

实验二

使用Flume、Thrift、Kafka、HBase进行数据收集传输与存储

班级：2018211310

学号：2018211527

姓名：陈泊任

时间：2021年4月6日

实验二

使用Flume、Thrift、Kafka、HBase进行数据收集传输与存储

一、实验目的:

二、实验平台:

三、实验内容

1、下载相关软件

(1) Flume

(2) Thrift

(3) Kafka

(4) HBase

2、启动HBase, 并创建HBase表 (利用shell)

3、启动三个Kafka服务器

3.1 启动Kafka自带的Zookeeper

3.2 修改conf的server文件

3.3 启动三个Kafka服务器

3.4 创建一个名为"Kafka_Orders"的Topic

4、编写Kafka消费者

4.1 连接Kafka服务器

4.2 连接HBase服务器

4.3 不断读取Kafka消息队列中的事件

4.4 对于每个事件, 将其转化为HBase Put类的形式, 并提交到HBase服务器中

5、启动Flume

6、启动具有数据生成功能的服务器

7、查看HBase表

四、选做内容

1、使用Java API 替代 Java shell 操作

(1) 创建一个"HBase_Orders"表

(2) 定时查看"HBase_Orders"表内的数据量, 完成最简单的统计

2、使用序列化技术

一、实验目的：

- 1、掌握 Flume、Thrift、Kafka、HBase 的安装；
- 2、掌握 Flume、Thrift、Kafka、HBase 的使用。

二、实验平台：

操作系统：Ubuntu 18.04 LTS

三、实验内容

1、下载相关软件

(1) Flume

版本：apache-flume-1.9.0

① 下载

地址：<https://flume.apache.org/download.html>

② 解压

在压缩包目录下解压

```
tar -zxvf apache-flume-1.9.0-bin.tar.gz
```

将解压文件移到flume-1.9.0文件里

```
mv apache-flume-1.9.0-bin flume-1.9.0
```

③ 配置

进入配置文件目录下

```
cd flume-1.9.0/conf
```

将flume-env.sh.template文件复制一份，并重命名为flume-env.sh

```
cp flume-env.sh.template flume-env.sh
```

在flume-env.sh文件中写入以下内容：

```
JAVA_HOME = /usr
```

④ 验证

在flume-1.9.0目录下执行以下命令

```
./bin/flume-ng version
```

若看到flume的版本号，则证明flume已成功安装

```
r52125@fly:~/flume/flume-1.9.0/bin$ ./flume-ng version
Flume 1.9.0
```

成功!

(2) Thrift

版本: thrift-0.14.0

① 下载

地址: <https://mirrors.bfsu.edu.cn/apache/thrift/0.14.1/thrift-0.14.1.tar.gz>

② 解压

在压缩包的目录下解压

```
tar -zxvf thrift-0.14.1.tar.gz
```

将解压文件移动到thrift-0.14.1目录下

```
mv thrift-0.14.1 thrift-0.14.1
```

③ 安装依赖

```
sudo apt-get install automake bison flex g++ git libboost-all-dev libevent-dev
libssl-dev libtool make pkg-config
```

④ 安装Thrift

进入thrift-0.14.1目录下，依次输入以下命令

```
./configure
```

```
make
```

```
sudo make install
```

⑤ 验证

```
thrift -version
```

```
r52125@fly:~/thrift/thrift-0.14.1$ thrift -version
Thrift version 0.14.1
```

成功!

(3) Kafka

版本: kafka-2.13-2.6.1

① 下载

从官网上将安装包下载到虚拟机

下载地址: <http://kafka.apache.org/downloads>

② 解压

在解压包目录下

```
tar -zxvf kafka_2.13-2.6.1.tgz
```

将解压文件移到kafka_2.13-2.6.1目录下

```
mv kafka_2.13-2.6.1 kafka-2.13-2.6.1
```

③ 启动zookeeper

修改配置

在kafka-2.13.2.6.1/config目录下修改zookeeper.properties

```
clientPort=2281
```

```
# the port at which the clients will connect
clientPort=2281
```

启动

在kafka-2.13-2.6.1目录下输入以下命令 (参数-daemon表示后台守护运行)

```
bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
```

验证:

方法1: jps命令

```
jps
```

```
帮助
r52125@fly:~/kafka/kafka-2.13-2.6.1$ jps
46832 Jps
41553 ResourceManager
41732 NodeManager
40948 NameNode
41380 SecondaryNameNode
45449 QuorumPeerMain
41116 DataNode
```

有QuorumPeerMain则启动成功

方法2: ps命令加上正则表达式

```
ps aux | grep 'zookeeper'
```

```
r52125 46849 0.0 0.0 21540 1064 pts/0 S+ 10:16 0:00 grep --color=
auto zookeeper
```

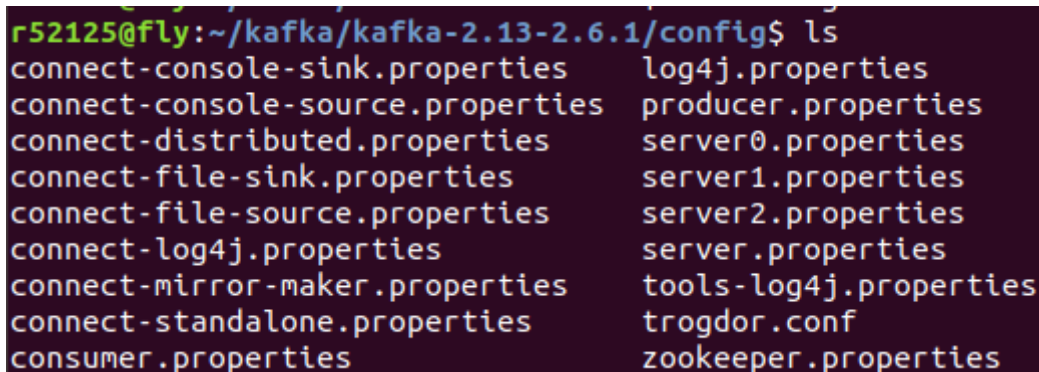
有返回则启动成功

④ 启动三个Kafka服务器

因为config里面只有一个server.properties，但我们需要启动三个kafka服务器，所以需要如下配置：

将server.properties文件复制三份，分别命名为server0.properties、server1.properties和server2.properties

```
cp config/server.properties config/server0.properties
cp config/server.properties config/server1.properties
cp config/server.properties config/server2.properties
```



```
r52125@fly:~/kafka/kafka-2.13-2.6.1/config$ ls
connect-console-sink.properties    log4j.properties
connect-console-source.properties  producer.properties
connect-distributed.properties      server0.properties
connect-file-sink.properties        server1.properties
connect-file-source.properties      server2.properties
connect-log4j.properties            server.properties
connect-mirror-maker.properties     tools-log4j.properties
connect-standalone.properties       trogdor.conf
consumer.properties                 zookeeper.properties
```

并对他们做出如下修改

server0.properties:

```
broker.id=0
listeners=PLAINTEXT://:9092
advertised.listeners=PLAINTEXT://fly:9092
log.dir=/tmp/kafka-logs-0
```

```
# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0
```

```
# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#   listeners = listener_name://host_name:port
#   EXAMPLE:
#   listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://:9092
```

```
# Hostname and port the broker will advertise to producers and consumers. If no
# t set,
# it uses the value for "listeners" if configured. Otherwise, it will use the
# value
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=PLAINTEXT://fly:9092
```

```
# A comma separated list of directories under which to store log files
log.dirs=/tmp/kafka-logs-0
```

server1.properties(截图跟上面的差不多，就不截图了):

```
broker.id=1
listeners=PLAINTEXT://:9093
advertised.listeners=PLAINTEXT://fly:9093
log.dir=/tmp/kafka-logs-1
```

server2.properties(同上):

```
broker.id=2  
listeners=PLAINTEXT://:9094  
advertised.listeners=PLAINTEXT://fly:9094  
log.dir=/tmp/kafka-logs-2
```

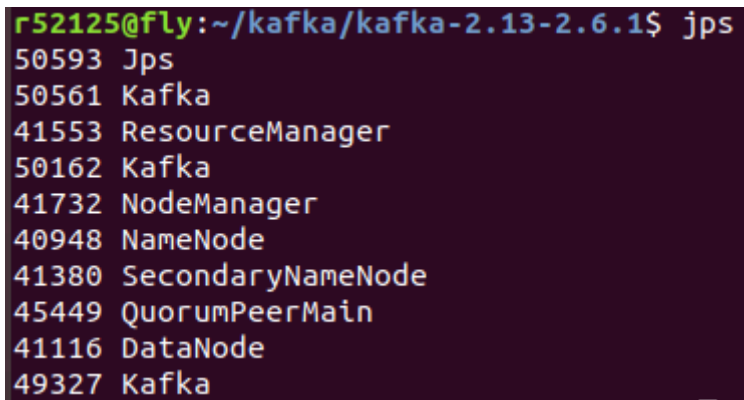
启动三个broker节点

```
bin/kafka-server-start.sh config/server0.properties  
bin/kafka-server-start.sh config/server1.properties  
bin/kafka-server-start.sh config/server2.properties
```

验证:

使用jps命令查看运行情况

```
jps
```



```
r52125@fly:~/kafka/kafka-2.13-2.6.1$ jps  
50593 Jps  
50561 Kafka  
41553 ResourceManager  
50162 Kafka  
41732 NodeManager  
40948 NameNode  
41380 SecondaryNameNode  
45449 QuorumPeerMain  
41116 DataNode  
49327 Kafka
```

可以看到三个kafka服务器已启动

(4) HBase

版本: apache-hbase-2.3.4

① 下载

地址: <https://hadoop.apache.org/releases.html>

注意, 需下载与Hadoop兼容的版本。Hadoop与HBase版本兼容性在下列地址中查询

<http://hbase.apache.org/book.html#hadoop>

官方推荐使用HBase2.3.4与Hadoop2.10.1集成

② 解压

执行tar命令, 解压下载的hbase-2.3.4-bin.tar.gz压缩包文件

```
tar -zxvf hbase-2.3.4-bin.tar.gz
```

③ 修改配置文件

1、在hbase-2.3.4/conf目录下, 修改hbase-env.sh文件的以下内容

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HBASE_CLASSPATH=~/.HBase/hbase-2.3.4/conf
export HBASE_MANAGES_ZK=true
```

```
# The java implementation to use. Java 1.8+ required.
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

# Extra Java CLASSPATH elements. Optional.
export HBASE_CLASSPATH=~/.HBase/hbase-2.3.4/conf
```

2、在hbase-2.3.4/conf目录下，修改hbase-site.xml文件

将属性hbase.cluter.distributed设置为true

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>hdfs://fly:9000/hbase</value>
  </property>
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
</configuration>
```

3、修改regionservers文件

```
fly
```

④ 启动

启动HBase之前，需先启动Hadoop

进入HBase安装的目录，执行以下命令启动HBase

```
./bin/start-hbase.sh
```

⑤ 使用jps命令查看HBase启动情况

以伪分布式方式启动HBase，会在jps命令中看到三个进程HMaster, HRegionServer, HQuorumPeer

```
jps
```



```
r52125@fly:~/HBase/hbase-2.3.4$ jps
41732 NodeManager
41380 SecondaryNameNode
65382 HMaster
65702 Jps
61418 Kafka
41553 ResourceManager
61010 Kafka
65524 HRegionServer
40948 NameNode
61813 Kafka
51222 QuorumPeerMain
65271 HQuorumPeer
41116 DataNode
```

安装成功

2、启动HBase，并创建HBase表（利用shell）

① 进入交互式shell命令行

```
./bin/hbase shell
```

```
^Cr52125@fly:~/HBase/hbase-2.3.4$ bin/hbase shell
2021-04-01 20:19:59,538 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.3.4, raf5e4fc3cd259257229df3422f2857ed35da4cc, Thu Jan 14 21:32:25 UTC 2021
Took 0.0014 seconds
hbase(main):001:0>
```

② 创建一个新的命名空间，'bigdata'

```
create_namespace 'bigdata'
```

```
hbase(main):002:0> create_namespace 'bigdata'
Took 0.1742 seconds
hbase(main):003:0> list_namespace
NAMESPACE
bigdata
default
hbase
3 row(s)
Took 0.0374 seconds
```

③ 创建HBase表

名: "HBase_Orders"

两个列族, "Order Detail" 与 "Transaction"

命令如下

```
create 'bigdata:HBase_Orders','Order Detail','Transaction'
```

```
hbase(main):004:0> create 'bigdata:HBase_Orders', 'Order Detail', 'Transaction'

Created table bigdata:HBase_Orders
Took 0.7038 seconds
=> Hbase::Table - bigdata:HBase_Orders
hbase(main):005:0> list
TABLE
bigdata:HBase_Orders
1 row(s)
Took 0.0233 seconds
=> ["bigdata:HBase_Orders"]
```

3、启动三个Kafka服务器

3.1、3.2、3.3在前面安装Kafka的步骤已经说过一次了，这里再简单说一次

3.1 启动Kafka自带的Zookeeper

在kafka-2.13.2.6.1/config目录下修改zookeeper.properties

```
clientPort=2281
```

```
# the port at which the clients will connect
clientPort=2281
```

启动

参数-daemon表示后台守护运行

不加-daemon的话就需要新开一个终端，效果一样

```
bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
```

```
r52125@fly:~/kafka/kafka-2.13-2.6.1$ bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
r52125@fly:~/kafka/kafka-2.13-2.6.1$
```

```
r52125@fly:~/kafka/kafka-2.13-2.6.1$ jps
6211 HMaster
6547 HRegionServer
6083 HQuorumPeer
2547 NameNode
3508 NodeManager
8261 Jps
8073 QuorumPeerMain
2987 SecondaryNameNode
2748 DataNode
3149 ResourceManager
```

3.2 修改conf的server文件

将server.properties文件复制三份，分别命名为server0.properties、server1.properties和server2.properties

```
cp config/server.properties config/server0.properties
cp config/server.properties config/server1.properties
cp config/server.properties config/server2.properties
```

```
r52125@fly:~/kafka/kafka-2.13-2.6.1/config$ ls
connect-console-sink.properties    log4j.properties
connect-console-source.properties  producer.properties
connect-distributed.properties     server0.properties
connect-file-sink.properties       server1.properties
connect-file-source.properties     server2.properties
connect-log4j.properties          server.properties
connect-mirror-maker.properties   tools-log4j.properties
connect-standalone.properties     trogdor.conf
consumer.properties               zookeeper.properties
```

并对他们做出如下修改

server0.properties:

```
broker.id=0
listeners=PLAINTEXT://:9092
advertised.listeners=PLAINTEXT://fly:9092
log.dir=/tmp/kafka-logs-0
```

```
# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0
```

```
# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#   listeners = listener_name://host_name:port
#   EXAMPLE:
#   listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://:9092

# Hostname and port the broker will advertise to producers and consumers. If no
# t set,
# it uses the value for "listeners" if configured. Otherwise, it will use the
# value
# returned from java.net.InetAddress.getCanonicalHostName().
advertised.listeners=PLAINTEXT://fly:9092
```

```
# A comma separated list of directories under which to store log files
log.dirs=/tmp/kafka-logs-0
```

server1.properties(截图跟上面的差不多，就不截图了):

```
broker.id=1
listeners=PLAINTEXT://:9093
advertised.listeners=PLAINTEXT://fly:9093
log.dir=/tmp/kafka-logs-1
```

server2.properties(同上):

```
broker.id=2
listeners=PLAINTEXT://:9094
advertised.listeners=PLAINTEXT://fly:9094
log.dir=/tmp/kafka-logs-2
```

3.3 启动三个Kafka服务器

启动三个broker节点

```
bin/kafka-server-start.sh -daemon config/server0.properties  
bin/kafka-server-start.sh -daemon config/server1.properties  
bin/kafka-server-start.sh -daemon config/server2.properties
```

```
r52125@fly:~/kafka/kafka-2.13-2.6.1$ bin/kafka-server-start.sh -daemon config/s  
erver0.properties  
r52125@fly:~/kafka/kafka-2.13-2.6.1$ bin/kafka-server-start.sh -daemon config/s  
erver1.properties  
r52125@fly:~/kafka/kafka-2.13-2.6.1$ bin/kafka-server-start.sh -daemon config/s  
erver2.properties
```

```
r52125@fly:~/kafka/kafka-2.13-2.6.1$ jps  
6211 HMaster  
6083 HQuorumPeer  
9833 Kafka  
8073 QuorumPeerMain  
2987 SecondaryNameNode  
3149 ResourceManager  
9042 Kafka  
6547 HRegionServer  
2547 NameNode  
3508 NodeManager  
2748 DataNode  
9916 Jps  
9438 Kafka
```

启动成功!

3.4 创建一个名为"Kafka_Orders"的Topic

一个分区和三个副本

```
bin/kafka-topics.sh --create --zookeeper fly:2281 --replication-factor 3 --  
partitions 1 --topic kafka_orders
```

```
r52125@fly:~/kafka/kafka-2.13-2.6.1$ bin/kafka-topics.sh --create --zookeeper f  
ly:2281 --replication-factor 3 --partitions 1 --topic Kafka_Orders  
WARNING: Due to limitations in metric names, topics with a period ('.') or unde  
rscore ('_') could collide. To avoid issues it is best to use either, but not b  
oth.  
Created topic Kafka_Orders.
```

创建成功

验证:

```
bin/kafka-topics.sh --list --zookeeper fly:2281
```

```
r52125@fly:~/kafka/kafka-2.13-2.6.1$ bin/kafka-topics.sh --list --zookeeper fly  
:2281  
Kafka_Orders
```

4、编写Kafka消费者

4.1 连接Kafka服务器

```
public kafka_consumer(){
    Properties props=new Properties();
    // 监听端口
    props.put("bootstrap.servers","fly:9094");
    // 指定消费者所属群组
    props.put("group.id","test-consumer-group");
    props.put("enable.auto.commit","true");
    props.put("auto.commit.interval.ms","1000");
    props.put("session.timeout.ms","30000");
    props.put("auto.offset.reset","earliest");

    props.put("key.deserializer","org.apache.kafka.common.serialization.StringDeserializer");

    props.put("value.deserializer","org.apache.kafka.common.serialization.StringDeserializer");
    consumer=new KafkaConsumer<String,String>(props);
}
```

4.2 连接HBase服务器

```
Configuration conf = HBaseConfiguration.create();
conf.set("hbase.rootdir","hdfs://fly:9000/hbase");
conf.set("hbase.zookeeper.property.dataDir","/home/r52125/HBase/hbase-2.3.4/zookeeper");
conf.set("hbase.zookeeper.quorum","fly");
conf.set("hbase.cluster.distributed","true");
Connection connection = ConnectionFactory.createConnection(conf);
```

4.3 不断读取Kafka消息队列中的事件

```
while(true){           // 消息轮询
    // 100是超时时间，在改时间内poll会等待服务器返回数据
    ConsumerRecords<String,String> records=consumer.poll(100);
    // poll返回一个纪录列表
    // 每条记录都包含了记录的键值对
    for(ConsumerRecord<String,String> record:records){
        System.out.println("Get message: key=[" +record.key()+ "], message=["
+record.value() + "]" );
        col = record.key();
        value = record.value();
        //System.out.println("col: "+col+" value: "+ value);
        if (col != null){
            try {
                putData.send_data(col, value);
            }
            catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

4.4 对于每个事件，将其转化为HBase Put类的形式，并提交到HBase服务器中

```
// 使用Connection类的getTable()方法获取HTable类对象
HTable table = (HTable)
connection.getTable(TableName.valueOf("bigdata:HBase_Orders"));
// 声明一个Put类对象，并初始化其行键
Put put=new Put(key[0].getBytes());
// 使用put类的addColumn()方法加入将写入数据
// System.out.println(key[0] + key[1]);
if (key[1].equals("Order Detail"))
    put.addColumn("Order Detail".getBytes(), key[1].getBytes(),
value.getBytes());
else if(key[1].equals("Transaction"))
    put.addColumn("Transaction".getBytes(), key[1].getBytes(),
value.getBytes());
// 使用HTable类的put()方法将数据写入HBase表中
table.put(put);
table.close();
connection.close();
```

5、启动Flume

在本次实验中，Source 使用 HTTP Source，Sink 使用 Kafka Sink

跟安装相关的一些配置前面已经有写了，这里再简单讲一下

① 安装配置

进入配置文件目录下

```
cd flume-1.9.0/conf
```

将flume-env.sh.template文件复制一份，并重命名为flume-env.sh

```
cp flume-env.sh.template flume-env.sh
```

在flume-env.sh文件中写入以下内容：

```
JAVA_HOME = /usr/lib/jvm/java-8-openjdk-amd64
```

② 安装验证

在flume-1.9.0目录下执行以下命令

```
./bin/flume-ng version
```

若看到flume的版本号，则证明flume已成功安装

```
r52125@fly:~/flume/flume-1.9.0/bin$ ./flume-ng version
Flume 1.9.0
```

③ 其他配置

复制 flume-conf.properties.template 文件并命名为 flume-kafka-hbase.conf，修改其中 Source、Channel、Sink 的参数

(I) 复制文件

```
cp flume-conf.properties.template flume-kafka-hbase.conf
```

(II) 修改Source配置

```
# For each one of the sources, the type is defined
agent.sources.seqGenSrc.type = http

# sources绑定ip
agent.sources.seqGenSrc.bind = fly

# sources绑定端口
agent.sources.seqGenSrc.port = 44444
```

(III) 修改Channel配置

```
#Specify the channel the sink should use
agent.sinks.loggerSink.channel = memoryChannel

# Each channel's type is defined.
agent.channels.memoryChannel.type = memory

# Other config values specific to each type of channel(sink or source)
# can be defined as well
# In this case, it specifies the capacity of the memory channel

agent.channels.memoryChannel.capacity = 10000
agent.channels.memoryChannel.transactionCapacity = 10000
agent.channels.memoryChannel.byteCapacityBufferPercentage = 20
agent.channels.memoryChannel.byteCapacity = 800000
```

(IV) 修改Sink配置

```
# Each sink's type must be defined
agent.sinks.loggerSink.type = org.apache.flume.sink.kafka.KafkaSink

agent.sinks.loggerSink.kafka.topic = Kafka_Orders
agent.sinks.loggerSink.kafka.bootstrap.servers = fly:9092
agent.sinks.loggerSink.kafka.flumeBatchSize = 20
agent.sinks.loggerSink.kafka.producer.acks = 1
agent.sinks.loggerSink.kafka.producer.linger.ms = 1
agent.sinks.loggerSink.kafka.producer.compression.type = snappy
```

(V) 连接sources、sinks、channels

```
# The channel can be defined as follows.
agent.sources.seqGenSrc.channels = memoryChannel
```

```
#Specify the channel the sink should use
agent.sinks.loggerSink.channel = memoryChannel
```

(VI) 运行所编写的flume-kafka-hbase.conf, 启动Flume服务器进行数据收集

启动命令:

```
./bin/flume-ng agent --name agent --conf ./conf --conf-file ./conf/flume-kafka-hbase.conf -Dflume.root.logger=DEBUG,console
```



```

^Cr52125@fly:~/flume/flume-1.9.0$ ./bin/flume-ng agent --name agent --conf ./co
--conf-file ./conf/flume-kafka-hbase.conf -Dflume.root.logger=DEBUG,console
Info: Sourcing environment configuration script /home/r52125/flume/flume-1.9.0/
conf/flume-env.sh
Info: Including Hive libraries found via () for Hive access
+ exec /usr/lib/jvm/java-8-openjdk-amd64/bin/java -Xmx20m -Dflume.root.logger=D
EBUG,console -cp '/home/r52125/flume/flume-1.9.0/conf:/home/r52125/flume/flume-
1.9.0/lib/*:/lib/*' -Djava.library.path= org.apache.flume.node.Application --na
me agent --conf-file ./conf/flume-kafka-hbase.conf
2021-04-03 12:03:35,334 (main) [DEBUG - org.apache.flume.util.SSLUtil.initSysPr
opFromEnvVar(SSLUtil.java:95)] No global SSL keystore path specified.
2021-04-03 12:03:35,338 (main) [DEBUG - org.apache.flume.util.SSLUtil.initSysPr
opFromEnvVar(SSLUtil.java:95)] No global SSL keystore password specified.

```

不解：这里的DEBUG换成INFO，也可以，不知道这个参数的值有什么用处

验证：

① 测试flume是否可以正常接收http信息

新开一个终端，输入以下测试命令

```

curl -X POST -d' [{"headers":{"h1":"v1","h2":"v2"},"body":"hello body"}]'
http://fly:44444

```

```

r52125@fly:~$ curl -X POST -d' [{"headers":{"h1":"v1","h2":"v2"},"body":"hello b
ody"}]' http://fly:44444
r52125@fly:~$ cd flume/flume-1.9.0/

```

看flume的控制台输出，如下

```

2021-04-04 16:25:11,750 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:95)] Event: { headers
:{h1=v1, h2=v2} body: 68 65 6C 6C 6F 20 62 6F 64 79          hello bod
y }

```

明显看到已经接收到信息，内容为"hello body"

结论：Flume正常开启，并且可以正常进行数据收集

② 测试flume和kafka能否连通

新开一个终端，运行Kafka_comsumer，只运行接收的那部分代码

```

java -jar Kafka_comsumer.jar

```

flume的启动和输入的测试命令与上面相同

结果如下图所示

```

r52125@fly: ~/flume/flume-1.9.0
nihao
HTTP/1.1 400 No URI
Content-Type: text/html; charset=iso-8859-1
Content-Length: 49
Connection: close
Server: Jetty(9.4.6.v20170531)

<h1>Bad Message 400</h1><pre>reason: No URI</pre>
r52125@fly:~/flume/flume-1.9.0$ nc fly 44444
[{"headers":{"h1":"v1","h2":"v2"},"body":"hello body"}]
HTTP/1.1 400 HTTP/0.9 not supported
Content-Type: text/html; charset=iso-8859-1
Content-Length: 65
Connection: close
Server: Jetty(9.4.6.v20170531)

<h1>Bad Message 400</h1><pre>reason: HTTP/0.9 not suppo
rted</pre>
r52125@fly:~/flume/flume-1.9.0$ curl -X POST -d' [{"head
ers":{"h1":"v1","h2":"v2"},"body":"hello body"}]' http
://fly:44444
r52125@fly:~/flume/flume-1.9.0$

r52125@fly: ~/kafka/kafka-2.13-2.6.1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
- javaagent:<jarpath>[=<选项>]
  加载 Java 编程语言代理，请参阅 java.lang.instrument
nt
- splash:<imagepath>
  使用指定的图像显示启动屏幕
有关详细信息，请参阅 http://www.oracle.com/technetwork/java/javase/
documentation/index.html。
r52125@fly:~/kafka/kafka-2.13-2.6.1$ java -jar kafka_comsumer.jar
Error: Unable to access jarfile kafka_comsumer.jar
r52125@fly:~/kafka/kafka-2.13-2.6.1$ ls
bin  Kafka_comsumer.jar  LICENSE  NOTICE
config  libs  logs  site-docs
r52125@fly:~/kafka/kafka-2.13-2.6.1$ java -jar Kafka_comsumer.jar
Get message: key=[null], message=[hello body]
Get message: key=[null], message=[hello body]
Get message: key=[null], message=[hello body]
Get message: key=[null], message=[hello body]

```

结论：flume和kafka可以正常连接

6、启动具有数据生成功能的服务器

运行所提供的服务器 generatorData.py，开始产生数据并发送到 Flume 进行数据收集。

generatorData.py 产生的数据如下图所示，该服务器会自动将产生的每一条数据中“headers”作为报文头字段（也就是 HTTP 报文中的“header”），将“body”作为报文内容字段（也就是 HTTP 报文中的“data”）

```
[{'headers': {'key': '000000-Order Detail'}, 'body': '{"consumerId': 2673830, 'itemId': 2446813, 'itemCategory': 7039, 'amount': 43, 'money': 3313.3}"}]
```

测试：

与上一点的测试类似，只是把手动输入信息改成用python文件输入

执行python文件命令：

```
python3 generatorData.v2.py -h fly -p 44444
```

```
r52125@fly:~/flume/flume-1.9.0$ python3 generatorData.v2.py -h fly -p 44444
fly 44444
[{'headers': {'key': '000000-Order Detail'}, 'body': '{"consumerId': 2673830, 'itemId': 2446813, 'itemCategory': 7039, 'amount': 43, 'money': 3313.3}"}]
```

The image shows two terminal windows. The left window displays the output of the script, which is a JSON array of objects. Each object has a 'headers' field with a 'key' and a 'body' field with a JSON string. The right window shows the output of the script, which is a JSON array of objects. Each object has a 'headers' field with a 'key' and a 'body' field with a JSON string.

发现一点小错误，所有的数据都放在第一行了，需要改动一下代码。

更改行键后，再次运行，结果如下：

```

001110      .258, value=30
001110      column=Order Detail:'consumerId', timestamp=2021-04-18T21:
49:52.258, value=3513264
001110      column=Order Detail:'itemCategory', timestamp=2021-04-18T2
1:49:52.258, value=8453
001110      column=Order Detail:'itemId', timestamp=2021-04-18T21:49:5
2.258, value=6322337
001110      column=Order Detail:'money', timestamp=2021-04-18T21:49:52
.258, value=980.3
001110      column=Transaction:'completeTime', timestamp=2021-04-18T21
:50:11.065, value='2020-05-21 02:59:43'
001110      column=Transaction:'createTime', timestamp=2021-04-18T21:4
9:59.651, value='2020-01-26 19:17:49'
001110      column=Transaction:'deliveryTime', timestamp=2021-04-18T21
:50:09.280, value='2020-03-27 23:06:16'
001110      column=Transaction:'paymentTime', timestamp=2021-04-18T21:
49:59.651, value='2020-01-26 19:26:33'
001111      column=Order Detail:'amout', timestamp=2021-04-18T21:50:00
.285, value=25
001111      column=Order Detail:'consumerId', timestamp=2021-04-18T21:
50:00.285, value=5448903
001111      column=Order Detail:'itemCategory', timestamp=2021-04-18T2
1:50:00.285, value=5148
001111      column=Order Detail:'itemId', timestamp=2021-04-18T21:50:0
0.285, value=373591
001111      column=Order Detail:'money', timestamp=2021-04-18T21:50:00
.285, value=1924.5
001112      column=Order Detail:'amout', timestamp=2021-04-18T21:50:02
.389, value=84
001112      column=Order Detail:'consumerId', timestamp=2021-04-18T21:
50:02.389, value=3815400
001112      column=Order Detail:'itemCategory', timestamp=2021-04-18T2
1:50:02.389, value=491
001112      column=Order Detail:'itemId', timestamp=2021-04-18T21:50:0
2.389, value=4373684
001112      column=Order Detail:'money', timestamp=2021-04-18T21:50:02
.389, value=6677.7
001113      column=Order Detail:'amout', timestamp=2021-04-18T21:50:02
.649, value=12
001113      column=Order Detail:'consumerId', timestamp=2021-04-18T21:

```

7、查看HBase表

在~/HBase/hbase目录下，进入shell命令操作

```
./bin/hbase shell
```

查看自己的HBase表

```
scan 'bigdata:HBase_Orders'
```

```

001110      .258, value=30
001110      column=Order Detail:'consumerId', timestamp=2021-04-18T21:
49:52.258, value=3513264
001110      column=Order Detail:'itemCategory', timestamp=2021-04-18T2
1:49:52.258, value=8453
001110      column=Order Detail:'itemId', timestamp=2021-04-18T21:49:5
2.258, value=6322337
001110      column=Order Detail:'money', timestamp=2021-04-18T21:49:52
.258, value=980.3
001110      column=Transaction:'completeTime', timestamp=2021-04-18T21
:50:11.065, value='2020-05-21 02:59:43'
001110      column=Transaction:'createTime', timestamp=2021-04-18T21:4
9:59.651, value='2020-01-26 19:17:49'
001110      column=Transaction:'deliveryTime', timestamp=2021-04-18T21
:50:09.280, value='2020-03-27 23:06:16'
001110      column=Transaction:'paymentTime', timestamp=2021-04-18T21:
49:59.651, value='2020-01-26 19:26:33'
001111      column=Order Detail:'amout', timestamp=2021-04-18T21:50:00
.285, value=25
001111      column=Order Detail:'consumerId', timestamp=2021-04-18T21:
50:00.285, value=5448903
001111      column=Order Detail:'itemCategory', timestamp=2021-04-18T2
1:50:00.285, value=5148
001111      column=Order Detail:'itemId', timestamp=2021-04-18T21:50:0
0.285, value=373591
001111      column=Order Detail:'money', timestamp=2021-04-18T21:50:00
.285, value=1924.5
001112      column=Order Detail:'amout', timestamp=2021-04-18T21:50:02
.389, value=84
001112      column=Order Detail:'consumerId', timestamp=2021-04-18T21:
50:02.389, value=3815400
001112      column=Order Detail:'itemCategory', timestamp=2021-04-18T2
1:50:02.389, value=491
001112      column=Order Detail:'itemId', timestamp=2021-04-18T21:50:0
2.389, value=4373684
001112      column=Order Detail:'money', timestamp=2021-04-18T21:50:02
.389, value=6677.7
001113      column=Order Detail:'amout', timestamp=2021-04-18T21:50:02
.649, value=12
001113      column=Order Detail:'consumerId', timestamp=2021-04-18T21:

```

四、选做内容

1、使用Java API 替代 Java shell 操作

(1) 创建一个“HBase_Orders”表

java代码如下

```

public class createTable {
    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        conf.set("hbase.rootdir", "hdfs://fly:9000/hbase");
        conf.set("hbase.zookeeper.property.dataDir", "/home/r52125/HBase/hbase-2.3.4/zookeeper");
        conf.set("hbase.zookeeper.quorum", "fly");
        conf.set("hbase.cluster.distributed", "true");
        Connection connection = ConnectionFactory.createConnection(conf);
        HBaseAdmin admin = (HBaseAdmin)connection.getAdmin();
        TableDescriptorBuilder tdb =
            TableDescriptorBuilder.newBuilder(TableName.valueOf("bigdata:HBase_Orders"));
        List<ColumnFamilyDescriptor> listColumns = new ArrayList<>();
        ColumnFamilyDescriptor cfd_info =
            ColumnFamilyDescriptorBuilder.newBuilder("Order_Detail".getBytes()).build();
        ColumnFamilyDescriptor cfd_res =
            ColumnFamilyDescriptorBuilder.newBuilder("Transaction".getBytes()).build();
        listColumns.add(cfd_info);
        listColumns.add(cfd_res);
        tdb.setColumnFamilies(listColumns);
        TableDescriptor td= tdb.build();
        admin.createTable(td);
        admin.close();
        connection.close();
    }
}

```

打包成jar后，在目录下运行

```
java -jar create_Table.jar
```

```

r52125@fly:~/HBase/hbase-2.3.4$ java -jar create_table.jar
2021-04-15 17:30:30,819 WARN [main] util.NativeCodeLoader
(NativeCodeLoader.java:<clinit>(62)) - Unable to load nativ
e-hadoop library for your platform... using builtin-java cl
asses where applicable
2021-04-15 17:30:31,813 INFO [ReadOnlyZKClient-fly:2181@0x
3bd94634] zookeeper.ZooKeeper (Environment.java:logEnv(109)
) - Client environment:zookeeper.version=3.5.7-f0fdd52973d3
73ffd9c86b81d99842dc2c7f660e, built on 02/10/2020 11:30 GMT
2021-04-15 17:30:31,813 INFO [ReadOnlyZKClient-fly:2181@0x
3bd94634] zookeeper.ZooKeeper (Environment.java:logEnv(109)
) - Client environment:host.name=fly
2021-04-15 17:30:31,813 INFO [ReadOnlyZKClient-fly:2181@0x
3bd94634] zookeeper.ZooKeeper (Environment.java:logEnv(109)
) - Client environment:java.version=1.8.0_282
2021-04-15 17:30:31,813 INFO [ReadOnlyZKClient-fly:2181@0x

```

执行代码前后的变化:

```

hbase(main):019:0> list
TABLE
0 row(s)
Took 0.0100 seconds
=> []
hbase(main):020:0> list
TABLE
bigdata:HBase_Orders
1 row(s)
Took 0.0097 seconds
=> ["bigdata:HBase_Orders"]
hbase(main):021:0>

```

(2) 定时查看“HBase_Orders”表内的数据量，完成最简单的统计

```

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.filter.FirstKeyOnlyFilter;

import java.io.IOException;

public class count_Table {
    public static void main(String[] args) throws IOException,
        InterruptedException {
        Configuration conf = HBaseConfiguration.create();
        conf.set("hbase.rootdir", "hdfs://fly:9000/hbase");
        conf.set("hbase.zookeeper.property.dataDir", "/home/r52125/HBase/hbase-
2.3.4/zookeeper");
        conf.set("hbase.zookeeper.quorum", "fly");
        conf.set("hbase.cluster.distributed", "true");
        Connection connection = ConnectionFactory.createConnection(conf);
        // 使用Connection类的getTable()方法获取HTable类对象
        while(true) {
            HTable table = (HTable)
connection.getTable(TableName.valueOf("bigdata:HBase_Orders"));
            Scan scan = new Scan("00000".getBytes());
            scan.setFilter(new FirstKeyOnlyFilter());
            ResultScanner resultScanner = table.getScanner(scan);

```



```

        long rowCount = 0;
        for (Result result : resultScanner) {
            rowCount += result.size();
        }
        System.out.println("rowCount-->" + rowCount);
        Thread.sleep(5000);
    }
}

```

```

2021-04-15 21:19:41,700 INFO [ReadOnlyZKClient-fly:2181@0x2f490758-SendThread(fly:2181)] zookeeper.ClientCnxn (ClientCnxn.java:logStartConnect(1112)) - Opening socket connection to server fly/127.0.1.1:2181. Will not attempt to authenticate using SASL (unknown error)
2021-04-15 21:19:41,709 INFO [ReadOnlyZKClient-fly:2181@0x2f490758-SendThread(fly:2181)] zookeeper.ClientCnxn (ClientCnxn.java:primeConnection(959)) - Socket connection established, initiating session, client: /127.0.0.1:56324, server: fly/127.0.1.1:2181
2021-04-15 21:19:41,856 INFO [ReadOnlyZKClient-fly:2181@0x2f490758-SendThread(fly:2181)] zookeeper.ClientCnxn (ClientCnxn.java:onConnected(1394)) - Session establishment complete on server fly/127.0.1.1:2181, sessionId = 0x100000301ba0025, negotiated timeout = 90000
s rowCount-->7
rowCount-->7
rowCount-->7
rowCount-->7
rowCount-->7
rowCount-->7

```

每隔5秒查看一次HBase_Orders表中的数据量

这里因为我没有对HBase表进行任何操作，所以每次都是7

输出最好再加入时间，这个也比较容易，我就不加了

2、使用序列化技术