Flume15 自定义Source

> Source 是负责接收数据到 Flume Agent 的组件。Source 组件可以处理各种类型、各种格式的日志数据，包括 avro、thrift、exec、jms、spooling directory、netcat、sequence generator、syslog、http、legacy。官方提供的 source 类型已经很多，但是有时候并不能满足实际开发当中的需求，此时我们就需要根据实际需求自定义某些 Source。如：实时监控 MySQL，从 MySQL 中获取数据传输到 HDFS 或者其他存储框架，所以此时需要我们自己实现 MySQLSource。

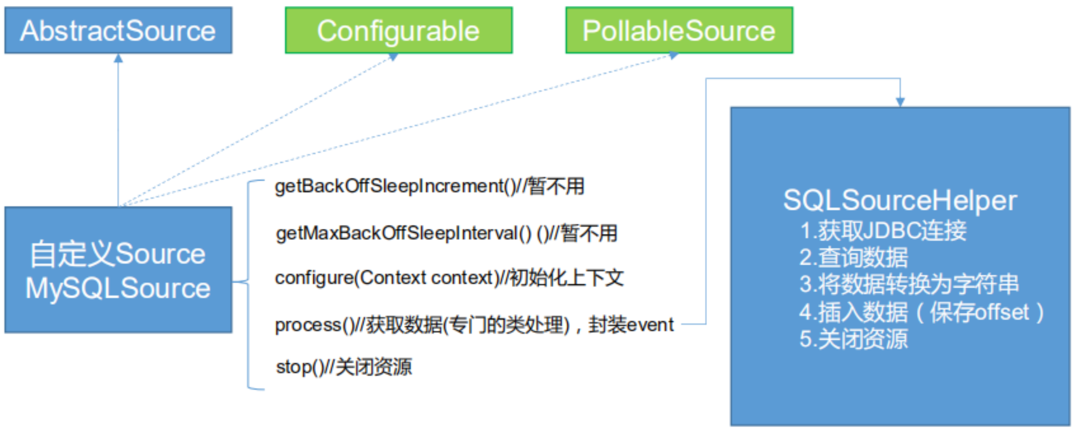
官方也提供了自定义 source 的接口：
官网说明：https://flume.apache.org/FlumeDeveloperGuide.html#source

自定义 Source简单版

自定义MySQLSource步骤

根据官方说明自定义 MySqlSource 需要继承 AbstractSource 类并实现Configurable 和PollableSource 接口。实现相应方法：

- getBackOffSleepIncrement()//暂不用
- getMaxBackOffSleepInterval()//暂不用
- configure(Context context)//初始化 context
- process()//获取数据（从 MySql 获取数据，业务处理比较复杂，所以我们定义一个专门的类——SQLSourceHelper 来处理跟 MySql 的交互），封装成 Event 并写入 Channel，这个方法被循环调用
- stop()//关闭相关的资源



代码实现

- 导入 Pom 依赖

```
<dependencies>
    <dependency>
        <groupId>org.apache.flume</groupId>
        <artifactId>flume-ng-core</artifactId>
        <version>1.7.0</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.27</version>
    </dependency>
</dependencies>
```

- **添加配置信息**

  在 ClassPath 下添加 jdbc.properties 和 log4j. properties

  jdbc.properties:

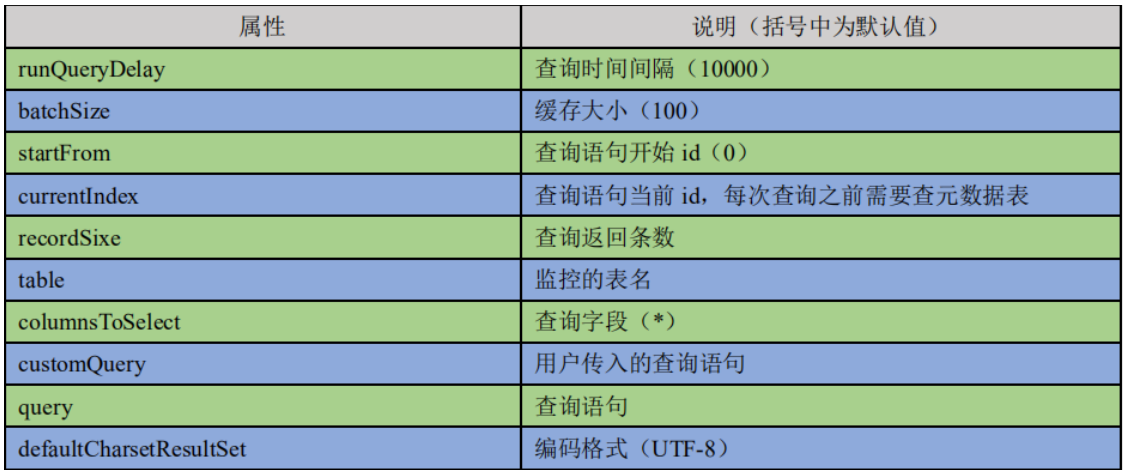
```
dbDriver=com.mysql.jdbc.Driver
dbUrl=jdbc:mysql://hadoop102:3306/mysqlsource?useUnicode=true&
characterEncoding=utf-8
dbUser=root
dbPassword=000000
```

  log4j. properties:

```
#--------console-----------
log4j.rootLogger=info,myconsole,myfile
log4j.appender.myconsole=org.apache.log4j.ConsoleAppender
log4j.appender.myconsole.layout=org.apache.log4j.SimpleLayout
#log4j.appender.myconsole.layout.ConversionPattern =%d [%t] %-5p
[%c] - %m%n
#log4j.rootLogger=error,myfile
log4j.appender.myfile=org.apache.log4j.DailyRollingFileAppender
log4j.appender.myfile.File=/tmp/flume.log
log4j.appender.myfile.layout=org.apache.log4j.PatternLayout
log4j.appender.myfile.layout.ConversionPattern =%d [%t] %-5p [%c]
- %m%n
```

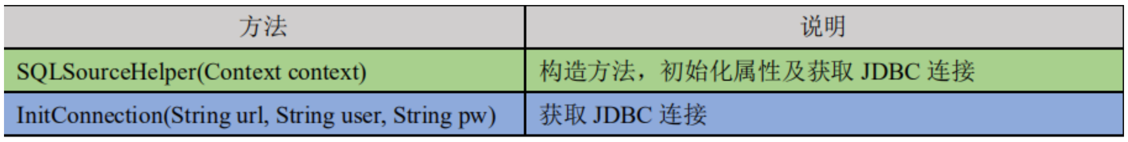
- SQLSourceHelper

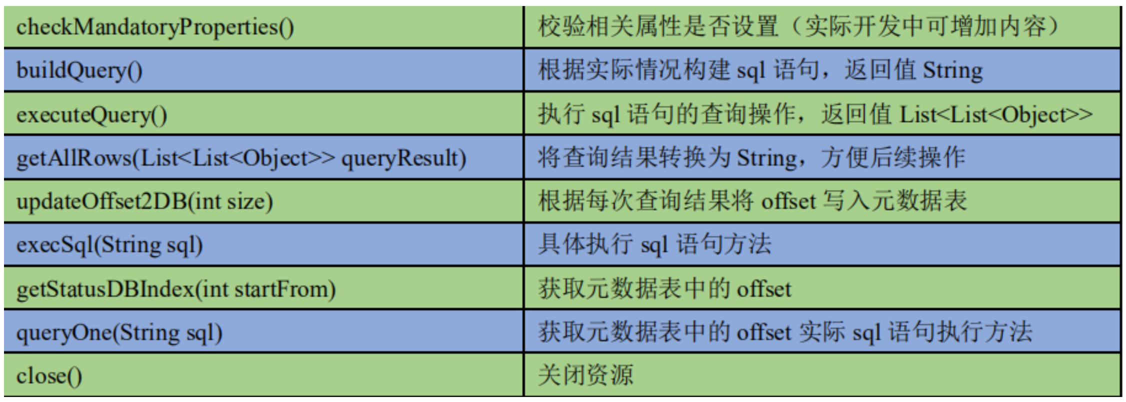
属性说明：



属性	说明（括号中为默认值）
runQueryDelay	查询时间间隔（10000）
batchSize	缓存大小（100）
startFrom	查询语句开始 id（0）
currentIndex	查询语句当前 id，每次查询之前需要查元数据表
recordSixe	查询返回条数
table	监控的表名
columnsToSelect	查询字段（*）
customQuery	用户传入的查询语句
query	查询语句
defaultCharsetResultSet	编码格式（UTF-8）

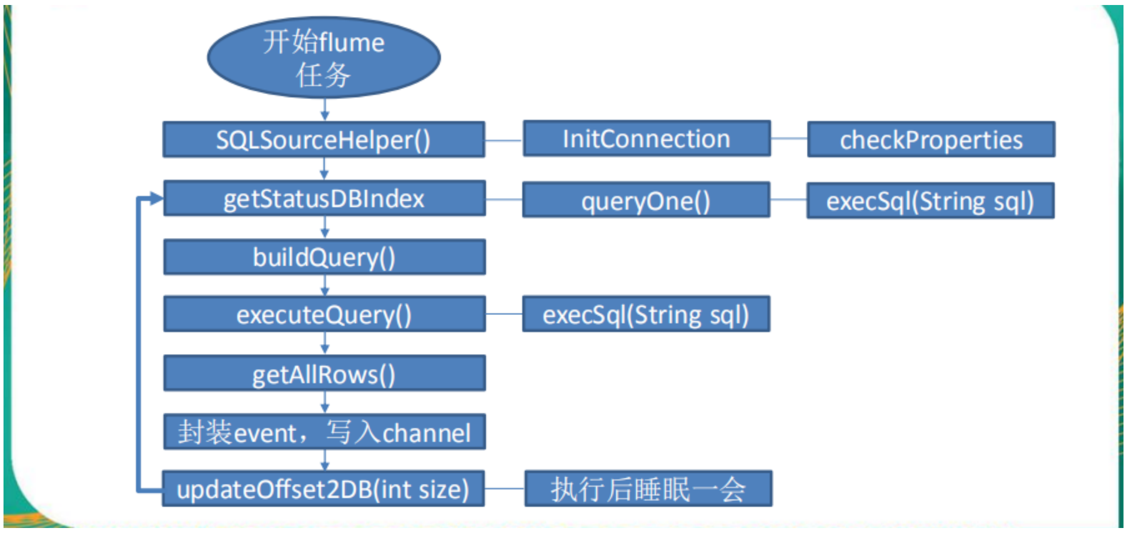
方法说明：





方法	说明
SQLSourceHelper(Context context)	构造方法，初始化属性及获取 JDBC 连接
InitConnection(String url, String user, String pw)	获取 JDBC 连接
checkMandatoryProperties()	校验相关属性是否设置（实际开发中可增加内容）
buildQuery()	根据实际情况构建 sql 语句，返回值 String
executeQuery()	执行 sql 语句的查询操作，返回值 List<List<Object>>
getAllRows(List<List<Object>> queryResult)	将查询结果转换为 String，方便后续操作
updateOffset2DB(int size)	根据每次查询结果将 offset 写入元数据表
execSql(String sql)	具体执行 sql 语句方法
getStatusDBIndex(int startFrom)	获取元数据表中的 offset
queryOne(String sql)	获取元数据表中的 offset 实际 sql 语句执行方法
close()	关闭资源

代码分析



- 代码实现

```
package com.atguigu.source;

import org.apache.flume.Context;
import org.apache.flume.conf.ConfigurationException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.IOException;

import java.sql.*;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

public class SQLSourceHelper {
    private static final Logger LOG =
LoggerFactory.getLogger(SQLSourceHelper.class);
    private int runQueryDelay, //两次查询的时间间隔
        startFrom, //开始 id
        currentIndex, //当前 id
        recordSixe = 0, //每次查询返回结果的条数
        maxRow; //每次查询的最大条数

    private String table, //要操作的表
        columnsToSelect, //用户传入的查询的列
        customQuery, //用户传入的查询语句
        query, //构建的查询语句
        defaultCharsetResultSet;//编码集

    //上下文，用来获取配置文件
    private Context context;
    //为定义的变量赋值（默认值），可在 flume 任务的配置文件中修改
    private static final int DEFAULT_QUERY_DELAY = 10000;
    private static final int DEFAULT_START_VALUE = 0;
    private static final int DEFAULT_MAX_ROWS = 2000;
    private static final String DEFAULT_COLUMNS_SELECT = "*";
    private static final String DEFAULT_CHARSET_RESULTSET =
"UTF-8";
    private static Connection conn = null;
    private static PreparedStatement ps = null;
    private static String connectionURL, connectionUserName,
connectionPassword;
    //加载静态资源
    static {
    Properties p = new Properties();
        try {
            p.load(SQLSourceHelper.class.getClassLoader().getResourceAsStr
eam("jdbc.properties"));
            connectionURL = p.getProperty("dbUrl");
            connectionUserName = p.getProperty("dbUser");
            connectionPassword = p.getProperty("dbPassword");
            Class.forName(p.getProperty("dbDriver"));
        } catch (IOException | ClassNotFoundException e) {
            LOG.error(e.toString());
        }
    }

    //获取 JDBC 连接
    private static Connection InitConnection(String url, String
user, String pw) {

        try {
            Connection conn = DriverManager.getConnection(url,
user, pw);
            if (conn == null)
            throw new SQLException();
            return conn;
        } catch (SQLException e) {

            e.printStackTrace();
        }

        return null;
    }

    //构造方法
    SQLSourceHelper(Context context) throws ParseException {

        //初始化上下文
        this.context = context;
        //有默认值参数：获取 flume 任务配置文件中的参数，读不到的采用默认值
        this.columnsToSelect = context.getString("columns.to.select",
DEFAULT_COLUMNS_SELECT);
        this.runQueryDelay = context.getInteger("run.query.delay",
DEFAULT_QUERY_DELAY);
        this.startFrom = context.getInteger("start.from",
DEFAULT_START_VALUE);
        this.defaultCharsetResultSet =
context.getString("default.charset.resultset", DEFAULT_CHARSET_RESULTSET);

        //无默认值参数：获取 flume 任务配置文件中的参数
        this.table = context.getString("table");
        this.customQuery = context.getString("custom.query");
        connectionURL = context.getString("connection.url");
        connectionUserName = context.getString("connection.user");
        connectionPassword = context.getString("connection.password");
        conn = InitConnection(connectionURL, connectionUserName,
connectionPassword);
```

```
        //校验相应的配置信息，如果没有默认值的参数也没赋值，抛出异常
        checkMandatoryProperties();
         //获取当前的 id
        currentIndex = getStatusDBIndex(startFrom);
        //构建查询语句
        query = buildQuery();
    }

    //校验相应的配置信息（表，查询语句以及数据库连接的参数）
    private void checkMandatoryProperties() {
        if (table == null) {
            throw new ConfigurationException("property table not set");
        }

        if (connectionURL == null) {
            throw new ConfigurationException("connection.url property not
set");
        }

        if (connectionUserName == null) {
            throw new ConfigurationException("connection.user property not
set");
        }

        if (connectionPassword == null) {throw new
ConfigurationException("connection.password property not set");
        }
    }

    //构建 sql 语句
    private String buildQuery() {
        String sql = "";
        //获取当前 id
        currentIndex = getStatusDBIndex(startFrom);
        LOG.info(currentIndex + "");
        if (customQuery == null) {
            sql = "SELECT " + columnsToSelect + " FROM " + table;
        } else {
            sql = customQuery;
        }
        StringBuilder execSql = new StringBuilder(sql);
        //以 id 作为 offset
        if (!sql.contains("where")) {
            execSql.append(" where ");

            execSql.append("id").append(">").append(currentIndex);
            return execSql.toString();
        } else {
            int length = execSql.toString().length();
            return execSql.toString().substring(0, length -
String.valueOf(currentIndex).length()) + currentIndex;
        }
    }

    //执行查询
    List<List<Object>> executeQuery() {
        try {
            //每次执行查询时都要重新生成 sql，因为 id 不同
            customQuery = buildQuery();
            //存放结果的集合
            List<List<Object>> results = new ArrayList<>();
            if (ps == null) {
            //
                ps = conn.prepareStatement(customQuery);
            }
            ResultSet result = ps.executeQuery(customQuery);
            while (result.next()) {
                //存放一条数据的集合（多个列）
                List<Object> row = new ArrayList<>();
                //将返回结果放入集合
                for (int i = 1; i <= result.getMetaData().getColumnCount();
i++) {
                    row.add(result.getObject(i));
                }
                results.add(row);
            }
            LOG.info("execSql:" + customQuery + "\nresultSize:" +
results.size());
            return results;
        } catch (SQLException e) {
            LOG.error(e.toString());
            // 重新连接
            conn = InitConnection(connectionURL, connectionUserName,
connectionPassword);
         }
        return null;
    }

    //将结果集转化为字符串，每一条数据是一个 list 集合，将每一个小的 list集合转化为字符
串
    List<String> getAllRows(List<List<Object>> queryResult) {
        List<String> allRows = new ArrayList<>();
        if (queryResult == null || queryResult.isEmpty())
            return allRows;
        StringBuilder row = new StringBuilder();
        for (List<Object> rawRow : queryResult) {
            Object value = null;
            for (Object aRawRow : rawRow) {
                value = aRawRow;
                if (value == null) {
                     row.append(",");
                } else {
                    row.append(aRawRow.toString()).append(",");
                }
            }
        allRows.add(row.toString());
        row = new StringBuilder();
         }
        return allRows;
    }

    //更新 offset 元数据状态，每次返回结果集后调用。必须记录每次查询的offset 值，为程序
中断续跑数据时使用，以 id 为 offset
    void updateOffset2DB(int size) {

        //以 source_tab 做为 KEY，如果不存在则插入，存在则更新（每个源表对应一条记录）
        String sql = "insert into flume_meta(source_tab,currentIndex)
VALUES('"
 + this.table + "','" + (recordSixe += size) + "') on DUPLICATE key update
source_tab=values(source_tab),currentIndex=values(currentIndex)";

        LOG.info("updateStatus Sql:" + sql);
 execSql(sql);
    }

    //执行 sql 语句
    private void execSql(String sql) {
        try {
            ps = conn.prepareStatement(sql);
            LOG.info("exec::" + sql);
            ps.execute();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //获取当前 id 的 offset
    private Integer getStatusDBIndex(int startFrom) {
        //从 flume_meta 表中查询出当前的 id 是多少
        String dbIndex = queryOne("select currentIndex from flume_meta where
source_tab='" + table + "'");
        if (dbIndex != null) {
             return Integer.parseInt(dbIndex);
        }
        //如果没有数据，则说明是第一次查询或者数据表中还没有存入数据，返回最初传入的值
        return startFrom;
    }

    //查询一条数据的执行语句(当前 id)
    private String queryOne(String sql) {
        ResultSet result = null;
        try {
            ps = conn.prepareStatement(sql);
            result = ps.executeQuery();
            while (result.next()) {
                return result.getString(1);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    //关闭相关资源
    void close() {
        try {
            ps.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    int getCurrentIndex() {
        return currentIndex;
    }

    void setCurrentIndex(int newValue) {
        currentIndex = newValue;
    }

    int getRunQueryDelay() {
        return runQueryDelay;
    }

    String getQuery() {
        return query;
    }

    String getConnectionURL() {
        return connectionURL;
    }

    private boolean isCustomQuerySet() {
        return (customQuery != null);
    }

    Context getContext() {
        return context;
    }

    public String getConnectionUserName() {
         return connectionUserName;
    }

    public String getConnectionPassword() {
        return connectionPassword;
    }

    String getDefaultCharsetResultSet() {
        return defaultCharsetResultSet;
    }
}
```

- **MySQLSource**

  代码实现：

```
package com.atguigu.source;

import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.EventDeliveryException;
import org.apache.flume.PollableSource;
import org.apache.flume.conf.Configurable;
import org.apache.flume.event.SimpleEvent;
import org.apache.flume.source.AbstractSource;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.text.ParseException;
```

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class SQLSource extends AbstractSource implements Configurable,
PollableSource {
    //打印日志
    private static final Logger LOG =
LoggerFactory.getLogger(SQLSource.class);
     //定义 sqlHelper
    private SQLSourceHelper sqlSourceHelper;

    @Override
    public long getBackOffSleepIncrement() {
        return 0;
    }

    @Override
    public long getMaxBackOffSleepInterval() {
        return 0;
    }

    @Override
    public void configure(Context context) {
        try {
            //初始化
            sqlSourceHelper = new SQLSourceHelper(context);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }

    @Override
    public Status process() throws EventDeliveryException {
        try {
            //查询数据表
            List<List<Object>> result = sqlSourceHelper.executeQuery();

            //存放 event 的集合
            List<Event> events = new ArrayList<>();
            //存放 event 头集合
            HashMap<String, String> header = new HashMap<>();
            //如果有返回数据，则将数据封装为 event
            if (!result.isEmpty()) {
                List<String> allRows = sqlSourceHelper.getAllRows(result);
                Event event = null;
                for (String row : allRows) {
                    event = new SimpleEvent();
                    event.setBody(row.getBytes());
                    event.setHeaders(header);
                    events.add(event);
                }

                //将 event 写入 channel
                this.getChannelProcessor().processEventBatch(events);
                //更新数据表中的 offset 信
                sqlSourceHelper.updateOffset2DB(result.size());
            }
            //等待时长
            Thread.sleep(sqlSourceHelper.getRunQueryDelay());
            return Status.READY;
        } catch (InterruptedException e) {
            LOG.error("Error procesing row", e);
            return Status.BACKOFF;
        }
    }

    @Override
    public synchronized void stop() {
        LOG.info("Stopping sql source {} ...", getName());
        try {
            //关闭资源
            sqlSourceHelper.close();
        } finally {
            super.stop();
        }
    }
}
```

- **测试**

Jar 包准备

将 MySql 驱动包放入 Flume 的 lib 目录下

```
[atguigu@hadoop102 flume]$ cp \
/opt/sorfware/mysql-libs/mysql-connector-java-5.1.27/mysql-con
nector-java-5.1.27-bin.jar \
/opt/module/flume/lib/
```

打包项目并将 Jar 包放入 Flume 的 lib 目录下

配置文件准备

创建配置文件并打开

```
[atguigu@hadoop102 job]$ touch mysql.conf
[atguigu@hadoop102 job]$ vim mysql.conf
```

添加如下内容：

```
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# Describe/configure the source
a1.sources.r1.type = com.atguigu.source.SQLSource
a1.sources.r1.connection.url =
jdbc:mysql://192.168.9.102:3306/mysqlsource
a1.sources.r1.connection.user = root
a1.sources.r1.connection.password = 000000
a1.sources.r1.table = student
a1.sources.r1.columns.to.select = *
#a1.sources.r1.incremental.column.name = id
#a1.sources.r1.incremental.value = 0
a1.sources.r1.run.query.delay=5000
```

```
# Describe the sink
a1.sinks.k1.type = logger
# Describe the channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

MySql 表准备

创建 MySqlSource 数据库

```
CREATE DATABASE mysqlsource;
```

在 MySqlSource 数据库下创建数据表 Student 和元数据表 Flume_meta

```
CREATE TABLE `student` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` varchar(255) NOT NULL,
PRIMARY KEY (`id`)
);
CREATE TABLE `flume_meta` (
`source_tab` varchar(255) NOT NULL,
`currentIndex` varchar(255) NOT NULL,
PRIMARY KEY (`source_tab`)
);
```

向数据表中添加数据

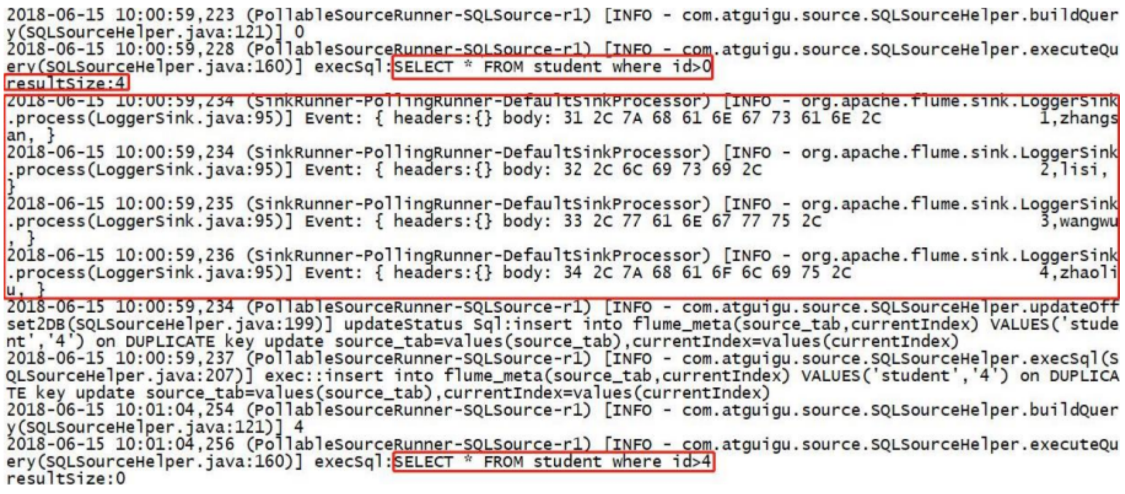
```
1 zhangsan
2 lisi
3 wangwu
4 zhaoliu
```

- **测试并查看结果**

  任务执行

```
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name
a1 \
--conf-file job/mysql.conf -Dflume.root.logger=INFO,console
```

结果展示，如图 所示



```
2018-06-15 10:00:59,223 (PollableSourceRunner-SQLSource-r1) [INFO - com.atguigu.source.SQLSourceHelper.buildQuer
y(SQLSourceHelper.java:121)] 0
2018-06-15 10:00:59,228 (PollableSourceRunner-SQLSource-r1) [INFO - com.atguigu.source.SQLSourceHelper.executeQu
ery(SQLSourceHelper.java:160)] execSql:SELECT * FROM student where id>0
resultSize:4
2018-06-15 10:00:59,234 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink
.process(LoggerSink.java:95)] Event: { headers:{} body: 31 2C 7A 68 61 6E 67 73 61 6E 2C                1,zhangs
an, }
2018-06-15 10:00:59,234 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink
.process(LoggerSink.java:95)] Event: { headers:{} body: 32 2C 6C 69 73 69 2C                            2,lisi,
}
2018-06-15 10:00:59,235 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink
.process(LoggerSink.java:95)] Event: { headers:{} body: 33 2C 77 61 6E 67 77 75 2C                      3,wangwu
, }
2018-06-15 10:00:59,236 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink
.process(LoggerSink.java:95)] Event: { headers:{} body: 34 2C 7A 68 61 6F 6C 69 75 2C                   4,zhaoli
u, }
2018-06-15 10:00:59,234 (PollableSourceRunner-SQLSource-r1) [INFO - com.atguigu.source.SQLSourceHelper.updateOff
set2DB(SQLSourceHelper.java:199)] updateStatus Sql:insert into flume_meta(source_tab,currentIndex) VALUES('stude
nt','4') on DUPLICATE key update source_tab=values(source_tab),currentIndex=values(currentIndex)
2018-06-15 10:00:59,237 (PollableSourceRunner-SQLSource-r1) [INFO - com.atguigu.source.SQLSourceHelper.execSql(S
QLSourceHelper.java:207)] exec::insert into flume_meta(source_tab,currentIndex) VALUES('student','4') on DUPLICA
TE key update source_tab=values(source_tab),currentIndex=values(currentIndex)
2018-06-15 10:01:04,254 (PollableSourceRunner-SQLSource-r1) [INFO - com.atguigu.source.SQLSourceHelper.buildQuer
y(SQLSourceHelper.java:121)] 4
2018-06-15 10:01:04,256 (PollableSourceRunner-SQLSource-r1) [INFO - com.atguigu.source.SQLSourceHelper.executeQu
ery(SQLSourceHelper.java:160)] execSql:SELECT * FROM student where id>4
resultSize:0
```