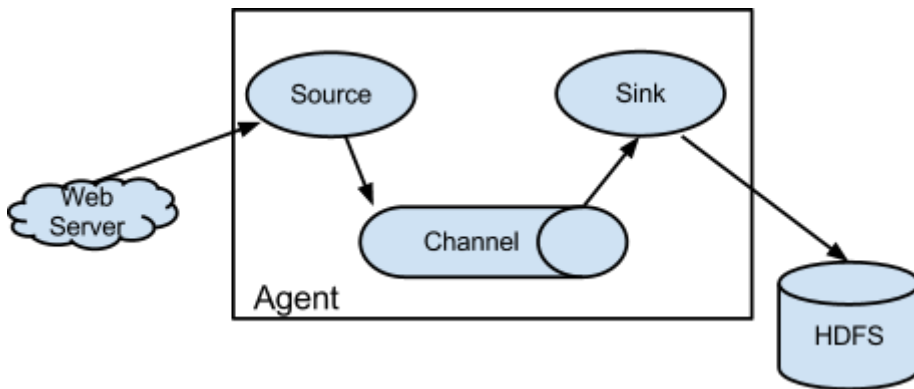


1.基本介绍

- flume是cloudera公司开发，用的是Java，一款专门进行数据采集的工具，是apache的顶级项目
- flume为了支持丰富的使用场景，满足各种各样需求，flume分成三大类的组件
 - source，数据从哪些地方获取
 - sink，目的地数据最终发送哪里
 - channel 缓存，连接source和sink
- flume版本，flume OG指的是0.9之前，flume NG，当前的版本，flume捐给apache后架构变动很大
- flume一旦启动，启动的一个采集程序，这个程序就被称为agent（flume实例），agent一般运行在数据所在的节点
- 一个节点可以启动多个agent



- 整个agent中数据是一条条进行传输的，flume会把数据封装成event格式的对象，event中处理存放数据还会放置一些自定义的东西
- source组件

- Flume Sources
 - Avro Source
 - Thrift Source
 - Exec Source
 - JMS Source
 - JMS message converter
 - SSL and JMS Source
 - Spooling Directory Source
 - Event Deserializers

网络端口

命令结果

监听目录

Deserializers

- LINE
- AVRO
- BlobDeserializer
- Taildir Source
- Twitter 1% firehose Source (experimental)
- Kafka Source
- NetCat TCP Source
- NetCat UDP Source
- Sequence Generator Source
- Syslog Sources
 - Syslog TCP Source
 - Multiport Syslog TCP Source
 - Syslog UDP Source
- HTTP Source

- taildir 监听目录和文件
- kafka 监听kafka主题
- netcat 监听网络端口
- channel组件，一般memchannel缓存在内存中
- sink组件
-

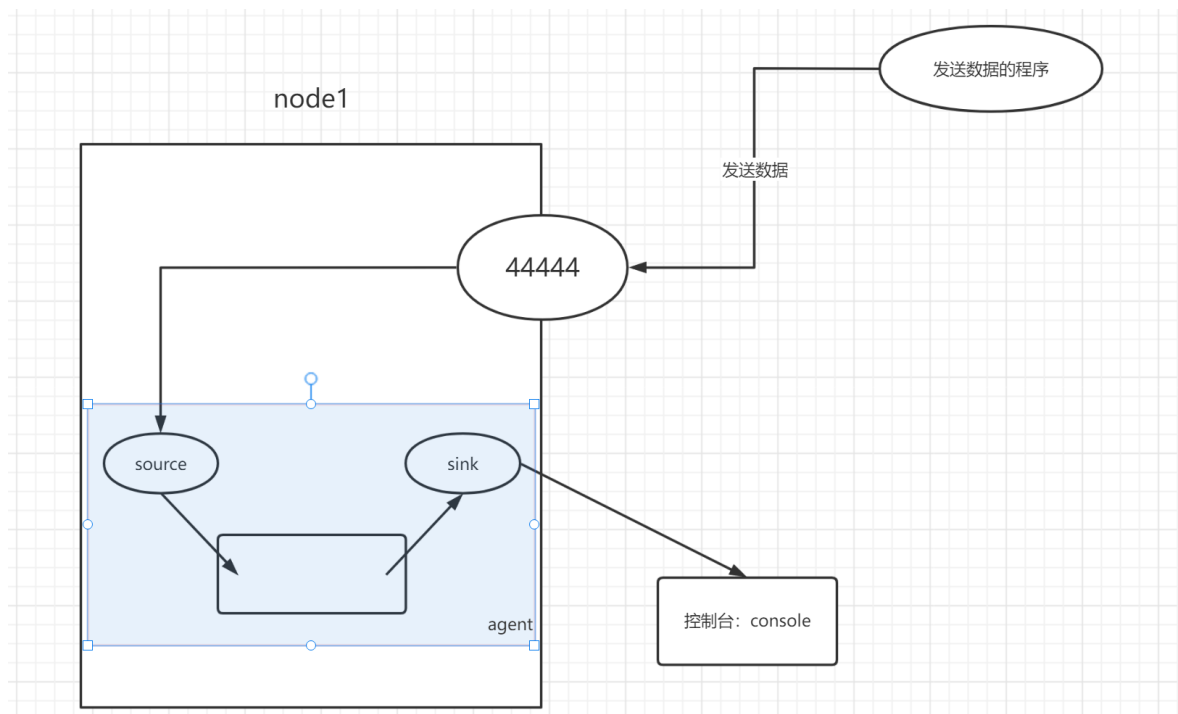
Flume Sinks

- HDFS Sink
- Hive Sink
- Logger Sink
- Avro Sink
- Thrift Sink
- IRC Sink
- File Roll Sink
- Null Sink
- HBaseSinks
 - HBaseSink
 - HBase2Sink
 - AsyncHBaseSink
- MorphlineSolrSink
- ElasticSearchSink
- Kite Dataset Sink
- Kafka Sink
- HTTP Sink
- Custom Sink

- sqoop更多的导入的是结构化数据，MySQL->hive，hive->MySQL，数据时迁移
- flume不限定数据结构

2.入门操作

- 拓扑结构



- 确定配置文件

• source组件

- 1 第一步确定三个组件类型
- 2 确定组件类型之前，一定要确定网络结构
- 3 source组件
- 4 功能：监听某一个端口
- 5 组件类型：netcat tcp source
- 6 实例
- 7 `a1.sources = r1`
- 8 #类型为netcat
- 9 `a1.sources.r1.type = netcat`
- 10 #连接的机器
- 11 `a1.sources.r1.bind = 0.0.0.0`
- 12 #监听端口
- 13 `a1.sources.r1.port = 6666`
- 14 #建立与channel连接
- 15 `a1.sources.r1.channels = c1`

• channel组件

- 1 功能：缓存
- 2 组件类型：memroy channel
- 3 实例：
- 4 `a1.channels = c1`
- 5 `a1.channels.c1.type = memory`

• sink组件

```
1 功能：把数据打印在控制台上
2 组件名称：logger sink
3 a1.sinks = k1
4 a1.sinks.k1.type = logger
5 a1.sinks.k1.channel = c1
```

• 示例

```
1 第二步根据需求对组件配置进行调整
2 source组件
3
4 a1.sources = r1
5 a1.sources.r1.type = netcat
6 a1.sources.r1.bind = node1
7 a1.sources.r1.port = 44444
8 a1.sources.r1.channels = c1
9
10 channel组件
11 a1.channels = c1
12 a1.channels.c1.type = memory
13
14
15
16 sink组件
17 a1.sinks = k1
18 a1.sinks.k1.type = logger
19 a1.sinks.k1.channel = c1
20
21 第三步把汇总写入文件
22 /export/server/apache-flume-1.9.0-bin/conf/intro_netcat_source_logger_sink.conf
23
24 文件名望文知义，文件中不能有中文注释
25
26
27 完整内容
28
29 a1.sources = r1
30 a1.sources.r1.type = netcat
```

```

31 a1.sources.r1.bind = node1
32 a1.sources.r1.port = 44444
33 a1.sources.r1.channels = c1
34
35 a1.channels = c1
36 a1.channels.c1.type = memory
37
38
39
40 a1.sinks = k1
41 a1.sinks.k1.type = logger
42 a1.sinks.k1.channel = c1

```

• 启动监听

```

1 flume-ng agent -c conf -f
  /export/server/flume/conf/intro_netcat_source_logger_sink.conf -n a1 -
  Dflume.root.logger=INFO,console
2
3 参数说明：
4     -c conf    指定flume自身的配置文件所在目录
5     -f conf/netcat-logger.conf 指定我们所描述的采集方案
6     -n a1     指定我们这个agent的名字

```

```

2022-03-14 14:50:27,298 INFO node.Application: Starting Channel c1
2022-03-14 14:50:27,299 INFO node.Application: Waiting for channel: c1 to start. Sleeping for 500 ms
2022-03-14 14:50:27,393 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: CHANNEL, name: c1: Success
fully registered new MBean.
2022-03-14 14:50:27,394 INFO instrumentation.MonitoredCounterGroup: Component type: CHANNEL, name: c1 started
2022-03-14 14:50:27,799 INFO node.Application: Starting Sink k1
2022-03-14 14:50:27,800 INFO node.Application: Starting Source r1
2022-03-14 14:50:27,801 INFO source.NetcatSource: Source starting
2022-03-14 14:50:27,823 INFO source.NetcatSource: Created serverSocket:sun.nio.ch.ServerSocketChannelImpl[192.168.88.161:44444]

```

• 启动一个发送数据的程序

```

1 启动telnet
2
3 [root@node1 ~]# telnet
4 -bash: telnet: command not found
5 [root@node1 ~]# yum install telnet
6 需要手动安装telnet
7
8 输入: telnet node1 44444
9 即可连接到node1的44444端口，就可以发送数据

```

```
[root@node1 ~]# telnet node1 44444
Trying 192.168.88.161...
Connected to node1.
Escape character is '^]'.
```



输入字符，回车键发送

```
2022-03-14 14:55:13,832 INFO sink.LoggerSink: Event: { headers:
{} body: 61 62 63 0D                abc. }
2022-03-14 14:55:14,708 INFO sink.LoggerSink: Event: { headers:
{} body: 66 6C 75 6D 65 0D          flume.
}
2022-03-14 14:55:16,921 INFO sink.LoggerSink: Event: { headers:
{} body: 6B 61 66 6B 61 0D          kafka.
}
```

3.项目实例，基于陌陌消息

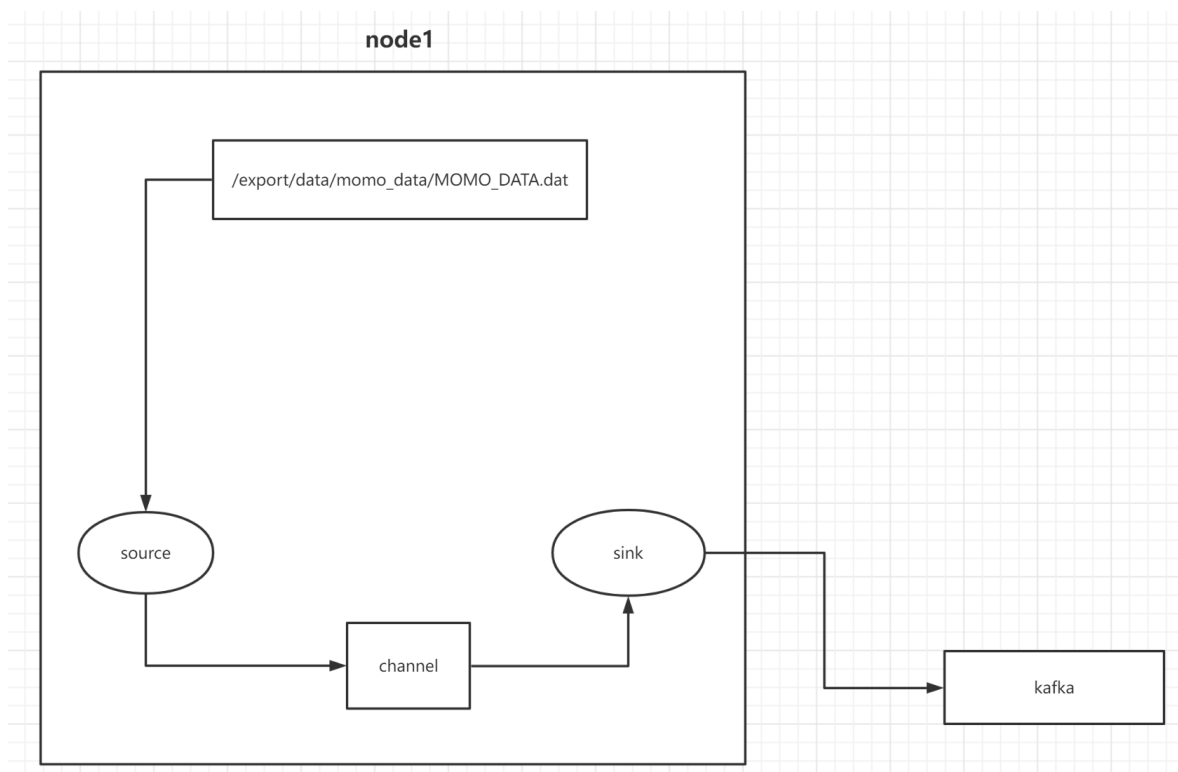


MoMo_DataGen.jar
18.8MB



MoMo_Data.xlsx
40.13KB

- 将上述两个文件上传到linux的 /export/data/momo_init 目录中
 - 没有目录就创建
 - `mkdir -p /export/data/momo_init`
 - `mkdir -p /export/data/momo_data`
- 确定拓扑结构/确定数据从哪里来，发送到哪里去



● 确定3个组件

```
1 确定三个组件
2
3 source组件
4 exec source 执行linux命令，命令输出结果 作为数据源
5 taildir source 可以监听文件和目录
6 使用taildir source
7
8
9 a1.sources = r1
10 a1.channels = c1
11 a1.sources.r1.type = TAILDIR
12 a1.sources.r1.channels = c1
13 a1.sources.r1.positionFile = /var/log/flume/taildir_position.json #断点续传文件
14 a1.sources.r1.filegroups = f1 f2
15 a1.sources.r1.filegroups.f1 = /var/log/test1/example.log
16 a1.sources.r1.headers.f1.headerKey1 = value1
17 a1.sources.r1.filegroups.f2 = /var/log/test2/*.log.*
18 a1.sources.r1.headers.f2.headerKey1 = value2
19 a1.sources.r1.headers.f2.headerKey2 = value2-2
20 a1.sources.r1.fileHeader = true
21 a1.sources.r1.maxBatchCount = 1000
22 sink组件
```



```
23 a1.sinks.k1.channel = c1
24 a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
25 a1.sinks.k1.kafka.topic = mytopic
26 a1.sinks.k1.kafka.bootstrap.servers = localhost:9092
27 a1.sinks.k1.kafka.flumeBatchSize = 20
28 a1.sinks.k1.kafka.producer.acks = 1
29 a1.sinks.k1.kafka.producer.linger.ms = 1
30 a1.sinks.k1.kafka.producer.compression.type = snappy
31
32 channel组件
33 memory channel
34 a1.channels = c1
35 a1.channels.c1.type = memory
36
37 组合
38
39
40
41 a1.sources = r1
42 a1.channels = c1
43 a1.sources.r1.type = TAILDIR
44 a1.sources.r1.channels = c1
45 a1.sources.r1.positionFile = /export/data/flume/taildir_position.json #断点续传文件
46 a1.sources.r1.filegroups = f1
47 a1.sources.r1.filegroups.f1 = /export/data/momo_data/MOMO_DATA.dat
48 #a1.sources.r1.maxBatchCount = 1000
49
50 a1.channels.c1.type = memory
51
52 a1.sinks = k1
53 a1.sinks.k1.channel = c1
54 a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
55 a1.sinks.k1.kafka.topic = MOMO_MSG
56 a1.sinks.k1.kafka.bootstrap.servers =
    node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092
57 a1.sinks.k1.kafka.flumeBatchSize = 20
58 a1.sinks.k1.kafka.producer.acks = 1
59 a1.sinks.k1.kafka.producer.linger.ms = 1
60
61
```

62 写入到文件

63

• 启动服务

- 启动zookeeper
 - 三台节点都需要启动
- 启动kafka服务
 - 三台节点都需要启动

```
1 nohup kafka-server-start.sh /export/server/kafka/config/server.properties 2>&1 &
```

- 在kafka中创建一个topic

```
1 kafka-topics.sh --create --bootstrap-server node1:9092 --topic MOMO_MSG --partitions 3 --replication-factor 2
```

- 启动一个消费者监听这个主题

```
1 kafka-console-consumer.sh --bootstrap-server node1:9092,node2:9092,node3n:9092 --topic MOMO_MSG
```

- 启动flume，监听数据

```
1 flume-ng agent -c conf -f /export/server/flume/conf/momo_taildir_source_kafka_sink.conf -n a1 -Dflume.root.logger=INFO,console
```

- 启动生产数据的jar包，如果消费者中有数据，说明通路正常

```
1 cd /export/data/momo_init/  
2 java -jar MoMo_DataGen.jar MoMo_Data.xlsx /export/data/momo_data/ 1000
```

4.写入HBase

• 1.准备工作

- 启动hbase

```
1 先启动zookeeper Hadoop集群（hdfs yarn）  
2 node1执行 start-all.sh  
3 node1执行 start-hbase.sh
```

- 创建hbase表

```
1 1 是否需要创建一个名称空间  
2 需要，MOMO_CHAT
```

```
3 2 列族怎么设计 名称叫什么
4 只有一个列族，C1
5 3 数据要不要压缩，如果需要 怎么压缩
6 数据需要压缩，用什么算法，GZ，因为读取的情况很少
7 4 是否需要对其进行预分区，如果需要，怎么做
8 需要，搞6预分区，使用hash预分区
9 5 数据是否需要设置版本和过期时间
10 默认一个版本，发出去的消息不能更改，默认长期存储
11
12
13
14 create_namespace "MOMO_CHAT"
15 create "MOMO_CHAT:MOMO_MSG", {NAME=>"C1", COMPRESSION=>"GZ"}, {NUMREGIONS=>6,
    SPLITALGO=>"HexStringSplit"}
```

2.rowkey设计

```
1 官方规定
2 1 不能使用递增或者时序数据，手机号 时间戳
3 2 rowkey设计时不要太长，推荐100Byte以内
4 3 数值类型的rowkey比字符串更节省空间
5 4 必须保证rowkey唯一
6
7 业务
8 保证相关性的数据放在一起，满足查询的需要
9
10 需求
11 根据发件人 收件人 账号 聊天时间进行查询
12
13
14 rowkey设计
15 MD5HASH值_发件人账号_收件人账号_时间戳
```

3.构建一个消费者完成数据写入到HBASE操作

- 创建Java工程导入依赖

```
1 <dependencies>
2 <dependency>
3 <groupId>org.apache.hbase</groupId>
```

```

4         <artifactId>hbase-client</artifactId>
5         <version>2.1.0</version>
6     </dependency>
7
8     <dependency>
9         <groupId>org.apache.kafka</groupId>
10        <artifactId>kafka-clients</artifactId>
11        <version>2.4.1</version>
12    </dependency>
13 </dependencies>

```

- 创建一个类MOMOKafkaToHBase

```

1  package sz.base.momo;
2
3  import org.apache.hadoop.conf.Configuration;
4  import org.apache.hadoop.hbase.HBaseConfiguration;
5  import org.apache.hadoop.hbase.TableName;
6  import org.apache.hadoop.hbase.client.*;
7  import org.apache.hadoop.hbase.util.MD5Hash;
8  import org.apache.kafka.clients.consumer.ConsumerRecord;
9  import org.apache.kafka.clients.consumer.ConsumerRecords;
10 import org.apache.kafka.clients.consumer.KafkaConsumer;
11
12 import java.io.IOException;
13 import java.text.ParseException;
14 import java.text.SimpleDateFormat;
15 import java.time.Duration;
16 import java.util.Arrays;
17 import java.util.Date;
18 import java.util.Properties;
19
20 public class MOMOKafkaToHBase {
21     //从kafka的MOMO_MSG主题中获取数据，写入到MOMO_CHAT:MOMO_MSG hbase表中
22     //1.从kafka中读取数据
23     //1.1设置相关属性
24     //1.2创建一个消费者
25     //1.3订阅主题 MOMO_MSG
26     //1.4循环读取消息
27     //2.把数据写入到hbase表中

```

```

28    //2.1连接hbase
29    //2.2获取表的管理对象
30    //2.3构造put对象，把数据写入
31    //2.4构造rowkey
32    //2.5释放资源
33    public static void main(String[] args) throws IOException, ParseException {
34        readFromKafka();
35
36    }
37
38    private static Connection hbConn;
39    private static Table table;
40
41    static {
42        //静态代码块，随着类的加载而加载，一般只执行一次
43        //2.1.建立连接，连接到hbase
44        Configuration conf = HBaseConfiguration.create();
45        conf.set("hbase.zookeeper.quorum", "node1,node2,node3");
46        //创建连接
47
48        try {
49            hbConn = ConnectionFactory.createConnection(conf);
50        } catch (IOException e) {
51            e.printStackTrace();
52        }
53        //从连接对象中获取管理对象
54        //Admin对象，对表的管理
55        //Table对象，对表的数据进行管理
56        TableName table1 = TableName.valueOf("MOMO_CHAT:MOMO_MSG");
57        try {
58            table = hbConn.getTable(table1);
59        } catch (IOException e) {
60            e.printStackTrace();
61        }
62    }
63
64
65    public static void writeToHBase(String message) throws IOException, ParseException {
66        //2.把数据写入到hbase表

```

```
67         //执行相关操作，put操作（写数据）
68         //构造put对象
69         //put命令: put rowkey 表名 ，列族: 列限定符 ,value
70         String rowkey = getRowkey(message);
71         Put put = new Put(rowkey.getBytes());
72         String[] fields = message.split("\\001");
73         put.addColumn("C1".getBytes(), "msg_time".getBytes(), fields[0].getBytes());
74         put.addColumn("C1".getBytes(), "sender_nickname".getBytes(),
75             fields[1].getBytes());
76         put.addColumn("C1".getBytes(), "sender_account".getBytes(),
77             fields[2].getBytes());
78         put.addColumn("C1".getBytes(), "sender_sex".getBytes(), fields[3].getBytes());
79         put.addColumn("C1".getBytes(), "sender_ip".getBytes(), fields[4].getBytes());
80         put.addColumn("C1".getBytes(), "sender_os".getBytes(), fields[5].getBytes());
81         put.addColumn("C1".getBytes(), "sender_phone_type".getBytes(),
82             fields[6].getBytes());
83         put.addColumn("C1".getBytes(), "sender_network".getBytes(),
84             fields[7].getBytes());
85         put.addColumn("C1".getBytes(), "sender_gps".getBytes(), fields[8].getBytes());
86         put.addColumn("C1".getBytes(), "receiver_nickname".getBytes(),
87             fields[9].getBytes());
88         put.addColumn("C1".getBytes(), "receiver_ip".getBytes(), fields[10].getBytes());
89         put.addColumn("C1".getBytes(), "receiver_account".getBytes(),
90             fields[11].getBytes());
91         put.addColumn("C1".getBytes(), "receiver_os".getBytes(), fields[12].getBytes());
92         put.addColumn("C1".getBytes(), "receiver_phone_type".getBytes(),
93             fields[13].getBytes());
94         put.addColumn("C1".getBytes(), "receiver_network".getBytes(),
95             fields[14].getBytes());
96         put.addColumn("C1".getBytes(), "receiver_gps".getBytes(),
97             fields[15].getBytes());
98         put.addColumn("C1".getBytes(), "receiver_sex".getBytes(),
99             fields[16].getBytes());
100        put.addColumn("C1".getBytes(), "msg_type".getBytes(), fields[17].getBytes());
101        put.addColumn("C1".getBytes(), "distance".getBytes(), fields[18].getBytes());
102        put.addColumn("C1".getBytes(), "message".getBytes(), fields[19].getBytes());
103        table.put(put);
104    }
105
106    //构造一个formatter，把字符串格式转化成日期，转成Date
107    private static SimpleDateFormat formmater = new SimpleDateFormat("yyyy-MM-dd
108        HH:mm:ss");
```

```

99
100     public static String getRowkey(String message) throws ParseException {
101         // TODO 生成rowkey
102         //MD5HASH值_发件人账号_收件人账号_时间戳
103         //计算hash值时，只针对收件人 发件人账号计算，不计算时间戳的hash值
104         String[] fields = message.split("\\001");
105         //获取发件人 收件人账号
106         String sender_account = fields[2];
107         String reciever_account = fields[11];
108         //时间戳
109         String msg_time = fields[0];
110         Date date = formmater.parse(msg_time);
111         long timeStamp = date.getTime();
112
113         //生成md5
114         String md5Hash = MD5Hash.getMD5AsHex((sender_account + "_" +
115         reciever_account).getBytes()).substring(0, 8);
116         //拼接
117         return md5Hash + "_" + sender_account + "_" + reciever_account + "_" +
118         timeStamp;
119     }
120
121     public static void readFromKafka() throws IOException, ParseException {
122         Properties props = new Properties();
123         //1.1设置相关属性
124         //设置kafka集群的地址
125         props.setProperty("bootstrap.servers", "node1:9092,node2:9092,node3:9092");
126         //设置消费者组，组名字自定义，组名字相同的消费者在一个组
127         props.setProperty("group.id", "momo_g1");
128         //开启offset自动提交
129         props.setProperty("enable.auto.commit", "true");
130         //自动提交时间间隔
131         props.setProperty("auto.commit.interval.ms", "1000");
132         //序列化器
133         props.setProperty("key.deserializer",
134         "org.apache.kafka.common.serialization.StringDeserializer");
135         props.setProperty("value.deserializer",
136         "org.apache.kafka.common.serialization.StringDeserializer");
137         //1.2创建一个消费者
138         //实例化一个消费者

```

```

135         KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>
            (props);
136         //1.3订阅主题MOMO_MSG
137         consumer.subscribe(Arrays.asList("MOMO_MSG"));
138         //1.4 循环读取数据
139         while (true) {
140             ConsumerRecords<String, String> records =
            consumer.poll(Duration.ofMillis(100));
141             for (ConsumerRecord<String, String> record : records) {
142                 //          System.out.printf("偏移量 = %d , 键 = %s , 值= %s\n ",
            record.offset(), record.key(), record.value());
143                 String value = record.value();
144                 if (value != null && !"".equals(value) && value.split("\\001").length ==
            20) {
145                     writeToHBase(value);
146                 }
147             }
148         }
149     }
150 }
151

```

• 4.测试

- 1 启动代码，启动flume，检查hbase中是否有数据存在

5.与Phoenix整合完成即席操作

- 启动Phoenix: node1

```

1 cd /export/server/apache-phoenix-5.0.0-HBase-2.0-bin/bin
2 python2 sqlline.py node1:2181
3
4

```

- 构建视图和hbase表进行关联

```

1 格式
2 create view "名称空间"."hbase中对应的表名" (    --表名必须是hbase中的表，不能随便起名
3 key varchar primary key, -- rowkey
4 "列族"."列名1" 类型,
5 "列族"."列名2" 类型,

```



```

6  "列族"."列名3" 类型,
7  ...
8  )[default_columns_family="列族名"]; 如果不在后面设置列族名, 就在前面设定
9
10 create view "MOMO_CHAT"."MOMO_MSG"(
11 id varchar primary key,
12 C1."msg_time" varchar,
13 C1."sender_nickname" varchar,
14 C1."sender_account" varchar,
15 C1."sender_sex" varchar,
16 C1."sender_ip" varchar,
17 C1."sender_os" varchar,
18 C1."sender_phone_type" varchar,
19 C1."sender_network" varchar,
20 C1."sender_gps" varchar,
21 C1."receiver_nickname" varchar,
22 C1."receiver_ip" varchar,
23 C1."receiver_account" varchar,
24 C1."receiver_os" varchar,
25 C1."receiver_phone_type" varchar,
26 C1."receiver_network" varchar,
27 C1."receiver_gps" varchar,
28 C1."receiver_sex" varchar,
29 C1."msg_type" varchar,
30 C1."distance" varchar,
31 C1."message" varchar
32 );
33

```

6.与hive整合

- 启动hive两个服务, metastore和hiveserver2服务
- 创建一个hive表

```

1  创建hive关联到hbase, 这个hive表是一个外部表,
2
3  格式
4
5  create external table 数据库名.表名 (
6  字段1 类型,

```

```
7  字段2 类型,
8  字段3 类型,
9  ....
10 ) stored by "org.apache.hadoop.hive.hbase.HBaseStorageHandler" WITH SERDEPROPERTIES
    ("hbase.columns.mapping"=":key,列族1:列名1,列族2:列名2,...")
    TBLPROPERTIES("hbase.table.name"="hbase表名");
11
12
13 例子
14 create database momo_chat;
15 use momo_chat;
16 create external table momo_chat.momo_msg (
17  id string,
18  msg_time string,
19  sender_nickname string,
20  sender_account string,
21  sender_sex string,
22  sender_ip string,
23  sender_os string,
24  sender_phone_type string,
25  sender_network string,
26  sender_gps string,
27  receiver_nickname string,
28  receiver_ip string,
29  receiver_account string,
30  receiver_os string,
31  receiver_phone_type string,
32  receiver_network string,
33  receiver_gps string,
34  receiver_sex string,
35  msg_type string,
36  distance string,
37  message string
38 ) stored by "org.apache.hadoop.hive.hbase.HBaseStorageHandler" with serdeproperties
    ("hbase.columns.mapping"=":key,C1:msg_time,
39  C1:sender_nickname,
40  C1:sender_account,
41  C1:sender_sex,
42  C1:sender_ip,
43  C1:sender_os,
44  C1:sender_phone_type,
```

```

45 C1:sender_network,
46 C1:sender_gps,
47 C1:receiver_nickname,
48 C1:receiver_ip,
49 C1:receiver_account,
50 C1:receiver_os,
51 C1:receiver_phone_type,
52 C1:receiver_network,
53 C1:receiver_gps,
54 C1:receiver_sex,
55 C1:msg_type,
56 C1:distance,
57 C1:message
58 ") TBLPROPERTIES("hbase.table.name"="MOMO_CHAT:MOMO_MSG");

```

7.整体测试

• 1.清空hbase所有的数据

```

1  disable "MOMO_CHAT:MOMO_MSG"
2  drop "MOMO_CHAT:MOMO_MSG"
3
4  #create_namespace "MOMO_CHAT"
5
6  create "MOMO_CHAT:MOMO_MSG", {NAME=>"C1", COMPRESSION=>"GZ"}, {NUMREGIONS=>6,
  SPLITALGO=>"HexStringSplit"}

```

• 2.删除Phoenix中的视图，删除hive表，重新创建

```

1  Phoenix中:
2  drop view momo_chat.momo_msg;
3  # 重建视图
4  create view MOMO_CHAT.MOMO_MSG (
5  id varchar primary key,
6  C1."msg_time" varchar,
7  C1."sender_nickname" varchar,
8  C1."sender_account" varchar,
9  C1."sender_sex" varchar,
10 C1."sender_ip" varchar,
11 C1."sender_os" varchar,

```

```
12 C1."sender_phone_type" varchar,
13 C1."sender_network" varchar,
14 C1."sender_gps" varchar,
15 C1."receiver_nickname" varchar,
16 C1."receiver_ip" varchar,
17 C1."receiver_account" varchar,
18 C1."receiver_os" varchar,
19 C1."receiver_phone_type" varchar,
20 C1."receiver_network" varchar,
21 C1."receiver_gps" varchar,
22 C1."receiver_sex" varchar,
23 C1."msg_type" varchar,
24 C1."distance" varchar,
25 C1."message" varchar
26 );
27
28 hive中
29 删除表:
30 drop table momo_chat.momo_msg;
31 重建表
32 create database momo_chat;
33 use momo_chat;
34 create external table momo_chat.momo_msg (
35 id string,
36 msg_time string,
37 sender_nickname string,
38 sender_account string,
39 sender_sex string,
40 sender_ip string,
41 sender_os string,
42 sender_phone_type string,
43 sender_network string,
44 sender_gps string,
45 receiver_nickname string,
46 receiver_ip string,
47 receiver_account string,
48 receiver_os string,
49 receiver_phone_type string,
50 receiver_network string,
51 receiver_gps string,
```

```

52 receiver_sex string,
53 msg_type string,
54 distance string,
55 message string
56 ) stored by "org.apache.hadoop.hive.hbase.HBaseStorageHandler" with serdeproperties
57 ("hbase.columns.mapping"=":key,C1:msg_time,
58 C1:sender_nickname,
59 C1:sender_account,
60 C1:sender_sex,
61 C1:sender_ip,
62 C1:sender_os,
63 C1:sender_phone_type,
64 C1:sender_network,
65 C1:receiver_nickname,
66 C1:receiver_ip,
67 C1:receiver_account,
68 C1:receiver_os,
69 C1:receiver_phone_type,
70 C1:receiver_network,
71 C1:receiver_gps,
72 C1:receiver_sex,
73 C1:msg_type,
74 C1:distance,
75 C1:message
76 ") TBLPROPERTIES("hbase.table.name"="MOMO_CHAT:MOMO_MSG");

```

• 3.删除生产数据的jar包产生数据

```

1 cd /export/data/momo_data
2 rm MOMO_DATA.dat

```

• 4.删除断点续传文件

```

1 rm /export/data/flume/taildir_position.json

```

• 5.删除kafka topic , 创建新的

```

1 kafka-topics.sh --delete --topic MOMO_MSG --zookeeper node1:2181,node2:2181,node3:2181
2 kafka-topics.sh --create --topic MOMO_MSG --zookeeper node1:2181,node2:2181,node3:2181 --
  -partitions 3 --replication-factor 2

```

• 测试操作

- 1 1) 先启动zookeeper
- 2 2) 接着启动 hadoop集群
- 3 3) 然后启动 kafka hbase
- 4 4) 最后启动: hive Phoenix
- 5 5) 启动写入hbase的程序
- 6 6) 启动flume程序
- 7 7) 启动一个监听 kafka对应topic的消费者
- 8 8) 启动生产数据jar包
- 9 9) 检查在Phoenix查询是否有数据生成, 以及在hive中查询数据是否存在, 同时观察 消费者是否消费到数据