

PB级企业电商离线数仓项目实战【下】 (讲师 回灯)

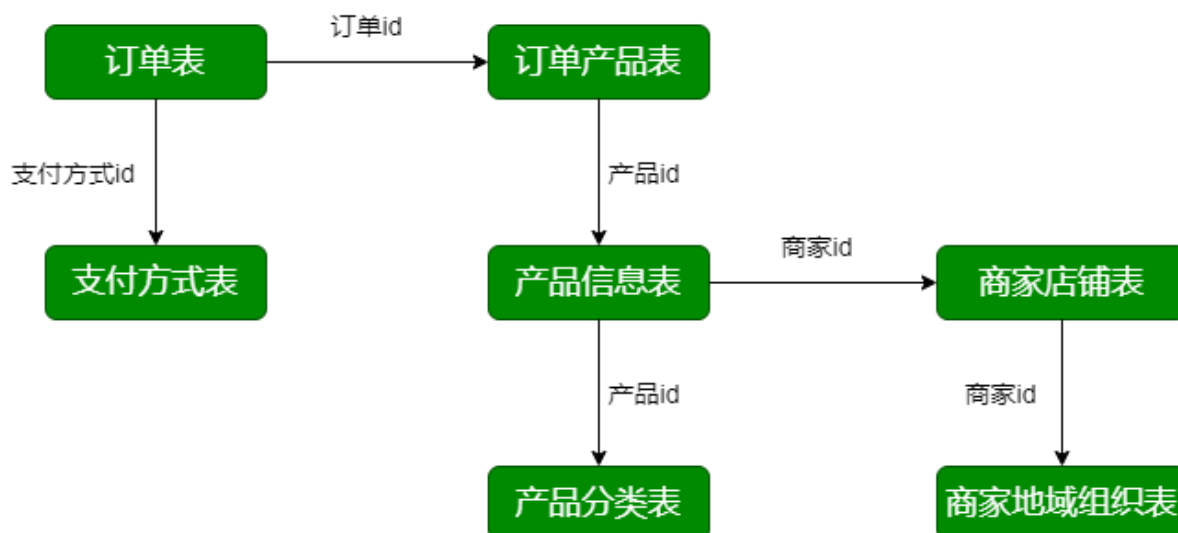
第一部分 电商分析之--核心交易

第1节 业务需求

本主题是电商系统业务中最关键的业务，电商的运营活动都是围绕这个主题展开。

选取的指标包括：订单数、商品数、支付金额。对这些指标按销售区域、商品类型进行分析。

第2节 业务数据库表结构



业务数据库：数据源

- 交易订单表 (trade_orders)
- 订单产品表 (order_product)
- 产品信息表 (product_info)
- 产品分类表 (product_category)
- 商家店铺表 (shops)
- 商家地域组织表 (shop_admin_org)
- 支付方式表 (payments)

交易订单表

```
CREATE TABLE `lagou_trade_orders` (  
  `orderId` bigint(11) NOT NULL AUTO_INCREMENT COMMENT '订单id',  
  `orderNo` varchar(20) NOT NULL COMMENT '订单编号',  
  `userId` bigint(11) NOT NULL COMMENT '用户id',
```

```

`status` tinyint(4) NOT NULL DEFAULT '-2' COMMENT '订单状态 -3:用户拒收
-2:未付款的订单 -1: 用户取消 0:待发货 1:配送中 2:用户确认收货',
`productMoney` decimal(11,2) NOT NULL COMMENT '商品金额',
`totalMoney` decimal(11,2) NOT NULL COMMENT '订单金额（包括运费）',
`payMethod` tinyint(4) NOT NULL DEFAULT '0' COMMENT '支付方式,0:未知;1:支
付宝, 2: 微信;3、现金; 4、其他',
`isPay` tinyint(4) NOT NULL DEFAULT '0' COMMENT '是否支付 0:未支付 1:已支
付',
`areaId` int(11) NOT NULL COMMENT '区域最低一级',
`tradeSrc` tinyint(4) NOT NULL DEFAULT '0' COMMENT '订单来源 0:商城 1:微
信 2:手机版 3:安卓App 4:苹果App',
`tradeType` int(11) DEFAULT '0' COMMENT '订单类型',
`isRefund` tinyint(4) NOT NULL DEFAULT '0' COMMENT '是否退款 0:否 1:
是',
`dataFlag` tinyint(4) NOT NULL DEFAULT '1' COMMENT '订单有效标志 -1:
删除 1:有效',
`createTime` varchar(25) NOT NULL COMMENT '下单时间',
`payTime` varchar(25) DEFAULT NULL COMMENT '支付时间',
`modifiedTime` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00' COMMENT
'订单更新时间',
PRIMARY KEY (`orderId`)
) ENGINE=InnoDB AUTO_INCREMENT=355 DEFAULT CHARSET=utf8;

```

备注:

- 记录订单的信息
- status。订单状态
- createTime、payTime、modifiedTime。创建时间、支付时间、修改时间

订单产品表

```

CREATE TABLE `lagou_order_product` (
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `orderId` bigint(11) NOT NULL COMMENT '订单id',
  `productId` bigint(11) NOT NULL COMMENT '商品id',
  `productNum` bigint(11) NOT NULL DEFAULT '0' COMMENT '商品数量',
  `productPrice` decimal(11,2) NOT NULL DEFAULT '0.00' COMMENT '商品价格',
  `money` decimal(11,2) DEFAULT '0.00' COMMENT '付款金额',
  `extra` text COMMENT '额外信息',
  `createTime` varchar(25) DEFAULT NULL COMMENT '创建时间',
  PRIMARY KEY (`id`),
  KEY `orderId` (`orderId`),
  KEY `goodsId` (`productId`)
) ENGINE=InnoDB AUTO_INCREMENT=1260 DEFAULT CHARSET=utf8;

```

备注:

- 记录订单中购买产品的信息，包括产品的数量、单价等

产品信息表

```
CREATE TABLE `lagou_product_info` (  
  `productId` bigint(11) NOT NULL AUTO_INCREMENT COMMENT '商品id',  
  `productName` varchar(200) NOT NULL COMMENT '商品名称',  
  `shopId` bigint(11) NOT NULL COMMENT '门店ID',  
  `price` decimal(11,2) NOT NULL DEFAULT '0.00' COMMENT '门店价',  
  `isSale` tinyint(4) NOT NULL DEFAULT '1' COMMENT '是否上架 0:不上架 1:  
上架',  
  `status` tinyint(4) NOT NULL DEFAULT '0' COMMENT '是否新品 0:否 1:是',  
  `categoryId` int(11) NOT NULL COMMENT 'goodsCatId 最后一级商品分类ID',  
  `createTime` varchar(25) NOT NULL,  
  `modifyTime` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT  
'修改时间',  
  PRIMARY KEY (`productId`),  
  KEY `shopId` (`shopId`) USING BTREE,  
  KEY `goodsStatus` (`isSale`)  
) ENGINE=InnoDB AUTO_INCREMENT=115909 DEFAULT CHARSET=utf8;
```

备注:

- 记录产品的详细信息，对应商家ID、商品属性（是否新品、是否上架）
- createTime、modifyTime。创建时间和修改时间

产品分类表

```
CREATE TABLE `lagou_product_category` (  
  `catId` int(11) NOT NULL AUTO_INCREMENT COMMENT '品类ID',  
  `parentId` int(11) NOT NULL COMMENT '父ID',  
  `catName` varchar(20) NOT NULL COMMENT '分类名称',  
  `isShow` tinyint(4) NOT NULL DEFAULT '1' COMMENT '是否显示 0:隐藏 1:显  
示',  
  `sortNum` int(11) NOT NULL DEFAULT '0' COMMENT '排序号',  
  `isDel` tinyint(4) NOT NULL DEFAULT '1' COMMENT '删除标志 1:有效 -1:删  
除',  
  `createTime` varchar(25) NOT NULL COMMENT '建立时间',  
  `level` tinyint(4) DEFAULT '0' COMMENT '分类级别，共3级',  
  PRIMARY KEY (`catId`),  
  KEY `parentId` (`parentId`,`isShow`,`isDel`)  
) ENGINE=InnoDB AUTO_INCREMENT=10442 DEFAULT CHARSET=utf8;
```

备注: 产品分类表，共分3个级别

```

-- 第一级产品目录
select catName, catid from lagou_product_category where level = 1;

-- 查看电脑、办公的子类（查看二级目录）
select catName, catid from lagou_product_category where level = 2 and
parentId = 32;

-- 查看电脑整机的子类（查看三级目录）
select catName, catid from lagou_product_category where level = 3 and
parentId = 10250;

```

商家店铺表

```

CREATE TABLE `lagou_shops` (
  `shopId` int(11) NOT NULL AUTO_INCREMENT COMMENT '店铺ID，自增',
  `userId` int(11) NOT NULL COMMENT '店铺联系人ID',
  `areaId` int(11) DEFAULT '0',
  `shopName` varchar(100) DEFAULT '' COMMENT '店铺名称',
  `shopLevel` tinyint(4) NOT NULL DEFAULT '1' COMMENT '店铺等级',
  `status` tinyint(4) NOT NULL DEFAULT '1' COMMENT '店铺状态',
  `createTime` date DEFAULT NULL,
  `modifyTime` datetime DEFAULT NULL COMMENT '修改时间',
  PRIMARY KEY (`shopId`),
  KEY `shopStatus` (`status`)
) ENGINE=InnoDB AUTO_INCREMENT=105317 DEFAULT CHARSET=utf8;

```

备注：记录店铺的详细信息

商家地域组织表

```

CREATE TABLE `lagou_shop_admin_org` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '组织ID',
  `parentId` int(11) NOT NULL COMMENT '父ID',
  `orgName` varchar(100) NOT NULL COMMENT '组织名称',
  `orgLevel` tinyint(4) NOT NULL DEFAULT '1' COMMENT '组织级别1:总部及大区级
部门;2:总部下属的各个部门及基部门;3:具体工作部门',
  `isDelete` tinyint(4) NOT NULL DEFAULT '0' COMMENT '删除标志,1:删除;0:有
效',
  `createTime` varchar(25) DEFAULT NULL COMMENT '创建时间',
  `updateTime` varchar(25) DEFAULT NULL COMMENT '最后修改时间',
  `isShow` tinyint(4) NOT NULL DEFAULT '1' COMMENT '是否显示,0:是 1:否',
  `orgType` tinyint(4) NOT NULL DEFAULT '1' COMMENT '组织类型,0:总裁办;1:研
发;2:销售;3:运营;4:产品',
  PRIMARY KEY (`id`),
  KEY `parentId` (`parentId`)
) ENGINE=InnoDB AUTO_INCREMENT=100332 DEFAULT CHARSET=utf8;

```

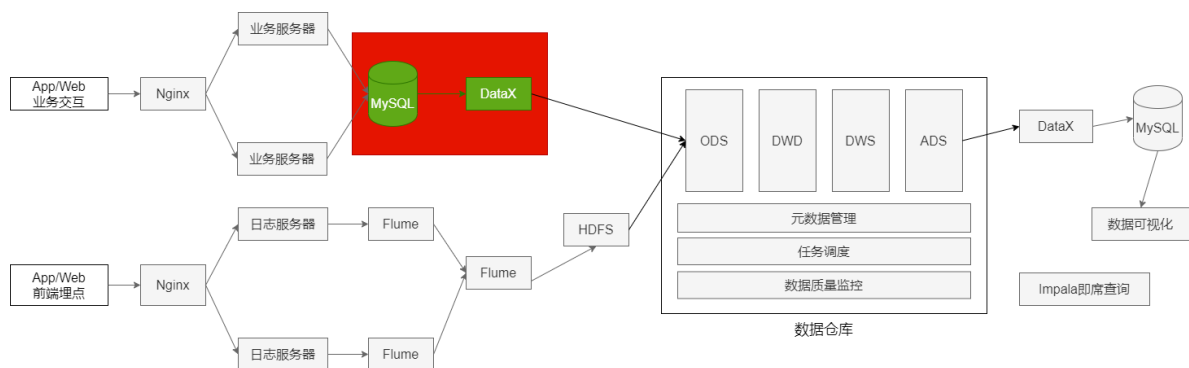
备注：记录店铺所属区域

支付方式表

```
CREATE TABLE `lagou_payments` (  
  `id` int(11) NOT NULL,  
  `payMethod` varchar(20) DEFAULT NULL,  
  `payName` varchar(255) DEFAULT NULL,  
  `description` varchar(255) DEFAULT NULL,  
  `payOrder` int(11) DEFAULT '0',  
  `online` tinyint(4) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `payCode` (`payMethod`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

备注：记录支付方式

第3节 数据导入

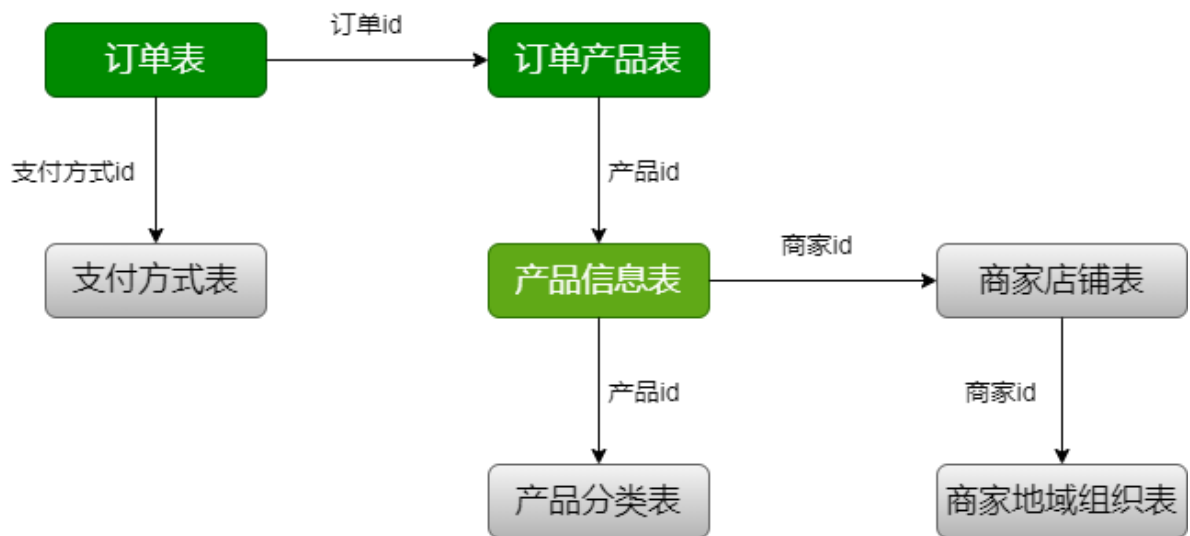


已经确定的事情：DataX、导出7张表的数据。

MySQL 导出：全量导出、增量导出（导出前一天的数据）。

业务数据保存在MySQL中，每日凌晨导入上一天的表数据。

- 表数据量少，采用全量方式导出MySQL
- 表数据量大，而且根据字段能区分出每天新增数据，采用增量方式导出MySQL



3张增量表:

- 订单表 lagou_trade_orders
- 订单产品表 lagou_order_produce
- 产品信息表 lagou_product_info

4张全量表:

- 产品分类表 lagou_product_category
- 商家店铺表 lagou_shops
- 商家地域组织表 lagou_shop_admin_org
- 支付方式表 lagou_payment

3.1、全量数据导入

MySQL => HDFS => Hive

每日加载全量数据，形成新的分区；(ODS如何建表有指导左右)

MySQLReader ==> HdfsWriter

ebiz.lagou_product_category ==> ods.ods_trade_product_category

1、产品分类表

/data/lagoudw/json/product_category.json

```

{
  "job": {
    "setting": {
      "speed": {
        "channel": 1
      }
    },
    "content": [{
      "reader": {

```

```

    "name": "mysqlreader",
    "parameter": {
      "username": "teacher",
      "password": "teacher123",
      "column": [
        "catId", "parentId", "catName",
        "isShow",
        "sortNum",
        "isDel",
        "createTime",
        "level"
      ],
      "connection": [{
        "table": [
          "lagou_product_category"
        ],
        "jdbcUrl": [
          "jdbc:mysql://hadoop1:3306/ebiz"
        ]
      }]
    },
  },
  "writer": {
    "name": "hdfswriter",
    "parameter": {
      "defaultFS": "hdfs://hadoop1:9000",
      "fileType": "text",
      "path":
"/user/data/trade.db/product_category/dt=$do_date",
      "fileName": "product_category_${do_date}",
      "column": [
        {
          "name": "catId",
          "type": "INT"
        },
        {
          "name": "parentId",
          "type": "INT"
        },
        {
          "name": "catName",
          "type": "STRING"
        },
        {
          "name": "isShow",
          "type": "TINYINT"
        },
        {
          "name": "sortNum",
          "type": "INT"
        }
      ]
    }
  }
}

```



```

        "speed": {
            "channel": 1
        },
        "errorLimit": {
            "record": 0
        }
    },
    "content": [{
        "reader": {
            "name": "mysqlreader",
            "parameter": {
                "username": "teacher",
                "password": "teacher123",
                "column": [
                    "shopId", "userId", "areaId", "shopName",
                    "shopLevel", "status", "createTime", "modifyTime"
                ],
                "connection": [{
                    "table": [
                        "lagou_shops"
                    ],
                    "jdbcurl": [
                        "jdbc:mysql://hadoop1:3306/ebiz"
                    ]
                }]
            }
        },
        "writer": {
            "name": "hdfswriter",
            "parameter": {
                "defaultFS": "hdfs://hadoop1:9000",
                "fileType": "text",
                "path": "/user/data/trade.db/shops/dt=$do_date",
                "fileName": "shops_$do_date",
                "column": [{
                    "name": "shopId",
                    "type": "INT"
                },
                {
                    "name": "userId",
                    "type": "INT"
                },
                {
                    "name": "areaId",
                    "type": "INT"
                },
                {
                    "name": "shopName",
                    "type": "STRING"
                }
            ]
        }
    }]
}

```

```

        "name": "shopLevel",
        "type": "TINYINT"
    },
    {
        "name": "status",
        "type": "TINYINT"
    },
    {
        "name": "createTime",
        "type": "STRING"
    },
    {
        "name": "modifyTime",
        "type": "STRING"
    }
],
"writeMode": "append",
"fieldDelimiter": ",",
    }
}
}
}
}

```

```
do_date='2020-07-02'
```

```
# 创建目录
```

```
hdfs dfs -mkdir -p /user/data/trade.db/shops/dt=$do_date
```

```
# 数据迁移
```

```
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/shops.json
```

```
# 加载数据
```

```
hive -e "alter table ods.ods_trade_shops add partition(dt='$do_date')"
```

3、商家地域组织表

lagou_shop_admin_org ==> ods.ods_trade_shop_admin_org

/data/lagoudw/json/shop_org.json

```

{
  "job": {
    "setting": {
      "speed": {
        "channel": 1
      },

```

```

        "errorLimit": {
            "record": 0
        }
    },
    "content": [{
        "reader": {
            "name": "mysqlreader",
            "parameter": {
                "username": "teacher",
                "password": "teacher123",
                "column": [
                    "id", "parentId", "orgName", "orgLevel",
                    "isDelete", "createTime", "updateTime", "isShow", "orgType"
                ],
                "connection": [{
                    "table": [
                        "lagou_shop_admin_org"
                    ],
                    "jdbcurl": [
                        "jdbc:mysql://hadoop1:3306/ebiz"
                    ]
                }
            ]
        },
        "writer": {
            "name": "hdfswriter",
            "parameter": {
                "defaultFS": "hdfs://hadoop1:9000",
                "fileType": "text",
                "path": "/user/data/trade.db/shop_org/dt=$do_date",
                "fileName": "shop_admin_org_$do_date.dat",
                "column": [{
                    "name": "id",
                    "type": "INT"
                },
                {
                    "name": "parentId",
                    "type": "INT"
                },
                {
                    "name": "orgName",
                    "type": "STRING"
                },
                {
                    "name": "orgLevel",
                    "type": "TINYINT"
                },
                {
                    "name": "isDelete",
                    "type": "TINYINT"
                }
            ]
        }
    }
    ],
    "writer": {
        "name": "hdfswriter",
        "parameter": {
            "defaultFS": "hdfs://hadoop1:9000",
            "fileType": "text",
            "path": "/user/data/trade.db/shop_org/dt=$do_date",
            "fileName": "shop_admin_org_$do_date.dat",
            "column": [{
                "name": "id",
                "type": "INT"
            },
            {
                "name": "parentId",
                "type": "INT"
            },
            {
                "name": "orgName",
                "type": "STRING"
            },
            {
                "name": "orgLevel",
                "type": "TINYINT"
            },
            {
                "name": "isDelete",
                "type": "TINYINT"
            }
        ]
    }
}

```



```

        "channel": 1
    },
    "errorLimit": {
        "record": 0
    }
},
"content": [{
    "reader": {
        "name": "mysqlreader",
        "parameter": {
            "username": "teacher",
            "password": "teacher123",
            "column": [
                "id", "payMethod", "payName", "description",
                "payOrder", "online"
            ],
            "connection": [{
                "table": [
                    "lagou_payments"
                ],
                "jdbcUrl": [
                    "jdbc:mysql://hadoop1:3306/ebiz"
                ]
            }]
        }
    },
    "writer": {
        "name": "hdfswriter",
        "parameter": {
            "defaultFS": "hdfs://hadoop1:9000",
            "fileType": "text",
            "path": "/user/data/trade.db/payments/dt=$do_date",
            "fileName": "payments_${do_date}.dat",
            "column": [{
                "name": "id",
                "type": "INT"
            },
            {
                "name": "payMethod",
                "type": "STRING"
            },
            {
                "name": "payName",
                "type": "STRING"
            },
            {
                "name": "description",
                "type": "STRING"
            },
            {
                "name": "payOrder",

```

```

        "type": "INT"
    },
    {
        "name": "online",
        "type": "TINYINT"
    }
],
"writeMode": "append",
"fieldDelimiter": ",",
}
}
}]
}
}

```

```

do_date='2020-07-01'

# 创建目录
hdfs dfs -mkdir -p /user/data/trade.db/payments/dt=$do_date

# 数据迁移
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/payments.json

# 加载数据
hive -e "alter table ods.ods_trade_payments add partition(dt='$do_date')"

```

3.2、增量数据导入

3张增量表：

- 订单表 lagou_trade_orders
- 订单产品表 lagou_order_produce
- 产品信息表 lagou_product_info

初始数据装载（执行一次）；可以将前面的全量加载作为初次装载

每日加载增量数据（每日数据形成分区）；

1、订单表

lagou_trade_orders ==> ods.ods_trade_orders
/data/lagoudw/json/orders.json

备注：条件的选择，选择时间字段 modifiedTime

```

{
  "job": {
    "setting": {
      "speed": {
        "channel": 1
      },
      "errorLimit": {
        "record": 0
      }
    },
    "content": [{
      "reader": {
        "name": "mysqlreader",
        "parameter": {
          "username": "teacher",
          "password": "teacher123",
          "connection": [{
            "querySql": [
              "select orderId, orderNo, userId, status,
productMoney, totalMoney, payMethod, isPay, areaId, tradeSrc, tradeType,
isRefund, dataFlag, createTime, payTime, modifiedTime from
lagou_trade_orders where date_format(modifiedTime, '%Y-%m-%d')='$do_date'"
            ],
            "jdbcurl": [
              "jdbc:mysql://hadoop1:3306/ebiz"
            ]
          }]
        }
      },
      "writer": {
        "name": "hdfswriter",
        "parameter": {
          "defaultFS": "hdfs://hadoop1:9000",
          "fileType": "text",
          "path": "/user/data/trade.db/orders/dt=$do_date",
          "fileName": "orders_$do_date",
          "column": [{
            "name": "orderId",
            "type": "INT"
          },
          {
            "name": "orderNo",
            "type": "STRING"
          },
          {
            "name": "userId",
            "type": "BIGINT"
          },
          {
            "name": "status",

```

```
        "type": "TINYINT"
    },
    {
        "name": "productMoney",
        "type": "Float"
    },
    {
        "name": "totalMoney",
        "type": "Float"
    },
    {
        "name": "payMethod",
        "type": "TINYINT"
    },
    {
        "name": "isPay",
        "type": "TINYINT"
    },
    {
        "name": "areaId",
        "type": "INT"
    },
    {
        "name": "tradeSrc",
        "type": "TINYINT"
    },
    {
        "name": "tradeType",
        "type": "INT"
    },
    {
        "name": "isRefund",
        "type": "TINYINT"
    },
    {
        "name": "dataFlag",
        "type": "TINYINT"
    },
    {
        "name": "createTime",
        "type": "STRING"
    },
    {
        "name": "payTime",
        "type": "STRING"
    },
    {
        "name": "modifiedTime",
        "type": "STRING"
    }
},
"writeMode": "append",
```



```

        "fieldDelimiter": ",",
    }
}
}]
}
}

```

-- MySQL 中的时间日期转换

```

select date_format(createTime, '%Y-%m-%d'), count(*)
  from lagou_trade_orders
 group by date_format(createTime, '%Y-%m-%d');

```

do_date='2020-07-12'

创建目录

```
hdfs dfs -mkdir -p /user/data/trade.db/orders/dt=$do_date
```

数据迁移

```
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/orders.json
```

加载数据

```
hive -e "alter table ods.ods_trade_orders add partition(dt='$do_date')"
```

2、订单明细表

lagou_order_product ==> ods.ods_trade_order_product

/data/lagoudw/json/order_product.json

```

{
  "job": {
    "setting": {
      "speed": {
        "channel": 1
      },
      "errorLimit": {
        "record": 0
      }
    },
    "content": [{
      "reader": {
        "name": "mysqlreader",
        "parameter": {
          "username": "teacher",

```

```

        "password": "teacher123",
        "connection": [{
            "querySql": [
                "select id, orderId, productId, productNum,
productPrice, money, extra, createTime from lagou_order_product where
date_format(createTime, '%Y-%m-%d') = '$do_date' "
            ],
            "jdbcurl": [
                "jdbc:mysql://hadoop1:3306/ebiz"
            ]
        }]
    },
    "writer": {
        "name": "hdfswriter",
        "parameter": {
            "defaultFS": "hdfs://hadoop1:9000",
            "fileType": "text",
            "path":
"/user/data/trade.db/order_product/dt=$do_date",
            "fileName": "order_product_$do_date.dat",
            "column": [{
                "name": "id",
                "type": "INT"
            },
            {
                "name": "orderId",
                "type": "INT"
            },
            {
                "name": "productId",
                "type": "INT"
            },
            {
                "name": "productNum",
                "type": "INT"
            },
            {
                "name": "productPrice",
                "type": "Float"
            },
            {
                "name": "money",
                "type": "Float"
            },
            {
                "name": "extra",
                "type": "STRING"
            },
            {
                "name": "createTime",

```

```

        "type": "STRING"
    }
    ],
    "writeMode": "append",
    "fieldDelimiter": ",",
}
}
}]
}
}

```

```
do_date='2020-07-12'
```

```
# 创建目录
```

```
hdfs dfs -mkdir -p /user/data/trade.db/order_product/dt=$do_date
```

```
# 数据迁移
```

```
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/order_product.json
```

```
# 加载数据
```

```
hive -e "alter table ods.ods_trade_order_product add
partition(dt='$do_date')"
```

3、产品明细表

lagou_product_info ==> ods.ods_trade_product_info

/data/lagoudw/json/product_info.json

```

{
  "job": {
    "setting": {
      "speed": {
        "channel": 1
      },
      "errorLimit": {
        "record": 0
      }
    },
    "content": [{
      "reader": {
        "name": "mysqlreader",
        "parameter": {
          "username": "teacher",
          "password": "teacher123",
          "connection": [{
            "querySql": [

```

```

        "select productid, productname, shopid, price,
issale, status, categoryid, createtime, modifytime from lagou_product_info
where date_format(modifyTime, '%Y-%m-%d') = '$do_date' "
    ],
    "jdbcurl": [
        "jdbc:mysql://hadoop1:3306/ebiz"
    ]
}
}],
    "writer": {
        "name": "hdfswriter",
        "parameter": {
            "defaultFS": "hdfs://hadoop1:9000",
            "fileType": "text",
            "path":
"/user/data/trade.db/product_info/dt=$do_date",
            "fileName": "product_info_$do_date.dat",
            "column": [{
                "name": "productid",
                "type": "BIGINT"
            },
            {
                "name": "productname",
                "type": "STRING"
            },
            {
                "name": "shopid",
                "type": "STRING"
            },
            {
                "name": "price",
                "type": "FLOAT"
            },
            {
                "name": "issale",
                "type": "TINYINT"
            },
            {
                "name": "status",
                "type": "TINYINT"
            },
            {
                "name": "categoryid",
                "type": "STRING"
            },
            {
                "name": "createTime",
                "type": "STRING"
            }
        ]
    }
}

```

```

        "name": "modifytime",
        "type": "STRING"
    }
],
"writeMode": "append",
"fieldDelimiter": ",",
}
}
}]
}
}

```

```
do_date='2020-07-12'
```

```
# 创建目录
```

```
hdfs dfs -mkdir -p /user/data/trade.db/product_info/dt=$do_date
```

```
# 数据迁移
```

```
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/product_info.json
```

```
# 加载数据
```

```
hive -e "alter table ods.ods_trade_product_info add
partition(dt='$do_date')"
```

第4节 ODS层建表与数据加载

ODS建表：

- ODS层的表结构与源数据基本类似（列名及数据类型）；
- ODS层的表名遵循统一的规范；

4.1 ODS层建表

所有的表都是分区表；字段之间的分隔符为 `,`；为表的数据数据文件指定了位置；

```

DROP TABLE IF EXISTS `ods.ods_trade_orders`;
CREATE EXTERNAL TABLE `ods.ods_trade_orders`(
  `orderid` int,
  `orderno` string,
  `userid` bigint,
  `status` tinyint,
  `productmoney` decimal(10, 0),
  `totalmoney` decimal(10, 0),
  `paymethod` tinyint,
  `ispay` tinyint,

```

```

`areaid` int,
`tradesrc` tinyint,
`tradetype` int,
`isrefund` tinyint,
`dataflag` tinyint,
`createtime` string,
`paytime` string,
`modifiedtime` string)
COMMENT '订单表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/orders/';

DROP TABLE IF EXISTS `ods.ods_trade_order_product`;
CREATE EXTERNAL TABLE `ods.ods_trade_order_product`(
  `id` string,
  `orderid` decimal(10,2),
  `productid` string,
  `productnum` string,
  `productprice` string,
  `money` string,
  `extra` string,
  `createtime` string)
COMMENT '订单明细表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/order_product/';

DROP TABLE IF EXISTS `ods.ods_trade_product_info`;
CREATE EXTERNAL TABLE `ods.ods_trade_product_info`(
  `productid` bigint,
  `productname` string,
  `shopid` string,
  `price` decimal(10,0),
  `issale` tinyint,
  `status` tinyint,
  `categoryid` string,
  `createtime` string,
  `modifytime` string)
COMMENT '产品信息表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/product_info/';

DROP TABLE IF EXISTS `ods.ods_trade_product_category`;
CREATE EXTERNAL TABLE `ods.ods_trade_product_category`(
  `catid` int,
  `parentid` int,
  `catname` string,
  `isshow` tinyint,
  `sortnum` int,
  `isdel` tinyint,

```

```
`createtime` string,
`level` tinyint)
COMMENT '产品分类表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/product_category';

DROP TABLE IF EXISTS `ods.ods_trade_shops`;
CREATE EXTERNAL TABLE `ods.ods_trade_shops`(
  `shopid` int,
  `userid` int,
  `areaaid` int,
  `shopname` string,
  `shoplevel` tinyint,
  `status` tinyint,
  `createtime` string,
  `modifytime` string)
COMMENT '商家店铺表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/shops';

DROP TABLE IF EXISTS `ods.ods_trade_shop_admin_org`;
CREATE EXTERNAL TABLE `ods.ods_trade_shop_admin_org`(
  `id` int,
  `parentid` int,
  `orgname` string,
  `orglevel` tinyint,
  `isdelete` tinyint,
  `createtime` string,
  `updatetime` string,
  `isshow` tinyint,
  `orgType` tinyint)
COMMENT '商家地域组织表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/shop_org/';

DROP TABLE IF EXISTS `ods.ods_trade_payments`;
CREATE EXTERNAL TABLE `ods.ods_trade_payments`(
  `id` string,
  `paymethod` string,
  `payname` string,
  `description` string,
  `payorder` int,
  `online` tinyint)
COMMENT '支付方式表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/payments/';
```

4.2 ODS层数据加载

DataX仅仅是将数据导入到了 HDFS，数据并没有与Hive表建立关联。

脚本的任务：数据迁移、数据加载到ODS层；

对于增量加载数据而言：初始数据加载；该任务仅执行一次，不在脚本中。

/data/lagoudw/script/trade/ods_load_trade.sh

```
#!/bin/bash

source /etc/profile

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

# 创建目录
hdfs dfs -mkdir -p /user/data/trade.db/product_category/dt=$do_date
hdfs dfs -mkdir -p /user/data/trade.db/shops/dt=$do_date
hdfs dfs -mkdir -p /user/data/trade.db/shop_org/dt=$do_date
hdfs dfs -mkdir -p /user/data/trade.db/payments/dt=$do_date
hdfs dfs -mkdir -p /user/data/trade.db/orders/dt=$do_date
hdfs dfs -mkdir -p /user/data/trade.db/order_product/dt=$do_date
hdfs dfs -mkdir -p /user/data/trade.db/product_info/dt=$do_date

# 数据迁移
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/product_category.json
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/shops.json
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/shop_org.json
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/payments.json
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/orders.json
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/order_product.json
python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
/data/lagoudw/json/product_info.json

# 加载 ODS 层数据
sql="
alter table ods.ods_trade_orders add partition(dt='$do_date');
alter table ods.ods_trade_order_product add partition(dt='$do_date');
alter table ods.ods_trade_product_info add partition(dt='$do_date');
alter table ods.ods_trade_product_category add partition(dt='$do_date');
```



```
alter table ods.ods_trade_shops add partition(dt='$do_date');
alter table ods.ods_trade_shop_admin_org add partition(dt='$do_date');
alter table ods.ods_trade_payments add partition(dt='$do_date');
"

hive -e "$sql"
```

特点：工作量大，繁琐，容易出错；与数据采集工作在一起；

第5节 缓慢变化维与周期性事实表

5.1、缓慢变化维

缓慢变化维（SCD；Slowly Changing Dimensions）。在现实世界中，维度的属性随着时间的流失发生缓慢的变化（缓慢是相对事实表而言，事实表数据变化的速度比维度表快）。

处理维度表的历史变化信息的问题称为处理缓慢变化维的问题，简称SCD问题。处理缓慢变化维的方法有以下几种常见方式：

- 保留原值
- 直接覆盖
- 增加新属性列
- 快照表
- 拉链表

1、保留原始值

维度属性值不做更改，保留原始值。

如商品上架售卖时间：一个商品上架售卖后由于其他原因下架，后来又再次上架，此种情况产生了多个商品上架售卖时间。如果业务重点关注的是商品首次上架售卖时间，则采用该方式。

2、直接覆盖

修改维度属性为最新值，直接覆盖，不保留历史信息。

如商品属于哪个品类：当商品品类发生变化时，直接重写为新品类。

3、增加新属性列

在维度表中增加新的一列，原先属性列存放上一版本的属性值，当前属性列存放当前版本的属性值，还可以增加一列记录变化的时间。

缺点：只能记录最后一次变化的信息。

维度属性值变化前：

商品ID	品类
SPU001	科技

维度属性值变化后：

商品ID	当前品类	原品类	变更时间
SPU001	教育	科技	2020-01-01

4、快照表

每天保留一份全量数据。

简单、高效。缺点是信息重复，浪费磁盘空间。

适用范围：维表不能太大

使用场景多，范围广；一般而言维表都不大。

5、拉链表

拉链表适合于：**表的数据量大**，而且数据会发生新增和变化，但是大部分是不变的（数据发生变化的百分比不大），且是缓慢变化的（如电商中用户信息表中的某些用户基本属性不可能每天都变化）。主要目的是节省存储空间。

适用场景：

- 表的数据量大
- 表中部分字段会被更新
- 表中记录变量的比例不高
- 需要保留历史信息

5.2、维表拉链表应用案例

userid	mobile	reg_date	start_date
001	13551111111	2020-03-01	2020-06-20
002	13561111111	2020-04-01	2020-06-20
003	13571111111	2020-05-01	2020-06-20
004	13581111111	2020-06-01	2020-06-20
002	13562222222	2020-04-01	2020-06-21
004	13582222222	2020-06-01	2020-06-21
005	13552222222	2020-06-21	2020-06-21
004	13333333333	2020-06-01	2020-06-22
005	13533333333	2020-06-21	2020-06-22
006	13733333333	2020-06-22	2020-06-22
001	13554444444	2020-03-01	2020-06-23
003	13574444444	2020-05-01	2020-06-23
005	13555544444	2020-06-21	2020-06-23
007	18600744444	2020-06-23	2020-06-23
008	18600844444	2020-06-23	2020-06-23

1、创建表加载数据（准备工作）

```
-- 用户信息
DROP TABLE IF EXISTS test.userinfo;
CREATE TABLE test.userinfo(
    userid STRING COMMENT '用户编号',
    mobile STRING COMMENT '手机号码',
    regdate STRING COMMENT '注册日期')
COMMENT '用户信息'
PARTITIONED BY (dt string)
row format delimited fields terminated by ',';

-- 拉链表（存放用户历史信息）
-- 拉链表不是分区表；多了两个字段start_date、end_date
DROP TABLE IF EXISTS test.userhis;
CREATE TABLE test.userhis(
    userid STRING COMMENT '用户编号',
    mobile STRING COMMENT '手机号码',
    regdate STRING COMMENT '注册日期',
    start_date STRING,
    end_date STRING)
COMMENT '用户信息拉链表'
row format delimited fields terminated by ',';

-- 数据(/data/lagoudw/data/userinfo.dat)
001,13551111111,2020-03-01,2020-06-20
002,13561111111,2020-04-01,2020-06-20
003,13571111111,2020-05-01,2020-06-20
004,13581111111,2020-06-01,2020-06-20
```

```

002,1356222222,2020-04-01,2020-06-21
004,1358222222,2020-06-01,2020-06-21
005,1355222222,2020-06-21,2020-06-21

004,1333333333,2020-06-01,2020-06-22
005,1353333333,2020-06-21,2020-06-22
006,1373333333,2020-06-22,2020-06-22

001,1355444444,2020-03-01,2020-06-23
003,1357444444,2020-05-01,2020-06-23
005,1355554444,2020-06-21,2020-06-23
007,1860074444,2020-06-23,2020-06-23
008,1860084444,2020-06-23,2020-06-23

-- 静态分区数据加载（略）
/data/lagoudw/data/userinfo0620.dat
001,1355111111,2020-03-01
002,1356111111,2020-04-01
003,1357111111,2020-05-01
004,1358111111,2020-06-01
load data local inpath '/data/lagoudw/data/userinfo0620.dat' into table
test.userinfo
partition(dt='2020-06-20');

-- 动态分区数据加载：分区的值是不固定的，由输入数据确定
-- 创建中间表(非分区表)
drop table if exists test.tmp1;
create table test.tmp1 as
select * from test.userinfo;
-- tmp1 非分区表，使用系统默认的分段分割符'\001'
alter table test.tmp1 set serdeproperties('field.delim'=',');
-- 向中间表加载数据
load data local inpath '/data/lagoudw/data/userinfo.dat' into table
test.tmp1;

-- 从中间表向分区表加载数据
set hive.exec.dynamic.partition.mode=nonstrict;
insert into table test.userinfo
partition(dt)
select * from test.tmp1;

```

与动态分区相关的参数

hive.exec.dynamic.partition

- Default Value: `false` prior to Hive 0.9.0; `true` in Hive 0.9.0 and later
- Added In: Hive 0.6.0

Whether or not to allow dynamic partitions in DML/DDDL.

表示开启动态分区功能

hive.exec.dynamic.partition.mode

- Default Value: `strict`
- Added In: Hive 0.6.0

In `strict` mode, the user must specify at least one static partition in case the user accidentally overwrites all partitions. In `nonstrict` mode all partitions are allowed to be dynamic.

Set to nonstrict to support INSERT ... VALUES, UPDATE, and DELETE transactions (Hive 0.14.0 and later).

strict: 最少需要有一个是静态分区

nonstrict: 可以全部是动态分区

hive.exec.max.dynamic.partitions

- Default Value: `1000`
- Added In: Hive 0.6.0

Maximum number of dynamic partitions allowed to be created in total.

表示一个动态分区语句可以创建的最大动态分区个数，超出报错

hive.exec.max.dynamic.partitions.pernode

- Default Value: `100`
- Added In: Hive 0.6.0

Maximum number of dynamic partitions allowed to be created in each mapper/reducer node.

表示每个mapper / reducer可以允许创建的最大动态分区个数，默认是100，超出则会报错。

hive.exec.max.created.files

- Default Value: `100000`
- Added In: Hive 0.7.0

Maximum number of HDFS files created by all mappers/reducers in a MapReduce job.

表示一个MR job可以创建的最大文件个数，超出报错。

2、拉链表的实现

userinfo(分区表) => userid、mobile、regdate => 每日变更的数据（修改的+新增的） / 历史数据（第一天）

userhis (拉链表) => 多了两个字段 start_date / end_date

```
-- 步骤:
-- 1、userinfo初始化(2020-06-20)。获取历史数据
001,1355111111,2020-03-01,2020-06-20
002,1356111111,2020-04-01,2020-06-20
003,1357111111,2020-05-01,2020-06-20
004,1358111111,2020-06-01,2020-06-20

-- 2、初始化拉链表(2020-06-20)。userinfo => userhis
insert overwrite table test.userhis
select  userid, mobile, regdate, dt as start_date, '9999-12-31' as
end_date
      from test.userinfo
     where dt='2020-06-20';

-- 3、次日新增数据(2020-06-21); 获取新增数据
002,1356222222,2020-04-01,2020-06-21
004,1358222222,2020-06-01,2020-06-21
005,1355222222,2020-06-21,2020-06-21

-- 4、构建拉链表(userhis)(2020-06-21)【核心】 userinfo(2020-06-21) +
userhis => userhis
-- userinfo: 新增数据
-- userhis: 历史数据

-- 第一步: 处理新增数据【userinfo】(处理逻辑与加载历史数据类似)
select  userid, mobile, regdate, dt as start_date, '9999-12-31' as
end_date
      from test.userinfo
     where dt='2020-06-21';

-- 第二步: 处理历史数据【userhis】(历史包括两部分: 变化的、未变化的)
-- 变化的: start_date:不变; end_date: 传入日期-1
-- 未变化的: 不做处理

-- 观察数据
select A.userid, B.userid, B.mobile, B.regdate, B.start_date, B.end_date
      from (select * from test.userinfo where dt='2020-06-21') A
     right join test.userhis B
        on A.userid=B.userid;

-- 编写SQL, 处理历史数据
select B.userid,
       B.mobile,
       B.regdate,
       B.start_date,
       case when B.end_date='9999-12-31' and A.userid is not null
            then date_add('2020-06-21', -1)
            else B.end_date
       end as end_date
      from (select * from test.userinfo where dt='2020-06-21') A
```

```

right join test.userhis B
on A.userid=B.userid;

-- 最终的处理（新增+历史数据）
insert overwrite table test.userhis
select userid, mobile, regdate, dt as start_date, '9999-12-31' as
end_date
from test.userinfo
where dt='2020-06-21'

union all

select B.userid,
B.mobile,
B.regdate,
B.start_date,
case when B.end_date='9999-12-31' and A.userid is not null
then date_add('2020-06-21', -1)
else B.end_date
end as end_date
from (select * from test.userinfo where dt='2020-06-21') A
right join test.userhis B
on A.userid=B.userid;

-- 5、第三日新增数据（2020-06-22）：获取新增数据
004,1333333333,2020-06-01,2020-06-22
005,1353333333,2020-06-21,2020-06-22
006,1373333333,2020-06-22,2020-06-22

-- 6、构建拉链表（2020-06-22） userinfo(2020-06-22) + userhis => userhis

-- 7、第四日新增数据（2020-06-23）
001,1355444444,2020-03-01,2020-06-23
003,1357444444,2020-05-01,2020-06-23
005,1355554444,2020-06-21,2020-06-23
007,1860074444,2020-06-23,2020-06-23
008,1860084444,2020-06-23,2020-06-23

-- 8、构建拉链表(2020-06-23)

```

处理拉链表的脚本(测试脚本):

/data/lagoudw/data/userzipper.sh

```

#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then

```

```

do_date=$1
else
do_date=`date -d "-1 day" +%F`
fi

sql="
insert overwrite table test.userhis
select  userid, mobile, regdate, dt as start_date, '9999-12-31' as
end_date
      from test.userinfo
      where dt='$do_date'

union all

select B.userid,
      B.mobile,
      B.regdate,
      B.start_Date,
      case when B.end_date='9999-12-31' and A.userid is not null
            then date_add('$do_date', -1)
            else B.end_date
      end as end_date
from (select * from test.userinfo where dt='$do_date') A
right join test.userhis B
on A.userid=B.userid;
"

hive -e "$sql"

```

拉链表的使用：

```

-- 查看拉链表中最新数据(2020-06-23以后的数据)
select * from userhis where end_date='9999-12-31';

-- 查看拉链表中给定日期数据("2020-06-22")
select * from userhis where start_date <= '2020-06-22' and end_date >=
'2020-06-22';

-- 查看拉链表中给定日期数据("2020-06-21")
select * from userhis where start_date <= '2020-06-21' and end_date >=
'2020-06-21';

-- 查看拉链表中给定日期数据("2020-06-20")
select * from userhis where start_date <= '2020-06-20' and end_date >=
'2020-06-20';

```

3、拉链表的回滚

06-20拉链表数据(sh xxx.sh 2020-06-20; 在2020-06-21日凌晨发出命令):

001	13551111111	2020-03-01	2020-06-20	9999-12-31	
002	13561111111	2020-04-01	2020-06-20	9999-12-31	
003	13571111111	2020-05-01	2020-06-20	9999-12-31	
004	13581111111	2020-06-01	2020-06-20	9999-12-31	
001	13551111111	2020-03-01	2020-06-20	9999-12-31	2
002	13561111111	2020-04-01	2020-06-20	9999-12-31	2
003	13571111111	2020-05-01	2020-06-20	9999-12-31	2
004	13581111111	2020-06-01	2020-06-20	9999-12-31	2

06-21拉链表数据(sh xxx.sh 2020-06-21):

001	13551111111	2020-03-01	2020-06-20	9999-12-31	
002	13561111111	2020-04-01	2020-06-20	2020-06-20	
002	13562222222	2020-04-01	2020-06-21	9999-12-31	
003	13571111111	2020-05-01	2020-06-20	9999-12-31	
004	13581111111	2020-06-01	2020-06-20	2020-06-20	
004	13582222222	2020-06-01	2020-06-21	9999-12-31	
005	13552222222	2020-06-21	2020-06-21	9999-12-31	
001	13551111111	2020-03-01	2020-06-20	9999-12-31	2
002	13561111111	2020-04-01	2020-06-20	2020-06-20	1
002	13562222222	2020-04-01	2020-06-21	9999-12-31	2
003	13571111111	2020-05-01	2020-06-20	9999-12-31	2
004	13581111111	2020-06-01	2020-06-20	2020-06-20	1
004	13582222222	2020-06-01	2020-06-21	9999-12-31	2
005	13552222222	2020-06-21	2020-06-21	9999-12-31	2

06-22拉链表数据:

001	13551111111	2020-03-01	2020-06-20	9999-12-31	
002	13561111111	2020-04-01	2020-06-20	2020-06-20	
002	13562222222	2020-04-01	2020-06-21	9999-12-31	
003	13571111111	2020-05-01	2020-06-20	9999-12-31	
004	13581111111	2020-06-01	2020-06-20	2020-06-20	
004	13582222222	2020-06-01	2020-06-21	2020-06-21	
004	13333333333	2020-06-01	2020-06-22	9999-12-31	
005	13552222222	2020-06-21	2020-06-21	2020-06-21	
005	13533333333	2020-06-21	2020-06-22	9999-12-31	
006	13733333333	2020-06-22	2020-06-22	9999-12-31	
001	13551111111	2020-03-01	2020-06-20	9999-12-31	2
002	13561111111	2020-04-01	2020-06-20	2020-06-20	1
002	13562222222	2020-04-01	2020-06-21	9999-12-31	2
003	13571111111	2020-05-01	2020-06-20	9999-12-31	2
004	13581111111	2020-06-01	2020-06-20	2020-06-20	1
004	13582222222	2020-06-01	2020-06-21	2020-06-21	1
004	13333333333	2020-06-01	2020-06-22	9999-12-31	2
005	13552222222	2020-06-21	2020-06-21	2020-06-21	1
005	13533333333	2020-06-21	2020-06-22	9999-12-31	2
006	13733333333	2020-06-22	2020-06-22	9999-12-31	2
001	13551111111	2020-03-01	2020-06-20	9999-12-31	2

002	13561111111	2020-04-01	2020-06-20	2020-06-20	1
002	13562222222	2020-04-01	2020-06-21	9999-12-31	2
003	13571111111	2020-05-01	2020-06-20	9999-12-31	2
004	13581111111	2020-06-01	2020-06-20	2020-06-20	1
004	13582222222	2020-06-01	2020-06-21	2020-06-21	1
004	13333333333	2020-06-01	2020-06-22	9999-12-31	2
005	13552222222	2020-06-21	2020-06-21	2020-06-21	1
005	13533333333	2020-06-21	2020-06-22	9999-12-31	2
006	13733333333	2020-06-22	2020-06-22	9999-12-31	2
06-23拉链表数据：					
001	13551111111	2020-03-01	2020-06-20	2020-06-22	
001	13554444444	2020-03-01	2020-06-23	9999-12-31	
002	13561111111	2020-04-01	2020-06-20	2020-06-20	
002	13562222222	2020-04-01	2020-06-21	9999-12-31	
003	13571111111	2020-05-01	2020-06-20	2020-06-22	
003	13574444444	2020-05-01	2020-06-23	9999-12-31	
004	13581111111	2020-06-01	2020-06-20	2020-06-20	
004	13582222222	2020-06-01	2020-06-21	2020-06-21	
004	13333333333	2020-06-01	2020-06-22	9999-12-31	
005	13552222222	2020-06-21	2020-06-21	2020-06-21	
005	13533333333	2020-06-21	2020-06-22	2020-06-22	
005	13555554444	2020-06-21	2020-06-23	9999-12-31	
006	13733333333	2020-06-22	2020-06-22	9999-12-31	
007	18600744444	2020-06-23	2020-06-23	9999-12-31	
008	18600844444	2020-06-23	2020-06-23	9999-12-31	

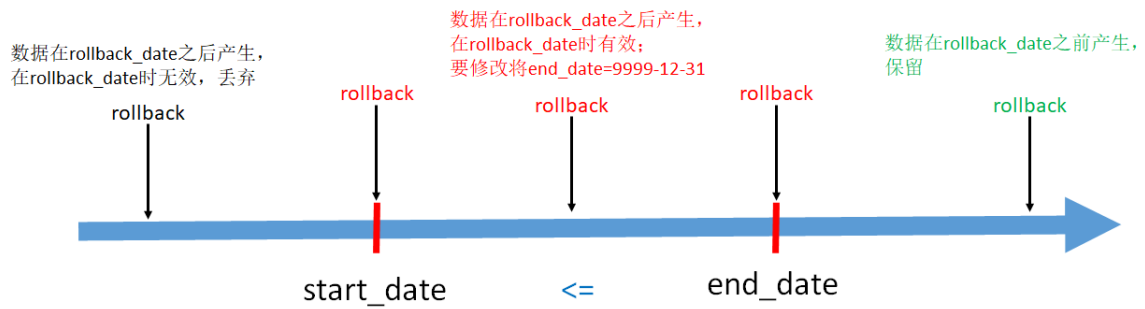
06-23拉链表的状态。假设回滚到2020-06-22，rollback_date

001	13551111111	2020-03-01	2020-06-20	2020-06-22	end_date:9999-12-31; s <= r <= e
001	13554444444	2020-03-01	2020-06-23	9999-12-31	
002	13561111111	2020-04-01	2020-06-20	2020-06-20	end_date < rollback_date 保留
002	13562222222	2020-04-01	2020-06-21	9999-12-31	end_date:9999-12-31
003	13571111111	2020-05-01	2020-06-20	2020-06-22	end_date:9999-12-31
003	13574444444	2020-05-01	2020-06-23	9999-12-31	
004	13581111111	2020-06-01	2020-06-20	2020-06-20	end_date < rollback_date 保留
004	13582222222	2020-06-01	2020-06-21	2020-06-21	end_date < rollback_date 保留
004	13333333333	2020-06-01	2020-06-22	9999-12-31	end_date:9999-12-31
005	13552222222	2020-06-21	2020-06-21	2020-06-21	end_date < rollback_date 保留
005	13533333333	2020-06-21	2020-06-22	2020-06-22	end_date:9999-12-31
005	13555554444	2020-06-21	2020-06-23	9999-12-31	
006	13733333333	2020-06-22	2020-06-22	9999-12-31	end_date:9999-12-31
007	18600744444	2020-06-23	2020-06-23	9999-12-31	
008	18600844444	2020-06-23	2020-06-23	9999-12-31	

由于种种原因需要将拉链表恢复到 rollback_date 那一天的数据。此时有：

- end_date < rollback_date，即结束日期 < 回滚日期。表示该行数据在 rollback_date 之前产生，这些数据需要原样保留
- start_date <= rollback_date <= end_date，即开始日期 <= 回滚日期 <= 结束日期。这些数据是回滚日期之后产生的，但是需要修改。将end_date 改为 9999-12-31

- 其他数据不用管



按以上方案进行编码：

1、处理 end_date < rollback_date 的数据，保留

```
select userid, mobile, regdate, start_date, end_date, '1' as tag
  from test.userhis
 where end_date < '2020-06-22';
```

2、处理 start_date <= rollback_date <= end_date 的数据，设置 end_date=9999-12-31

```
select userid, mobile, regdate, start_date, '9999-12-31' as end_date, '2'
as tag
  from test.userhis
 where start_date <= '2020-06-22' and end_date >= '2020-06-22';
```

3、将前面两步的数据写入临时表tmp（拉链表）

```
drop table test.tmp;
create table test.tmp as
select userid, mobile, regdate, start_date, end_date, '1' as tag
  from test.userhis
 where end_date < '2020-06-22'

union all

select userid, mobile, regdate, start_date, '9999-12-31' as end_date, '2'
as tag
  from test.userhis
 where start_date <= '2020-06-22' and end_date >= '2020-06-22';

select * from test.tmp cluster by userid, start_date;
```

4、模拟脚本

/data/lagoudw/data/zipptmp.sh

```
#!/bin/bash

source /etc/profile

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

sql="
drop table test.tmp;

create table test.tmp as
select userid, mobile, regdate, start_date, end_date, '1' as tag
    from test.userhis
    where end_date < '$do_date'

union all

select userid, mobile, regdate, start_date, '9999-12-31' as end_date, '2'
as tag
    from test.userhis
    where start_date <= '$do_date' and end_date >= '$do_date';
"

hive -e "$sql"
```

逐天回滚，检查数据；

方案二：保存一段时间的增量数据(userinfo)，定期对拉链表做备份（如一个月做一次备份）；如需回滚，直接在备份的拉链表上重跑增量数据。处理简单

5.3、周期性事实表

有如下订单表，6月20号有3条记录(001/002/003)：

订单创建日期	订单编号	订单状态
2020-06-20	001	创建订单
2020-06-20	002	创建订单
2020-06-20	003	支付完成

6月21日，表中有5条记录。其中新增2条记录（004/005），修改1条记录（001）：

订单创建日期	订单编号	订单状态
2020-06-20	001	支付完成（从创建到支付）
2020-06-20	002	创建订单
2020-06-20	003	支付完成
2020-06-21	004	创建订单
2020-06-21	005	创建订单

6月22日，表中有6条记录。其中新增1条记录（006），修改2条记录（003/005）：

订单创建日期	订单编号	订单状态
2020-06-20	001	支付完成
2020-06-20	002	创建订单
2020-06-20	003	已发货（从支付到发货）
2020-06-21	004	创建订单
2020-06-21	005	支付完成（从创建到支付）
2020-06-22	006	创建订单

订单事实表的处理方法：

- 只保留一份全量。数据和6月22日的记录一样，如果需要查看6月21日订单001的状态，则无法满足；
- 每天都保留一份全量。在数据仓库中可以在找到所有的历史信息，但数据量大了，而且很多信息都是重复的，会造成较大的存储浪费；

使用拉链表保存历史信息，会有下面这张表。历史拉链表，既能满足保存历史数据的需求，也能节省存储资源。

订单创建日期	订单编号	订单状态	begin_date	end_date
2020-06-20	001	创建订单	2020-06-20	2020-06-20
2020-06-20	001	支付完成	2020-06-21	9999-12-31
2020-06-20	002	创建订单	2020-06-20	9999-12-31
2020-06-20	003	支付完成	2020-06-20	2020-06-21

2020-06-20 订单创建日期	003 订单编号	已发货 订单状态	2020-06-22 begin_date	9999-12-31 end_date
2020-06-21	004	创建订单	2020-06-21	9999-12-31
2020-06-21	005	创建订单	2020-06-21	2020-06-21
2020-06-21	005	支付完成	2020-06-22	9999-12-31
2020-06-22	006	创建订单	2020-06-22	9999-12-31

1、前提条件

- 订单表的刷新频率为一天，当天获取前一天的增量数据；
- 如果一个订单在一天内有多次状态变化，只记录最后一个状态的信息；
- 订单状态包括三个：创建、支付、完成；
- 创建时间和修改时间只取到天，如果源订单表中没有状态修改时间，那么抽取增量就比较麻烦，需要有个机制来确保能抽取到每天的增量数据；

数仓ODS层有订单表，数据按日分区，存放每天的增量数据：

```
DROP TABLE test.ods_orders;
CREATE TABLE test.ods_orders(
  orderid INT,
  createtime STRING,
  modifiedtime STRING,
  status STRING
) PARTITIONED BY (dt STRING)
row format delimited fields terminated by ',';
```

数仓DWD层有订单拉链表，存放订单的历史状态数据：

```
DROP TABLE test.dwd_orders;
CREATE TABLE test.dwd_orders(
  orderid INT,
  createtime STRING,
  modifiedtime STRING,
  status STRING,
  start_date STRING,
  end_date STRING
)
row format delimited fields terminated by ',';
```

2、周期性事实表拉链表的实现

1、全量初始化

```

-- 数据文件order1.dat
001,2020-06-20,2020-06-20,创建
002,2020-06-20,2020-06-20,创建
003,2020-06-20,2020-06-20,支付

load data local inpath '/data/lagoudw/data/order1.dat' into table
test.ods_orders partition(dt='2020-06-20');

INSERT overwrite TABLE test.dwd_orders
SELECT orderid, createtime, modifiedtime, status,
       createtime AS start_date,
       '9999-12-31' AS end_date
FROM test.ods_orders
WHERE dt='2020-06-20';

```

增量抽取

```

-- 数据文件order2.dat
001,2020-06-20,2020-06-21,支付
004,2020-06-21,2020-06-21,创建
005,2020-06-21,2020-06-21,创建

load data local inpath '/data/lagoudw/data/order2.dat' into table
test.ods_orders partition(dt='2020-06-21');

```

增量刷新历史数据

```

-- 拉链表中的数据分两部实现：新增数据(ods_orders)、历史数据(dwd_orders)

-- 处理新增数据
SELECT orderid,
       createtime,
       modifiedtime,
       status,
       modifiedtime AS start_date,
       '9999-12-31' AS end_date
FROM test.ods_orders
where dt='2020-06-21';

-- 处理历史数据。历史数据包括：有修改、无修改的数据
-- ods_orders 与 dwd_orders 进行表连接
-- 连接上，说明数据被修改
-- 未连接上，说明数据未被修改
select A.orderid,
       A.createtime,
       A.modifiedtime,
       A.status,

```

```

        A.start_date,
        case when B.orderid is not null and A.end_date>'2020-06-21'
            then '2020-06-20'
            else A.end_date
        end end_date
    from dwd_orders A
    left join (select * from ods_orders where dt='2020-06-21') B
    on A.orderid=B.orderid;

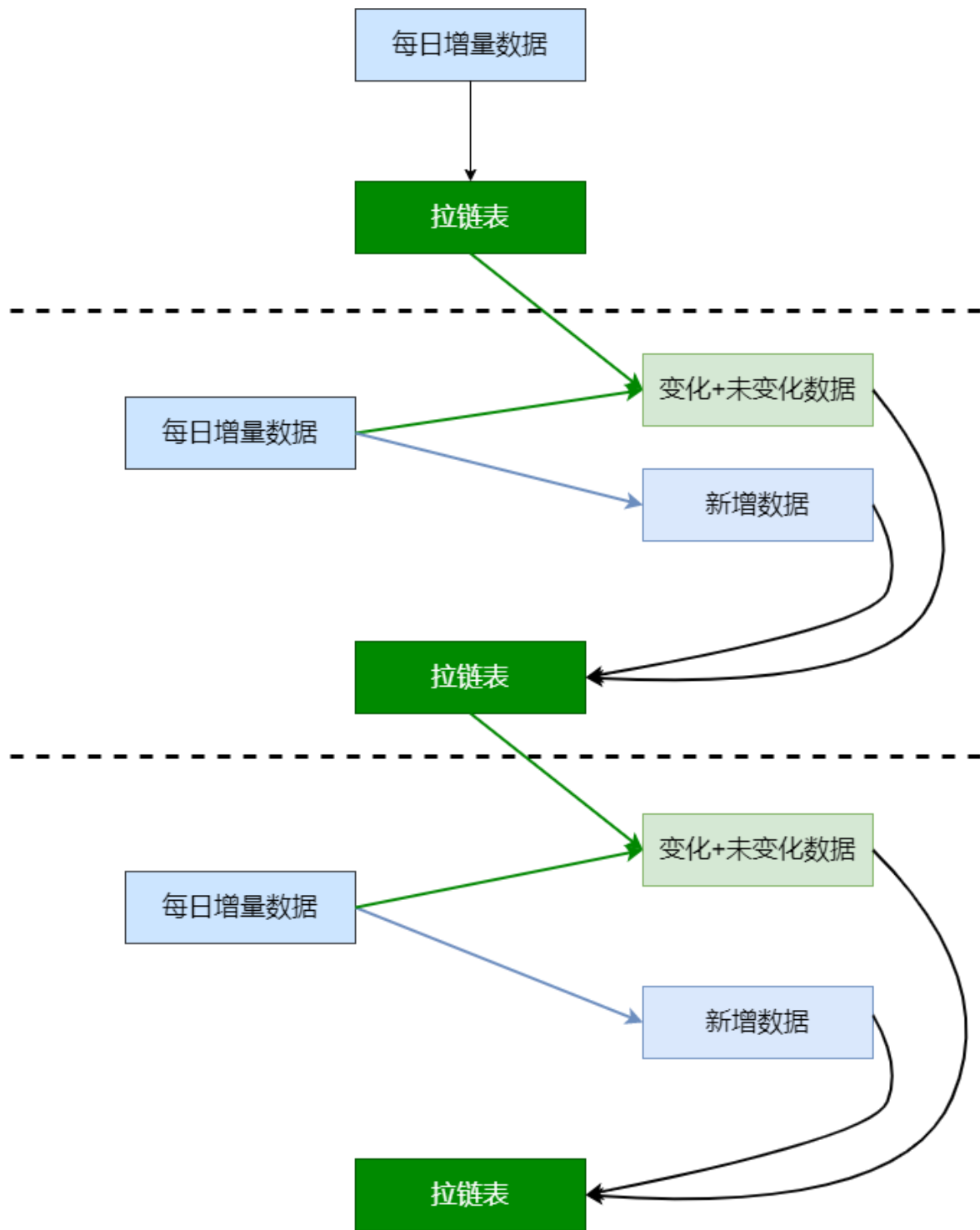
-- 用以上信息覆写拉链表
insert overwrite table test.dwd_orders
SELECT orderid,
        createtime,
        modifiedtime,
        status,
        modifiedtime AS start_date,
        '9999-12-31' AS end_date
    FROM test.ods_orders
where dt='2020-06-21'

union all

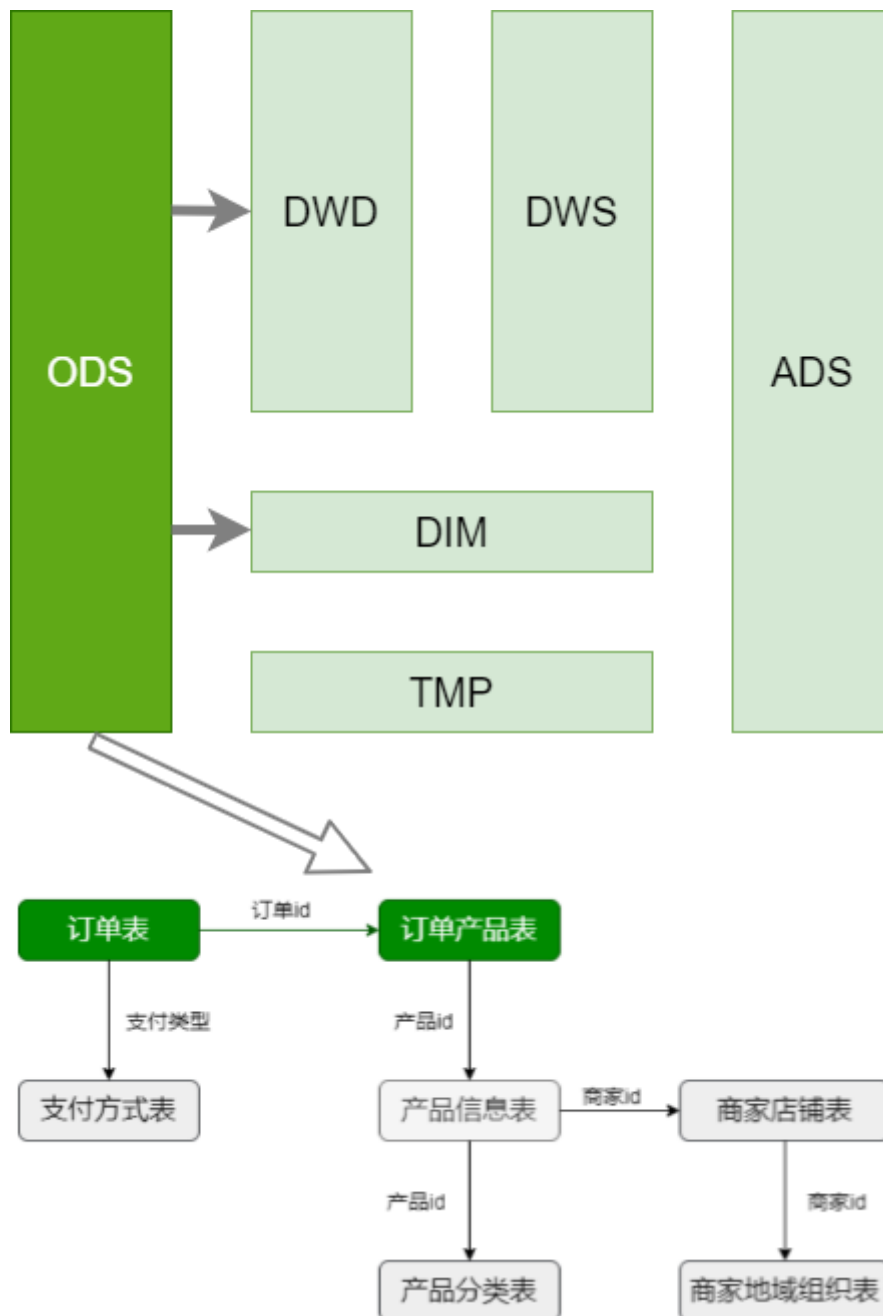
select A.orderid,
        A.createtime,
        A.modifiedtime,
        A.status,
        A.start_date,
        case when B.orderid is not null and A.end_date>'2020-06-21'
            then '2020-06-20'
            else A.end_date
        end end_date
    from dwd_orders A
    left join (select * from ods_orders where dt='2020-06-21') B
    on A.orderid=B.orderid;

```

5.4、拉链表小结



第6节 DIM层建表加载数据



首先要确定哪些是事实表、哪些是维表。绿色的是事实表，灰色的维表

用什么方式处理维表，每日快照、拉链表？

小表使用每日快照：产品分类表、商家店铺表、商家地域组织表、支付方式表

大表使用拉链表：产品信息表

6.1 商品分类表

数据库中的数据是规范的（满足三范式），但是规范化的数据给查询带来不便。

备注：这里对商品分类维度表做了逆规范化

省略了无关信息，做成了宽表

```

DROP TABLE IF EXISTS dim.dim_trade_product_cat;
create table if not exists dim.dim_trade_product_cat(
    firstId int,                -- 一级商品分类id
    firstName string,          -- 一级商品分类名称
    secondId int,              -- 二级商品分类Id
    secondName string,         -- 二级商品分类名称
    thirdId int,               -- 三级商品分类id
    thirdName string           -- 三级商品分类名称
)
partitioned by (dt string)
STORED AS PARQUET;

```

实现:

```

select T1.catid, T1.catname, T2.catid, T2.catname, T3.catid, T3.catname
from (select catid, catname, parentid
      from ods.ods_trade_product_category
      where level=3 and dt='2020-07-01') T3

left join

(select catid, catname, parentid
 from ods.ods_trade_product_category
 where level=2 and dt='2020-07-01') T2
on T3.parentid=T2.catid

left join

(select catid, catname, parentid
 from ods.ods_trade_product_category
 where level=1 and dt='2020-07-01') T1
on T2.parentid=T1.catid;

```

数据加载:

/data/lagoudw/script/trade/dim_load_product_cat.sh

```

#!/bin/bash

source /etc/profile

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

```

```

sql="
insert overwrite table dim.dim_trade_product_cat
partition(dt='$do_date')
select
    t1.catid,          -- 一级分类id
    t1.catname,        -- 一级分类名称
    t2.catid,          -- 二级分类id
    t2.catname,        -- 二级分类名称
    t3.catid,          -- 三级分类id
    t3.catname         -- 三级分类名称
from
    -- 商品三级分类数据
    (select catid, catname, parentid
     from ods.ods_trade_product_category
     where level=3 and dt='$do_date') t3

    left join
    -- 商品二级分类数据
    (select catid, catname, parentid
     from ods.ods_trade_product_category
     where level=2 and dt='$do_date') t2
    on t3.parentid = t2.catid

    left join
    -- 商品一级分类数据
    (select catid, catname, parentid
     from ods.ods_trade_product_category
     where level=1 and dt='$do_date') t1
    on t2.parentid = t1.catid;
"

hive -e "$sql"

```

6.2 商品地域组织表

商家店铺表、商家地域组织表 => 一张维表

这里也是逆规范化的设计，将商家店铺表、商家地域组织表组织成一张表，并拉宽。

在一行数据中体现：商家信息、城市信息、地域信息。信息中包括 id 和 name；

```

drop table if exists dim.dim_trade_shops_org;
create table dim.dim_trade_shops_org(
shopid int,
shopName string,
cityId int,
cityName string ,
regionId int ,
regionName string
)
partitioned by (dt string)
STORED AS PARQUET;

```

实现

```

select T1.shopid, T1.shopname, T2.id cityid, T2.orgname cityname, T3.id
regionid, T3.orgname regionname
  from
(select shopid, shopname, areaid
  from ods.ods_trade_shops
 where dt='2020-07-01') T1

left join

(select id, parentid, orgname, orglevel
  from ods.ods_trade_shop_admin_org
  where orglevel=2 and dt='2020-07-01') T2
on T1.areaid=T2.id

left join

(select id, orgname, orglevel
  from ods.ods_trade_shop_admin_org
  where orglevel=1 and dt='2020-07-01') T3
on T2.parentid=T3.id
limit 10;

```

/data/lagoudw/script/trade/dim_load_shop_org.sh

```

#!/bin/bash

source /etc/profile

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

```

```

fi

sql="
insert overwrite table dim.dim_trade_shops_org
partition(dt='$do_date')
select t1.shopid,
       t1.shopname,
       t2.id as cityid,
       t2.orgname as cityName,
       t3.id as region_id,
       t3.orgname as region_name
from (select shopId, shopName, areaId
      from ods.ods_trade_shops
      where dt='$do_date') t1

      left join
      (select id, parentId, orgname, orglevel
       from ods.ods_trade_shop_admin_org
       where orglevel=2 and dt='$do_date') t2
      on t1.areaId = t2.id

      left join
      (select id, parentId, orgname, orglevel
       from ods.ods_trade_shop_admin_org
       where orglevel=1 and dt='$do_date') t3
      on t2.parentid = t3.id;
"

hive -e "$sql"

```

6.3 支付方式表

对ODS中表的信息做了裁剪，只保留了必要的信息。

```

drop table if exists dim.dim_trade_payment;
create table if not exists dim.dim_trade_payment(
    paymentId string,          -- 支付方式id
    paymentName string        -- 支付方式名称
)
partitioned by (dt string)
STORED AS PARQUET;

```

/data/lagoudw/script/trade/dim_load_payment.sh

```

#!/bin/bash

source /etc/profile

```

```

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

sql="
insert overwrite table dim.dim_trade_payment
partition(dt='$do_date')
select id, payName
    from ods.ods_trade_payments
    where dt='$do_date';
"

hive -e "$sql"

```

6.4 商品信息表

使用拉链表对商品信息进行处理。

- 1、历史数据 => 初始化拉链表(开始日期：当日；结束日期：9999-12-31)【只执行一次】
- 2、链表表的每日处理【每次加载数据时处理】
 - 新增数据。每日新增数据(ODS) => 开始日期：当日；结束日期：9999-12-31
 - 历史数据。拉链表(DIM) 与 每日新增数据(ODS) 做左连接
 - 连接上数据。数据有变化，结束日期：当日；
 - 未连接上数据。数据无变化，结束日期保持不变；

1、创建维表

拉链表要增加两列，分别记录生效日期和失效日期

```

drop table if exists dim.dim_trade_product_info;
create table dim.dim_trade_product_info(
    `productId` bigint,
    `productName` string,
    `shopId` string,
    `price` decimal,
    `isSale` tinyint,
    `status` tinyint,
    `categoryId` string,
    `createTime` string,
    `modifyTime` string,
    `start_dt` string,
    `end_dt` string
) COMMENT '产品表'
STORED AS PARQUET;

```

2、初始数据加载（历史数据加载，只做一次）

```
insert overwrite table dim.dim_trade_product_info
select productId,
       productName,
       shopId,
       price,
       issale,
       status,
       categoryId,
       createTime,
       modifyTime,
       -- modifyTime非空取modifyTime, 否则取createTime; substr取日期
       case when modifyTime is not null
            then substr(modifyTime, 0, 10)
            else substr(createTime, 0, 10)
       end as start_dt,
       '9999-12-31' as end_dt
from ods.ods_trade_product_info
where dt = '2020-07-12';
```

3、增量数据导入（重复执行，每次加载数据执行）

/data/lagoudw/script/trade/dim_load_product_info.sh

```
#!/bin/bash

source /etc/profile

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

sql="
insert overwrite table dim.dim_trade_product_info
select productId,
       productName,
       shopId,
       price,
       issale,
       status,
       categoryId,
       createTime,
       modifyTime,
```



```

        case when modifyTime is not null
            then substr(modifyTime,0,10)
            else substr(createTime,0,10)
        end as start_dt,
        '9999-12-31' as end_dt
    from ods.ods_trade_product_info
    where dt='$do_date'

union all

select dim.productId,
       dim.productName,
       dim.shopId,
       dim.price,
       dim.isSale,
       dim.status,
       dim.categoryId,
       dim.createTime,
       dim.modifyTime,
       dim.start_dt,
       case when dim.end_dt >= '9999-12-31' and ods.productId is not null
           then '$do_date'
           else dim.end_dt
       end as end_dt
    from dim.dim_trade_product_info dim left join
        (select *
         from ods.ods_trade_product_info
         where dt='$do_date' ) ods
        on dim.productId = ods.productId
"

hive -e "$sql"

```

第7节 DWD层建表加载数据

要处理的表有两张：订单表、订单产品表。其中：

- 订单表是周期性事实表；为保留订单状态，可以使用拉链表进行处理；
- 订单产品表普通的事实表，用常规的方法进行处理；
 - 如果有数据清洗、数据转换的业务需求，ODS => DWD
 - 如果没有数据清洗、数据转换的业务需求，保留在ODS，不做任何变化。这个是本项目的处理方式

订单状态：

- -3：用户拒收
- -2：未付款的订单
- -1：用户取消

- 0: 待发货
- 1: 配送中
- 2: 用户确认收货

订单从创建到最终完成，是有时间限制的；业务上也不允许订单在一个月之后，状态仍然在发生变化；

7.1、DWD层建表

备注：

- 与维表不同，订单事实表的记录数非常多
- 订单有生命周期；订单的状态不可能永远处于变化之中（订单的生命周期一般在15天左右）
- 订单是一个拉链表，而且是分区表
- 分区的目的：订单一旦终止，不会重复计算
- 分区的条件：订单创建日期；保证相同的订单在用同一个分区

```
-- 订单事实表(拉链表)
DROP TABLE IF EXISTS dwd.dwd_trade_orders;
create table dwd.dwd_trade_orders(
  `orderId`      int,
  `orderNo`      string,
  `userId`       bigint,
  `status`       tinyint,
  `productMoney` decimal,
  `totalMoney`   decimal,
  `payMethod`    tinyint,
  `isPay`        tinyint,
  `areaId`       int,
  `tradeSrc`     tinyint,
  `tradeType`    int,
  `isRefund`     tinyint,
  `dataFlag`     tinyint,
  `createTime`   string,
  `payTime`      string,
  `modifiedTime` string,
  `start_date`   string,
  `end_date`     string
) COMMENT '订单事实拉链表'
partitioned by (dt string)
STORED AS PARQUET;
```

7.2、DWD层数据加载

```

-- 备注：时间日期格式转换
-- 'yyyy-MM-dd HH:mm:ss' => timestamp => 'yyyy-MM-dd'
select unix_timestamp(modifiedtime, 'yyyy-MM-dd HH:mm:ss')
  from ods.ods_trade_orders limit 10;

select from_unixtime(unix_timestamp(modifiedtime, 'yyyy-MM-dd HH:mm:ss'),
'yyyy-MM-dd')
  from ods.ods_trade_orders limit 10;

```

/data/lagoudw/script/trade/dwd_load_trade_orders.sh

```

#!/bin/bash

source /etc/profile

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

sql="
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.dynamic.partition=true;
INSERT OVERWRITE TABLE dwd.dwd_trade_orders
partition(dt)
SELECT orderId,
        orderNo,
        userId,
        status,
        productMoney,
        totalMoney,
        payMethod,
        isPay,
        areaId,
        tradeSrc,
        tradeType,
        isRefund,
        dataFlag,
        createTime,
        payTime,
        modifiedTime,
        case when modifiedTime is not null
            then from_unixtime(unix_timestamp(modifiedTime, 'yyyy-MM-dd
HH:mm:ss'), 'yyyy-MM-dd')
            else from_unixtime(unix_timestamp(createTime, 'yyyy-MM-dd
HH:mm:ss'), 'yyyy-MM-dd')
        end as start_date,

```

```

        '9999-12-31' as end_date,
        from_unixtime(unix_timestamp(createTime, 'yyyy-MM-dd HH:mm:ss'),
        'yyyy-MM-dd') as dt
    FROM ods.ods_trade_orders
    WHERE dt='$do_date'

union all

SELECT A.orderId,
       A.orderNo,
       A.userId,
       A.status,
       A.productMoney,
       A.totalMoney,
       A.payMethod,
       A.isPay,
       A.areaId,
       A.tradeSrc,
       A.tradeType,
       A.isRefund,
       A.dataFlag,
       A.createTime,
       A.payTime,
       A.modifiedTime,
       A.start_date,
       CASE WHEN B.orderid IS NOT NULL AND A.end_date > '$do_date'
            THEN date_add('$do_date', -1)
            ELSE A.end_date END AS end_date,
       from_unixtime(unix_timestamp(A.createTime, 'yyyy-MM-dd HH:mm:ss'),
        'yyyy-MM-dd') as dt
    FROM (SELECT * FROM dwd.dwd_trade_orders WHERE dt>date_add('$do_date',
-15)) A
    left outer join (SELECT * FROM ods.ods_trade_orders WHERE
dt='$do_date') B
        ON A.orderId = B.orderId;
"

hive -e "$sql"

```

第8节 DWS层建表及数据加载

DIM、DWD => 数据仓库分层、数据仓库理论

需求：计算当天

- 全国所有订单信息
- 全国、一级商品分类订单信息
- 全国、二级商品分类订单信息
- 大区所有订单信息
- 大区、一级商品分类订单信息

- 大区、二级商品分类订单信息
- 城市所有订单信息
- 城市、一级商品分类订单信息
- 城市、二级商品分类订单信息

需要的信息：订单表、订单商品表、商品信息维表、商品分类维表、商家地域维表

订单表 => 订单id、订单状态

订单商品表 => 订单id、商品id、商家id、单价、数量

商品信息维表 => 商品id、三级分类id

商品分类维表 => 一级名称、一级分类id、二级名称、二级分类id、三级名称、三级分类id

商家地域维表 => 商家id、区域名称、区域id、城市名称、城市id

订单表、订单商品表、商品信息维表 => 订单id、商品id、商家id、三级分类id、单价、数量
(订单明细表)

订单明细表、商品分类维表、商家地域维表 => 订单id、商品id、商家id、三级分类名称、三级分类名称、三级分类名称、单价、数量、区域、城市 => 订单明细宽表

8.1、DWS层建表

dws_trade_orders (订单明细) 由以下表轻微聚合而成：

- dwd.dwd_trade_orders (拉链表、分区表)
- ods.ods_trade_order_product (分区表)
- dim.dim_trade_product_info (维表、拉链表)

dws_trade_orders_w (订单明细宽表) 由以下表组成：

- ads.dws_trade_orders (分区表)
- dim.dim_trade_product_cat (分区表)
- dim.dim_trade_shops_org (分区表)

```
-- 订单明细表(轻度汇总事实表)。每笔订单的明细
DROP TABLE IF EXISTS dws.dws_trade_orders;
create table if not exists dws.dws_trade_orders(
    orderid      string,           -- 订单id
    cat_3rd_id   string,           -- 商品三级分类id
    shopid       string,           -- 店铺id
    paymethod    tinyint,          -- 支付方式
    productsnum  bigint,           -- 商品数量
    paymoney     double,           -- 订单商品明细金额
```

```

        paytime      string          -- 订单时间
    )
    partitioned by (dt string)
    STORED AS PARQUET;

-- 订单明细表宽表
DROP TABLE IF EXISTS dws.dws_trade_orders_w;
create table if not exists dws.dws_trade_orders_w(
    orderid string,          -- 订单id
    cat_3rd_id string,      -- 商品三级分类id
    thirdname string,       -- 商品三级分类名称
    secondname string,      -- 商品二级分类名称
    firstname string,       -- 商品一级分类名称
    shopid string,          -- 店铺id
    shopname string,        -- 店铺名
    regionname string,      -- 店铺所在大区
    cityname string,        -- 店铺所在城市
    paymethod tinyint,      -- 支付方式
    productsnum bigint,     -- 商品数量
    paymoney double,        -- 订单明细金额
    paytime string          -- 订单时间
)
    partitioned by (dt string)
    STORED AS PARQUET;

```

8.2、DWS层加载数据

/data/lagoudw/script/trade/dws_load_trade_orders.sh

备注：dws_trade_orders/dws_trade_orders_w 中一笔订单可能出现多条记录！

```

#!/bin/bash

source /etc/profile

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

sql="
insert overwrite table dws.dws_trade_orders
partition(dt='${do_date}')
select t1.orderid      as orderid,
       t3.categoryid as cat_3rd_id,
       t3.shopid      as shopid,
       t1.paymethod   as paymethod,
       t2.productnum  as productsnum,

```

```

        t2.productnum*t2.productprice as pay_money,
        t1.paytime as paytime
from (select orderid, paymethod, paytime
      from dwd.dwd_trade_orders
      where dt='$do_date'
            and status >= 0) T1

left join

(select orderid, productid, productnum, productprice
 from ods.ods_trade_order_product
 where dt='$do_date') T2
on t1.orderid = t2.orderid

left join

(select productid, shopid, categoryid
 from dim.dim_trade_product_info
 where start_dt <= '$do_date'
        and end_dt >= '$do_date' ) T3
on t2.productid=t3.productid;

insert overwrite table dws.dws_trade_orders_w
partition(dt='$do_date')
select t1.orderid,
       t1.cat_3rd_id,
       t2.thirdname,
       t2.secondname,
       t2.firstname,
       t1.shopid,
       t3.shopname,
       t3.regionname,
       t3.cityname,
       t1.paymethod,
       t1.productsnum,
       t1.paymoney,
       t1.paytime
from (select orderid,
             cat_3rd_id,
             shopid,
             paymethod,
             productsnum,
             paymoney,
             paytime
      from dws.dws_trade_orders
      where dt='$do_date') T1

join

(select thirdid, thirdname, secondid, secondname, firstid,
firstname
 from dim.dim_trade_product_cat

```

```

        where dt='$do_date') T2
    on T1.cat_3rd_id = T2.thirdid

join

(select shopid, shopname, regionname, cityname
    from dim.dim_trade_shops_org
    where dt='$do_date') T3
on T1.shopid = T3.shopid
"

hive -e "$sql"

```

备注：要自己准备测试数据！

- dwd.dwd_trade_orders (拉链表、分区表)
- ods.ods_trade_order_product (分区表)
- dim.dim_trade_product_info (维表、拉链表)
- dim.dim_trade_product_cat (分区表)
- dim.dim_trade_shops_org (分区表)

保证测试的日期有数据。

构造测试数据（拉链表分区表）：

```

insert overwrite table dwd.dwd_trade_orders
partition(dt='2020-07-12')
select
orderid,
orderno,
userid,
status,
productmoney,
totalmoney,
paymethod,
ispay,
areaid,
tradesrc,
tradetype,
isrefund,
dataflag,
'2020-07-12',
paytime,
modifiedtime,
start_date,
end_date
from dwd.dwd_trade_orders
where end_date='9999-12-31';

```

第9节 ADS层开发

需求：计算当天

- 全国所有订单信息
- 全国、一级商品分类订单信息
- 全国、二级商品分类订单信息
- 大区所有订单信息
- 大区、一级商品分类订单信息
- 大区、二级商品分类订单信息
- 城市所有订单信息
- 城市、一级商品分类订单信息
- 城市、二级商品分类订单信息

用到的表：

- dws.dws_trade_orders_w

9.1、ADS层建表

```
-- ADS层订单分析表
DROP TABLE IF EXISTS ads.ads_trade_order_analysis;
create table if not exists ads.ads_trade_order_analysis(
    areatype string,           -- 区域范围：区域类型（全国、大区、城市）
    regionname string,        -- 区域名称
    cityname string,          -- 城市名称
    categorytype string,      -- 商品分类类型（一级、二级）
    category1 string,         -- 商品一级分类名称
    category2 string,         -- 商品二级分类名称
    totalcount bigint,        -- 订单数量
    total_productnum bigint,   -- 商品数量
    totalmoney double         -- 支付金额
)
partitioned by (dt string)
row format delimited fields terminated by ',';
```

9.2、ADS层加载数据

/data/lagoudw/script/trade/ads_load_trade_order_analysis.sh

备注：1笔订单，有多个商品；多个商品有不同的分类；这会导致一笔订单有多个分类，它们是分别统计的；

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ]
then
```

```

do_date=$1
else
do_date=`date -d "-1 day" +%F`
fi

sql="
with mid_orders as (
select regionname,
       cityname,
       firstname category1,
       secondname category2,
       count(distinct orderid) as totalcount,
       sum(productsnum) as total_productnum,
       sum(paymoney) as totalmoney
  from dws.dws_trade_orders_w
 where dt='$do_date'
 group by regionname, cityname, firstname, secondname
)
insert overwrite table ads.ads_trade_order_analysis
partition(dt='$do_date')
select '全国' as areatype,
       '' as regionname,
       '' as cityname,
       '' as categorytype,
       '' as category1,
       '' as category2,
       sum(totalcount),
       sum(total_productnum),
       sum(totalmoney)
  from mid_orders

union all

select '全国' as areatype,
       '' as regionname,
       '' as cityname,
       '一级' as categorytype,
       category1,
       '' as category2,
       sum(totalcount),
       sum(total_productnum),
       sum(totalmoney)
  from mid_orders
 group by category1

union all

select '全国' as areatype,
       '' as regionname,
       '' as cityname,
       '二级' as categorytype,
       '' as category1,

```

```

        category2,
        sum(totalcount),
        sum(total_productnum),
        sum(totalmoney)
    from mid_orders
group by category2

union all
select '大区' as areatype,
       regionname,
       '' as cityname,
       '' as categorytype,
       '' as category1,
       '' as category2,
       sum(totalcount),
       sum(total_productnum),
       sum(totalmoney)
    from mid_orders
group by regionname

union all

select '大区' as areatype,
       regionname,
       '' as cityname,
       '一级' as categorytype,
       category1,
       '' as category2,
       sum(totalcount),
       sum(total_productnum),
       sum(totalmoney)
    from mid_orders
group by regionname, category1

union all

select '大区' as areatype,
       regionname,
       '' as cityname,
       '二级' as categorytype,
       '' as category1,
       category2,
       sum(totalcount),
       sum(total_productnum),
       sum(totalmoney)
    from mid_orders
group by regionname, category2

union all

select '城市' as areatype,
       '' as regionname,
```

```

        cityname,
        '' as categorytype,
        '' as category1,
        '' as category2,
        sum(totalcount),
        sum(total_productnum),
        sum(totalmoney)
    from mid_orders
group by cityname

union all

select '城市' as areatype,
        '' as regionname,
        cityname,
        '一级' as categorytype,
        category1,
        '' as category2,
        sum(totalcount),
        sum(total_productnum),
        sum(totalmoney)
    from mid_orders
group by cityname, category1

union all

select '城市' as areatype,
        '' as regionname,
        cityname,
        '二级' as categorytype,
        '' as category1,
        category2,
        sum(totalcount),
        sum(total_productnum),
        sum(totalmoney)
    from mid_orders
group by cityname, category2;
"

hive -e "$sql"

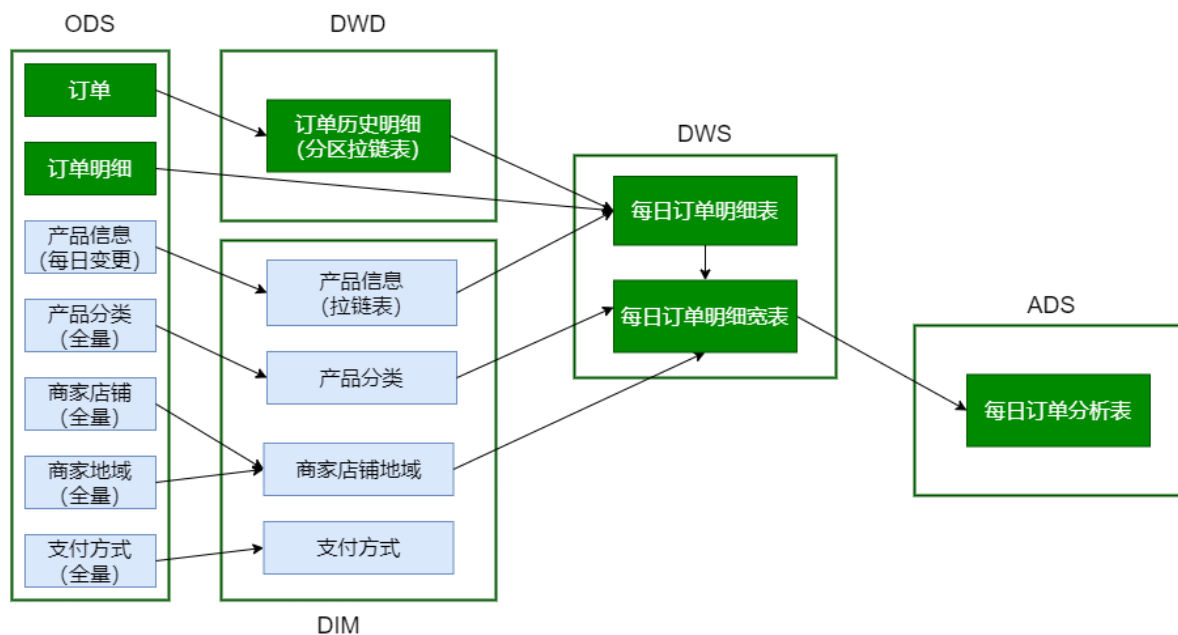
```

备注：由于在dws.dws_trade_orders_w中，一笔订单可能有多条记录，所以在统计订单数量的时候要用count(distinct orderid)

第10节 数据导出

ads.ads_trade_order_analysis 分区表，使用DataX导出到MySQL

第11节 小结



脚本调用次序：

```
# 加载ODS数据（含DataX迁移数据）
/data/lagoudw/script/trade/ods_load_trade.sh

# 加载DIM层数据
/data/lagoudw/script/trade/dim_load_product_cat.sh
/data/lagoudw/script/trade/dim_load_shop_org.sh
/data/lagoudw/script/trade/dim_load_payment.sh
/data/lagoudw/script/trade/dim_load_product_info.sh

# 加载DWD层数据
/data/lagoudw/script/trade/dwd_load_trade_orders.sh

# 加载DWS层数据
/data/lagoudw/script/trade/dws_load_trade_orders.sh

# 加载ADS层数据
/data/lagoudw/script/trade/ads_load_trade_order_analysis.sh
```

主要技术点：

- 拉链表。创建、使用与回滚；商品信息表、订单表（周期性事实表；分区表+拉链表）
- 宽表（逆规范化）：商品分类表、商品地域组织表、订单明细及订单明细宽表（轻度汇总的事实表）

第二部分 任务调度系统Airflow

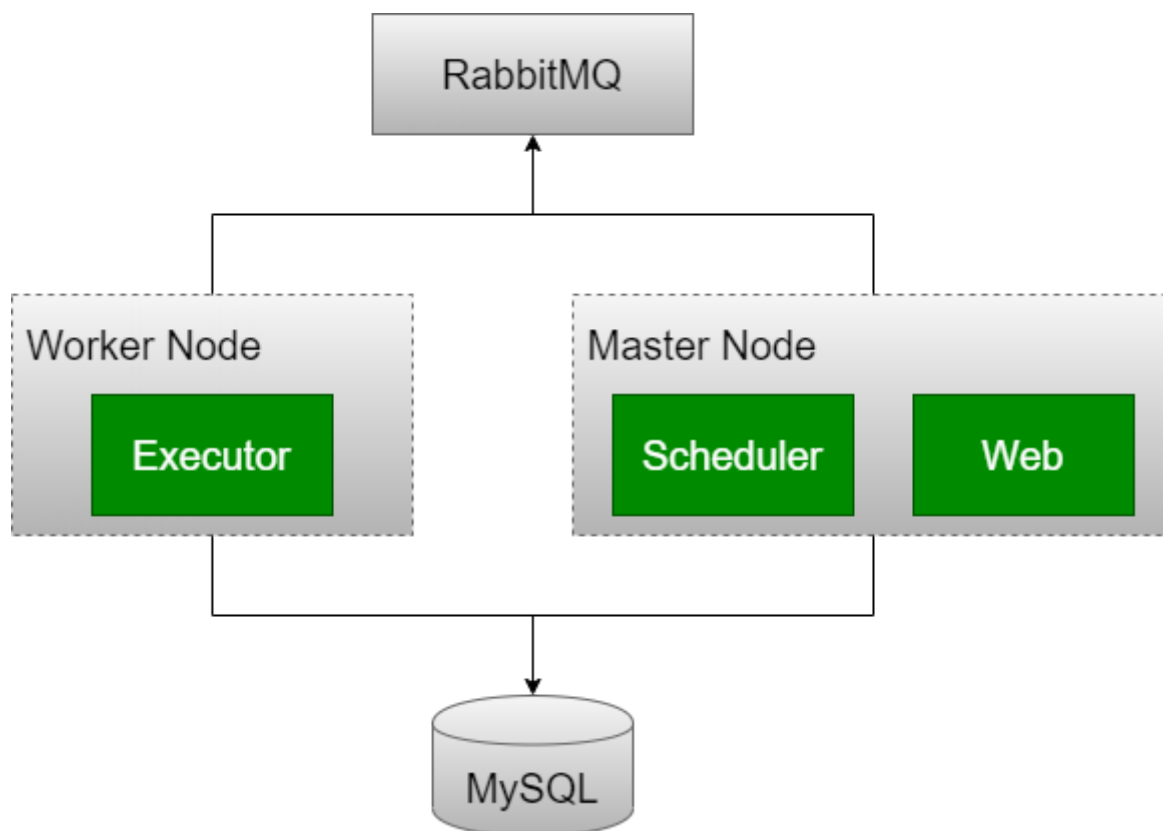
第1节 Airflow简介

Airflow 是 Airbnb 开源的一个用 Python 编写的调度工具。于 2014 年启动，2015 年春季开源，2016 年加入 Apache 软件基金会的孵化计划。

Airflow 将 workflow 制定为一组任务的有向无环图（DAG），并指派到一组计算节点上，根据相互之间的依赖关系，有序执行。Airflow 有以下优势：

- 灵活易用。Airflow 是 Python 编写的，工作流的定义也使用 Python 编写；
- 功能强大。支持多种不同类型的作业，可自定义不同类型的作业。如 Shell、Python、Mysql、Oracle、Hive 等；
- 简洁优雅。作业的定义简单明了；
- 易扩展。提供各种基类供扩展，有多种执行器可供选择；

1.1、体系架构



Webserver 守护进程。接受 HTTP 请求，通过 Python Flask Web 应用程序与 airflow 进行交互。Webserver 提供功能的功能包括：中止、恢复、触发任务；监控正在运行的任务，断点续跑任务；查询任务的状态，日志等详细信息。

Scheduler 守护进程。周期性地轮询任务的调度计划，以确定是否触发任务执行。

Worker 守护进程。Worker 负责启动机器上的 executor 来执行任务。使用 celeryExecutor 后可以在多个机器上部署 worker 服务。

1.2、重要概念

DAG (Directed Acyclic Graph) 有向无环图

- 在Airflow中，一个DAG定义了一个完整的作业。同一个DAG中的所有Task拥有相同的调度时间。
- 参数：
 - dag_id: 唯一识别DAG
 - default_args: 默认参数，如果当前DAG实例的作业没有配置相应参数，则采用DAG实例的default_args中的相应参数
 - schedule_interval: 配置DAG的执行周期，可采用crontab语法

Task

- Task为DAG中具体的作业任务，依赖于DAG，必须存在于某个DAG中。Task在DAG中可以配置依赖关系
- 参数：
 - dag: 当前作业属于相应DAG
 - task_id: 任务标识符
 - owner: 任务的拥有者
 - start_date: 任务的开始时间

第2节 Airflow安装部署

2.1、安装依赖

- CentOS 7.X
- Python 3.5或以上版本（推荐）
- MySQL 5.7.x
- Apache-Airflow 1.10.11
- 虚拟机可上网，需在线安装包

正式安装之前给虚拟机做一个备份；

2.2、Python环境准备

备注：提前下载 Python-3.6.6.tgz

备注：使用linux122安装

```
# 卸载 mariadb
rpm -qa | grep mariadb
mariadb-libs-5.5.65-1.el7.x86_64
mariadb-5.5.65-1.el7.x86_64
```

```

mariadb-devel-5.5.65-1.el7.x86_64

yum remove mariadb
yum remove mariadb-libs

# 安装依赖
rpm -ivh mysql57-community-release-el7-11.noarch.rpm

yum install readline readline-devel -y
yum install gcc -y
yum install zlib* -y
yum install openssl openssl-devel -y
yum install sqlite-devel -y
yum install python-devel mysql-devel -y

# 提前到python官网下载好包
tar -zxvf Python-3.6.6.tgz

# 安装 python3 运行环境
cd Python-3.6.6/
# configure文件是一个可执行的脚本文件。如果配置了--prefix，安装后的所有资源文件
# 都会放在目录中
./configure --prefix=/usr/local/python3.6
make && make install
/usr/local/python3.6/bin/pip3 install virtualenv

# 启动 python3 环境
cd /usr/local/python3.6/bin/
./virtualenv env
. env/bin/activate

# 检查 python 版本
python -V

```

2.3、安装Airflow

```

# 设置目录（配置文件）
# 添加到配置文件/etc/profile。未设置是缺省值为 ~/airflow
export AIRFLOW_HOME=/opt/lagou/servers/airflow

# 使用豆瓣源非常快。-i：指定库的安装源（可选选项）
pip install apache-airflow -i https://pypi.douban.com/simple

# 备注：可以设置安装的版本【不用执行】
pip install apache-airflow=1.10.10 -i https://pypi.douban.com/simple

```

备注：

- 软件安装路径在\$AIRFLOW_HOME（缺省为~/airflow），此时目录不存在
- 安装的是版本是1.10.11，不指定下载源时下载过程非常慢

2.4、创建数据库用户并授权

```
-- 创建数据库
create database airflowlinux122;

-- 创建用户airflow, 设置所有ip均可以访问
create user 'airflow'@'%' identified by '12345678';
create user 'airflow'@'localhost' identified by '12345678';

-- 用户授权, 为新建的airflow用户授予Airflow库的所有权限
grant all on airflowlinux122.* to 'airflow'@'%';
SET GLOBAL explicit_defaults_for_timestamp = 1;
flush privileges;
```

2.5、修改Airflow DB配置

```
# python3 环境中执行
pip install mysqlclient
airflow initdb
```

备注: 有可能在安装完Airflow找不到 \$AIRFLOW_HOME/airflow.cfg 文件, 执行完airflow initdb才会对应的位置找到该文件。

修改 \$AIRFLOW_HOME/airflow.cfg:

```
# 约 75 行
sql_alchemy_conn = mysql://airflow:12345678@linux123:3306/airflowlinux122

# 重新执行
airflow initdb
```

可能出现的错误: Exception: Global variable explicit_defaults_for_timestamp needs to be on (1) for mysql

解决方法:

```
SET GLOBAL explicit_defaults_for_timestamp = 1;
FLUSH PRIVILEGES;
```

2.6、安装密码模块

安装password组件:

```
pip install apache-airflow[password]
```

修改 airflow.cfg 配置文件（第一行修改，第二行增加）：

```
# 约 281 行
[webserver]
# 约 353行
authenticate = True
auth_backend = airflow.contrib.auth.backends.password_auth
```

- 添加密码文件

python命令，执行一遍；添加用户登录，设置口令

```
import airflow
from airflow import models, settings
from airflow.contrib.auth.backends.password_auth import PasswordUser

user = PasswordUser(models.User())
user.username = 'airflow'
user.email = 'airflow@lagou.com'
user.password = 'airflow123'

session = settings.Session()
session.add(user)
session.commit()
session.close()
exit()
```

2.7、启动服务

```
# 备注：要先进入python3的运行环境
cd /usr/local/python3.6/bin/
./virtualenv env
. env/bin/activate

# 退出虚拟环境命令
deactivate

# 启动scheduler调度器：
airflow scheduler -D

# 服务页面启动：
airflow webserver -D
```

备注：airflow命令所在位置：/usr/local/python3.6/bin/env/bin/airflow

安装完成，可以使用浏览器登录 linux122:8080；输入用户名、口令：airflow / airflow123

2.8、修改时区

Airflow默认使用UTC时间，在中国时区需要用+8小时。将UTC修改为中国时区，需要修改Airflow源码。

1、在修改 \$AIRFLOW_HOME/airflow.cfg 文件

```
# 约 65 行
default_timezone = Asia/Shanghai
```

2、修改 timezone.py

```
# 进入Airflow包的安装位置
cd /usr/local/python3.6/bin/env/lib/python3.6/site-packages/

# 修改airflow/utils/timezone.py
cd airflow/utils
vi timezone.py
```

第27行注释，增加29-37行：

```
27 # utc = pendulum.timezone('UTC')
28
29 from airflow import configuration as conf
30 try:
31     tz = conf.get("core", "default_timezone")
32     if tz == "system":
33         utc = pendulum.local_timezone()
34     else:
35         utc = pendulum.timezone(tz)
36 except Exception:
37     pass
```

备注：以上的修改方式有警告，可以使用下面的方式（推荐）：

```

27 # utc = pendulum.timezone('UTC')
28
29 from airflow import configuration
30 try:
31     tz = configuration.conf("core", "default_timezone")
32     if tz == "system":
33         utc = pendulum.local_timezone()
34     else:
35         utc = pendulum.timezone(tz)
36 except Exception:
37     pass

```

修改utcnow()函数 (注释掉72行, 增加73行内容)

```

62 def utcnow():
63     """
64     Get the current date and time in UTC
65
66     :return:
67     """
68
69     # pendulum utcnow() is not used as that sets a TimezoneInfo
object
70     # instead of a Timezone. This is not pickable and also creates
issues
71     # when using replace()
72     # d = dt.datetime.utcnow()
73     d = dt.datetime.now()
74     d = d.replace(tzinfo=utc)
75
76     return d

```

3、修改 airflow/utils/sqlalchemy.py

```

# 进入Airflow包的安装位置
cd /usr/local/python3.6/bin/env/lib/python3.6/site-packages/

# 修改 airflow/utils/sqlalchemy.py
cd airflow/utils
vi sqlalchemy.py

```

在38行之后增加 39 - 47 行的内容:

```

38 utc = pendulum.timezone('UTC')
39 from airflow import configuration as conf
40 try:
41     tz = conf.get("core", "default_timezone")
42     if tz == "system":
43         utc = pendulum.local_timezone()
44     else:
45         utc = pendulum.timezone(tz)
46 except Exception:
47     pass

```

备注：以上的修改方式有警告，可以使用下面的方式（推荐）：

```

38 utc = pendulum.timezone('UTC')
39 from airflow import configuration
40 try:
41     tz = configuration.conf("core", "default_timezone")
42     if tz == "system":
43         utc = pendulum.local_timezone()
44     else:
45         utc = pendulum.timezone(tz)
46 except Exception:
47     pass

```

4、修改airflow/www/templates/admin/master.html

```

# 进入Airflow包的安装位置
cd /usr/local/python3.6/bin/env/lib/python3.6/site-packages/

# 修改 airflow/www/templates/admin/master.html
cd airflow/www/templates/admin
vi master.html

```

```

# 将第40行修改为以下内容：
40     var UTCseconds = x.getTime();

# 将第43行修改为以下内容：
43         "timeFormat": "H:i:s",

```

重启airflow webserver

```
# 关闭 airflow webserver 对应的服务
ps -ef | grep 'airflow-webserver' | grep -v 'grep' | awk '{print $2}' |
xargs -i kill -9 {}

# 关闭 airflow scheduler 对应的服务
ps -ef | grep 'airflow' | grep 'scheduler' | awk '{print $2}' | xargs -i
kill -9 {}

# 删除对应的pid文件
cd $AIRFLOW_HOME
rm -rf *.pid

# 重启服务（在python3.6虚拟环境中执行）
airflow scheduler -D
airflow webserver -D
```

2.9、Airflow的web界面

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	example_bash_operator	@*	Airflow				
	example_branch_dop_operator_v3	*T*	Airflow				
	example_branch_operator	@daily	Airflow				
	example_complex	None	airflow				
	example_external_task_marker_child	None	airflow				
	example_external_task_marker_parent	None	airflow				
	example_http_operator	1 day, 0:00:00	Airflow				

Trigger Dag：人为执行触发

Tree View：当dag执行的时候，可以点入，查看每个task的执行状态（基于树状视图）。状态：success、running、failed、skipped、retry、queued、no status

Graph View：基于图视图（有向无环图），查看每个task的执行状态

Tasks Duration：每个task的执行时间统计，可以选择最近多少次执行

Task Tries：每个task的重试次数

Gantt View：基于甘特图的视图，每个task的执行状态

Code View：查看任务执行代码

Logs：查看执行日志，比如失败原因

Refresh：刷新dag任务

Delete Dag：删除该dag任务

2.10、禁用自带的DAG任务

停止服务：

```
# 关闭 airflow webserver 对应的服务
ps -ef | grep 'airflow-webserver' | grep -v 'grep' | awk '{print $2}' |
xargs -i kill -9 {}

# 关闭 airflow scheduler 对应的服务
ps -ef | grep 'airflow' | grep 'scheduler' | awk '{print $2}' | xargs -i
kill -9 {}

# 删除对应的pid文件
cd $AIRFLOW_HOME
rm -rf *.pid
```

修改文件 \$AIRFLOW_HOME/airflow.cfg：

```
# 修改文件第 136 行
136 # load_examples = True
137 load_examples = False

# 重新设置db
airflow resetdb -y
```

重新设置账户、口令：

```
import airflow
from airflow import models, settings
from airflow.contrib.auth.backends.password_auth import PasswordUser

user = PasswordUser(models.User())
user.username = 'airflow'
user.email = 'airflow@lagou.com'
user.password = 'airflow123'

session = settings.Session()
session.add(user)
session.commit()
session.close()
exit()
```

重启服务

```
# 重启服务
airflow scheduler -D
airflow webserver -D
```

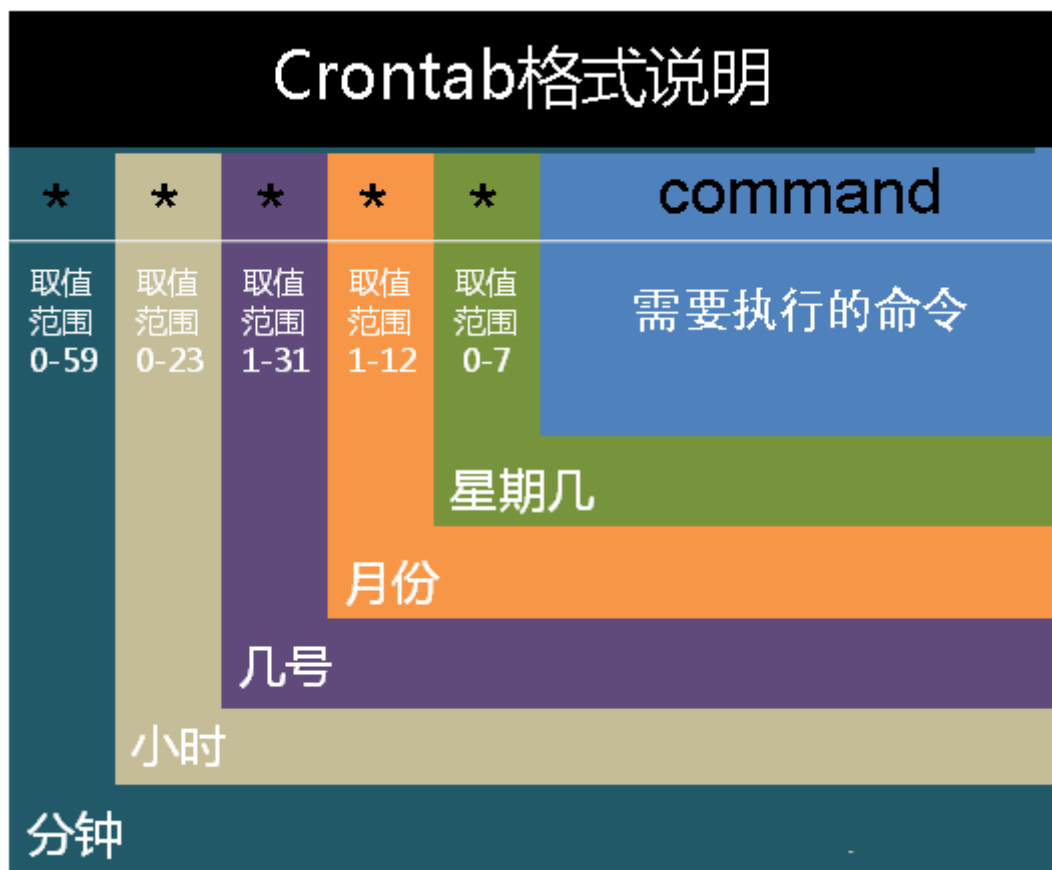
2.11、crontab简介

Linux 系统则是由 cron (crond) 这个系统服务来控制的。Linux 系统上面原本就有非常多的计划性工作，因此这个系统服务是默认启动的。

Linux 系统也提供了Linux用户控制计划任务的命令：crontab 命令。



- 日志文件：ll /var/log/cron*
- 编辑文件：vim /etc/crontab
- 进程：ps -ef | grep crond ==> /etc/init.d/crond restart
- 作用：任务（命令）定时调度（如：定时备份，实时备份）
- 简要说明：cat /etc/crontab



在以上各个字段中，还可以使用以下特殊字符：

* 代表所有的取值范围内的数字。如月份字段为*，则表示1到12个月；

/ 代表每一定时间间隔的意思。如分钟字段为*/10，表示每10分钟执行1次；

- 代表从某个区间范围，是闭区间。如2-5表示2,3,4,5，小时字段中0-23/2表示在0~23点范围内每2个小时执行一次；

, 分散的数字（不连续）。如1,2,3,4,7,9；

注：由于各个地方每周第一天不一样，因此Sunday=0（第1天）或Sunday=7（最后1天）。

crontab配置实例

```
# 每一分钟执行一次command（因cron默认每1分钟扫描一次，因此全为*即可）
```

```
* * * * * command
```

```
# 每小时的第3和第15分钟执行command
```

```
3,15 * * * * command
```

```
# 每天上午8-11点的第3和15分钟执行command
```

```
3,15 8-11 * * * command
```

```
# 每隔2天的上午8-11点的第3和15分钟执行command
```

```
3,15 8-11 */2 * * command
```

```
# 每个星期一的上午8点到11点的第3和第15分钟执行command
```

```
3,15 8-11 * * 1 command
```

```
# 每晚的21:30执行command
```

```
30 21 * * * command
```

```
# 每月1、10、22日的4:45执行command
```

```
45 4 1,10,22 * * command
```

```
# 每周六、周日的1 : 10执行command
```

```
10 1 * * 6,0 command
```

```
# 每小时执行command
```

```
0 */1 * * * command
```

```
# 晚上11点到早上7点之间，每隔一小时执行command
```

```
* 23-7/1 * * * command
```

第3节 任务集成部署

3.1、Airflow核心概念

- DAGs：有向无环图(Directed Acyclic Graph)，将所有需要运行的tasks按照依赖关系组织起来，描述的是所有tasks执行的顺序；

- Operators: Airflow内置了很多operators
 - BashOperator 执行一个bash 命令
 - PythonOperator 调用任意的 Python 函数
 - EmailOperator 用于发送邮件
 - HTTPOperator 用于发送HTTP请求
 - SqlOperator 用于执行SQL命令
 - 自定义Operator
- Tasks: Task 是 Operator的一个实例;
- Task Instance: 由于Task会被重复调度, 每次task的运行就是不同的 Task instance。Task instance 有自己的状态, 包括 `success`、`running`、`failed`、`skipped`、`up_for_reschedule`、`up_for_retry`、`queued`、`no_status` 等;
- Task Relationships: DAGs中的不同Tasks之间可以有依赖关系;
- 执行器 (Executor) 。Airflow支持的执行器就有四种:
 - SequentialExecutor: 单进程顺序执行任务, 默认执行器, 通常只用于测试
 - LocalExecutor: 多进程本地执行任务
 - CeleryExecutor: 分布式调度, 生产常用。Celery是一个分布式调度框架, 其本身无队列功能, 需要使用第三方组件, 如RabbitMQ
 - DaskExecutor : 动态任务调度, 主要用于数据分析
 - 执行器的修改。修改 `$AIRFLOW_HOME/airflow.cfg` 第 70行: `executor = LocalExecutor`。修改后启动服务

3.2、入门案例

放置在 `$AIRFLOW_HOME/dags` 目录下

```
from datetime import datetime, timedelta

from airflow import DAG
from airflow.utils import dates
from airflow.utils.helpers import chain
from airflow.operators.bash_operator import BashOperator
from airflow.operators.python_operator import PythonOperator

def default_options():
    default_args = {
        'owner': 'airflow',                # 拥有者名称
        'start_date': dates.days_ago(1),   # 第一次开始执行的时间
        'retries': 1,                       # 失败重试次数
        'retry_delay': timedelta(seconds=5) # 失败重试间隔
    }
    return default_args

# 定义DAG
def task1(dag):
    t = "pwd"
    # operator支持多种类型, 这里使用 BashOperator
```

```

task = BashOperator(
    task_id='MyTask1',                # task_id
    bash_command=t,                   # 指定要执行的命令
    dag=dag                           # 指定归属的dag
)
return task

def hello_world():
    current_time = str(datetime.today())
    print('hello world at {}'.format(current_time))

def task2(dag):
    # Python Operator
    task = PythonOperator(
        task_id='MyTask2',
        python_callable=hello_world,  # 指定要执行的函数
        dag=dag)
    return task

def task3(dag):
    t = "date"
    task = BashOperator(
        task_id='MyTask3',
        bash_command=t,
        dag=dag)
    return task

with DAG(
    'HelloworldDag',                 # dag_id
    default_args=default_options(),  # 指定默认参数
    schedule_interval="*/2 * * * *"  # 执行周期，每分钟2次
) as d:
    task1 = task1(d)
    task2 = task2(d)
    task3 = task3(d)
    chain(task1, task2, task3)       # 指定执行顺序

```

执行命令检查脚本是否有错误。如果命令行没有报错，就表示没问题
python \$AIRFLOW_HOME/dags/helloworld.py

查看生效的 dags
airflow list_dags -sd \$AIRFLOW_HOME/dags

查看指定dag中的task
airflow list_tasks HelloworldDag

测试dag中的task
airflow test HelloworldDag MyTask2 20200801

3.3、核心交易调度任务集成

核心交易分析

```
# 加载ODS数据（DataX迁移数据）
/data/lagoudw/script/trade/ods_load_trade.sh

# 加载DIM层数据
/data/lagoudw/script/trade/dim_load_product_cat.sh
/data/lagoudw/script/trade/dim_load_shop_org.sh
/data/lagoudw/script/trade/dim_load_payment.sh
/data/lagoudw/script/trade/dim_load_product_info.sh

# 加载DWD层数据
/data/lagoudw/script/trade/dwd_load_trade_orders.sh

# 加载DWS层数据
/data/lagoudw/script/trade/dws_load_trade_orders.sh

# 加载ADS层数据
/data/lagoudw/script/trade/ads_load_trade_order_analysis.sh
```

备注： `depends_on_past`，设置为True时，上一次调度成功了，才可以触发。

`$AIRFLOW_HOME/dags`

```
from datetime import timedelta
import datetime
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago

# 定义dag的缺省参数
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': '2020-06-20',
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

# 定义DAG
coretradedag = DAG(
    'coretrade',
    default_args=default_args,
```

```

        description='core trade analyze',
        schedule_interval='30 0 * * *',
    )

today=datetime.date.today()
oneday=timedelta(days=1)
yesterday=(today-oneday).strftime("%Y-%m-%d")

odstask = BashOperator(
    task_id='ods_load_data',
    depends_on_past=False,
    bash_command='sh /data/lagoudw/script/trade/ods_load_trade.sh ' +
yesterday,
    dag=coretradedag
)

dimtask1 = BashOperator(
    task_id='dimtask_product_cat',
    depends_on_past=False,
    bash_command='sh /data/lagoudw/script/trade/dim_load_product_cat.sh '
+ yesterday,
    dag=coretradedag
)

dimtask2 = BashOperator(
    task_id='dimtask_shop_org',
    depends_on_past=False,
    bash_command='sh /data/lagoudw/script/trade/dim_load_shop_org.sh ' +
yesterday,
    dag=coretradedag
)

dimtask3 = BashOperator(
    task_id='dimtask_payment',
    depends_on_past=False,
    bash_command='sh /data/lagoudw/script/trade/dim_load_payment.sh ' +
yesterday,
    dag=coretradedag
)

dimtask4 = BashOperator(
    task_id='dimtask_product_info',
    depends_on_past=False,
    bash_command='sh /data/lagoudw/script/trade/dim_load_product_info.sh '
+ yesterday,
    dag=coretradedag
)

dwddtask = BashOperator(
    task_id='dwd_load_data',
    depends_on_past=False,

```

```

        bash_command='sh /data/lagoudw/script/trade/dwd_load_trade_orders.sh
'+ yesterday,
        dag=coretradedag
    )

    dwstask = BashOperator(
        task_id='dws_load_data',
        depends_on_past=False,
        bash_command='sh /data/lagoudw/script/trade/dws_load_trade_orders.sh '
+ yesterday,
        dag=coretradedag
    )

    adstask = BashOperator(
        task_id='ads_load_data',
        depends_on_past=False,
        bash_command='sh
/data/lagoudw/script/trade/ads_load_trade_order_analysis.sh ' + yesterday,
        dag=coretradedag
    )

    odstask >> dimtask1
    odstask >> dimtask2
    odstask >> dimtask3
    odstask >> dimtask4
    odstask >> dwdtask

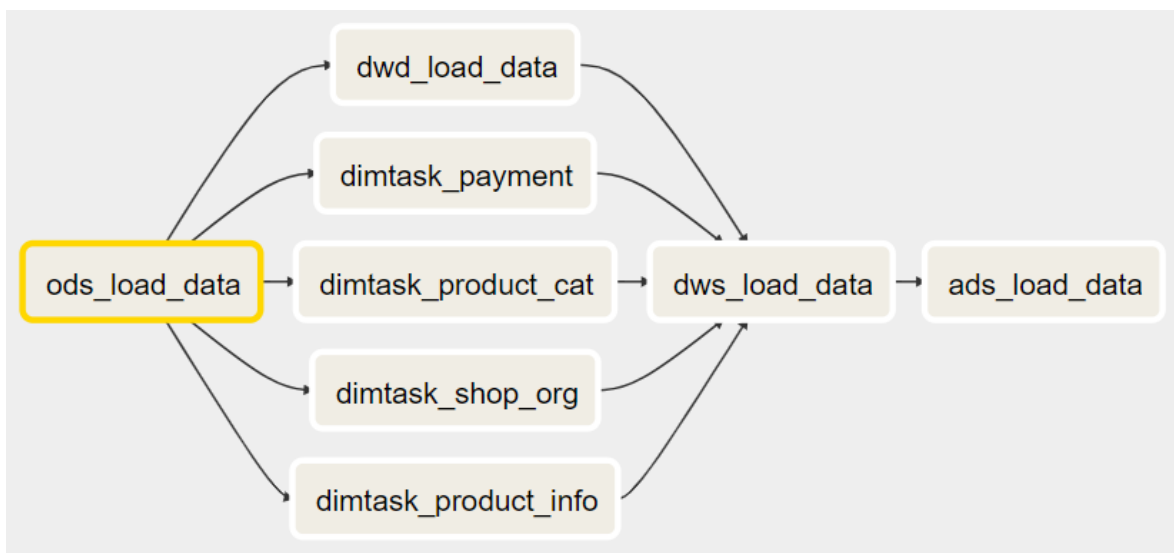
    dimtask1 >> dwstask
    dimtask2 >> dwstask
    dimtask3 >> dwstask
    dimtask4 >> dwstask
    dwdtask >> dwstask

    dwstask >> adstask

```

airflow list_dags

airflow list_tasks coretrade --tree



第三部分 元数据管理工具Atlas（扩展）

第1节 数据仓库元数据管理

元数据（MetaData）狭义的解释是用来描述数据的数据。广义的来看，除了业务逻辑直接读写处理的那些业务数据，所有其它用来维持整个系统运转所需的信息 / 数据都可以叫作元数据。如数据库中表的Schema信息，任务的血缘关系，用户和脚本 / 任务的权限映射关系信息等。

管理元数据的目的是，是为了让用户能够更高效的使用数据，也是为了让平台管理人员能更加有效的做好数据的维护管理工作。

但通常这些元数据信息是散落在平台的各个系统，各种流程之中的，它们的管理也可能或多或少可以通过各种子系统自身的工具，方案或流程逻辑来实现。

元数据管理平台很重要的一个功能就是信息的收集，至于收集哪些信息，取决于业务的需求和需要解决的目标问题。

元数据管理平台还需要考虑如何以恰当的形式对这些元数据信息进行展示；进一步的，如何将这些元数据信息通过服务的形式提供给周边上下游系统使用，真正帮助大数据平台完成质量管理的闭环工作。

应该收集那些信息，没有绝对的标准，但是对大数据开发平台来说，常见的元数据信息包括：

- 表结构信息
- 数据的空间存储，读写记录，权限归属和其它各类统计信息
- 数据的血缘关系信息
- 数据的业务属性信息

数据血缘关系。血缘信息或者叫做Lineage的血统信息是什么，简单的说就是数据之间的上下游来源去向关系，数据从哪里来到哪里去。如果一个数据有问题，可以根据血缘关系往上游排查，看看到底在哪个环节出了问题。此外也可以通过数据的血缘关系，建立起生产这些数据的任务之间的依赖关系，进而辅助调度系统的工作调度，或者用来判断一个失败或错误的任务可能对哪些

下游数据造成影响等等。

分析数据的血缘关系看起来简单，但真的要做起来，并不容易，因为数据的来源多种多样，加工数据的手段，所使用的计算框架可能也各不相同，此外也不是所有的系统天生都具备获取相关信息的能力。而针对不同的系统，血缘关系具体能够分析到的粒度可能也不一样，有些能做到表级别，有些甚至可以做到字段级别。

以Hive表为例，通过分析Hive脚本的执行计划，是可以做到相对精确的定位出字段级别的数据血缘关系的。而如果是一个MapReduce任务生成的数据，从外部来看，可能就只能通过分析MR任务输出的Log日志信息来粗略判断目录级别的读写关系，从而间接推导数据的血缘依赖关系了。

数据的业务属性信息。业务属性信息都有哪些呢？如一张数据表的统计口径信息，这张表干什么用的，各个字段的具体统计方式，业务描述，业务标签，脚本逻辑的历史变迁记录，变迁原因等，此外还包括对应的数据表格是由谁负责开发的，具体数据的业务部门归属等。数据的业务属性信息，首先是为业务服务的，它的采集和展示也就需要尽可能的和业务环境相融合，只有这样才能真正发挥这部分元数据信息的作用。

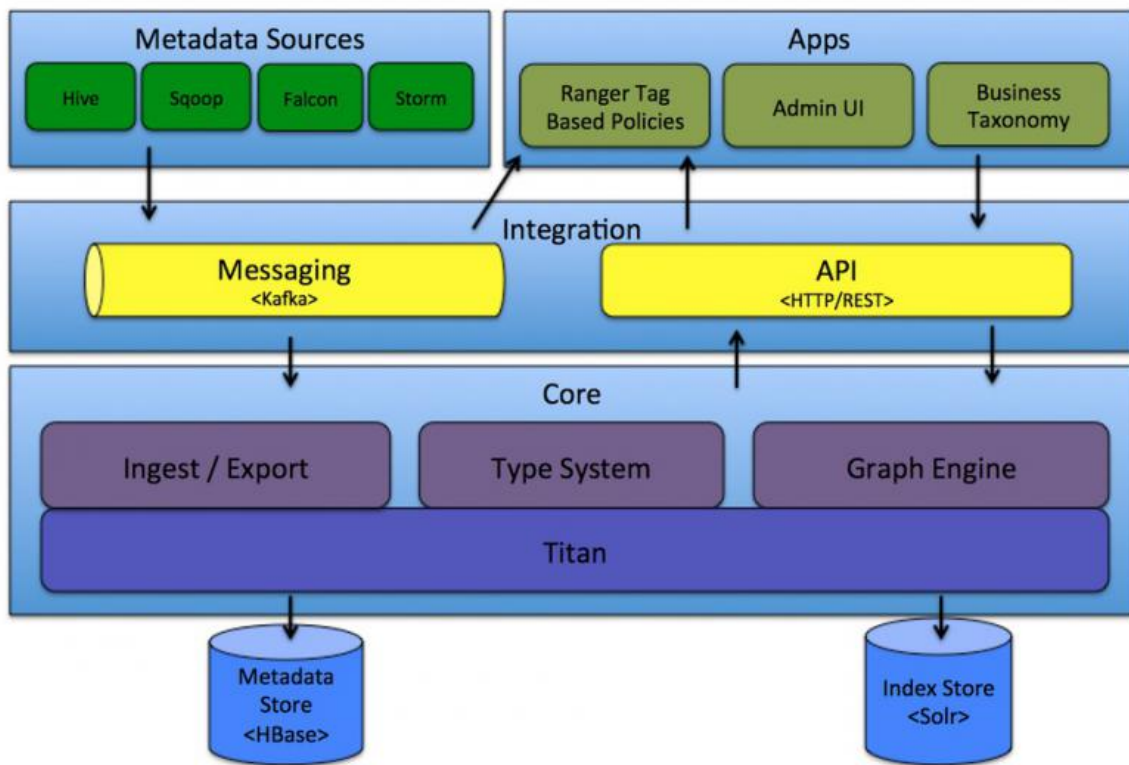
很长一段时间内，市面都没有成熟的大数据元数据管理解决方案。直到2015年，Hortonworks终于坐不住了，约了一众小伙伴公司倡议：咱们开始整个数据治理方案吧。然后，包含数据分类、集中策略引擎、数据血缘、安全和生命周期管理功能的Atlas应运而生。（类似的产品还有Linkedin 在2016年新开源的项目 whereHows）

第2节 Atlas简介

Atlas是Hadoop平台元数据框架；

Atlas是一组可扩展的核心基础治理服务，使企业能够有效，高效地满足Hadoop中的合规性要求，并能与整个企业数据生态系统集成；

Apache Atlas为组织提供了开放的元数据管理和治理功能，以建立数据资产的目录，对这些资产进行分类和治理，并为IT团队、数据分析团队提供围绕这些数据资产的协作功能。



Atlas由元数据的收集，存储和查询展示三部分核心组件组成。此外，还会有一个管理后台对整体元数据的采集流程以及元数据格式定义和服务的部署等各项内容进行配置管理。

Atlas包括以下组件：

- **Core。**Atlas功能核心组件，提供元数据的获取与导出(Ingets/Export)、类型系统(Type System)、元数据存储索引查询等核心功能
- **Integration。**Atlas对外集成模块。外部组件的元数据通过该模块将元数据交给Atlas管理
- **Metadata source。**Atlas支持的元数据数据源，以插件形式提供。当前支持从以下来源提取和管理元数据：
 - Hive
 - HBase
 - Sqoop
 - Kafka
 - Storm
- **Applications。**Atlas的上层应用，可以用来查询由Atlas管理的元数据类型和对象
- **Graph Engine（图计算引擎）。**Altas使用图模型管理元数据对象。图数据库提供了极大的灵活性，并能有效处理元数据对象之间的关系。除了管理图对象之外，图计算引擎还为元数据对象创建适当的索引，以便进行高效的访问。在Atlas 1.0 之前采用Titan作为图存储引擎，从1.0开始采用JanusGraph 作为图存储引擎。JanusGraph 底层又分为两块：
 - **Metadata Store。**采用 HBase 存储 Atlas 管理的元数据；
 - **Index Store。**采用Solr存储元数据的索引，便于高效搜索；

第3节 安装配置

重点讲解Atlas，不对Atlas的依赖组件做讲解，组件均采用单机模式安装。

编译才能安装。

3.1 安装依赖

- Maven 3.6.3 (完成)
- HBase 1.1.2 (不需要安装，需要软件包)
- Solr 5.5.1 (不需要安装，需要软件包)
- atlas 1.2.0 (需要编译)

官方只提供了源码，没有提供二进制的安装版本，因此Atlas需要编译。

3.2 安装步骤

1、准备软件包

apache-atlas-1.2.0-sources.tar.gz

solr-5.5.1.tgz

hbase-1.1.2.tar.gz

2、解压缩源码，修改配置

```
# 解压缩
cd /opt/lagou/software
tar zxvf apache-atlas-1.2.0-sources.tar.gz
cd apache-atlas-sources-1.2.0/

# 修改配置
vi pom.xml

# 修改
645         <npm-for-v2.version>3.10.8</npm-for-v2.version>
652         <hadoop.version>2.9.2</hadoop.version>
```

3、将HBase、Solr的包拷贝到对应的目录中

如果不拷贝这些包，就需要下载，下载 HBase 和 Solr 时速度很慢。这里提前下载完所需的这两个组件，拷贝到对应目录中。

```
cd /opt/lagou/software/apache-atlas-sources-1.2.0
```

```
# 创建目录
```

```
cd distro/
```

```
mkdir solr
```

```
mkdir hbase
```

```
# 拷贝软件包
```

```
cp /opt/lagou/software/solr-5.5.1.tgz ./solr/
```

```
cp /opt/lagou/software/hbase-1.1.2.tar.gz ./hbase/
```

4、maven设置阿里镜像

备注：重要，否则非常慢

```
cd $MAVEN_HOME/conf
```

```
# 在配置文件中添加
```

```
vi settings.xml
```

```
# 加在 158 行后
```

```
<mirror>
  <id>alimaven</id>
  <name>aliyun maven</name>
  <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
  <mirrorOf>central</mirrorOf>
</mirror>
```

5、Atlas编译

```
cd /opt/lagou/software/apache-atlas-sources-1.2.0
```

```
export MAVEN_OPTS="-Xms2g -Xmx2g"
```

```
mvn clean -DskipTests package -Pdist,embedded-hbase-solr
```

编译过程中大概要下载600M左右的jar，持续的时间比较长。

```
[INFO] atlas-client-common 1.2.0 ..... SUCCESS [ 1.617 s]
[INFO] atlas-client-v1 1.2.0 ..... SUCCESS [ 1.342 s]
[INFO] Apache Atlas Server API 1.2.0 ..... SUCCESS [ 1.076 s]
[INFO] Apache Atlas Notification 1.2.0 ..... SUCCESS [ 30.059 s]
[INFO] atlas-client-v2 1.2.0 ..... SUCCESS [ 0.958 s]
[INFO] Apache Atlas Graph Database Projects 1.2.0 ..... SUCCESS [ 0.100 s]
[INFO] Apache Atlas Graph Database API 1.2.0 ..... SUCCESS [ 6.426 s]
[INFO] Graph Database Common Code 1.2.0 ..... SUCCESS [ 1.789 s]
[INFO] Apache Atlas JanusGraph DB Impl 1.2.0 ..... SUCCESS [02:18 min]
[INFO] Apache Atlas Graph Database Implementation Dependencies 1.2.0 ..... SUCCESS [
[INFO] Shaded version of Apache hbase client 1.2.0 ..... SUCCESS [ 20.978 s]
[INFO] Shaded version of Apache hbase server 1.2.0 ..... SUCCESS [01:02 min]
[INFO] Apache Atlas Authorization 1.2.0 ..... SUCCESS [ 1.783 s]
[INFO] Apache Atlas Repository 1.2.0 ..... SUCCESS [01:17 min]
[INFO] Apache Atlas UI 1.2.0 ..... SUCCESS [04:44 min]
[INFO] Apache Atlas Web Application 1.2.0 ..... SUCCESS [02:21 min]
[INFO] Apache Atlas Documentation 1.2.0 ..... SUCCESS [ 47.682 s]
[INFO] Apache Atlas FileSystem Model 1.2.0 ..... SUCCESS [ 2.515 s]
[INFO] Apache Atlas Plugin Classloader 1.2.0 ..... SUCCESS [ 0.733 s]
[INFO] Apache Atlas Hive Bridge Shim 1.2.0 ..... SUCCESS [01:45 min]
[INFO] Apache Atlas Hive Bridge 1.2.0 ..... SUCCESS [ 17.256 s]
[INFO] Apache Atlas Falcon Bridge Shim 1.2.0 ..... SUCCESS [ 52.770 s]
[INFO] Apache Atlas Falcon Bridge 1.2.0 ..... SUCCESS [ 2.693 s]
[INFO] Apache Atlas Sqoop Bridge Shim 1.2.0 ..... SUCCESS [ 15.507 s]
[INFO] Apache Atlas Sqoop Bridge 1.2.0 ..... SUCCESS [ 2.748 s]
[INFO] Apache Atlas Storm Bridge Shim 1.2.0 ..... SUCCESS [ 28.811 s]
[INFO] Apache Atlas Storm Bridge 1.2.0 ..... SUCCESS [ 6.475 s]
[INFO] Apache Atlas Hbase Bridge Shim 1.2.0 ..... SUCCESS [ 1.359 s]
[INFO] Apache Atlas Hbase Bridge 1.2.0 ..... SUCCESS [ 41.697 s]
[INFO] Apache Atlas Kafka Bridge 1.2.0 ..... SUCCESS [ 1.641 s]
[INFO] Apache Atlas Distribution 1.2.0 ..... SUCCESS [04:00 min]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 35:19 min
```

编译完的软件位置：/opt/lagou/software/apache-atlas-sources-1.2.0/distro/target

编译完的软件：apache-atlas-1.2.0-bin.tar.gz

6、Atlas安装

```
cd /opt/lagou/software/apache-atlas-sources-1.2.0/distro/target
```

解压缩

```
tar zxvf apache-atlas-1.2.0-bin.tar.gz
```

```
mv apache-atlas-1.2.0/ /opt/lagou/servers/atlas-1.2.0
```

修改 /etc/profile，设置环境变量 ATLAS_HOME

启动服务(第一次启动服务的时间比较长)

```
cd $ATLAS_HOME/bin
```

```
./atlas_start.py
```

检查后台进程 (1个atlas、2个HBase、1个solr后台进程)

```
ps -ef | grep atlas
```

```
/opt/lagou/servers/atlas-1.2.0/server/webapp/atlas
```

```
hbase-daemon.sh
```

```
org.apache.hadoop.hbase.master.HMaster
```

```
/opt/lagou/servers/atlas-1.2.0/solr/server
```

停止服务

```
./atlas_stop.py
```

检查 solr 的状态:

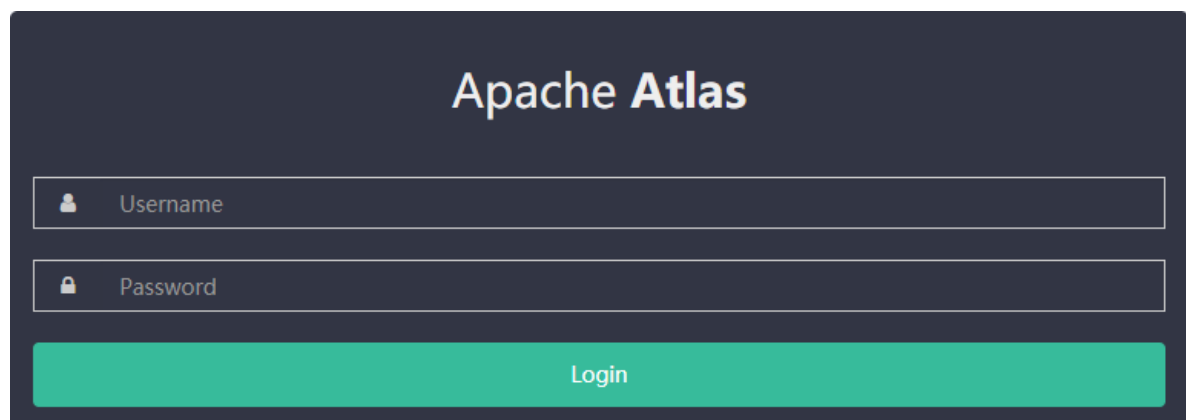
```
cd /opt/lagou/servers/atlas-1.2.0/solr/bin
./solr status

solr process 25038 running on port 9838
{
  "solr_home":"/opt/lagou/servers/atlas-1.2.0/solr/server/solr",
  "version":"5.5.1 c08f17bca0d9cbf516874d13d221ab100e5b7d58 - anshum -
2016-04-30 13:28:18",
  "startTime":"2020-07-31T11:58:45.638Z",
  "uptime":"0 days, 14 hours, 55 minutes, 11 seconds",
  "memory":"54.8 MB (%11.2) of 490.7 MB",
  "cloud":{
    "zookeeper":"localhost:2181",
    "liveNodes":"1",
    "collections":"3"}}}
```

检查 zk 状态:

```
echo stat|nc localhost 2181
```

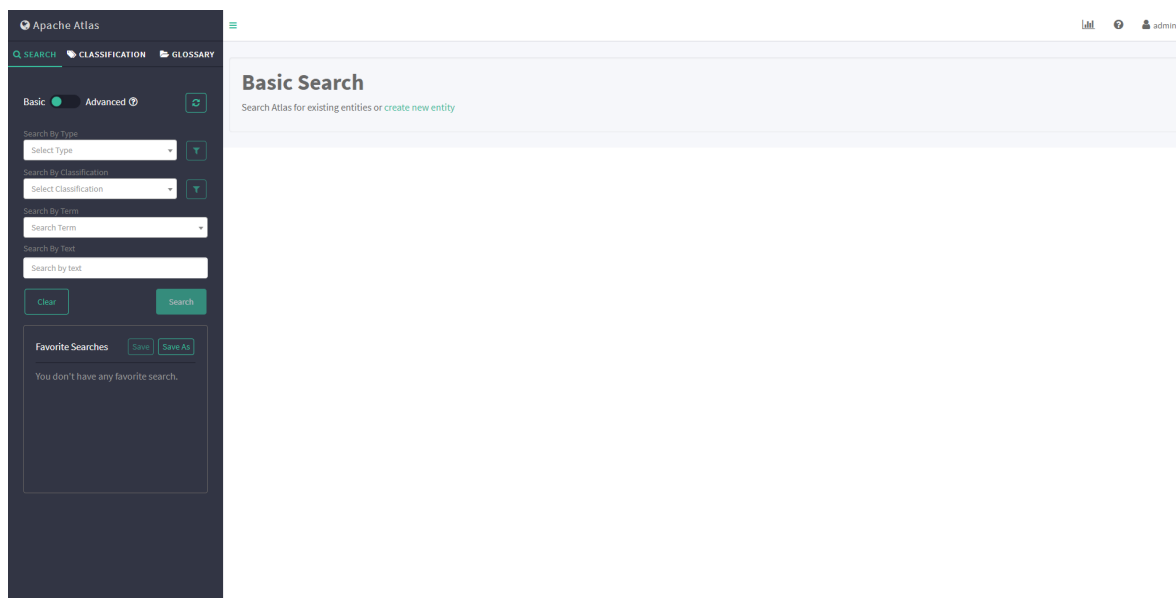
Web服务: <http://linux122:21000/login.jsp>

The image shows the Apache Atlas login page. It has a dark blue header with the 'Apache Atlas' logo in white. Below the header, there are two input fields: 'Username' with a person icon and 'Password' with a lock icon. At the bottom, there is a large green button labeled 'Login'.

用户名 / 口令: admin / admin

账号的信息存储在文件 `conf/users-credentials.properties` 中。其中 Password 通过如下方式产生 sha256sum 摘要信息:

```
echo -n "admin" | sha256sum
```



第4节 Hive血缘关系导入

1、配置HIVE_HOME环境变量；将 \$ATLAS_HOME/conf/atlas-application.properties 拷贝到 \$HIVE_HOME/conf 目录下

```
ln -s $ATLAS_HOME/conf/atlas-application.properties $HIVE_HOME/conf/atlas-application.properties
```

2、拷贝jar包

```
# $ATLAS_HOME/server/webapp/atlas/WEB-INF/lib/ 目录下的3个jar，拷贝到
$ATLAS_HOME/hook/hive/atlas-hive-plugin-impl/ 目录下
jackson-jaxrs-base-2.9.9.jar
jackson-jaxrs-json-provider-2.9.9.jar
jackson-module-jaxb-annotations-2.9.9.jar

ln -s $ATLAS_HOME/server/webapp/atlas/WEB-INF/lib/jackson-jaxrs-base-
2.9.9.jar $ATLAS_HOME/hook/hive/atlas-hive-plugin-impl/jackson-jaxrs-base-
2.9.9.jar

ln -s $ATLAS_HOME/server/webapp/atlas/WEB-INF/lib/jackson-jaxrs-json-
provider-2.9.9.jar $ATLAS_HOME/hook/hive/atlas-hive-plugin-impl/jackson-
jaxrs-json-provider-2.9.9.jar

ln -s $ATLAS_HOME/server/webapp/atlas/WEB-INF/lib/jackson-module-jaxb-
annotations-2.9.9.jar $ATLAS_HOME/hook/hive/atlas-hive-plugin-
impl/jackson-module-jaxb-annotations-2.9.9.jar
```

3、修改Hive的配置

hive-site.xml增加 hook

```
<property>
  <name>hive.exec.post.hooks</name>
  <value>org.apache.atlas.hive.hook.HiveHook</value>
</property>
```

\$HIVE_HOME/conf/hive-env.sh中添加HIVE_AUX_JARS_PATH变量

```
export HIVE_AUX_JARS_PATH=/opt/lagou/servers/atlas-1.2.0/hook/hive
```

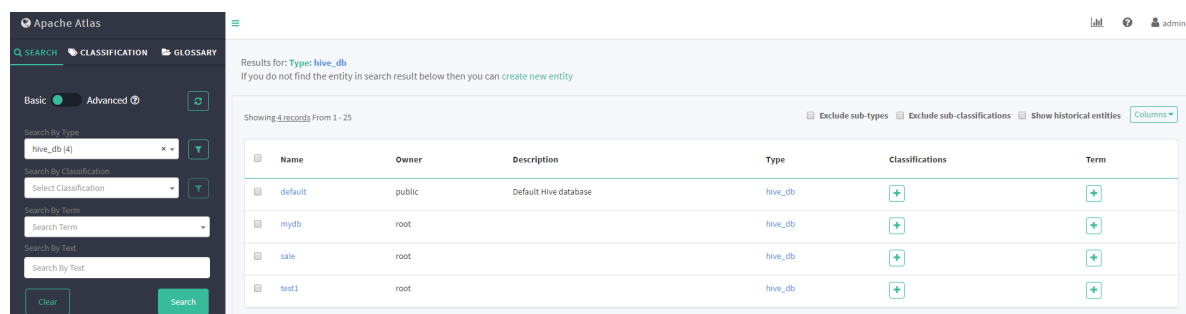
4、批量导入hive数据

备注：Hive能正常启动；在执行的过程中需要用户名/口令：admin/admin

```
import-hive.sh
```

成功导出可以看见最后的提示信息：Hive Meta Data imported successfully!!!

在浏览器中可以看见：Search 中的选项有变化



The screenshot shows the Apache Atlas web interface. On the left is a sidebar with search filters. The main area displays search results for 'hive_db'. The results table has columns: Name, Owner, Description, Type, Classifications, and Term. There are four rows of results, each with a '+' icon in the Classifications column.

Name	Owner	Description	Type	Classifications	Term
default	public	Default Hive database	hive_db	+	+
mydb	root		hive_db	+	+
sale	root		hive_db	+	+
test1	root		hive_db	+	+

Hive hook 可捕获以下操作：

- create database
- create table/view, create table as select
- load, import, export
- DMLs (insert)
- alter database
- alter table
- alter view

第5节 与电商业务集成

开发（建库、建表） => 导入数据 => 执行Hive脚本

导入Hive的血缘关系

电商业务建表语句（可省略）：

```
-- 创建DataBases;
CREATE DATABASE ODS;
CREATE DATABASE DIM;
CREATE DATABASE DWD;
CREATE DATABASE DWS;
CREATE DATABASE ADS;

-- 创建ODS表
DROP TABLE IF EXISTS `ods.ods_trade_orders`;
CREATE EXTERNAL TABLE `ods.ods_trade_orders`(
  `orderid` int,
  `orderno` string,
  `userid` bigint,
  `status` tinyint,
  `productmoney` decimal(10,0),
  `totalmoney` decimal(10,0),
  `paymethod` tinyint,
  `ispay` tinyint,
  `areaid` int,
  `tradesrc` tinyint,
  `tradetype` int,
  `isrefund` tinyint,
  `dataflag` tinyint,
  `createtime` string,
  `paytime` string,
  `modifiedtime` string)
COMMENT '订单表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/orders/';

DROP TABLE IF EXISTS `ods.ods_trade_order_product`;
CREATE EXTERNAL TABLE `ods.ods_trade_order_product`(
  `id` string,
  `orderid` decimal(10,2),
  `productid` string,
  `productnum` string,
  `productprice` string,
  `money` string,
  `extra` string,
  `createtime` string)
COMMENT '订单明细表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/order_product/';

DROP TABLE IF EXISTS `ods.ods_trade_product_info`;
CREATE EXTERNAL TABLE `ods.ods_trade_product_info`(
  `productid` bigint,
  `productname` string,
```



```

    `shopid` string,
    `price` decimal(10,0),
    `issale` tinyint,
    `status` tinyint,
    `categoryid` string,
    `createtime` string,
    `modifytime` string)
COMMENT '产品信息表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/product_info/';

DROP TABLE IF EXISTS `ods.ods_trade_product_category`;
CREATE EXTERNAL TABLE `ods.ods_trade_product_category`(
    `catid` int,
    `parentid` int,
    `catname` string,
    `isshow` tinyint,
    `sortnum` int,
    `isdel` tinyint,
    `createtime` string,
    `level` tinyint)
COMMENT '产品分类表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/product_category';

DROP TABLE IF EXISTS `ods.ods_trade_shops`;
CREATE EXTERNAL TABLE `ods.ods_trade_shops`(
    `shopid` int,
    `userid` int,
    `areaaid` int,
    `shopname` string,
    `shoplevel` tinyint,
    `status` tinyint,
    `createtime` string,
    `modifytime` string)
COMMENT '商家店铺表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/shops';

DROP TABLE IF EXISTS `ods.ods_trade_shop_admin_org`;
CREATE EXTERNAL TABLE `ods.ods_trade_shop_admin_org`(
    `id` int,
    `parentid` int,
    `orgname` string,
    `orglevel` tinyint,
    `isdelete` tinyint,
    `createtime` string,
    `updatetime` string,
    `isshow` tinyint,

```

```

    `orgType` tinyint)
COMMENT '商家地域组织表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/shop_org/';

DROP TABLE IF EXISTS `ods.ods_trade_payments`;
CREATE EXTERNAL TABLE `ods.ods_trade_payments`(
    `id` string,
    `paymethod` string,
    `payname` string,
    `description` string,
    `payorder` int,
    `online` tinyint)
COMMENT '支付方式表'
PARTITIONED BY (`dt` string)
row format delimited fields terminated by ','
location '/user/data/trade.db/payments/';

-- 创建DIM表
DROP TABLE IF EXISTS dim.dim_trade_product_cat;
create table if not exists dim.dim_trade_product_cat(
    firstId int,                -- 一级商品分类id
    firstName string,           -- 一级商品分类名称
    secondId int,               -- 二级商品分类Id
    secondName string,          -- 二级商品分类名称
    thirdId int,                -- 三级商品分类id
    thirdName string            -- 三级商品分类名称
)
partitioned by (dt string)
STORED AS PARQUET;

drop table if exists dim.dim_trade_shops_org;
create table dim.dim_trade_shops_org(
    shopid int,
    shopName string,
    cityId int,
    cityName string ,
    regionId int ,
    regionName string
)
partitioned by (dt string)
STORED AS PARQUET;

drop table if exists dim.dim_trade_payment;
create table if not exists dim.dim_trade_payment(
    paymentId string,           -- 支付方式id
    paymentName string          -- 支付方式名称
)
partitioned by (dt string)
STORED AS PARQUET;

```

```

drop table if exists dim.dim_trade_product_info;
create table dim.dim_trade_product_info(
    `productId` bigint,
    `productName` string,
    `shopId` string,
    `price` decimal,
    `issale` tinyint,
    `status` tinyint,
    `categoryId` string,
    `createTime` string,
    `modifyTime` string,
    `start_dt` string,
    `end_dt` string
) COMMENT '产品表'
STORED AS PARQUET;

```

-- 创建DWD表

-- 订单事实表(拉链表)

```

DROP TABLE IF EXISTS dwd.dwd_trade_orders;
create table dwd.dwd_trade_orders(
    `orderId` int,
    `orderNo` string,
    `userId` bigint,
    `status` tinyint,
    `productMoney` decimal,
    `totalMoney` decimal,
    `payMethod` tinyint,
    `isPay` tinyint,
    `areaId` int,
    `tradeSrc` tinyint,
    `tradeType` int,
    `isRefund` tinyint,
    `dataFlag` tinyint,
    `createTime` string,
    `payTime` string,
    `modifiedTime` string,
    `start_date` string,
    `end_date` string
) COMMENT '订单事实拉链表'
partitioned by (dt string)
STORED AS PARQUET;

```

-- 创建DWS表

```

DROP TABLE IF EXISTS dws.dws_trade_orders;
create table if not exists dws.dws_trade_orders(
   orderid string, -- 订单id
    cat_3rd_id string, -- 商品三级分类id
    shopid string, -- 店铺id
    paymethod tinyint, -- 支付方式
    productsnum bigint, -- 商品数量
    paymoney double, -- 订单商品明细金额
    paytime string -- 订单时间

```

```

)
partitioned by (dt string)
STORED AS PARQUET;

-- 订单明细表宽表
DROP TABLE IF EXISTS dws.dws_trade_orders_w;
create table if not exists dws.dws_trade_orders_w(
    orderid string,          -- 订单id
    cat_3rd_id string,       -- 商品三级分类id
    thirdname string,        -- 商品三级分类名称
    secondname string,       -- 商品二级分类名称
    firstname string,        -- 商品一级分类名称
    shopid string,           -- 店铺id
    shopname string,         -- 店铺名
    regionname string,       -- 店铺所在大区
    cityname string,         -- 店铺所在城市
    paymethod tinyint,       -- 支付方式
    productsnum bigint,      -- 商品数量
    paymoney double,         -- 订单明细金额
    paytime string          -- 订单时间
)
partitioned by (dt string)
STORED AS PARQUET;

-- 创建ADS表
-- ADS层订单分析表
DROP TABLE IF EXISTS ads.ads_trade_order_analysis;
create table if not exists ads.ads_trade_order_analysis(
    areatype string,         -- 区域范围：区域类型（全国、大区、城市）
    regionname string,       -- 区域名称
    cityname string,         -- 城市名称
    categorytype string,     -- 商品分类类型（一级、二级）
    category1 string,        -- 商品一级分类名称
    category2 string,        -- 商品二级分类名称
    totalcount bigint,       -- 订单数量
    total_productnum bigint,  -- 商品数量
    totalmoney double        -- 支付金额
)
partitioned by (dt string)
row format delimited fields terminated by ',';

```

使用Sqoop加载数据（可省略）：

```

sqoop import \
--connect jdbc:mysql://linux123:3306/ebiz \
--username hive \
--password 12345678 \
--target-dir /user/data/trade.db/orders/dt=2020-07-21/ \
--table lagou_trade_orders \
--delete-target-dir \

```

```

--num-mappers 1 \
--fields-terminated-by ','

sqoop import \
--connect jdbc:mysql://linux123:3306/ebiz \
--username hive \
--password 12345678 \
--target-dir /user/data/trade.db/payments/dt=2020-07-21/ \
--table lagou_payments \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by ','

sqoop import \
--connect jdbc:mysql://linux123:3306/ebiz \
--username hive \
--password 12345678 \
--target-dir /user/data/trade.db/product_category/dt=2020-07-21/ \
--table lagou_product_category \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by ','

sqoop import \
--connect jdbc:mysql://linux123:3306/ebiz \
--username hive \
--password 12345678 \
--target-dir /user/data/trade.db/product_info/dt=2020-07-21/ \
--table lagou_product_info \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by ','

sqoop import \
--connect jdbc:mysql://linux123:3306/ebiz \
--username hive \
--password 12345678 \
--target-dir /user/data/trade.db/order_product/dt=2020-07-21/ \
--table lagou_order_product \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by ','

sqoop import \
--connect jdbc:mysql://linux123:3306/ebiz \
--username hive \
--password 12345678 \
--target-dir /user/data/trade.db/shop_org/dt=2020-07-21/ \
--table lagou_shop_admin_org \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by ','

```

```

sqoop import \
--connect jdbc:mysql://linux123:3306/ebiz \
--username hive \
--password 12345678 \
--target-dir /user/data/trade.db/shops/dt=2020-07-21/ \
--table lagou_shops \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by ','

alter table ods.ods_trade_orders add partition(dt='2020-07-21');
alter table ods.ods_trade_payments add partition(dt='2020-07-21');
alter table ods.ods_trade_product_category add partition(dt='2020-07-21');
alter table ods.ods_trade_product_info add partition(dt='2020-07-21');
alter table ods.ods_trade_order_product add partition(dt='2020-07-21');
alter table ods.ods_trade_shop_admin_org add partition(dt='2020-07-21');
alter table ods.ods_trade_shops add partition(dt='2020-07-21');

```

电商业务脚本(省略了ODS层数据加载):

```

# 加载DIM层数据
sh /data/lagoudw/script/trade/dim_load_product_cat.sh 2020-07-21
sh /data/lagoudw/script/trade/dim_load_shop_org.sh 2020-07-21
sh /data/lagoudw/script/trade/dim_load_payment.sh 2020-07-21
sh /data/lagoudw/script/trade/dim_load_product_info.sh 2020-07-21

# 加载DWD层数据
sh /data/lagoudw/script/trade/dwd_load_trade_orders.sh 2020-07-21

# 加载DWS层数据
sh /data/lagoudw/script/trade/dws_load_trade_orders.sh 2020-07-21

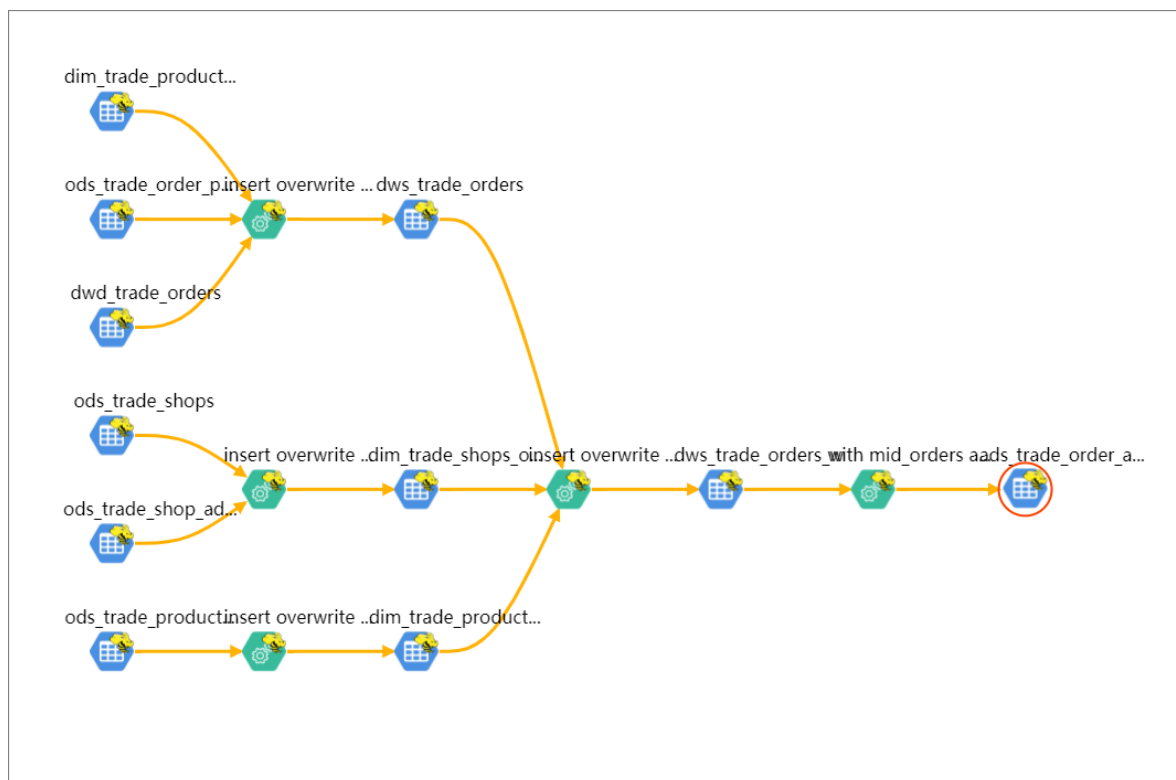
# 加载ADS层数据
sh /data/lagoudw/script/trade/ads_load_trade_order_analysis.sh 2020-07-21

```

创建 Classification: order_analysis

创建Glossary: ODS层 => 电商业务

查看血缘关系 ads_trade_order_analysis:



第四部分 数据质量监控工具Griffin（扩展）

第1节 为什么要做数据质量监控

garbage in garbage out

1、数据不一致

企业早期没有进行统一规划设计，大部分信息系统是逐步迭代建设的，系统建设时间长短各异，各系统数据标准也不同。企业业务系统更关注业务层面，各个业务系统均有不同的侧重点，各类数据的属性信息设置和要求不统一。另外，由于各系统的相互独立使用，无法及时同步更新相关信息等各种原因造成各系统间的数据不一致，严重影响了各系统间的数据交互和统一识别，基础数据难以共享利用，数据的深层价值也难以体现。

2、数据不完整

由于企业信息系统的孤立使用，各个业务系统或模块按照各自的需要录入数据，没有统一的录入工具和数据出口，业务系统不需要的信息就不录，造成同样的数据在不同的系统有不同的属性信息，数据完整性无法得到保障。

3、数据不合规

没有统一的数据管理平台和数据源头，数据全生命周期管理不完整，同时企业各信息系统的数据录入环节过于简单且手工参与较多，就数据本身而言，缺少是否重复、合法、对错等校验环节，导致各个信息系统的数据不够准确，格式混乱，各类数据难以集成和统一，没有质量控制导致海量数据因质量过低而难以被利用，且没有相应的数据管理流程。

4、数据不可控

海量数据多头管理，缺少专门对数据管理进行监督和控制的组织。企业各单位和部门关注数据的角度不一样，缺少一个组织从全局的视角对数据进行管理，导致无法建立统一的数据管理标准、流程等，相应的数据管理制度、办法等无法得到落实。同时，企业基础数据质量考核体系也尚未建立，无法保障一系列数据标准、规范、制度、流程得到长效执行。

5、数据冗余

各个信息系统针对数据的标准规范不一、编码规则不一、校验标准不一，且部分业务系统针对数据的验证标准严重缺失，造成了企业顶层视角的数据出现“一物多码”、“一码多物”等现象。

第2节 数据质量监控方法

1、设计思路

数据质量监控的设计要分为4个模块：数据，规则，告警和反馈

①数据：需要被监控的数据，可能存放在不同的存储引擎中

②规则：值如何设计发现异常的规则，一般而言主要是数值的异常和环比等异常监控方式。也会有一些通过算法来发掘异常数据的方法

③告警：告警是指发告警的动作，这里可以通过微信消息，电话或者短信，邮件

④反馈：反馈是指对告警内容的反馈，比如说收到的告警内容，要有人员回应该告警消息是否是真的异常，是否需要忽略该异常，是否已经处理了该异常。有了反馈机制，整个数据监控才能形成闭环

2、技术方案

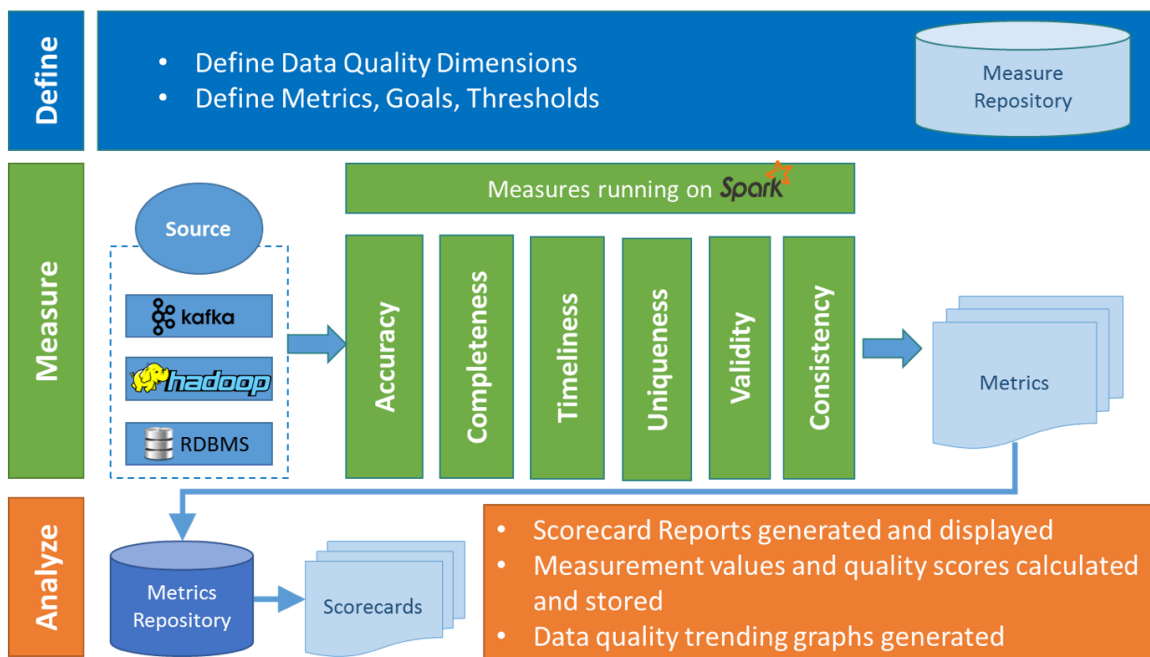
- 最开始可以先关注核心要监控的内容，比如说准确性，那么就对核心的一些指标做监控即可
- 监控平台尽量不要做太复杂的规则逻辑，尽量只对结果数据进行监控。比如要监控日质量是否波动过大，那么把该计算流程提前，先计算好结果表，最后监控平台只监控结果表是否异常即可
- 多数据源。多数据源的监控有两种方式：针对每个数据源定制实现一部分计算逻辑，也可以通过额外的任务将多数据源中的数据结果通过任务写入一个数据源中，再对该数据源进行监控
- 实时数据监控：区别在于扫描周期的不同，因此在设计的时候可以先以离线为主，但是尽量预留好实时监控的设计

第3节 Griffin架构

Apache Griffin是一个开源的大数据数据质量解决方案，它支持批处理和流模式两种数据质量检测方式，可以从不同维度（如离线任务执行完毕后检查源端和目标端的数据数量是否一致、源表的数据空值数量等）度量数据资产，从而提升数据的准确度、可信度。

Griffin主要分为Define、Measure和Analyze三个部分，如下图所示：

Apache Griffin Architecture



各部分的职责如下：

- Define：主要负责定义数据质量统计的维度，比如数据质量统计的时间跨度、统计的目标（源端和目标端的数据数量是否一致，数据源里某一字段的非空的数量、不重复值的数量、最大值、最小值、top5的值数量等）
- Measure：主要负责执行统计任务，生成统计结果
- Analyze：主要负责保存与展示统计结果

第4节 编译安装

4.1 相关依赖

重点讲解 Griffin，不对依赖组件做过多讲解，所有组件均采用单机模式安装。

- JDK (1.8 or later versions)
- MySQL(version 5.6及以上)
- Hadoop (2.6.0 or later)
- Hive (version 2.x)
- Maven
- Spark (version 2.2.1)
- Livy (livy-0.5.0-incubating)
- ElasticSearch (5.0 or later versions)

备注：

- Spark：计算批量、实时指标

- Livy: 为服务提供 RESTful API 调用 Apache Spark
- ElasticSearch: 存储指标数据
- MySQL: 服务元数据

4.2 Spark安装

1、解压缩，设置环境变量 \$SPARK_HOME

```
tar zxvf spark-2.2.1-bin-hadoop2.7.tgz
mv spark-2.2.1-bin-hadoop2.7/ /opt/lagou/servers/spark-2.2.1/

# 设置环境变量
vi /etc/profile

export SPARK_HOME=/opt/lagou/servers/spark-2.2.1/
export PATH=$PATH:$SPARK_HOME/bin

source /etc/profile
```

2、修改配置文件 \$SPARK_HOME/conf/spark-defaults.conf

```
spark.master                yarn
spark.eventLog.enabled      true
spark.eventLog.dir          hdfs://linux121:9000/spark/logs
spark.serializer            org.apache.spark.serializer.KryoSerializer
spark.yarn.jars              hdfs://linux121:9000/spark/spark_2.2.1_jars/*
```

备注：上面的路径要创建

拷贝 MySQL 驱动

```
cp $HIVE_HOME/lib/mysql-connector-java-5.1.46.jar $SPARK_HOME/jars/
```

将 Spark 的 jar 包上传到 hdfs://hadoop1:9000/spark/spark_2.2.1_jars/

```
hdfs dfs -mkdir -p /spark/logs
hdfs dfs -mkdir -p /spark/spark_2.2.1_jars/
hdfs dfs -put /opt/lagou/servers/spark-2.2.1/jars/*.jar
/spark/spark_2.2.1_jars/
```

3、修改配置文件spark-env.sh

```
export JAVA_HOME=/opt/lagou/servers/jdk1.8.0_231/
export HADOOP_HOME=/opt/lagou/servers/hadoop-2.9.2/
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

4、yarn-site.xml 添加配置

```
<property>
  <name>yarn.nodemanager.vmem-check-enabled</name>
  <value>>false</value>
</property>
```

yarn.nodemanager.vmem-check-enabled：是否检查虚拟内存。

修改所有节点，并重启yarn服务。

不添加该配置启动spark-shell，有如下错误：Yarn application has already ended! It might have been killed or unable to launch application master.

5、测试spark

spark-shell

```
// /wcinput/wc.txt : HDFS上的文件
val lines = sc.textFile("/wcinput/wc.txt")
lines.flatMap(_.split(" ")).map((_,1)).reduceByKey(_+_).collect()
```

4.3 Livy安装

1、解压缩，设置环境变量 \$LIVY_HOME

```
unzip livy-0.5.0-incubating-bin.zip
mv livy-0.5.0-incubating-bin/ ../servers/livy-0.5.0

# 设置环境变量
vi /etc/profile

export LIVY_HOME=/opt/lagou/servers/livy-0.5.0
export PATH=$PATH:$LIVY_HOME/bin

source /etc/profile
```

2、修改配置文件 conf/livy.conf

```
livy.server.host = 127.0.0.1
livy.spark.master = yarn
livy.spark.deployMode = cluster
livy.repl.enable-hive-context = true
```

3、修改配置文件 conf/livy-env.sh

```
export SPARK_HOME=/opt/lagou/servers/spark-2.2.1
export HADOOP_HOME=/opt/lagou/servers/hadoop-2.9.2/
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

4、启动服务

```
cd /opt/lagou/servers/livy-0.5.0
mkdir logs

nohup bin/livy-server &
```

4.4 ES安装

1、解压缩

```
tar zxvf elasticsearch-5.6.0.tar.gz
mv elasticsearch-5.6.0/ ../software/
```

2、创建 elasticsearch用户组 及 elasticsearch 用户。不能使用root用户启动ES程序，需要创建单独的用户去启动ES 服务；

```
# 创建用户组
groupadd elasticsearch

# 创建用户
useradd elasticsearch -g elasticsearch

# 修改安装目录的宿主
chown -R elasticsearch:elasticsearch elasticsearch-5.6.0/
```

3、修改linux系统文件 /etc/security/limits.conf

```
elasticsearch hard nofile 1000000
elasticsearch soft nofile 1000000
* soft nproc 4096
* hard nproc 4096
```

4、修改系统文件 /etc/sysctl.conf

```
# 文件末尾增加：
vm.max_map_count=262144

# 执行以下命令，修改才能生效
sysctl -p
```

5、修改es配置文件

/opt/lagou/servers/elasticsearch-5.6.0/config/elasticsearch.yml

```
network.host: 0.0.0.0
```

/opt/lagou/servers/elasticsearch-5.6.0/config/jvm.options

jvm内存的分配，原来都是2g，修改为1g

```
-Xms1g
-Xmx1g
```

6、启动ES服务

```
# 到ES安装目录下，执行命令(-d表示后台启动)
su elasticsearch
cd /opt/lagou/servers/elasticsearch-5.6.0/
bin/elasticsearch -d
```

在浏览器中检查: <http://linux122:9200/>

```
{
  "name" : "14TSMYw",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "ZSy0cL5hTtiuEL_g0nDzUQ",
  "version" : {
    "number" : "5.6.0",
    "build_hash" : "781a835",
    "build_date" : "2017-09-07T03:09:58.087Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.0"
  },
  "tagline" : "You Know, for Search"
}
```

7、在ES里创建griffin索引

```
# linux122 为 ES 服务所在节点
curl -XPUT http://linux122:9200/griffin -d '
{
  "aliases": {},
  "mappings": {
    "accuracy": {
      "properties": {
        "name": {
          "fields": {
            "keyword": {
              "ignore_above": 256,
              "type": "keyword"
            }
          },
          "type": "text"
        },
        "tmst": {
          "type": "date"
        }
      }
    }
  },
  "settings": {
    "index": {
      "number_of_replicas": "2",
      "number_of_shards": "5"
    }
  }
}
```

4.5 Griffin编译准备

1、软件解压缩

```
cd /opt/lagou/software
unzip griffin-griffin-0.5.0.zip
mv griffin-griffin-0.5.0/ ../servers/griffin-0.5.0/
cd griffin-0.5.0
```

2、在MySQL中创建数据库quartz，并初始化

/opt/lagou/servers/griffin-0.5.0/service/src/main/resources/Init_quartz_mysql_innodb.sql

备注：要做简单的修改，主要是增加 use quartz；

```
# mysql中执行创建数据库
create database quartz;

# 命令行执行，创建表
mysql -uhive -p12345678 < Init_quartz_mysql_innodb.sql
```

3、Hadoop和Hive

在HDFS上创建/spark/spark_conf目录，并将Hive的配置文件hive-site.xml上传到该目录下

```
hdfs dfs -mkdir -p /spark/spark_conf
hdfs dfs -put $HIVE_HOME/conf/hive-site.xml /spark/spark_conf/
```

备注：将安装 griffin 所在节点上的 hive-site.xml 文件，上传到 HDFS 对应目录中；

4、确保设置以下环境变量(/etc/profile)

```
export JAVA_HOME=/opt/lagou/servers/hadoop-2.9.2
export SPARK_HOME=/opt/lagou/servers/spark-2.2.1/
export LIVY_HOME=/opt/lagou/servers/livy-0.5.0
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

4.6 Griffin编译

1、service/pom.xml文件配置

编辑 service/pom.xml（约113-117行），增加MySQL JDBC 依赖（即删除注释）：

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>${mysql.java.version}</version>
</dependency>
```

2、修改配置文件 service/src/main/resources/application. Properties

```
server.port = 9876

spring.application.name=griffin_service
spring.datasource.url=jdbc:mysql://linux123:3306/quartz?
autoReconnect=true&useSSL=false
spring.datasource.username=hive
spring.datasource.password=12345678
spring.jpa.generate-ddl=true
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.show-sql=true

# Hive metastore
hive.metastore.uris=thrift://linux123:9083
hive.metastore.dbname=hivemetadata
hive.hmshandler.retry.attempts=15
hive.hmshandler.retry.interval=2000ms

# Hive cache time
cache.evict.hive.fixedRate.in.milliseconds=900000

# Kafka schema registry
kafka.schema.registry.url=http://localhost:8081

# Update job instance state at regular intervals
jobInstance.fixedDelay.in.milliseconds=60000

# Expired time of job instance which is 7 days that is 604800000
milliseconds.Time unit only supports milliseconds
jobInstance.expired.milliseconds=604800000

# schedule predicate job every 5 minutes and repeat 12 times at most
#interval time unit s:second m:minute h:hour d:day,only support these four
units
predicate.job.interval=5m
predicate.job.repeat.count=12

# external properties directory location
external.config.location=

# external BATCH or STREAMING env
external.env.location=

# login strategy ("default" or "ldap")
login.strategy=default

# ldap
ldap.url=ldap://hostname:port
ldap.email=@example.com
```



```

ldap.searchBase=DC=org,DC=example
ldap.searchPattern=(sAMAccountName={0})

# hdfs default name
fs.defaultFS=

# elasticsearch
elasticsearch.host=linux122
elasticsearch.port=9200
elasticsearch.scheme=http

# elasticsearch.user = user
# elasticsearch.password = password

# livy
livy.uri=http://localhost:8998/batches
livy.need.queue=false
livy.task.max.concurrent.count=20
livy.task.submit.interval.second=3
livy.task.appId.retry.count=3

# yarn url
yarn.uri=http://linux123:8088

# griffin event listener
internal.event.listeners=GriffinJobEventHook

```

备注：

- 默认端口是8080，为避免和spark端口冲突，这里端口修改为9876
- 需要启动Hive的 metastore 服务
- 如果Griffin、MySQL没有安装在同一节点，请确认用户有权限能够远程登录

3、修改配置文件 service/src/main/resources/quartz. Properties

```

# 将第26行修改为以下内容：
org.quartz.jobStore.driverDelegateClass=org.quartz.impl.jdbcjobstore.StdJDBCDelegate

```

4、修改配置文件 service/src/main/resources/sparkProperties.json

sparkProperties.json 在测试环境下使用：

```

{
  "file": "hdfs:///griffin/griffin-measure.jar",
  "className": "org.apache.griffin.measure.Application",
  "name": "griffin",
  "queue": "default",
  "numExecutors": 2,

```

```

    "executorCores": 1,
    "driverMemory": "1g",
    "executorMemory": "1g",
    "conf": {
        "spark.yarn.dist.files": "hdfs:///spark/spark_conf/hive-site.xml"
    },
    "files": [
    ]
}

```

备注：修改第11行："spark.yarn.dist.files": "hdfs:///spark/spark_conf/hive-site.xml"

sparkProperties.json 在生产环境中根据实际资源配置进行修改【生产环境】

```

{
    "file": "hdfs:///griffin/griffin-measure.jar",
    "className": "org.apache.griffin.measure.Application",
    "name": "griffin",
    "queue": "default",
    "numExecutors": 8,
    "executorCores": 2,
    "driverMemory": "4g",
    "executorMemory": "5g",
    "conf": {
        "spark.yarn.dist.files": "hdfs:///spark/spark_conf/hive-site.xml"
    },
    "files": [
    ]
}

```

5、修改配置文件 service/src/main/resources/env/env_batch.json

```

{
    "spark": {
        "log.level": "WARN"
    },
    "sinks": [
        {
            "type": "CONSOLE",
            "config": {
                "max.log.lines": 10
            }
        },
        {
            "type": "HDFS",
            "config": {
                "path": "hdfs:///griffin/persist",
                "max.persist.lines": 10000,
            }
        }
    ]
}

```

```

        "max.lines.per.file": 10000
    }
},
{
    "type": "ELASTICSEARCH",
    "config": {
        "method": "post",
        "api": "http://liunx122:9200/griffin/accuracy",
        "connection.timeout": "1m",
        "retry": 10
    }
}
],
"griffin.checkpoint": []
}

```

备注：仅修改第24行

6、编译

```

cd /opt/lagou/software/griffin-0.5.0
mvn -Dmaven.test.skip=true clean install

```

备注：

- 编译过程中需要下载500M+左右的jar，要将Maven的源设置到阿里
- 如果修改了前面的配置文件，需要重新编译

7、修改文件

编译报错：

```

[ERROR] ERROR in /opt/lagou/servers/griffin-0.5.0/ui/angular/node_modules/@types/jquery/JQuery.d.ts (4137,26): Cannot find name 'SVGElementTagNameMap'.
[ERROR] ERROR in /opt/lagou/servers/griffin-0.5.0/ui/angular/node_modules/@types/jquery/JQuery.d.ts (4137,89): Cannot find name 'SVGElementTagNameMap'.

```

这个文件在编译之前是没有的

/opt/lagou/servers/griffin-0.5.0/ui/angular/node_modules/@types/jquery/JQuery.d.ts

删除 4137 行

```

find<K extends keyof SVGElementTagNameMap>(selector_element: K | JQuery<K>): JQuery<SVGElementTagNameMap[K]>;

```

8、再次编译

```
cd /opt/lagou/servers/griffin-0.5.0
mvn -Dmaven.test.skip=true clean install
```

9、jar拷贝

编译完成后，会在service和measure模块的target目录下分别看到 service-0.5.0.jar 和 measure-0.5.0.jar 两个jar，将这两个jar分别拷贝到服务器目录下。

```
# 将 service-0.5.0.jar 拷贝到 /opt/lagou/servers/griffin-0.5.0/
cd /opt/lagou/servers/griffin-0.5.0/service/target
cp service-0.5.0.jar /opt/lagou/service/griffin-0.5.0/

# 将 measure-0.5.0.jar 拷贝到 /opt/lagou/servers/griffin-0.5.0/，并改名
cd /opt/lagou/servers/griffin-0.5.0/measure/target
cp measure-0.5.0.jar /opt/lagou/servers/griffin-0.5.0/griffin-measure.jar

# 将 griffin-measure.jar 上传到 hdfs:///griffin 中
cd /opt/lagou/servers/griffin-0.5.0
hdfs dfs -mkdir /griffin
hdfs dfs -put griffin-measure.jar /griffin
```

备注：spark在yarn集群上执行任务时，需要到HDFS的/griffin目录下加载griffin-measure.jar，避免发生类org.apache.griffin.measure.Application找不到的错误。

4.7 启动Griffin服务

启动Griffin管理后台：

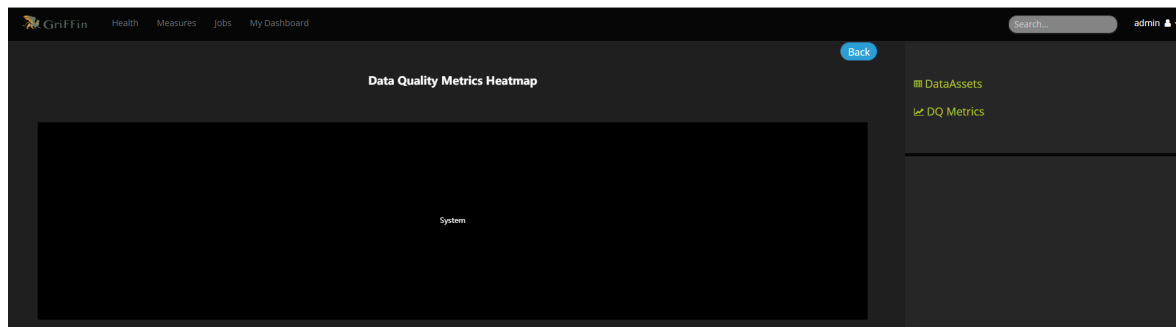
```
cd /opt/lagou/servers/griffin-0.5.0
nohup java -jar service-0.5.0.jar>service.out 2>&1 &
```

Apache Griffin的UI: <http://linux122:9876>

用户名口令：admin / admin



登录后的界面：



第6节 与电商业务集成

6.1 数据资产

单击右上角的 DataAssets 来检查数据资产

Table Name	DB Name	Owner	Creation Time	Location
dwd_trade_orders	dwd	root	9/1/2020, 5:56 PM	hdfs://linux121:9000/user/hive/warehouse/dwd.db/dwd_trade_orders
ads_trade_order_analysis	ads	root	9/1/2020, 5:56 PM	hdfs://linux121:9000/user/hive/warehouse/ads.db/ads_trade_order_analysis
dws_trade_orders	dws	root	9/1/2020, 5:56 PM	hdfs://linux121:9000/user/hive/warehouse/dws.db/dws_trade_orders
dws_trade_orders_w	dws	root	9/1/2020, 5:56 PM	hdfs://linux121:9000/user/hive/warehouse/dws.db/dws_trade_orders_w
rowline1	default	root	7/3/2020, 1:32 PM	hdfs://linux121:9000/user/hive/warehouse/rowline1

备注：这里的数据数据资产主要是保存在Hive上的表，要求 Hive Metastore 服务正常

6.2 创建 measure

- 如果要测量源和目标之间的匹配率，请选择 **Accuracy**（精确度验证）
- 如果要检查数据的特定值（例如：空列计数），请选择 **Data Profiling**（数据统计分析）
 - 统计表的特定列里面值为空、唯一或是重复的数量
 - 统计最大值、最小值、平均数、中值等
 - 用正则表达式来对数据的频率和模式进行分析

Accuracy

Definition: Measured by how the values agree with an identified source of truth

1. Choose Source 2. Choose Target 3. Mapping Source and Target 4. Partition Configuration 5. Configuration

1. Select the source dataset and fields which will be used for comparison
2. Select the target dataset and fields which will be used for comparison
3. Mapping the target fields with source
4. Set partition configuration for source dataset and target dataset
5. Set basic configuration for your model (name, system, threshold, etc.)

Example: suppose source table A has 1000 records and target table B only has 999 records perfectly matched with A for selected fields, then Accuracy Rate(%) = $999/1000 * 100\% = 99.9\%$

Data Profiling

Definition: Data profiling is the process of examining the data available in an existing data set and collecting statistics and information about that data

1. Choose Target 2. Define/Select Models 3. Partition Configuration 4. Configuration

1. Select the target dataset and fields which want to be checked
2. Define your syntax check logic which will be applied on the selected fields
3. Set partition configuration for target dataset
4. Set basic configuration for your model(name, system, threshold, etc.)

Example: Check the data range(minimum, maximum) within a set of allowable values

核心交易分析中有两张表：

- dws_trade_orders (订单明细)
- dws_trade_orders_w (订单明细宽表)

这两张表的数据量应该是相等的 (Accuracy)

计算ODS层

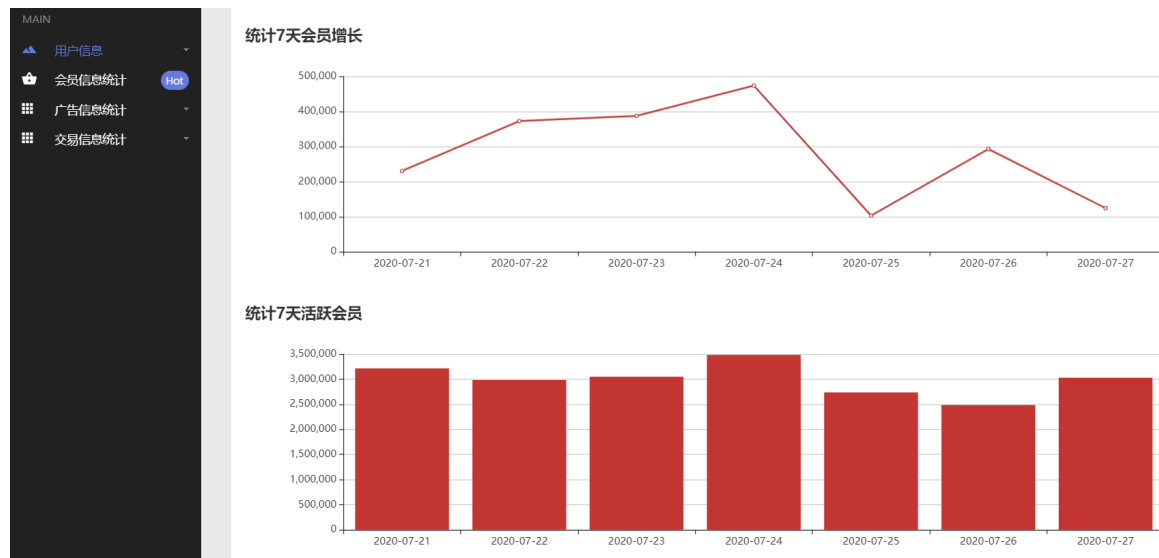
- ods_trade_orders(订单表)

订单表的数据量(Data Profiling)

第五部分 数据可视化

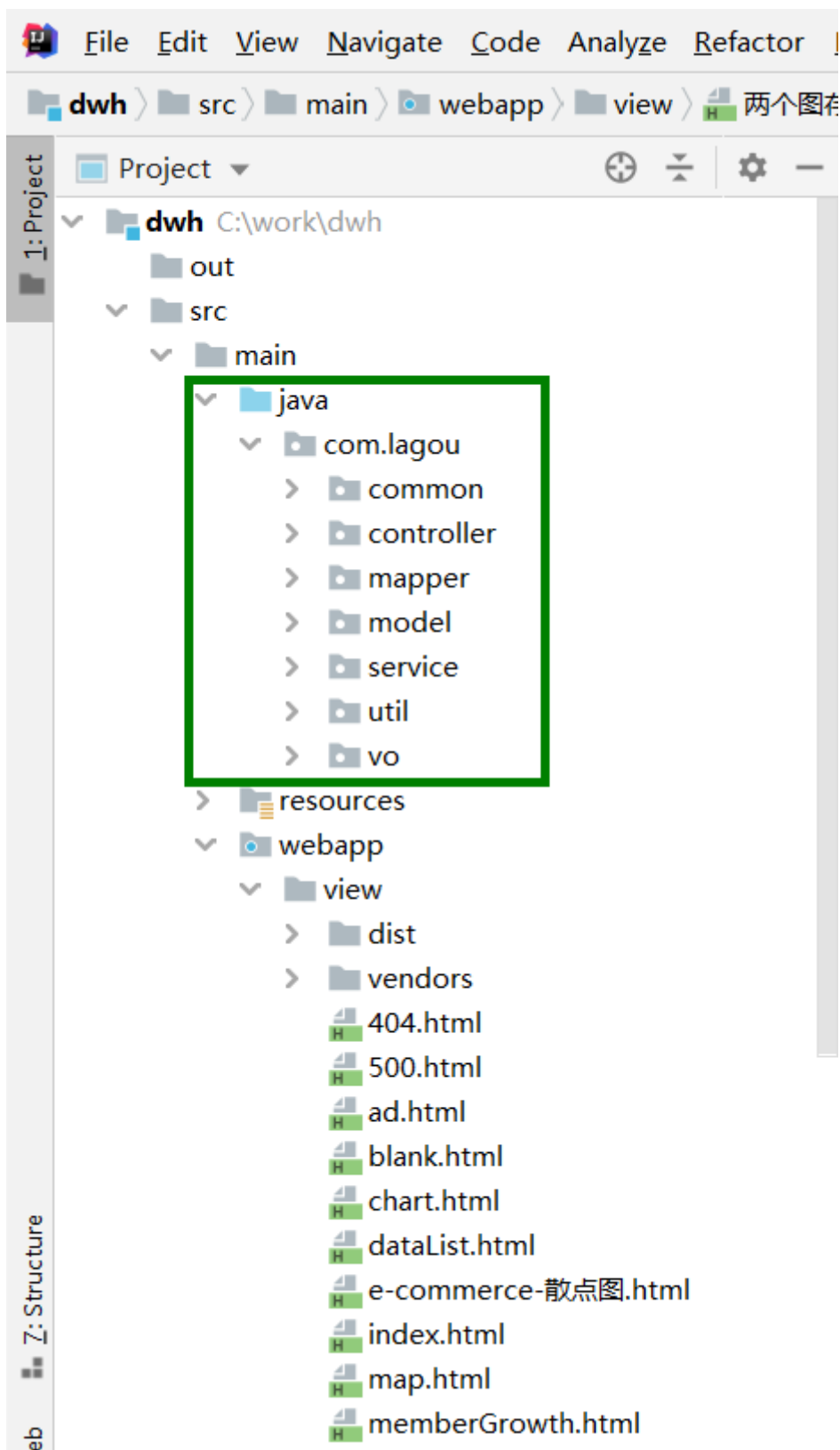
ADS => DataX => MySQL => 浏览器呈现

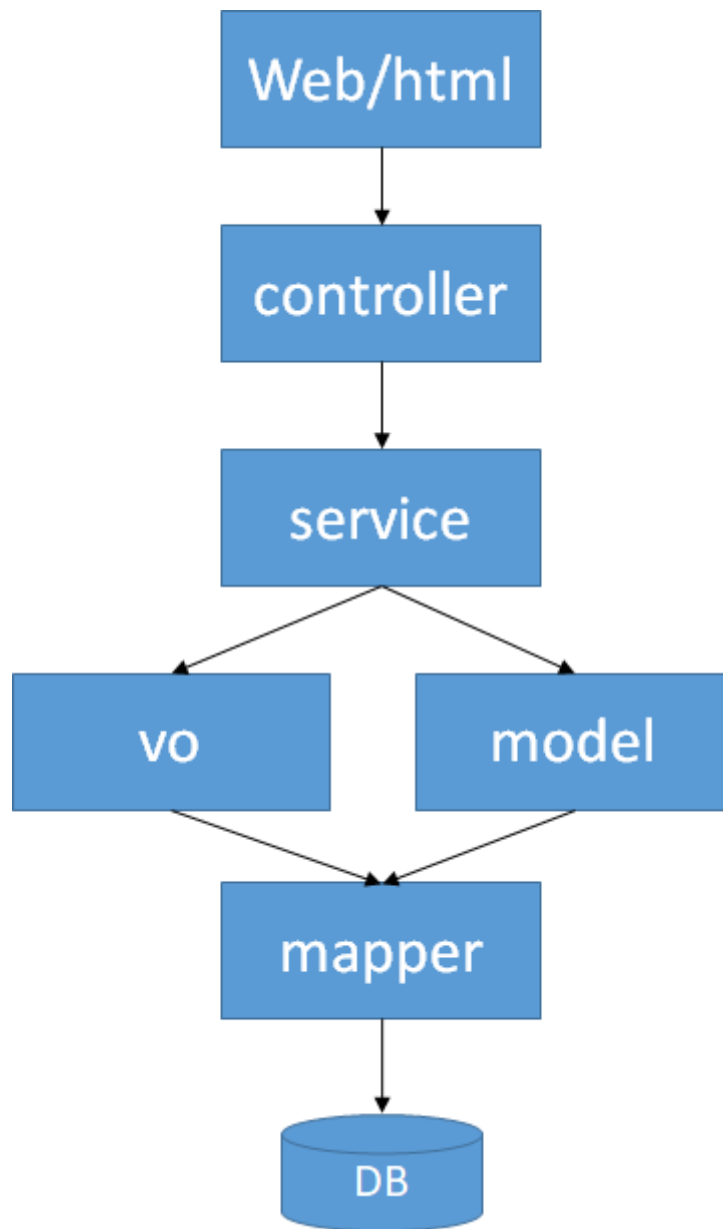
对统计数据进行展示，一般都是以图、表方式呈现；常见方式有 ECharts、HighCharts、G2、Chart.js、FineBI等。本项目使用SSM (Spring + SpringMVC + MyBatis) 、ECharts。



备注:

- src/main/resources/jdbc.properties: jdbc连接信息
- src/main/resources/log4j.properties: 日志存放位置
- MySQL中创建user表, 添加信息
- <http://localhost:8080/dwh/login.html>
- admin/admin





项目总结与回顾

1、数据仓库概念

数据仓库是一个面向主题的、集成的、相对稳定的、反映历史变化的数据集合，用于支持管理决策。

OLAP（数据仓库）与OLTP（数据库）的区别；

数据仓库分层：ODS、DWD、DWS、ADS

为什么要分层：

- 清晰的数据结构
- 将复杂的问题简单化
- 减少重复开发
- 屏蔽原始数据的异常

- 数据血缘的追踪

数据仓库建模：维度建模、ER建模

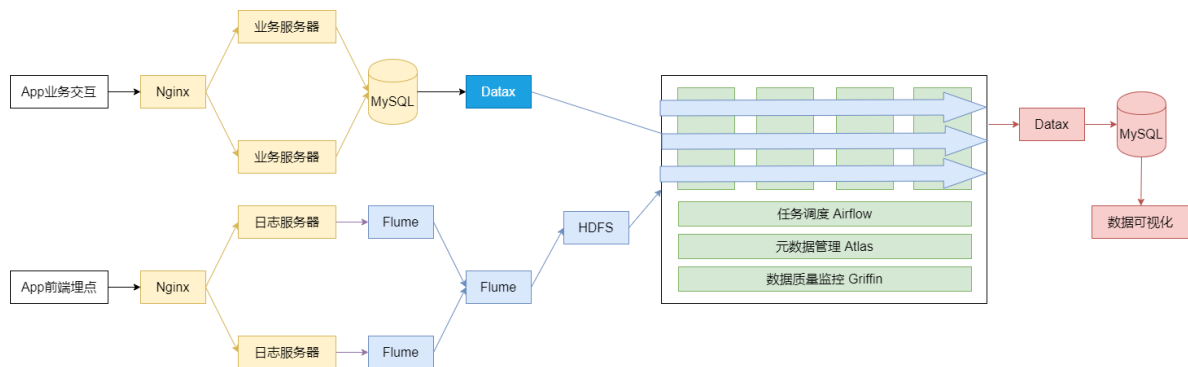
维度建模的4个步骤：

- 选择业务
- 定义粒度
- 选定维度
- 确定事实

集群的规划：

- 集群可以做水平扩展
- 初始时可依据数据量估算集群规模

框架版本的选型：CDH国内选用最多的版本



2、数据采集模块

Flume采集日志数据、DataX采集业务数据（数据的全量或增量）；

Flume组成、Put事务(Source到Channel是Put事务)、Take事务(Channel到Sink是Take事务)

Taildir Source：断点续传、监控多目录。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。

File Channel：数据存储在磁盘，宕机数据可以保存。但是传输速率慢。适合对数据传输可靠性要求高的场景，比如，金融行业；

Memory Channel：数据存储在内存中，宕机数据丢失。传输速率快。适合对数据传输可靠性要求不高的场景，比如，普通的日志数据；

Kafka Channel：减少了Flume的Sink阶段，提高了传输效率；

HDFS Sink：如何避免小文件(HDFS文件的滚动方式)

Flume自定义拦截器：

- initialize 初始化
- intercept(Event event) 处理单个Event【实现的重点】
- intercept(List events) 处理多个Event
- close 方法

设置Agent JVM heap为4G或更高，部署在单独的服务器上；

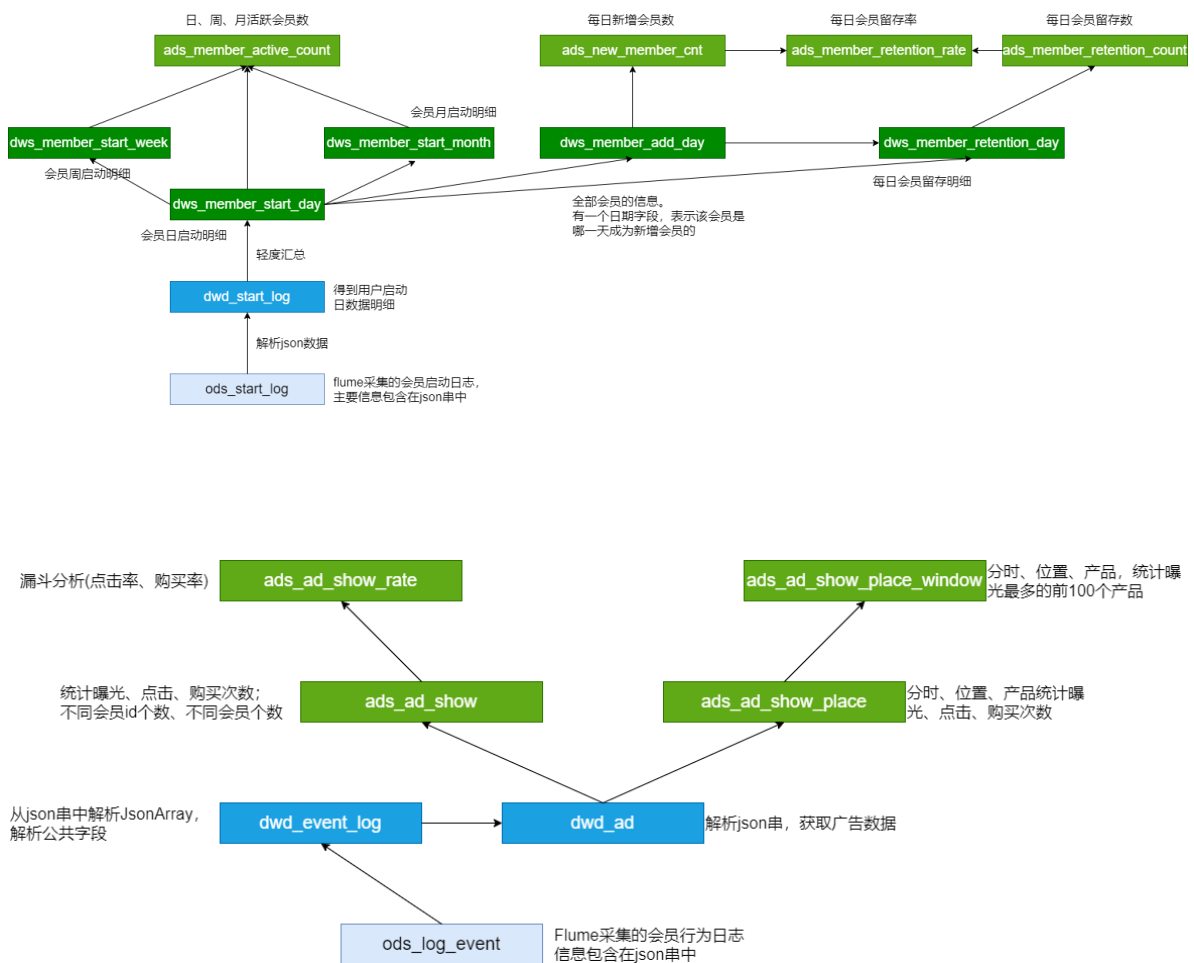
-Xmx与-Xms设置一致，减少内存抖动带来的性能影响，设置不一致容易导致频繁full gc；

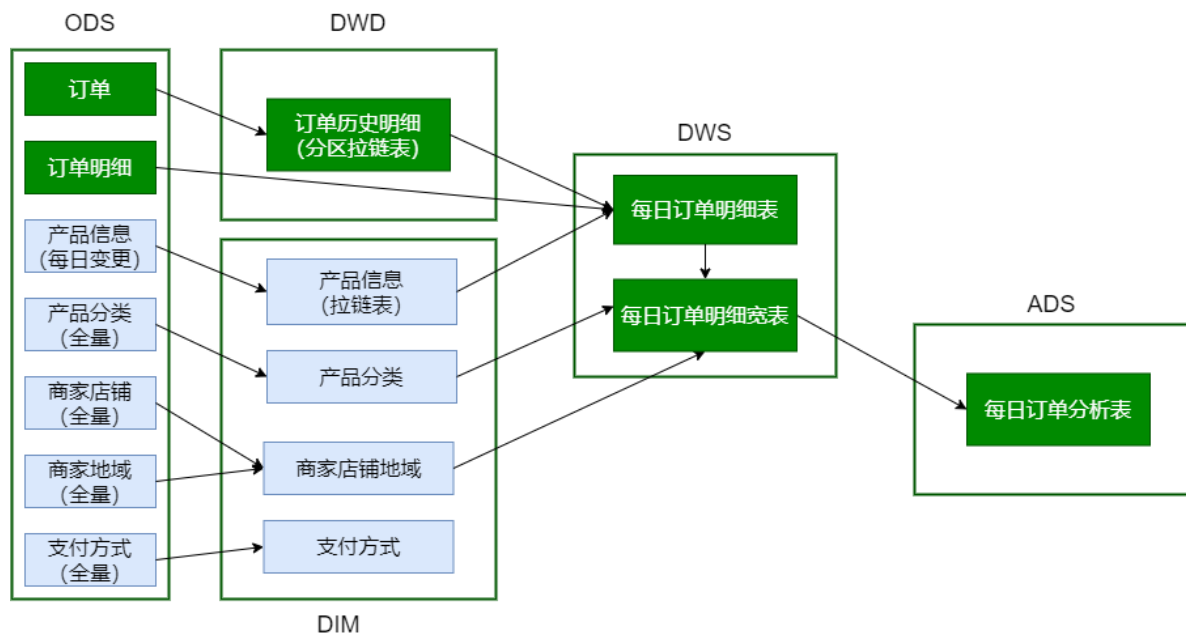
3、主题分析模块【重点】

会员活跃度分析、广告业务分析、核心交易分析；

Json数据的处理、动态分区、拉链表、宽表(逆规范化)、Tez引擎（缺点：对资源要求高）

ODS、DWD、DWS、ADS、DIM各层模型如何建立；





4、调度系统

5、元数据管理数据、数据质量监控 (扩展)

6、数据可视化