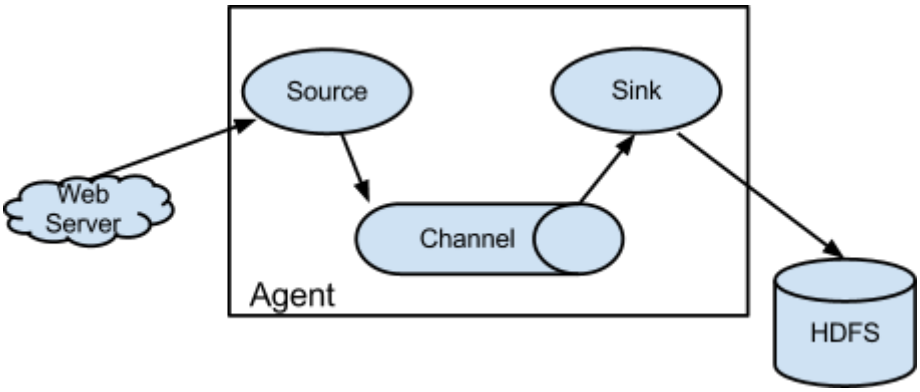
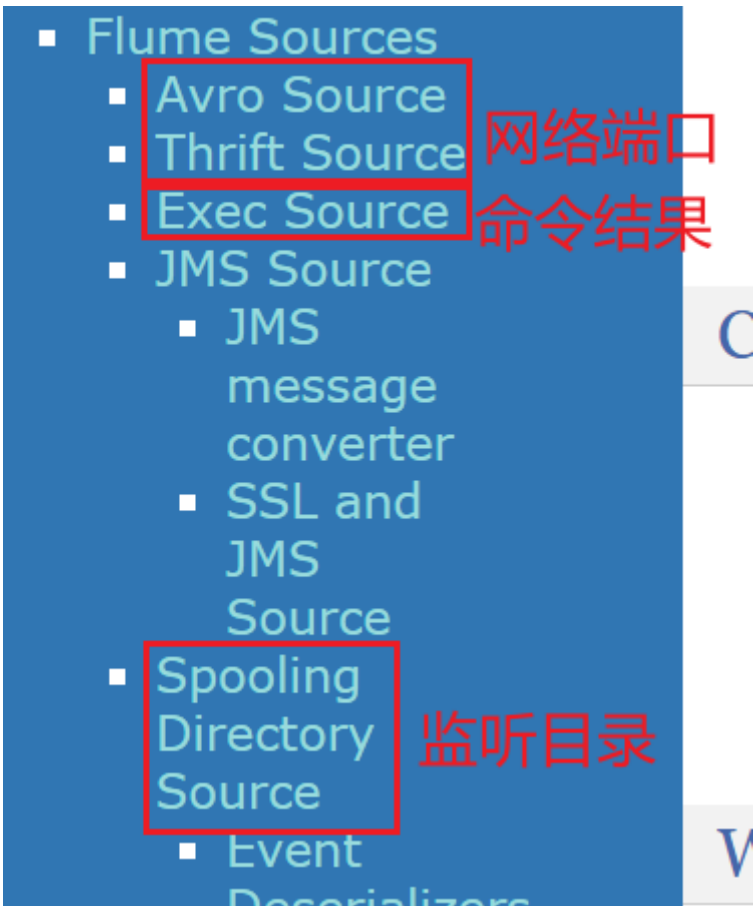
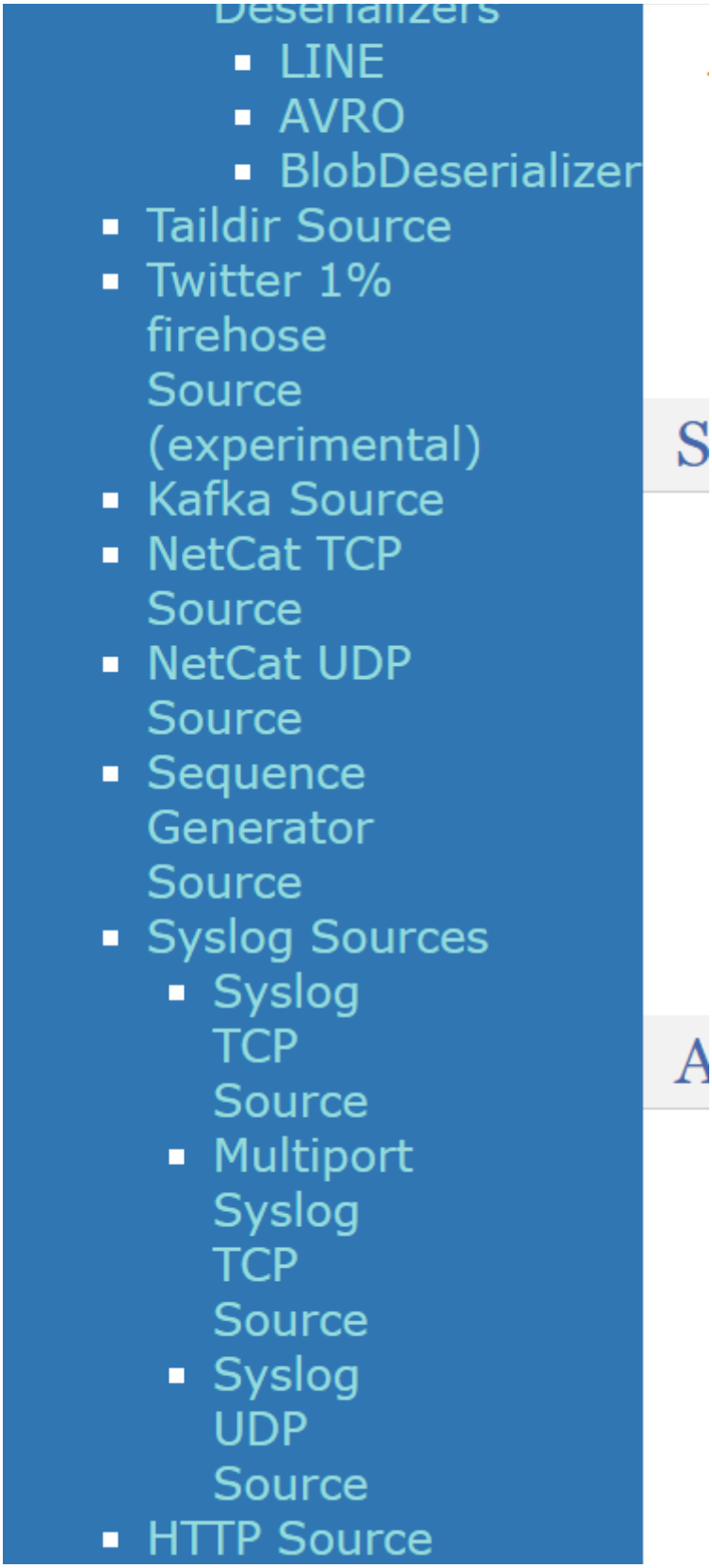
# 1.基本介绍

- 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

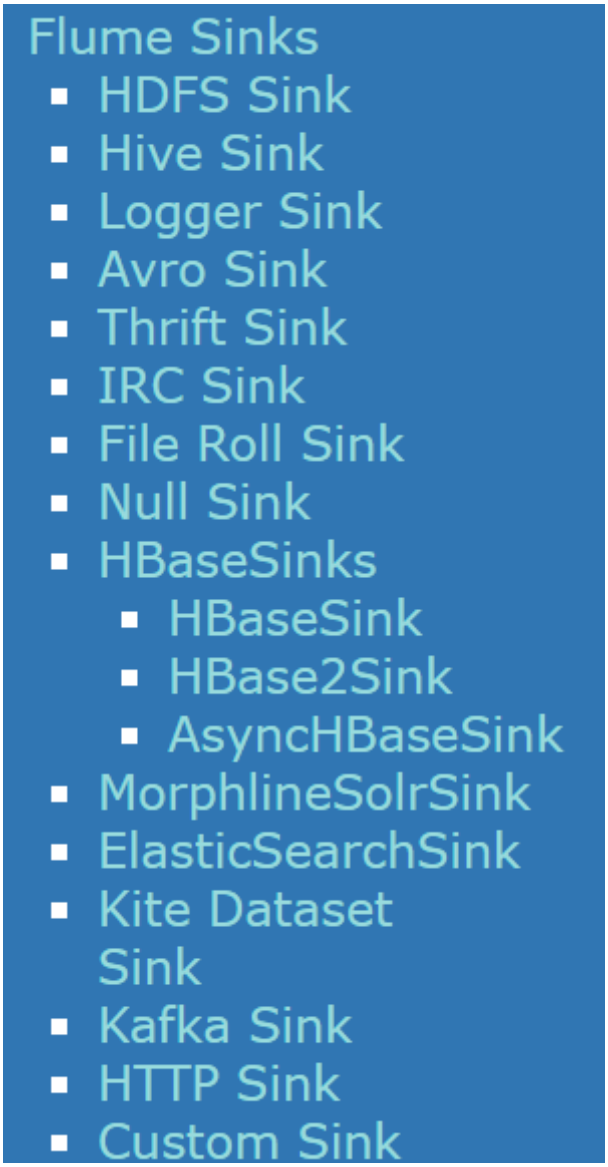



- 整个agent中数据是一条条进行传输的，flume会把数据封装成event格式的对象，event中处理存放数据还会放置一些自定义的东西
- source组件





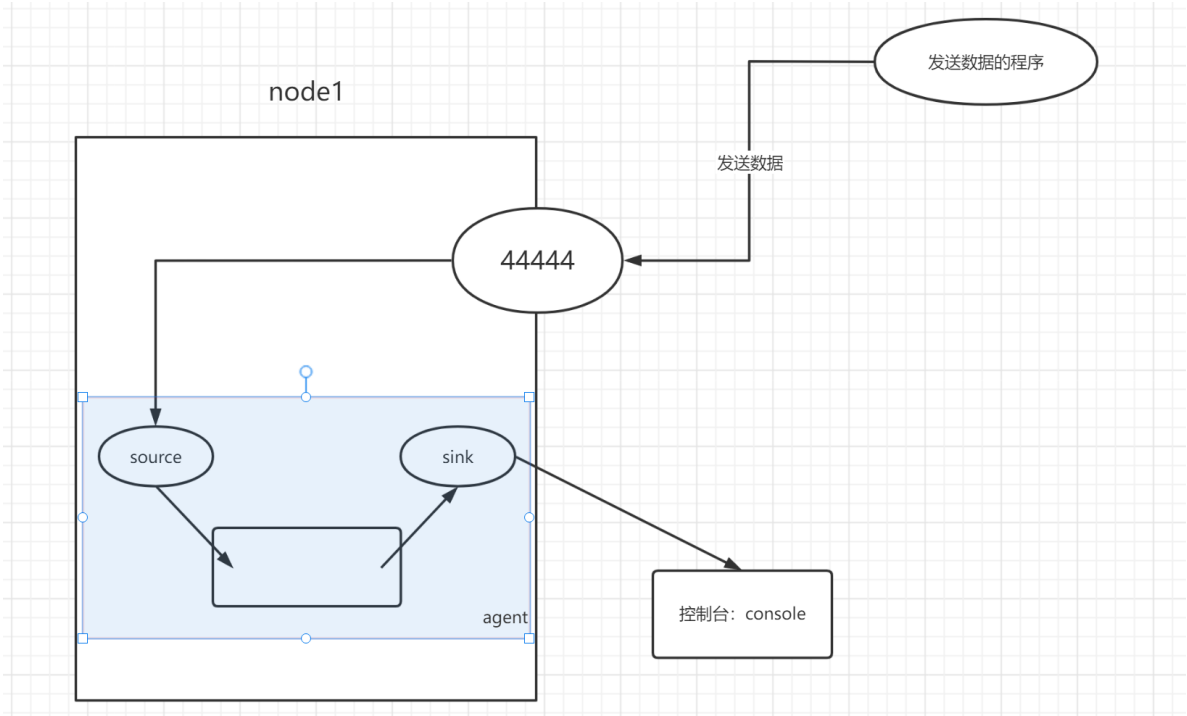

- taildir 监听目录和文件
- kafka 监听kafka主题
- netcat 监听网络端口
- channel组件，一般memchannel缓存在内存中
- sink组件
-




- 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**

```
第一步确定三个组件类型
确定组件类型之前，一定要确定网络结构
source组件
功能：监听某一个端口
组件类型：netcat tcp source
实例
a1.sources = r1
#类型为netcat
a1.sources.r1.type = netcat
#连接的机器
a1.sources.r1.bind = 0.0.0.0
#监听端口
a1.sources.r1.port = 6666
#建立与channel连接
a1.sources.r1.channels = c1
```

- **channel组件**

```
功能 ：缓存
组件类型 ：memroy channel
实例：
a1.channels = c1
a1.channels.c1.type = memory
```

- **sink组件**

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

- **示例**

```
第二步根据需求对组件配置进行调整
source组件

a1.sources = r1
a1.sources.r1.type = netcat
a1.sources.r1.bind = node1
a1.sources.r1.port = 44444
a1.sources.r1.channels = c1

channel组件
a1.channels = c1
a1.channels.c1.type = memory



sink组件
a1.sinks = k1
a1.sinks.k1.type = logger
a1.sinks.k1.channel = c1

第三步把汇总写入文件
/export/server/apache-flume-1.9.0-bin/conf/intro_netcat_source_logger_sink.conf

文件名望文知义，文件中不能有中文注释


完整内容

a1.sources = r1
a1.sources.r1.type = netcat
```

```
a1.sources.r1.bind = node1
a1.sources.r1.port = 44444
a1.sources.r1.channels = c1

a1.channels = c1
a1.channels.c1.type = memory



a1.sinks = k1
a1.sinks.k1.type = logger
a1.sinks.k1.channel = c1
```

- **启动监听**

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

参数说明:
     -c conf   指定flume自身的配置文件所在目录
     -f conf/netcat-logger.con  指定我们所描述的采集方案
     -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

```
启动telnet

[root@node1 ~]# telnet
-bash: telnet: command not found
[root@node1 ~]# yum install telnet
需要手动安装telnet

输入：telnet node1 44444
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 '^]'.
```

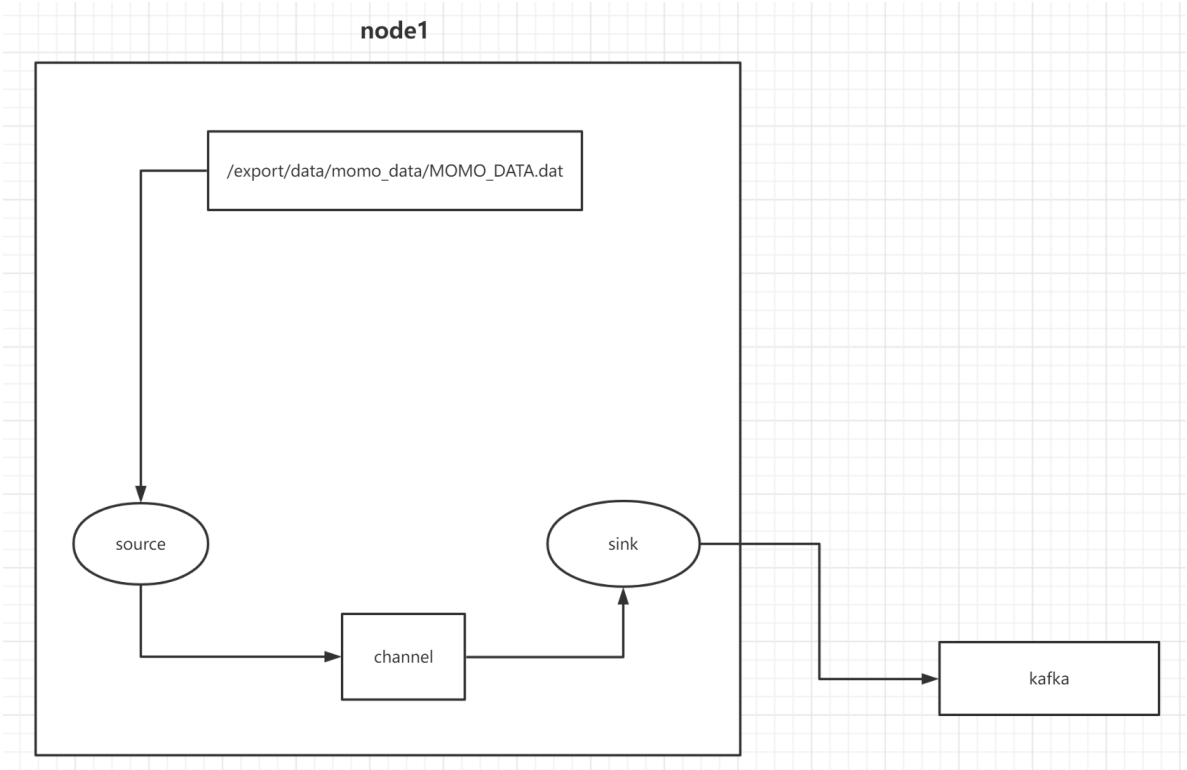

```
2022-03-14 14:55:13,832 INFO sink.LoggerSink: Event: { headers:
{} body: 61 62 63 0D                                        abc. }
2022-03-14 14:55:14,708 INFO sink.LoggerSink: Event: { headers:
{} body: 66 6C 75 6D 65 0D                                  flume.
 }
2022-03-14 14:55:16,921 INFO sink.LoggerSink: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

```
确定三个组件

source组件
exec source 执行linux命令，命令输出结果 作为数据源
taildir source 可以监听文件和目录
使用taildir source


a1.sources = r1
a1.channels = c1
a1.sources.r1.type = TAILDIR
a1.sources.r1.channels = c1
a1.sources.r1.positionFile = /var/log/flume/taildir_position.json #断点续传文件
a1.sources.r1.filegroups = f1 f2
a1.sources.r1.filegroups.f1 = /var/log/test1/example.log
a1.sources.r1.headers.f1.headerKey1 = value1
a1.sources.r1.filegroups.f2 = /var/log/test2/.*log.*
a1.sources.r1.headers.f2.headerKey1 = value2
a1.sources.r1.headers.f2.headerKey2 = value2-2
a1.sources.r1.fileHeader = true
a1.sources.ri.maxBatchCount = 1000

sink组件
```

```
a1.sinks.k1.channel = c1
a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k1.kafka.topic = mytopic
a1.sinks.k1.kafka.bootstrap.servers = localhost:9092
a1.sinks.k1.kafka.flumeBatchSize = 20
a1.sinks.k1.kafka.producer.acks = 1
a1.sinks.k1.kafka.producer.linger.ms = 1
a1.sinks.k1.kafka.producer.compression.type = snappy

channel组件
memory channel
a1.channels = c1
a1.channels.c1.type = memory

组合



a1.sources = r1
a1.channels = c1
a1.sources.r1.type = TAILDIR
a1.sources.r1.channels = c1
a1.sources.r1.positionFile = /export/data/flume/taildir_position.json #断点续传文件
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /export/data/momo_data/MOMO_DATA.dat
#a1.sources.r1.maxBatchCount = 1000

a1.channels.c1.type = memory

a1.sinks = k1
a1.sinks.k1.channel = c1
a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k1.kafka.topic = MOMO_MSG
a1.sinks.k1.kafka.bootstrap.servers =
node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092
a1.sinks.k1.kafka.flumeBatchSize = 20
a1.sinks.k1.kafka.producer.acks = 1
a1.sinks.k1.kafka.producer.linger.ms = 1


```

```
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
```

```
2 列族怎么设计 名称叫什么
只有一个列族，C1
3 数据要不要压缩，如果需要 怎么压缩
数据需要压缩，用什么算法，GZ，因为读取的情况很少
4 是否需要对数据进行预分区，如果需要，怎么做
需要，搞6预分区，使用hash预分区
5 数据是否需要设置版本和过期时间
默认一个版本，发出去的消息不能更改，默认长期存储



create_namespace "MOMO_CHAT"
create "MOMO_CHAT:MOMO_MSG", {NAME=>"C1", COMPRESSION=>"GZ"}, {NUMREGIONS=>6, SPLITALGO=>"HexStringSplit"}
```

# 2.rowkey设计

```
官方规定
1 不能使用递增或者时序数据，手机号 时间戳
2 rowkey设计时不要太长，推荐100Byte以内
3 数值类型的rowkey比字符串更节省空间
4 必须保证rowkey唯一

业务
保证相关性的数据放在一起，满足查询的需要

需求
根据发件人 收件人 账号 聊天时间进行查询


rowkey设计
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

```
    <dependencies>
        <dependency>
            <groupId>org.apache.hbase</groupId>
```

```
            <artifactId>hbase-client</artifactId>
            <version>2.1.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>2.4.1</version>
        </dependency>
    </dependencies>
```

- 创建一个类MOMOKafkaToHBase

```
package sz.base.momo;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.MD5Hash;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.Duration;
import java.util.Arrays;
import java.util.Date;
import java.util.Properties;

public class MOMOKafkaToHBase {
    //从kafka的MOMO_MSG主题中获取数据，写入到MOMO_CHAT:MOMO_MSG hbase表中
    //1.从kafka中读取数据
    //1.1设置相关属性
    //1.2创建一个消费者
    //1.3订阅主题 MOMO_MSG
    //1.4循环读取消息
    //2.把数据写入到hbase表中
```

```
    //2.1连接hbase
    //2.2获取表的管理对象
    //2.3构造put对象，把数据写入
    //2.4构造rowkey
    //2.5释放资源
    public static void main(String[] args) throws IOException, ParseException {
        readFromKafka();

    }

    private static Connection hbConn;
    private static Table table;

    static {
        //静态代码块，随着类的加载而加载，一般只执行一次
        //2.1.建立连接，连接到hbase
        Configuration conf = HBaseConfiguration.create();
        conf.set("hbase.zookeeper.quorum", "node1,node2,node3");
        //创建连接

        try {
            hbConn = ConnectionFactory.createConnection(conf);
        } catch (IOException e) {
            e.printStackTrace();
        }
        //从连接对象中获取管理对象
        //Admin对象，对表的管理
        //Table对象，对表的数据进行管理
        TableName table1 = TableName.valueOf("MOMO_CHAT:MOMO_MSG");
        try {
            table = hbConn.getTable(table1);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    public static void writeToHBase(String message) throws IOException, ParseException {
        //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_nickyname".getBytes(),
    fields[1].getBytes());
75          put.addColumn("C1".getBytes(), "sender_account".getBytes(),
    fields[2].getBytes());
76          put.addColumn("C1".getBytes(), "sender_sex".getBytes(), fields[3].getBytes());
77          put.addColumn("C1".getBytes(), "sender_ip".getBytes(), fields[4].getBytes());
78          put.addColumn("C1".getBytes(), "sender_os".getBytes(), fields[5].getBytes());
79          put.addColumn("C1".getBytes(), "sender_phone_type".getBytes(),
    fields[6].getBytes());
80          put.addColumn("C1".getBytes(), "sender_network".getBytes(),
    fields[7].getBytes());
81          put.addColumn("C1".getBytes(), "sender_gps".getBytes(), fields[8].getBytes());
82          put.addColumn("C1".getBytes(), "receiver_nickyname".getBytes(),
    fields[9].getBytes());
83          put.addColumn("C1".getBytes(), "receiver_ip".getBytes(), fields[10].getBytes());
84          put.addColumn("C1".getBytes(), "receiver_account".getBytes(),
    fields[11].getBytes());
85          put.addColumn("C1".getBytes(), "receiver_os".getBytes(), fields[12].getBytes());
86          put.addColumn("C1".getBytes(), "receiver_phone_type".getBytes(),
    fields[13].getBytes());
87          put.addColumn("C1".getBytes(), "receiver_network".getBytes(),
    fields[14].getBytes());
88          put.addColumn("C1".getBytes(), "receiver_gps".getBytes(),
    fields[15].getBytes());
89          put.addColumn("C1".getBytes(), "receiver_sex".getBytes(),
    fields[16].getBytes());
90          put.addColumn("C1".getBytes(), "msg_type".getBytes(), fields[17].getBytes());
91          put.addColumn("C1".getBytes(), "distance".getBytes(), fields[18].getBytes());
92          put.addColumn("C1".getBytes(), "message".getBytes(), fields[19].getBytes());
93          table.put(put);
94      }
95
96
97      //构造一个formatter，把字符串格式转化成日期，转成Date
98      private static SimpleDateFormat formmater = new SimpleDateFormat("yyyy-MM-dd
    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 + "_" +
    reciever_account).getBytes()).substring(0, 8);
115         //拼接
116         return md5Hash + "_" + sender_account + "_" + reciever_account + "_" +
    timeStamp;
117     }
118
119     public static void readFromKafka() throws IOException, ParseException {
120         Properties props = new Properties();
121         //1.1设置相关属性
122         //设置kafka集群的地址
123         props.setProperty("bootstrap.servers", "node1:9092,node2:9092,node3:9092");
124         //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
125         props.setProperty("group.id", "momo_g1");
126         //开启offset自动提交
127         props.setProperty("enable.auto.commit", "true");
128         //自动提交时间间隔
129         props.setProperty("auto.commit.interval.ms", "1000");
130         //序列化器
131         props.setProperty("key.deserializer",
    "org.apache.kafka.common.serialization.StringDeserializer");
132         props.setProperty("value.deserializer",
    "org.apache.kafka.common.serialization.StringDeserializer");
133         //1.2创建一个消费者
134         //实例化一个消费者
```

```
        KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);
        //1.3订阅主题MOMO_MSG
        consumer.subscribe(Arrays.asList("MOMO_MSG"));
        //1.4 循环读取数据
        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
            for (ConsumerRecord<String, String> record : records) {
//                System.out.printf("偏移量 = %d , 键 = %s ,  值= %s%n ", record.offset(), record.key(), record.value());
                String value = record.value();
                if (value != null && !"".equals(value) && value.split("\001").length == 20) {
                    writeToHBase(value);
                }
            }
        }
    }
}

```

- **4.测试**

```
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

```
cd /export/server/apache-phoenix-5.0.0-HBase-2.0-bin/bin
python2 sqlline.py node1:2181


```

- 构建视图和hbase表进行关联

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

```
"列族"."列名3" 类型,
...
)[default_columns_family="列族名"]; 如果不在后面设置列族名，就在前面设定

create view "MOMO_CHAT"."MOMO_MSG"(
id varchar primary key,
C1."msg_time" varchar,
C1."sender_nickyname" varchar,
C1."sender_account" varchar,
C1."sender_sex" varchar,
C1."sender_ip" varchar,
C1."sender_os" varchar,
C1."sender_phone_type" varchar,
C1."sender_network" varchar,
C1."sender_gps" varchar,
C1."receiver_nickyname" varchar,
C1."receiver_ip" varchar,
C1."receiver_account" varchar,
C1."receiver_os" varchar,
C1."receiver_phone_type" varchar,
C1."receiver_network" varchar,
C1."receiver_gps" varchar,
C1."receiver_sex" varchar,
C1."msg_type" varchar,
C1."distance" varchar,
C1."message" varchar
);

```

# 6.与hive整合

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

```
创建hive关联到hbase，这个hive表是一个外部表，

格式

create external table 数据库名.表名 (
字段1 类型,
```

```
字段2 类型,
字段3 类型,
....
) 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");


例子
create database momo_chat;
use momo_chat;
create external table momo_chat.momo_msg (
id string,
msg_time string,
sender_nickyname string,
sender_account string,
sender_sex string,
sender_ip string,
sender_os string,
sender_phone_type string,
sender_network string,
sender_gps string,
receiver_nickyname string,
receiver_ip string,
receiver_account string,
receiver_os string,
receiver_phone_type string,
receiver_network string,
receiver_gps string,
receiver_sex string,
msg_type string,
distance string,
message string
) stored by "org.apache.hadoop.hive.hbase.HBaseStorageHandler" with serdeproperties ("hbase.columns.mapping"=":key,C1:msg_time,
C1:sender_nickyname,
C1:sender_account,
C1:sender_sex,
C1:sender_ip,
C1:sender_os,
C1:sender_phone_type,
```

```
C1:sender_network,
C1:sender_gps,
C1:receiver_nickyname,
C1:receiver_ip,
C1:receiver_account,
C1:receiver_os,
C1:receiver_phone_type,
C1:receiver_network,
C1:receiver_gps,
C1:receiver_sex,
C1:msg_type,
C1:distance,
C1:message
") TBLPROPERTIES("hbase.table.name"="MOMO_CHAT:MOMO_MSG");
```

# 7.整体测试

- **1.清空hbase所有的数据**

```
disable "MOMO_CHAT:MOMO_MSG"
drop "MOMO_CHAT:MOMO_MSG"

#create_namespace "MOMO_CHAT"

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**

```
Phoenix中:
drop view momo_chat.momo_msg;
# 重建视图
create view MOMO_CHAT.MOMO_MSG (
id varchar primary key,
C1."msg_time" varchar,
C1."sender_nickyname" varchar,
C1."sender_account" varchar,
C1."sender_sex" varchar,
C1."sender_ip" varchar,
C1."sender_os" varchar,
```

```
C1."sender_phone_type" varchar,
C1."sender_network" varchar,
C1."sender_gps" varchar,
C1."receiver_nickyname" varchar,
C1."receiver_ip" varchar,
C1."receiver_account" varchar,
C1."receiver_os" varchar,
C1."receiver_phone_type" varchar,
C1."receiver_network" varchar,
C1."receiver_gps" varchar,
C1."receiver_sex" varchar,
C1."msg_type" varchar,
C1."distance" varchar,
C1."message" varchar
);

hive中
删除表:
drop table momo_chat.momo_msg;
重建表
create database momo_chat;
use momo_chat;
create external table momo_chat.momo_msg (
id string,
msg_time string,
sender_nickyname string,
sender_account string,
sender_sex string,
sender_ip string,
sender_os string,
sender_phone_type string,
sender_network string,
sender_gps string,
receiver_nickyname string,
receiver_ip string,
receiver_account string,
receiver_os string,
receiver_phone_type string,
receiver_network string,
receiver_gps string,
```

```
receiver_sex string,
msg_type string,
distance string,
message string
) stored by "org.apache.hadoop.hive.hbase.HBaseStorageHandler" with serdeproperties ("hbase.columns.mapping"=":key,C1:msg_time,
C1:sender_nickyname,
C1:sender_account,
C1:sender_sex,
C1:sender_ip,
C1:sender_os,
C1:sender_phone_type,
C1:sender_network,
C1:sender_gps,
C1:receiver_nickyname,
C1:receiver_ip,
C1:receiver_account,
C1:receiver_os,
C1:receiver_phone_type,
C1:receiver_network,
C1:receiver_gps,
C1:receiver_sex,
C1:msg_type,
C1:distance,
C1:message
") TBLPROPERTIES("hbase.table.name"="MOMO_CHAT:MOMO_MSG");
```

- **3.删除生产数据的jar包产生数据**

```
cd /export/data/momo_data
rm MOMO_DATA.dat
```

- **4.删除断点续传文件**

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

- **5.删除kafka topic， 创建新的**

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

- **测试操作**

```
1）  先启动zookeeper
2）接着启动 hadoop集群
3）然后启动 kafka  hbase
4）最后启动：  hive  Phoenix
5）启动写入hbase的程序
6）启动flume程序
7）启动一个监听 kafka对应topic的消费者
8）启动生产数据jar包
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
```