

电商数仓数据采集平台&离线数据仓库

目录

第一章 用户行为采集平台	5
1.项目需求及架构设计	5
1.1 需求分析	5
1.2 项目框架	5
1.3 脚本	8
2.用户行为日志	9
2.1 用户行为日志概述	9
2.2 用户行为日志内容	10
2.3 用户行为日志格式	11
2.4 服务器和 JDK 准备	13
2.5 模拟数据（网上资源）	16
3.用户行为数据采集模块	20
3.1 数据通道	20
3.2 环境准备	20
3.3 日志采集 Flume	33
第二章 业务数据采集平台	36
1.电商业务简介	36
1.1 电商业务流程	36
1.2 电商常识	37
2.业务数据介绍	39
2.1 电商系统表结构	39
2.2 MySQL 安装	40
2.2 业务数据模拟	41
3.业务数据采集模块	44
3.1 采集通道	44
3.2 采集工具	44

3.3 Maxwell 使用	47
第三章 数仓数据同步策略	49
1. 实时数仓同步数据	49
2. 离线数仓同步数据	50
2.1 用户行为数据同步	50
2.2 业务数据同步	55
第四章 数据仓库环境准备	70
4.0 Hbase 简单应用	72
4.0.1 Hbase 解压	72
4.0.2 Hbase 配置文件	72
4.0.3 Hbase 运行	73
4.1 数据仓库运行环境	74
4.1.1 Hive 环境搭建	74
4.1.2 Hive on Spark 配置	78
4.1.3 Yarn 环境配置	80
4.2 数据仓库开发环境	81
第五章 数仓开发之 ODS 层	83
5.1 日志表	83
5.2 业务表	85
5.2.1 活动信息表（全量表）	85
5.2.2 活动规则表（全量表）	85
5.2.3 一级品类表（全量表）	85
5.2.4 二级品类表（全量表）	86
5.2.5 三级品类表（全量表）	86
5.2.6 编码字典表（全量表）	86
5.2.7 省份表（全量表）	86
5.2.8 地区表（全量表）	87
5.2.9 品牌表（全量表）	87
5.2.10 购物车表（全量表）	87
5.2.11 优惠券信息表（全量表）	88
5.2.12 商品平台属性表（全量表）	88
5.2.13 商品表（全量表）	88

5.2.14 商品销售属性值表（全量表）	89
5.2.15 SPU 表（全量表）	89
5.2.16 购物车表（增量表）	90
5.2.17 评论表（增量表）	90
5.2.18 优惠券领用表（增量表）	90
5.2.19 收藏表（增量表）	91
5.2.20 订单明细表（增量表）	91
5.2.21 订单明细活动关联表（增量表）	91
5.2.22 订单明细优惠券关联表（增量表）	92
5.2.23 订单表（增量表）	92
5.2.24 退单表（增量表）	92
5.2.25 订单状态流水表（增量表）	93
5.2.26 支付表（增量表）	93
5.2.27 退款表（增量表）	93
5.2.28 用户表（增量表）	94
5.2.29 数据装载脚本	94
5.3 结果展示	97
第六章 数仓开发之 DIM 层	98
6.1 商品维度表	98
6.2 优惠券维度表	101
6.3 活动维度表	103
6.4 地区维度表	105
6.5 日期维度表	106
6.6 用户维度表	107
6.7 数据装载脚本	111
6.7.1 首日装载脚本	111
6.7.2 每日装载脚本	118
6.8 结果展示	126

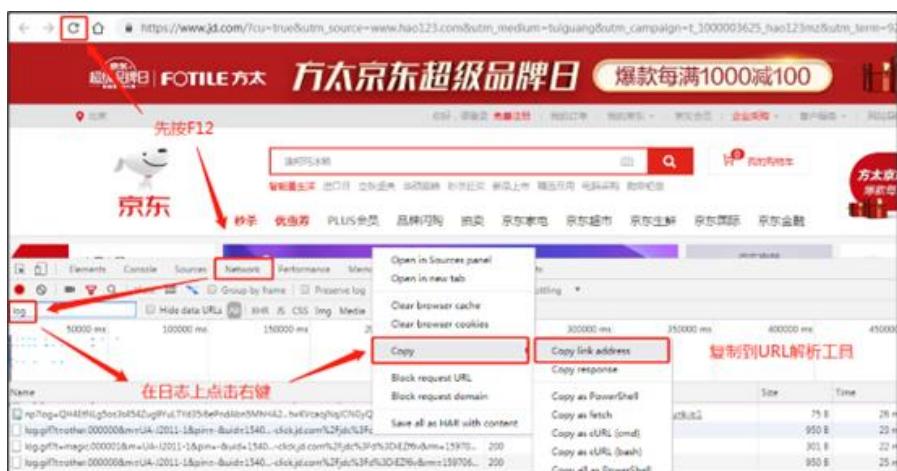
应用开发-电商数仓数据采集平台

数据仓库（ Data Warehouse ），是为企业制定决策，提供数据支持的。可以帮助企业，改进业务流程、提高产品质量等。数据仓库的输入数据通常包括：业务数据、用户行为数据和爬虫数据等

业务数据：就是各行业在处理事务过程中产生的数据。比如用户在电商网站中登录、下单、支付等过程中，需要和网站后台数据库进行增删改查交互，产生的数据就是业务数据。业务数据通常存储在 MySQL、Oracle 等数据库中。



用户行为数据：用户在使用产品过程中，通过埋点收集与客户端产品交互过程中产生的数据，并发往日志服务器进行保存。比如页面浏览、点击、停留、评论、点赞、收藏等。用户行为数据通常存储在日志文件中。



爬虫数据：通常是通过技术手段获取其他公司网站的数据。出于版权考虑一般不会在商业环境使用。我们有同学完成了爬虫→flume→kafka 的项目，具体内容请参考我们的另一份文档“研究与其他应用项目报告”

在这份文档中，我们主要介绍我们的用户行为采集平台与离线数据仓库的开发过程。

第一章 用户行为采集平台

1. 项目需求及架构设计

1.1 需求分析

本项目首先要开发一个为电商实时数仓与电商离线数仓采集数据的平台，主要考量要从参考数仓的数据体系。需要搭建用户行为数据采集平台和业务数据采集平台。接着本项目考虑实现离线或实时数仓。

[离线数仓数据体系 \(点击图片可查看完整电子表格\)](#)



电商离线指标体系.
xlsx

[实时数仓数据体系 \(点击图片可查看完整电子表格\)](#)



电商实时指标体系.
xlsx

1.2 项目框架

1.2.1 技术选型

技术选型主要考虑因素：数据量大小、业务需求、行业内经验、技术成熟度、开发维护成本、总成本预算

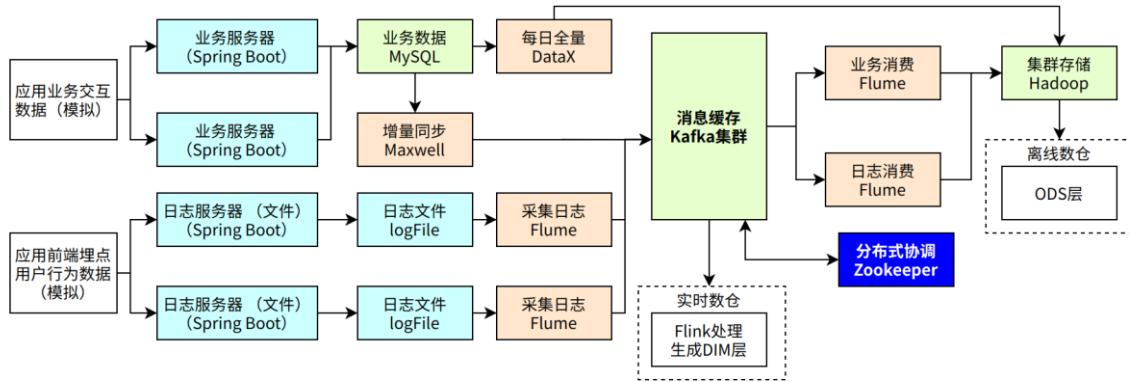
数据采集传输：[Flume](#), [Kafka](#), [DataX](#), [Maxwell](#), Sqoop

- █ 离线实时共用
- █ 离线
- █ 实时
- █ 不采用

数据存储：[MySQL](#), [HDFS](#), [HBase](#), [Redis](#), MongoDB

数据计算：[Hive](#), [Spark](#), [Flink](#)

流程结构图



1.2.2 框架版本选型

选择 Apache/CDH/HDP 版本

- (1) Apache: 运维麻烦, 组件间兼容性需要自己调研。 (一般大厂使用, 技术实力雄厚, 有专业的运维人员) (最终选择)
- (2) CDH: 国内使用最多的版本, 但 CM 不开源, 今年开始收费, 一个节点 1 万美金/年。
- (3) HDP: 开源, 可以进行二次开发, 但是没有 CDH 稳定, 国内使用较少

云服务选择

- (1) 阿里云的 EMR、MaxCompute、DataWorks
- (2) 亚马逊云 EMR
- (3) 腾讯云 EMR
- (4) 华为云 EMR

框架版本

注意事项: 框架选型尽量不要选择最新的框架, 选择最新框架半年前左右的稳定版。

框架	旧版本	选择版本
Hadoop	2.7.2	3.1.3
Zookeeper	3.4.10	3.8.1
MySQL	5.6.24	8.0.32-Ubuntu0.22.10.2
Flume	1.7.0	1.9.0
Kafka	2.4.1	3.0.0
DataX		3.0.0
Maxwell		1.29.2
Hive	1.2.1	1.2.1
Spark		3.0.0
Hbase		2.4.11
Flink		1.13.0
Redis		6.0.8

*Hbase, Flink, Redis 仅做了学习与其他实践, 未整合进本报告中的项目, 关于它们的内容请

参考“研究与其他应用报告”

服务器选型

物理机：

- 以 128G 内存, 20 核物理 CPU, 40 线程, 8THDD 和 2TSSD 硬盘, 戴尔品牌单台报价 4W 出头。一般物理机寿命 5 年左右。
- 需要有专业的运维人员, 平均一个月 1 万。电费也是不少的开销。

云主机：

- 云主机：以阿里云为例, 差不多相同配置, 每年 5W。
- 很多运维工作都由阿里云完成, 运维相对较轻松。

企业选择

- 金融有钱公司和阿里没有直接冲突的公司选择阿里云。
- 中小公司、为了融资上市, 选择阿里云, 获得融资后买物理机。
- 有长期打算, 资金比较足, 选择物理机。

集群规模

确认集群规模(假设:每台服务器 8T 磁盘, 128G 内存)

- 每天日活跃用户 100 万, 每人一天平均 100 条: $100 \text{ 万} * 100 \text{ 条} = 1 \text{ 亿条}$
- 每条日志 1K 左右, 每天 1 亿条: $100000000 / 1024 / 1024 = \text{约 } 100\text{G}$
- 半年内不扩容服务器来算: $100\text{G} * 180 \text{ 天} = \text{约 } 18\text{T}$
- 保存 3 副本: $18\text{T} * 3 = 54\text{T}$
- 预留 20%~30%Buf = $54\text{T} / 0.7 = 77\text{T}$
- 约 $8\text{T} * 10 \text{ 台服务器}$

如果考虑数仓分层、数据采用压缩等, 需要重新再计算

集群资源规划设计(设想企业场景)

在企业中通常会搭建一套生产集群和一套测试集群。生产集群运行生产任务, 测试集群用于上线前代码编写和测试。

生产集群(建议)

- (1) 消耗内存的分开
- (2) 数据传输数据比较紧密的放在一起(Kafka、Zookeeper)
- (3) 客户端尽量放在一到两台服务器上, 方便外部访问
- (4) 有依赖关系的尽量放到同一台服务器(例如: Hive 和 mysql)

Master	Master	core	core	core	common	common	common
---------------	---------------	-------------	-------------	-------------	---------------	---------------	---------------

nn	nn	dn	dn	dn	JournalNode	JournalNode	JournalNode
rm	rm	nm	nm	nm			
					zk	zk	zk
hive	hive	hive	hive	hive			
		kafka	kafka	kafka			
spark	spark	spark	spark	spark			
datax	datax	datax	datax	datax			
Ds-master	Ds-master	Ds-worker	Ds-worker	Ds-worker			
maxwell							
superset							
mysql							
flume	flume						
flink	flink						
		clickhouse					
			redis				
hbase							

测试集群服务器规划（本项目）

服务名称	子服务	服务器 Master	服务器 Slave1	服务器 Slave2
HDFS	NameNode	√		
	DataNode	√	√	√
	SecondaryNameNode			√
Yarn	NodeManager	√	√	√
	Resourcemanager		√	
Zookeeper	Zookeeper Server	√	√	√
Flume (采集日志)	Flume	√	√	
Kafka	Kafka	√	√	√
Flume (消费 Kafka 日志)	Flume			√
Flume (消费 Kafka 业务)	Flume			√
MySQL	MySQL	√		
DataX		√	√	√
Maxwell		√		
Spark		√	√	√
Hive		√	√	√
Flink	仅学习，未整合			
Redis	仅学习，未整合			
Hbase	仅学习，未整合			
Sqoop	仅学习，未整合			
服务数总计				

1.3 脚本

```

hadoop@Master:~/bin$ ls
clean-snap.sh    gen_import_config.py  log.sh          xsync
env.sh           gen_import_config.sh  mxw.sh         zk.sh
f1.sh           jpsall               myhadoop.sh
f2.sh           kf.sh                mysql_to_hdfs_full.sh

```

项目的搭建过程中创建了很多脚本，主要是为了进行工程的自动化。

xsync.sh 自动化分发在集群中分发脚本

log.sh 自动调用网上的软件包资源生成用户行为日志数据

myhadoop.sh 自动启动 hadoop 集群

zk.sh 自动启动 Zookeeper 集群

kf.sh 自动启动 Kafka 集群

mxw.sh 自动启动 Maxwell

f1.sh 日志采集 Flume 进程的启停脚本

f2.sh 日志消费 Flume 启停脚本

jpsall 所有虚拟机 java 进程快速查看脚本

```

hadoop@Master:/opt/module/flume$ jpsall
===== Master =====
5505 Kafka
5026 QuorumPeerMain
14039 Jps
6667 NameNode
6973 JobHistoryServer
===== Slave1 =====
5877 Kafka
5416 QuorumPeerMain
7304 NodeManager
6986 DataNode
9931 Jps
7167 ResourceManager
===== Slave2 =====
5330 SecondaryNameNode
8787 Jps
4327 QuorumPeerMain
5435 NodeManager
5197 DataNode
4782 Kafka

```

2. 用户行为日志

2.1 用户行为日志概述

用户行为日志的内容，主要包括用户的各项**行为信息**以及行为所处的**环境信息**。收集这些信息的主要目的是优化产品和为各项分析统计指标提供数据支撑。收集这些信息的手段通常为**埋点**。

目前主流的埋点方式，有**代码埋点（前端/后端）、可视化埋点、全埋点**等。

代码埋点是通过调用埋点 SDK 函数，在需要埋点的业务逻辑功能位置调用接口，上报埋点数据。例如，我们对页面中的某个按钮埋点后，当这个按钮被点击时，可以在这个按钮对应的 `onClick` 函数里面调用 SDK 提供的数据发送接口，来发送数据。

可视化埋点只需要研发人员集成采集 SDK，不需要写埋点代码，业务人员就可以通过访问分析平台的“圈选”功能，来“圈”出需要对用户行为进行捕捉的控件，并对该事件进行命名。圈选完毕后，这些配置会同步到各个用户的终端上，由采集 SDK 按照圈选的配置自动进行用户行为数据的采集和发送。

全埋点是通过在产品中嵌入 SDK，前端自动采集页面上的全部用户行为事件，上报埋点数据，相当于做了一个统一的埋点。然后再通过界面配置哪些数据需要在系统里面进行分析。

企业一般采用全埋点。

2.2 用户行为日志内容

本项目收集和分析的用户行为信息主要有**页面浏览记录、动作记录、曝光记录、启动记录和错误记录**。

页面浏览记录

页面浏览记录，记录的是访客对页面的浏览行为，该行为的环境信息主要有用户信息、时间信息、地理位置信息、设备信息、应用信息、渠道信息及页面信息等。



用户信息	包括用户ID、设备ID
时间信息	用户跳入页面的时间
地理位置信息	用户浏览页面时所处的地理位置
设备信息	包括设备品牌、设备型号、设备系统
应用信息	指用户访问的应用信息，例如应用版本
渠道信息	指应用的下载渠道
页面信息	用户浏览的页面相关信息，包括页面ID，页面对象

动作记录

动作记录，记录的是用户的业务操作行为，该行为的环境信息主要有用户信息、时间信息、地理位置信息、设备信息、应用信息、渠道信息及动作目标对象信息等。



用户信息	包括用户ID、设备ID
时间信息	动作时间
地理位置信息	动作发生时所处的地理位置
设备信息	包括设备品牌、设备型号、设备系统
应用信息	指用户访问的应用信息，例如应用版本
渠道信息	指应用的下载渠道
动作目标信息	动作目标对象相关信息，包括对象类型，对象ID

曝光记录

曝光记录，记录的是曝光行为，该行为的环境信息主要有用户信息、时间信息、地理位置信息、

设备信息、应用信息、渠道信息及曝光对象信息等。



用户信息	包括用户ID、设备ID
时间信息	曝光时间
地理位置信息	曝光行为发生时所处的地理位置
设备信息	包括设备品牌、设备型号、设备系统
应用信息	指用户访问的应用信息，例如应用版本
渠道信息	指应用的下载渠道
曝光对象信息	曝光对象相关信息，包括对象类型、对象ID

启动记录

启动记录，记录的是用户启动应用的行为，该行为的环境信息主要有用户信息、时间信息、地理位置信息、设备信息、应用信息、渠道信息、启动类型及开屏广告信息等。



用户信息	包括用户ID、设备ID
时间信息	启动时间
地理位置信息	启动时所处的地理位置
设备信息	包括设备品牌、设备型号、设备系统
应用信息	指用户访问的应用信息，例如应用版本
渠道信息	指应用的下载渠道
启动类型	包括图标和推送
开屏广告信息	包括广告ID等信息

错误记录

启动记录，记录的是用户在使用应用过程中的报错行为，该行为的环境信息主要有用户信息、时间信息、地理位置信息、设备信息、应用信息、渠道信息、以及可能与报错相关的页面信息、动作信息、曝光信息和动作信息。

2.3 用户行为日志格式

我们的日志结构大致可分为两类，一是页面日志，二是启动日志。

页面日志

页面日志，以页面浏览为单位，即一个页面浏览记录，生成一条页面埋点日志。一条完整的页面日志包含，一个页面浏览记录，若干个用户在该页面所做的动作记录，若干个该页面的曝光记录，以及一个在该页面发生的报错记录。除上述行为信息，页面日志还包含了这些行为所处的各种环境

信息，包括用户信息、时间信息、地理位置信息、设备信息、应用信息、渠道信息等。

```
{
    "common": {
        "ar": "230000", -- 环境信息
        "ba": "iPhone", -- 地区编码
        "ch": "Appstore", -- 手机品牌
        "is_new": "1", -- 渠道
        "is_new": "1", -- 是否首日使用，首次使用的当日，该字段值为 1，过了
24:00，该字段置为 0。
        "md": "iPhone 8", -- 手机型号
        "mid": "YXfhjAYH6As2z9Iq", -- 设备 id
        "os": "iOS 13.2.9", -- 操作系统
        "uid": "485", -- 会员 id
        "vc": "v2.1.134" -- app 版本号
    },
    "actions": [
        {
            "action_id": "favor_add", -- 动作(事件)
            "item": "3", -- 动作 id
            "item_type": "sku_id", -- 目标 id
            "ts": 1585744376605 -- 目标类型
            "ts": 1585744376605 -- 动作时间戳
        }
    ],
    "displays": [
        {
            "displayType": "query", -- 曝光
            "item": "3", -- 曝光类型
            "item_type": "sku_id", -- 曝光对象 id
            "order": 1, -- 曝光对象类型
            "pos_id": 2 -- 出现顺序
        },
        {
            "displayType": "promotion",
            "item": "6", -- 曝光位置
            "item_type": "sku_id",
            "order": 2,
            "pos_id": 1
        },
        {
            "displayType": "promotion",
            "item": "9", -- 曝光
            "item_type": "sku_id",
            "order": 3,
            "pos_id": 1
        },
        {
            "displayType": "recommend",
            "item": "6", -- 推荐
            "item_type": "sku_id",
            "order": 4,
            "pos_id": 2
        },
        {
            "displayType": "query",
            "item": "6", -- 搜索
            "item_type": "sku_id",
            "order": 5,
            "pos_id": 1
        }
    ],
    "page": { -- 页面信息
    }
}
```

```

    "during_time": 7648,           -- 持续时间毫秒
    "item": "3",                  -- 目标 id
    "item_type": "sku_id",        -- 目标类型
    "last_page_id": "login",     -- 上页类型
    "page_id": "good_detail",   -- 页面 ID
    "sourceType": "promotion"   -- 来源类型
},
"err": {
    "error_code": "1234",        -- 错误码
    "msg": "*****"              -- 错误信息
},
"ts": 1585744374423          -- 跳入时间戳
}

```

启动日志

启动日志以启动为单位，及一次启动行为，生成一条启动日志。一条完整的启动日志包括一个启动记录，一个本次启动时的报错记录，以及启动时所处的环境信息，包括用户信息、时间信息、地理位置信息、设备信息、应用信息、渠道信息等。

```

{
  "common": {
    "ar": "370000",
    "ba": "Honor",
    "ch": "wandoujia",
    "is_new": "1",
    "md": "Honor 20s",
    "mid": "eQF5boERMJFOujcp",
    "os": "Android 11.0",
    "uid": "76",
    "vc": "v2.1.134"
  },
  "start": {
    "entry": "icon",           -- icon 手机图标  notice 通知  install 安装后启动
    "loading_time": 18803,     -- 启动加载时间
    "open_ad_id": 7,           -- 广告页 ID
    "open_ad_ms": 3449,        -- 广告总共播放时间
    "open_ad_skip_ms": 1989   -- 用户跳过广告时点
  },
  "err": {                    -- 错误
    "error_code": "1234",      -- 错误码
    "msg": "*****"            -- 错误信息
  },
  "ts": 1585744304000
}

```

2.4 服务器和 JDK 准备

按照之前实践的步骤，安装 Master、Slave1、Slave2 三台虚拟机。

使用的虚拟机软件为 virtualbox，三台机器的配置规划如下

Master（固定 IP 设为 192.168.56.3）

- 操作系统:Ubuntu 22.04 LTS (Jammy Jellyfish) (64-bit)
- 内存大小:10240 MB
- 处理器:6

- 启动顺序:光驱,硬盘
- 存储 SATA 端口 0: 20G + 10G (扩充磁盘)
- 网卡 1:Intel PRO/1000 MT 桌面(仅主机(Host-Only)网络,'VirtualBox Host-Only Ethernet Adapter')
- 网卡 2:Intel PRO/1000M[T 桌面(网络地址转换(NAT))]

Slave1 (固定 IP 设为 192.168.56.4)

- 操作系统:Ubuntu (64-bit)
- 内存大小:5120 MB
- 处理器:3
- 启动顺序:光驱,硬盘
- 存储 SATA 端口 0: 20G
- 网卡 1:Intel PRO/1000 MT 桌面(仅主机(Host-Only)网络,'VirtualBox Host-Only Ethernet Adapter')
- 网卡 2:Intel PRO/1000M[T 桌面(网络地址转换(NAT))]

Slave2 (固定 IP 设为 192.168.56.5)

- 操作系统:Ubuntu (64-bit)
- 内存大小:5120 MB
- 处理器:3
- 启动顺序:光驱,硬盘
- 存储 SATA 端口 0: 20G
- 网卡 1:Intel PRO/1000 MT 桌面(仅主机(Host-Only)网络,'VirtualBox Host-Only Ethernet Adapter')
- 网卡 2:Intel PRO/1000M[T 桌面(网络地址转换(NAT))]

三台虚拟机以及主机两两之间已经进行了 IP 地址和 host 名称的映射设定。每台虚拟机均创建了名为"hadoop"的用户，这是在本项目中我们会主要使用的用户。三台虚拟机之间均配置了 ssh 无密码登录，对于 hadoop，Master 设置为 NameNode，其他两个主机设置为 DataNode。每台主机均使用 JDK1.8。在所有虚拟机上的，**hadoop 的根目录在/usr/local/lib/hadoop，其他软件组件的根目录在/opt/module 中。**在 Master 主机上，部署软件的过程中会随时进行环境变量的设定。

2.4.1 编写集群分发脚本 xsync

- (1) 需求：循环复制文件到所有节点的相同目录下
- (2) 需求分析

①rsync 命令原始拷贝：

```
rsync -av /opt/module hadoop@Slave1:/opt/
```

②期望脚本：

```
xsync 要同步的文件名称
```

③说明：在 Master 主机/home/hadoop/bin 这个目录下存放的脚本，将该目录加入环境变量，hadoop@Master 用户可以在系统任何地方直接执行。

(3) 脚本实现

① 在用户家目录/home/hadoop 下创建 bin 文件夹 ②在/home/hadoop/bin 目录下创建 xsync 文件，以便全局调用

```
[hadoop@Master ~]$ mkdir bin
[hadoop@Master ~]$ cd /home/hadoop/bin
[hadoop@Master ~]$ vim xsync
```

在该文件中编写如下代码

```
#!/bin/bash
#1. 判断参数个数
if [ $# -lt 1 ]
then
    echo Not Enough Argument!
    exit;
fi
#2. 遍历集群所有机器
for host in Master Slave1 Slave2
do
    echo ===== $host =====
    #3. 遍历所有目录，挨个发送
    for file in $@
    do
        #4 判断文件是否存在
        if [ -e $file ]
        then
            #5. 获取父目录
            pdir=$(cd -P $(dirname $file); pwd)
            #6. 获取当前文件的名称
            fname=$(basename $file)
            ssh $host "mkdir -p $pdir"
            rsync -av $pdir/$fname $host:$pdir
        else
            echo $file does not exists!
        fi
    done
done
```

③修改脚本 xsync 具有执行权限 ④测试脚本

```
[hadoop@Master bin]$ chmod +x xsync
[hadoop@Master bin]$ xsync xsync
```

```

hadoop@Master:~$ xsync homenet.txt
===== Master =====
sending incremental file list

sent 66 bytes received 12 bytes 156.00 bytes/sec
total size is 70 speedup is 0.90
===== Slave1 =====
sending incremental file list
homenet.txt

sent 183 bytes received 35 bytes 436.00 bytes/sec
total size is 70 speedup is 0.32
===== Slave2 =====
sending incremental file list
homenet.txt

sent 183 bytes received 35 bytes 436.00 bytes/sec
total size is 70 speedup is 0.32

```

2.5 模拟数据（网上资源）

2.5.1 使用说明

1) 将准备好的 `application.yml`、`gmall2020-mock-log-2021-10-10.jar`、`path.json`、`logback.xml` 上传到 Master 的 `/opt/module/applog` 目录下

- 创建 `applog` 路径

```
[hadoop@Master module]$ mkdir /opt/module/applog
```

- 上传文件到 `/opt/module/applog` 目录

2) 配置文件

- `application.yml` 文件 可以根据需求生成对应日期的用户行为日志。

```
[hadoop@Master applog]$ vim application.yml
```

修改如下内容

```

# 外部配置打开
logging.config: "./logback.xml"
#业务日期 注意：并不是 Linux 系统生成日志的日期，而是生成数据中的时间
mock.date: "2020-06-14"

#模拟数据发送模式
#mock.type: "http"
#mock.type: "kafka"
mock.type: "log"

#http 模式下，发送的地址
mock.url: "http://hdp1/applog"

#kafka 模式下，发送的地址
mock:

```

```

kafka-server: "hdp1:9092,hdp2:9092,hdp3:9092"
kafka-topic: "ODS_BASE_LOG"

#启动次数
mock.startup.count: 200
#设备最大值
mock.max.mid: 500000
#会员最大值
mock.max.uid: 100
#商品最大值
mock.max.sku-id: 35
#页面平均访问时间
mock.page.during-time-ms: 20000
#错误概率 百分比
mock.error.rate: 3
#每条日志发送延迟 ms
mock.log.sleep: 10
#商品详情来源 用户查询, 商品推广, 智能推荐, 促销活动
mock.detail.source-type-rate: "40:25:15:20"
#领取购物券概率
mock.if_get_coupon_rate: 75
#购物券最大 id
mock.max.coupon-id: 3
#搜索关键词
mock.search.keyword: "图书,小米,iphone11,电视,口红,ps5,苹果手机,小米盒子"

```

- **path.json**, 该文件用来配置访问路径。根据需求, 可以灵活配置用户点击路径。

```

[
    {"path": ["home", "good_list", "good_detail", "cart", "trade", "payment"], "rate": 20 },
    {"path": ["home", "search", "good_list", "good_detail", "login", "good_detail", "cart", "trade", "payment"], "rate": 40 },
    {"path": ["home", "mine", "orders_unpaid", "trade", "payment"], "rate": 10 },
    {"path": ["home", "mine", "orders_unpaid", "good_detail", "good_spec", "comment", "trade", "payment"], "rate": 5 },
    {"path": ["home", "mine", "orders_unpaid", "good_detail", "good_spec", "comment", "home"], "rate": 5 },
    {"path": ["home", "good_detail"], "rate": 10 },
    {"path": ["home"], "rate": 10 }
]

```

- **logback** 配置文件。可配置日志生成路径, 修改内容如下

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <property name="LOG_HOME" value="/opt/module/applog/log" />
    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%msg%n</pattern>
        </encoder>
    </appender>

    <appender
        class="ch.qos.logback.core.rolling.RollingFileAppender">
        <name="rollingFile">
        <rollingPolicy
            class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${LOG_HOME}/app.%d{yyyy-MM-
dd}.log</fileNamePattern>
        </rollingPolicy>
        <encoder>

```

```

        <pattern>%msg%n</pattern>
    </encoder>
</appender>

<!-- 将某一个包下日志单独打印日志 -->
<logger name="com.atgugu.gmall2020.mock.log.util.LogUtil"
    level="INFO" additivity="false">
    <appender-ref ref="rollingFile" />
    <appender-ref ref="console" />
</logger>

<root level="error" >
    <appender-ref ref="console" />
</root>
</configuration>
```

3) 生成日志

进入到/opt/module/applog 路径，执行以下命令

```
[hadoop@Master applog]$ java -jar gmall2020-mock-log-2021-10-10.jar
```

在/opt/module/applog/log 目录下查看生成日志

```
[hadoop@Master log]$ ll
```

2.5.2 集群日志生成脚本

在 Master 的/home/hadoop 目录下创建 bin 目录，这样脚本可以在服务器的任何目录执行。

```
[hadoop@Master ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/hadoop/.local/bin:/home/hadoop/bin
```

(1) 在/home/hadoop/bin 目录下创建脚本 **log.sh**

```
[hadoop@Master bin]$ vim log.sh
```

(2) 在脚本中编写如下内容

```
#!/bin/bash
for i in Master Slave1; do
    echo "===== $i ====="
    ssh $i "cd /opt/module/applog/; java -jar gmall2020-mock-log-2021-10-10.jar >/dev/null 2>&1 &"
done
```

注：

① /opt/module/applog/ 为 jar 包及配置文件所在路径

② /dev/null 代表 Linux 的空设备文件，所有往这个文件里面写入的内容都会丢失，俗称“黑洞”。

标准输入 0：从键盘获得输入 /proc/self/fd/0

标准输出 1：输出到屏幕（即控制台） /proc/self/fd/1

错误输出 2：输出到屏幕（即控制台） /proc/self/fd/2

(3) 修改脚本执行权限

```
[hadoop@Master bin]$ chmod 777 log.sh
```

(4) 将 jar 包及配置文件上传至 Slave1 的 /opt/module/applog/ 路径

(5) 启动脚本

```
[hadoop@Master module]$ log.sh
```

(6) 分别在 Master, Slave1 的 /opt/module/applog/log 目录上查看生成的数据

```
[hadoop@Master logs]$ ls  
app.2020-06-14.log  
[hadoop@Slave1 logs]$ ls  
app.2020-06-14.log
```

```
hadoop@Master:~$ log.sh
```

-Master
-Slave1

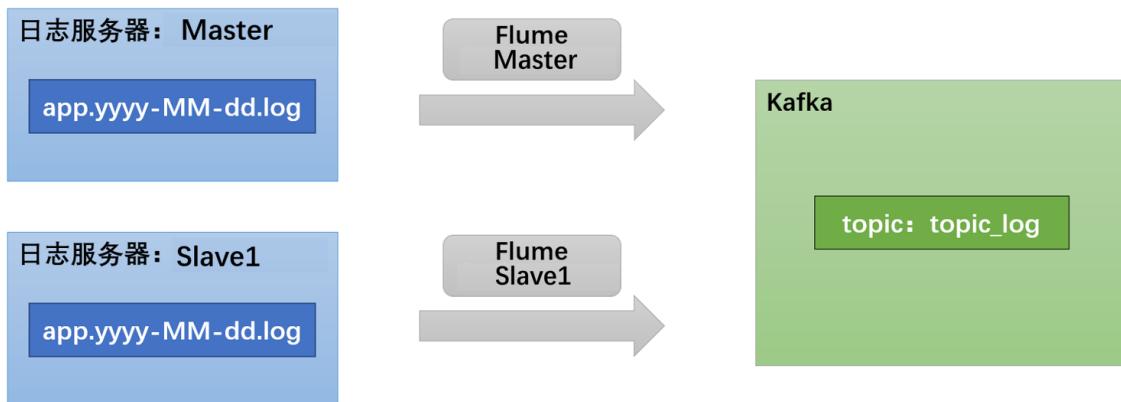
```
hadoop@Master:/opt/module/applog/log$ ls  
app.2023-04-11.log  app.2023-04-12.log  app.2023-05-10.log
```

```
hadoop@Slave1:/opt/module/applog/log$ ls  
app.2023-04-11.log  app.2023-04-12.log  app.2023-05-10.log
```

```
{ "common": { "ar": "110000", "ba": "iPhone", "ch": "Appstore", "is_new": "0", "md": "iPhone 8", "mid": "mid_36100", "os": "iOS 13.2.9", "uid": "670", "vc": "v2.1.134" }, "start": { "entry": "install", "loading_time": 1258, "open_ad_id": 6, "open_ad_ms": 5595, "open_ad_skip_ms": 4262 }, "ts": 1592116523000 }  
{ "common": { "ar": "110000", "ba": "iPhone", "ch": "Appstore", "is_new": "0", "md": "iPhone 8", "mid": "mid_36100", "os": "iOS 13.2.9", "uid": "670", "vc": "v2.1.134" }, "displays": [ { "display_type": "activity", "item": "2", "item_type": "activity_id", "order": 1, "pos_id": 5 }, { "display_type": "promotion", "item": "21", "item_type": "sku_id", "order": 2, "pos_id": 1 }, { "display_type": "query", "item": "12", "item_type": "sku_id", "order": 3, "pos_id": 5 }, { "display_type": "query", "item": "22", "item_type": "sku_id", "order": 4, "pos_id": 4 }, { "display_type": "query", "item": "21", "item_type": "sku_id", "order": 5, "pos_id": 2 }, { "display_type": "promotion", "item": "6", "item_type": "sku_id", "order": 6, "pos_id": 3 } ], "err": { "error_code": 2605, "msg": "Exception in thread \\"java.net.SocketTimeoutException\\n \\\tat com.atguigu.gmall2020.mock.bean.log.AppError.main(AppError.java:xxxxxx)" }, "page": { "during_time": 15997, "page_id": "home" }, "ts": 1592116523000 }  
{ "actions": [ { "action_id": "get_coupon_id", "item": "3", "item_type": "coupon_id", "ts": "1592116531649" }, { "common": { "ar": "110000", "ba": "iPhone", "ch": "Appstore", "is_new": "0", "md": "iPhone 8", "mid": "mid_36100", "os": "iOS 2.9", "uid": "670", "vc": "v2.1.134" }, "displays": [ { "display_type": "query", "item": "34", "item_type": "sku_id", "order": 1, "pos_id": 2 }, { "display_type": "query", "item": "20", "item_type": "sku_id", "order": 2, "pos_id": 3 }, { "display_type": "query", "item": "28", "item_type": "sku_id", "order": 3, "pos_id": 2 }, { "display_type": "query", "item": "14", "item_type": "sku_id", "order": 4, "pos_id": 3 }, { "display_type": "query", "item": "26", "item_type": "sku_id", "order": 5, "pos_id": 5 }, { "display_type": "query", "item": "19", "item_type": "sku_id", "order": 6, "pos_id": 5 }, { "display_type": "promotion", "item": "31", "item_type": "sku_id", "order": 7, "pos_id": 5 }, { "display_type": "query", "item": "16", "item_type": "sku_id", "order": 8, "pos_id": 5 }, { "display_type": "query", "item": "16", "item_type": "sku_id", "order": 9, "pos_id": 4 }, { "display_type": "recommend", "item": "30", "item_type": "sku_id", "order": 10, "pos_id": 11 }, { "page": { "during_time": 159291, "item": "30", "item_type": "sku_id" }, "last_page_id": "home", "page_id": "good_detail", "source_type": "query" }, "ts": "1592116524000" }  
{ "common": { "ar": "230000", "ba": "vivo", "ch": "web", "is_new": "0", "md": "vivo iqoo3", "mid": "mid_47441", "os": "Android 11.0", "uid": "721", "vc": "v2.1.134" }, "start": { "entry": "icon", "loading_time": 8502, "open_ad_id": 11, "open_ad_ms": 4519, "open_ad_skip_ms": 0 }, "ts": 1592116523000 }  
{ "common": { "ar": "230000", "ba": "vivo", "ch": "web", "is_new": "0", "md": "vivo iqoo3", "mid": "mid_47441", "os": "Android 11.0", "uid": "721", "vc": "v2.1.134" }, "displays": [ { "display_type": "activity", "item": "2", "item_type": "activity_id", "order": 1, "pos_id": 1 }, { "display_type": "activity", "item": "2", "item_type": "activity_id", "order": 2, "pos_id": 1 }, { "display_type": "query", "item": "28", "item_type": "sku_id", "order": 3, "pos_id": 4 }, { "display_type": "recommend", "item": "7", "item_type": "sku_id", "order": 5, "pos_id": 5 }, { "display_type": "query", "item": "12", "item_type": "sku_id", "order": 6, "pos_id": 4 }, { "display_type": "query", "item": "12", "item_type": "sku_id", "order": 6, "pos_id": 5 }, { "display_type": "query", "item": "7", "item_type": "sku_id", "order": 7, "pos_id": 4 }, { "page": { "during_time": 5671, "page_id": "home" }, "ts": "1592116523000" }  
{ "common": { "ar": "230000", "ba": "vivo", "ch": "web", "is_new": "0", "md": "vivo iqoo3", "mid": "mid_47441", "os": "Android 11.0", "uid": "721", "vc": "v2.1.134" }, "displays": [ { "display_type": "query", "item": "15", "item_type": "sku_id", "order": 1, "pos_id": 3 }, { "display_type": "promotion", "item": "23", "item_type": "sku_id", "order": 2, "pos_id": 1 }, { "display_type": "query", "item": "21", "item_type": "sku_id", "order": 3, "pos_id": 1 }, { "display_type": "query", "item": "2", "item_type": "sku_id", "order": 4, "pos_id": 2 } ], "page": { "during_time": 9439, "item": "小米盒子", "item_type": "keyword", "last_page_id": "home", "page_id": "good_list" }, "ts": 1592116524000 }  
{ "actions": [ { "action_id": "favor_add", "item": "20", "item_type": "sku_id", "ts": "1592116525784" }, { "action_id": "get_coupon", "item": "3", "item_type": "coupon_id", "ts": "1592116526568" }, { "common": { "ar": "230000", "ba": "vivo", "ch": "web", "is_new": "0", "md": "vivo iqoo3", "mid": "mid_47441", "os": "Android 11.0", "uid": "721", "vc": "v2.1.134" }, "displays": [ { "display_type": "query", "item": "32", "item_type": "sku_id", "order": 1, "pos_id": 4 }, { "display_type": "promotion", "item": "24", "item_type": "sku_id", "order": 2, "pos_id": 3 }, { "display_type": "query", "item": "35", "item_type": "sku_id", "order": 3, "pos_id": 5 }, { "display_type": "promotion", "item": "28", "item_type": "sku_id", "order": 4, "pos_id": 5 }, { "display_type": "query", "item": "35", "item_type": "sku_id", "order": 5, "pos_id": 5 }, { "display_type": "promotion", "item": "29", "item_type": "sku_id", "order": 6, "pos_id": 4 }, { "display_type": "recommend", "item": "15", "item_type": "sku_id", "order": 7, "pos_id": 5 }, { "display_type": "promotion", "item": "4", "item_type": "sku_id", "order": 8, "pos_id": 3 } ], "page": { "during_time": 2354, "item": "20", "item_type": "sku_id", "la_st_page_id": "good_list", "page_id": "good_detail", "source_type": "query" }, "ts": 1592116525000 } 
```

3. 用户行为数据采集模块

3.1 数据通道



3.2 环境准备

3.2.1 集群所有进程查看脚本

1) 在/home/hadoop/bin 目录下创建脚本 xcall

```
[hadoop@Master bin]$ vim xcall
```

2) 在脚本中编写如下内容

```
#!/bin/bash

for i in Master Slave1 Slave2
do
    echo ----- $i -----
    ssh $i "$*"
done
```

3) 修改脚本执行权限

```
[hadoop@Master bin]$ chmod 777 xcall
```

4) 启动脚本

```
[hadoop@Master bin]$ xcall.sh jps
```

3.2.2 Hadoop 安装

Hadoop 的安装与各类文件配置主要参考下面这个文件，不做赘述。主要的区别是我们的三台 Hadoop 的 Ubuntu 虚拟机名为 **Master**、**Slave1**、**Slave2**，同时我们的 Hadoop HOME 目录设定为 **/usr/local/lib/hadoop**。更多设置可以参考



尚硅谷大数据技术
之Hadoop（入门）

集群部署规划如下：

注意：NameNode 和 SecondaryNameNode 不要安装在同一台服务。 ResourceManager 也很消耗内存，不要和 NameNode、SecondaryNameNode 配置在同一台机器上。

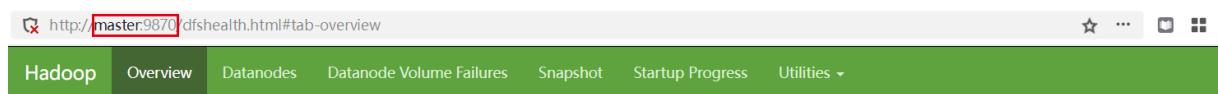
	Master	Slave1	Slave2
HDFS	NameNode DataNode	DataNode	SecondaryName Node DataNode
YARN	NodeManager	ResourceManager NodeManager	NodeManager

同时开启了日志聚集功能。日志聚集概念：应用运行完成以后，将程序运行日志信息上传到 HDFS 系统上。日志聚集功能好处：可以方便的查看到程序运行详情，方便开发调试。

注意：开启日志聚集功能，需要重新启动 *NodeManager*、*ResourceManager* 和 *HistoryManager*。

如果集群是第一次启动，需要在 Master 节点格式化 NameNode（注意格式化之前，一定要先停止上次启动的所有 namenode 和 datanode 进程，然后再删除 data 和 log 数据）。一般在配置了 ResourceManager 的节点（Slave1）启动 YARN

Web 端查看 HDFS 的 Web 页面：<http://Master:9870/>



Overview 'Master:8020' (active)

Started:	Wed May 10 14:58:22 +0800 2023
Version:	3.1.3, rba631c436b806728f8ec2f54ab1e289526c90579
Compiled:	Thu Sep 12 10:47:00 +0800 2019 by ztang from branch-3.1.3
Cluster ID:	CID-39a52f26-fdc3-4867-aa98-6c12f80cccd45
Block Pool ID:	BP-1710093684-192.168.56.3-1679555004549

Web 端查看 SecondaryNameNode 的 Web 页面：<http://Slave2:9868/status.html>

3.2.2.1 Hadoop 群起脚本

```
[hadoop@Master bin]$ pwd
/home/hadoop/bin
[hadoop@Master bin]$ vim myhadoop.sh
```

输入如下内容：

```
#!/bin/bash
if [ $# -lt 1 ]
then
    echo "No Args Input..."
    exit ;
fi
case $1 in
"start")
    echo " ===== 启动 hadoop 集群 ====="
    echo " ----- 启动 hdfs -----"
    ssh Master "/usr/local/lib/hadoop/sbin/start-dfs.sh"
    echo " ----- 启动 yarn -----"
```

```

ssh Slave1 "/usr/local/lib/hadoop/sbin/start-yarn.sh"
echo " ----- 启动 historyserver -----"
ssh Master "/usr/local/lib/hadoop/bin/mapred --daemon start historyserver"
;;
"stop")
echo " ===== 关闭 hadoop 集群 ====="

echo " ----- 关闭 historyserver -----"
ssh Master "/usr/local/lib/hadoop/bin/mapred --daemon stop historyserver"
echo " ----- 关闭 yarn -----"
ssh Slave1 "/usr/local/lib/hadoop/sbin/stop-yarn.sh"
echo " ----- 关闭 hdfs -----"
ssh Master "/usr/local/lib/hadoop/sbin/stop-dfs.sh"
;;
*)
echo "Input Args Error..."
;;
esac

```

[hadoop@Master bin]\$ chmod 777 myhadoop.sh

```

[hadoop@Master:/opt/module/applog/log$ myhadoop.sh start
===== 启动 hadoop 集群 =====
----- 启动 hdfs -----
Starting namenodes on [Master]
Starting datanodes
Starting secondary namenodes [Slave2]
----- 启动 yarn -----
Starting resourcemanager
Starting nodemanagers
----- 启动 historyserver -----
[hadoop@Master:/opt/module/applog/log$ jpsall
===== Master =====
3105 JobHistoryServer
3212 Jps
2798 NameNode
===== Slave1 =====
3201 DataNode
3892 Jps
3382 ResourceManager
3519 NodeManager
===== Slave2 =====
2840 Jps
2536 SecondaryNameNode
2682 NodeManager
2444 DataNode
[hadoop@Master:/opt/module/applog/log$ myhadoop.sh stop
===== 关闭 hadoop 集群 =====
----- 关闭 historyserver -----
----- 关闭 yarn -----
Stopping nodemanagers
Stopping resourcemanager
----- 关闭 hdfs -----
Stopping namenodes on [Master]
Stopping datanodes
Stopping secondary namenodes [Slave2]

```

3.2.2.2 Hadoop 生产经验

(1) 项目经验之 HDFS 存储多目录

①生产环境服务器磁盘情况

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda7	3.6T	1.6T	2.0T	45%	/
devtmpfs	63G	0	63G	0%	/dev
tmpfs	63G	20K	63G	1%	/dev/shm
tmpfs	63G	714M	63G	2%	/run
tmpfs	63G	0	63G	0%	/sys/fs/cgroup
/dev/sdc1	954G	177G	777G	19%	/hd3
/dev/sdd1	954G	177G	777G	19%	/hd4
/dev/sdb1	3.7T	989G	2.7T	27%	/hd2
/dev/sda5	1014M	123M	892M	13%	/boot

②在 hdfs-site.xml 文件中配置多目录，注意新挂载磁盘的访问权限问题。

HDFS 的 DataNode 节点保存数据的路径由 `dfs.datanode.data.dir` 参数决定，其默认值为 `file://${hadoop.tmp.dir}/dfs/data`，若服务器有多个磁盘，必须对该参数进行修改。如服务器磁盘如上图所示，则该参数应修改为如下的值。

```
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///dfs/data1,file:///hd2/dfs/data2,file:///hd3/dfs/data3,file:///hd4/dfs/data4</value>
</property>
```

注意：每台服务器挂载的磁盘不一样，所以每个节点的多目录配置可以不一致。单独配置即可。

(2) 项目经验之集群数据均衡

①节点间数据均衡

开启数据均衡命令：

```
start-balancer.sh -threshold 10
```

对于参数 10，代表的是集群中各个节点的磁盘空间利用率相差不超过 10%，可根据实际情况进行调整。

停止数据均衡命令：

```
stop-balancer.sh
```

②磁盘间数据均衡

生成均衡计划（我们只有一块磁盘，不会生成计划）

```
hdfs diskbalancer -plan Slave1
```

执行均衡计划

```
hdfs diskbalancer -execute Slave1.plan.json
```

查看当前均衡任务的执行情况

```
hdfs diskbalancer -query Slave1
```

取消均衡任务

```
hdfs diskbalancer -cancel Slave1.plan.json
```

(3) 项目经验之 Hadoop 参数调优

①HDFS 参数调优 hdfs-site.xml

The number of Namenode RPC server threads that listen to requests from clients. If `dfs.namenode.servicerpc-address` is not configured then Namenode RPC server threads listen to requests from all nodes.

NameNode 有一个工作线程池，用来处理不同 DataNode 的并发心跳以及客户端并发的元数据操作。

对于大集群或者有大量客户端的集群来说，通常需要增大参数 `dfs.namenode.handler.count` 的默认值 10。

```
<property>
    <name>dfs.namenode.handler.count</name>
    <value>10</value>
</property>
```

$\text{dfs.namenode.handler.count} = 20 \times \log_e^{\text{Cluster Size}}$ ，比如集群规模为 8 台时，此参数设置为 41。可通过简单的 python 代码计算该值，代码如下。

```
[hadoop@Master ~] $ python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> print int(20*math.log(8))
41
>>> quit()
```

② YARN 参数调优 `yarn-site.xml`

情景描述：总共 7 台机器，每天几亿条数据，数据源->Flume->Kafka->HDFS->Hive

面临问题：数据统计主要用 HiveSQL，没有数据倾斜，小文件已经做了合并处理，开启的 JVM 重用，而且 IO 没有阻塞，内存用了不到 50%。但是还是跑的非常慢，而且数据量洪峰过来时，整个集群都会宕掉。基于这种情况有没有优化方案。

解决办法：

内存利用率不够。这个一般是 Yarn 的 2 个配置造成的，单个任务可以申请的最大内存大小，和 Hadoop 单个节点可用内存大小。调节这两个参数能提高系统内存的利用率。

(a) `yarn.nodemanager.resource.memory-mb` 表示该节点上 YARN 可使用的物理内存总量，默认是 8192 (MB)，注意，如果你的节点内存资源不够 8GB，则需要调减小这个值，而 YARN 不会智能的探测节点的物理内存总量。

(b) `yarn.scheduler.maximum-allocation-mb` 单个任务可申请的最多物理内存量，默认是 8192 (MB)。

3.2.3 Zookeeper 安装

3.2.3.1 Zookeeper 分布式安装部署

1) 集群规划

在 Master、Slave1 和 Slave2 三个节点上部署 Zookeeper。

	服务器 Master	服务器 Slave1	服务器 Slave2
Zookeeper	Zookeeper	Zookeeper	Zookeeper

2) 解压安装

(1) 解压 Zookeeper 安装包到 /opt/module/ 目录下

```
[hadoop@Master software]$ tar -zvxf apache-zookeeper-3.5.7-bin.tar.gz -C
/opt/module/
```

(2) 修改 /opt/module/apache-zookeeper-3.5.7-bin 名称为 zookeeper-3.5.7

```
[hadoop@Master module]$ mv apache-zookeeper-3.5.7-bin/ zookeeper
```

3) 配置服务器编号

(1) 在 /opt/module/zookeeper-3.5.7/ 这个目录下创建 zkData

```
[hadoop@Master zookeeper]$ mkdir zkData
```

(2) 在 /opt/module/zookeeper-3.5.7/zkData 目录下创建一个 myid 的文件

```
[hadoop@Master zkData]$ vim myid
```

添加 myid 文件，注意一定要在 linux 里面创建，在 notepad++ 里面很可能乱码

在文件中添加与 server 对应的编号：

```
2
```

4) 配置 **zoo.cfg** 文件

(1) 重命名 /opt/module/zookeeper-3.5.7/conf 这个目录下的 **zoo_sample.cfg** 为 **zoo.cfg**

```
[hadoop@Master conf]$ mv zoo_sample.cfg zoo.cfg
```

(2) 打开 **zoo.cfg** 文件

```
[hadoop@Master conf]$ vim zoo.cfg
```

修改数据存储路径配置

```
dataDir=/opt/module/zookeeper/zkData
```

增加如下配置

```
#####
#cluster#####
server.2=Master:2888:3888
server.3=Slave1:2888:3888
server.4=Slave2:2888:3888
```

(3) 同步 /opt/module/zookeeper-3.5.7 目录内容到 Slave1、Slave2

```
[hadoop@Master module]$ xsync zookeeper/
```

(4) 分别修改 slave1、Slave2 上的 myid 文件中内容为 3、4

(5) **zoo.cfg** 配置参数解读

```
server.A=B:C:D.
```

A 是一个数字，表示这个是第几号服务器；

集群模式下配置一个文件 myid，这个文件在 dataDir 目录下，这个文件里面有一个数据就是 **A** 的值，Zookeeper 启动时读取此文件，拿到里面的数据与 **zoo.cfg** 里面的配置信息比较从而判断到底是哪个 server。

B 是这个服务器的地址；

C 是这个服务器 Follower 与集群中的 Leader 服务器交换信息的端口；

D 是万一集群中的 Leader 服务器挂了，需要一个端口来重新进行选举，选出一个新的 Leader，而这个端口就是用来执行选举时服务器相互通信的端口。

5) 集群操作

(1) 分别启动 Zookeeper

```
[hadoop@Master zookeeper]$ bin/zkServer.sh start
[hadoop@Slave1 zookeeper]$ bin/zkServer.sh start
[hadoop@Slave2 zookeeper]$ bin/zkServer.sh start
```

(2) 查看状态

```
[hadoop@Master zookeeper]# bin/zkServer.sh status
JMX enabled by default
Using config: /opt/module/zookeeper/bin/../conf/zoo.cfg
Mode: follower
[hadoop@Slave1 zookeeper]# bin/zkServer.sh status
```

```
JMX enabled by default
Using config: /opt/module/zookeeper/bin/../conf/zoo.cfg
Mode: leader
[hadoop@Slave2 zookeeper]# bin/zkServer.sh status
JMX enabled by default
Using config: /opt/module/zookeeper/bin/../conf/zoo.cfg
Mode: follower
```

3.2.3.2 zk 集群启动停止脚本

- (1) 在 Master 的/home/hadoop/bin 目录下创建脚本

```
[hadoop@Master bin]$ vim zk.sh
```

在脚本中编写如下内容

```
#!/bin/bash

case $1 in
"start"){
for i in Master Slave1 Slave2
do
    echo ----- zookeeper $i 启动 -----
    ssh $i "/opt/module/zookeeper/bin/zkServer.sh start"
done
};;
"stop"){
for i in Master Slave1 Slave2
do
    echo ----- zookeeper $i 停止 -----
    ssh $i "/opt/module/zookeeper/bin/zkServer.sh stop"
done
};;
"status"){
for i in Master Slave1 Slave2
do
    echo ----- zookeeper $i 状态 -----
    ssh $i "/opt/module/zookeeper/bin/zkServer.sh status"
done
};;
esac
```

- (2) 增加脚本执行权限

```
[hadoop@Master bin]$ chmod 777 zk.sh
```

- (3) Zookeeper 集群启动脚本

```
[hadoop@Master module]$ zk.sh start
```

- (4) Zookeeper 集群停止脚本

```
[hadoop@Master module]$ zk.sh stop
```

```

hadoop@Master:/opt/module/applog/log$ zk.sh start
----- zookeeper Master 启动 -----
ZooKeeper JMX enabled by default
Using config: /opt/module/zookeeper/bin/..../conf/zoo.cfg
Starting zookeeper ... STARTED
----- zookeeper Slave1 启动 -----
ZooKeeper JMX enabled by default
Using config: /opt/module/zookeeper/bin/..../conf/zoo.cfg
Starting zookeeper ... STARTED
----- zookeeper Slave2 启动 -----
ZooKeeper JMX enabled by default
Using config: /opt/module/zookeeper/bin/..../conf/zoo.cfg
Starting zookeeper ... STARTED
hadoop@Master:/opt/module/applog/log$ jpsall
=====
Master =====
3802 Jps
3708 QuorumPeerMain
=====
Slave1 =====
4372 Jps
4281 QuorumPeerMain
=====
Slave2 =====
3120 QuorumPeerMain
3204 Jps
hadoop@Master:/opt/module/applog/log$ zk.sh stop
----- zookeeper Master 停止 -----
ZooKeeper JMX enabled by default
Using config: /opt/module/zookeeper/bin/..../conf/zoo.cfg
Stopping zookeeper ... STOPPED
----- zookeeper Slave1 停止 -----
ZooKeeper JMX enabled by default
Using config: /opt/module/zookeeper/bin/..../conf/zoo.cfg
Stopping zookeeper ... STOPPED
----- zookeeper Slave2 停止 -----
ZooKeeper JMX enabled by default
Using config: /opt/module/zookeeper/bin/..../conf/zoo.cfg
Stopping zookeeper ... STOPPED

```

客户端命令行操作

命令基本语法	功能描述
help	显示所有操作命令
ls path	使用 ls 命令来查看当前 znode 的子节点 -w 监听子节点变化 -s 附加次级信息
create	普通创建 -s 含有序列 -e 临时（重启或者超时消失）
get path	获得节点的值

	-w 监听节点内容变化 -s 附加次级信息
set	设置节点的具体值
stat	查看节点状态
delete	删除节点
deleteall	递归删除节点

启动客户端

```
[hadoop@Slave1 zookeeper]$ bin/zkCli.sh
```

3.2.4 Kafka 安装

3.2.4.1 Kafka 分布式部署

集群规划

Master	Slave1	Slave2
zk	zk	zk
kafka	kafka	kafka

集群部署

0) 官方下载地址: <http://kafka.apache.org/downloads.html>

1) 解压安装包

```
[hadoop@Master software]$ tar -zxvf kafka_2.12-3.0.0.tgz -C /opt/module/
```

2) 修改解压后的文件名称

```
tguigu@Master module]$ mv kafka_2.12-3.0.0/ kafka
```

3) 进入到/opt/module/kafka 目录, 修改配置文件

```
[hadoop@Master kafka]$ cd config/  
[hadoop@Master config]$ vim server.properties
```

输入以下内容:

```
#broker 的全局唯一编号, 不能重复, 只能是数字。  
broker.id=0  
#处理网络请求的线程数量  
num.network.threads=3  
#用来处理磁盘 IO 的线程数量  
num.io.threads=8  
#发送套接字的缓冲区大小  
socket.send.buffer.bytes=102400  
#接收套接字的缓冲区大小  
socket.receive.buffer.bytes=102400  
#请求套接字的缓冲区大小  
socket.request.max.bytes=104857600  
#kafka 运行日志(数据)存放的路径, 路径不需要提前创建, kafka 自动帮你创建, 可以配置多个磁盘路径,  
路径与路径之间可以用", "分隔  
log.dirs=/opt/module/kafka/datas  
#topic 在当前 broker 上的分区个数  
num.partitions=1  
#用来恢复和清理 data 下数据的线程数量  
num.recovery.threads.per.data.dir=1  
# 每个 topic 创建时的副本数, 默认时 1 个副本  
offsets.topic.replication.factor=1
```

```
#segment 文件保留的最长时间, 超时将被删除
log.retention.hours=168
#每个 segment 文件的大小, 默认最大 1G
log.segment.bytes=1073741824
# 检查过期数据的时间, 默认 5 分钟检查一次是否数据过期
log.retention.check.interval.ms=300000
#配置连接 Zookeeper 集群地址 (在 zk 根目录下创建/kafka, 方便管理)
zookeeper.connect=Master:2181,Slave1:2181,Slave2:2181/kafka
```

4) 分发安装包

```
[hadoop@Master module]$ xsync kafka/
```

5) 分别在 Slave1 和 Slave2 上修改配置文件

/opt/module/kafka/config/server.properties 中的 broker.id=1、broker.id=2

注: broker.id 不得重复, 整个集群中唯一。

```
[hadoop@Slave1 module]$ vim kafka/config/server.properties
```

修改:

```
# The id of the broker. This must be set to a unique integer for each broker.
broker.id=1
```

```
[hadoop@Slave2 module]$ vim kafka/config/server.properties
```

修改:

```
# The id of the broker. This must be set to a unique integer for each broker.
broker.id=2
```

接着配置环境变量并熟悉卡夫卡的启停操作。要先启动 zookeeper 集群, 然后启动 Kafka。

启动 kafka 使用 kafka 根目录下的 **bin/kafka-server-start.sh**; 启动语法要声明设定文件, 如: **bin/kafka-server-start.sh -daemon config/server.properties**。

关闭 kafka 使用 kafka 根目录下的 **bin/kafka-server-stop.sh**。

3.2.4.2 Kafka 集群启停脚本

1) 在/home/hadoop/bin 目录下创建文件 kf.sh 脚本文件

```
[hadoop@Master bin]$ vim kf.sh
```

脚本如下:

```
#!/bin/bash

case $1 in
"start") {
    for i in Master Slave1 Slave2
    do
        echo " -----启动 $i Kafka-----"
        ssh $i "/opt/module/kafka/bin/kafka-server-start.sh -daemon
/opt/module/kafka/config/server.properties"
        done
    ;;
"stop") {
    for i in Master Slave1 Slave2
    do
        echo " -----停止 $i Kafka-----"
        ssh $i "/opt/module/kafka/bin/kafka-server-stop.sh "
        done
    ;;
esac
```

2) 添加执行权限

```
[hadoop@Master bin]$ chmod +x kf.sh
```

3) 启动集群命令

```
[hadoop@Master ~]$ kf.sh start
```

4) 停止集群命令

```
[hadoop@Master ~]$ kf.sh stop
```

注意: 停止 Kafka 集群时,一定要等 Kafka 所有节点进程全部停止后再停止 Zookeeper 集群。因为 Zookeeper 集群当中记录着 Kafka 集群相关信息, Zookeeper 集群一旦先停止, Kafka 集群就没有办法再获取停止进程的信息,只能手动杀死 Kafka 进程了。

```

hadoop@Master:/opt/module/applog/log$ kf.sh start
----- 启动 Master kafka -----
----- 启动 Slave1 kafka -----
----- 启动 Slave2 kafka -----
hadoop@Master:/opt/module/applog/log$ jpsall
===== Master =====
4000 QuorumPeerMain
4478 Kafka
4622 Jps
===== Slave1 =====
5010 Kafka
4548 QuorumPeerMain
5131 Jps
===== Slave2 =====
3381 QuorumPeerMain
3948 Jps
3836 Kafka
hadoop@Master:/opt/module/applog/log$ kf.sh stop
----- 关闭 Master kafka -----
----- 关闭 Slave1 kafka -----
----- 关闭 Slave2 kafka -----

```

3.2.4.3 Kafka 命令行操作

主题命令行操作

1) 查看操作主题命令参数

```
[hadoop@Master kafka]$ bin/kafka-topics.sh
```

参数	描述
--bootstrap-server <String: server to connect to>	连接的 Kafka Broker 主机名称和端口号。
--topic <String: topic>	操作的 topic 名称。
--create	创建主题。
--delete	删除主题。
--alter	修改主题。
--list	查看所有主题。
--describe	查看主题详细描述。

--partitions <Integer: # of partitions>	设置分区数。
--replication-factor<Integer: replication factor>	设置分区副本。
--config <String: name=value>	更新系统默认的配置。

2) 查看当前服务器中的所有 topic

```
[hadoop@Master kafka]$ bin/kafka-topics.sh --bootstrap-server Master:9092 --list
```

3) 创建 first topic

```
[hadoop@Master kafka]$ bin/kafka-topics.sh --bootstrap-server Master:9092 --create --partitions 1 --replication-factor 3 --topic first
```

选项说明:

--topic 定义 topic 名

--replication-factor 定义副本数

--partitions 定义分区数

4) 查看 first 主题的详情

```
[hadoop@Master kafka]$ bin/kafka-topics.sh --bootstrap-server Master:9092 --describe --topic first
```

5) 修改分区数 (注意: 分区数只能增加, 不能减少)

```
[hadoop@Master kafka]$ bin/kafka-topics.sh --bootstrap-server Master:9092 --alter --topic first --partitions 3
```

6) 再次查看 first 主题的详情

```
[hadoop@Master kafka]$ bin/kafka-topics.sh --bootstrap-server Master:9092 --describe --topic first
```

7) 删除 topic

```
[hadoop@Master kafka]$ bin/kafka-topics.sh --bootstrap-server Master:9092 --delete --topic first
```

生产者命令行操作

1) 查看操作生产者命令参数

```
[hadoop@Master kafka]$ bin/kafka-console-producer.sh
```

参数	描述
--bootstrap-server <String: server to connect to>	连接的 Kafka Broker 主机名称和端口号。
--topic <String: topic>	操作的 topic 名称。

2) 发送消息

```
[hadoop@Master kafka]$ bin/kafka-console-producer.sh --bootstrap-server Master:9092 --topic first
>hello world
>hadoop hadoop
```

消费者命令行操作

1) 查看操作消费者命令参数

```
[hadoop@Master kafka]$ bin/kafka-console-consumer.sh
```

参数	描述
--bootstrap-server <String: server to connect to>	连接的 Kafka Broker 主机名称和端口号。
--topic <String: topic>	操作的 topic 名称。
--from-beginning	从头开始消费。
--group <String: consumer group id>	指定消费者组名称。

2) 消费消息

(1) 消费 first 主题中的数据。

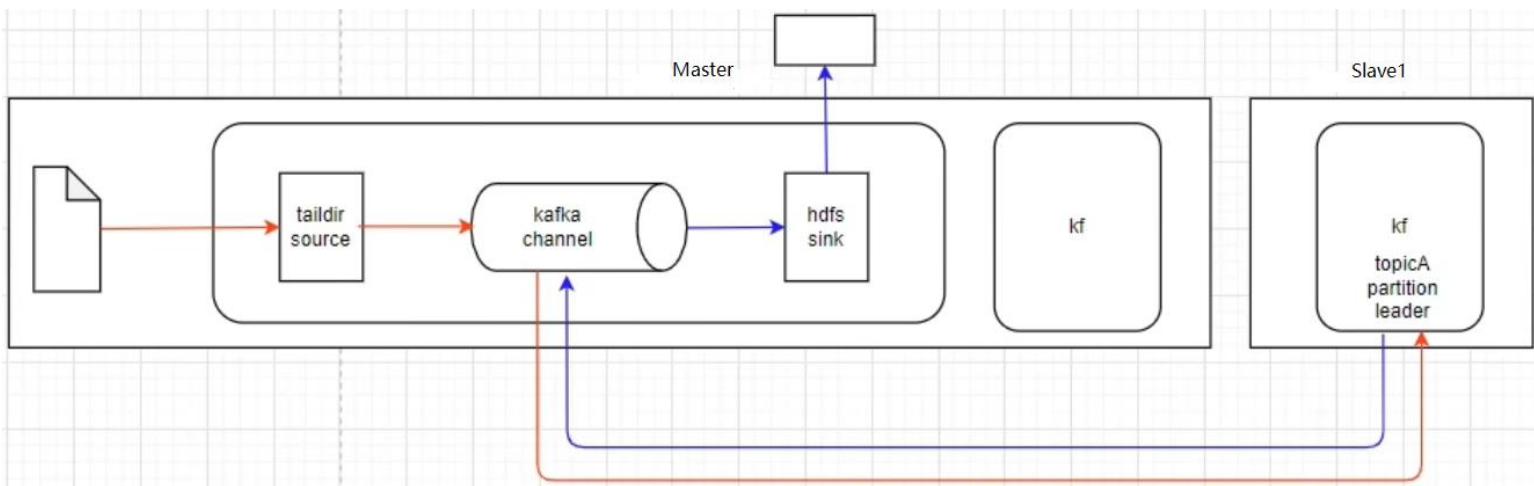
```
[hadoop@Master kafka]$ bin/kafka-console-consumer.sh --bootstrap-server Master:9092 --topic first
```

(2) 把主题中所有的数据都读取出来 (包括历史数据)。

```
[hadoop@Master kafka]$ bin/kafka-console-consumer.sh --bootstrap-server Master:9092 --from-beginning --topic first
```

3.2.5 Flume 通道规划安装

Flume 的通道规划如下。



- 使用 Flume 将日志文件 (log 文件) 传送到 kafka
- 使用 Flume 将 kafka 内的日志文件传送到 HDFS
- 使用 Flume 将 Kafka 内的业务数据 (由 Maxwell 生成) 文件传送到 HDFS

按照采集通道规划，需在 Master, Slave1, Slave2 三台节点分别部署一个 Flume。参照以下步骤先在 Master 安装，然后再进行分发。

安装地址

- (1) Flume 官网地址: <http://flume.apache.org/>
- (2) 文档查看地址: <http://flume.apache.org/FlumeUserGuide.html>
- (3) 下载地址: <http://archive.apache.org/dist/flume/>

安装部署

- (1) 将 apache-flume-1.9.0-bin.tar.gz 上传到 linux 的 /opt/software 目录下
- (2) 解压 apache-flume-1.9.0-bin.tar.gz 到 /opt/module/ 目录下

```
[hadoop@Master software]$ tar -zxf /opt/software/apache-flume-1.9.0-bin.tar.gz -C /opt/module/
```

(3) 修改 apache-flume-1.9.0-bin 的名称为 flume

```
[hadoop@Master module]$ mv /opt/module/apache-flume-1.9.0-bin /opt/module/flume
```

(4) 将 lib 文件夹下的 guava-11.0.2.jar 删除以兼容 Hadoop 3.1.3

```
[hadoop@Master module]$ rm /opt/module/flume/lib/guava-11.0.2.jar
```

注意：删除 guava-11.0.2.jar 的服务器节点，一定要配置 hadoop 环境变量。否则会报如下异常。

```
Caused by: java.lang.ClassNotFoundException: com.google.common.collect.Lists
        at java.net.URLClassLoader.findClass(URLClassLoader.java:382)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:349)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
        ... 1 more
```

(5) 修改 conf 目录下的 log4j.properties 配置文件，配置日志文件路径

```
[hadoop@Master conf]$ vim log4j.properties
flume.log.dir=/opt/module/flume/logs
```

分发 Flume

```
[hadoop@Master ~]$ xsync /opt/module/flume/
```

3.3 日志采集 Flume

3.3.1 日志采集 Flume 配置概述

按照规划，需要采集的用户行为日志文件分布在 Master, Slave1 两台日志服务器，故需要在 Master, Slave1 两台节点配置日志采集 Flume。日志采集 Flume 需要采集日志文件内容，并对日志格式（JSON）进行校验，然后将校验通过的日志发送到 Kafka。此处可选择 TailDirSource 和 KafkaChannel，并配置日志校验拦截器。选择 TailDirSource 和 KafkaChannel 的原因如下：

1) TailDirSource

- TailDirSource 相比 ExecSource、SpoolingDirectorySource 的优势
- TailDirSource：断点续传、多目录。Flume1.6 以前需要自己自定义 Source 记录每次读取文件位置，实现断点续传。
- ExecSource 可以实时搜集数据，但是在 Flume 不运行或者 Shell 命令出错的情况下，数据将会丢失。
- SpoolingDirectorySource 监控目录，支持断点续传。

2) KafkaChannel

- 采用 Kafka Channel，省去了 Sink，提高了效率。

日志采集 Flume 关键配置如下：

TailDirSource

```
#通过正则表达式匹配需要采集的日志文件
a1.sources.r1.filegroups=f1
a1.sources.r1.filegroups.f1=/opt/module/applog/log/app.*

#配置ETL拦截器，对非法数据进行过滤
interceptors=i1
interceptors.i1.type=ETLInterceptor.Builder
```

KafkaChannel

```
#指定数据要发往的目标Topic
a1.channels.c1.kafka.topic = topic_log
```

3.3.2 日志采集 Flume 配置实操

1) 创建 Flume 配置文件

在 Master 节点的 Flume 的 job 目录下创建 file_to_kafka.conf

```
[hadoop@Slave2 flume]$ mkdir job
[hadoop@Slave2 flume]$ vim job/file_to_kafka.conf
```

2) 配置文件内容如下

```
#定义组件
a1.sources = r1
a1.channels = c1

#配置 source
a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/module/applog/log/app.*
a1.sources.r1.positionFile = /opt/module/flume/taildir_position.json
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type =
com.mymall.flume.interceptor.ETLInterceptor$Builder

#配置 channel
a1.channels.c1.type = org.apache.flume.channel.kafka.KafkaChannel
a1.channels.c1.kafka.bootstrap.servers = Master:9092,Slave1:9092
a1.channels.c1.kafka.topic = topic_log
a1.channels.c1.parseAsFlumeEvent = false

#组装
a1.sources.r1.channels = c1
```

3) 编写 Flume 拦截器

具体代码参考下面的文件



Flume拦截器.doc

x

所有 Flume 通道的拦截器写好后都打包在一个 jar 包里 flume-interceptor-1.0-SNAPSHOT-jar-with-dependencies.jar。分发到 Master、Slave1、Slave2 三台机器上。

3.3.3 日志采集 Flume 测试

- 1) 启动 Zookeeper、Kafka 集群
- 2) 启动 Master 的日志采集 Flume

```
[hadoop@Master flume]$ bin/flume-ng agent -n a1 -c conf/ -f job/file_to_kafka.conf
-Dflume.root.logger=info,console
```

3) 启动一个 Kafka 的 Console-Consumer

```
[hadoop@Master kafka]$ bin/kafka-console-consumer.sh --bootstrap-server Master:9092
--topic topic_log
```

4) 生成模拟数据

```
[hadoop@Master ~]$ log.sh
```

5) 观察 Kafka 消费者是否能消费到数据

```
hadoop@Slave1:/opt/module/kafka$ bin/kafka-console-consumer.sh --bootstrap-server Master:9092 --topic topic_log
{"common": {"ar": "440000", "ba": "iPhone", "ch": "Appstore", "is_new": "1", "md": "iPhone X", "mid": "mid_553043", "os": "iOS 13.2.3", "uid": "687", "vc": "v2.1.134"}, "start": {"entry": "icon", "loading_time": 11873, "open_ad_id": 7, "open_ad_ms": 4495, "open_ad_skip_ms": 2154}, "ts": 1592117509000}
{"common": {"ar": "440000", "ba": "iPhone", "ch": "Appstore", "is_new": "1", "md": "iPhone X", "mid": "mid_553043", "os": "iOS 13.2.3", "uid": "687", "vc": "v2.1.134"}, "displays": [{"display_type": "activity", "item": "1", "item_type": "activity_id", "order": 1, "pos_id": 2}, {"display_type": "activity", "item": "2", "item_type": "activity_id", "order": 2, "pos_id": 2}, {"display_type": "query", "item": "2", "item_type": "sku_id", "order": 3, "pos_id": 5}, {"display_type": "query", "item": "2", "item_type": "sku_id", "order": 4, "pos_id": 2}, {"display_type": "recommend", "item": "14", "item_type": "sku_id", "order": 5, "pos_id": 5}, {"display_type": "query", "item": "17", "item_type": "sku_id", "order": 6, "pos_id": 3}, {"display_type": "promotion", "item": "24", "item_type": "sku_id", "order": 7, "pos_id": 4}, {"display_type": "promotion", "item": "35", "item_type": "sku_id", "order": 8, "pos_id": 4}, {"display_type": "promotion", "item": "29", "item_type": "sku_id", "order": 9, "pos_id": 4}], "page": {"during_time": 10718, "page_id": "home"}, "ts": 1592117509000}
```

3.3.4 日志采集 Flume 启停脚本

1) 方便起见，此处编写一个日志采集 Flume 进程的启停脚本

在 Master 节点的 /home/hadoop/bin 目录下创建脚本 f1.sh

```
[hadoop@Master bin]$ vim f1.sh
```

在脚本中填写如下内容

```
#!/bin/bash

case $1 in
"start"){
    for i in Master Slave1
    do
        echo " -----启动 $i 采集 flume-----"
        ssh $i "nohup /opt/module/flume/bin/flume-ng agent -n a1 -c
/opt/module/flume/conf/ -f /opt/module/flume/job/file_to_kafka.conf >/dev/null 2>&1
&" 
        done
    ;;
"stop"){
    for i in Master Slave1
```

```

do
    echo " -----停止 $i 采集 flume-----"
    ssh $i "ps -ef | grep file_to_kafka | grep -v grep | awk '{print \$2}' | xargs -n1 kill -9 "
    done

} ;;
esac

```

2) 增加脚本执行权限

```
[hadoop@Master bin]$ chmod 777 f1.sh
```

3) f1 启动

```
[hadoop@Master module]$ f1.sh start
```

4) f1 停止

```
[hadoop@Master module]$ f1.sh stop
```

```

hadoop@Master:/opt/module/flume$ f1.sh start
-----启动 Master 采集flume-----
-----启动 Slave1 采集flume-----
hadoop@Master:/opt/module/flume$ jpsall
===== Master =====
5505 Kafka
6146 Jps
5026 QuorumPeerMain
6006 Application
===== Slave1 =====
6068 ConsoleConsumer
5877 Kafka
5416 QuorumPeerMain
6620 Jps
6492 Application
===== Slave2 =====
5010 Jps
4327 QuorumPeerMain
4782 Kafka
hadoop@Master:/opt/module/flume$ f1.sh stop
-----停止 Master 采集flume-----
-----停止 Slave1 采集flume-----

```

第二章 业务数据采集平台

1. 电商业务简介

1.1 电商业务流程

电商的业务流程可以以一个普通用户的浏览足迹为例进行说明，用户点开电商首页开始浏览，可能会通过分类查询也可能通过全文搜索寻找自己中意的商品，这些商品无疑都是存储在后台的管理系统中的。

当用户寻找到自己中意的商品，可能会想要购买，将商品添加到购物车后发现需要登录，登录后对商品进行结算，这时候购物车的管理和商品订单信息的生成都会对业务数据库产生影响，会生成相应的订单数据和支付数据。

订单正式生成之后，还会对订单进行跟踪处理，直到订单全部完成。

电商的主要业务流程包括用户前台浏览商品时的商品详情的管理，用户商品加入购物车进行支付时用户个人中心&支付服务的管理，用户支付完成后订单后台服务的管理，这些流程涉及到了十几个甚至几十个业务数据表，甚至更多。

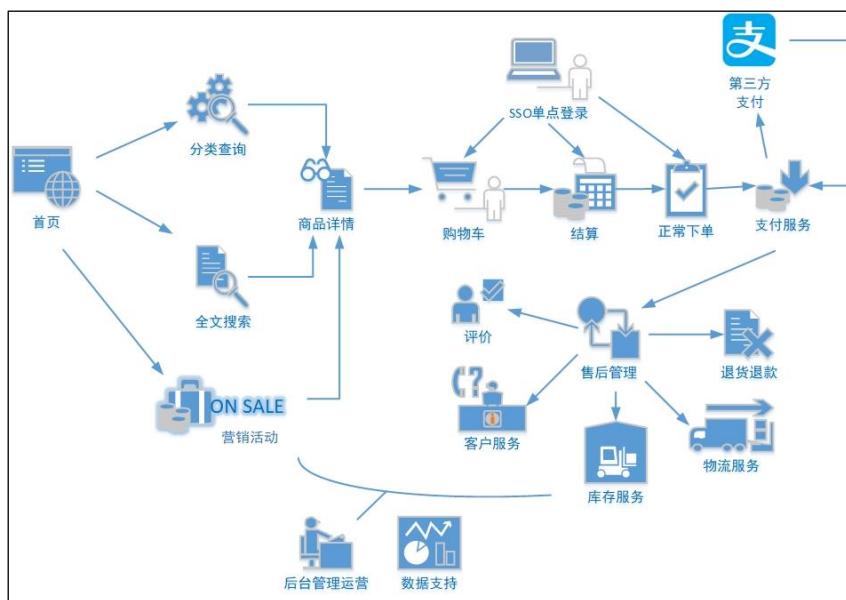


图 电商业务流程

1.2 电商常识

1.2.1 SKU 和 SPU

SKU = Stock Keeping Unit (库存量基本单位)。现在已经被引申为产品统一编号的简称，每种产品均对应有唯一的 SKU 号。

SPU (Standard Product Unit): 是商品信息聚合的最小单位，是一组可复用、易检索的标准信息集合。

例如：iPhoneX 手机就是 SPU。一台银色、128G 内存的、支持联通网络的 iPhoneX，就是 SKU。



图 SKU、PKU 示意

SKU 表示一类商品。同一 SKU 的商品可以共用商品图片、海报、销售属性等。

1.2.2 平台属性和销售属性

1) 平台属性

2) 销售属性



2. 业务数据介绍

2.1 电商系统表结构

以下为本电商数仓系统涉及到的业务数据表结构关系。这 34 个表以订单表、用户表、SKU 商品表、活动表和优惠券表为中心，延伸出了优惠券领用表、支付流水表、活动订单表、订单详情表、订单状态表、商品评论表、编码字典表退单表、SPU 商品表等，用户表提供用户的详细信息，支付流水表提供该订单的支付详情，订单详情表提供订单的商品数量等情况，商品表给订单详情表提供商品的详细信息。本次讲解以此 34 个表为例，实际项目中，业务数据库中表格远远不止这些。

示例：活动信息表 (`activity_info`)

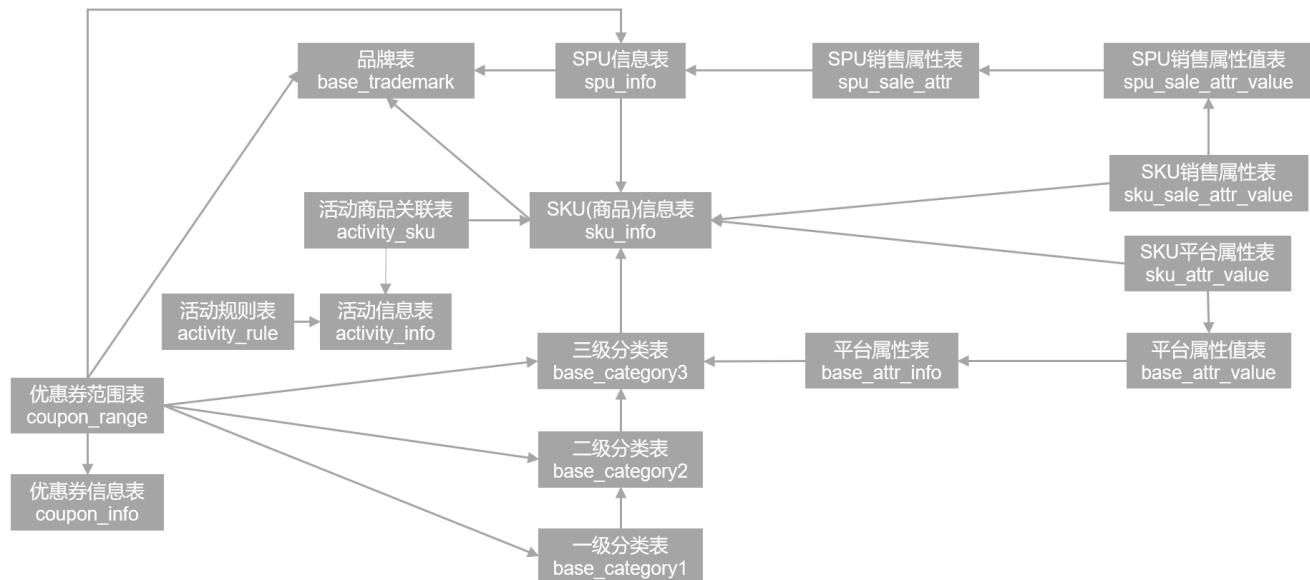
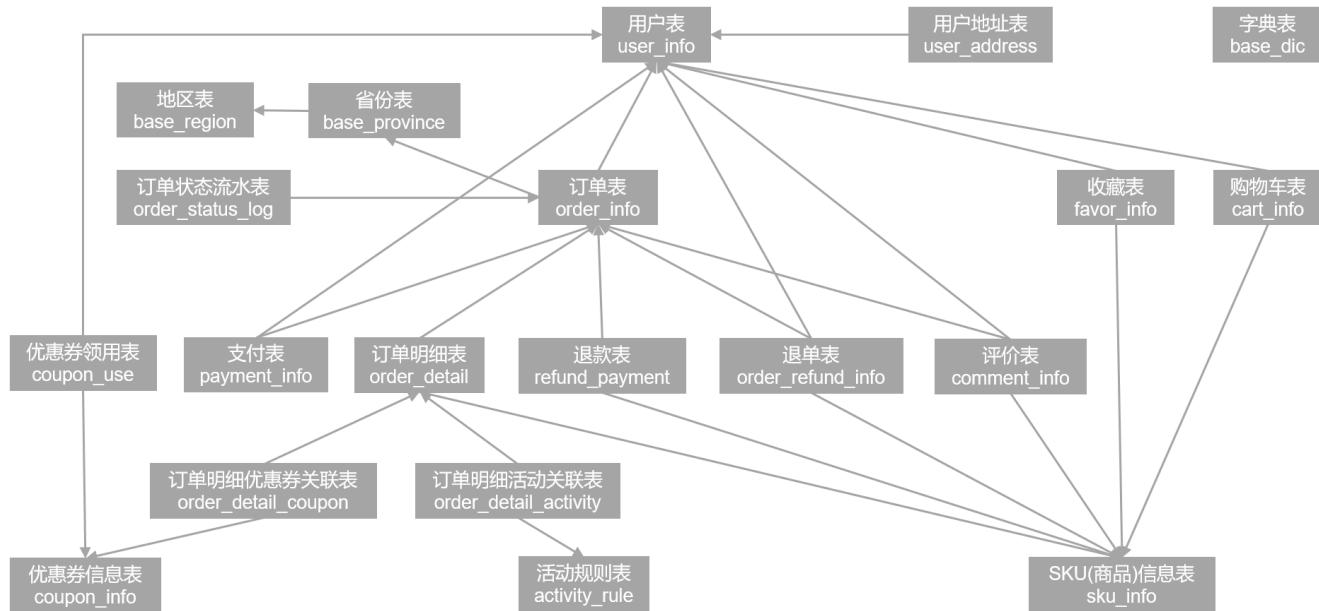
字段名	字段说明
<code>id</code>	活动 id
<code>activity_name</code>	活动名称
<code>activity_type</code>	活动类型 (1: 满减, 2: 折扣)
<code>activity_desc</code>	活动描述
<code>start_time</code>	开始时间
<code>end_time</code>	结束时间
<code>create_time</code>	创建时间

具体的所有表格结构参见下面这个文件：



业务数据表结构.d
OCX

各个业务表之间的联系如下：



2.2 MySQL 安装

使用 Ubuntu 自带的命令进行安装

```
[hadoop@Master ~]$ sudo apt-get update
[hadoop@Master ~]$ sudo apt-get install mysql-server
[hadoop@Master ~]$ sudo mysql_secure_installation
[hadoop@Master ~]$ sudo apt-get install mysql-client
[hadoop@Master ~]$ sudo apt-get install libmysql-java
```

启动 MySQL

```
[hadoop@Master ~]$ sudo systemctl start mysqld
```

查看 MySQL 密码

```
[hadoop@Master ~]$ sudo cat /var/log/mysqld.log | grep password
```

配置 MySQL

配置只要是 root 用户 + 密码，在任何主机上都能登录 MySQL 数据库。

1) 用刚刚查到的密码进入 MySQL（如果报错，给密码加单引号）

```
[hadoop@Master ~]$ mysql -uroot -p'password'
```

2) 设置复杂密码（由于 MySQL 密码策略，此密码必须足够复杂）

```
mysql> set password=password("Qs23=zs32");
```

3) 更改 MySQL 密码策略

```
mysql> set global validate_password_length=4;
mysql> set global validate_password_policy=0;
```

4) 设置简单好记的密码

```
mysql> set password=password("yudingyi");
```

5) 进入 MySQL 库

```
mysql> use mysql
```

6) 查询 user 表

```
mysql> select user, host from user;
```

7) 修改 user 表，把 Host 表内容修改为%

```
mysql> update user set host=% where user=root;
```

8) 刷新

```
mysql> flush privileges;
```

9) 退出

```
mysql> quit;
```

2.2 业务数据模拟

2.2.1 连接 MySQL

通过 MySQL 可视化客户端连接数据库。在这里使用破解版的 SQLyog。

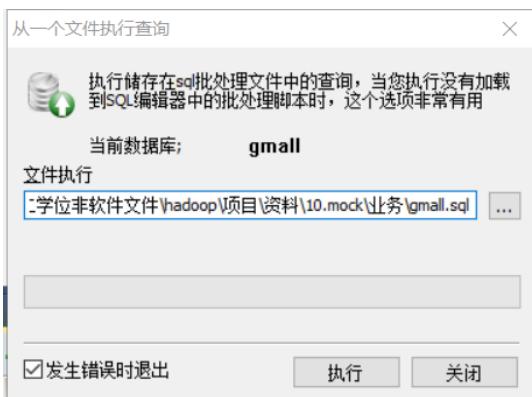


2.2.2 建表语句

1) 通过 SQLyog 创建数据库, 设置数据库名称为 `gmall`, 编码为 `utf-8`, 排序规则为 `utf8_general_ci`



2) 导入数据库结构脚本 (`gmall.sql`) 该文件是从网上搜集的, 放在了附件里。



注意: 完成后, 右键刷新一下对象浏览器, 就可以看见数据库中的表了。

2.2.3 生成业务数据

1) 在 Master 的 `/opt/module/` 目录下创建 `db_log` 文件夹

```
[hadoop@Master module]$ mkdir db_log/
```

2) 把 `gmall2020-mock-db-2021-11-14.jar` 和 `application.properties` 上传到 Master 的 `/opt/module/db_log` 路径上。这些文件是从网上搜集的, 放在了附件里。

3) 根据需求修改 `application.properties` 相关配置

```
logging.level.root=info

spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://Master:3306/gmall?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=GMT%2B8
spring.datasource.username=root
spring.datasource.password=yudingyi

logging.pattern.console=%m%n

mybatis-plus.global-config.db-config.field-strategy=not_null
```

```

#业务日期
mock.date=2020-06-14
#是否重置 注意: 第一次执行必须设置为 1, 后续不需要重置不用设置为 1
mock.clear=1
#是否重置用户 注意: 第一次执行必须设置为 1, 后续不需要重置不用设置为 1
mock.clear.user=1

#生成新用户数量
mock.user.count=100
#男性比例
mock.user.male-rate=20
#用户数据变化概率
mock.user.update-rate:20

#收藏取消比例
mock.favor.cancel-rate=10
#收藏数量
mock.favor.count=100

#每个用户添加购物车的概率
mock.cart.user-rate=50
#每次每个用户最多添加多少种商品进购物车
mock.cart.max-sku-count=8
#每个商品最多买几个
mock.cart.max-sku-num=3

#购物车来源 用户查询, 商品推广, 智能推荐, 促销活动
mock.cart.source-type-rate=60:20:10:10

#用户下单比例
mock.order.user-rate=50
#用户从购物中购买商品比例
mock.order.sku-rate=50
#是否参加活动
mock.order.join-activity=1
#是否使用购物券
mock.order.use-coupon=1
#购物券领取人数
mock.coupon.user-count=100

#支付比例
mock.payment.rate=70
#支付方式 支付宝: 微信 : 银联
mock.payment.payment-type=30:60:10

#评价比例 好: 中: 差: 自动
mock.comment.appraise-rate=30:10:10:50

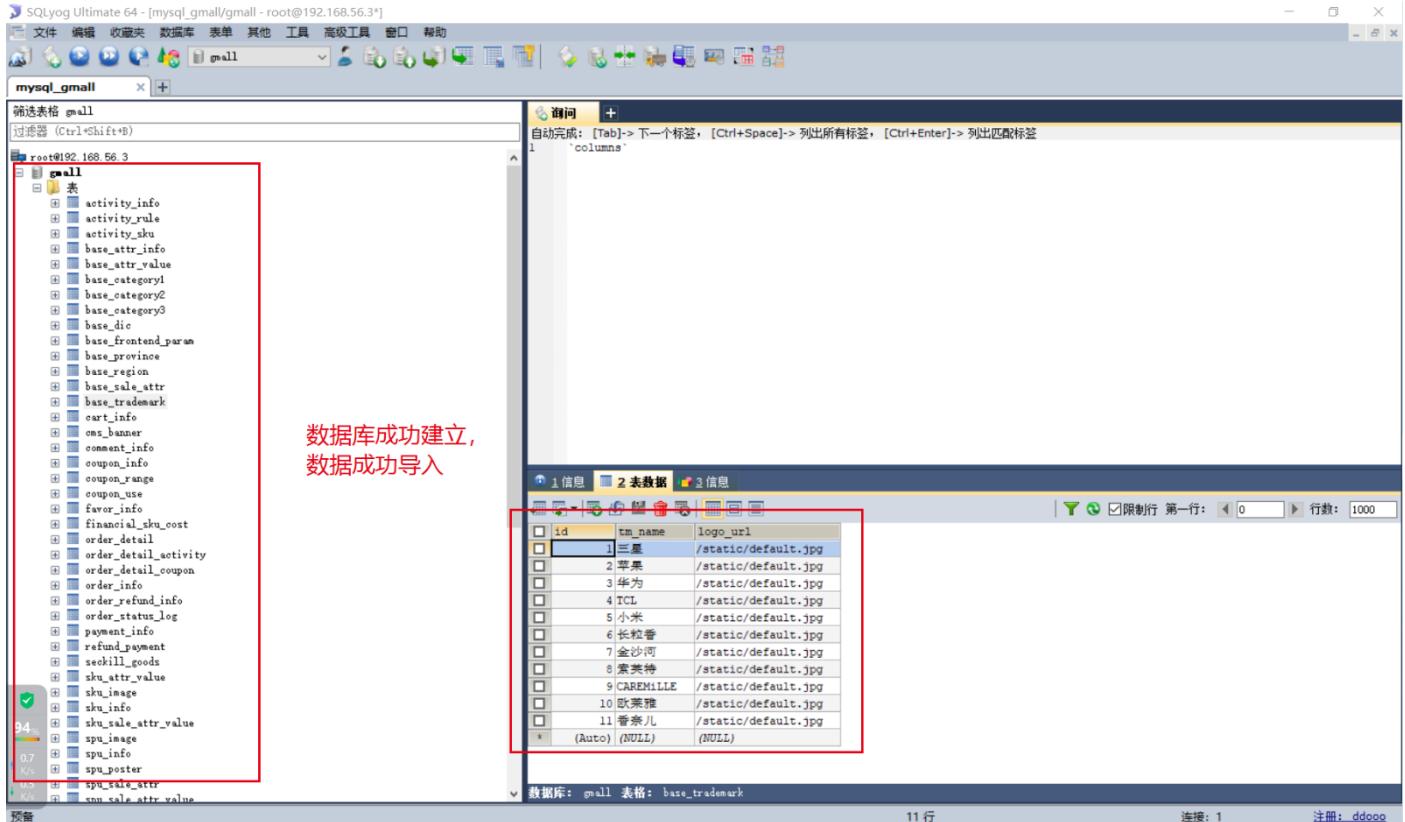
#退款原因比例: 质量问题 商品描述与实际描述不一致 缺货 号码不合适 拍错 不想买了 其他
mock.refund.reason-rate=30:10:20:5:15:5:5

```

4) 并在该目录下执行, 如下命令, 生成 2020-06-14 日期数据:

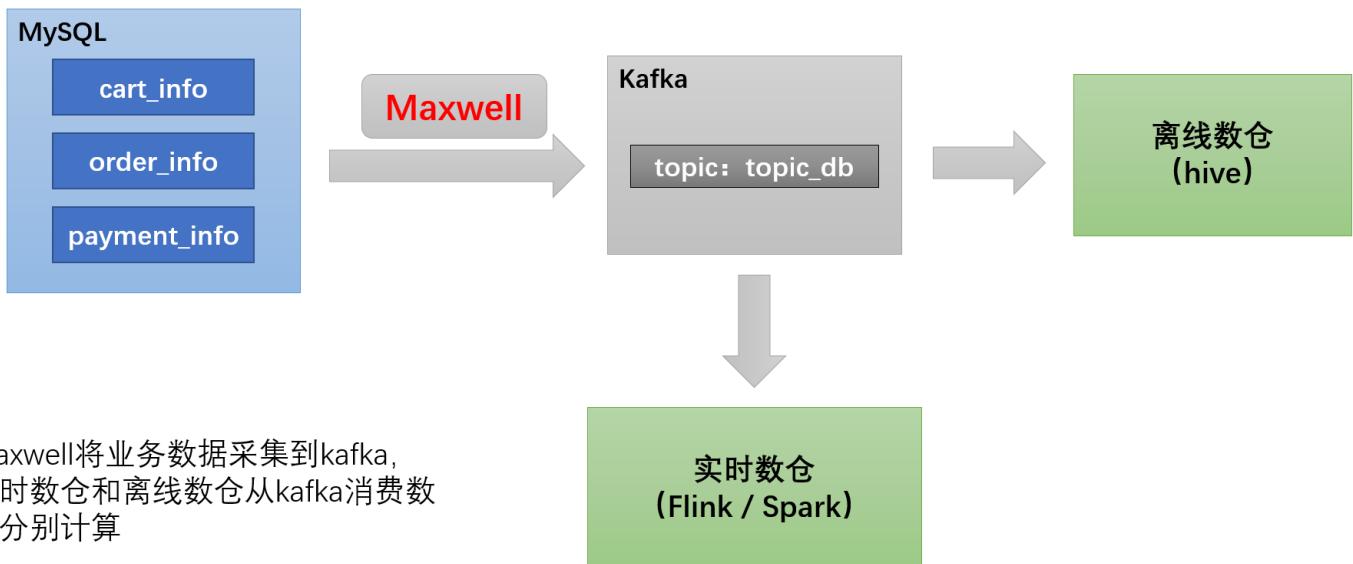
```
[hadoop@Master db_log]$ java -jar gmall2020-mock-db-2021-11-14.jar
```

5) 查看 gmall 数据库, 观察是否有 2020-06-14 的数据出现



3. 业务数据采集模块

3.1 采集通道



3.2 采集工具

使用 Maxwell 进行数据采集，Maxwell 原理与背景相关介绍可见另一份 word [资料文档](#)。

3.2.1 安装 Maxwell

1) 下载安装包

(1) 地址:

<https://github.com/zendesk/maxwell/releases/download/v1.29.2/maxwell-1.29.2.tar.gz>

注：Maxwell-1.30.0 及以上版本不再支持 JDK1.8。

(2) 将安装包上传到 Master 节点的 /opt/software 目录

注：此处使用教学版安装包，教学版对原版进行了改造，增加了自定义 Maxwell 输出数据中 ts 时间戳的参数，生产环境请使用原版。

2) 将安装包解压至 /opt/module

```
[hadoop@Master maxwell]$ tar -zxvf maxwell-1.29.2.tar.gz -C /opt/module/
```

3) 修改名称

```
[hadoop@Master module]$ mv maxwell-1.29.2/ maxwell
```

3.2.2 配置 MySQL

3.2.2.1 启用 MySQL Binlog

MySQL 服务器的 Binlog 默认是未开启的，如需进行同步，需要先进行开启。

1) 修改 MySQL 配置文件 /etc/my.cnf

```
[hadoop@Master ~]$ sudo vim /etc/my.cnf
```

2) 增加如下配置

```
[mysqld]
#数据库 id
server-id = 1
#启动 binlog，该参数的值会作为 binlog 的文件名
log-bin=mysql-bin
#binlog 类型，maxwell 要求为 row 类型
binlog_format=row
#启用 binlog 的数据库，需根据实际情况作出修改
binlog-do-db=gmail
```

注：MySQL Binlog 模式

Statement-based： 基于语句，Binlog 会记录所有写操作的 SQL 语句，包括 insert、update、delete 等。

- 优点：节省空间
- 缺点：有可能造成数据不一致，例如 insert 语句中包含 now() 函数。
Row-based： 基于行，Binlog 会记录每次写操作后被操作行记录的变化。
- 优点：保持数据的绝对一致性。
- 缺点：占用较大空间。

Mixed： 混合模式，默认是 Statement-based，如果 SQL 语句可能导致数据不一致，就自动切换到 Row-based。

Maxwell 要求 Binlog 采用 Row-based 模式。

3) 重启 MySQL 服务

```
[hadoop@Master ~]$ sudo systemctl restart mysqld
```

3.2.2.2 创建 Maxwell 所需数据库和用户

Maxwell 需要在 MySQL 中存储其运行过程中的所需的一些数据，包括 binlog 同步的断点位置（Maxwell 支持断点续传）等等，故需要在 MySQL 为 Maxwell 创建数据库及用户。

1) 创建数据库

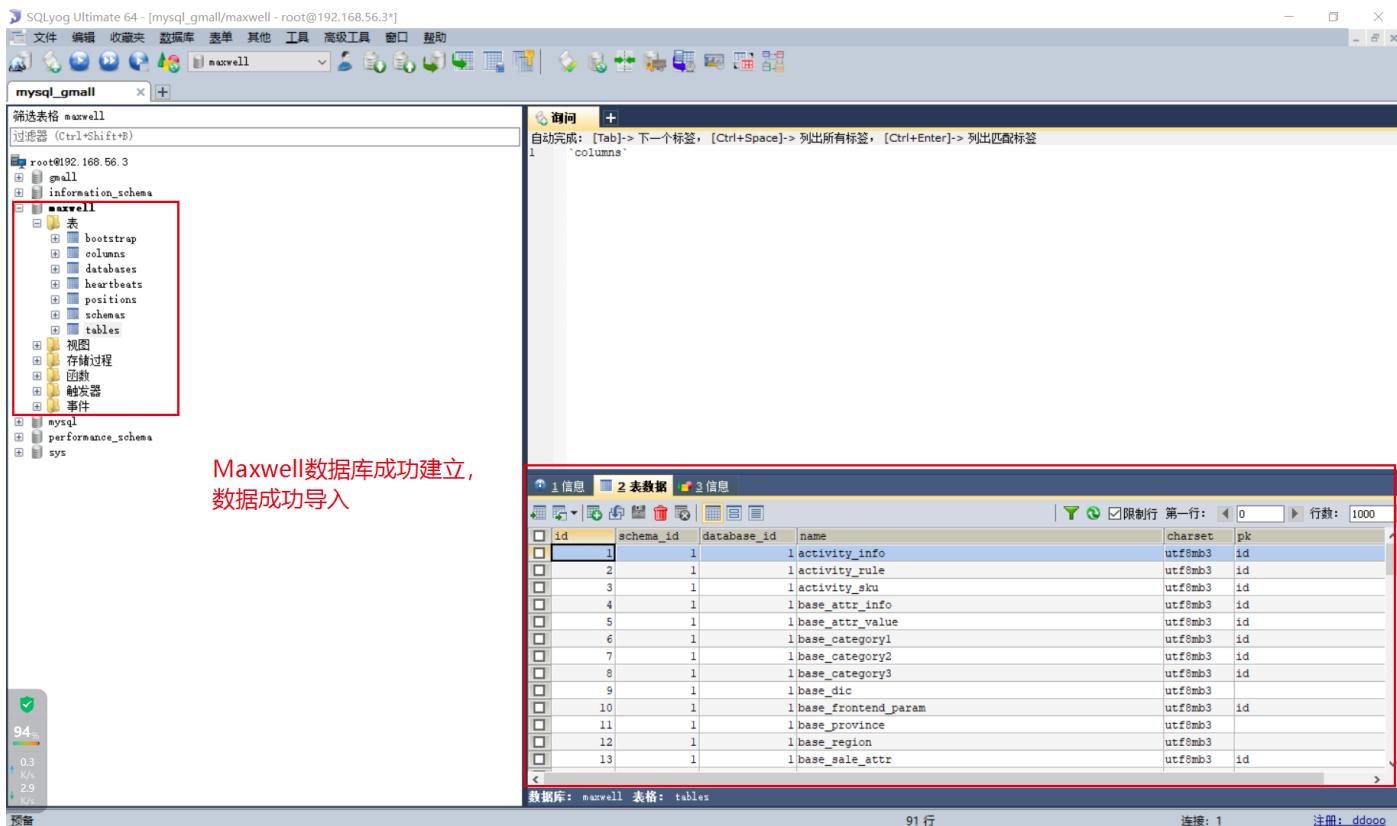
```
mysql> CREATE DATABASE maxwell;
```

2) 调整 MySQL 数据库密码级别

```
mysql> set global validate_password_policy=0;
mysql> set global validate_password_length=4;
```

3) 创建 Maxwell 用户并赋予其必要权限

```
mysql> CREATE USER 'maxwell'@'%' IDENTIFIED BY 'maxwell';
mysql> GRANT ALL ON maxwell.* TO 'maxwell'@'%';
mysql> GRANT SELECT, REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'maxwell'@'%';
```



3.2.3 配置 Maxwell

1) 修改 Maxwell 配置文件名称

```
[hadoop@Master maxwell]$ cd /opt/module/maxwell
[hadoop@Master maxwell]$ cp config.properties.example config.properties
```

2) 修改 Maxwell 配置文件

```
[hadoop@Master maxwell]$ vim config.properties
```

```
#Maxwell 数据发送目的地, 可选配置有 stdout|file|kafka|kinesis|pubsub|sqrs|rabbitmq|redis
producer=kafka
#目标 Kafka 集群地址
kafka.bootstrap.servers= Master:9092,Slave1:9092
#目标 Kafka topic, 可静态配置, 例如:maxwell, 也可动态配置, 例如: ${database}_${table}
kafka_topic=topic_db

#MySQL 相关配置
host=Master
user=maxwell
password=maxwell
```

```
jdbc_options=useSSL=false&serverTimezone=Asia/Shanghai
```

3.3 Maxwell 使用

3.3.1 启动 Kafka 集群

若 Maxwell 发送数据的目的地为 Kafka 集群，则需要先确保 Kafka 集群为启动状态。

3.3.2 Maxwell 启停

1) 启动 Maxwell

```
[hadoop@Master ~]$ /opt/module/maxwell/bin/maxwell --config
/opt/module/maxwell/config.properties --daemon
```

2) 停止 Maxwell

```
[hadoop@Master ~]$ ps -ef | grep maxwell | grep -v grep | grep maxwell | awk
'{print $2}' | xargs kill -9
```

3) Maxwell 启停脚本

(1) 创建并编辑 Maxwell 启停脚本

```
[hadoop@Master bin]$ vim mxw.sh
```

(2) 脚本内容如下

```
#!/bin/bash

MAXWELL_HOME=/opt/module/maxwell

status_maxwell(){
    result=`ps -ef | grep com.zendesk.maxwell | grep -v grep | wc -l`
    return $result
}

start_maxwell(){
    status_maxwell
    if [[ $? -lt 1 ]]; then
        echo "启动 Maxwell"
        $MAXWELL_HOME/bin/maxwell --config $MAXWELL_HOME/config.properties --daemon
    else
        echo "Maxwell 正在运行"
    fi
}

stop_maxwell(){
    status_maxwell
    if [[ $? -gt 0 ]]; then
        echo "停止 Maxwell"
        ps -ef | grep com.zendesk.maxwell | grep -v grep | awk '{print $2}' |
        xargs kill -9
    else
        echo "Maxwell 未在运行"
    fi
}

case $1 in
    start )
        start_maxwell
    ;;
    stop )
```

```

    stop_maxwell
;;
restart )
    stop_maxwell
    start_maxwell
;;
esac

```

```

hadoop@Master:/opt/module/applog/log$ mxw.sh start
启动Maxwell
Redirecting STDOUT to /opt/module/maxwell/bin/../logs/MaxwellDaemon.out
Using kafka version: 1.0.0
hadoop@Master:/opt/module/applog/log$ jpsall
=====
Master =====
4000 QuorumPeerMain
4715 Maxwell
4813 Jps
=====
Slave1 =====
4548 QuorumPeerMain
5238 Jps
=====
Slave2 =====
3381 QuorumPeerMain
4054 Jps
hadoop@Master:/opt/module/applog/log$ mxw.sh stop
停止Maxwell

```

3.3.3 增量数据同步

1) 启动 Zookeeper、Kafka 集群以及 Maxwell

```
[hadoop@Master ~]$ zk.sh start
[hadoop@Master ~]$ kf.sh start
[hadoop@Master ~]$ mxw.sh start
```

2) 启动 Kafka 消费者

```
[hadoop@Slave1 kafka]$ bin/kafka-console-consumer.sh --bootstrap-server Master:9092
--topic topic_db
```

3) 模拟生成数据

```
[hadoop@Master db_log]$ java -jar gmall2020-mock-db-2021-01-22.jar
```

4) 观察 Kafka 消费者是否能消费到数据

```
{"database":"gmall","table":"cart_info","type":"update","ts":1592270938,"xid":13090
,"xoffset":1573,"data": {"id":100924,"user_id":"93","sku_id":16,"cart_price":4488.00
,"sku_num":1,"img_url":"http://47.93.148.192:8080/group1/M00/00/02/rBHu81-
sklaALrngAAHGDqdpFtU741.jpg","sku_name":"华为 HUAWEI P40 麒麟990 5G SoC 芯片 5000 万超
感知徕卡三摄 30 倍数字变焦 8GB+128GB 亮黑色全网通 5G 手机
","","is_checked":null,"create_time":"2020-06-14
09:28:57","operate_time":null,"is_ordered":1,"order_time":"2021-10-17
09:28:58","source_type":2401,"source_id":null}, "old": {"is_ordered":0,"order_time":null}}
```

3.3.4 历史数据全量同步

有时只有增量数据是不够的，我们可能需要使用到 MySQL 数据库中从历史至今的一个完整的数据集。这就需要我们在进行增量同步之前，先进行一次历史数据的全量同步。这样就能保证得到一个完整的数据集。

Maxwell-bootstrap

Maxwell 提供了 bootstrap 功能来进行历史数据的全量同步，命令如下：

```
[hadoop@Master maxwell]$ /opt/module/maxwell/bin/maxwell-bootstrap --database gmail
--table user_info --config /opt/module/maxwell/config.properties
```

bootstrap 数据格式

采用 bootstrap 方式同步的输出数据格式如下：

```
{
  "database": "fooDB",
  "table": "barTable",
  "type": "bootstrap-start",
  "ts": 1450557744,
  "data": {}

}

{
  "database": "fooDB",
  "table": "barTable",
  "type": "bootstrap-insert",
  "ts": 1450557744,
  "data": {
    "txt": "hello"
  }
}

{
  "database": "fooDB",
  "table": "barTable",
  "type": "bootstrap-insert",
  "ts": 1450557744,
  "data": {
    "txt": "bootstrap!"
  }
}

{
  "database": "fooDB",
  "table": "barTable",
  "type": "bootstrap-complete",
  "ts": 1450557744,
  "data": {}
}
```

注意事项：

- 1) 第一条 type 为 bootstrap-start 和最后一条 type 为 bootstrap-complete 的数据，是 bootstrap 开始和结束的标志，不包含数据，中间的 type 为 bootstrap-insert 的数据才包含数据。
- 2) 一次 bootstrap 输出的所有记录的 ts 都相同，为 bootstrap 开始的时间。

第三章 数仓数据同步策略

1. 实时数仓同步数据

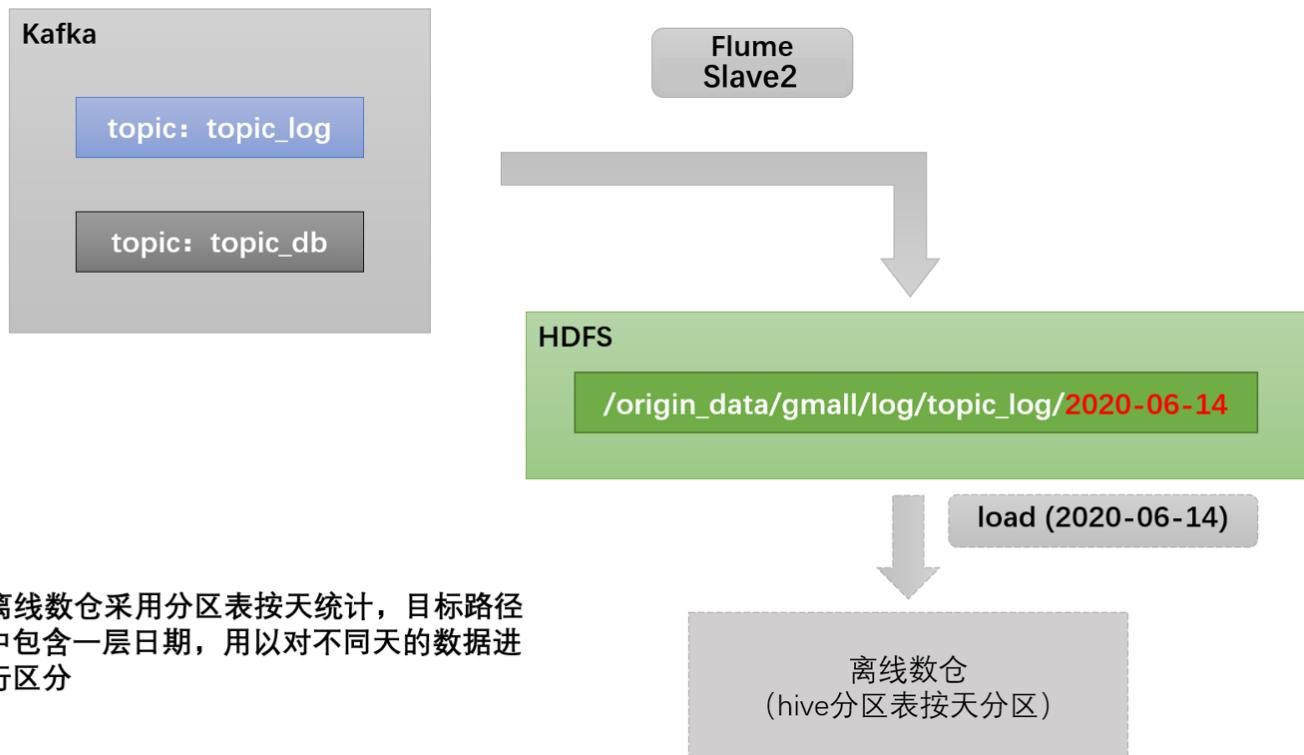
实时数仓由 Flink 源源不断从 Kafka 当中读数据计算，所以不需要手动同步数据到实时数仓。

2. 离线数仓同步数据

2.1 用户行为数据同步

2.1.1 数据通道

用户行为数据由 Flume 从 Kafka 直接同步到 HDFS，由于离线数仓采用 Hive 的分区表按天统计，所以目标路径要包含一层日期。具体数据流向如下图所示。



2.1.2 日志消费 Flume 配置概述

按照规划，该 Flume 需将 Kafka 中 topic_log 的数据发往 HDFS。并且对每天产生的用户行为日志进行区分，将不同天的数据发往 HDFS 不同天的路径。

此处选择 **KafkaSource**、**FileChannel**、**HDFSSink**。

关键配置如下：

```

KafkaSource
#订阅Kafka中的topic_log
a1.sources.r1.kafka.topics=topic_log

#使用时间截拦截器为Event增加一个header, key为
#timestamp, value为json字符串中ts字段的值
interceptors = i1
interceptors.i1.type=TimeStampInterceptor.Builder

```

FileChannel

```

HDFSSink
#path中包含时间转移序列，用于将不同日期的数据
#放到不同的路径
path=/origin_data/gmail/log/topic_log/%Y-%m-%d

```

2.1.3 日志消费 Flume 配置实操

1) 创建 Flume 配置文件

在 Slave2 节点的 Flume 的 job 目录下创建 kafka_to_hdfs_log.conf

```
[hadoop@Slave2 flume]$ vim job/kafka_to_hdfs_log.conf
```

2) 配置文件内容如下

```
#定义组件
a1.sources=r1
a1.channels=c1
a1.sinks=k1

#配置 source1
a1.sources.r1.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.r1.batchSize = 5000
a1.sources.r1.batchDurationMillis = 2000
a1.sources.r1.kafka.bootstrap.servers = Master:9092,Slave1:9092,Slave2:9092
a1.sources.r1.kafka.topics=topic_log
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type =
com.mymall.flume.interceptor.TimestampInterceptor$Builder

#配置 channel
a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /opt/module/flume/checkpoint/behavior1
a1.channels.c1.dataDirs = /opt/module/flume/data/behavior1
a1.channels.c1.maxFileSize = 2146435071
a1.channels.c1.capacity = 1000000
a1.channels.c1.keep-alive = 6

#配置 sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = /origin_data/gmall/log/topic_log/%Y-%m-%d
a1.sinks.k1.hdfs.filePrefix = log
a1.sinks.k1.hdfs.round = false

a1.sinks.k1.hdfs.rollInterval = 10
a1.sinks.k1.hdfs.rollSize = 134217728
a1.sinks.k1.hdfs.rollCount = 0

#控制输出文件类型
a1.sinks.k1.hdfs.fileType = CompressedStream
a1.sinks.k1.hdfs.codec = gzip

#组装
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

注：配置优化

1) FileChannel 优化

通过配置 `dataDirs` 指向多个路径，每个路径对应不同的硬盘，增大 Flume 吞吐量。

官方说明如下：

```
Comma separated list of directories for storing log files. Using multiple
directories on separate disks can improve file channel performance
checkpointDir 和 backupCheckpointDir 也尽量配置在不同硬盘对应的目录中，保证
checkpoint 坏掉后，可以快速使用 backupCheckpointDir 恢复数据
```

2) HDFS Sink 优化

(1) HDFS 存入大量小文件，有什么影响？

元数据层面：每个小文件都有一份元数据，其中包括文件路径，文件名，所有者，所属组，权限，创建时间等，这些信息都保存在 Namenode 内存中。所以小文件过多，会占用 Namenode 服务器大量内存，影响 Namenode 性能和使用寿命

计算层面：默认情况下 MR 会对每个小文件启用一个 Map 任务计算，非常影响计算性能。同时也影响磁盘寻址时间。

(2) HDFS 小文件处理

官方默认的这三个参数配置写入 HDFS 后会产生小文件，`hdfs.rollInterval`、`hdfs.rollSize`、`hdfs.rollCount`

基于以上 `hdfs.rollInterval=3600`, `hdfs.rollSize=134217728`, `hdfs.rollCount=0` 几个参数综合作用，效果如下：

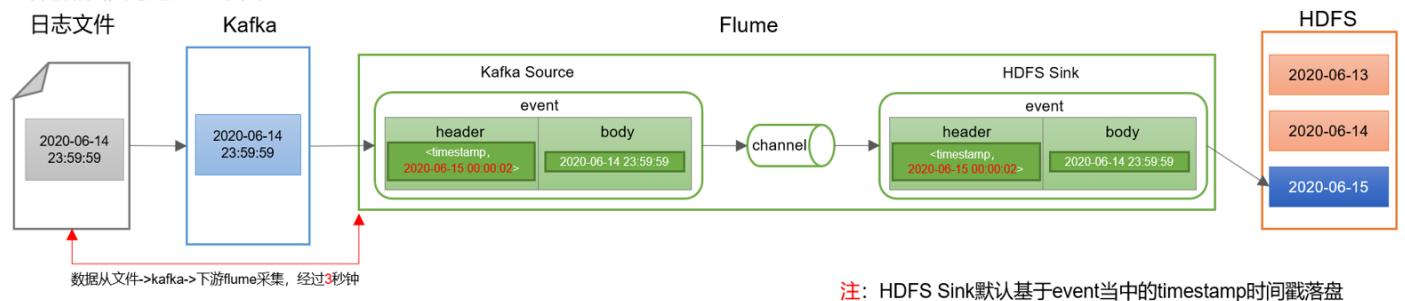
(1) 文件在达到 128M 时会滚动生成新文件

(2) 文件创建超 3600 秒时会滚动生成新文件

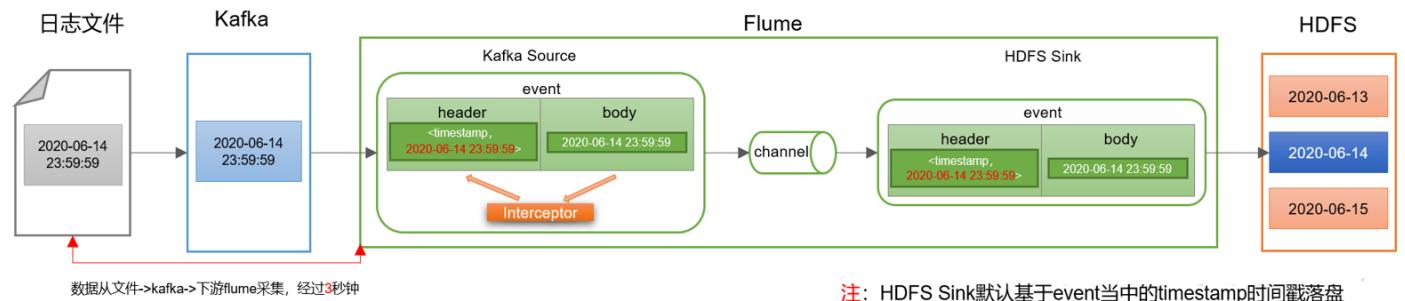
3) 编写 Flume 拦截器

(1) 数据漂移问题

数据漂移问题产生原因



如何解决数据漂移问题？



(2) 编写拦截器

具体代码参考下面的文件



所有 Flume 通道的拦截器写好后都打包在一个 jar 包里 `flume-interceptor-1.0-SNAPSHOT-jar-with-dependencies.jar`。分发到 Master、Slave1、Slave2 三台机器上。

2.1.4 日志消费 Flume 测试

1) 启动 Zookeeper、Kafka 集群

2) 启动日志采集 Flume

```
[hadoop@Master ~]$ f1.sh start
```

3) 启动 Slave2 的日志消费 Flume

```
[hadoop@Slave2 flume]$ bin/flume-ng agent -n a1 -c conf/ -f job/kafka_to_hdfs_log.conf -Dflume.root.logger=info,console
```

4) 生成模拟数据

```
[hadoop@Master ~]$ log.sh
```

5) 观察 HDFS 是否出现数据

```
2023-05-10 16:40:32,234 (hdfs-k1-roll-timer-0) [INFO - org.apache.flume.sink.hdfs.HDFSEventSink$1.run(HDFSEventSink.java:393)] Writer callback called.
2023-05-10 16:40:32,234 (hdfs-k1-roll-timer-0) [INFO - org.apache.flume.sink.hdfs.BucketWriter.doClose(BucketWriter.java:438)] Closing /origin_data/gmail/log/topic_log/2020-06-14/1683708022198.gz.tmp
2023-05-10 16:40:32,265 (hdfs-k1-call-runner-9) [INFO - org.apache.flume.sink.hdfs.BucketWriter$7.call(BucketWriter.java:681)] Renaming /origin_data/gmail/log/topic_log/2020-06-14/Log.1683708022198.gz.tmp to /origin_data/gmail/log/topic_log/2020-06-14/log.1683708022198.gz
2023-05-10 16:40:37,685 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.hdfs.HDFSCompressedDataStream.configure(HDFSCompressedDataStream.java:64)] SerializedName = TEXT, UserRawLocalFileSystem = false
2023-05-10 16:40:37,697 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.hdfs.BucketWriter.open(BucketWriter.java:246)] Creating /origin_data/gmail/log/topic_log/2020-06-14/1683708037686.gz.tmp
2023-05-10 16:40:37,774 (Thread-14) [INFO - org.apache.hadoop.hdfs.protocol.datatransfer.SaslDataTransferClient.checkTrustAndSend(SaslDataTransferClient.java:239)] SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-05-10 16:40:47,728 (hdfs-k1-roll-timer-0) [INFO - org.apache.flume.sink.hdfs.HDFSEventSink$1.run(HDFSEventSink.java:393)] Writer callback called.
2023-05-10 16:40:47,728 (hdfs-k1-roll-timer-0) [INFO - org.apache.flume.sink.hdfs.BucketWriter.doClose(BucketWriter.java:438)] Closing /origin_data/gmail/log/topic_log/2020-06-14/1683708037686.gz.tmp
2023-05-10 16:40:47,759 (hdfs-k1-call-runner-3) [INFO - org.apache.flume.sink.hdfs.BucketWriter$7.call(BucketWriter.java:681)] Renaming /origin_data/gmail/log/topic_log/2020-06-14/Log.1683708037686.gz.tmp to /origin_data/gmail/log/topic_log/2020-06-14/log.1683708037686.gz
2023-05-10 16:41:00,217 (Log-BackgroundWorker-c1) [INFO - org.apache.flume.channel.file.EventQueueBackingStoreFile.beginCheckpoint(EventQueueBackingStoreFile.java:230)] Start checkpoint for /opt/module/flume/checkpoint/behavior1/checkpoint, elements to sync = 509
2023-05-10 16:41:00,226 (Log-BackgroundWorker-c1) [INFO - org.apache.flume.channel.file.EventQueueBackingStoreFile.checkpoint(EventQueueBackingStoreFile.java:255)] Updating checkpoint metadata: logWriteOrderID: 1683707916095, queueSize: 0, queueHead: 11217
2023-05-10 16:41:00,240 (Log-BackgroundWorker-c1) [INFO - org.apache.flume.channel.file.Log.writeCheckpoint(Log.java:1065)] Updated checkpoint for file: /opt/module/flume/data/behavior1/log-5 position: 2071775 logWriteOrderID: 1683707916095
```

Browse Directory

/origin_data/gmail/log/topic_log/2020-06-14							Go!			
Show 25 entries		Search:								
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name		
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	66.62 KB	May 10 16:40	2	128 MB	log.1683708005427.gz		
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	51.97 KB	May 10 16:40	2	128 MB	log.1683708022198.gz		
<input type="checkbox"/>	-rw-r--r--	hadoop	supergroup	10.25 KB	May 10 16:40	2	128 MB	log.1683708037686.gz		

Showing 1 to 3 of 3 entries

Previous 1 Next

Hadoop, 2019.

2.1.5 日志消费 Flume 启停脚本

1) 在 Master 节点的/home/hadoop/bin 目录下创建脚本 f2.sh

```
[hadoop@Master bin]$ vim f2.sh
```

在脚本中填写如下内容

```
#!/bin/bash

case $1 in
"start")
    echo " -----启动 Slave2 日志数据 flume-----"
    ssh Slave2 "nohup /opt/module/flume/bin/flume-ng agent -n a1 -c
/opt/module/flume/conf -f /opt/module/flume/job/kafka_to_hdfs_log.conf >/dev/null
2>&1 &""
;;
"stop")
    echo " -----停止 Slave2 日志数据 flume-----"
    ssh Slave2 "ps -ef | grep kafka_to_hdfs_log | grep -v grep | awk '{print
\$2}' | xargs -n1 kill"
;;
esac
```

2) 增加脚本执行权限

```
[hadoop@Master bin]$ chmod 777 f2.sh
```

3) f2 启动

```
[hadoop@Master module]$ f2.sh start
```

4) f2 停止

```
[hadoop@Master module]$ f2.sh stop
```

```
hadoop@Master:/opt/module/flume$ f2.sh start
-----启动 Slave2 日志数据flume-----
hadoop@Master:/opt/module/flume$ jpsall
===== Master =====
5505 Kafka
5026 QuorumPeerMain
6667 NameNode
6973 JobHistoryServer
13597 Jps
===== Slave1 =====
8978 Jps
5877 Kafka
5416 QuorumPeerMain
7304 NodeManager
6986 DataNode
7167 ResourceManager
===== Slave2 =====
5330 SecondaryNameNode
4327 QuorumPeerMain
5435 NodeManager
7740 Application
7869 Jps
5197 DataNode
4782 Kafka
hadoop@Master:/opt/module/flume$ f2.sh stop
-----停止 Slave2 日志数据flume-----
```

2.2 业务数据同步

2.2.1 数据同步策略

业务数据是数据仓库的重要数据来源，我们需要每日定时从业务数据库中抽取数据，传输到数据仓库中，之后再对数据进行分析统计。

为保证统计结果的正确性，需要保证数据仓库中的数据与业务数据库是同步的，离线数仓的计算周期通常为天，所以数据同步周期也通常为天，即每天同步一次即可。

数据的同步策略有**全量同步**和**增量同步**。

全量同步，就是每天都将业务数据库中的全部数据同步一份到数据仓库，这是保证两侧数据同步的最简单的方式。

增量同步，就是每天只将业务数据中的新增及变化数据同步到数据仓库。采用每日增量同步的表，通常需要在首日先进行一次全量同步。

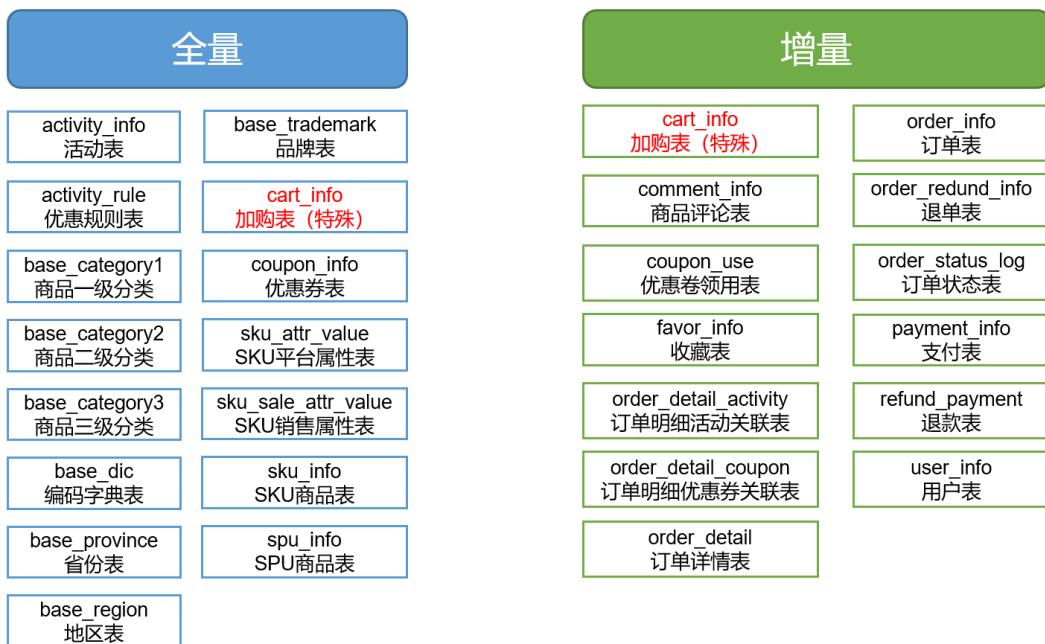
两种策略都能保证数据仓库和业务数据库的数据同步，那应该如何选择呢？下面对两种策略进行简要对比。

同步策略	优点	缺点
全量同步	逻辑简单	在某些情况下效率较低。例如某张表数据量较大，但是每天数据的变化比例很低，若对其采用每日全量同步，则会重复同步和存储大量相同的数据。
增量同步	效率高，无需同步和存储重复数据	逻辑复杂，需要将每日的新增及变化数据同原来的数据进行整合，才能使用

根据上述对比，可以得出以下结论：

通常情况，业务表数据量比较大，优先考虑增量，数据量比较小，优先考虑全量；具体选择由数仓模型决定，此处暂不详解。

下图为各表同步策略：**(网上资源)**



2.2.2 数据同步工具概述

数据同步工具种类繁多，大致可分为两类，一类是以 DataX、Sqoop 为代表的基于 Select 查询的离线、批量同步工具，另一类是以 Maxwell、Canal 为代表的基于数据库数据变更日志（例如 MySQL 的 binlog，其会实时记录所有的 insert、update 以及 delete 操作）的实时流式同步工具。

全量同步通常使用 DataX、Sqoop 等基于查询的离线同步工具。而增量同步既可以使用 DataX、Sqoop 等工具，也可使用 Maxwell、Canal 等工具，下面对增量同步不同方案进行简要对比。

增量同步方案	DataX/Sqoop	Maxwell/Canal
对数据库的要求	原理是基于查询，故若想通过 select 查询获取新增及变化数据，就要求数据表中存在 create_time、update_time 等字段，然后根据这些字段获取变更数据。	要求数据库记录变更操作，例如 MySQL 需开启 binlog。
数据的中间状态	由于是离线批量同步，故若一条数据在一天中变化多次，该方案只能获取最后一个状态，中间状态无法获取。	由于是实时获取所有的数据变更操作，所以可以获取变更数据的所有中间状态。

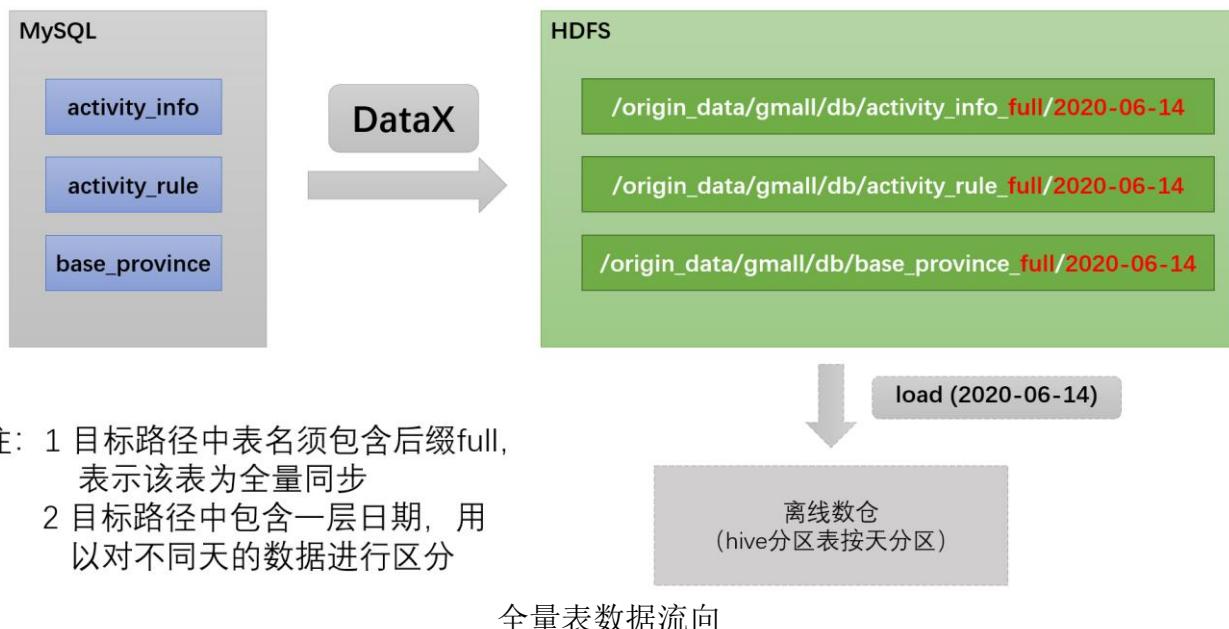
本项目中，全量同步采用 DataX，增量同步采用 Maxwell。

2.2.3 全量表数据同步

2.2.3.1 数据同步工具 DataX 部署

2.2.3.2 数据通道

全量表数据由 DataX 从 MySQL 业务数据库直接同步到 HDFS，具体数据流向如下图所示。



2.2.3.3 DataX 配置文件

我们需要为每张全量表编写一个 DataX 的 json 配置文件，此处以 `activity_info` 为例，配置文件内容如下：

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "mysqlreader",
          "parameter": {
            "column": [
              "id",
              "activity_name",
              "activity_type",
              "activity_desc",
              "start_time",
              "end_time",
              "create_time"
            ],
            "connection": [
              {
                "jdbcUrl": [
                  "jdbc:mysql://Master:3306/gmall"
                ],
                "table": [
                  "activity_info"
                ]
              }
            ],
            "password": "000000",
            "splitPk": "",
            "username": "root"
          }
        },
        "writer": {
          "name": "hivewriter",
          "parameter": {
            "table": "activity_info"
          }
        }
      }
    ]
  }
}
```

```

    "writer": {
        "name": "hdfswriter",
        "parameter": {
            "column": [
                {
                    "name": "id",
                    "type": "bigint"
                },
                {
                    "name": "activity_name",
                    "type": "string"
                },
                {
                    "name": "activity_type",
                    "type": "string"
                },
                {
                    "name": "activity_desc",
                    "type": "string"
                },
                {
                    "name": "start_time",
                    "type": "string"
                },
                {
                    "name": "end_time",
                    "type": "string"
                },
                {
                    "name": "create_time",
                    "type": "string"
                }
            ],
            "compress": "gzip",
            "defaultFS": "hdfs://Master:8020",
            "fieldDelimiter": "\t",
            "fileName": "activity_info",
            "fileType": "text",
            "path": "${targetdir}",
            "writeMode": "append"
        }
    },
    "setting": {
        "speed": {
            "channel": 1
        }
    }
}
}

```

注：由于目标路径包含一层日期，用于对不同天的数据加以区分，故 path 参数并未写死，需在提交任务时通过参数动态传入，参数名称为 **targetdir**。

2.2.3.4 DataX 配置文件生成脚本

方便起见，此处提供了 DataX 配置文件批量生成脚本，脚本内容及使用方式如下。

1) 在~/bin 目录下创建 gen_import_config.py 脚本

```
[hadoop@Master bin]$ vim ~/bin/gen_import config.py
```

脚本内容如下

```
# This is a sample Python script.
# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions,
and settings.
# encoding=utf-8
import json
import getopt
import os
import sys
import mysql.connector

#MySQL 相关配置，需根据实际情况作出修改
mysql_host = "Master"
mysql_port = "3306"
mysql_user = "root"
mysql_passwd = "yudingyi"

#HDFS NameNode 相关配置，需根据实际情况作出修改
hdfs_nn_host = "Master"
hdfs_nn_port = "8020"

#生成配置文件的目标路径，可根据实际情况作出修改
output_path = '/home/hadoop/sda4/module/datex/job/import'

def get_connection():
    return mysql.connector.connect(host=mysql_host, port=int(mysql_port),
user=mysql_user, passwd=mysql_passwd)

def get_mysql_meta(database, table):
    connection = get_connection()
    cursor = connection.cursor()
    sql = "SELECT COLUMN_NAME,DATA_TYPE from information_schema.COLUMNS WHERE
TABLE_SCHEMA=%s AND TABLE_NAME=%s ORDER BY ORDINAL_POSITION"
    cursor.execute(sql, [database, table])
    fetchall = cursor.fetchall()
    cursor.close()
    connection.close()
    return fetchall

def get_mysql_columns(database, table):
    #return map(lambda x: x[0], get_mysql_meta(database, table))
    return list(map(lambda x: x[0], get_mysql_meta(database, table)))

def get_hive_columns(database, table):
    def type_mapping(mysql_type):
        mappings = {
            "bigint": "bigint",
            "int": "bigint",
            "smallint": "bigint",
            "tinyint": "bigint",
            "decimal": "string",
            "double": "double",
            "float": "float",
            "binary": "string",
            "char": "string",
            "varchar": "string",
        }
        return mappings.get(mysql_type, "string")
```

```

        "datetime": "string",
        "time": "string",
        "timestamp": "string",
        "date": "string",
        "text": "string"
    }
    return mappings[mysql_type]

meta = get_mysql_meta(database, table)
return list(map(lambda x: {"name": x[0], "type": type_mapping(x[1].decode("utf-8").lower())}, meta))

#return map(lambda x: {"name": x[0], "type": type_mapping(x[1].lower())}, meta)
#return list(map(lambda x: {"name": x[0], "type": type_mapping(x[1].lower())}, meta))

def generate_json(source_database, source_table):
    job = {
        "job": {
            "setting": {
                "speed": {
                    "channel": 3
                },
                "errorLimit": {
                    "record": 0,
                    "percentage": 0.02
                }
            },
            "content": [
                {
                    "reader": {
                        "name": "mysqlreader",
                        "parameter": {
                            "username": mysql_user,
                            "password": mysql_passwd,
                            "column": get_mysql_columns(source_database, source_table),
                            "splitPk": "",
                            "connection": [
                                {
                                    "table": [source_table],
                                    "jdbcUrl": ["jdbc:mysql://" + mysql_host + ":" +
mysql_port + "/" + source_database]
                                }
                            ]
                        }
                    },
                    "writer": {
                        "name": "hdfswriter",
                        "parameter": {
                            "defaultFS": "hdfs://" + hdfs_nn_host + ":" + hdfs_nn_port,
                            "fileType": "text",
                            "path": "${targetdir}",
                            "fileName": source_table,
                            "column": get_hive_columns(source_database, source_table),
                            "writeMode": "append",
                            "fieldDelimiter": "\t",
                            "compress": "gzip"
                        }
                    }
                }
            ]
        }
    }
    if not os.path.exists(output_path):
        os.makedirs(output_path)

```

```

        with open(os.path.join(output_path, ".".join([source_database, source_table,
"json"])), "w") as f:
            json.dump(job, f)

def main(args):
    source_database = ""
    source_table = ""

    options, arguments = getopt.getopt(args, '-d:-t:', ['sourcedb=', 'sourcetbl='])
    for opt_name, opt_value in options:
        if opt_name in ('-d', '--sourcedb'):
            source_database = opt_value
        if opt_name in ('-t', '--sourcetbl'):
            source_table = opt_value

    generate_json(source_database, source_table)

if __name__ == '__main__':
    main(sys.argv[1:])

# See PyCharm help at https://www.jetbrains.com/help/pycharm/

```

注:

(1) 安装 Python Mysql 驱动

由于需要使用 Python 访问 Mysql 数据库, 故需安装驱动: 如 mysql.connector

(2) 脚本使用说明

```
Python3 gen_import_config.py -d database -t table
```

通过-d 传入数据库名, -t 传入表名, 执行上述命令即可生成该表的 DataX 同步配置文件。

2) 在~/bin 目录下创建 gen_import_config.sh 脚本

```
[hadoop@Master bin]$ vim ~/bin/gen_import_config.sh
```

脚本内容如下

```
#!/bin/bash

Python3 ~/bin/gen_import_config.py -d gmall -t activity_info
Python3 ~/bin/gen_import_config.py -d gmall -t activity_rule
Python3 ~/bin/gen_import_config.py -d gmall -t base_category1
Python3 ~/bin/gen_import_config.py -d gmall -t base_category2
Python3 ~/bin/gen_import_config.py -d gmall -t base_category3
Python3 ~/bin/gen_import_config.py -d gmall -t base_dic
Python3 ~/bin/gen_import_config.py -d gmall -t base_province
Python3 ~/bin/gen_import_config.py -d gmall -t base_region
Python3 ~/bin/gen_import_config.py -d gmall -t base_trademark
Python3 ~/bin/gen_import_config.py -d gmall -t cart_info
Python3 ~/bin/gen_import_config.py -d gmall -t coupon_info
Python3 ~/bin/gen_import_config.py -d gmall -t sku_attr_value
Python3 ~/bin/gen_import_config.py -d gmall -t sku_info
Python3 ~/bin/gen_import_config.py -d gmall -t sku_sale_attr_value
Python3 ~/bin/gen_import_config.py -d gmall -t spu_info
```

3) 为 gen_import_config.sh 脚本增加执行权限

```
[hadoop@Master bin]$ chmod 777 ~/bin/gen_import_config.sh
```

4) 执行 gen_import_config.sh 脚本, 生成配置文件

```
[hadoop@Master bin]$ gen_import_config.sh
```

5) 观察生成的配置文件

```
[hadoop@Master bin]$ ll /home/hadoop/sda4/module/datax/job/import/
```

```

hadoop@Master:~/bin$ ll /home/hadoop/sda4/module/datax/job/import/
hadoop@Master:~/bin$ ll /home/hadoop/sda4/module/datax/job/import
总用量 68
drwxrwxr-x 2 hadoop hadoop 4096 5月 8 00:22 .
drwxrwxrwx 3 hadoop users 4096 5月 8 08:33 ..
-rw-rw-r-- 1 hadoop hadoop 953 5月 8 00:22 gmall.activity_info.json
-rw-rw-r-- 1 hadoop hadoop 1045 5月 8 00:22 gmall.activity_rule.json
-rw-rw-r-- 1 hadoop hadoop 647 5月 8 00:22 gmall.base_category1.json
-rw-rw-r-- 1 hadoop hadoop 707 5月 8 00:22 gmall.base_category2.json
-rw-rw-r-- 1 hadoop hadoop 707 5月 8 00:22 gmall.base_category3.json
-rw-rw-r-- 1 hadoop hadoop 831 5月 8 00:22 gmall.base_dic.json
-rw-rw-r-- 1 hadoop hadoop 861 5月 8 00:22 gmall.base_province.json
-rw-rw-r-- 1 hadoop hadoop 655 5月 8 00:22 gmall.base_region.json
-rw-rw-r-- 1 hadoop hadoop 705 5月 8 00:22 gmall.base_trademark.json
-rw-rw-r-- 1 hadoop hadoop 1297 5月 8 00:22 gmall.cart_info.json
-rw-rw-r-- 1 hadoop hadoop 1541 5月 8 00:22 gmall.coupon_info.json
-rw-rw-r-- 1 hadoop hadoop 863 5月 8 00:22 gmall.sku_attr_value.json
-rw-rw-r-- 1 hadoop hadoop 1117 5月 8 00:22 gmall.sku_info.json
-rw-rw-r-- 1 hadoop hadoop 981 5月 8 00:22 gmall.sku_sale_attr_value.json
-rw-rw-r-- 1 hadoop hadoop 807 5月 8 00:22 gmall.spu_info.json

```

2.2.3.5 测试生成的 DataX 配置文件

以 `activity_info` 为例，测试用脚本生成的配置文件是否可用。

1) 创建目标路径

由于 DataX 同步任务要求目标路径提前存在，故需手动创建路径，当前 `activity_info` 表的目标路径应为 `/origin_data/gmall/db/activity_info_full/2020-06-14`。

```
[hadoop@Master bin]$ hadoop fs -mkdir /origin_data/gmall/db/activity_info_full/2020-06-14
```

2) 执行 DataX 同步命令

```
[hadoop@Master bin]$ python2 /home/hadoop/sda4/module/datax/bin/datax.py
-p"-Dtargertdir=/origin_data/gmall/db/activity_info_full/2020-06-14"
/home/hadoop/sda4/module/datax/job/import/gmall.activity_info.json
```

3) 观察同步结果

观察 HDFS 目标路径是否出现数据。

DataX 成功运行

```

2023-05-10 19:22:46.114 [job-0] INFO JobContainer - PerfTrace not enable!
2023-05-10 19:22:46.115 [job-0] INFO StandAloneJobContainerCommunicator - Total 3 records,
erTime 0.000s | All Task WaitReaderTime 0.000s | Percentage 100.00%
2023-05-10 19:22:46.117 [job-0] INFO JobContainer -
任务启动时刻 : 2023-05-10 19:22:32
任务结束时刻 : 2023-05-10 19:22:46
任务总计耗时 : 13s
任务平均流量 : 9B/s
记录写入速度 : 0rec/s
读出记录总数 : 3
读写失败总数 : 0

```

HDFS 目录出现数据

Browse Directory

/origin_data/gmall/db/activity_info_full/2020-06-14								Go!		
Show	25	entries							Search:	
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name		
	-rw-r--r--	hadoop	supergroup	156 B	May 10 19:22	3	128 MB	activity_info_2f9195de_7f1d_46fb_8ac3_c3f867bbc507.gz		

Showing 1 to 1 of 1 entries

Previous 1 Next

2.2.3.6 全量表数据同步脚本

为方便使用以及后续的任务调度，此处编写一个全量表数据同步脚本。

1) 在~/bin 目录创建 mysql_to_hdfs_full.sh

```
[hadoop@Master bin]$ vim ~/bin/mysql_to_hdfs_full.sh
```

脚本内容如下

```
#!/bin/bash

DATAAX_HOME=/home/hadoop/sda4/module/dataax

# 如果传入日期则 do_date 等于传入的日期，否则等于前一天日期
if [ -n "$2" ] ;then
    do_date=$2
else
    do_date=`date -d "-1 day" +%F`
fi

#处理目标路径，此处的处理逻辑是，如果目标路径不存在，则创建；若存在，则清空，目的是保证同步任务可重复执行
handle_targetdir() {
    hadoop fs -test -e $1
    if [[ $? -eq 1 ]]; then
        echo "路径$1 不存在，正在创建....."
        hadoop fs -mkdir -p $1
    else
        echo "路径$1 已经存在"
        fs_count=$(hadoop fs -count $1)
        content_size=$(echo $fs_count | awk '{print $3}')
        if [[ $content_size -eq 0 ]]; then
            echo "路径$1 为空"
        else
            echo "路径$1 不为空，正在清空....."
            hadoop fs -rm -r -f $1/*
        fi
    fi
}

#数据同步
import_data() {
    dataax_config=$1
    target_dir=$2

    handle_targetdir $target_dir
    python2 $DATAAX_HOME/bin/dataax.py -p"-Dtargertdir=$target_dir" $dataax_config
}

case $1 in
"activity_info")
    import_data /home/hadoop/sda4/module/dataax/job/import/gmall.activity_info.json
    /origin_data/gmall/db/activity_info_full/$do_date
    ;;
"activity_rule")
    import_data /home/hadoop/sda4/module/dataax/job/import/gmall.activity_rule.json
    /origin_data/gmall/db/activity_rule_full/$do_date
    ;;
"base_category1")
    import_data /home/hadoop/sda4/module/dataax/job/import/gmall.base_category1.json
    /origin_data/gmall/db/base_category1_full/$do_date
    ;;
```

```

;;
"base_category2")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_category2.json
/origin_data/gmall/db/base_category2_full/$do_date
;;
"base_category3")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_category3.json
/origin_data/gmall/db/base_category3_full/$do_date
;;
"base_dic")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_dic.json
/origin_data/gmall/db/base_dic_full/$do_date
;;
"base_province")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_province.json
/origin_data/gmall/db/base_province_full/$do_date
;;
"base_region")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_region.json
/origin_data/gmall/db/base_region_full/$do_date
;;
"base_trademark")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_trademark.json
/origin_data/gmall/db/base_trademark_full/$do_date
;;
"cart_info")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.cart_info.json
/origin_data/gmall/db/cart_info_full/$do_date
;;
"coupon_info")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.coupon_info.json
/origin_data/gmall/db/coupon_info_full/$do_date
;;
"sku_attr_value")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.sku_attr_value.json
/origin_data/gmall/db/sku_attr_value_full/$do_date
;;
"sku_info")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.sku_info.json
/origin_data/gmall/db/sku_info_full/$do_date
;;
"sku_sale_attr_value")
    import_data
/home/hadoop/sda4/module/datax/job/import/gmall.sku_sale_attr_value.json
/origin_data/gmall/db/sku_sale_attr_value_full/$do_date
;;
"spu_info")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.spu_info.json
/origin_data/gmall/db/spu_info_full/$do_date
;;
"all")
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.activity_info.json
/origin_data/gmall/db/activity_info_full/$do_date
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.activity_rule.json
/origin_data/gmall/db/activity_rule_full/$do_date
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_category1.json
/origin_data/gmall/db/base_category1_full/$do_date
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_category2.json
/origin_data/gmall/db/base_category2_full/$do_date
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_category3.json
/origin_data/gmall/db/base_category3_full/$do_date
    import_data /home/hadoop/sda4/module/datax/job/import/gmall.base_dic.json

```

```

/origin_data/gmall/db/base_dic_full/$do_date
    import_data /home/hadoop/sda4/module/dax/job/import/gmall.base_province.json
/origin_data/gmall/db/base_province_full/$do_date
    import_data /home/hadoop/sda4/module/dax/job/import/gmall.base_region.json
/origin_data/gmall/db/base_region_full/$do_date
    import_data /home/hadoop/sda4/module/dax/job/import/gmall.base_trademark.json
/origin_data/gmall/db/base_trademark_full/$do_date
    import_data /home/hadoop/sda4/module/dax/job/import/gmall.cart_info.json
/origin_data/gmall/db/cart_info_full/$do_date
    import_data /home/hadoop/sda4/module/dax/job/import/gmall.coupon_info.json
/origin_data/gmall/db/coupon_info_full/$do_date
    import_data /home/hadoop/sda4/module/dax/job/import/gmall.sku_attr_value.json
/origin_data/gmall/db/sku_attr_value_full/$do_date
    import_data /home/hadoop/sda4/module/dax/job/import/gmall.sku_info.json
/origin_data/gmall/db/sku_info_full/$do_date
    import_data
/home/hadoop/sda4/module/dax/job/import/gmall.sku_sale_attr_value.json
/origin_data/gmall/db/sku_sale_attr_value_full/$do_date
    import_data /home/hadoop/sda4/module/dax/job/import/gmall.spu_info.json
/origin_data/gmall/db/spu_info_full/$do_date
;;
esac

```

2) 为 mysql_to_hdfs_full.sh 增加执行权限

```
[hadoop@Master bin]$ chmod 777 ~/bin/mysql_to_hdfs_full.sh
```

3) 测试同步脚本

```
[hadoop@Master bin]$ mysql_to_hdfs_full.sh all 2020-06-14
```

4) 检查同步结果

查看 HDFS 目表路径是否出现全量表数据，全量表共 15 张。

运行脚本之后，开始在数据库里创建 15 张全量表。

```

路径/origin_data/gmall/db/base_dic_full/2020-06-1不存在, 正在创建......

DataX (DATAx-OPENSOURCE-3.0), From Alibaba !
Copyright (C) 2010-2017, Alibaba Group. All Rights Reserved.

2023-05-10 19:32:40.517 [main] INFO  VMInfo - VMInfo# operatingSystem class => sun.management.OperatingSystemImpl
2023-05-10 19:32:40.524 [main] INFO  Engine - the machine info =>

```

生成全量表

/origin_data/gmall/db

Show 25 entries Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:31	0	0 B	activity_info_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:31	0	0 B	activity_rule_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:32	0	0 B	base_category1_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:32	0	0 B	base_category2_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:32	0	0 B	base_category3_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:32	0	0 B	base_dic_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:33	0	0 B	base_province_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:33	0	0 B	base_region_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:33	0	0 B	base_trademark_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:33	0	0 B	cart_info_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	cart_info_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	comment_info_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:34	0	0 B	coupon_info_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	coupon_use_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	favor_info_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_detail_activity_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_detail_coupon_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_detail_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_info_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_refund_info_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_status_log_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	payment_info_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	refund_payment_inc
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:34	0	0 B	sku_attr_value_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:34	0	0 B	sku_info_full

Showing 1 to 25 of 28 entries

Previous 1 2 Next

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:34	0	0 B	sku_sale_attr_value_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 19:35	0	0 B	spu_info_full
□	drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	user_info_inc

Showing 26 to 28 of 28 entries

Previous 1 2 Next

Browse Directory

/origin_data/gmall/db/activity_info_full/2020-06-14

Show 25 entries Search:

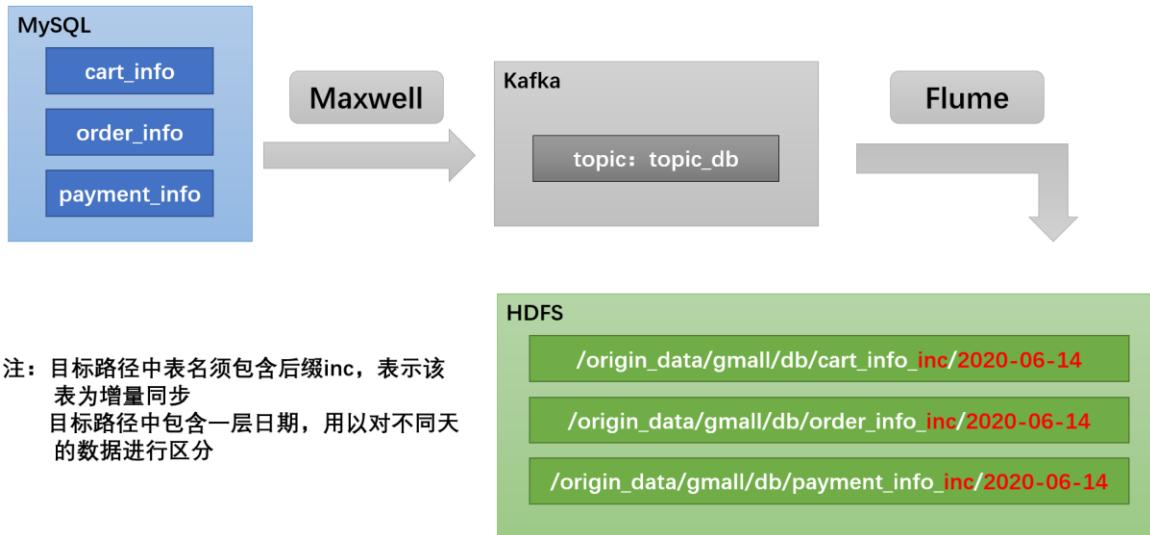
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	-rw-r--r--	hadoop	supergroup	156 B	May 10 19:22	3	128 MB	activity_info_2f9195de_7f1d_46fb_8ac3_c3f867bbc507.gz

Showing 1 to 1 of 1 entries

Previous 1 Next

2.2.4 增量表数据同步

2.2.4.1 数据通道

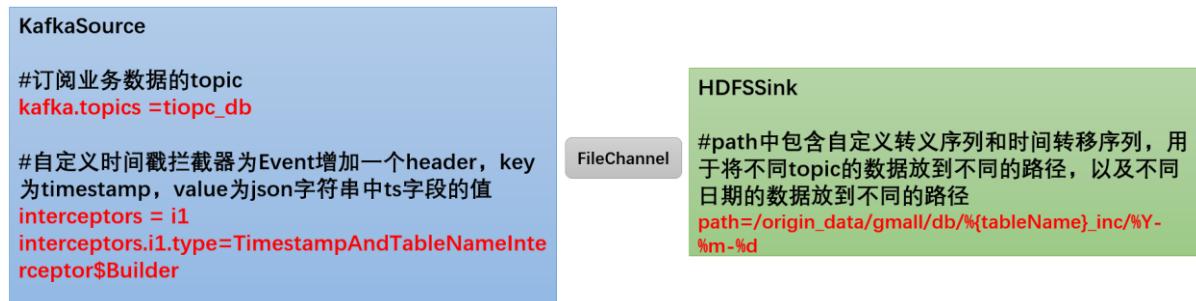


2.2.4.2 Flume 配置

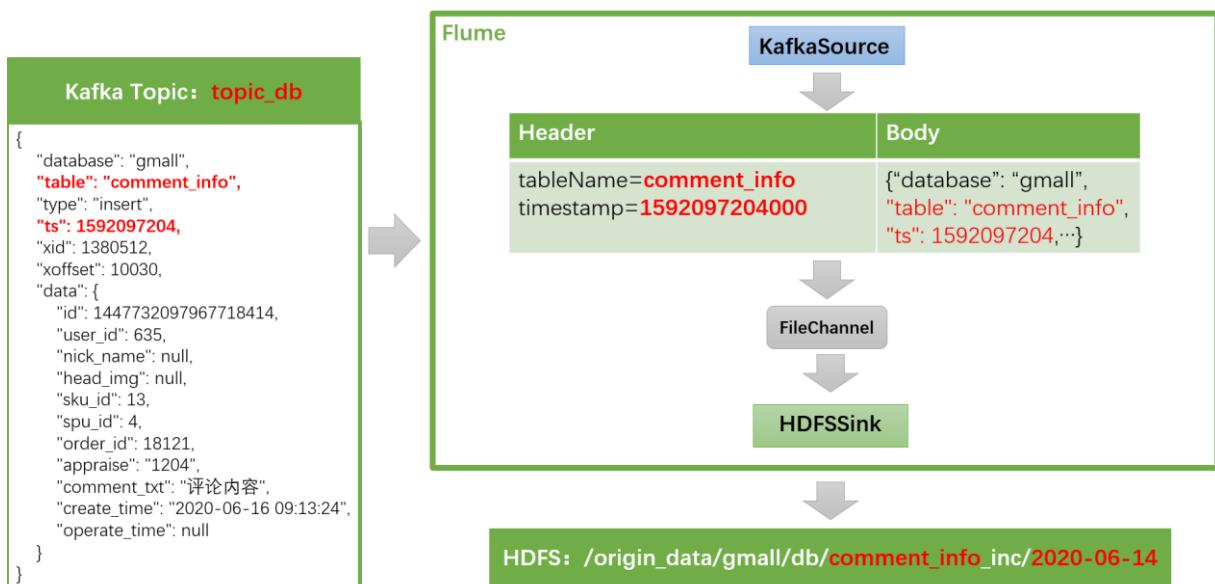
1) Flume 配置概述

Flume 需要将 Kafka 中 `topic_db` 主题的数据传输到 HDFS，故其需选用 KafkaSource 以及 HDFSSink，Channel 选用 FileChannel。

需要注意的是，HDFSSink 需要将不同 mysql 业务表的数据写到不同的路径，并且路径中应当包含一层日期，用于区分每天的数据。关键配置如下：



具体数据示例如下：



2) Flume 配置实操

(1) 创建 Flume 配置文件

在 Slave2 节点的 Flume 的 job 目录下创建 kafka_to_hdfs_db.conf

```
[hadoop@Slave2 flume]$ mkdir job  
[hadoop@Slave2 flume]$ vim job/kafka_to_hdfs_db.conf
```

(2) 配置文件内容如下

```
a1.sources = r1
a1.channels = c1
a1.sinks = k1

a1.sources.r1.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.r1.batchSize = 5000
a1.sources.r1.batchDurationMillis = 2000
a1.sources.r1.kafka.bootstrap.servers = Master:9092,Slave1:9092
a1.sources.r1.kafka.topics = topic_db
a1.sources.r1.kafka.consumer.group.id = flume
a1.sources.r1.setTopicHeader = true
a1.sources.r1.topicHeader = topic
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type = com.mymall.flume.interceptor.TimestampAndTableNameInterceptor$Builder

a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /opt/module/flume/checkpoint/behavior2
a1.channels.c1.dataDirs = /opt/module/flume/data/behavior2/
a1.channels.c1.maxFileSize = 2146435071
a1.channels.c1.capacity = 1000000
a1.channels.c1.keep-alive = 6

## sink1
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = /origin_data/gmall/db/{tableName}_inc/%Y-%m-%d
a1.sinks.k1.hdfs.filePrefix = db
a1.sinks.k1.hdfs.round = false

a1.sinks.k1.hdfs.rollInterval = 10
a1.sinks.k1.hdfs.rollSize = 134217728
a1.sinks.k1.hdfs.rollCount = 0

a1.sinks.k1.hdfs.fileType = CompressedStream
a1.sinks.k1.hdfs.codeC = gzip

## 拼装
a1.sources.r1.channels = c1
a1.sinks.k1.channel= c1
```

(3) 编写 Flume 拦截器

具体代码参考下面的文件



Flume拦截器.doc

X

所有 **Flume** 通道的拦截器写好后都打包在一个 **jar** 包里 **flume-interceptor-1.0-SNAPSHOT-jar-with-dependencies.jar**。分发到 **Master**、**Slave1**、**Slave2** 三台机器上。

3) 通道测试

(1) 启动 Zookeeper、Kafka 集群

(2) 启动 Slave2 的 Flume

```
[hadoop@Slave2 flume]$ bin/flume-ng agent -n a1 -c conf/ -f job/kafka_to_hdfs_db.conf -Dflume.root.logger=info,console
```

(3) 生成模拟数据

```
[hadoop@Master bin]$ cd /opt/module/db_log/
[hadoop@Master db_log]$ java -jar gmall2020-mock-db-2021-11-14.jar
```

(4) 观察 HDFS 上的目标路径是否有数据出现

若 HDFS 上的目标路径已有增量表的数据出现了，就证明数据通道已经打通。

(5) 数据目标路径的日期说明

仔细观察，会发现目标路径中的日期，并非模拟数据的业务日期，而是当前日期。这是由于 Maxwell 输出的 JSON 字符串中的 `ts` 字段的值，是数据的变动日期。而真实场景下，数据的业务日期与变动日期应当是一致的。

```
2023-05-10 16:46:51,223 (hdfs-k1-call-runner-0) [INFO - org.apache.hadoop.io.compress.CodecPool.getCompressor(CodecPool.java:153)] Got brand-new compressor [.gz]
2023-05-10 16:46:51,340 (Thread-9) [INFO - org.apache.hadoop.hdfs.protocol.datatransfer.sasl.SaslDataTransferClient.checkTrustAndSend(SaslDataTransferClient.java:239)] SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-05-10 16:46:51,604 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.hdfs.HDFSCo...Stream.configure(HDFSCo...Stream.java:64)] Serializer = TEXT, UseRawLocalFileSystem = false
2023-05-10 16:46:51,614 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.hdfs.BucketWriter.java:246] Creating /origin_data/gma...l/db/favor_info_inc/2023-05-08/db.1683708411605.gz.tmp
2023-05-10 16:46:51,646 (hdfs-k1-call-runner-5) [INFO - org.apache.hadoop.io.compress.CodecPool.getCompressor(CodecPool.java:153)] Got brand-new compressor [.gz]
2023-05-10 16:46:51,687 (Thread-12) [INFO - org.apache.hadoop.hdfs.protocol.datatransfer.sasl.SaslDataTransferClient.checkTrustAndSend(SaslDataTransferClient.java:239)] SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-05-10 16:46:51,745 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.hdfs.HDFSCo...Stream.configure(HDFSCo...Stream.java:64)] Serializer = TEXT, UseRawLocalFileSystem = false
2023-05-10 16:46:51,757 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.hdfs.BucketWriter.java:246] Creating /origin_data/gma...l/db/cart_info_inc/2023-05-08/db.1683708411746.gz.tmp
2023-05-10 16:46:51,775 (hdfs-k1-call-runner-9) [INFO - org.apache.hadoop.io.compress.CodecPool.getCompressor(CodecPool.java:153)] Got brand-new compressor [.gz]
2023-05-10 16:46:51,822 (Thread-14) [INFO - org.apache.hadoop.hdfs.protocol.datatransfer.sasl.SaslDataTransferClient.checkTrustAndSend(SaslDataTransferClient.java:239)] SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-05-10 16:46:51,895 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.hdfs.HDFSCo...Stream.configure(HDFSCo...Stream.java:64)] Serializer = TEXT, UseRawLocalFileSystem = false
2023-05-10 16:46:51,908 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.hdfs.BucketWriter.java:246] Creating /origin_data/gma...l/db/coupon_use_inc/2023-05-08/db.1683708411896.gz.tmp
2023-05-10 16:46:51,928 (hdfs-k1-call-runner-9) [INFO - org.apache.hadoop.io.compress.CodecPool.getCompressor(CodecPool.java:153)] Got brand-new compressor [.gz]
2023-05-10 16:46:51,971 (Thread-16) [INFO - org.apache.hadoop.hdfs.protocol.datatransfer.sasl.SaslDataTransferClient.checkTrustAndSend(SaslDataTransferClient.java:239)] SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2023-05-10 16:46:52,133 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.hdfs.HDFSCo...Stream.configure(HDFSCo...Stream.java:64)] Serializer = TEXT, UseRawLocalFileSystem = false
```

Browse Directory

/origin_data/gmall/db

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	cart_info_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	comment_info_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	coupon_use_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	favor_info_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_detail_activity_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_detail_coupon_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_detail_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_info_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_refund_info_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	order_status_log_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	payment_info_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	refund_payment_inc
drwxr-xr-x	hadoop	supergroup	0 B	May 10 16:46	0	0 B	user_info_inc

Showing 1 to 13 of 13 entries

Previous 1 Next

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities ▾

Browse Directory

/origin_data/gmall/db/order_info_inc/2023-05-08

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	5.77 KB	May 10 16:47	2	128 MB	db.1683708412134.gz

Showing 1 to 1 of 1 entries

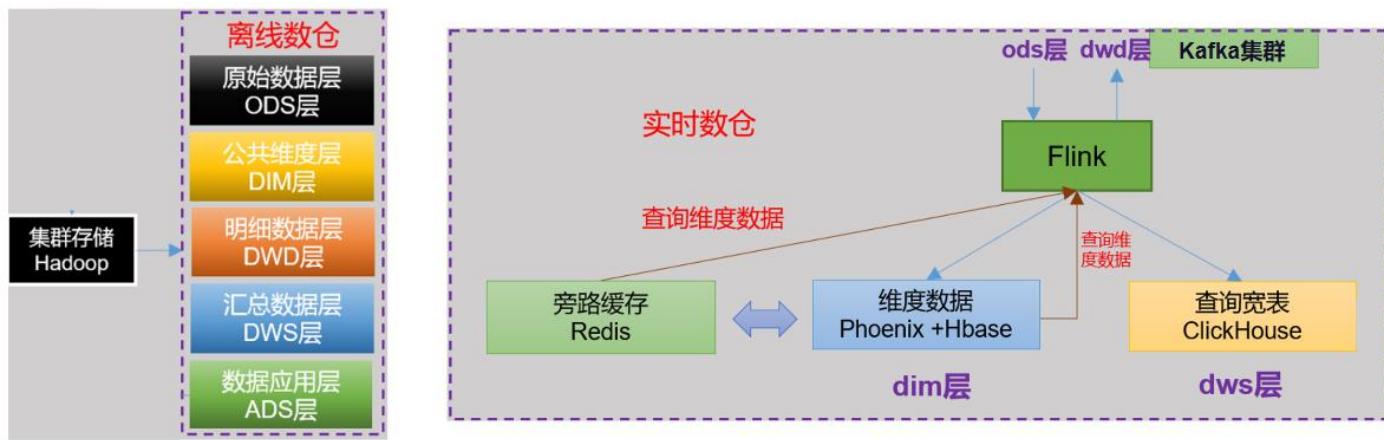
Previous 1 Next

第四章 数据仓库环境准备

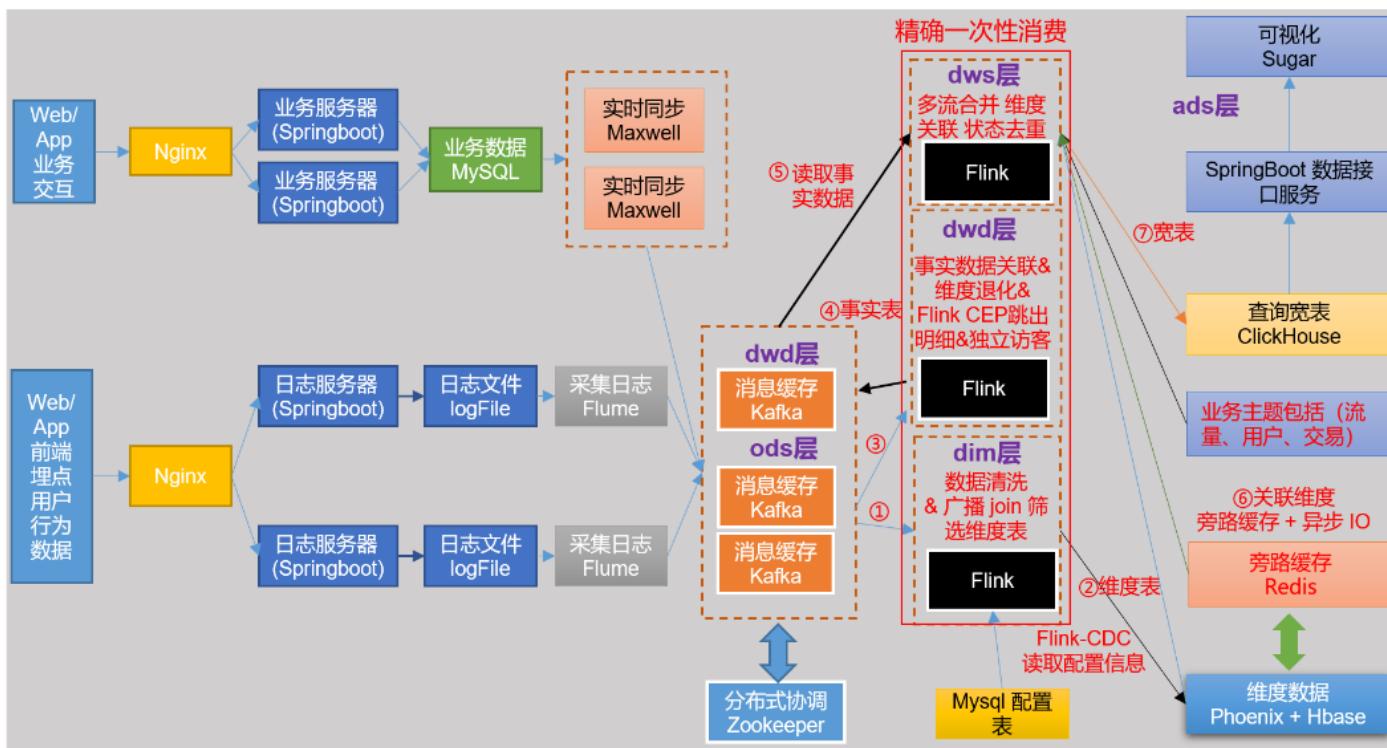
离线数仓: 使用 Hive on Spark 完成离线数仓的分层规划与计算。

实时数仓: 使用 Flink 完成实时数仓 ods 层、dwd 层计算；视情况考虑更引入 Redis、更深层次计算等。

下图分别为实时、离线数仓的规划



但后来我们决定弃做实时数仓，主要是因为需要同时开启主机 IDEA 应用运行 jar 包，同时也要开启虚拟机 Hbase、flume、Kafka、Phoenix、Hbase、Redis、Flink、Hadoop、Zookeeper。计算机资源不够，遂放弃。下图是对一个完善的实时 Flink 数仓具体的设计路线，以后能力足够我们会进行尝试。



但我们也对 Redis、Flink、Hbase 做了简单的学习。

- 吴其乐同学使用 Flink 完成了批处理数据分析与可视化项目
- 王锦龙同学完成了 Redis 的简单实操与高可用分布式集群搭建
- 于丁一同学完成了 Hbase 的分布式部署与简单使用

前两位同学的项目请在我们的另一份文档研究报告中查看，Hbase 的简单应用则在本文档 4.0 部分展示。

4.0 Hbase 简单应用

4.0.1 Hbase 解压

1) 解压 Hbase 到指定目录

```
[hadoop@Master software]$ tar -zxvf hbase-2.4.11-bin.tar.gz -C
/home/hadoop/sda4/module/
[hadoop@Master software]$ mv /home/hadoop/sda4/module/hbase-2.4.11
/home/hadoop/sda4/module/hbase
```

2) 配置环境变量

```
[hadoop@Master ~]$ sudo vim ~/.bashrc
```

添加

```
#HBASE_HOME
export HBASE_HOME= /home/hadoop/sda4/module/hbase
export PATH=$PATH:$HBASE_HOME/bin
```

3) 使用 source 让配置的环境变量生效

```
[hadoop@Master module]$ source ~/.bashrc
```

4.0.2 Hbase 配置文件

1) **hbase-env.sh** 修改内容，可以添加到最后：

```
export HBASE_MANAGES_ZK=false
```

2) **hbase-site.xml** 修改内容：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>

  <property>
    <name>hbase.zookeeper.quorum</name>
    <value>Master,Slave1,Slave2</value>
    <description>The directory shared by RegionServers.
    </description>
  </property>

  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://Master:8020/hbase</value>
    <description>The directory shared by RegionServers.
    </description>
  </property>

  <property>
    <name>hbase.tmp.dir</name>
    <value>./tmp</value>
  </property>

  <property>
    <name>hbase.unsafe.stream.capability.enforce</name>
    <value>false</value>
  </property>

  <property>
    <name>hbase.wal.provider</name>
    <value>filesystem</value>
  </property>
```

```

        </property>

        <property>
        <name>hbase.master.maxclockskew</name>
        <value>300000</value>
        <description>Time difference of regionserver from master</description>
    </property>

</configuration>

```

3) regionservers

```

Master
Slave1
Slave2

```

4) 解决 HBase 和 Hadoop 的 log4j 兼容性问题，修改 HBase 的 jar 包，使用 Hadoop 的 jar 包

```

[hadoop@Master hbase]$ mv /home/hadoop/sda4/module/hbase/lib/client-facing-
thirdparty/slf4j-reload4j-1.7.33.jar
/home/hadoop/sda4/module/hbase/lib/client-facing-thirdparty/slf4j-reload4j-
1.7.33.jar.bak

```

在 HBase 中 HMaster 负责监控 HRegionServer 的生命周期，均衡 RegionServer 的负载，如果 HMaster 挂掉了，那么整个 HBase 集群将陷入不健康的状态，并且此时的工作状态并不会维持太久。所以 HBase 支持对 HMaster 的高可用配置。

1) 在 conf 目录下创建 backup-masters 文件

```
[hadoop@Master hbase]$ touch conf/backup-masters
```

2) 在 backup-masters 文件中配置高可用 HMaster 节点

```
[hadoop@Master hbase]$ echo Slave1 > conf/backup-masters
```

3) 将整个 conf 目录 scp 到其他节点

```
[hadoop@Master hbase]$ xsync conf
```

4.0.3 Hbase 运行

HBase 远程发送到其他集群

```
[hadoop@Master module]$ xsync hbase/
```

HBase 服务的启动

1) 单点启动

```
[hadoop@Master hbase]$ bin/hbase-daemon.sh start master
[hadoop@Master hbase]$ bin/hbase-daemon.sh start regionserver
```

2) 群启

```
[hadoop@Master hbase]$ bin/start-hbase.sh
```

3) 对应的停止服务

```
[hadoop@Master hbase]$ bin/stop-hbase.sh
```

查看 HBase 页面 启动成功后，可以通过“host:port”的方式来访问 HBase 管理页面，例如：

<http://Master:16010>

可以看到 Hmaster 和三台 regionserver 都在正常工作，同时 Slave1 上的备用 Hmaster 也准备完毕

http://master:16010/master-status

APACHE HBASE

Home Table Details Procedures & Locks HBCK Report Named Queue Logs Process Metrics Local Logs Log Level Debug Dump Metrics Dump Profiler HBase Configuration

Master Master

Region Servers

Base Stats	Memory	Requests	Storefiles	Compactions	Replications	Last contact	Version	Requests Per Second	Num. Regions
ServerName	Start time					Last contact	Version	Requests Per Second	Num. Regions
master,16020,1687249896209	Tue Jun 20 16:31:36 CST 2023					1 s	2.4.11	0	1
slave1,16020,1687249897735	Tue Jun 20 16:31:37 CST 2023					2 s	2.4.11	57	2
slave2,16020,1687249897329	Tue Jun 20 16:31:37 CST 2023					1 s	2.4.11	0	3
Total:3								57	6

Backup Masters

ServerName	Port	Start Time
Slave1	16000	Tue Jun 20 16:31:39 CST 2023
Total:1		

以下是一些与 Phoenix 连接时创建的表格

APACHE HBASE

Home Table Details Procedures & Locks HBCK Report Named Queue Logs Process Metrics Local Logs Log Level Debug Dump Metrics Dump Profiler HBase Configuration

User Tables

4 table(s) in set.

Table	Description
SYSTEM.CATALOG	'SYSTEM.CATALOG', {TABLE_ATTRIBUTES => {PRIORITY => '2000', coprocessor\$1 => 'org.apache.phoenix.coprocessor.ScanRegionObserver[805306366]', coprocessor\$2 => 'org.apache.phoenix.coprocessor.Ungr oupedAggregateRegionObserver[805306366]', coprocessor\$3 => 'org.apache.phoenix.coprocessor.GroupedAggregateRegionObserver[805306366]', coprocessor\$4 => 'org.apache.phoenix.coprocessor.ServerCachingEnd pointImpl[805306366]', coprocessor\$5 => 'org.apache.hadoop.hbase.coprocessor.MultiRowMutationEndpoint[805306366]', coprocessor\$6 => 'org.apache.phoenix.coprocessor.MetaDataEndpointImpl[805306366]', coprocessor\$7 => 'org.apache.phoenix.coprocessor.MetaDataRegionObserver[805306367]', coprocessor\$8 => 'org.apache.phoenix.coprocessor.SystemCatalogRegionObserver[805306368]', METADATA => {'SPLIT_POLICY' => 'org.apache.phoenix.schema.MetaDataSplitPolicy'}}}, {NAME => '0', BLOOMFILTER => 'NONE', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'false', DATA_BLOCK_ENCODING => 'FAST_D IFF', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
SYSTEM.FUNCTION	'SYSTEM.FUNCTION', {TABLE_ATTRIBUTES => {PRIORITY => '2000', coprocessor\$1 => 'org.apache.phoenix.coprocessor.ScanRegionObserver[805306366]', coprocessor\$2 => 'org.apache.phoenix.coprocessor.Ungr oupedAggregateRegionObserver[805306366]', coprocessor\$3 => 'org.apache.phoenix.coprocessor.GroupedAggregateRegionObserver[805306366]', coprocessor\$4 => 'org.apache.phoenix.coprocessor.ServerCachingEnd pointImpl[805306366]', coprocessor\$5 => 'org.apache.phoenix.hbase.index.IndexRegionObserver[805306366]index.builder=org.apache.phoenix.index.PhoenixIndexBuilder,org.apache.hadoop.hbase.index.codec.class=org.apache.phoenix.index.PhoenixIndexCodec', coprocessor\$6 => 'org.apache.phoenix.coprocessor.MetaDataEndpointImpl[805306366]', METADATA => {'SPLIT_POLICY' => 'org.apache.phoenix.schema.SystemFunc tionsSplitPolicy'}}}, {NAME => '0', BLOOMFILTER => 'NONE', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'false', DATA_BLOCK_ENCODING => 'FAST_DIFF', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
SYSTEM.SEQUENCE	'SYSTEM.SEQUENCE', {TABLE_ATTRIBUTES => {PRIORITY => '2000', coprocessor\$1 => 'org.apache.phoenix.coprocessor.ScanRegionObserver[805306366]', coprocessor\$2 => 'org.apache.phoenix.coprocessor.Ungr oupedAggregateRegionObserver[805306366]', coprocessor\$3 => 'org.apache.phoenix.coprocessor.GroupedAggregateRegionObserver[805306366]', coprocessor\$4 => 'org.apache.phoenix.coprocessor.ServerCachingEnd poi ntImpl[805306366]', coprocessor\$5 => 'org.apache.phoenix.hbase.index.IndexRegionObserver[805306366]index.builder=org.apache.phoenix.index.PhoenixIndexBuilder,org.apache.hadoop.hbase.index.codec.class=org.apache.phoenix.index.PhoenixIndexCodec', coprocessor\$6 => 'org.apache.phoenix.coprocessor.SequenceRegionObserver[805306366]'}, {NAME => '0', BLOOMFILTER => 'NONE', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'false', DATA_BLOCK_ENCODING => 'FAST_DIFF', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
SYSTEM.STATUS	'SYSTEM.STATUS', {TABLE_ATTRIBUTES => {PRIORITY => '2000', coprocessor\$1 => 'org.apache.phoenix.coprocessor.ScanRegionObserver[805306366]', coprocessor\$2 => 'org.apache.phoenix.coprocessor.Ungr oupedAggregateRegionObserver[805306366]', coprocessor\$3 => 'org.apache.phoenix.coprocessor.GroupedAggregateRegionObserver[805306366]', coprocessor\$4 => 'org.apache.phoenix.coprocessor.ServerCachingEndpoi ntImpl[805306366]', coprocessor\$5 => 'org.apache.hadoop.hbase.coprocessor.MultiRowMutationEndpoint[805306366]', METADATA => {'SPLIT_POLICY' => 'org.apache.phoenix.schema.SystemStatsSplitPolicy'}}}, {NAME => '0', BLOOMFILTER => 'NONE', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'FAST_DIFF', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}

4.1 数据仓库运行环境

4.1.1 Hive 环境搭建

Hive 引擎简介

Hive 引擎包括：默认 MR、Tez、Spark。

Hive on Spark: Hive 既作为存储元数据又负责 SQL 的解析优化，语法是 HQL 语法，执行引擎变成了 Spark，Spark 负责采用 RDD 执行。本项目采用 **Hive on Spark**。

Spark on Hive : Hive 只作为存储元数据, Spark 负责 SQL 解析优化, 语法是 Spark SQL 语法, Spark 负责采用 RDD 执行。

Hive 安装部署

1) 把 `apache-hive-3.1.2-bin.tar.gz` 上传到虚拟机 Master 的 `/home/hadoop/sda4/software` 目录下

2) 解压 `apache-hive-3.1.2-bin.tar.gz` 到 `/home/hadoop/sda4/module` 目录下面

```
[hadoop@Master software]$ tar -zxvf /home/hadoop/sda4/software/apache-hive-3.1.2-bin.tar.gz -C /home/hadoop/sda4/module/
```

3) 修改 `apache-hive-3.1.2-bin.tar.gz` 的名称为 `hive`

```
[hadoop@Master software]$ mv /home/hadoop/sda4/module/apache-hive-3.1.2-bin/ /home/hadoop/sda4/module/hive
```

4) 添加环境变量到 `.bashrc` 文件

```
#HIVE_HOME
export HIVE_HOME=/opt/module/hive
export PATH=$PATH:$HIVE_HOME/bin
```

重启 Xshell 对话框或者 source 一下 `.bashrc` 文件, 使环境变量生效

```
[hadoop@Master software]$ source /etc/profile.d/my_env.sh
```

5) 解决日志 Jar 包冲突, 进入 `/home/hadoop/sda4/module/hive/lib` 目录

```
[hadoop@Master lib]$ mv log4j-slf4j-impl-2.10.0.jar log4j-slf4j-impl-2.10.0.jar.bak
```

Hive 元数据配置到 MySQL

拷贝驱动

将 MySQL 的 JDBC 驱动拷贝到 Hive 的 lib 目录下

```
[hadoop@Master lib]$ cp /home/hadoop/sda4/software/ mysql-connector-java-8.0.33.jar /home/hadoop/sda4/module/hive/lib/
```

配置 Metastore 到 MySQL

在 `$HIVE_HOME/conf` 目录下新建 `hive-site.xml` 文件

```
[hadoop@Master conf]$ vim hive-site.xml
```

添加如下内容

```
<?xml version="1.0"?>
<?xmlstylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://Master:3306/metastore?useSSL=false&useUnicode=true&characterEncoding=UTF-8</value>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.jdbc.Driver</value>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>root</value>
    </property>
```

```

<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>yudingyi</value>
</property>

<property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/user/hive/warehouse</value>
</property>

<property>
    <name>hive.metastore.schema.verification</name>
    <value>false</value>
</property>

<property>
<name>hive.server2.thrift.port</name>
<value>10000</value>
</property>

<property>
    <name>hive.server2.thrift.bind.host</name>
    <value>Master</value>
</property>

<property>
    <name>hive.metastore.event.db.notification.api.auth</name>
    <value>false</value>
</property>

<property>
    <name>hive.cli.print.header</name>
    <value>true</value>
</property>

<property>
    <name>hive.cli.print.current.db</name>
    <value>true</value>
</property>

<property>
    <name>metastore.storage.schema.reader.impl</name>
<value>org.apache.hadoop.hive.metastore.SerDeStorageSchemaReader</value>
</property>

<!--Spark 依赖位置（注意：端口号 8020 必须和 namenode 的端口号一致）-->
<property>
    <name>spark.yarn.jars</name>
    <value>hdfs://Master:8020/spark-jars/*</value>
</property>

<!--Hive 执行引擎-->
<property>
    <name>hive.execution.engine</name>
    <value>spark</value>
</property>

<property>
    <name>hive.spark.client.connect.timeout</name>
    <value>10000</value>

```

```

</property>
<property>
    <name>hive.spark.client.server.connect.timeout</name>
    <value>90000</value>
</property>

</configuration>

```

启动 Hive

初始化元数据库

1) 登陆 MySQL

```
[hadoop@Master conf]$ mysql -uroot -pyudingyi
```

2) 新建 Hive 元数据库

```
mysql> create database metastore;
```

3) 初始化 Hive 元数据库

```
[hadoop@Master conf]$ schematool -initSchema -dbType mysql -verbose
```

4) 修改元数据库字符集

Hive 元数据库的字符集默认为 Latin1，由于其不支持中文字符，故若建表语句中包含中文注释，会出现乱码现象。如需解决乱码问题，须做以下修改。

修改 Hive 元数据库中存储注释的字段的字符集为 utf-8

(1) 字段注释

```
mysql> alter table COLUMNS_V2 modify column COMMENT varchar(256) character set utf8;
```

(2) 表注释

```
mysql> alter table TABLE_PARAMS modify column PARAM_VALUE mediumtext character set utf8;
```

4) 退出 mysql

```
mysql> quit;
```

同样可以看到 HDFS 已建立 Hive 相关辅助文件

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	drwxrwx---	hadoop	supergroup	0 B	Jun 16 14:19	0	0 B	hadoop-yarn
□	drwx-wx-wx	hadoop	supergroup	0 B	Jun 19 15:33	0	0 B	hive
□	drwxrwxrwt	hadoop	hadoop	0 B	Jun 19 09:18	0	0 B	logs

Showing 1 to 3 of 3 entries

Previous 1 Next

下图为 SQLyog 里查看建立好的 Hive 源数据库

The screenshot shows the metastore database structure in SQLyog Ultimate. The left pane displays the database schema with various tables such as AUX_TABLE, BUCKETING_COLS, CDS, and many others under the metastore schema. The right pane provides a detailed view of the 'COMPLETED_TXN_COMPONENTS' table, showing its columns and their types. The table has 241 rows.

COLUMN_NAME	TYPE_NAME
CD_ID	int
COMMENT	string
1 (NULL)	string
6 动作信息	array<struct<action_id:string>>
6 公共信息	struct<ar:string,ba:string>
6 曝光信息	array<struct<display_type:string,display_value:string>>
6 错误信息	struct<error_code:bigint,err_message:string>
6 页面信息	struct<during_time:string,page:string>
6 启动信息	struct<entry:string,load_time:string>
6 时间戳	struct<start:bigint,ts:bigint>
7 活动描述	string
7 活动名称	string
7 活动类型	string
7 创建时间	string
7 结束时间	string
7 活动id	string
7 开始时间	string
8 类型	string
8 活动类型	string
8 优惠金额	decimal(16,2)
8 优惠折扣	decimal(16,2)
8 优惠级别	string
8 满减金额	decimal(16,2)
8 满减件数	bigint
8 编号	string
9 编号	string
9 分类名称	string
10 一级分类编号	string
10 编号	string
10 二级分类名称	string
11 二级分类编号	string
11 编号	string

建立好的元数据仓库

4.1.2 Hive on Spark 配置

Hive on Spark 配置

(1) 兼容性说明

注意：官网下载的 Hive3.1.2 和 Spark3.0.0 默认是不兼容的。因为 Hive3.1.2 支持的 Spark 版本是 2.4.5，所以需要我们重新编译 Hive3.1.2 版本。

编译步骤：官网下载 Hive3.1.2 源码，修改 pom 文件中引用的 Spark 版本为 3.0.0，如果编译通过，直接打包获取 jar 包。如果报错，就根据提示，修改相关方法，直到不报错，打包获取 jar 包。

(2) 在 Hive 所在节点部署 Spark

如果之前已经部署了 Spark，则该步骤可以跳过。

① Spark 官网下载 jar 包地址

<http://spark.apache.org/downloads.html>

② 上传并解压解压 spark-3.0.0-bin-hadoop3.2.tgz

```
[hadoop@Master software]$ tar -zxvf spark-3.0.0-bin-hadoop3.2.tgz -C /opt/module/
[hadoop@Master software]$ mv /home/hadoop/sda4/module/spark-3.0.0-bin-hadoop3.2 /home/hadoop/sda4/module/spark
```

(3) 配置 SPARK_HOME 环境变量

添加如下内容。

```
# SPARK_HOME
export SPARK_HOME=/home/hadoop/sda4/module/spark
export PATH=$PATH:$SPARK_HOME/bin
source 使其生效。
```

```
[hadoop@Master software]$ source ~/.bashrc
```

(4) 在 **hive** 中创建 **spark** 配置文件

```
[hadoop@Master software]$ vim /home/hadoop/sda4/module/hive/conf/spark-defaults.conf
```

添加如下内容（在执行任务时，会根据如下参数执行）。

spark.master	yarn
spark.eventLog.enabled	true
spark.eventLog.dir	hdfs://Master:8020/spark-history
spark.executor.memory	1g
spark.driver.memory	1g

在 HDFS 创建如下路径，用于存储历史日志。

```
[hadoop@Master software]$ hadoop fs -mkdir /spark-history
```

(5) 向 HDFS 上传 Spark 纯净版 jar 包

说明 1：由于 Spark3.0.0 非纯净版默认支持的是 hive2.3.7 版本，直接使用会和安装的 Hive3.1.2 出现兼容性问题。所以采用 Spark 纯净版 jar 包，不包含 hadoop 和 hive 相关依赖，避免冲突。

说明 2：Hive 任务最终由 Spark 来执行，Spark 任务资源分配由 Yarn 来调度，该任务有可能被分配到集群的任何一个节点。所以需要将 Spark 的依赖上传到 HDFS 集群路径，这样集群中任何一个节点都能获取到。

① 上传并解压 spark-3.0.0-bin-without-hadoop.tgz

```
[hadoop@Master software]$ tar -zxvf /home/hadoop/sda4/software/spark-3.0.0-bin-without-hadoop.tgz
```

② 上传 Spark 纯净版 jar 包到 HDFS

```
[hadoop@Master software]$ hadoop fs -mkdir /spark-jars
```

```
[hadoop@Master software]$ hadoop fs -put spark-3.0.0-bin-without-hadoop/jars/* /spark-jars
```

(6) 修改 **hive-site.xml** 文件

```
[hadoop@Master ~]$ vim /home/hadoop/sda4/module/hive/conf/hive-site.xml
```

添加如下内容。

```
<!--Spark 依赖位置（注意：端口号 8020 必须和 namenode 的端口号一致）-->
<property>
  <name>spark.yarn.jars</name>
  <value>hdfs://Master:8020/spark-jars/*</value>
</property>
```

```
<!--Hive 执行引擎-->
<property>
    <name>hive.execution.engine</name>
    <value>spark</value>
</property>
```

Hive on Spark 测试

(1) 启动 hive 客户端

```
[hadoop@Master hive]$ bin/hive
```

(2) 创建一张测试表

```
hive (default)> create table student(id int, name string);
```

(3) 通过 insert 测试效果

```
hive (default)> insert into table student values(1,'abc');
```

若结果如下，则说明配置成功。

```
hive (default)> insert into table student values(1,'abc');
Query ID = atguigu_20200719001740_b025ae13-c573-4a68-9b74-50a4d018664b
Total jobs = 1
Launching Job 1 out of 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
-----
      STAGES   ATTEMPT     STATUS TOTAL COMPLETED RUNNING PENDING FAILED
-----
Stage-2 .....      0    FINISHED     1       1      0      0      0
Stage-3 .....      0    FINISHED     1       1      0      0      0
-----
STAGES: 02/02  [=====>>] 100% ELAPSED TIME: 1.01 s
-----
Spark job[1] finished successfully in 1.01 second(s)
Loading data to table default.student
OK
col1  col2
Time taken: 1.514 seconds
hive (default)> 
```

4.1.3 Yarn 环境配置

1) 增加 ApplicationMaster 资源比例

容量调度器对每个资源队列中同时运行的 Application Master 占用的资源进行了限制，该限制通过 `yarn.scheduler.capacity.maximum-am-resource-percent` 参数实现，其默认值是 0.1，表示每个资源队列上 Application Master 最多可使用的资源为该队列总资源的 10%，目的是防止大部分资源都被 Application Master 占用，而导致 Map/Reduce Task 无法执行。

生产环境该参数可使用默认值。但学习环境，集群资源总数很少，如果只分配 10% 的资源给 Application Master，则可能出现，同一时刻只能运行一个 Job 的情况，因为一个 Application Master 使用的资源就可能已经达到 10% 的上限了。故此处可将该值适当调大。

(1) 在 Master 的

`/usr/local/hadoop/etc/hadoop/capacity-scheduler.xml` 文件中修改如下

参数值

```
[hadoop@Master hadoop]$ vim capacity-scheduler.xml
```

```
<property>
  <name>yarn.scheduler.capacity.maximum-am-resource-percent</name>
  <value>0.8</value>
</property>
```

(2) 分发 capacity-scheduler.xml 配置文件

```
[hadoop@Master hadoop]$ xsync capacity-scheduler.xml
```

(3) 关闭正在运行的任务，重新启动 yarn 集群

```
[hadoop@Slave1 hadoop-3.1.3]$ sbin/stop-yarn.sh
```

```
[hadoop@Slave1 hadoop-3.1.3]$ sbin/start-yarn.sh
```

4.2 数据仓库开发环境

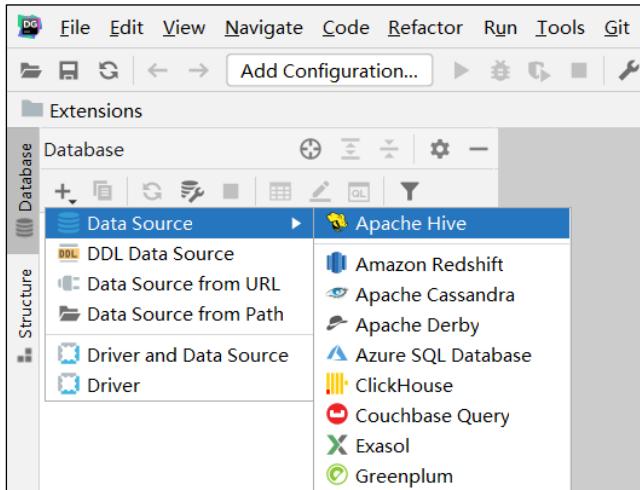
数仓开发工具可选用 DBeaver 或者 DataGrip。两者都需要用到 JDBC 协议连接到 Hive，故需要启动 HiveServer2。

1) 启动 HiveServer2

```
[hadoop@Master hive]$ hiveserver2
```

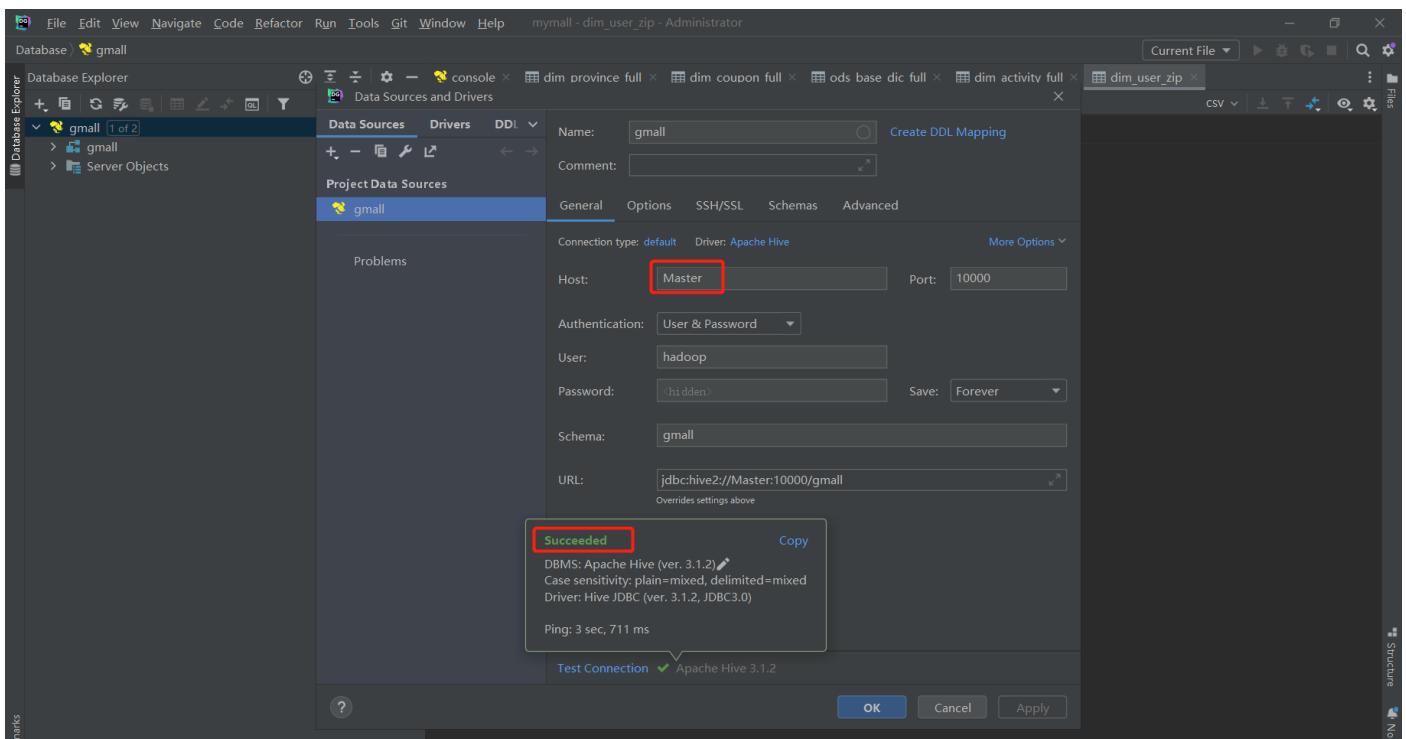
2) 配置 DataGrip 连接

(1) 创建连接



(2) 配置连接属性

所有属性配置，和 Hive 的 beeline 客户端配置一致即可。初次使用，配置过程会提示缺少 JDBC 驱动，按照提示下载即可。

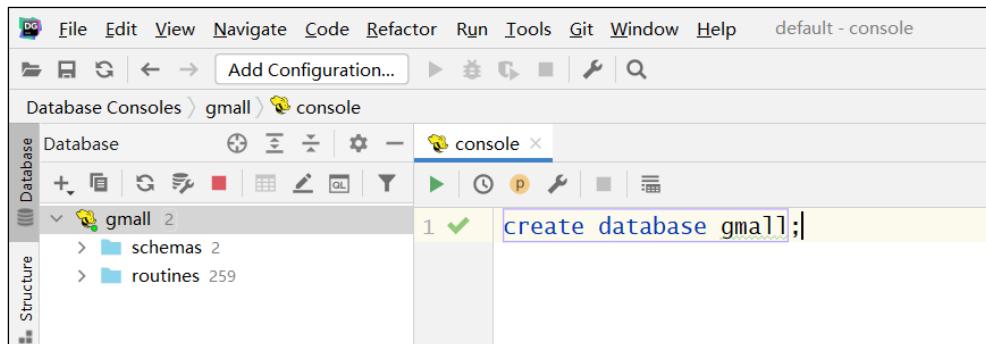


连接成功截图

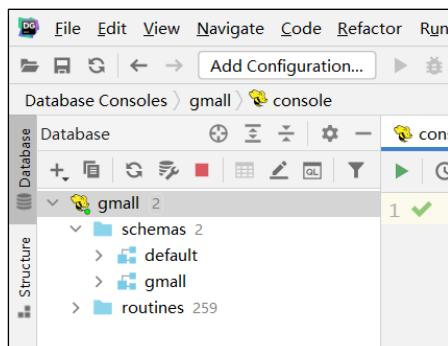
3) 测试使用

创建数据库 `gmall`，并观察是否创建成功。

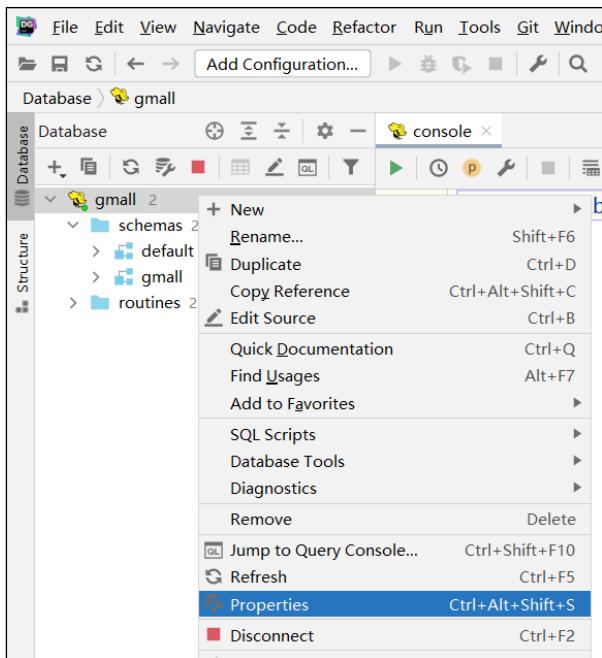
(1) 创建数据库



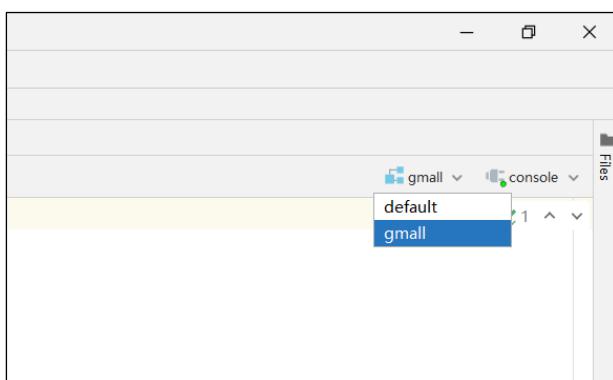
(2) 查看数据库



(3) 修改连接，指明连接数据库



(4) 选择当前数据库为 `gmail`



数仓的设计理论可以见研究文档。

第五章 数仓开发之 ODS 层

ODS 层的设计要点如下：

- (1) ODS 层的表结构设计依托于从业务系统同步过来的数据结构。
- (2) ODS 层要保存全部历史数据，故其压缩格式应选择压缩比较高的，此处选择 `gzip`。
- (3) ODS 层表名的命名规范为：`ods_表名_单分区增量全量标识 (inc/full)`。

5.1 日志表

1) 建表数据

```
DROP TABLE IF EXISTS ods_log_inc;
CREATE EXTERNAL TABLE ods_log_inc
(
    `common`
```

```

STRUCT<ar :STRING,ba :STRING,ch :STRING,is_new :STRING,md :STRING,mid :S
TRING,os :STRING,uid :STRING,vc
                           :STRING> COMMENT '公共信息',
`page`  

STRUCT<during_time :STRING,item :STRING,item_type :STRING,last_page_id :S
TRING,page_id
                           :STRING,source_type :STRING> COMMENT '页面信息',
`actions`  

ARRAY<STRUCT<action_id:STRING,item:STRING,item_type:STRING,ts:BIGINT>>
COMMENT '动作信息',
`displays`  

ARRAY<STRUCT<display_type :STRING,item :STRING,item_type :STRING,`order`:
STRING,pos_id
                           :STRING>> COMMENT '曝光信息',
`start`  

STRUCT<entry :STRING,loading_time :BIGINT,open_ad_id :BIGINT,open_ad_ms
:BIGINT,open_ad_skip_ms
                           :BIGINT> COMMENT '启动信息',
`err`      STRUCT<error_code:BIGINT,msg:STRING> COMMENT '错误信息',
`ts`        BIGINT COMMENT '时间戳',
) COMMENT '活动信息表',
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_log_inc/';

```

2) 数据装载

```
load data inpath '/origin_data/gmall/log/topic_log/2020-06-14' into
table ods_log_inc partition(dt='2020-06-14');
```

3) 每日数据装载脚本

(1) 在 Master 的/home/hadoop/bin 目录下创建 hdfs_to_ods_log.sh

```
[hadoop@Master bin]$ vim hdfs_to_ods_log.sh
```

(2) 编写如下内容

```
#!/bin/bash

# 定义变量方便修改
APP=gmall

# 如果是输入的日期按照取输入日期; 如果没输入日期取当前时间的前一天
if [ -n "$1" ] ;then
  do_date=$1
else
  do_date=`date -d "-1 day" +%F`
fi

echo ===== 日志日期为 $do_date =====
sql="
load data inpath '/origin_data/$APP/log/topic_log/$do_date' into table
${APP}.ods_log_inc partition(dt='$do_date');
"
hive -e "$sql"
```

(3) 增加脚本执行权限

```
[hadoop@Master bin]$ chmod +x hdfs_to_ods_log.sh
(4) 脚本用法
```

```
[hadoop@Master bin]$ hdfs_to_ods_log.sh 2020-06-14
```

5.2 业务表

5.2.1 活动信息表（全量表）

```
DROP TABLE IF EXISTS ods_activity_info_full;
CREATE EXTERNAL TABLE ods_activity_info_full
(
    `id`          STRING COMMENT '活动 id',
    `activity_name` STRING COMMENT '活动名称',
    `activity_type` STRING COMMENT '活动类型',
    `activity_desc` STRING COMMENT '活动描述',
    `start_time`   STRING COMMENT '开始时间',
    `end_time`     STRING COMMENT '结束时间',
    `create_time`  STRING COMMENT '创建时间'
) COMMENT '活动信息表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_activity_info_full/';
```

5.2.2 活动规则表（全量表）

```
DROP TABLE IF EXISTS ods_activity_rule_full;
CREATE EXTERNAL TABLE ods_activity_rule_full
(
    `id`          STRING COMMENT '编号',
    `activity_id`  STRING COMMENT '类型',
    `activity_type` STRING COMMENT '活动类型',
    `condition_amount` DECIMAL(16, 2) COMMENT '满减金额',
    `condition_num`   BIGINT COMMENT '满减件数',
    `benefit_amount`  DECIMAL(16, 2) COMMENT '优惠金额',
    `benefit_discount` DECIMAL(16, 2) COMMENT '优惠折扣',
    `benefit_level`   STRING COMMENT '优惠级别'
) COMMENT '活动规则表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_activity_rule_full/';
```

5.2.3 一级品类表（全量表）

```
DROP TABLE IF EXISTS ods_base_category1_full;
CREATE EXTERNAL TABLE ods_base_category1_full
(
    `id`      STRING COMMENT '编号',
    `name`    STRING COMMENT '分类名称'
) COMMENT '一级品类表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
```

```
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_base_category1_full/';
```

5.2.4 二级品类表（全量表）

```
DROP TABLE IF EXISTS ods_base_category2_full;
CREATE EXTERNAL TABLE ods_base_category2_full
(
    `id`          STRING COMMENT '编号',
    `name`        STRING COMMENT '二级分类名称',
    `category1_id` STRING COMMENT '一级分类编号'
) COMMENT '二级品类表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_base_category2_full/';
```

5.2.5 三级品类表（全量表）

```
DROP TABLE IF EXISTS ods_base_category3_full;
CREATE EXTERNAL TABLE ods_base_category3_full
(
    `id`          STRING COMMENT '编号',
    `name`        STRING COMMENT '三级分类名称',
    `category2_id` STRING COMMENT '二级分类编号'
) COMMENT '三级品类表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_base_category3_full/';
```

5.2.6 编码字典表（全量表）

```
DROP TABLE IF EXISTS ods_base_dic_full;
CREATE EXTERNAL TABLE ods_base_dic_full
(
    `dic_code`      STRING COMMENT '编号',
    `dic_name`      STRING COMMENT '编码名称',
    `parent_code`   STRING COMMENT '父编号',
    `create_time`   STRING COMMENT '创建日期',
    `operate_time`  STRING COMMENT '修改日期'
) COMMENT '编码字典表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_base_dic_full/';
```

5.2.7 省份表（全量表）

```
DROP TABLE IF EXISTS ods_base_province_full;
CREATE EXTERNAL TABLE ods_base_province_full
(
    `id`          STRING COMMENT '编号',
    `name`        STRING COMMENT '省份名称',
    `region_id`   STRING COMMENT '地区 ID',
    `area_code`   STRING COMMENT '地区编码',
```

```

`iso_code`      STRING COMMENT '旧版 ISO-3166-2 编码, 供可视化使用',
`iso_3166_2`   STRING COMMENT '新版 IOS-3166-2 编码, 供可视化使用'
) COMMENT '省份表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_base_province_full/';

```

5.2.8 地区表（全量表）

```

DROP TABLE IF EXISTS ods_base_region_full;
CREATE EXTERNAL TABLE ods_base_region_full
(
  `id`          STRING COMMENT '编号',
  `region_name` STRING COMMENT '地区名称'
) COMMENT '地区表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_base_region_full/';

```

5.2.9 品牌表（全量表）

```

DROP TABLE IF EXISTS ods_baseTrademark_full;
CREATE EXTERNAL TABLE ods_baseTrademark_full
(
  `id`          STRING COMMENT '编号',
  `tm_name`    STRING COMMENT '品牌名称',
  `logo_url`   STRING COMMENT '品牌 logo 的图片路径'
) COMMENT '品牌表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_baseTrademark_full/';

```

5.2.10 购物车表（全量表）

```

DROP TABLE IF EXISTS ods_cart_info_full;
CREATE EXTERNAL TABLE ods_cart_info_full
(
  `id`          STRING COMMENT '编号',
  `user_id`    STRING COMMENT '用户 id',
  `sku_id`     STRING COMMENT 'sku_id',
  `cart_price` DECIMAL(16, 2) COMMENT '放入购物车时价格',
  `sku_num`    BIGINT COMMENT '数量',
  `img_url`   BIGINT COMMENT '商品图片地址',
  `sku_name`  STRING COMMENT 'sku 名称 (冗余)',
  `is_checked` STRING COMMENT '是否被选中',
  `create_time` STRING COMMENT '创建时间',
  `operate_time` STRING COMMENT '修改时间',
  `is_ordered` STRING COMMENT '是否已经下单',
  `order_time` STRING COMMENT '下单时间',
  `source_type` STRING COMMENT '来源类型',
  `source_id`  STRING COMMENT '来源编号'
)

```

```
) COMMENT '购物车全量表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_cart_info_full/';
```

5.2.11 优惠券信息表（全量表）

```
DROP TABLE IF EXISTS ods_coupon_info_full;
CREATE EXTERNAL TABLE ods_coupon_info_full
(
  `id`          STRING COMMENT '购物券编号',
  `coupon_name` STRING COMMENT '购物券名称',
  `coupon_type` STRING COMMENT '购物券类型 1 现金券 2 折扣券 3 满减券
4 满件打折券',
  `condition_amount` DECIMAL(16, 2) COMMENT '满额数',
  `condition_num` BIGINT COMMENT '满件数',
  `activity_id` STRING COMMENT '活动编号',
  `benefit_amount` DECIMAL(16, 2) COMMENT '减金额',
  `benefit_discount` DECIMAL(16, 2) COMMENT '折扣',
  `create_time` STRING COMMENT '创建时间',
  `range_type` STRING COMMENT '范围类型 1、商品 2、品类 3、品牌',
  `limit_num` BIGINT COMMENT '最多领用次数',
  `taken_count` BIGINT COMMENT '已领用次数',
  `start_time` STRING COMMENT '开始领取时间',
  `end_time` STRING COMMENT '结束领取时间',
  `operate_time` STRING COMMENT '修改时间',
  `expire_time` STRING COMMENT '过期时间'
) COMMENT '优惠券信息表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_coupon_info_full/';
```

5.2.12 商品平台属性表（全量表）

```
DROP TABLE IF EXISTS ods_sku_attr_value_full;
CREATE EXTERNAL TABLE ods_sku_attr_value_full
(
  `id`          STRING COMMENT '编号',
  `attr_id`    STRING COMMENT '平台属性 ID',
  `value_id`   STRING COMMENT '平台属性值 ID',
  `sku_id`     STRING COMMENT '商品 ID',
  `attr_name`  STRING COMMENT '平台属性名称',
  `value_name` STRING COMMENT '平台属性值名称'
) COMMENT 'sku 平台属性表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_sku_attr_value_full/';
```

5.2.13 商品表（全量表）

```
DROP TABLE IF EXISTS ods_sku_info_full;
```

```

CREATE EXTERNAL TABLE ods_sku_info_full
(
  `id`          STRING COMMENT 'skuId',
  `spu_id`      STRING COMMENT 'spuid',
  `price`       DECIMAL(16, 2) COMMENT '价格',
  `sku_name`    STRING COMMENT '商品名称',
  `sku_desc`    STRING COMMENT '商品描述',
  `weight`      DECIMAL(16, 2) COMMENT '重量',
  `tm_id`       STRING COMMENT '品牌 id',
  `category3_id` STRING COMMENT '品类 id',
  `sku_default_igm` STRING COMMENT '商品图片地址',
  `is_sale`     STRING COMMENT '是否在售',
  `create_time` STRING COMMENT '创建时间'
) COMMENT 'SKU 商品表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_sku_info_full/';

```

5.2.14 商品销售属性值表（全量表）

```

DROP TABLE IF EXISTS ods_sku_sale_attr_value_full;
CREATE EXTERNAL TABLE ods_sku_sale_attr_value_full
(
  `id`          STRING COMMENT '编号',
  `sku_id`      STRING COMMENT 'sku_id',
  `spu_id`      STRING COMMENT 'spu_id',
  `sale_attr_value_id` STRING COMMENT '销售属性值 id',
  `sale_attr_id` STRING COMMENT '销售属性 id',
  `sale_attr_name` STRING COMMENT '销售属性名称',
  `sale_attr_value_name` STRING COMMENT '销售属性值名称'
) COMMENT 'sku 销售属性名称'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_sku_sale_attr_value_full/';

```

5.2.15 SPU 表（全量表）

```

DROP TABLE IF EXISTS ods_spu_info_full;
CREATE EXTERNAL TABLE ods_spu_info_full
(
  `id`          STRING COMMENT 'spu_id',
  `spu_name`    STRING COMMENT 'spu 名称',
  `description` STRING COMMENT '描述信息',
  `category3_id` STRING COMMENT '品类 id',
  `tm_id`       STRING COMMENT '品牌 id'
) COMMENT 'SPU 商品表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
NULL DEFINED AS ''
LOCATION '/warehouse/gmall/ods/ods_spu_info_full/';

```

5.2.16 购物车表（增量表）

```

DROP TABLE IF EXISTS ods_cart_info_inc;
CREATE EXTERNAL TABLE ods_cart_info_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,user_id :STRING,sku_id :STRING,cart_price :DECIMAL(16,2),sku_num :BIGINT,img_url :STRING,sku_name :STRING,is_checked :STRING,create_time :STRING,operate_time :STRING,is_ordered :STRING,order_time :STRING,source_type :STRING,source_id :STRING> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值'
) COMMENT '购物车增量表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_cart_info_inc/';

```

5.2.17 评论表（增量表）

```

DROP TABLE IF EXISTS ods_comment_info_inc;
CREATE EXTERNAL TABLE ods_comment_info_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,user_id :STRING,nick_name :STRING,head_img :STRING,sku_id :STRING,spu_id :STRING,order_id :STRING,appraise :STRING,comment_txt :STRING,create_time :STRING,operate_time :STRING> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值'
) COMMENT '评价表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_comment_info_inc/';

```

5.2.18 优惠券领用表（增量表）

```

DROP TABLE IF EXISTS ods_coupon_use_inc;
CREATE EXTERNAL TABLE ods_coupon_use_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,coupon_id :STRING,user_id :STRING,order_id :STRING,coupon_status :STRING,get_time :STRING,using_time :STRING,used_time :STRING,expire_time :STRING> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值'
) COMMENT '优惠券领用表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_coupon_use_inc/';

```

5.2.19 收藏表（增量表）

```

DROP TABLE IF EXISTS ods_favor_info_inc;
CREATE EXTERNAL TABLE ods_favor_info_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,user_id :STRING,sku_id :STRING,spu_id :STRING,is_cancelled :STRING,create_time :STRING,cancel_time :STRING> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值'
) COMMENT '收藏表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_favor_info_inc/';

```

5.2.20 订单明细表（增量表）

```

DROP TABLE IF EXISTS ods_order_detail_inc;
CREATE EXTERNAL TABLE ods_order_detail_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,order_id :STRING,sku_id :STRING,sku_name :STRING,img_url :STRING,order_price :DECIMAL(16,2),sku_num :BIGINT,create_time :STRING,source_type :STRING,source_id :STRING,split_total_amount :DECIMAL(16,2),split_activity_amount :DECIMAL(16,2),split_coupon_amount :DECIMAL(16,2)> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值'
) COMMENT '订单明细表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_order_detail_inc/';

```

5.2.21 订单明细活动关联表（增量表）

```

DROP TABLE IF EXISTS ods_order_detail_activity_inc;
CREATE EXTERNAL TABLE ods_order_detail_activity_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,order_id :STRING,order_detail_id :STRING,activity_id :STRING,activity_rule_id :STRING,sku_id :STRING,create_time :STRING> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值'
) COMMENT '订单明细活动关联表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_order_detail_activity_inc/';

```

5.2.22 订单明细优惠券关联表（增量表）

```
DROP TABLE IF EXISTS ods_order_detail_coupon_inc;
CREATE EXTERNAL TABLE ods_order_detail_coupon_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,order_id :STRING,order_detail_id :STRING,coupon_id :STRING,coupon_use_id :STRING,sku_id :STRING,create_time :STRING> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值'
) COMMENT '订单明细优惠券关联表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_order_detail_coupon_inc/';
```

5.2.23 订单表（增量表）

```
DROP TABLE IF EXISTS ods_order_info_inc;
CREATE EXTERNAL TABLE ods_order_info_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,consignee :STRING,consignee_tel :STRING,total_amount :DECIMAL(16, 2),order_status :STRING,user_id :STRING,payment_way :STRING,delivery_address :STRING,order_comment :STRING,out_trade_no :STRING,trade_body :STRING,create_time :STRING,operate_time :STRING,expire_time :STRING,process_status :STRING,tracking_no :STRING,parent_order_id :STRING,img_url :STRING,province_id :STRING,activity_reduce_amount :DECIMAL(16, 2),coupon_reduce_amount :DECIMAL(16, 2),original_total_amount :DECIMAL(16, 2),freight_fee :DECIMAL(16, 2),freight_fee_reduce :DECIMAL(16, 2),refundable_time :DECIMAL(16, 2)> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值'
) COMMENT '订单表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_order_info_inc/';
```

5.2.24 退单表（增量表）

```
DROP TABLE IF EXISTS ods_order_refund_info_inc;
CREATE EXTERNAL TABLE ods_order_refund_info_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,user_id :STRING,order_id :STRING,sku_id :STRING,refund_type :STRING,refund_num :BIGINT,refund_amount :DECIMAL(16, 2),refund_reason_type :STRING,refund_reason_txt :STRING,refund_status :S
```

```

TRING,create_time
    :STRING> COMMENT '数据',
`old` MAP<STRING,STRING> COMMENT '旧值',
) COMMENT '退单表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_order_refund_info_inc/';

```

5.2.25 订单状态流水表（增量表）

```

DROP TABLE IF EXISTS ods_order_status_log_inc;
CREATE EXTERNAL TABLE ods_order_status_log_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,order_id :STRING,order_status :STRING,operate_time :STRING> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值',
) COMMENT '退单表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_order_status_log_inc/';

```

5.2.26 支付表（增量表）

```

DROP TABLE IF EXISTS ods_payment_info_inc;
CREATE EXTERNAL TABLE ods_payment_info_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,out_trade_no :STRING,order_id :STRING,user_id :STRING,payment_type :STRING,trade_no :STRING,total_amount :DECIMAL(16,2),subject :STRING,payment_status :STRING,create_time :STRING,callback_time :STRING,callback_content :STRING> COMMENT '数据',
    `old` MAP<STRING,STRING> COMMENT '旧值',
) COMMENT '支付表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_payment_info_inc/';

```

5.2.27 退款表（增量表）

```

DROP TABLE IF EXISTS ods_refund_payment_inc;
CREATE EXTERNAL TABLE ods_refund_payment_inc
(
    `type` STRING COMMENT '变动类型',
    `ts` BIGINT COMMENT '变动时间',
    `data` STRUCT<id :STRING,out_trade_no :STRING,order_id :STRING,sku_id :STRING,payment_type :STRING,trade_no :STRING,total_amount :DECIMAL(16,2),subject :STRING,refund_status :STRING,create_time :STRING,callback_ti

```

```

me :STRING,callback_content
      :STRING> COMMENT '数据',
`old` MAP<STRING,STRING> COMMENT '旧值',
) COMMENT '退款表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_refund_payment_inc/';

```

5.2.28 用户表（增量表）

```

DROP TABLE IF EXISTS ods_user_info_inc;
CREATE EXTERNAL TABLE ods_user_info_inc
(
  `type` STRING COMMENT '变动类型',
  `ts` BIGINT COMMENT '变动时间',
  `data` STRUCT<id :STRING,login_name :STRING,nick_name :STRING,passwd :STRING,
name :STRING,phone_num :STRING,email
          :STRING,head_img :STRING,user_level :STRING,birthday :STRING,
gender :STRING,create_time :STRING,operate_time
          :STRING,status :STRING> COMMENT '数据',
  `old` MAP<STRING,STRING> COMMENT '旧值',
) COMMENT '用户表'
PARTITIONED BY (`dt` STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.JsonSerDe'
LOCATION '/warehouse/gmall/ods/ods_user_info_inc/';

```

5.2.29 数据装载脚本

(1) 在 Master 的/home/hadoop/bin 目录下创建 hdfs_to_ods_db.sh

```
[hadoop@Master bin]$ vim hdfs_to_ods_db.sh
```

(2) 编写如下内容

```

#!/bin/bash

APP=gmall

if [ -n "$2" ] ;then
  do_date=$2
else
  do_date=`date -d '-1 day' +%F`
fi

load_data(){
  sql=""
  for i in $*; do
    #判断路径是否存在
    hadoop fs -test -e /origin_data/$APP/db/${i:4}/$do_date
    #路径存在方可装载数据
    if [[ $? = 0 ]]; then
      sql=$sql"load          data          inpath
'/origin_data/$APP/db/${i:4}/$do_date' OVERWRITE into table ${APP}.\$i
partition(dt='$do_date');"
      fi
  done
}

```

```

    hive -e "$sql"
}

case $1 in
    "ods_activity_info_full")
        load_data "ods_activity_info_full"
;;
    "ods_activity_rule_full")
        load_data "ods_activity_rule_full"
;;
    "ods_base_category1_full")
        load_data "ods_base_category1_full"
;;
    "ods_base_category2_full")
        load_data "ods_base_category2_full"
;;
    "ods_base_category3_full")
        load_data "ods_base_category3_full"
;;
    "ods_base_dic_full")
        load_data "ods_base_dic_full"
;;
    "ods_base_province_full")
        load_data "ods_base_province_full"
;;
    "ods_base_region_full")
        load_data "ods_base_region_full"
;;
    "ods_base_trademark_full")
        load_data "ods_base_trademark_full"
;;
    "ods_cart_info_full")
        load_data "ods_cart_info_full"
;;
    "ods_coupon_info_full")
        load_data "ods_coupon_info_full"
;;
    "ods_sku_attr_value_full")
        load_data "ods_sku_attr_value_full"
;;
    "ods_sku_info_full")
        load_data "ods_sku_info_full"
;;
    "ods_sku_sale_attr_value_full")
        load_data "ods_sku_sale_attr_value_full"
;;
    "ods_spu_info_full")
        load_data "ods_spu_info_full"
;;
    "ods_cart_info_inc")
        load_data "ods_cart_info_inc"
;;
    "ods_comment_info_inc")
;
```

```

        load_data "ods_comment_info_inc"
;;
"ods_coupon_use_inc")
    load_data "ods_coupon_use_inc"
;;
"ods_favor_info_inc")
    load_data "ods_favor_info_inc"
;;
"ods_order_detail_inc")
    load_data "ods_order_detail_inc"
;;
"ods_order_detail_activity_inc")
    load_data "ods_order_detail_activity_inc"
;;
"ods_order_detail_coupon_inc")
    load_data "ods_order_detail_coupon_inc"
;;
"ods_order_info_inc")
    load_data "ods_order_info_inc"
;;
"ods_order_refund_info_inc")
    load_data "ods_order_refund_info_inc"
;;
"ods_order_status_log_inc")
    load_data "ods_order_status_log_inc"
;;
"ods_payment_info_inc")
    load_data "ods_payment_info_inc"
;;
"ods_refund_payment_inc")
    load_data "ods_refund_payment_inc"
;;
"ods_user_info_inc")
    load_data "ods_user_info_inc"
;;
"all")
    load_data      "ods_activity_info_full"      "ods_activity_rule_full"
"ods_base_category1_full"          "ods_base_category2_full"
"ods_base_category3_full"          "ods_base_dic_full"      "ods_base_province_full"
"ods_base_region_full"            "ods_base_trademark_full"  "ods_cart_info_full"
"ods_coupon_info_full"            "ods_sku_attr_value_full"  "ods_sku_info_full"
"ods_sku_sale_attr_value_full"    "ods_spu_info_full"      "ods_cart_info_inc"
"ods_comment_info_inc"            "ods_coupon_use_inc"    "ods_favor_info_inc"
"ods_order_detail_inc"           "ods_order_detail_activity_inc"
"ods_order_detail_coupon_inc"     "ods_order_info_inc"
"ods_order_refund_info_inc"       "ods_order_status_log_inc"
"ods_payment_info_inc"           "ods_refund_payment_inc" "ods_user_info_inc"
;;
esac

```

(3) 增加脚本执行权限

```
[hadoop@Master bin]$ chmod +x hdfs_to_ods_db.sh
```

(4) 脚本用法

```
[hadoop@Master bin]$ hdfs_to_ods_db.sh all 2020-06-14
```

5.3 结果展示

所使用的脚本

```

hadoop@Master:~/bin$ ls
clean-snap.sh  gen_import_config.py  log.sh
cluster.sh      gen import config.sh mxw.sh
env.sh          hdfs_to_ods_db.sh    myhadoop.sh
f1.sh           hdfs_to_ods_log.sh  mysql_to_hdfs_full.sh
f2.sh           jpsall            mysql_to_kafka_inc_init.sh
f3.sh           kf.sh             ods_to_dim_init.sh

```

cluster.sh 数据采集平台自动启停的脚本

hdfs_to_ods_db 将业务数据从 hdfs 转储到 Hive on Spark 数据仓库

hdfs_to_ods_log 将用户行为日志数据从 hdfs 转储到 Hive on Spark 数据仓库

下图为装载数据成功的用户行为日志数据仓库:

下图为正在运行的 Hive on Spark 数据装载任务

Application Overview

User: hadoop
Name: Hive on Spark (sessionId = e6f25dcb-4771-488e-b9cb-131326c6cbc7)
Application Type: SPARK
Application Tags:
Application Priority: 0 (Higher Integer value indicates higher priority)
YarnApplicationState: RUNNING: AM has registered with RM and started running.
Queue: default
FinalStatus Reported by AM: Application has not completed yet.
Started: 星期一 六月 19 20:36:43 +0800 2023
Launched: 星期一 六月 19 20:36:45 +0800 2023
Finished: N/A
Elapsed: 2mins, 23sec
Tracking URL: ApplicationMaster
Log Aggregation Status: NOT_START
Application Timeout (Remaining Time): Unlimited
Diagnostics:
Unmanaged Application: false
Application Node Label expression: <Not set>
AM container Node Label expression: <DEFAULT_PARTITION>

Application Metrics

Total Resource Preempted: <memory:0, vCores:0>
Total Number of Non-AM Containers Preempted: 0
Total Number of AM Containers Preempted: 0
Resource Preempted from Current Attempt: <memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt: 0
Aggregate Resource Allocation: 816325 MB-seconds, 396 vcore-seconds
Aggregate Preempted Resource Allocation: 0 MB-seconds, 0 vcore-seconds

Show 20	entries	Search:			
Attempt ID	Started	Node	Logs	Nodes blacklisted by the app	Nodes blacklisted by the system
appattempt_1687160726624_0001_000001	Mon Jun 19 20:36:43 +0800 2023	http://Slave2:8042	Logs	0	0

Showing 1 to 1 of 1 entries First Previous 1 Next Last

第六章 数仓开发之 DIM 层

DIM 层设计要点：

- (1) DIM 层的设计依据是维度建模理论，该层存储维度模型的维度表。
- (2) DIM 层的数据存储格式为 orc 列式存储+snappy 压缩。
- (3) DIM 层表名的命名规范为 dim_ 表名_ 全量表或者拉链表标识 (full/zip)

6.1 商品维度表

1) 建表语句

```
DROP TABLE IF EXISTS dim_sku_full;
CREATE EXTERNAL TABLE dim_sku_full
(
  `id`          STRING COMMENT 'sku_id',
  `price`       DECIMAL(16, 2) COMMENT '商品价格',
  `sku_name`    STRING COMMENT '商品名称',
  `sku_desc`    STRING COMMENT '商品描述',
  `weight`      DECIMAL(16, 2) COMMENT '重量',
  `is_sale`     BOOLEAN COMMENT '是否在售',
  `spu_id`      STRING COMMENT 'spu 编号',
  `spu_name`    STRING COMMENT 'spu 名称',
  `category3_id` STRING COMMENT '三级分类 id',
  `category3_name` STRING COMMENT '三级分类名称',
  `category2_id` STRING COMMENT '二级分类 id',
  `category2_name` STRING COMMENT '二级分类名称'
)
```

```

`category2_name`           STRING COMMENT '二级分类名称',
`category1_id`             STRING COMMENT '一级分类 id',
`category1_name`           STRING COMMENT '一级分类名称',
`tm_id`                   STRING COMMENT '品牌 id',
`tm_name`                 STRING COMMENT '品牌名称',
`sku_attr_values`          STRING COMMENT '平台属性',
ARRAY<STRUCT<attr_id :STRING,value_id :STRING,attr_name :STRING,value_na
me:STRING>> COMMENT '平台属性',
`sku_sale_attr_values`    STRING COMMENT '销售属性',
ARRAY<STRUCT<sale_attr_id :STRING,sale_attr_value_id :STRING,sale_attr_n
ame :STRING,sale_attr_value_name:STRING>> COMMENT '销售属性',
`create_time`              STRING COMMENT '创建时间'
) COMMENT '商品维度表',
PARTITIONED BY (`dt` STRING)
STORED AS ORC
LOCATION '/warehouse/gmall/dim/dim_sku_full/'
TBLPROPERTIES ('orc.compress' = 'snappy');

```

2) 数据装载

```

with
sku as
(
  select
    id,
    price,
    sku_name,
    sku_desc,
    weight,
    is_sale,
    spu_id,
    category3_id,
    tm_id,
    create_time
  from ods_sku_info_full
  where dt='2020-06-14'
),
spu as
(
  select
    id,
    spu_name
  from ods_spu_info_full
  where dt='2020-06-14'
),
c3 as
(
  select
    id,
    name,
    category2_id
  from ods_base_category3_full
  where dt='2020-06-14'
),

```

```

c2 as
(
  select
    id,
    name,
    category1_id
  from ods_base_category2_full
  where dt='2020-06-14'
),
c1 as
(
  select
    id,
    name
  from ods_base_category1_full
  where dt='2020-06-14'
),
tm as
(
  select
    id,
    tm_name
  from ods_base_trademark_full
  where dt='2020-06-14'
),
attr as
(
  select
    sku_id,
    collect_set(named_struct('attr_id', attr_id, 'value_id', value_id, 'attr_name', attr_name, 'value_name', value_name)) attrs
    from ods_sku_attr_value_full
    where dt='2020-06-14'
    group by sku_id
),
sale_attr as
(
  select
    sku_id,
    collect_set(named_struct('sale_attr_id', sale_attr_id, 'sale_attr_value_id', sale_attr_value_id, 'sale_attr_value_name', sale_attr_value_name, 'sale_attr_value_name')) saleAttrs
    from ods_sku_sale_attr_value_full
    where dt='2020-06-14'
    group by sku_id
)
insert overwrite table dim_sku_full partition(dt='2020-06-14')
select
  sku.id,
  sku.price,
  sku.sku_name,
  sku.sku_desc,
  c2.category1_id,
  c1.name,
  tm.tm_name,
  attrs,
  saleAttrs
)

```

```

sku.weight,
sku.is_sale,
sku.spu_id,
spu.spu_name,
sku.category3_id,
c3.name,
c3.category2_id,
c2.name,
c2.category1_id,
c1.name,
sku.tm_id,
tm.tm_name,
attr.attrs,
sale_attr.sale_attrs,
sku.create_time
from sku
left join spu on sku.spu_id=spu.id
left join c3 on sku.category3_id=c3.id
left join c2 on c3.category2_id=c2.id
left join c1 on c2.category1_id=c1.id
left join tm on sku.tm_id=tm.id
left join attr on sku.id=attr.sku_id
left join sale_attr on sku.id=sale_attr.sku_id;

```

6.2 优惠券维度表

1) 建表语句

```

DROP TABLE IF EXISTS dim_coupon_full;
CREATE EXTERNAL TABLE dim_coupon_full
(
  `id`          STRING COMMENT '购物券编号',
  `coupon_name` STRING COMMENT '购物券名称',
  `coupon_type_code` STRING COMMENT '购物券类型编码',
  `coupon_type_name` STRING COMMENT '购物券类型名称',
  `condition_amount` DECIMAL(16, 2) COMMENT '满额数',
  `condition_num` BIGINT COMMENT '满件数',
  `activity_id` STRING COMMENT '活动编号',
  `benefit_amount` DECIMAL(16, 2) COMMENT '减金额',
  `benefit_discount` DECIMAL(16, 2) COMMENT '折扣',
  `benefit_rule` STRING COMMENT '优惠规则:满元*减*元, 满*件打*折',
  `create_time` STRING COMMENT '创建时间',
  `range_type_code` STRING COMMENT '优惠范围类型编码',
  `range_type_name` STRING COMMENT '优惠范围类型名称',
  `limit_num` BIGINT COMMENT '最多领取次数',
  `taken_count` BIGINT COMMENT '已领取次数',
  `start_time` STRING COMMENT '可以领取的开始日期',
  `end_time` STRING COMMENT '可以领取的结束日期',
  `operate_time` STRING COMMENT '修改时间',
  `expire_time` STRING COMMENT '过期时间'
) COMMENT '优惠券维度表';

```

```
PARTITIONED BY (`dt` STRING)
STORED AS ORC
LOCATION '/warehouse/gmall/dim/dim_coupon_full/'
TBLPROPERTIES ('orc.compress' = 'snappy');
```

2) 数据装载

```
insert overwrite table dim_coupon_full partition(dt='2020-06-14')
select
    id,
    coupon_name,
    coupon_type,
    coupon_dic.dic_name,
    condition_amount,
    condition_num,
    activity_id,
    benefit_amount,
    benefit_discount,
    case coupon_type
        when '3201' then concat(' 满 ', condition_amount, ' 元 减',
        benefit_amount, '元')
        when '3202' then concat(' 满 ', condition_num, ' 件 打 ', 10*(1-
        benefit_discount), '折')
        when '3203' then concat('减', benefit_amount, '元')
    end benefit_rule,
    create_time,
    range_type,
    range_dic.dic_name,
    limit_num,
    taken_count,
    start_time,
    end_time,
    operate_time,
    expire_time
from
(
    select
        id,
        coupon_name,
        coupon_type,
        condition_amount,
        condition_num,
        activity_id,
        benefit_amount,
        benefit_discount,
        create_time,
        range_type,
        limit_num,
        taken_count,
        start_time,
        end_time,
        operate_time,
        expire_time
    from ods_coupon_info_full
    where dt='2020-06-14'
```

```

)ci
left join
(
    select
        dic_code,
        dic_name
    from ods_base_dic_full
    where dt='2020-06-14'
        and parent_code='32'
)coupon_dic
on ci.coupon_type=coupon_dic.dic_code
left join
(
    select
        dic_code,
        dic_name
    from ods_base_dic_full
    where dt='2020-06-14'
        and parent_code='33'
)range_dic
on ci.range_type=range_dic.dic_code;

```

6.3 活动维度表

1) 建表语句

```

DROP TABLE IF EXISTS dim_activity_full;
CREATE EXTERNAL TABLE dim_activity_full
(
    `activity_rule_id`      STRING COMMENT '活动规则 ID',
    `activity_id`           STRING COMMENT '活动 ID',
    `activity_name`         STRING COMMENT '活动名称',
    `activity_type_code`    STRING COMMENT '活动类型编码',
    `activity_type_name`    STRING COMMENT '活动类型名称',
    `activity_desc`         STRING COMMENT '活动描述',
    `start_time`            STRING COMMENT '开始时间',
    `end_time`              STRING COMMENT '结束时间',
    `create_time`            STRING COMMENT '创建时间',
    `condition_amount`      DECIMAL(16, 2) COMMENT '满减金额',
    `condition_num`          BIGINT COMMENT '满减件数',
    `benefit_amount`         DECIMAL(16, 2) COMMENT '优惠金额',
    `benefit_discount`      DECIMAL(16, 2) COMMENT '优惠折扣',
    `benefit_rule`           STRING COMMENT '优惠规则',
    `benefit_level`          STRING COMMENT '优惠级别'
) COMMENT '活动信息表'
PARTITIONED BY (`dt` STRING)
STORED AS ORC
LOCATION '/warehouse/gmall/dim/dim_activity_full/'
TBLPROPERTIES ('orc.compress' = 'snappy');

```

2) 数据装载

```
insert overwrite table dim_activity_full partition(dt='2020-06-14')
```

```

select
    rule.id,
    info.id,
    activity_name,
    rule.activity_type,
    dic.dic_name,
    activity_desc,
    start_time,
    end_time,
    create_time,
    condition_amount,
    condition_num,
    benefit_amount,
    benefit_discount,
    case rule.activity_type
        when '3101' then concat(' 满 ', condition_amount, ' 元 减',
        benefit_amount, '元')
        when '3102' then concat(' 满 ', condition_num, ' 件 打 ', 10*(1-
        benefit_discount), '折')
        when '3103' then concat('打', 10*(1-benefit_discount), '折')
    end benefit_rule,
    benefit_level
from
(
    select
        id,
        activity_id,
        activity_type,
        condition_amount,
        condition_num,
        benefit_amount,
        benefit_discount,
        benefit_level
    from ods_activity_rule_full
    where dt='2020-06-14'
)rule
left join
(
    select
        id,
        activity_name,
        activity_type,
        activity_desc,
        start_time,
        end_time,
        create_time
    from ods_activity_info_full
    where dt='2020-06-14'
)info
on rule.activity_id=info.id
left join
(
    select
        dic_code,

```

```

    dic_name
from ods_base_dic_full
where dt='2020-06-14'
and parent_code='31'
)dic
on rule.activity_type=dic.dic_code;

```

6.4 地区维度表

1) 建表语句

```

DROP TABLE IF EXISTS dim_province_full;
CREATE EXTERNAL TABLE dim_province_full
(
  `id`          STRING COMMENT 'id',
  `province_name` STRING COMMENT '省市名称',
  `area_code`    STRING COMMENT '地区编码',
  `iso_code`     STRING COMMENT '旧版 ISO-3166-2 编码, 供可视化使用',
  `iso_3166_2`   STRING COMMENT '新版 IOS-3166-2 编码, 供可视化使用',
  `region_id`    STRING COMMENT '地区 id',
  `region_name`  STRING COMMENT '地区名称'
) COMMENT '地区维度表'
PARTITIONED BY (`dt` STRING)
STORED AS ORC
LOCATION '/warehouse/gmall/dim/dim_province_full/'
TBLPROPERTIES ('orc.compress' = 'snappy');

```

2) 数据装载

```

insert overwrite table dim_province_full partition(dt='2020-06-14')
select
  province.id,
  province.name,
  province.area_code,
  province.iso_code,
  province.iso_3166_2,
  region_id,
  region_name
from
(
  select
    id,
    name,
    region_id,
    area_code,
    iso_code,
    iso_3166_2
  from ods_base_province_full
  where dt='2020-06-14'
)province
left join
(
  select
    id,
    region_name

```

```

from ods_base_region_full
where dt='2020-06-14'
)region
on province.region_id=region.id;

```

6.5 日期维度表

1) 建表语句

```

DROP TABLE IF EXISTS dim_date;
CREATE EXTERNAL TABLE dim_date
(
    `date_id`      STRING COMMENT '日期 ID',
    `week_id`      STRING COMMENT '周 ID,一年中的第几周',
    `week_day`     STRING COMMENT '周几',
    `day`          STRING COMMENT '每月的第几天',
    `month`        STRING COMMENT '一年中的第几月',
    `quarter`      STRING COMMENT '一年中的第几季度',
    `year`         STRING COMMENT '年份',
    `is_workday`   STRING COMMENT '是否是工作日',
    `holiday_id`   STRING COMMENT '节假日'
) COMMENT '时间维度表'
STORED AS ORC
LOCATION '/warehouse/gmall/dim/dim_date/'
TBLPROPERTIES ('orc.compress' = 'snappy');

```

2) 数据装载

通常情况下，时间维度表的数据并不是来自于业务系统，而是手动写入，并且由于时间维度表数据的可预见性，无须每日导入，一般可一次性导入一年的数据。

(1) 创建临时表

```

DROP TABLE IF EXISTS tmp_dim_date_info;
CREATE EXTERNAL TABLE tmp_dim_date_info (
    `date_id`      STRING COMMENT '日',
    `week_id`      STRING COMMENT '周 ID',
    `week_day`     STRING COMMENT '周几',
    `day`          STRING COMMENT '每月的第几天',
    `month`        STRING COMMENT '第几月',
    `quarter`      STRING COMMENT '第几季度',
    `year`         STRING COMMENT '年',
    `is_workday`   STRING COMMENT '是否是工作日',
    `holiday_id`   STRING COMMENT '节假日'
) COMMENT '时间维度表'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/warehouse/gmall/tmp/tmp_dim_date_info/';

```

(2) 将数据文件上传到 HDFS 上临时表路径

/warehouse/gmall/tmp/tmp_dim_date_info



date_info.txt

(3) 执行以下语句将其导入时间维度表

```
insert overwrite table dim_date select * from tmp_dim_date_info;
```

(4) 检查数据是否导入成功

```
select * from dim_date;
```

6.6 用户维度表

1) 建表语句

```
DROP TABLE IF EXISTS dim_user_zip;
CREATE EXTERNAL TABLE dim_user_zip
(
    `id`          STRING COMMENT '用户 id',
    `login_name`  STRING COMMENT '用户名称',
    `nick_name`   STRING COMMENT '用户昵称',
    `name`        STRING COMMENT '用户姓名',
    `phone_num`   STRING COMMENT '手机号码',
    `email`       STRING COMMENT '邮箱',
    `user_level`  STRING COMMENT '用户等级',
    `birthday`   STRING COMMENT '生日',
    `gender`     STRING COMMENT '性别',
    `create_time` STRING COMMENT '创建时间',
    `operate_time` STRING COMMENT '操作时间',
    `start_date`  STRING COMMENT '开始日期',
    `end_date`    STRING COMMENT '结束日期'
) COMMENT '用户表'
PARTITIONED BY (`dt` STRING)
STORED AS ORC
LOCATION '/warehouse/gmall/dim/dim_user_zip/'
TBLPROPERTIES ('orc.compress' = 'snappy');
```

2) 分区规划

用户拉链表分区

用户维度表

dt=2020-06-14
当日过期的用户数据

dt=2020-06-15
当日过期的用户数据

dt=2020-06-16
当日过期的用户数据

dt=2020-06-17
当日过期的用户数据

dt=2020-06-18
当日过期的用户数据

dt=9999-12-31
全量最新的用户数据

3) 数据装载

(1) 数据装载过程

1) 假设，2019年1月1日的用户全量表是最初始的用户表，如下

用户ID	姓名
1	张三
2	李四
3	王五



2) 初始的拉链表就等于最开始的2019年1月1日的用户全量表

用户ID	姓名	开始时间	结束时间
1	张三	2019-01-01	9999-12-31
2	李四	2019-01-01	9999-12-31
3	王五	2019-01-01	9999-12-31

3) 第二天1月2日用户全量表(用户2发生状态修改；用户4、5增加)

用户ID	姓名
1	张三
2	李小四
3	王五
4	赵六
5	田七



5) 用户变化表与之前的拉链表合并得到

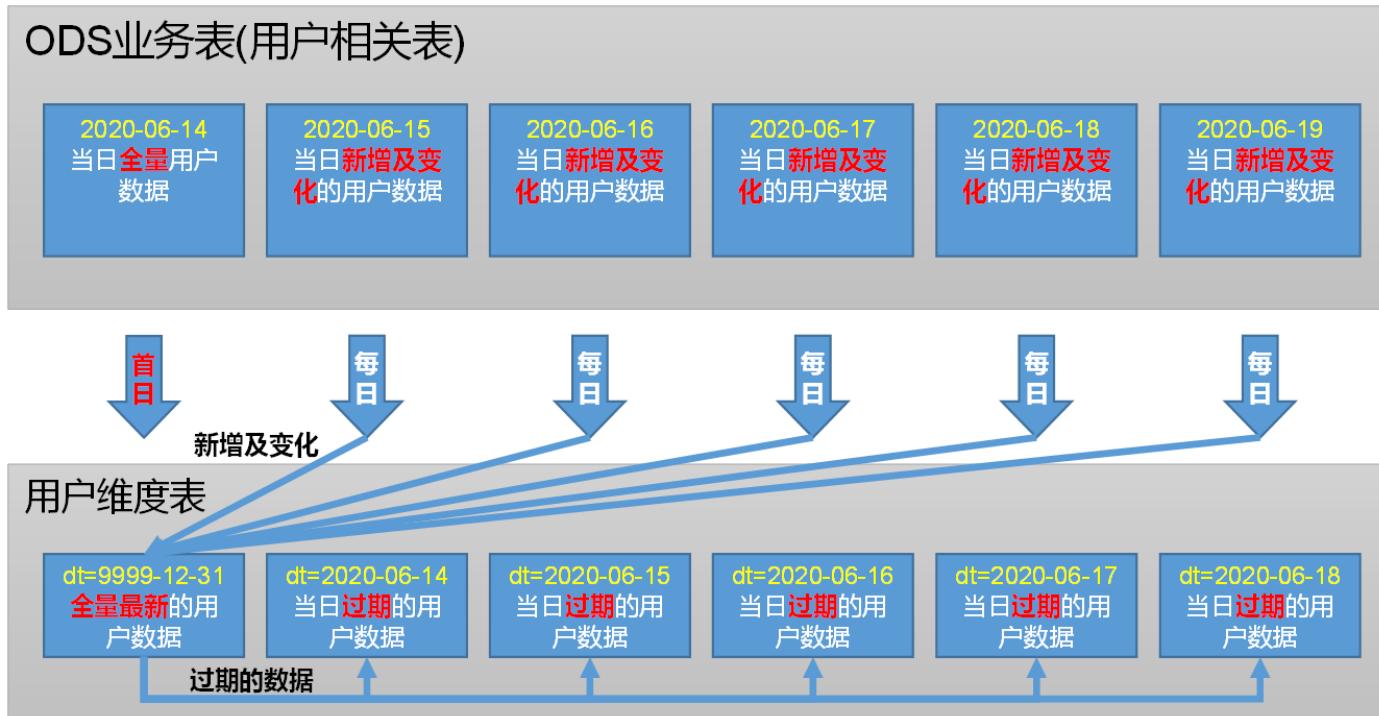
用户ID	姓名	开始时间	结束时间
1	张三	2019-01-01	9999-12-31
2	李四	2019-01-01	2019-01-01
2	李小四	2019-01-02	9999-12-31
3	王五	2019-01-01	9999-12-31
4	赵六	2019-01-02	9999-12-31
5	田七	2019-01-02	9999-12-31

4) 根据用户表的创建时间和操作时间，得到用户变化表。

用户ID	姓名
2	李小四
4	赵六
5	田七



(2) 数据流向



(3) 首日装载

```
insert overwrite table dim_user_zip partition (dt='9999-12-31')
select
    data.id,
    data.login_name,
    data.nick_name,
    md5(data.name),
    md5(data.phone_num),
    md5(data.email),
    data.user_level,
    data.birthday,
    data.gender,
```

```

data.create_time,
data.operate_time,
'2020-06-14' start_date,
'9999-12-31' end_date
from ods_user_info_inc
where dt='2020-06-14'
and type='bootstrap-insert';

```

(4) 每日装载

① 装载思路

拉链表9999-12-31分区，
截至前一日的全量最新

截至当日的全量最新，
需覆盖之前的9999-12-31分区



② 装载语句

```

with
tmp as
(
  select
    old.id old_id,
    old.login_name old_login_name,
    old.nick_name old_nick_name,
    old.name old_name,
    old.phone_num old_phone_num,
    old.email old_email,
    old.user_level old_user_level,
    old.birthday old_birthday,
    old.gender old_gender,
    old.create_time old_create_time,
    old.operate_time old_operate_time,
    old.start_date old_start_date,
    old.end_date old_end_date,
    new.id new_id,
    new.login_name new_login_name,
    new.nick_name new_nick_name,
    new.name new_name,
    new.phone_num new_phone_num,
    new.email new_email,
    new.user_level new_user_level,

```

```
new.birthday new_birthday,
new.gender new_gender,
new.create_time new_create_time,
new.operate_time new_operate_time,
new.start_date new_start_date,
new.end_date new_end_date
from
(
    select
        id,
        login_name,
        nick_name,
        name,
        phone_num,
        email,
        user_level,
        birthday,
        gender,
        create_time,
        operate_time,
        start_date,
        end_date
    from dim_user_zip
    where dt='9999-12-31'
)old
full outer join
(
    select
        id,
        login_name,
        nick_name,
        md5(name) name,
        md5(phone_num) phone_num,
        md5(email) email,
        user_level,
        birthday,
        gender,
        create_time,
        operate_time,
        '2020-06-15' start_date,
        '9999-12-31' end_date
    from
    (
        select
            data.id,
            data.login_name,
            data.nick_name,
            data.name,
            data.phone_num,
            data.email,
            data.user_level,
            data.birthday,
            data.gender,
            data.create_time,
```

```

        data.operate_time,
        row_number() over (partition by data.id order by ts desc)
rn
        from ods_user_info_inc
        where dt='2020-06-15'
    )t1
    where rn=1
)new
on old.id=new.id
)
insert overwrite table dim_user_zip partition(dt)
select
    if(new_id is not null,new_id,old_id),
    if(new_id is not null,new_login_name,old_login_name),
    if(new_id is not null,new_nick_name,old_nick_name),
    if(new_id is not null,new_name,old_name),
    if(new_id is not null,new_phone_num,old_phone_num),
    if(new_id is not null,new_email,old_email),
    if(new_id is not null,new_user_level,old_user_level),
    if(new_id is not null,new_birthday,old_birthday),
    if(new_id is not null,new_gender,old_gender),
    if(new_id is not null,new_create_time,old_create_time),
    if(new_id is not null,new_operate_time,old_operate_time),
    if(new_id is not null,new_start_date,old_start_date),
    if(new_id is not null,new_end_date,old_end_date),
    if(new_id is not null,new_end_date,old_end_date) dt
from tmp
union all
select
    old_id,
    old_login_name,
    old_nick_name,
    old_name,
    old_phone_num,
    old_email,
    old_user_level,
    old_birthday,
    old_gender,
    old_create_time,
    old_operate_time,
    old_start_date,
    cast(date_add('2020-06-15',-1) as string) old_end_date,
    cast(date_add('2020-06-15',-1) as string) dt
from tmp
where old_id is not null
and new_id is not null;

```

6.7 数据装载脚本

6.7.1 首日装载脚本

(1) 在 Master 的/home/hadoop/bin 目录下创建 ods_to_dim_init.sh

```
[hadoop@Master bin]$ vim ods_to_dim_init.sh
```

(2) 编写如下内容

```

#!/bin/bash

APP=gmall

if [ -n "$2" ] ;then
    do_date=$2
else
    echo "请传入日期参数"
    exit
fi

dim_user_zip="
insert overwrite table ${APP}.dim_user_zip partition (dt='9999-12-31')
select
    data.id,
    data.login_name,
    data.nick_name,
    md5(data.name),
    md5(data.phone_num),
    md5(data.email),
    data.user_level,
    data.birthday,
    data.gender,
    data.create_time,
    data.operate_time,
    '$do_date' start_date,
    '9999-12-31' end_date
from ${APP}.ods_user_info_inc
where dt='$do_date'
and type='bootstrap-insert';
"

dim_sku_full="
with
sku as
(
    select
        id,
        price,
        sku_name,
        sku_desc,
        weight,
        is_sale,
        spu_id,
        category3_id,
        tm_id,
        create_time
    from ${APP}.ods_sku_info_full
    where dt='$do_date'
),
spu as
(
    select
        id,

```

```

    spu_name
  from ${APP}.ods_spu_info_full
  where dt='$do_date'
),
c3 as
(
  select
    id,
    name,
    category2_id
  from ${APP}.ods_base_category3_full
  where dt='$do_date'
),
c2 as
(
  select
    id,
    name,
    category1_id
  from ${APP}.ods_base_category2_full
  where dt='$do_date'
),
c1 as
(
  select
    id,
    name
  from ${APP}.ods_base_category1_full
  where dt='$do_date'
),
tm as
(
  select
    id,
    tm_name
  from ${APP}.ods_base_trademark_full
  where dt='$do_date'
),
attr as
(
  select
    sku_id,
    collect_set(named_struct('attr_id', attr_id, 'value_id', value_id, 'attr_name', attr_name, 'value_name', value_name)) attrs
    from ${APP}.ods_sku_attr_value_full
    where dt='$do_date'
    group by sku_id
),
sale_attr as
(
  select
    sku_id,

```

```

collect_set(named_struct('sale_attr_id',sale_attr_id,'sale_attr_value_id',
'sale_attr_value_id','sale_attr_name',sale_attr_name,'sale_attr_value_na
me',sale_attr_value_name)) saleAttrs
    from ${APP}.ods_sku_sale_attr_value_full
    where dt='$do_date'
    group by sku_id
)
insert overwrite table ${APP}.dim_sku_full partition(dt='$do_date')
select
    sku.id,
    sku.price,
    sku.sku_name,
    sku.sku_desc,
    sku.weight,
    sku.is_sale,
    sku.spu_id,
    spu.spu_name,
    sku.category3_id,
    c3.name,
    c3.category2_id,
    c2.name,
    c2.category1_id,
    c1.name,
    sku.tm_id,
    tm.tm_name,
    attr.attrs,
    sale_attr.saleAttrs,
    sku.create_time
from sku
left join spu on sku.spu_id=spu.id
left join c3 on sku.category3_id=c3.id
left join c2 on c3.category2_id=c2.id
left join c1 on c2.category1_id=c1.id
left join tm on sku.tm_id=tm.id
left join attr on sku.id=attr.sku_id
left join sale_attr on sku.id=sale_attr.sku_id;
"
dim_province_full=""
insert overwrite table ${APP}.dim_province_full partition(dt='$do_date')
select
    province.id,
    province.name,
    province.area_code,
    province.iso_code,
    province.iso_3166_2,
    region_id,
    region_name
from
(
    select
        id,
        name,
        region_id,

```

```

    area_code,
    iso_code,
    iso_3166_2
from ${APP}.ods_base_province_full
where dt='$do_date'
)province
left join
(
    select
        id,
        region_name
    from ${APP}.ods_base_region_full
    where dt='$do_date'
)region
on province.region_id=region.id;
"

dim_coupon_full="
insert overwrite table ${APP}.dim_coupon_full partition(dt='$do_date')
select
    id,
    coupon_name,
    coupon_type,
    coupon_dic.dic_name,
    condition_amount,
    condition_num,
    activity_id,
    benefit_amount,
    benefit_discount,
    case coupon_type
        when '3201' then concat(' 满 ',condition_amount,' 元 减',
        benefit_amount,'元')
        when '3202' then concat(' 满 ',condition_num,' 件 打 ',10*(1-
        benefit_discount),'折')
        when '3203' then concat('减',benefit_amount,'元')
    end benefit_rule,
    create_time,
    range_type,
    range_dic.dic_name,
    limit_num,
    taken_count,
    start_time,
    end_time,
    operate_time,
    expire_time
from
(
    select
        id,
        coupon_name,
        coupon_type,
        condition_amount,
        condition_num,
        activity_id,

```

```

    benefit_amount,
    benefit_discount,
    create_time,
    range_type,
    limit_num,
    taken_count,
    start_time,
    end_time,
    operate_time,
    expire_time
  from ${APP}.ods_coupon_info_full
  where dt='$do_date'
)ci
left join
(
  select
    dic_code,
    dic_name
  from ${APP}.ods_base_dic_full
  where dt='$do_date'
  and parent_code='32'
)coupon_dic
on ci.coupon_type=coupon_dic.dic_code
left join
(
  select
    dic_code,
    dic_name
  from ${APP}.ods_base_dic_full
  where dt='$do_date'
  and parent_code='33'
)range_dic
on ci.range_type=range_dic.dic_code;
"
dim_activity_full=""
insert overwrite table ${APP}.dim_activity_full partition(dt='$do_date')
select
  rule.id,
  info.id,
  activity_name,
  rule.activity_type,
  dic.dic_name,
  activity_desc,
  start_time,
  end_time,
  create_time,
  condition_amount,
  condition_num,
  benefit_amount,
  benefit_discount,
  case rule.activity_type
    when '3101' then concat(' 满 ', condition_amount, ' 元 减
', benefit_amount, ' 元')
  end
)

```

```

        when '3102' then concat(' 满 ',condition_num,' 件 打 ',10*(1-
benefit_discount),'折')
        when '3103' then concat('打',10*(1-benefit_discount),'折')
    end benefit_rule,
    benefit_level
from
(
    select
        id,
        activity_id,
        activity_type,
        condition_amount,
        condition_num,
        benefit_amount,
        benefit_discount,
        benefit_level
    from ${APP}.ods_activity_rule_full
    where dt='$do_date'
)rule
left join
(
    select
        id,
        activity_name,
        activity_type,
        activity_desc,
        start_time,
        end_time,
        create_time
    from ${APP}.ods_activity_info_full
    where dt='$do_date'
)info
on rule.activity_id=info.id
left join
(
    select
        dic_code,
        dic_name
    from ${APP}.ods_base_dic_full
    where dt='$do_date'
    and parent_code='31'
)dic
on rule.activity_type=dic.dic_code;
"
case $1 in
"dim_user_zip")
    hive -e "$dim_user_zip"
;;
"dim_sku_full")
    hive -e "$dim_sku_full"
;;
"dim_province_full")
    hive -e "$dim_province_full"

```

```
;;
"dim_coupon_full")
    hive -e "$dim_coupon_full"
;;
"dim_activity_full")
    hive -e "$dim_activity_full"
;;
"all")
    hive
"$dim_user_zip$dim_sku_full$dim_province_full$dim_coupon_full$dim_activity_full"
;;
esac
```

(3) 增加脚本执行权限

```
[hadoop@Master bin]$ chmod +x ods_to_dim_init.sh
```

(4) 脚本用法

```
[hadoop@Master bin]$ ods_to_dim_init.sh all 2020-06-14
```

6.7.2 每日装载脚本

(1) 在 Master 的/home/hadoop/bin 目录下创建 ods_to_dim.sh

```
[hadoop@Master bin]$ vim ods_to_dim.sh
```

(2) 编写如下内容

```
#!/bin/bash

APP=gmall

# 如果是输入的日期按照取输入日期; 如果没输入日期取当前时间的前一天
if [ -n "$2" ] ;then
    do_date=$2
else
    do_date=`date -d "-1 day" +%F`
fi

dim_user_zip=""
set hive.exec.dynamic.partition.mode=nonstrict;
with
tmp as
(
    select
        old.id old_id,
        old.login_name old_login_name,
        old.nick_name old_nick_name,
        old.name old_name,
        old.phone_num old_phone_num,
        old.email old_email,
        old.user_level old_user_level,
        old.birthday old_birthday,
        old.gender old_gender,
        old.create_time old_create_time,
        old.operate_time old_operate_time,
        old.start_date old_start_date,
```

```

old.end_date old_end_date,
new.id new_id,
new.login_name new_login_name,
new.nick_name new_nick_name,
new.name new_name,
new.phone_num new_phone_num,
new.email new_email,
new.user_level new_user_level,
new.birthday new_birthday,
new.gender new_gender,
new.create_time new_create_time,
new.operate_time new_operate_time,
new.start_date new_start_date,
new.end_date new_end_date
from
(
    select
        id,
        login_name,
        nick_name,
        name,
        phone_num,
        email,
        user_level,
        birthday,
        gender,
        create_time,
        operate_time,
        start_date,
        end_date
    from ${APP}.dim_user_zip
    where dt='9999-12-31'
)old
full outer join
(
    select
        id,
        login_name,
        nick_name,
        md5(name) name,
        md5(phone_num) phone_num,
        md5(email) email,
        user_level,
        birthday,
        gender,
        create_time,
        operate_time,
        '$do_date' start_date,
        '9999-12-31' end_date
    from
    (
        select
            data.id,
            data.login_name,

```

```

        data.nick_name,
        data.name,
        data.phone_num,
        data.email,
        data.user_level,
        data.birthday,
        data.gender,
        data.create_time,
        data.operate_time,
        row_number() over (partition by data.id order by ts desc)
rn
    from ${APP}.ods_user_info_inc
    where dt='${do_date}'
) t1
    where rn=1
) new
on old.id=new.id
)
insert overwrite table ${APP}.dim_user_zip partition(dt)
select
    if(new_id is not null,new_id,old_id),
    if(new_id is not null,new_login_name,old_login_name),
    if(new_id is not null,new_nick_name,old_nick_name),
    if(new_id is not null,new_name,old_name),
    if(new_id is not null,new_phone_num,old_phone_num),
    if(new_id is not null,new_email,old_email),
    if(new_id is not null,new_user_level,old_user_level),
    if(new_id is not null,new_birthday,old_birthday),
    if(new_id is not null,new_gender,old_gender),
    if(new_id is not null,new_create_time,old_create_time),
    if(new_id is not null,new_operate_time,old_operate_time),
    if(new_id is not null,new_start_date,old_start_date),
    if(new_id is not null,new_end_date,old_end_date),
    if(new_id is not null,new_end_date,old_end_date) dt
from tmp
union all
select
    old_id,
    old_login_name,
    old_nick_name,
    old_name,
    old_phone_num,
    old_email,
    old_user_level,
    old_birthday,
    old_gender,
    old_create_time,
    old_operate_time,
    old_start_date,
    cast(date_add('${do_date}',-1) as string) old_end_date,
    cast(date_add('${do_date}',-1) as string) dt
from tmp
where old_id is not null
and new_id is not null;

```

```
"  
dim_sku_full="  
with  
sku as  
(  
    select  
        id,  
        price,  
        sku_name,  
        sku_desc,  
        weight,  
        is_sale,  
        spu_id,  
        category3_id,  
        tm_id,  
        create_time  
    from ${APP}.ods_sku_info_full  
    where dt='$do_date'  
)  
spu as  
(  
    select  
        id,  
        spu_name  
    from ${APP}.ods_spu_info_full  
    where dt='$do_date'  
)  
c3 as  
(  
    select  
        id,  
        name,  
        category2_id  
    from ${APP}.ods_base_category3_full  
    where dt='$do_date'  
)  
c2 as  
(  
    select  
        id,  
        name,  
        category1_id  
    from ${APP}.ods_base_category2_full  
    where dt='$do_date'  
)  
c1 as  
(  
    select  
        id,  
        name  
    from ${APP}.ods_base_category1_full  
    where dt='$do_date'  
)
```

```

tm as
(
  select
    id,
    tm_name
  from ${APP}.ods_base_trademark_full
  where dt='$do_date'
),
attr as
(
  select
    sku_id,
    collect_set(named_struct('attr_id', attr_id, 'value_id', value_id, 'attr_name', attr_name, 'value_name', value_name)) attrs
    from ${APP}.ods_sku_attr_value_full
    where dt='$do_date'
    group by sku_id
),
sale_attr as
(
  select
    sku_id,
    collect_set(named_struct('sale_attr_id', sale_attr_id, 'sale_attr_value_id', sale_attr_value_id, 'sale_attr_value_name', sale_attr_name, 'sale_attr_value_name', sale_attr_value_name)) saleAttrs
    from ${APP}.ods_sku_sale_attr_value_full
    where dt='$do_date'
    group by sku_id
)
insert overwrite table ${APP}.dim_sku_full partition(dt='$do_date')
select
  sku.id,
  sku.price,
  sku.sku_name,
  sku.sku_desc,
  sku.weight,
  sku.is_sale,
  sku.spu_id,
  spu.spu_name,
  sku.category3_id,
  c3.name,
  c3.category2_id,
  c2.name,
  c2.category1_id,
  c1.name,
  sku.tm_id,
  tm.tm_name,
  attr.attrs,
  sale_attr.saleAttrs,
  sku.create_time
from sku
left join spu on sku.spu_id=spu.id

```

```

left join c3 on sku.category3_id=c3.id
left join c2 on c3.category2_id=c2.id
left join c1 on c2.category1_id=c1.id
left join tm on sku.tm_id=tm.id
left join attr on sku.id=attr.sku_id
left join sale_attr on sku.id=sale_attr.sku_id;
"

dim_province_full=""
insert overwrite table ${APP}.dim_province_full partition(dt='$do_date')
select
    province.id,
    province.name,
    province.area_code,
    province.iso_code,
    province.iso_3166_2,
    region_id,
    region_name
from
(
    select
        id,
        name,
        region_id,
        area_code,
        iso_code,
        iso_3166_2
    from ${APP}.ods_base_province_full
    where dt='$do_date'
)province
left join
(
    select
        id,
        region_name
    from ${APP}.ods_base_region_full
    where dt='$do_date'
)region
on province.region_id=region.id;
"

dim_coupon_full=""
insert overwrite table ${APP}.dim_coupon_full partition(dt='$do_date')
select
    id,
    coupon_name,
    coupon_type,
    coupon_dic.dic_name,
    condition_amount,
    condition_num,
    activity_id,
    benefit_amount,
    benefit_discount,
    case coupon_type

```

```

        when '3201' then concat(' 满 ', condition_amount, ' 元 减
', benefit_amount, '元')
        when '3202' then concat(' 满 ', condition_num, ' 件 打 ', 10*(1-
benefit_discount), '折')
        when '3203' then concat('减', benefit_amount, '元')
end benefit_rule,
create_time,
range_type,
range_dic.dic_name,
limit_num,
taken_count,
start_time,
end_time,
operate_time,
expire_time
from
(
    select
        id,
        coupon_name,
        coupon_type,
        condition_amount,
        condition_num,
        activity_id,
        benefit_amount,
        benefit_discount,
        create_time,
        range_type,
        limit_num,
        taken_count,
        start_time,
        end_time,
        operate_time,
        expire_time
    from ${APP}.ods_coupon_info_full
    where dt='$do_date'
)ci
left join
(
    select
        dic_code,
        dic_name
    from ${APP}.ods_base_dic_full
    where dt='$do_date'
    and parent_code='32'
)coupon_dic
on ci.coupon_type=coupon_dic.dic_code
left join
(
    select
        dic_code,
        dic_name
    from ${APP}.ods_base_dic_full
    where dt='$do_date'

```

```

    and parent_code='33'
)range_dic
on ci.range_type=range_dic.dic_code;
"

dim_activity_full=""
insert overwrite table ${APP}.dim_activity_full partition(dt='$do_date')
select
    rule.id,
    info.id,
    activity_name,
    rule.activity_type,
    dic.dic_name,
    activity_desc,
    start_time,
    end_time,
    create_time,
    condition_amount,
    condition_num,
    benefit_amount,
    benefit_discount,
    case rule.activity_type
        when '3101' then concat(' 满 ',condition_amount,' 元 减 ',
        benefit_amount,'元')
        when '3102' then concat(' 满 ',condition_num,' 件 打 ',10*(1-
        benefit_discount),'折')
        when '3103' then concat('打',10*(1-benefit_discount),'折')
    end benefit_rule,
    benefit_level
from
(
    select
        id,
        activity_id,
        activity_type,
        condition_amount,
        condition_num,
        benefit_amount,
        benefit_discount,
        benefit_level
    from ${APP}.ods_activity_rule_full
    where dt='$do_date'
)rule
left join
(
    select
        id,
        activity_name,
        activity_type,
        activity_desc,
        start_time,
        end_time,
        create_time
    from ${APP}.ods_activity_info_full
)

```

```

        where dt='$do_date'
)info
on rule.activity_id=info.id
left join
(
    select
        dic_code,
        dic_name
    from ${APP}.ods_base_dic_full
    where dt='$do_date'
    and parent_code='31'
)dic
on rule.activity_type=dic.dic_code;
"
case $1 in
"dim_user_zip")
    hive -e "$dim_user_zip"
;;
"dim_sku_full")
    hive -e "$dim_sku_full"
;;
"dim_province_full")
    hive -e "$dim_province_full"
;;
"dim_coupon_full")
    hive -e "$dim_coupon_full"
;;
"dim_activity_full")
    hive -e "$dim_activity_full"
;;
"all")
    hive
"$dim_user_zip$dim_sku_full$dim_province_full$dim_coupon_full$dim_activity_full"
;;
esac

```

(3) 增加脚本执行权限

```
[hadoop@Master bin]$ chmod +x ods_to_dim.sh
```

(4) 脚本用法

```
[hadoop@Master bin]$ ods_to_dim.sh all 2020-06-14
```

6.8 结果展示

所使用的的脚本

hadoop@Master:~/bin\$ ls			
clean-snap.sh	gen_import_config.py	log.sh	ods_to_dim.sh
cluster.sh	gen_import_config.sh	mxw.sh	xsync
env.sh	hdfs_to_ods_db.sh	myhadoop.sh	zk.sh
f1.sh	hdfs_to_ods_log.sh	mysql_to_hdfs_full.sh	
f2.sh	jpsall	mysql_to_kafka_inc_init.sh	
f3.sh	kf.sh	ods_to_dim_init.sh	

`ods_to_dim_init` 从 ods 层的所有过往数据中提取 dim 层数据

`ods_to_dim` 从 ods 层的最新当日数据中提取 dim 层数据（主要是为了保持用户维度表 up to date）

下图是 `dim_sku_full` 建表工作的截图

```

1 DROP TABLE IF EXISTS dim_sku_full;
2 CREATE EXTERNAL TABLE dim_sku_full(
3     `id` STRING COMMENT 'sku_id',
4     `price` DECIMAL(16, 2) COMMENT '商品价格',
5     `sku_name` STRING COMMENT '商品名称',
6     `sku_desc` STRING COMMENT '商品描述',
7     `weight` DECIMAL(16, 2) COMMENT '重量',
8     `is_sale` BOOLEAN COMMENT '|是否在售',
9     `spu_id` STRING COMMENT 'spu编号',
10    `spu_name` STRING COMMENT 'spu名称',
11    `category3_id` STRING COMMENT '三级分类id',
12    `category3_name` STRING COMMENT '三级分类名称',
13    `category2_id` STRING COMMENT '二级分类id',
14    `category2_name` STRING COMMENT '二级分类名称',
15    `category1_id` STRING COMMENT '一级分类id',
16    `category1_name` STRING COMMENT '一级分类名称',
17    `tm_id` STRING COMMENT '品牌id',
18    `tm_name` STRING COMMENT '品牌名称',
19    `sku_attr_values` ARRAY<STRUCT<attr_id : STRING, value_id : STRING, attr_name : STRING, value_name : STRING>> COMMENT '平台属性',
20    `sku_sale_attr_values` ARRAY<STRUCT<sale_attr_id : STRING, sale_attr_value_id : STRING, sale_attr_name : STRING, sale_attr_value_name : STRING>> COMMENT '销售属性',
21    `create_time` STRING COMMENT '创建时间'
22 ) COMMENT '商品维度表'
23 PARTITIONED BY (`dt` STRING)
24 STORED AS ORC
25 LOCATION '/warehouse/gmall/dim/dim_sku_full/'
26 TBLPROPERTIES ('orc.compress' = 'snappy');
27
28 DROP TABLE IF EXISTS dim_coupon_full;
29 CREATE EXTERNAL TABLE dim_coupon_full(
30     ...
31 ) ...
32 
```

下图是一些装载完成的 dim 层数据展示

	1	4pnpsx1	沛昌	628be85ac...	62eeec1cbe8...	7be7b349a2...	1	1999-01-10	M	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
2	2	n4bocys	梅梅	073365b...	6f9f09dbb72f84...	6106f42de1...	1	1992-11-10	F	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
3	3	w32nql...	阿致	d079d622...	0b96fb174713d...	a1f650600f...	1	2000-05-10		2020-...	<null>	2020-06...	2020-06...	2020-06-13
4	4	iua7uc...	蓓蕾	83aaafae9...	e1f11945874b83...	8a87e69f79...	1	1984-12-10	F	2020-...	<null>	2020-06...	2020-06...	2020-06-13
5	5	dyzstf...	山仁	416a6519...	3295d552fc176...	3bb6ecf9e8...	1	1973-09-10	M	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
6	6	9f15r8...	贞菊	67ac6d64...	555e22fe45074...	b1c48380de...	1	1988-09-10		2020-...	<null>	2020-06...	2020-06...	2020-06-13
7	7	pj87bn...	军平	71cf85b5...	f0b648cc76425...	f713550fc0...	1	1991-03-10	M	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
8	8	yi96gl...	环雪	4ca1b8f9...	286298c05c707...	4f83a91610...	2	1992-09-10	F	2020-...	<null>	2020-06...	2020-06...	2020-06-13
9	9	hpsu5tm...	阿博	2f6ca4bb...	1c17fd148e634...	0ac3bf5d09...	2	1981-03-10		2020-...	<null>	2020-06...	2020-06...	2020-06-13
10	10	9um9v6...	娅娅	8a8e449f...	29df4d6a89ee2...	c32df43ec4...	1	1985-08-10	F	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
11	11	e3xk4x...	阿成	c3ca592f...	f5c72cdab86...	80384dc1cc...	1	1993-01-10	M	2020-...	<null>	2020-06...	2020-06...	2020-06-13
12	12	03j7iv...	卿卿	546664d4...	c1c755d34be7f...	48e011c93...	1	1993-02-10		2020-...	<null>	2020-06...	2020-06...	2020-06-13
13	13	ch2hi...	福生	9e0010f0...	358d687ec962...	8c9db81199...	2	2004-08-10	M	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
14	14	jqk9ag...	茜茜	f2e20fc5...	beb331d39ef41...	8d3d22cb85...	1	1966-04-10	F	2020-...	<null>	2020-06...	2020-06...	2020-06-13
15	15	leq9l3...	阿杰	2e1f8103...	dbe7b9c0eb810...	05e0160fae...	3	1969-09-10		2020-...	2020-...	2020-06...	2020-06...	2020-06-13
16	16	5thq7rp...	荣荣	a92d8e90...	a1795b2eef14c...	e5bcbbc8c...	2	1971-03-10	F	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
17	17	ng4hd1...	阿谦	b907aeef2...	7bc1cfb45f7be...	f06b2e17b1...	2	2003-10-10	M	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
18	18	fd16n7...	枫芸	f238bb47...	7d5a3423d7998...	6722b910a...	2	1998-10-10		2020-...	2020-...	2020-06...	2020-06...	2020-06-13
19	19	q6w84l...	阿安	4b36c2f27...	c442146f9537d...	41ea0e1329...	1	1987-01-10	M	2020-...	<null>	2020-06...	2020-06...	2020-06-13
20	20	7hqsoi...	玲玲	3328f798...	ba1226b1d5151...	8ce96a312e...	1	1999-12-10	F	2020-...	<null>	2020-06...	2020-06...	2020-06-13
21	21	emn6mk...	阿祥	7c19d0df...	08de216e82386...	585b0d7973...	1	1974-05-10		2020-...	2020-...	2020-06...	2020-06...	2020-06-13
22	22	832f6d...	纯纯	85bae01d...	cbfb87df0e28...	fc525e5c60...	1	1996-06-10	F	2020-...	<null>	2020-06...	2020-06...	2020-06-13
23	23	81r784...	宏言	dd42bb79...	9deeba48e12da...	a023746b69...	1	2004-08-10	M	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
24	24	18bw6v...	亚亚	a6631791...	8adf2f4ed341f...	57acf28ad85...	1	1983-04-10		2020-...	<null>	2020-06...	2020-06...	2020-06-13
25	25	ajt5c9...	阿波	31ddece...	9ec77e245f1cd...	97f7a2042b...	1	1997-02-10	M	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
26	26	uqc008...	叶叶	57df867f...	05cb9f898008f...	3eb60fc1fe...	2	1980-04-10	F	2020-...	2020-...	2020-06...	2020-06...	2020-06-13
27	27	y6fd18x...	朋斌	fd3907a4...	563b4bafcd471...	2a222c9596...	2	2001-01-10		2020-...	<null>	2020-06...	2020-06...	2020-06-13

下图是一些正在进行的 dim 层 Hive on Spark 数据装载任务



All Applications

Logged in as: hadoop

Cluster Metrics																				
Nodes	Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Vcores Used	Vcores Total	Vcores Reserved									
<small>Nodes</small>																				
New	3	0	1	2	3	6 GB	24 GB	0 B	3	24	0									
Now Saving																				
Submitted																				
Active																				
Running																				
Finished																				
Killed																				
Scheduler																				
Cluster Nodes Metrics																				
Active Nodes	Decommissioning Nodes					Decommissioned Nodes					Lost Nodes	Unhealthy Nodes								
3	0					0					0	0								
Scheduler Metrics																				
Scheduler Type	Scheduling Resource Type					Minimum Allocation					Maximum Allocation			Maximum Cluster Application Priority						
Capacity Scheduler	[memory-mb (unit=M), vcores]					<memory:1024, vCores:1>					<memory:8192, vCores:4>			0						
Show 20 entries	Search																			
ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Reserved CPU Vcores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1687160726624_0003	hadoop	Hive on Spark	SPARK	default	0	Mon Jun 19 21:35:29 2023	Mon Jun 19 21:35:29 2023	Mon Jun 19 21:36:47 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	0.0	0.0	History	0		
application_1687160726624_0002	hadoop	Hive on Spark	SPARK	default	0	Mon Jun 19 21:31:33 2023	Mon Jun 19 21:31:33 2023	Mon Jun 19 21:32:43 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	0.0	0.0	History	0		
application_1687160726624_0001	hadoop	Hive on Spark	SPARK	default	0	Mon Jun 19 20:36:43 2023	Mon Jun 19 20:36:43 2023	Mon Jun 19 20:36:45 2023	RUNNING	UNDEFINED	3	3	6144	0	0	25.0	25.0	ApplicationMaster	0	

Showing 1 to 3 of 3 entries

First Previous 1 Next Last

下图是一些 shell 端的显示，可以看到 Spark 任务的完成进度

STAGES		ATTEMPT	STATUS		TOTAL	COMPLETED	RUNNING	PENDING	FAILED
Stage-2	0	0	FINISHED		1	1	0		
Stage-3	0	0	FINISHED		2	2	0		
Stage-2	0	0	FINISHED		1	1	0		
Stage-3	0	0	FINISHED		2	2	0		
Stage-2	0	0	FINISHED		1	1	0		
Stage-3	0	0	FINISHED		2	2	0		
Stage-2	0	0	FINISHED		1	1	0		
Stage-3	0	0	FINISHED		2	2	0		
Stage-2	0	0	FINISHED		1	1	0		
Stage-3	0	0	FINISHED		2	2	0		
Stage-4	0	0	FINISHED		2	2	0		
Stage-5	0	0	FINISHED		1	1	0		
STAGES: 04/04 [=====>>] 100% ELAPSED TIME: 7.25 s									
Spark job[1] finished successfully in 7.25 seconds(s) WARNING: Spark Job[1] Spent 12% (610 ms / 1730 ms) of task time in GC Loading data to table gmall.dim_user_zip partition (dt=null) 47390 rows inserted 19:51:40.000 07/07/2020 19:51:40.000 main WARN org.apache.hadoop.hive.metastore.ObjectStore - datanucleus.autoStartMechanismMode is set to									

Spark job[8] finished successfully in 2.03 second(s)									
Launching Job 2 out of 2									
In order to change the average load for a reducer (in bytes):									
set hive.exec.reducers.bytes.per.reducer=<number>									
In order to limit the maximum number of reducers:									
set hive.exec.reducers.max=<number>									
In order to set a constant number of reducers:									
set mapreduce.job.reduces=<number>									
Running with YARN Application = application_1687160726624_0003									
Kill Command = /usr/local/hadoop/bin/yarn application -kill application_1687160726624_0003									
Hive on Spark Session Web UI URL: http://Master:44683									
Query Hive on Spark job[9] stages: [17, 18]									
Spark job[9] status = RUNNING									
STAGES		ATTEMPT	STATUS		TOTAL	COMPLETED	RUNNING	PENDING	FAILED
Stage-17	1	0	PENDING		1	0	0		
STAGES		ATTEMPT	STATUS		TOTAL	COMPLETED	RUNNING	PENDING	FAILED
Stage-17	0	0	FINISHED		1	1	0		
STAGES		ATTEMPT	STATUS		TOTAL	COMPLETED	RUNNING	PENDING	FAILED
Stage-17	0	0	FINISHED		2	2	0		
STAGES: 02/02 [=====>>] 100% ELAPSED TIME: 4.10 s									
Spark job[9] finished successfully in 4.10 seconds(s) Loading data to table gmall.dim_activity full partition (dt=2020-06-14)									

下图是系统内 HDFS 的最终状态

/

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	drwxr-xr-x	hadoop	supergroup	0 B	Jun 20 16:31	0	0 B	Hbase
□	drwxr-xr-x	hadoop	supergroup	0 B	Jun 19 14:40	0	0 B	源数据
□	drwxr-xr-x	hadoop	supergroup	0 B	Jun 19 21:43	0	0 B	spark-history
□	drwxr-xr-x	hadoop	supergroup	0 B	Jun 19 14:22	0	0 B	spark-jars
□	drwxrwxrwx	hadoop	supergroup	0 B	Jun 19 09:18	0	0 B	Hive相关
□	drwxr-xr-x	hadoop	supergroup	0 B	Jun 18 15:58	0	0 B	tmp
□	drwxr-xr-x	hadoop	supergroup	0 B	Jun 19 16:47	0	0 B	数据仓库

Showing 1 to 7 of 7 entries

Previous 1 Next

可以看到数据仓库 ODS 和 DIM 层已建立

Browse Directory

/warehouse/gmail

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	drwxr-xr-x	hadoop	supergroup	0 B	Jun 19 20:35	0	0 B	dim
□	drwxr-xr-x	hadoop	supergroup	0 B	Jun 19 17:07	0	0 B	ods

Showing 1 to 2 of 2 entries

Previous 1 Next