SWIFTS guide

修订

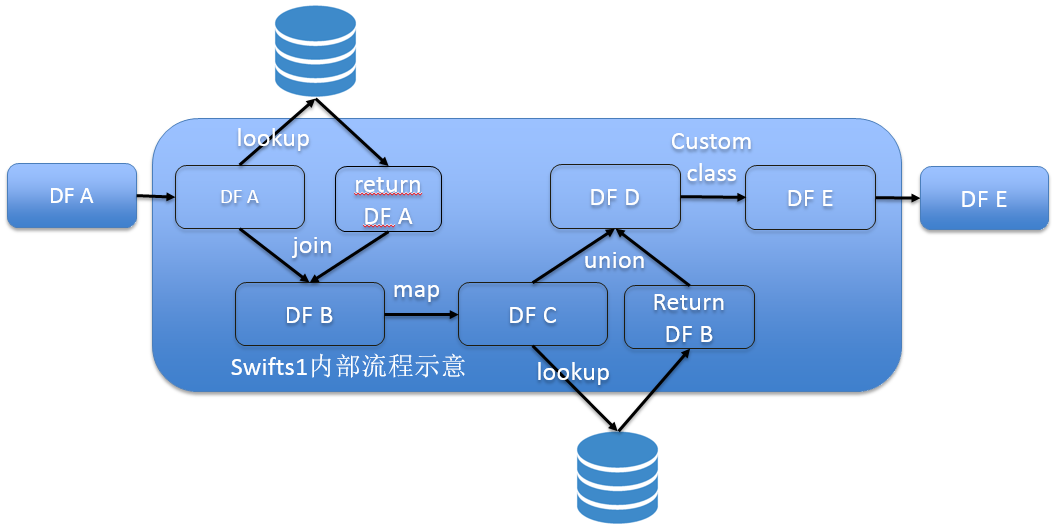
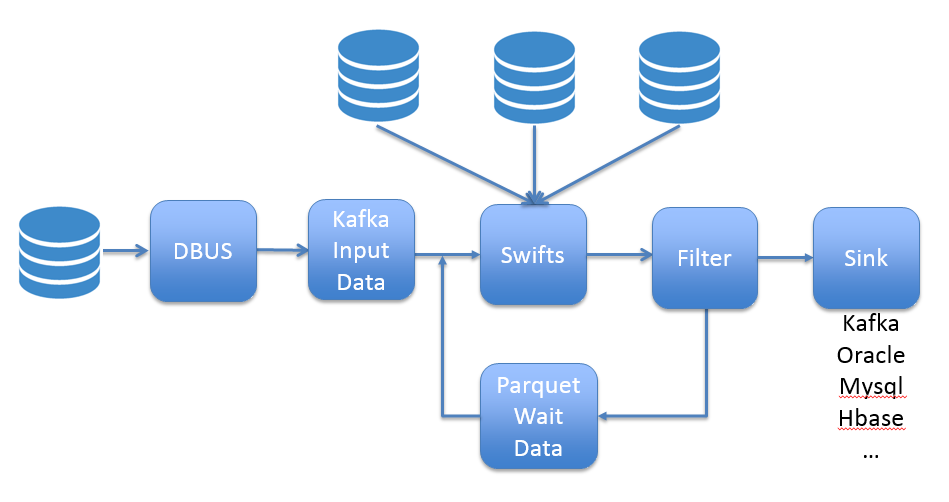
|  |  |  |
| --- | --- | --- |
| 编号 | 日期 | 修订内容 |
| 1 | 20161207 | 添加流上join功能，对directive进行了变更，添加swifts2部分文档。 |
| 2 | 2017060620 | 1. 整合swifts进入wh3 2. 超时等待机制变更为读写hdfs 3. 对PushDown\_Sql与Spark\_Sql中操作类型进行重新定义，并优化了书写规则，支持Custom class，流上join操作（待精细测试） 4. 优化了重写了Pushdonwn\_Sql中的join与union操作，提升4倍性能 |
| 3 | 20171117 | 增加redis和hbase的lookup，语法类似sql |
| 4 | 20180502 | Lookup redis和hbase支持as别名 |
| 5 | 20180904 | 增加lookup hbase示例 |
| 6 | 20190225 | 增加flink swifts配置 |
| 7 | 20190313 | 调整格式内容 |

# 什么是swifts

## 解决问题

Swifts是一款动态流式处理框架，致力于通过业务人员写Sql语句，完成对数据的实时处理，并获得最终结果。

## 设计流程图（sparkx）



## 详细说明

1. DBUS通过源端数据库日志实时从源端抽取数据，并将数据统一成ums数据格式，写入Kafka。
2. Swifts通过用户配置识别从Kafka中抽取出的主流数据，根据业务人员写的Sql逻辑不断地只对主流数据进行操作变换（变换一次后的数据对第二次变换来说是主流数据）。结果输出后进行sink操作。
3. 用户可配置检查规则(option)，通过filter(option ) Swifts生成的宽表（不限于用户最终选择字段，如join后引入很多字段，但用户没有选）某字段或某些字段值为null（and 或者 or关系），视为该结果为无效数据（没有join成功），则将原始数据写入parquet，等待下次与kafka中数据union后通过Swifts继续处理；如检查结果有效，则将最终结果（用户选的字段）（最后面的例子中的output）发往sink进行后续操作。
4. 检查规则配合超时等待一起使用。用户可以配置超时等待时间，如果在超时后根据检查规则判断后仍旧无效，则最终数据可drop, alert，send（用户配置）
5. 当主流数据流入Swifts，Swifts会根据业务人员的Sql语句向对应数据库查找数据，并只将与该批数据相关数据拉回Swifts，进行变换操作（如join，union等）。
6. 目前Swifts可从Mysql，Oracle，ElasticSearch，MongoDB, Cassandra，Hbase，SQL Server, 通过JDBC方式进行查找（lookup）操作。查找出的数据可能已经是经过加工的（如已经在查找的数据库中进行了group by， join等操作），该加工过程是否支持，需要用户对查找数据库有了解，swifts无法保证。

# 支持操作与书写规则

## Lookup\_SQL（sparkx/flinkx）

同spark配置方式，目前只支持left join

1. 单字段join

select col1, col2, … from tableName where (colA) in (mainStreamNamespace.X);

或者

select col1, col2, … from tableName where colA in (${X}); (建议使用此方式)

1. 多字段join

select col1, col2, … from tableName where (colA,colB) in (mainStreamNamespace.X,mainStreamNamespace.Y)

或者

select col1, col2, … from tableName where (colA,colB) in (${X},$(B)); (建议使用此方式)

1. HBase/Redis（hbase和redis不支持sql，swifts进行了类似sql的语法支持）
2. Lookup Redis

select name:type as n1 from default(simple) joinby (key1+'\_'+key2)

select name:type,name:type,name:type from default(json) joinby (key1+'\_'+key2)

1. Lookup HBase

select h1:string as hx,h3:string from test\_lookup(cf1) joinby mod(hash(sub(reverse(md5(id2)),6)),1000)/value(id2+'\_');

joinby字段的规则是嵌套的。：例如mod(hash(sub(reverse(md5(id2)))))

如果是多个字段拼接的rowkey规则，则表示为：value(id)+ '\_'+value(name)

1. 说明
2. in后的括号里面mainStreamNamespace代表主流。框架会把此处替换为主流中的对应数据，以此将有用数据拿到swifts中，而不是将所有数据拿回。然后swifts会做join操作，生成临时表，等待下一个操作。用户可将in条件看做join的on条件。
3. Sql语句选出字段必要时应该给别名。什么是必要时呢？如主流数据有id一项。查找表有id，name，age三项，且与主流用id做关联，选出需要的id，name字段，（age字段用户不需要，不选）。最终join后的宽表为id，id，name三项。该步join操作不会有问题，但是之后如再做select操作，select id字段时，框架无法知道用户要选哪个id字段，会产生歧义，并报错。所以在从查找表中选出这要的字段时，应给一个与主流对应字段不一样的别名如newId。

注意：正常Sql语句部分， in后面的括号中swifts会用真实主流数据做替换，除此用户应保证sql能够正常在数据库中执行。( in 前后都需要有空格)

1. 每次lookup中若包含group by，只能有一次，且group by的字段一定要所有包含 where in条件中的字段。
2. Join 字段必须选出。

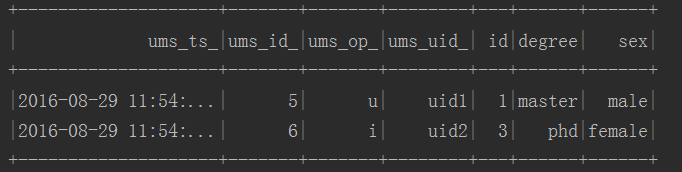
## Spark\_SQL（sparkx）

SPARK\_SQL操作用于对主流的变换，与查找数据库无关, 凡是spark sql支持的这里都支持。

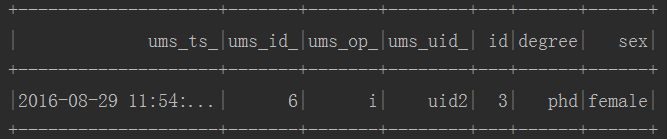
例子：

select id, degree, sex from AL\_APPLY where id > 1;

主流数据AL\_APPLY：



变换后数据：

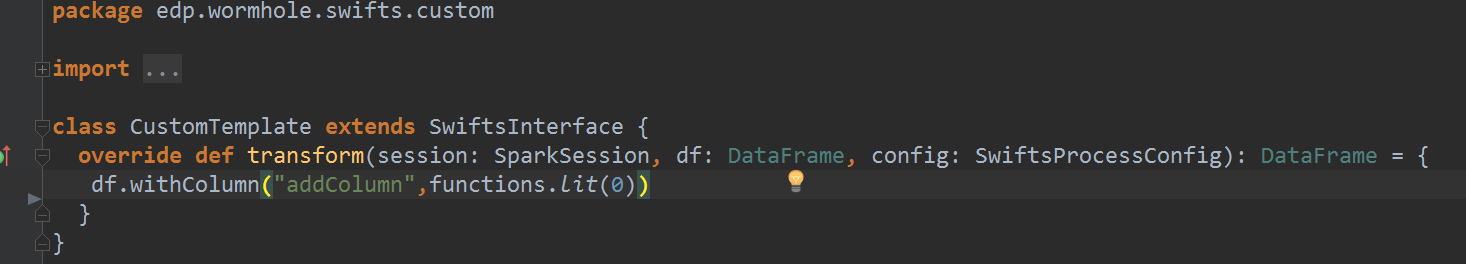


## Custom\_class（sparkx）

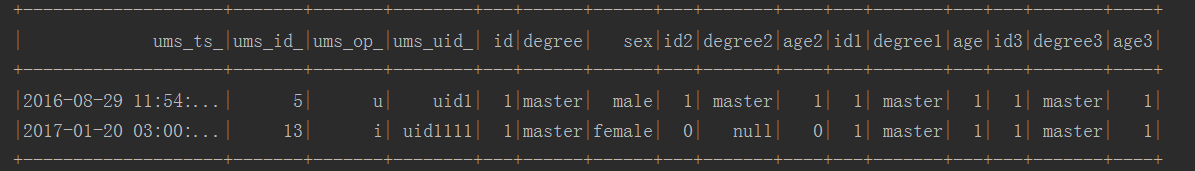
用户通过自己写代码，对输入df进行操作。

例子

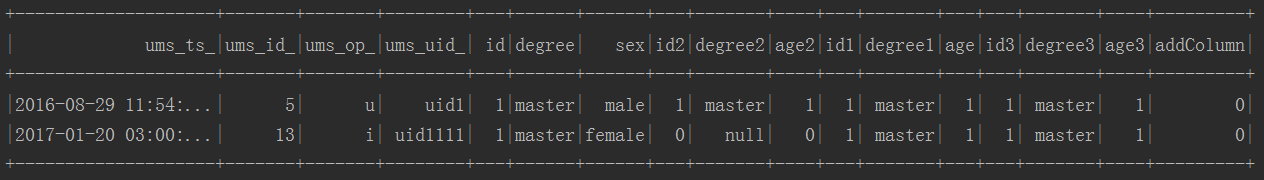
edp.wormhole.swifts.custom.CustomTemplate;



输入数据：



输出数据



## Flink\_SQL（flinkx）

FLINK\_SQL操作用于对主流的变换，与查找数据库无关, 凡是flink sql支持的这里都支持。

1. 普通

select id, degree, sex from AL\_APPLY where id > 1;

1. group by

select key, tumble\_start(ums\_ts\_, interval '1' minute) as wstart, sum(value2)

from tb

group by tumble(ums\_ts\_, interval '1' minute), key;

1. 嵌套(内层嵌套不支持udf)

select key,value1, udf(value2) from (select \* from tb) tb1;

## CEP（flinkx）

1. WindowTime

规则匹配的窗口时间。如果时间窗口截止时，未匹配完成的事件将会被丢弃。

1. Strategy

目前只支持NO\_SKIP（滑动匹配）及SKIP\_LAST\_PAST\_EVENT(滚动匹配)

1. KeyBy

分流字段，可以keyBy在多个字段上，写法为：a,b,c

1. Output

输出方式：

* 1. Agg

按照某个字段聚合后输出，输出字段会自动添加ums\_ts\_ , ums\_id\_ , ums\_op\_以及keyBy分流字段。

聚合单个字段：

keyBy : key , col1:sum 输出字段包含：ums\_ts\_ , ums\_id\_ , ums\_op，col1, key

聚合多个字段：

keyBy : key , col1:sum,col2:avg,newFiled:count 输出字段包含：ums\_ts\_ , ums\_id\_ , ums\_op，col1, clo2, newField, key

* 1. Detail

输出满足规则的事件明细

* 1. FilteredRow

按照条件输出满足规则的事件中的某一条

Head: 输出第一条

Last: 输出最后一条

Max: 输出某个字段最大的那一条（col1:max）

Min: 输出某个字段最小的那一条 (col1:min)

1. Pattern

具体匹配规则，包含三个要素：Operator, Quantifier, Conditions

Quantifier：Times, TimesOrMore, TimesGreedy, TimesOptional

Conditions: 满足规则的条件

| **Pattern Operation** | **Description** |
| --- | --- |
| **begin(#name)** | 定义开始规则：  Pattern**<**Event**,** **?>** start **=** Pattern**.<**Event**>**begin**(**"start"**);** |
| **next(#name)** | 严格邻近上一个规则，中间不允许有别的事件出现  Pattern**<**Event**,** **?>** next **=** start**.**next**(**"middle"**);** |
| **followedBy(#name)** | 非严格临近，会丢弃部分匹配的事件  Pattern**<**Event**,** **?>** followedBy **=** start**.**followedBy**(**"middle"**);** |
| **notNext()** | 严格临近（否定）规则  Pattern**<**Event**,** **?>** notNext **=** start**.**notNext**(**"not"**);** |
| **notFollowedBy()** | 非严格临近（否定）规则  Pattern**<**Event**,** **?>** notFollowedBy **=** start**.**notFollowedBy**(**"not"**);** |

Operator：

# Flinkx配置

* sql相关

当使用无窗口的group by 操作时，flink输出的事件包含message\_flag（表示是修改还是插入），用户可选择是否保留修改的事件。配置方式为

{    
   "preserve\_message\_flag":"true", #是否保留message\_flag字段  
   "min\_idle\_state\_retention\_time":12, #状态留存最小时间  
   "max\_idle\_state\_retention\_time":24 #状态留存最大时间  
}

* 异常处理相关

用户可以选择异常的处理方式，有三种选择，分别是： interrupt，feedback，unhandle。Interrupt表示如果遇到异常则直接抛出中断运行；feedback表示如果遇到异常对异常进行捕获并反馈给wh rider；unhandle表示对异常捕获后仅记录日志不做任何处理。

配置方式为：

{    
   "exception\_process\_method":"interrupt", #是否保留message\_flag字段  
  }