配置化任务开发框架 dataConnector 功能简介

data-connector

基于 spark，用于连接不同异构数据源，通过可配置的、兼容 sql 语法的形式实现多数据源的组合、转换并持久化到不同存储介质。

主要亮点

不用写代码即可实现任务输出

可替代数据开发角色

逻辑简单明了

入门门槛低

使用场景

目前主要定位日常周期性的任务统计和常规 ETL 任务开发等。

主要算子

dataSource，dataJoin，dataGroup，dataCollect ，dataMap 操作结果都表示数据源，其中 dataSource 是初始数据源。

dataSource：从存储介质加载数据源并生成临时表。可支持的数据源有 hdfs，关系数据库，hive 表等。

```
<yuedu:dataSource id="push" format="parquet" path="/user/pris/warehouse/fenzhan/app/dt=${statDate}"
                  filter="eventId in ('e-8','e-12','e-14','e-16','e-18','e-20','e-22')"
                  condition="msgId is not null" cache="true">
    <yuedu:column expr="eventId"/>
    <yuedu:column expr="userId" alias="pushUserId"/>
    <yuedu:column expr="siteId"/>
    <yuedu:column expr="linkId"/>
    <yuedu:column expr="timestamp" alias="pushTime"/>
    <yuedu:column expr="params['pos']" alias="pos"/>
    <yuedu:column expr="params['msgId']" alias="msgId"/>
    <yuedu:column expr="params['sourceUuid']" alias="sourceUuid"/>
</yuedu:dataSource>
```

select eventide,userId ad pushUserId,siteId,linkId,timestamp as pushTime,params['pos'] as pos, ,params['msgId'] as msgId ,params['sourceUuid'] as sourceUuid' from push where eventId in ('e-8','e-12','e-14','e-16','e-18','e-20','e-22') and msgId is not null

dataJoin: 实现两个表的连接，包括 inner，outer，cross join 等。

```
<yuedu:dataJoin id="push_receive" joinType="LEFT OUTER JOIN" leftRef="push" rightRef="receive"
                filter="push.msgId=receive.msgId and push.pos=receive.pos and ((push.sourceUuid is null and receive.sourceUuid is null) or (push.sourceUuid=receive.sourceUuid))">
    <yuedu:column expr="push.*"/>
    <yuedu:column expr="receive.userId" alias="receiveUserId"/>
</yuedu:dataJoin>
```

select push.*,receive.userId as receiveUserId from push left outer join receive on push.msgId=receive.msgId and push.pos=receive.pos and push.sourceUuid=receive.sourceUuid

dataGroup:在现有表数据基础上实现分组统计

```
<yuedu:dataGroup id="push_receive_user_count" target="push_receive" groupBy="siteId,eventId,pos">
    <yuedu:column expr="count(receiveUserId)" alias="receiveCount"/>
    <yuedu:column expr="count(distinct receiveUserId)" alias="receiveUserCount"/>
    <yuedu:column expr="count(pushUserId)" alias="pushCount"/>
    <yuedu:column expr="count(distinct pushUserId)" alias="pushUserCount"/>
    <yuedu:column expr="siteId"/>
    <yuedu:column expr="eventId"/>
    <yuedu:column expr="pos"/>
</yuedu:dataGroup>
```

select count(receiveUserId) as receiveCount,
count(distinct receiveUserId) as receiveUserCount,
count(pushUserId) as pushCount,
count(distinct pushUserId) as pushUserCount,siteId,eventId,pos
from push_receive group by siteId,eventId,pos

dataMap:在现有数据源基础上进行选择字段域的变换等

```
<yuedu:dataMap id="tag_read_user" target="ext_read_user">
    <yuedu:column expr="userId"/>
    <yuedu:column expr="siteId"/>
    <yuedu:column expr="sourceUuid"/>
    <yuedu:column expr="case when registerDate=readDate then 1 else 2 end" alias="readUserTag"/>
    <yuedu:column expr="case when registerDate=readDate and rank=1 then 3 else 4 end" alias="accessUserTag"/>
</yuedu:dataMap>
```

select userId,siteId,sourceUuid,
case when registerDate=readDate then 1 else 2 end as readUserTag,
case when registerDate=readDate and rank=1 then 3 else 4 end as accessUserTag
from ext_read_user

dataCollect:实现两个数据表的 union，INTERSECT，except 等。

```
<yuedu:dataCollect id="union_trade_user" collectType="UNION" leftRef="trade" rightRef="simple_vip_read">
    <yuedu:column expr="userId"/>
    <yuedu:column expr="siteId"/>
    <yuedu:column expr="sourceUuid"/>
    <yuedu:column expr="price1"/>
    <yuedu:column expr="price2"/>
    <yuedu:column expr="date"/>
    <yuedu:column expr="vipReadRecord"/>
    <yuedu:column expr="rank"/>
</yuedu:dataCollect>
```

select userId,siteId, sourceUuid,price1,price2,date,vipReadRecord,rank from trade UNION
select userId,siteId, sourceUuid,price1,price2,date,vipReadRecord,rank from simple_vip_read

配置了 savePath 或者 jdbcUrl 的数据源结果将会被持久化，例如

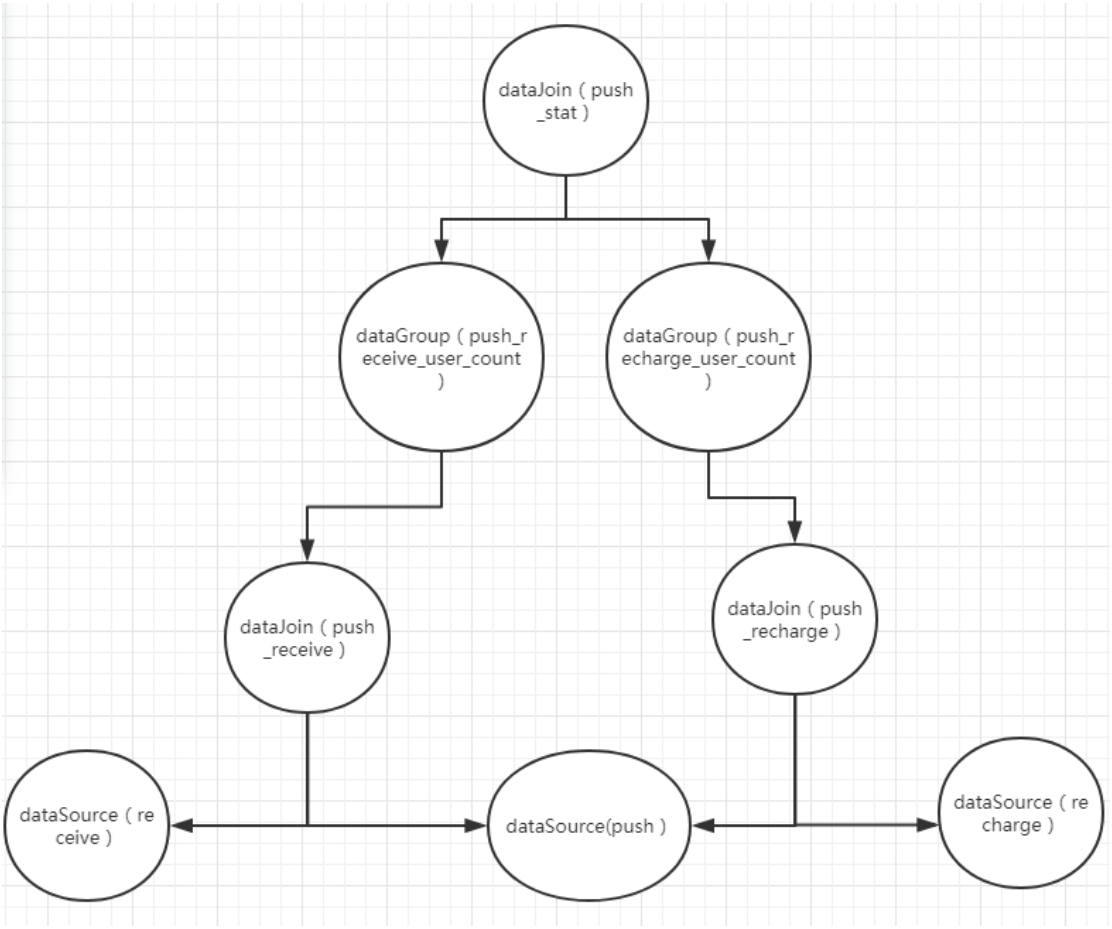
```
<yuedu:dataJoin id="push_stat2" joinType="LEFT OUTER JOIN" leftRef="push_stat1" rightRef="read_user_count"
                filter="push_stat1.siteId=read_user_count.siteId and push_stat1.eventId=read_user_count.eventId and push_stat1.pos=read_user_count.pos"
                savePath="/user/pris/tmp/push_stat/${statDate}-${duration}"
                saveFormat="parquet" saveMode="append" partitionNum="1"
                jdbcUrl="jdbc:mysql://10.201.241.27:3306/yaolu_rds?user=yaolu_static&amp;password=roNaCAmJ{"
                table="fenzhan_push_stat">
    <yuedu:column expr="push_stat1.*"/>
    <yuedu:column expr="readUserCount"/>
</yuedu:dataJoin>
```

算子组合

dataJoin 和 dataCollect 是两元操作算子，分别用 leftRef 和 rightRef 指示左右数据源，dataGroup，dataMap 是一元操作算子，分别用 target 指示操作对象

dataSource 是原子数据源，被其他操作算子指定，位于依赖树的叶子节点。

依赖树示例：



开发指南

TaskProcessor 是 main class，支持 gnu like 的参数形式，主要的参数 config 和 hdfs，如果 hdfs 选项指定了，就表示配置文件使用的 hdfs 路径，否则代表本地文件路径。

```
usage: TaskProcessor
 -config <file>          use given file for xml configuration
 -D <property=value>     use value for given property
 -hdfs                   if hdfs option is set,a hdfs path is coming
                         hereafter
 -statDate <date>        statDate represents yestoday usually
```

本框架基于猛犸调度，只需要上传该项目 jar 包，指定配置文件路径即可。对于日常开发，可以随时修改并且上传配置文件然后调度即可,如下命令：

bin/hadoop fs -put -f push_translation.xml /user/pris/conf

功能完善

1. **dataMap 和 dataGroup target 属性支持多表，达到支持多表的隐式连接功能**
2. **sql 自动翻译为配置文件，减少人工编辑负担。**