基于增量变量的数据增量抽取调度框架设计

一、目标

常见的数据增量抽取方案有全量数据对比、基于存储系统变更日志、基于增量变量（比如时间戳）等，各有其适用场景：全量数据对比适用于数据总量比较小的场景；基于存储系统变更日志可以捕获增删改等耕种变更情况，缺点是部署较为麻烦；基于增量变量（比如时间戳）的方式实现简单，缺点是源系统的数据记录必须维护一个增量标识字段，并且无法捕获物理删除事件，也无法直接感知更新操作的发生。

由于不少业务系统不需要物理删除(比如只有基于标记的逻辑删除），并且不需要直接感知更新操作（比如抽取程序可以通过下游系统的UPSERT--update or insert类型的操作来达到更新的目的），因此基于增量变量的方案也经常被采用。 本系统实现了基于增量变量的数据增量抽取调度框架，可以满足不同源系统不同类型增量变量的增量抽取场景。

二、应用场景分析

常见的增量变量类型有时间戳和增量序列等，例如记录的最近更新时间或者序列值等。通常一次增量抽取过程涉及两个增量变量：开始值和结束值，分别指与本次抽取相关的增量变量的最小值和最大值。一个ETL过程一般涉及三个系统：源数据存储系统、抽取程序、目标数据存储系统，因此增量变量值一般是基于以上三个系统获取，当然也不排除需要从其它途径获取的特殊情况。框架主要是围绕如何获取增量变量的值来进行设计的。

1、开始值

开始值的获取方式分为第一次运行和后续的增量运行。对于第一次运行， 通常会设计一个初始值作为开始值；对于后续的增量运行，开始值通常是基于上次运行所采用的结束值。因此一般来说，抽取的区间设计是左开右闭的。如下示例：SELECT * FROM users WHERE last_modified_time > ${BEGIN_TIME} AND last_modified_time <=${END_TIME} 。这里的${BEGIN_TIME}取得就是上次成功运行后所记录的${END_TIME}值。

通常初始值可以是预设的一个常量，例如序列值'0'或者 时间值'1970-01-01 01:01:01'。也可能有其它情况例如从源存储系统获取，如下示例从数据库中查询序列的最小值:SELECT MIN(book_SEQ)-1 FROM books。

2、结束值

在不同的场景下，结束值也存在多种获取需求：

1）不需要结束值

一些抽取逻辑可以不用关注结束值，只需要判定从源目标返回的结果集是否为空，就可以认为本轮抽取结束。这种情况下，一般需要从返回的结果集里找出最大值最为下次增量抽取的开始值。这种方式的限制是每次抽取的区间范围具有不确定性，并且需要定制实现获取最大值的逻辑。

2） 采用系统时间作为结束值

这种场景下变量类型必须是时间戳。有两类场景：对源系统数据的产生时间有较严格的要求，例如报表系统需要统计当天的业务指标，需要将当天的11点59分59秒作为结束时间；定时抽取场景，例如每5分钟抽取一次，需要将抽取时的系统时间戳作为结束时间。

3）结束值需要从源系统中或者其它系统中获取

当增量变量的类型是非时间戳类型，例如增量序列时，而又不希望采用第1种方式（不需要结束值）时，需要定制实现如何获取结束值。例如从源数据存储系统里查询当前时刻的最大增量变量值：

SELECT MAX(book_SEQ) FROM books。

三、框架设计

综合以上分析，框架需要重点解决的问题是增量变量如开始值和结束值的获取。由于数据存储系统的多样性，框架需要支持不同类型的数据存储系统,包括RDBMS、HBASE等。

1、配置说明

程序采用一种开放的配置框架来实现增量逻辑的可配置性。配置框架包括运行状态管理配置项和增量逻辑配置项:

1） 运行状态管理配置项

运行状态主要包括上次完成的抽取过程所记录下来的结束值。配置项包括运行状态存储记录的主键（key)。示例如下:

```
#唯一标识Job状态存储信息的主键。
# 例如：对于采用文件存储 Job Status的实现，key一般采用状态文件的绝对路径名;
# 对于采用数据库存储Job Status的实现，key可以是用于唯一标识一条记录的主键，这里主键需要配置自动保持
# 全局唯一性
job.status.persist.key=d:/tmp/job_demo1.status
```

2）增量逻辑配置项

对于不同类型数据源和增量逻辑，需要基于系统提供的接口实现不同的配置变量解析器。如下示例是基于关系型数据库的配置:

```
##############################################
以下配置实现功能：
第一次运行时从配置项中获取配置的job.var.conf.begin.init.val值作为开始值;
此后从运行状态数据中获取上次的结束值作为本次运行的开始值;
从数据库中查询max值作为本次运行的结束值。
##############################################
#自定义的可变配置项。key名需要以job.var开头。job.var.impl.后面一个串的名字作为datax job里环境变量的变量名.
例如job.var.impl.begin,那么begin将作为变量名
#从本地配置项中提取开始位置的变量解析器实现
job.var.impl.begin=vsincr.variable.impl.LocalBeginVarResolver
#状态文件中对应的结束变量值的Key的名字，默认__SYS_NOW
job.var.conf.begin.lastend.var.name=end
#用于RDBMS的提取结束位置的变量解析器实现
job.var.impl.end=vsincr.scheduler.rdbms.var.DbEndVarResolver
#RDBMS的数据库连接用户名字
job.var.conf.db.conn.user=XXXX
#RDBMS的数据库连接密码
job.var.conf.db.conn.password=XXXXXXX
#RDBMS的数据库连接url
job.var.conf.db.conn.url=jdbc:mysql://mydev:3306/kg_demo?useUnicode=true
#RDBMS的数据库连接的jdbc驱动
job.var.conf.db.conn.driver=com.mysql.jdbc.Driver
######################################################################

######################################################################
#数据库解析程序用到的自定义配置项
#初始值。第一次抽取时的起始位置
# 初始值属性的配置优先级顺序:init.val,init.query.
job.var.conf.begin.init.val=0
#获取结束值的查询语句
#The type of the return value must be string
job.var.conf.end.query=select max(cast(buyer_id as SIGNED INTEGER)) from buyer
```

2、接口说明

2.1变量解析器接口 IVarResolver

变量值的获取需要实现该接口。系统内置了常用的变量解析器实现:

1）基于RDBMS的变量解析器实现：DbBeginVarResolver(获取开始值)和DbEndVarResolver(获取结束值)

2)基于运行状态数据的变量解析器：LocalBeginVarResolver。用于单纯从本地配置和状态数据中获取开始值的解析器实现。

后续计划扩展更多的实现，如基于HBase、Hive等数据存储系统的变量解析器。

```
/**
* 作业运行变量解析器。
* @author JerryHuang
*/
public interface IVarResolver extends Serializable {
/**
* 初始化变量。
* 为了方便在同一个Job配置里复用多个VarResolver实例（即同一个implementation，不同的配置项的情况），这里将变量名字varName
* 作为上下文传入。例如：
jobstatus.var.impl.begin=vsdatax.scheduler.rdbms.var.DbBeginVarResolver,varName为begin,则在VarResolver
* 实现内部可以将对应的配置项的key设置为job.var.conf.begin.init.val=0,这里begin对应到varName.
* @param varName 变量名。
* @param incrEnvConf 增量抽取逻辑相关的配置项。如数据存储系统相关的配置，获取值的SQL语句等。
*/
public void init( String varName, IncrEnvConf incrEnvConf);

/**
* Resolve the variables.
*
* @param lastJobStatus     作业状态的model类。主要包括上次抽取过程的开始值、结束值以及其它一些可能的系统变量等。
* @return
*/
public Object resolve( JobStatus lastJobStatus);
}
```

2.2 增量变量生成器接口 IIncrEnvGenerator

增量变量生成的入口程序。通过调用该程序，可以生成所有配置的增量变量值。

```
/**
* 增量变量生成器接口
* @author JerryHuang
*/
public interface IIncrEnvGenerator {

/**
* 遍历所配置的所有的解析器，解析出所有的变量值
* @param incrEnvConf 与增量变量相关的配置项
* @param lastJobStatus 上次成功运行的job 状态数据
* @return 解析出来的变量值集合:key 变量名,value 变量值
*/
public Map<String,Object> resolveAll(IncrEnvConf incrEnvConf, JobStatus lastJobStatus);

}
```