-->
    <repositories>
        <repository>
            <id>aliyun</id>
            <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        </repository>
        <repository>
            <id>cloudera</id>
            <url>https://repository.cloudera.com/artifactory/cloudera-
repos/</url>
```

```
        </repository>

    </repositories>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <encoding>UTF-8</encoding>
        <scala.version>2.11.8</scala.version>
        <scala.compat.version>2.11</scala.compat.version>
        <spark.version>2.4.0</spark.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.scala-lang</groupId>
            <artifactId>scala-library</artifactId>
            <version>${scala.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.11</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_2.11</artifactId>
            <version>${spark.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql-kafka-0-10_2.11</artifactId>
            <version>${spark.version}</version>
        </dependency>

        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>2.6.0-mr1-cdh5.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-client</artifactId>
            <version>1.2.0-cdh5.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-server</artifactId>
            <version>1.2.0-cdh5.14.0</version>
        </dependency>

    </dependencies>

    <build>
        <sourceDirectory>src/main/scala</sourceDirectory>
        <plugins>
            <!-- 指定编译java的插件 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
            </plugin>
            <!-- 指定编译scala的插件 -->
            <plugin>
                <groupId>net.alchim31.maven</groupId>
                <artifactId>scala-maven-plugin</artifactId>
                <version>3.2.2</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>compile</goal>
                            <goal>testCompile</goal>
                        </goals>
                        <configuration>
                            <args>
                                <arg>-dependencyfile</arg>

<arg>${project.build.directory}/.scala_dependencies</arg>
                            </args>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.18.1</version>
                <configuration>
                    <useFile>false</useFile>
                    <disableXmlReport>true</disableXmlReport>
                    <includes>
                        <include>**/*Test.*</include>
                        <include>**/*Suite.*</include>
                    </includes>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>2.3</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <filters>
                                <filter>
                                    <artifact>*:*</artifact>
                                    <excludes>
                                        <exclude>META-INF/*.SF</exclude>
                                        <exclude>META-INF/*.DSA</exclude>
                                        <exclude>META-INF/*.RSA</exclude>
                                    </excludes>
                                </filter>
                            </filters>
                            <transformers>
                                <transformer

implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTran
sformer">
                                    <mainClass></mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

1.3.2 计算

获得到Source之后的基本数据处理方式和之前学习的DataFrame、DataSet一致，不再赘述

官网示例代码

```
case class DeviceData(device: String, deviceType: String, signal: Double, time:
DateTime)
val df: DataFrame = ...
val ds: Dataset[DeviceData] = df.as[DeviceData]
// Select the devices which have signal more than 10
df.select("device").where("signal > 10")      // using untyped APIs
ds.filter(_.signal > 10).map(_.device)         // using typed APIs
// Running count of the number of updates for each device type
df.groupBy("deviceType").count()                 // using untyped API
// Running average signal for each device type
import org.apache.spark.sql.expressions.scalalang.typed
ds.groupByKey(_.deviceType).agg(typed.avg(_.signal))    // using typed API
```

1.3.3 输出

计算结果可以选择输出到多种设备并进行如下设定

1.output mode：以哪种方式将result table的数据写入sink

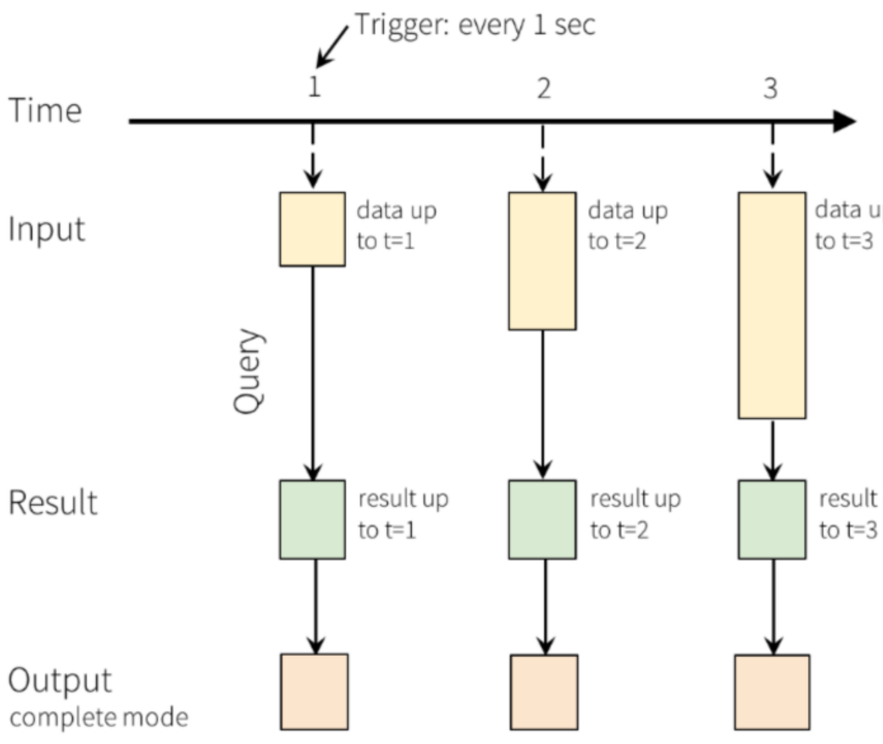
2.format/output sink的一些细节：数据格式、位置等。

3.query name：指定查询的标识。类似tempview的名字

4.trigger interval：触发间隔，如果不指定，默认会尽可能快速地处理数据

5.checkpoint地址：一般是hdfs上的目录。注意：Socket不支持数据恢复，如果设置了，第二次启动会报错 ,Kafka支持

1、output mode



Programming Model for Structured Streaming

每当结果表更新时，我们都希望将更改后的结果行写入外部接收器。

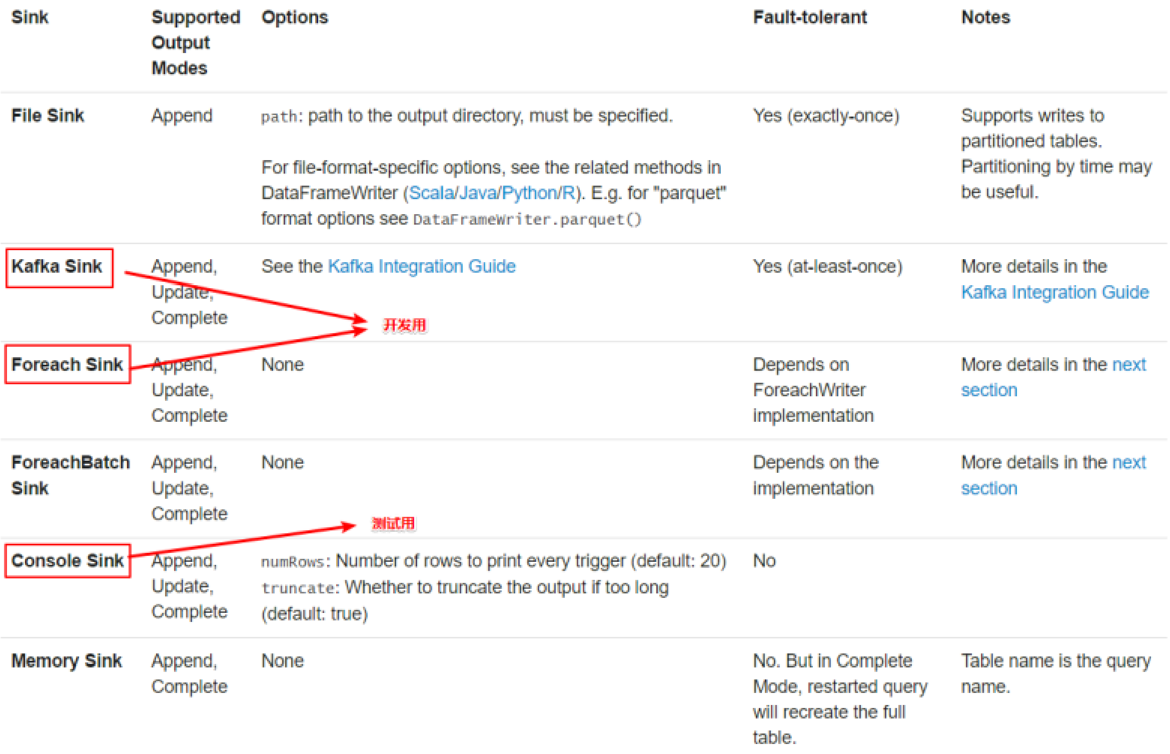
这里有三种输出模型:

1.Append mode:默认模式，新增的行才输出，每次更新结果集时，只将新添加到结果集的结果行输出到接收器。仅支持那些添加到结果表中的行永远不会更改的查询。因此，此模式保证每行仅输出一次。例如，仅查询select，where，map，flatMap，filter，join等会支持追加模式。不支持聚合

2.Complete mode: 所有内容都输出，每次触发后，整个结果表将输出到接收器。聚合查询支持此功能。仅适用于包含聚合操作的查询。

3.Update mode:更新的行才输出，每次更新结果集时，仅将被更新的结果行输出到接收器(自Spark 2.1.1起可用)，不支持排序

2、output sink



Sink	Supported Output Modes	Options	Fault-tolerant	Notes
File Sink	Append	`path`: path to the output directory, must be specified. For file-format-specific options, see the related methods in DataFrameWriter (Scala/Java/Python/R). E.g. for "parquet" format options see `DataFrameWriter.parquet()`	Yes (exactly-once)	Supports writes to partitioned tables. Partitioning by time may be useful.
Kafka Sink	Append, Update, Complete	See the Kafka Integration Guide	Yes (at-least-once)	More details in the Kafka Integration Guide
Foreach Sink	Append, Update, Complete	None	Depends on ForeachWriter implementation	More details in the next section
ForeachBatch Sink	Append, Update, Complete	None	Depends on the implementation	More details in the next section
Console Sink	Append, Update, Complete	`numRows`: Number of rows to print every trigger (default: 20) `truncate`: Whether to truncate the output if too long (default: true)	No	
Memory Sink	Append, Complete	None	No. But in Complete Mode, restarted query will recreate the full table.	Table name is the query name.

File sink - Stores the output to a directory.支持parquet文件,以及append模式

```
writeStream

   .format("parquet")     **// can be "orc", "json", "csv", etc.**

   .option("path", "path/to/destination/dir")

   .start()
```

Kafka sink - Stores the output to one or more topics in Kafka.

```
writeStream

   .format("kafka")

   .option("kafka.bootstrap.servers", "host1:port1,host2:port2")

   .option("topic", "updates")

   .start()
```

Foreach sink - Runs arbitrary computation on the records in the output. See later in the section for more details.

```
writeStream
   .foreach(...)
   .start()
```

Console sink (for debugging) - Prints the output to the console/stdout every time there is a trigger. Both, Append and Complete output modes, are supported. This should be used for debugging purposes on low data volumes as the entire output is collected and stored in the driver’s memory after every trigger.

```
writeStream
  .format("console")
  .start()
```

Memory sink (for debugging) - The output is stored in memory as an in-memory table. Both, Append and Complete output modes, are supported. This should be used for debugging purposes on low data volumes as the entire output is collected and stored in the driver’s memory. Hence, use it with caution.

```
writeStream
  .format("memory")
  .queryName("tableName")
  .start()
```

●官网示例代码

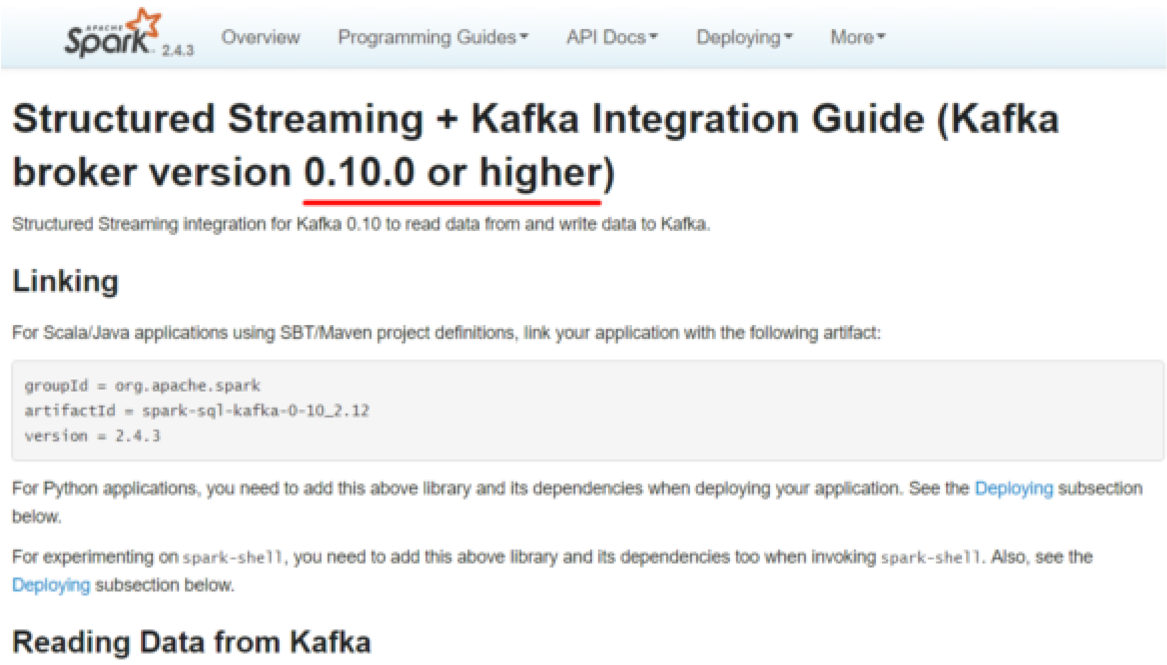
```
// ========== DF with no aggregations ==========
val noAggDF = deviceDataDf.select("device").where("signal > 10")
// Print new data to console
noAggDF
  .writeStream
  .format("console")
  .start()
// Write new data to Parquet files
noAggDF
  .writeStream
  .format("parquet")
  .option("checkpointLocation", "path/to/checkpoint/dir")
  .option("path", "path/to/destination/dir")
  .start()
// ========== DF with aggregation ==========
val aggDF = df.groupBy("device").count()
// Print updated aggregations to console
aggDF
  .writeStream
  .outputMode("complete")
  .format("console")
  .start()
// Have all the aggregates in an in-memory table
aggDF
  .writeStream
  .queryName("aggregates")    // this query name will be the table name
  .outputMode("complete")
  .format("memory")
  .start()
```

```
spark.sql("select * from aggregates").show()   // interactively query in-memory
table
```

1.4 Structured Streaming整合Kafka

1.4.1 官网介绍

```
http://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html
```



Structured Streaming + Kafka Integration Guide (Kafka broker version 0.10.0 or higher)

Structured Streaming integration for Kafka 0.10 to read data from and write data to Kafka.

Linking

For Scala/Java applications using SBT/Maven project definitions, link your application with the following artifact:

```
groupId = org.apache.spark
artifactId = spark-sql-kafka-0-10_2.12
version = 2.4.3
```

For Python applications, you need to add this above library and its dependencies when deploying your application. See the Deploying subsection below.

For experimenting on `spark-shell`, you need to add this above library and its dependencies too when invoking `spark-shell`. Also, see the Deploying subsection below.

Reading Data from Kafka

●Creating a Kafka Source for Streaming Queries

```
// Subscribe to 1 topic
val df = spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
  .as[(String, String)]
// Subscribe to multiple topics
val df = spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1,topic2")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
  .as[(String, String)]
// Subscribe to a pattern
val df = spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribePattern", "topic.*")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

```
  .as[(String, String)]
```

●Creating a Kafka Source for Batch Queries

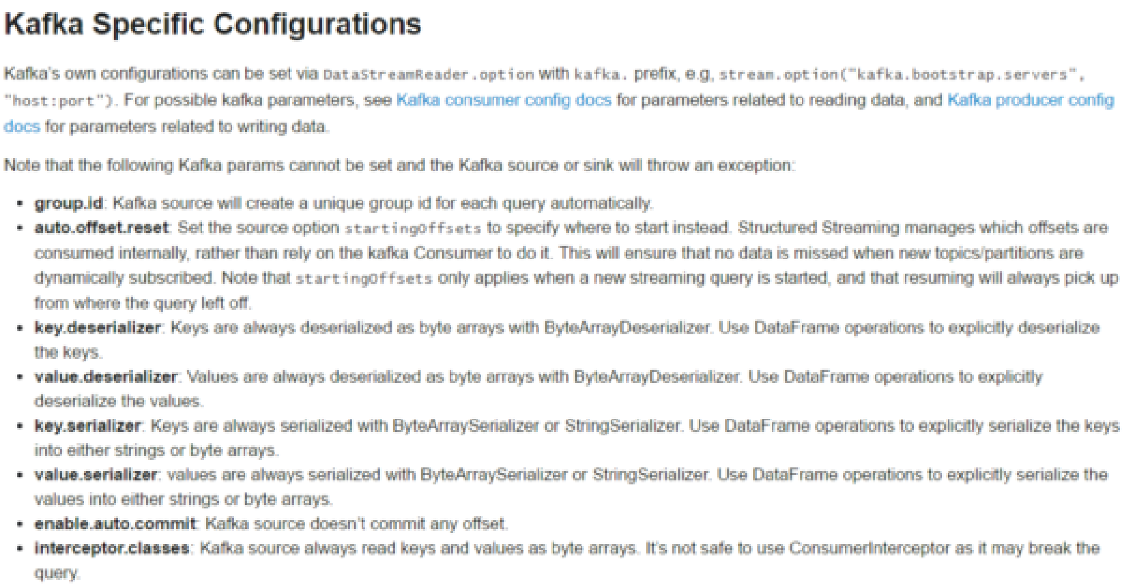
```
// Subscribe to 1 topic defaults to the earliest and latest offsets
val df = spark
  .read
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1")
  .load()
df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
  .as[(String, String)]
// Subscribe to multiple topics, specifying explicit Kafka offsets
val df = spark
  .read
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1,topic2")
  .option("startingOffsets", """{"topic1":{"0":23,"1":-2},"topic2":{"0":-2}}""")
  .option("endingOffsets", """{"topic1":{"0":50,"1":-1},"topic2":{"0":-1}}""")
  .load()df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
  .as[(String, String)]
// Subscribe to a pattern, at the earliest and latest offsets
val df = spark
  .read
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribePattern", "topic.*")
  .option("startingOffsets", "earliest")
  .option("endingOffsets", "latest")
  .load()df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
  .as[(String, String)]
```

●注意：读取后的数据的Schema是固定的，包含的列如下：

Column	Type	说明
key	binary	消息的key
value	binary	消息的value
topic	string	主题
partition	int	分区
offset	long	偏移量
timestamp	long	时间戳
timestampType	int	类型

●注意：下面的参数是不能被设置的，否则kafka会抛出异常：

- group.id:kafka的source会在每次query的时候自定创建唯一的group id
- auto.offset.reset :为了避免每次手动设置startingoffsets的值，structured streaming在内部消费时会自动管理offset。这样就能保证订阅动态的topic时不会丢失数据。startingOffsets在流处理时，只会作用于第一次启动时，之后的处理都会自动的读取保存的offset。
- key.deserializer，value.deserializer，key.serializer，value.serializer 序列化与反序列化，都是ByteArraySerializer
- enable.auto.commit:Kafka源不支持提交任何偏移量



Kafka Specific Configurations

Kafka's own configurations can be set via `DataStreamReader.option` with `kafka.` prefix, e.g, `stream.option("kafka.bootstrap.servers", "host:port")`. For possible kafka parameters, see Kafka consumer config docs for parameters related to reading data, and Kafka producer config docs for parameters related to writing data.

Note that the following Kafka params cannot be set and the Kafka source or sink will throw an exception:

- **group.id**: Kafka source will create a unique group id for each query automatically.
- **auto.offset.reset**: Set the source option `startingOffsets` to specify where to start instead. Structured Streaming manages which offsets are consumed internally, rather than rely on the kafka Consumer to do it. This will ensure that no data is missed when new topics/partitions are dynamically subscribed. Note that `startingOffsets` only applies when a new streaming query is started, and that resuming will always pick up from where the query left off.
- **key.deserializer**: Keys are always deserialized as byte arrays with ByteArrayDeserializer. Use DataFrame operations to explicitly deserialize the keys.
- **value.deserializer**: Values are always deserialized as byte arrays with ByteArrayDeserializer. Use DataFrame operations to explicitly deserialize the values.
- **key.serializer**: Keys are always serialized with ByteArraySerializer or StringSerializer. Use DataFrame operations to explicitly serialize the keys into either strings or byte arrays.
- **value.serializer**: values are always serialized with ByteArraySerializer or StringSerializer. Use DataFrame operations to explicitly serialize the values into either strings or byte arrays.
- **enable.auto.commit**: Kafka source doesn't commit any offset.
- **interceptor.classes**: Kafka source always read keys and values as byte arrays. It's not safe to use ConsumerInterceptor as it may break the query.

1.4.2 准备工作

启动kafka

向topic中生产数据

代码实现

```
package com.lg.test

import org.apache.spark.sql.{DataFrame, Dataset, Row, SparkSession}

/**
  * 使用结构化流读取kafka中的数据
  */
object StructuredKafka {
  def main(args: Array[String]): Unit = {
    //1 获取sparksession
    val spark: SparkSession =
SparkSession.builder().master("local[*]").appName(StructuredKafka.getClass.getNa
me)
      .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    import spark.implicits._
    //2 定义读取kafka数据源
    val kafkaDf: DataFrame = spark.readStream
      .format("kafka")
      .option("kafka.bootstrap.servers", "hadoop4:9092")
      .option("subscribe", "spark_kafka")
      .load()
    //3 处理数据
    val kafkaValDf: DataFrame = kafkaDf.selectExpr("CAST(value AS STRING)")
    //转为ds
    val kafkaDs: Dataset[String] = kafkaValDf.as[String]
    val kafkaWordDs: Dataset[String] = kafkaDs.flatMap(_.split(" "))
    //执行聚合
    val res: Dataset[Row] =
kafkaWordDs.groupBy("value").count().sort($"count".desc)
    //4 输出
    res.writeStream
      .format("console")
      .outputMode("complete")
      .start()
      .awaitTermination()
  }
}
```

第二节 轨迹数据处理

使用结构化流消费kafka中数据，解析经纬度数据，分别写入Redis支持实时轨迹查询，写入Hbase中支持历史轨迹回放。

2.1 安装Redis

1、下载redis安装包

```
hadoop5服务器执行以下命令下载redis安装包
mkdir -p /opt/lagou/softwares
mkdir -p /opt/lagou/servers
cd /opt/lagou/softwares
wget http://download.redis.io/releases/redis-3.2.8.tar.gz
```

2、解压redis压缩包

```
cd /opt/lagou/softwares
tar -zxvf redis-3.2.8.tar.gz -C ../servers/
```

3、安装运行环境

```
yum -y install gcc-c++
yum  -y  install  tcl
```

4、编译安装

```
cd /opt/lagou/servers/redis-3.2.8/
yum reinstall binutils -y
make MALLOC=libc
make  && make install
```

5、修改redis配置文件

```
修改redis配置文件

cd /opt/lagou/servers/redis-3.2.8/
mkdir -p /opt/lagou/servers/redis-3.2.8/logs
mkdir -p /opt/lagou/servers/redis-3.2.8/redisdata

vim redis.conf
bind hadoop5
daemonize yes
pidfile /var/run/redis_6379.pid
logfile "/opt/lagou/servers/redis-3.2.8/logs/redis.log"
dir /opt/lagou/servers/redis-3.2.8/redisdata
```

6、启动redis

```
启动redis
cd /opt/lagou/servers/redis-3.2.8/src
redis-server  ../redis.conf
```

7、连接redis客户端

```
cd /opt/lagou/servers/redis-3.2.8/src
redis-cli -h hadoop5
```

2.2 轨迹数据写入Redis

```
<!-- https://mvnrepository.com/artifact/redis.clients/jedis -->
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>3.2.0</version>
</dependency>

```

RedisWriter

```
package com.lg.test

import org.apache.spark.sql.ForeachWriter
import redis.clients.jedis.{Jedis, JedisPool, JedisPoolConfig}

object RedisWriter {
  //连接配置
  val config = new JedisPoolConfig
  //最大连接数
  config.setMaxTotal(20)
  //最大空闲连接数
  config.setMaxIdle(10)
  //设置连接池属性分别有： 配置  主机名   端口号  连接超时时间
  val pool = new JedisPool(config, "hadoop1", 6379, 10000)

  //连接池
  def getConnection(): Jedis = {
    pool.getResource
  }
}

class RedisWriter extends ForeachWriter[BusInfo] {
  var jedisPool: JedisPool = null
  var jedis: Jedis = null

  override def open(partitionId: Long, epochId: Long): Boolean = {
    jedis = RedisWriter.getConnection()
    true;
  }

  override def process(value: BusInfo): Unit = {
    val lglat: String = value.lglat
    val deployNum: String = value.deployNum
    jedis.set(deployNum, lglat);
  }

  override def close(errorOrNull: Throwable): Unit = {
    jedis.close()
  }
}
```

```
package com.lg.test

import org.apache.spark.sql._

/**
  * 使用结构化流读取kafka中的数据
  */

case class BusInfo(
                    deployNum: String,
                    simNum: String,
                    transpotNum: String,
                    plateNum: String,
                    lglat: String,
                    speed: String,
                    direction: String,
                    mileage: String,
                    timeStr: String,
                    oilRemain: String,
                    weight: String,
                    acc: String,
                    locate: String,
                    oilWay: String,
                    electric: String

                  )

object BusInfo {
  def apply(
             msg: String
           ): BusInfo = {
    val arr: Array[String] = msg.split(",")
    new BusInfo(
      arr(0),
      arr(1),
      arr(2),
      arr(3),
      arr(4),
      arr(5),
      arr(6),
      arr(7),
      arr(8),
      arr(9),
      arr(10),
      arr(11),
      arr(12),
      arr(13),
      arr(14)
    )
  }
}

object StructuredKafka1 {
  def main(args: Array[String]): Unit = {
    //1 获取sparksession
    val spark: SparkSession = SparkSession.builder()
      .master("local[*]")
      .appName("kafka-redis")
      .getOrCreate()
    val sc = spark.sparkContext
    sc.setLogLevel("WARN")
    import spark.implicits._
    //2 定义读取kafka数据源
    val kafkaDf: DataFrame = spark.readStream

      .format("kafka")
      .option("kafka.bootstrap.servers", "hadoop4:9092")
      .option("subscribe", "lagou_bus_info")
      .load()
    //3 处理数据
    val kafkaValDf: DataFrame = kafkaDf.selectExpr("CAST(value AS STRING)")
    //转为ds
    val kafkaDs: Dataset[String] = kafkaValDf.as[String]
    val busInfoDs: Dataset[BusInfo] = kafkaDs.map(msg => {
      BusInfo(msg)
    })

    val writer = new RedisWriter
    //4 输出,写出数据到redis和hbase
    busInfoDs.writeStream
      .foreach(
        writer
      )
      .outputMode("append")
      .start()
      .awaitTermination()
  }
```

```
}
```

2.3 轨迹数据写入Hbase

创建表

```
create 'htb_gps','car_info'
```

```
<dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>2.6.0-mr1-cdh5.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-client</artifactId>
            <version>1.2.0-cdh5.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-server</artifactId>
            <version>1.2.0-cdh5.14.0</version>
        </dependency>
```

```
import org.apache.hadoop.hbase.{HBaseConfiguration, TableName}
import org.apache.hadoop.hbase.client.{Connection, ConnectionFactory, Put,
Table}
import org.apache.hadoop.hbase.util.Bytes
import org.apache.spark.sql.ForeachWriter

object HbaseWriter {
  var connect: Connection = _
  var table: Table = _

  def getHbaseTable() = {
    if (table != null) {
      table;
    } else {
      if (connect != null) {
        table = connect.getTable(TableName.valueOf("htb_gps"))
        table
      } else {
        val conf = HBaseConfiguration.create
        conf.set("hbase.zookeeper.property.clientPort", "2181")
        conf.set("hbase.zookeeper.quorum", "hadoop3,hadoop4")
        connect = ConnectionFactory.createConnection(conf)
        table = connect.getTable(TableName.valueOf("htb_gps"))
        table
      }
    }

  }
}
```

```
class HbaseWriter extends ForeachWriter[BusInfo] {
  var connect: Connection = _
  var table: Table = _

  override def open(partitionId: Long, epochId: Long): Boolean = {

    table = HbaseWriter.getHbaseTable()
    true;
  }

  override def process(info: BusInfo): Unit = {

    //Rowkey：行程唯一编号+车牌编号+timestamp
    //列族：car_info
    //列：数据的全部字段
    var rowkey = info.deployNum + info.plateNum + info.timeStr
    val put = new Put(Bytes.toBytes(rowkey))
    val lglat: String = info.lglat
    val arr: Array[String] = lglat.split("_")
    put.addColumn(
      Bytes.toBytes("car_info"),
      Bytes.toBytes("lng"),
      Bytes.toBytes(arr(0)))

    put.addColumn(
      Bytes.toBytes("car_info"),
      Bytes.toBytes("lat"),
      Bytes.toBytes(arr(1)))
    table.put(put)
  }

  override def close(errorOrNull: Throwable): Unit = {
    table.close()

  }
}
```

第三节 异常检测

监听剩余油量小于百分之三十的运输车辆

写入Kafka

```
val oilWarnDs: Dataset[BusInfo] = busInfoDs.filter(info => {
      val oilRemain: Int = info.oilRemain.toInt
      oilRemain < 30
    })


//写出到kafka
    oilWarnDs.withColumn("value", new Column("deployNum"))
      .writeStream
      .format("kafka")
      .option("kafka.bootstrap.servers",
"hadoop2:9092,hadoop3:9092,hadoop4:9092")
      .option("topic", "lagou_bus_warn_info")
      .option("checkpointLocation", "./ssc")
      .start()
```

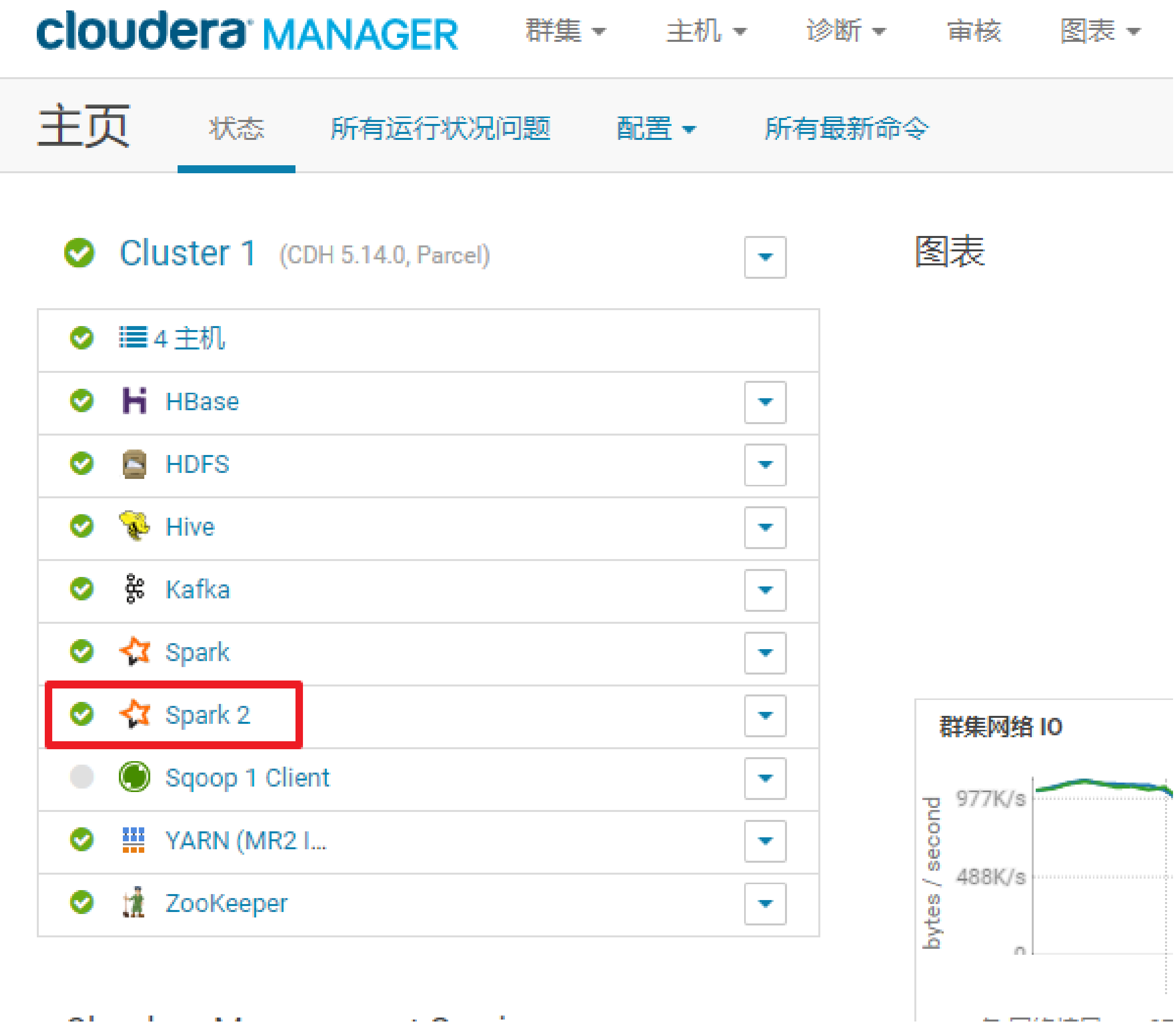
提交任务

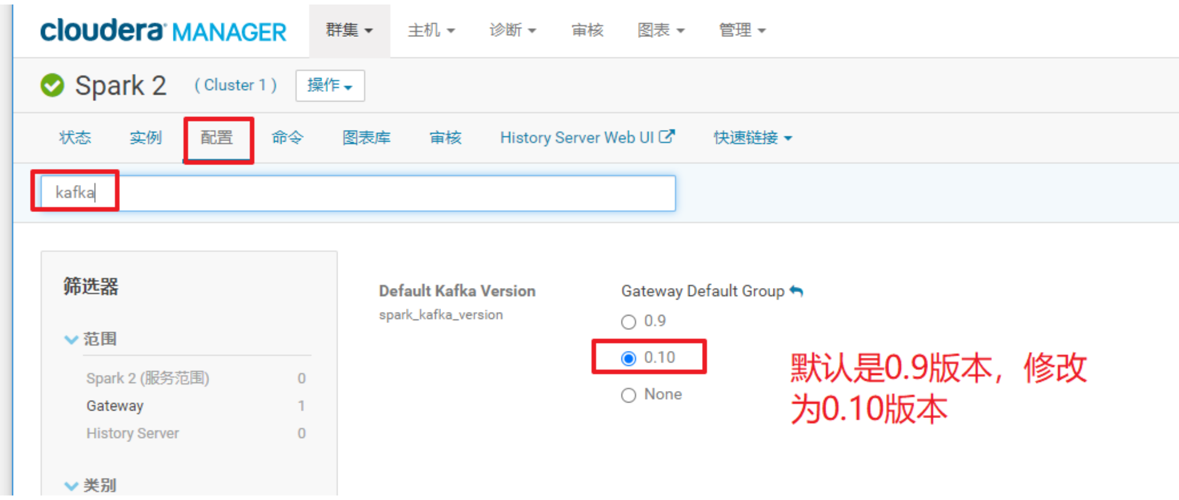
报错信息

```

User class threw exception:
com.fasterxml.jackson.databind.exc.MismatchedInputException: No content to map
due to end-of-input
删除之前运行的ck目录（修改了代码之前目录的数据无法匹配所以报错）
```

```
spark2-submit --class com.lg.monitor.RealTimeProcess --master yarn --deploy-mode
cluster --executor-memory 1G --num-executors 5 /root/monitor/bus_monitor.jar;
```

注意：1、需要修改CDH中spark2依赖的Kafka版本





kafka

Default Kafka Version
spark_kafka_version

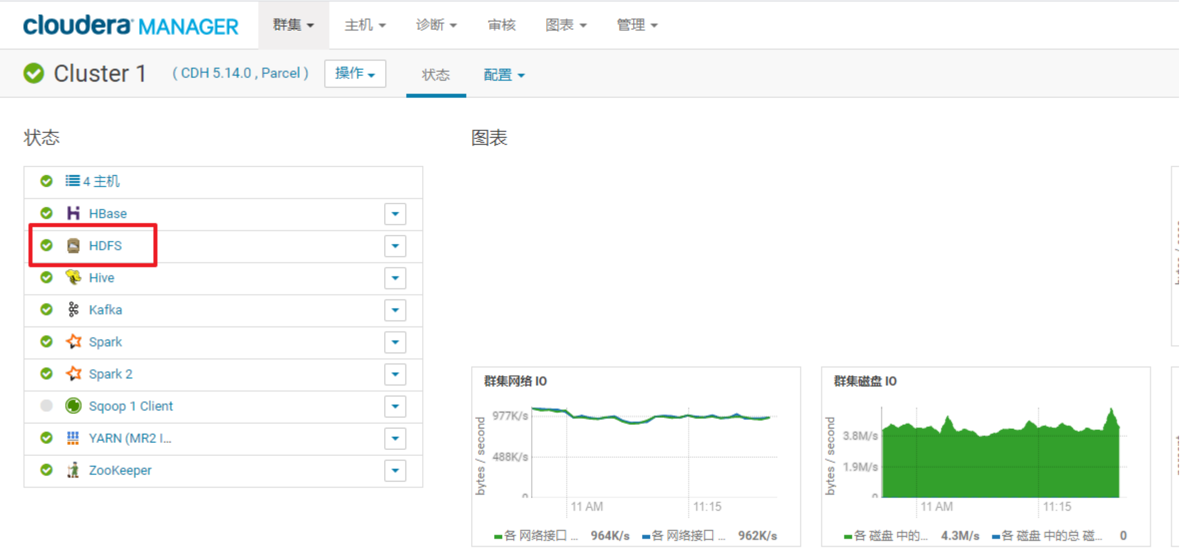
Gateway Default Group

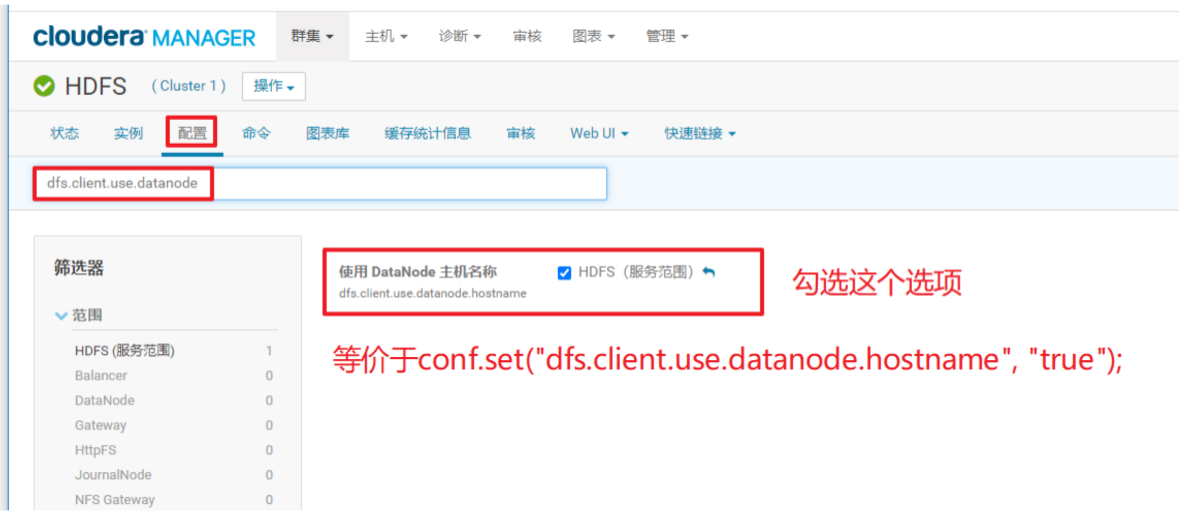
○ 0.9
● 0.10
○ None

默认是0.9版本，修改为0.10版本

2、如果ck目录选择HDFS

可以修改HDFS返回主机名称而不是ip地址，可以在直接更改集群的配置，然后把hdfs-site.xml引入工程中





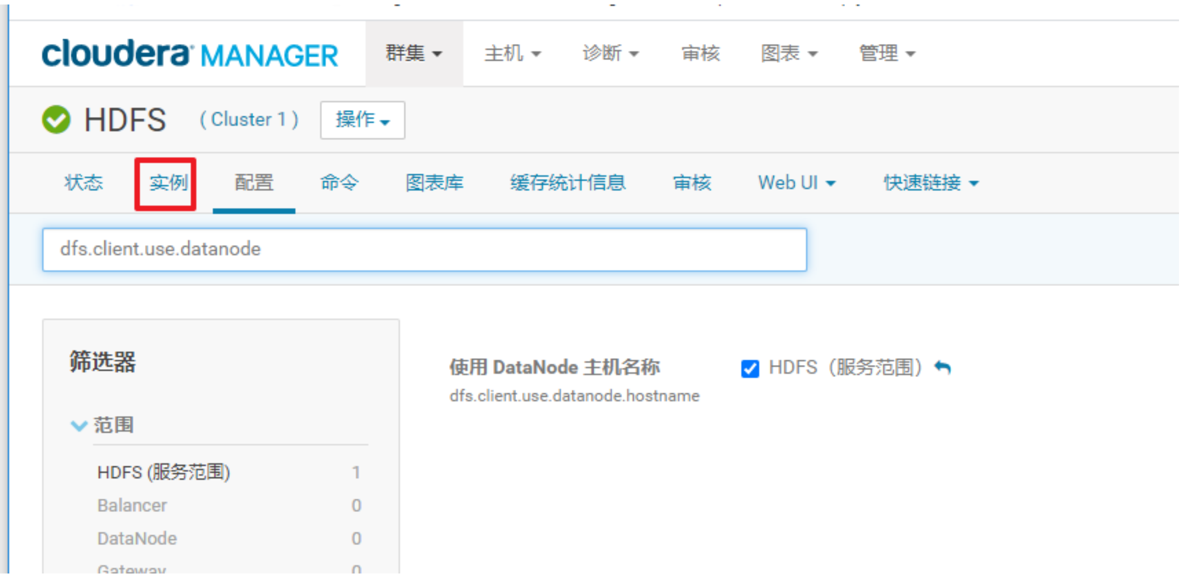
dfs.client.use.datanode

使用 DataNode 主机名称
dfs.client.use.datanode.hostname
☑ HDFS（服务范围）

勾选这个选项

等价于conf.set("dfs.client.use.datanode.hostname", "true");

重新导出hdfs-site.xml



dfs.client.use.datanode

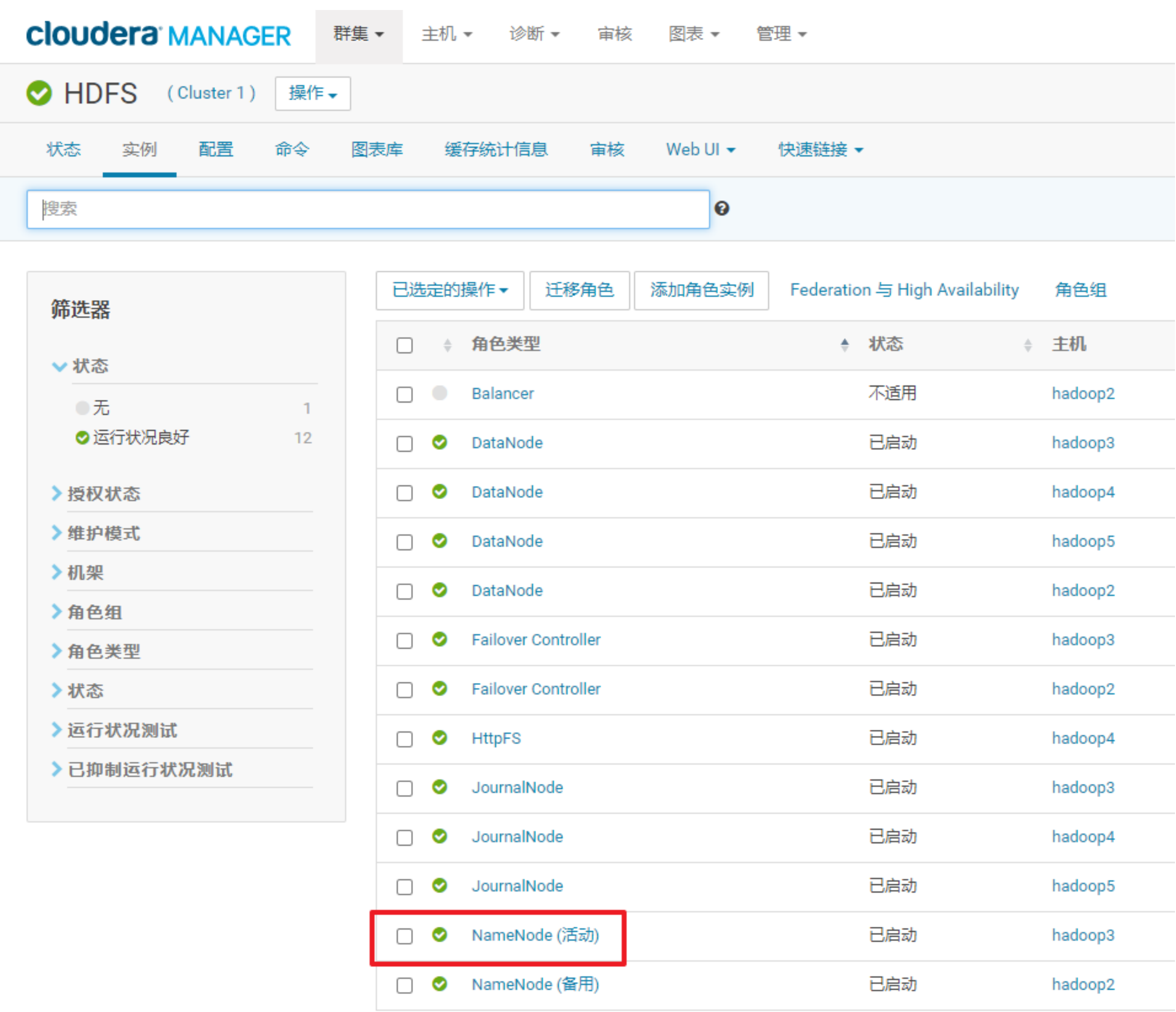
筛选器

范围

HDFS (服务范围)	1
Balancer	0
DataNode	0
Gateway	0

使用 DataNode 主机名称
dfs.client.use.datanode.hostname

☑ HDFS（服务范围）



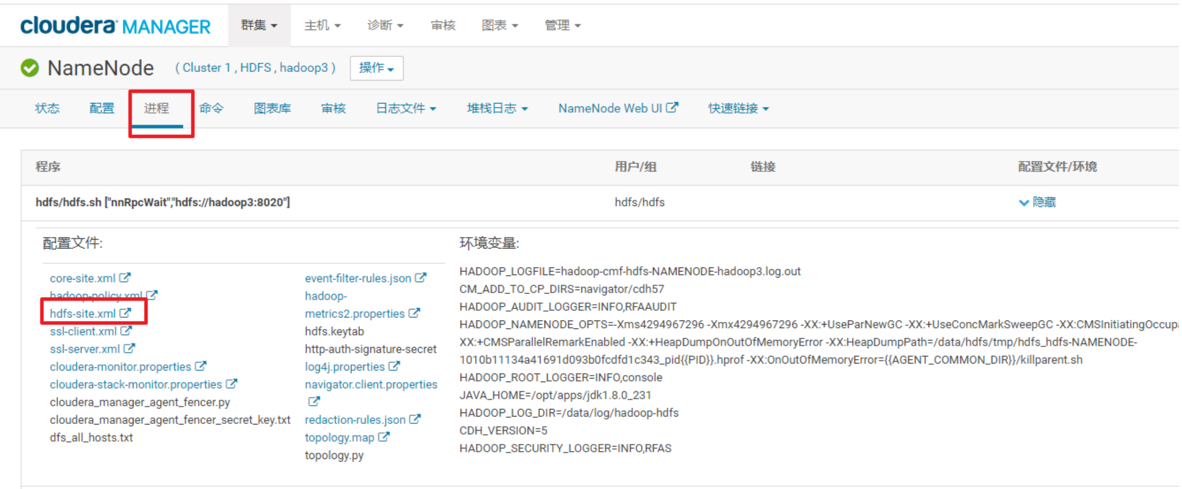
筛选器

状态

无	1
运行状况良好	12

- 授权状态
- 维护模式
- 机架
- 角色组
- 角色类型
- 状态
- 运行状况测试
- 已抑制运行状况测试

□		角色类型	状态	主机
□	●	Balancer	不适用	hadoop2
□	✓	DataNode	已启动	hadoop3
□	✓	DataNode	已启动	hadoop4
□	✓	DataNode	已启动	hadoop5
□	✓	DataNode	已启动	hadoop2
□	✓	Failover Controller	已启动	hadoop3
□	✓	Failover Controller	已启动	hadoop2
□	✓	HttpFS	已启动	hadoop4
□	✓	JournalNode	已启动	hadoop3
□	✓	JournalNode	已启动	hadoop4
□	✓	JournalNode	已启动	hadoop5
□	✓	NameNode (活动)	已启动	hadoop3
□	✓	NameNode (备用)	已启动	hadoop2



程序	用户/组	链接	配置文件/环境
hdfs/hdfs.sh ["nnRpcWait","hdfs://hadoop3:8020"]	hdfs/hdfs		隐藏

配置文件:

core-site.xml
hadoop-policy.xml
hdfs-site.xml
ssl-client.xml
ssl-server.xml
cloudera-monitor.properties
cloudera-stack-monitor.properties
cloudera_manager_agent_fencer.py
cloudera_manager_agent_fencer_secret_key.txt
dfs_all_hosts.txt
event-filter-rules.json
hadoop-metrics2.properties
hdfs.keytab
http-auth-signature-secret
log4j.properties
navigator.client.properties
redaction-rules.json
topology.map
topology.py

环境变量:

```
HADOOP_LOGFILE=hadoop-cmf-hdfs-NAMENODE-hadoop3.log.out
CM_ADD_TO_CP_DIRS=navigator/cdh57
HADOOP_AUDIT_LOGGER=INFO,RFAAUDIT
HADOOP_NAMENODE_OPTS=-Xms4294967296 -Xmx4294967296 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccup
XX:+CMSParallelRemarkEnabled -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/data/hdfs/tmp/hdfs_hdfs-NAMENODE-
1010b11134a41691d093b0fcdfd1c343_pid{{PID}}.hprof -XX:OnOutOfMemoryError={{AGENT_COMMON_DIR}}/killparent.sh
HADOOP_ROOT_LOGGER=INFO,console
JAVA_HOME=/opt/apps/jdk1.8.0_231
HADOOP_LOG_DIR=/data/log/hadoop-hdfs
CDH_VERSION=5
HADOOP_SECURITY_LOGGER=INFO,RFAS
```

中止任务

```
yarn application -kill application_1602905611313_0048
```

13 数据可视化

参考logistics-front工程

启动生产者

```
kafka-console-producer --broker-list hadoop3:9092 --topic lg_bus_info
```

测试数据

```
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.372841_39.94876,70,north,7000,1594076127,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.376274_39.948299,70,north,7000,1594072527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.378763_39.948431,70,north,7000,1594073527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.382282_39.948562,70,north,7000,1594074527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.382883_39.948431,70,north,7000,1594075527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.386574_39.948496,70,north,7000,1594076527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.38872_39.948628,70,north,7000,1594076727,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.38872_39.94876,70,north,7000,1594076827,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.393741_39.948957,70,north,7000,1594076927,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.393784_39.949977,70,north,7000,1594071527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.393805_39.950158,70,north,7000,1594072527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-
11111,116.393741_39.951227,70,north,7000,1594073527,70,500,1,1,0,0
```

```
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-11111,116.393698_39.951038,70,north,7000,1594074527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-11111,116.393988_39.952453,70,north,7000,1594075527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-11111,116.393676_39.952551,70,north,7000,1594076527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-11111,116.395168_39.955857,70,north,7000,1594077527,70,500,1,1,0,0
316d5c75-e860-4cc9-a7de-ea2148c244a0,4444,ysz11111,京A-11111,116.395157_39.957223,70,north,7000,1594078527,70,500,1,1,0,0
```