一、dataX概览

1.1 DataX

DataX 是阿里巴巴集团内被广泛使用的离线数据同步工具/平台，实现包括 MySQL、SQL Server、Oracle、PostgreSQL、HDFS、Hive、HBase、OTS、ODPS 等各种异构数据源之间高效的数据同步功能。

1.2 Features

DataX本身作为数据同步框架，将不同数据源的同步抽象为从源头数据源读取数据的Reader插件，以及向目标端写入数据的Writer插件，理论上DataX框架可以支持任意数据源类型的数据同步工作。同时DataX插件体系作为一套生态系统, 每接入一套新数据源该新加入的数据源即可实现和现有的数据源互通。

1.3 System Requirements

- Linux
- JDK(1.8以上，推荐1.8)
- Python(推荐Python2.6.X)
- Apache Maven 3.x (Compile DataX)

1.4 Quick Start

- 工具部署
  - 方法一、直接下载DataX工具包：DataX下载地址

    下载后解压至本地某个目录，进入bin目录，即可运行同步作业：

    ```
    $ cd  {YOUR_DATAX_HOME}/bin
    $ python datax.py {YOUR_JOB.json}
    ```

    自检脚本： python {YOUR_DATAX_HOME}/bin/datax.py {YOUR_DATAX_HOME}/job/job.json
  - 方法二、下载DataX源码，自己编译：DataX源码

    (1)、下载DataX源码：

    ```
    $ git clone git@github.com:alibaba/DataX.git
    ```

    (2)、通过maven打包：

    ```
    $ cd  {DataX_source_code_home}
    $ mvn -U clean package assembly:assembly -Dmaven.test.skip=true
    ```

    打包成功，日志显示如下：

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------
[INFO] Total time: 08:12 min
[INFO] Finished at: 2015-12-13T16:26:48+08:00
[INFO] Final Memory: 133M/960M
[INFO] ------------------------------------------------------------------
```

打包成功后的DataX包位于 {DataX_source_code_home}/target/datax/datax/ ， 结构如下：

```
$ cd  {DataX_source_code_home}
$ ls ./target/datax/datax/
bin     conf        job     lib     log     log_perf    plugin      script
tmp
```

- 配置示例：从stream读取数据并打印到控制台
  - 第一步、创建创业的配置文件（json格式）

    可以通过命令查看配置模板： python datax.py -r {YOUR_READER} -w {YOUR_WRITER}

```
$ cd  {YOUR_DATAX_HOME}/bin
$  python datax.py -r streamreader -w streamwriter
DataX (UNKNOWN_DATAX_VERSION), From Alibaba !
Copyright (C) 2010-2015, Alibaba Group. All Rights Reserved.
Please refer to the streamreader document:

https://github.com/alibaba/DataX/blob/master/streamreader/doc/streamreader.md

Please refer to the streamwriter document:

https://github.com/alibaba/DataX/blob/master/streamwriter/doc/streamwriter.md

Please save the following configuration as a json file and  use
     python {DATAX_HOME}/bin/datax.py {JSON_FILE_NAME}.json
to run the job.

{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "streamreader",
                    "parameter": {
                        "column": [],
                        "sliceRecordCount": ""
                    }
                },
                "writer": {
                    "name": "streamwriter",
                    "parameter": {
                        "encoding": "",
                        "print": true
                    }
```

```
                }
            }
        ],
        "setting": {
            "speed": {
                "channel": ""
            }
        }
    }
}
```

根据模板配置json如下：

```
#stream2stream.json
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "sliceRecordCount": 10,
            "column": [
              {
                "type": "long",
                "value": "10"
              },
              {
                "type": "string",
                "value": "hello，你好，世界-DataX"
              }
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "encoding": "UTF-8",
            "print": true
          }
        }
      }
    ],
    "setting": {
      "speed": {
        "channel": 5
       }
    }
  }
}
```

- 第二步：启动DataX

```
$ cd {YOUR_DATAX_DIR_BIN}
$ python datax.py ./stream2stream.json
```

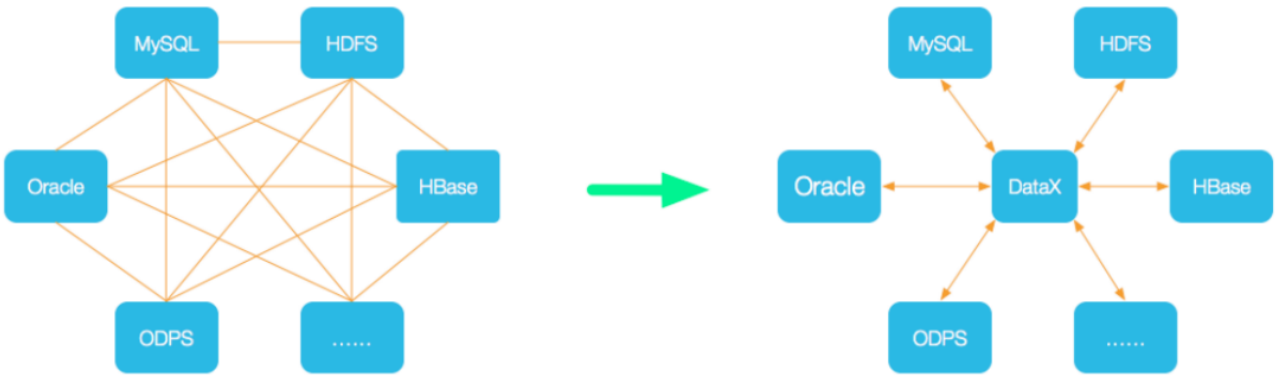
同步结束，显示日志如下：

```
...
2015-12-17 11:20:25.263 [job-0] INFO  JobContainer -
任务启动时刻                    : 2015-12-17 11:20:15
任务结束时刻                    : 2015-12-17 11:20:25
任务总计耗时                    :                 10s
任务平均流量                    :              205B/s
记录写入速度                    :              5rec/s
读出记录总数                    :                 50
读写失败总数                    :                  0
```

二、dataX详解

2.1 DataX 3.0概览

DataX 是一个异构数据源离线同步工具，致力于实现包括关系型数据库(MySQL、Oracle等)、HDFS、Hive、ODPS、HBase、FTP等各种异构数据源之间稳定高效的数据同步功能。



- **设计理念**

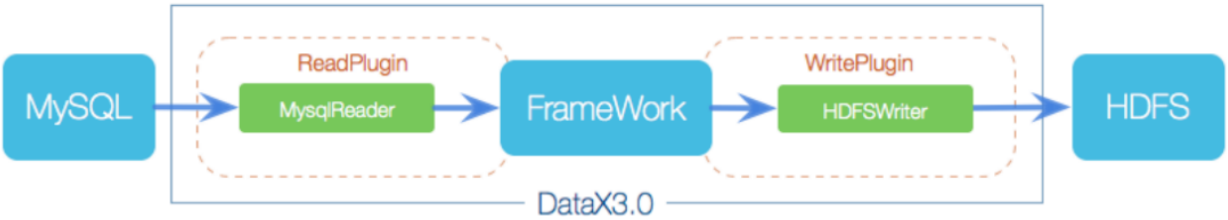
  为了解决异构数据源同步问题，DataX将复杂的网状的同步链路变成了星型数据链路，DataX作为中间传输载体负责连接各种数据源。当需要接入一个新的数据源的时候，只需要将此数据源对接到DataX，便能跟已有的数据源做到无缝数据同步。

- **当前使用现状**

  DataX在阿里巴巴集团内被广泛使用，承担了所有大数据的离线同步业务，并已持续稳定运行了6年之久。目前每天完成同步8w多道作业，每日传输数据量超过300TB。

此前已经开源DataX1.0版本，此次介绍为阿里云开源全新版本DataX3.0，有了更多更强大的功能和更好的使用体验。Github主页地址：https://github.com/alibaba/DataX

2.2 DataX3.0框架设计



DataX本身作为离线数据同步框架，采用Framework + plugin架构构建。将数据源读取和写入抽象成为Reader/Writer插件，纳入到整个同步框架中。

- Reader：Reader 为数据采集模块，负责采集数据源的数据，将数据发送给Framework。
- Writer： Writer为数据写入模块，负责不断向Framework取数据，并将数据写入到目的端。
- Framework：Framework用于连接reader和writer，作为两者的数据传输通道，并处理缓冲，流控，并发，数据转换等核心技术问题。

2.3 DataX3.0插件体系

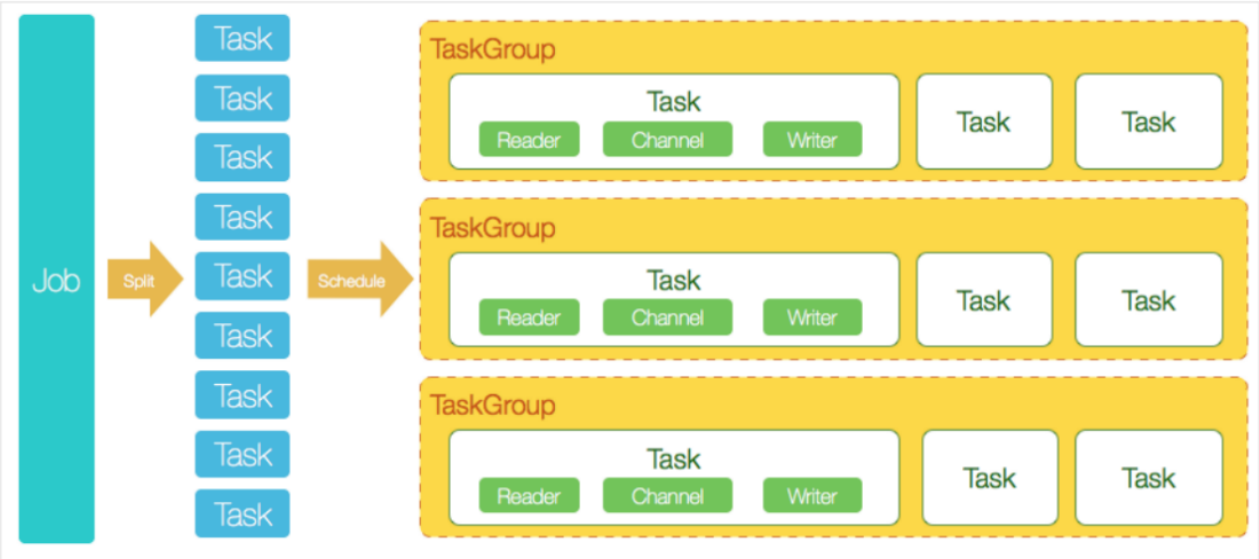
经过几年积累，DataX目前已经有了比较全面的插件体系，主流的RDBMS数据库、NOSQL、大数据计算系统都已经接入。DataX目前支持数据如下：

类型	数据源	Reader(读)	Writer(写)	文档
RDBMS 关系型数据库	MySQL	√	√	读、写
	Oracle	√	√	读、写
	SQLServer	√	√	读、写
	PostgreSQL	√	√	读、写
	DRDS	√	√	读、写
	达梦	√	√	读、写
	通用RDBMS(支持所有关系型数据库)	√	√	读、写
阿里云数仓数据存储	ODPS	√	√	读、写
	ADS		√	写
	OSS	√	√	读、写
	OCS	√	√	读、写
NoSQL数据存储	OTS	√	√	读、写
	Hbase0.94	√	√	读、写
	Hbase1.1	√	√	读、写
	MongoDB	√	√	读、写
	Hive	√	√	读、写
无结构化数据存储	TxtFile	√	√	读、写
	FTP	√	√	读、写
	HDFS	√	√	读、写
	Elasticsearch		√	写

DataX Framework提供了简单的接口与插件交互，提供简单的插件接入机制，只需要任意加上一种插件，就能无缝对接其他数据源。详情请看：DataX数据源指南

2.4 DataX3.0核心架构

DataX 3.0 开源版本支持单机多线程模式完成同步作业运行，本小节按一个DataX作业生命周期的时序图，从整体架构设计非常简要说明DataX各个模块相互关系。



2.4.1 核心模块介绍：

1. DataX完成单个数据同步的作业，我们称之为Job，DataX接受到一个Job之后，将启动一个进程来完成整个作业同步过程。DataX Job模块是单个作业的中枢管理节点，承担了数据清理、子任务切分(将单一作业计算转化为多个子Task)、TaskGroup管理等功能。
2. DataXJob启动后，会根据不同的源端切分策略，将Job切分成多个小的Task(子任务)，以便于并发执行。Task便是DataX作业的最小单元，每一个Task都会负责一部分数据的同步工作。
3. 切分多个Task之后，DataX Job会调用Scheduler模块，根据配置的并发数据量，将拆分成的Task重新组合，组装成TaskGroup(任务组)。每一个TaskGroup负责以一定的并发运行完毕分配好的所有Task，默认单个任务组的并发数量为5。
4. 每一个Task都由TaskGroup负责启动，Task启动后，会固定启动Reader—>Channel—>Writer的线程来完成任务同步工作。
5. DataX作业运行起来之后， Job监控并等待多个TaskGroup模块任务完成，等待所有TaskGroup任务完成后Job成功退出。否则，异常退出，进程退出值非0

2.4.2 DataX调度流程：

举例来说，用户提交了一个DataX作业，并且配置了20个并发，目的是将一个100张分表的mysql数据同步到odps里面。 DataX的调度决策思路是：

1. DataXJob根据分库分表切分成了100个Task。
2. 根据20个并发，DataX计算共需要分配4个TaskGroup。
3. 4个TaskGroup平分切分好的100个Task，每一个TaskGroup负责以5个并发共计运行25个Task。

2.5 DataX 3.0六大核心优势

- **可靠的数据质量监控**
    - 完美解决数据传输个别类型失真问题

      DataX旧版对于部分数据类型(比如时间戳)传输一直存在毫秒阶段等数据失真情况，新版本DataX3.0已经做到支持所有的强数据类型，每一种插件都有自己的数据类型转换策略，让数据可以完整无损的传输到目的端。
    - 提供作业全链路的流量、数据量运行时监控

DataX3.0运行过程中可以将作业本身状态、数据流量、数据速度、执行进度等信息进行全面的展示，让用户可以实时了解作业状态。并可在作业执行过程中智能判断源端和目的端的速度对比情况，给予用户更多性能排查信息。

- 提供脏数据探测

  在大量数据的传输过程中，必定会由于各种原因导致很多数据传输报错(比如类型转换错误)，这种数据DataX认为就是脏数据。DataX目前可以实现脏数据精确过滤、识别、采集、展示，为用户提供多种的脏数据处理模式，让用户准确把控数据质量大关！

- **丰富的数据转换功能**

DataX作为一个服务于大数据的ETL工具，除了提供数据快照搬迁功能之外，还提供了丰富数据转换的功能，让数据在传输过程中可以轻松完成数据脱敏，补全，过滤等数据转换功能，另外还提供了自动groovy函数，让用户自定义转换函数。详情请看DataX3的transformer详细介绍。

- **精准的速度控制**

还在为同步过程对在线存储压力影响而担心吗？新版本DataX3.0提供了包括通道(并发)、记录流、字节流三种流控模式，可以随意控制你的作业速度，让你的作业在库可以承受的范围内达到最佳的同步速度。

```
"speed": {
   "channel": 5,
   "byte": 1048576,
   "record": 10000
}
```

- **强劲的同步性能**

DataX3.0每一种读插件都有一种或多种切分策略，都能将作业合理切分成多个Task并行执行，单机多线程执行模型可以让DataX速度随并发成线性增长。在源端和目的端性能都足够的情况下，单个作业一定可以打满网卡。另外，DataX团队对所有的已经接入的插件都做了极致的性能优化，并且做了完整的性能测试。性能测试相关详情可以参照每单个数据源的详细介绍：DataX数据源指南

- **健壮的容错机制**

DataX作业是极易受外部因素的干扰，网络闪断、数据源不稳定等因素很容易让同步到一半的作业报错停止。因此稳定性是DataX的基本要求，在DataX 3.0的设计中，重点完善了框架和插件的稳定性。目前DataX3.0可以做到线程级别、进程级别(暂时未开放)、作业级别多层次局部/全局的重试，保证用户的作业稳定运行。

- 线程内部重试

  DataX的核心插件都经过团队的全盘review，不同的网络交互方式都有不同的重试策略。

- 线程级别重试

  目前DataX已经可以实现TaskFailover，针对于中间失败的Task，DataX框架可以做到整个Task级别的重新调度。

- **极简的使用体验**

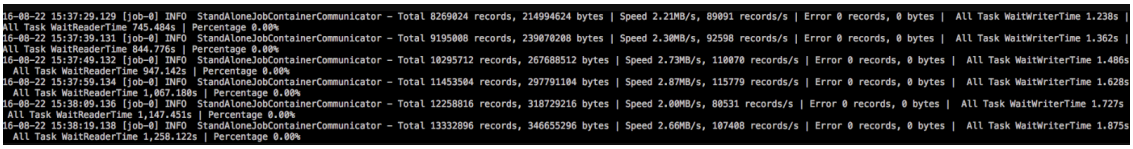
  - 易用

    下载即可用，支持linux和windows，只需要短短几步骤就可以完成数据的传输。请点击：Quick Start

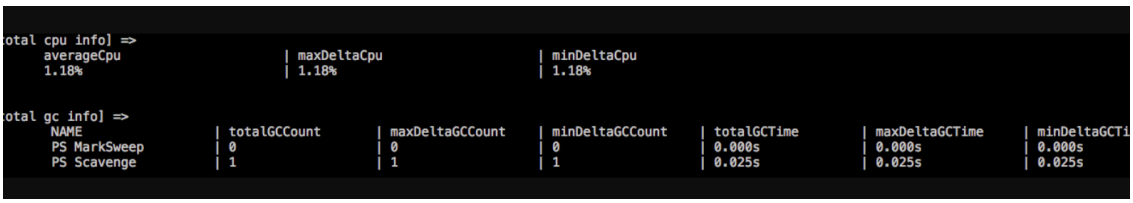
  - 详细

DataX在运行日志中打印了大量信息，其中包括传输速度，Reader、Writer性能，进程CPU，JVM和GC情况等等。

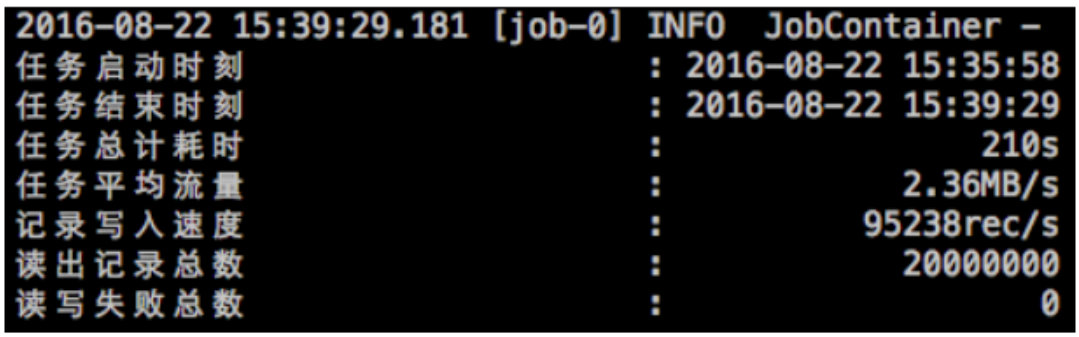
- 传输过程中打印传输速度、进度等



- 传输过程中会打印进程相关的CPU、JVM等



- 在任务结束之后，打印总体运行情况



三、dataX案例

3.1 案例1(stream--->stream)

datax使用插件式开发，官方参考文档如下:https://github.com/alibaba/DataX/blob/master/dataxPluginDev.md

描述:streaming reader--->streaming writer (官网例子)

```
[root@hadoop01 home]# cd /usr/local/datax/

[root@hadoop01 datax]# vi ./job/first.json
内容如下:
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "sliceRecordCount": 10,
            "column": [
              {
```

```
                "type": "long",
                "value": "10"
              },
              {
                "type": "string",
                "value": "hello，你好，世界-DataX"
              }
            ]
          }
        },
        "writer": {
          "name": "streamwriter",
          "parameter": {
            "encoding": "UTF-8",
            "print": true
          }
        }
      }
    ],
    "setting": {
      "speed": {
        "channel": 5
       }
    }
  }
}
```

3.2 案例2(mysql--->hdfs)

mysql--->hive过程类似，只不过把导出的目录改成hive的数据存放目录就行。

描述:mysql reader----> hdfs writer

```
[root@hadoop01 datax]# vi ./job/mysql2hdfs.json
内容如下:
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "column": [
                            "id",
                            "name"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": ["jdbc:mysql://mini1:3306/test"],
```

```
                                "table": ["aa"]
                            }
                        ],
                        "password": "root",
                        "username": "root"
                    }
                },
                "writer": {
                    "name": "hdfswriter",
                    "parameter": {
                        "defaultFS": "hdfs://mini1:9000",
                        "fileType": "orc",
                        "path": "/datax/",
                        "fileName": "testdatax",
                        "column": [
                            {
                                "name": "col1",
                                "type": "INT"
                            },
                            {
                                "name": "col2",
                                "type": "STRING"
                            }
                        ],
                        "writeMode": "append",
                        "fieldDelimiter": "\t",
                        "compress":"NONE"
                    }
                }
            }
        ],
        "setting": {
            "speed": {
                "channel": "1"
            }
        }
    }
}

注:
运行前，需提前创建好输出目录:
```

注意:
如果想把导到hdfs的文件，当作hive表的原文件。
hdfswrite中column的值要和hive表列名，类型一致

3.3 案例3(hdfs--->mysql)

描述:hdfs reader----> mysql writer

首先是测试一下hdfs--->stream

```
[root@hadoop01 datax]# vi ./job/hdfs2mysql.json
内容如下:

{
    "job": {
        "setting": {
            "speed": {
                "channel": 1
            }
        },
        "content": [
            {
                "reader": {
                    "name": "hdfsreader",
                    "parameter": {
                        "path": "/datax/*",
                        "defaultFS": "hdfs://qf",
                         "hadoopConfig":{
                        "dfs.nameservices": "qf",
                        "dfs.ha.namenodes.qf": "nn1,nn2",
                        "dfs.namenode.rpc-address.qf.nn1": "mini1:9000",
                        "dfs.namenode.rpc-address.qf.nn2": "mini2:9000",
                        "dfs.client.failover.proxy.provider.qf":
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
                        },
                        "fileType": "textfile",
                        "encoding": "UTF-8",
                        "fieldDelimiter": ","
                    }

                },
                "writer": {
                    "name": "streamwriter",
                    "parameter": {
                        "print": true
                    }
                }
            }
        ]
    }
}


============================================================
{
    "job": {
        "content": [
            {

                "reader": {
                    "name": "hdfsreader",
                    "parameter": {
                        "path": "/datax/*",
```

```
                        "defaultFS": "hdfs://qf/",
                        "hadoopConfig":{
                        "dfs.nameservices": "qf",
                        "dfs.ha.namenodes.qf": "nn1,nn2",
                        "dfs.namenode.rpc-address.qf.nn1": "mini1:9000",
                        "dfs.namenode.rpc-address.qf.nn2": "mini2:9000",
                        "dfs.client.failover.proxy.provider.qf":
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
                        },
                        "column":[
                         {
                                "index": 0,
                                "type": "long"
                         },
                        {
                                "index": 1,
                                "type": "string"
                         }
                        ],
                        "fileType": "orc",
                        "encoding": "UTF-8",
                        "fieldDelimiter": ","
                    }

                },
                "writer": {
                    "name": "mysqlwriter",
                    "parameter": {
                        "column": [
                        "id",
                        "name"
                        ],
                        "connection": [
                            {
                                "jdbcUrl": "jdbc:mysql://hadoop01:3306/test",
                                "table": ["stu1"]
                            }
                        ],
                        "password": "root",
                        "username": "root"
                    }
                }
            }
        ],
        "setting": {
            "speed": {
                "channel": "1"
            }
        }
    }
}
```

```
注:
运行前，需提前创建好输出的stu1表:
CREATE TABLE `stu1` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(32) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

3.4案例4（hive--->mysql）

```
```

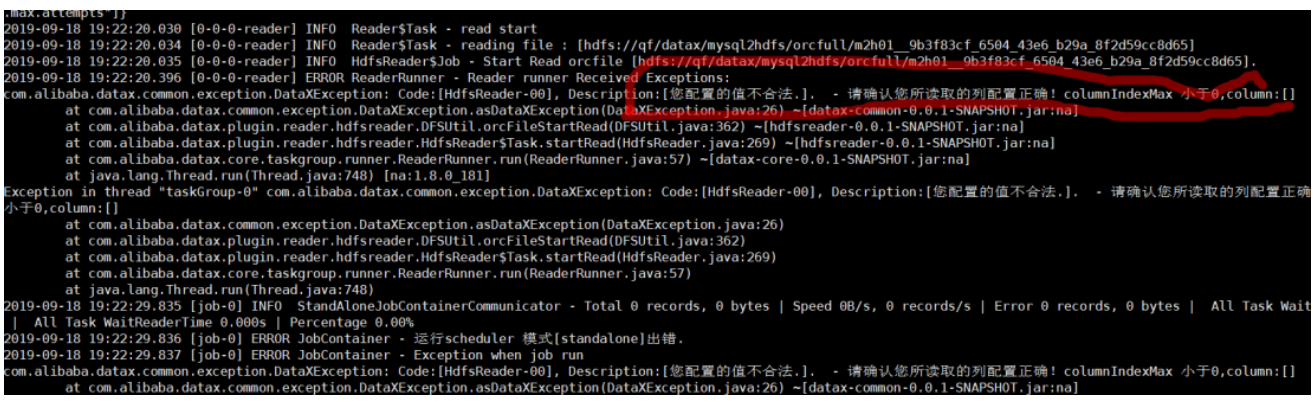
3.5案例5（mysql--->hive）

```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 3
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0.02
            }
        },
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "username": "root",
                        "password": "123456",
                        "column": [
                            "id",
                            "name"
                        ],
                        "splitPk": "id",
                        "connection": [
                            {
                                "table": [
                                    "hive2mysql"
                                ],
                                "jdbcUrl": [
                                    "jdbc:mysql://hadoop01/datax"
                                ]
                            }
                        ]
                    }
                },
```

```
                    "writer": {
                        "name": "hdfswriter",
                        "parameter": {
                            "defaultFS": "hdfs://yuli",
                            "fileType": "textfile",
                            "path": "/user/hive/warehouse2/day18homework.db/datax3",
                            "fileName": "1.dat",
                            "column": [
                                {
                                    "name": "id",
                                    "type": "int"
                                },
                                {
                                    "name": "name",
                                    "type": "STRING"
                                }
                            ],
                            "writeMode": "append",
                            "fieldDelimiter": "\t",
                            "hadoopConfig": {
                                "dfs.nameservices": "yuli",
                                "dfs.ha.namenodes.yuli": "nn1,nn2",
                                "dfs.namenode.rpc-address.yuli.nn1": "hadoop01:9000",
                                "dfs.namenode.rpc-address.yuli.nn2": "hadoop02:9000",
                                "dfs.client.failover.proxy.provider.yuli":
  "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
                            },
                            "compress":"NONE"
                        }
                    }
                }
            ]
        }
    }
```

遇到的问题

1.columnIndexMax小于0



原因：

column是所有字段全为string的时候才可以用。

- column
  - 描述：读取字段列表，type指定源数据的类型，index指定当前列来自于文本第几列(以0开始)，value指定当前类型为常量，不从源头文件读取数据，而是根据value值自动生成对应的列。

    默认情况下，用户可以全部按照String类型读取数据，配置如下：

    ```
    "column": ["*"]
    ```

    用户可以指定Column字段信息，配置如下：

    ```
    ```

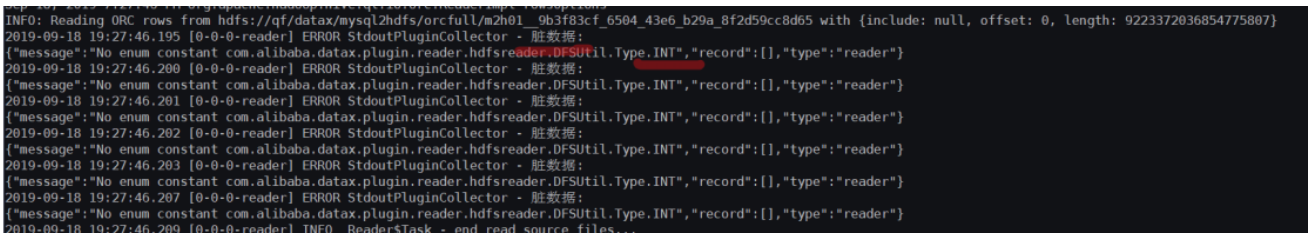
2.脏数据问题

{ "type": "long", "index": 0 //从本地文件文本第一列获取int字段 }, { "type": "string", "value": "alibaba" //HdfsReader内部生成alibaba的字符串字段作为当前字段 } ```

```
        对于用户指定Column信息，type必须填写，index/value必须选择其一。

* 必选：是 <br />

* 默认值：全部按照string类型读取 <br />
```



原因：

int类型写成long

3.3 类型转换

目前 HdfsWriter 支持大部分 Hive 类型，请注意检查你的类型。

下面列出 HdfsWriter 针对 Hive 数据类型转换列表:

DataX 内部类型	HIVE 数据类型
Long	TINYINT,SMALLINT,INT,BIGINT
Double	FLOAT,DOUBLE
String	STRING,VARCHAR,CHAR
Boolean	BOOLEAN
Date	DATE,TIMESTAMP