目录

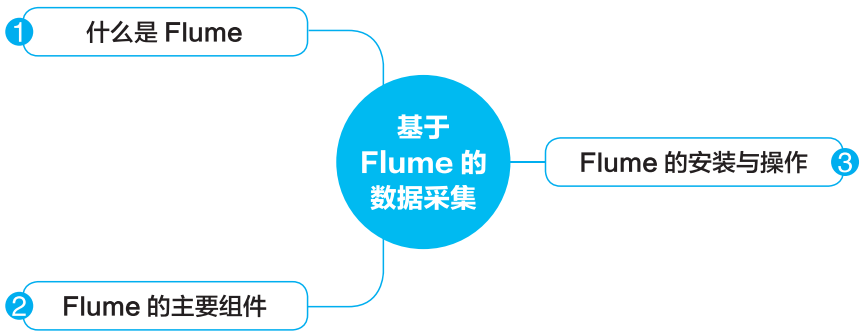
CONTENTS

第 1 章 基于 Flume 的数据采集

学习目标

1. 掌握 Flume 的安装与配置。
2. 掌握 Flume 采集方案的编写。
3. 熟悉 Flume 三大组件的作用。
4. 熟悉 Flume 三大组件的常用类型。

知识导图



1.1 什么是 Flume

Flume 是一个进行海量流式（streaming）事件数据采集的分布式系统。Flume 多用于对软件系统的日志文件数据的采集，目前已成为 Apache 软件基金会的顶级项目之一。

Flume 的运行平台，多以 Linux 系统为主，对 Java 的运行时环境要求不低于 Java 1.8 版。

1.2 Flume 的主要组件

Flume 采集系统以 Agent 为单位，一个 Agent 就是一个工作进程。一个完整的 Agent 由三个组件构成（见图 1.1），分别叙述如下。

（1）Source：用于从数据源接收数据（采集数据），需要设置数据的来源类型；

（2）Sink：用于将数据传递给目的地（保存数据），需要设置数据的目的地类型；

（3）Channel：用于 Source 和 Sink 的连接，并缓存传输的数据，需要设置缓存类型。

在 Flume 中，传输的基本单位是 event（事件），一个 event 包含 headers（头部）和 body（主体）两部分，这种设计类似于网络报文的结构，headers 用于标识信息，body 用于携带数据。

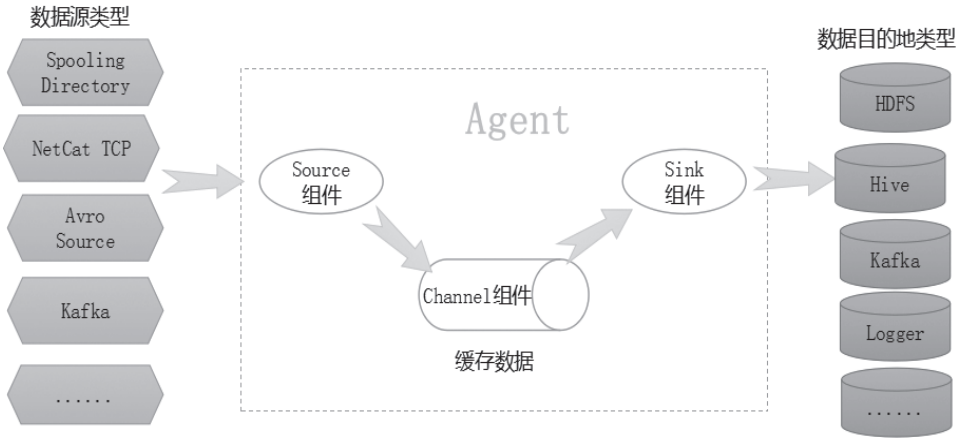


图 1.1　Flume 的数据采集模型

Source、Sink、Channel 的常用类型如表 1.1~ 表 1.3 所示。

表 1.1　Source 的常用类型

序号	类型名称	功能
1	spooldir	表明数据来源于磁盘文件目录
2	exec	表明数据来源于 Linux 命令
3	taildir	表明数据来源于文件的实时新增内容
4	avro	表明数据来源于指定 IP 和端口发送的 AVRO 消息
5	thrift	表明数据来源于指定 IP 和端口发送的 THRIFT 消息
6	netcat	表明数据来源于指定 IP 和端口发送的 netcat TCP 消息
7	netcatudp	表明数据来源于指定 IP 和端口发送的 netcat UDP 消息

表 1.2　Sink 的常用类型

序号	类型名称	功能
1	hdfs	表明数据的目的地是 HDFS
2	file_roll	表明数据的目的地是系统的本地目录
3	KafkaSink	表明数据的目的地是 Kafka 的生产者
4	avro	表明采用的是 AVRO 消息方式传递数据到数据目的地，需指定数据目的地的主机名和端口
5	hbase	表明数据的目的地是 HBase
6	asynchbase	表明以异步方式写数据到 HBase
7	hive	表明数据的目的地是 Hive
8	elasticsearch	表明数据的目的地是 elasticsearch

表 1.3　Channel 的常用类型

序号	类型名称	功能
1	memory	以内存作为数据的缓存，读写速度快，但数据可能会丢失
2	file	以磁盘文件作为数据的缓存，读写速度相对慢，但数据不会丢失

更多的 Source、Sink 和 Channel 类型，请查阅 Flume 的官方文档。

1.3 Flume 的安装与操作

1.3.1 安装 Flume

在浏览器的地址栏中输入 Flume 的官网地址“ http://flume.apache.org/download.html”，在打开的页面中，单击链接（见图 1.2 ），打开 Flume 的下载镜像地址页面。

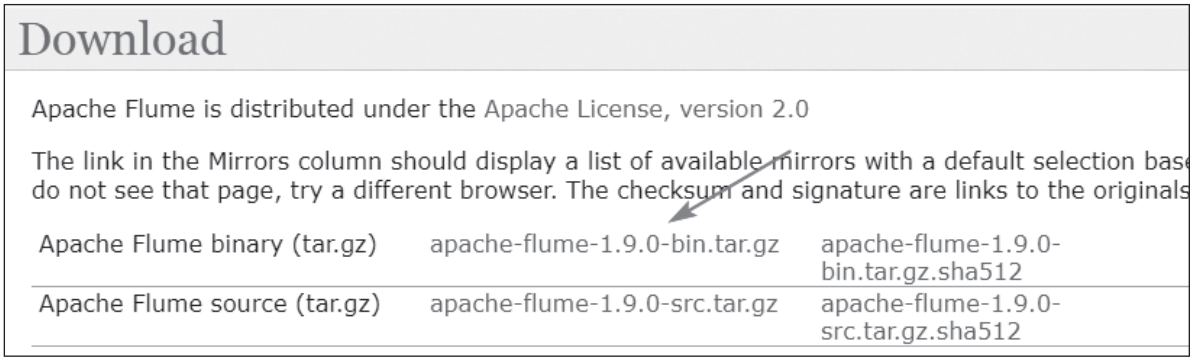


图 1.2　Flume 安装包压缩文件链接

在打开后的 Flume 下载镜像地址页面（见图 1.3 ），复制 Flume 的下载镜像地址。

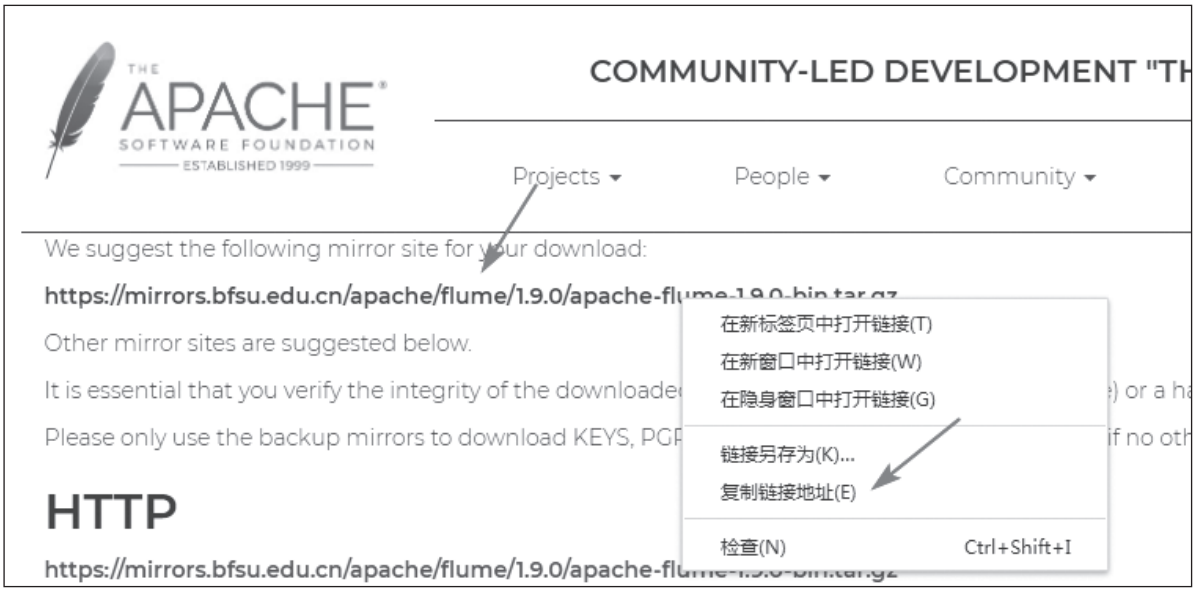


图 1.3　复制 Flume 的下载镜像地址

进入 Hadoop 集群主节点 node1 的“/usr/software/”目录，其中“software”为提前自建好的目录。

```
[root@node1 ~]# cd /usr/software/
```

下载 Flume 安装包到此目录，以下代码中所列地址只作为格式说明，不能作为实际下载地址使用，实际地址从官网中复制。

```
[root@node1 software]# wget https://mirrors.tuna.tsinghua.edu.cn/apache/flume/1.9.0
/apache-flume-1.9.0-bin.tar.gz
```

将下载的 Flume 文件解压到“/usr/app”目录，其中“app”为提前自建好的目录。

```
[root@node1 software]# tar -zxvf apache-flume-1.9.0-bin.tar.gz -C /usr/app/
```

执行以下命令，进入 Flume 的配置目录。

```
[root@node1 app]# cd /usr/app/apache-flume-1.9.0-bin/conf
```

将“conf”目录下的“flume-env.sh.template”文件复制，并重命名。

```
[root@node1 conf]# cp flume-env.sh.template flume-env.sh
```

打开复制重命名后的“flume-env.sh”文件。

```
[root@node1 conf]# vi flume-env.sh
```

向文件中添加 Java 的安装位置，以下代码只作为格式说明，实际安装位置根据自己安装的 Java 实际路径填写。

```
export JAVA_HOME=/usr/java/jdk
```

打开系统的环境配置文件。

```
[root@node1 conf]# vi /etc/profile
```

在文件中追加以下内容。

```
export FLUME_HOME=/usr/app/apache-flume-1.9.0-bin/
export FLUME_CONF_DIR=$FLUME_HOME/conf
export PATH=$PATH:$FLUME_HOME/bin
```

保存并退出“profile”文件后，执行下列命令，使更新后的环境变量生效。

```
[root@node1 conf]# source /etc/profile
```

执行下列命令，验证 Flume 是否安装成功。

```
[root@node1 conf]# flume-ng version
```

如果安装成功，会返回以下信息内容：

```
[root@node1 conf]# Flume 1.9.0
```

```
Source code repository: https://git-wip-us.apache.org/repos/asf/flume.git
Revision: d4fcab4f501d41597bc616921329a4339f73585e
Compiled by fszabo on Mon Dec 17 20:45:25 CET 2018
From source with checksum 35db629a3bda49d23e9b3690c80737f9
```

1.3.2 单机操作 Flume

单机操作 Flume 指的是只使用集群中的一个节点进行数据的采集（见图 1.4）。但无论是采用单机还是集群操作 Flume，Flume 的采集方案中至少要配置一个 Agent，一个 Agent 中可以包含多个 Source、Channel 和 Sink，一个 Source 可以指定多个 Channel，但是一个 Sink 只能绑定一个 Channel。

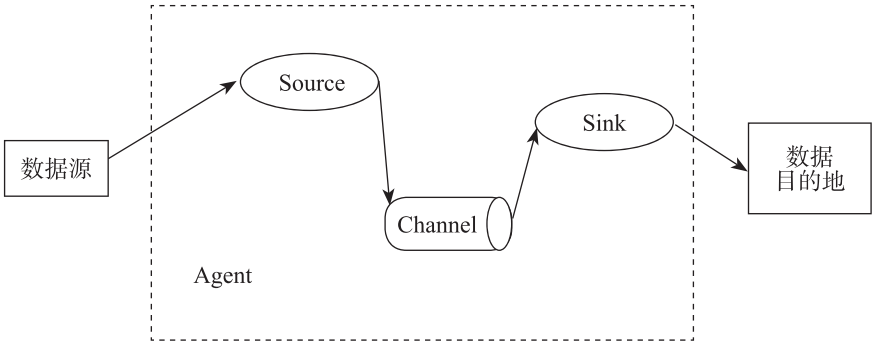


图 1.4　Flume 的单点采集模型

准备工作，进入主节点 node1 的“/usr/test”目录，“test”为提前自建好的目录。

```
[root@node1 conf]# cd /usr/test
```

在“test”目录下，创建子目录“flume-test”。

```
[root@node1 test]# mkdir flume-test
```

1. 单点采集数据到本地

进入“flume-test”目录。

```
[root@node1 test]# cd flume-test
```

在“flume-test”目录下，创建子目录“1”，以及“1”目录下的两个子目录“input”和“output”。

```
[root@node1 flume-test]# mkdir -p 1/input
[root@node1 flume-test]# mkdir  1/output
```

进入目录 1。

```
[root@node1 flume-test]# cd 1
```

创建新文件“file-logger.conf”，编写采集方案。

```
[root@node1 flume-test]# vi file-logger.conf
```

在文件中写入以下内容。

```
# 定义 Agent 中各组件名称，Agent 名为 a1
a1.sources = r1  # sources 名为 r1
a1.sinks = k1  # sinks 名为 k1
a1.channels = c1 # channels 名为 c1

a1.sources.r1.type = spooldir  # 配置 Sources 的数据源类型
a1.sources.r1.spoolDir = /usr/test/flume-test/1/input  # 根据数据源类型，配置数据源的位置

a1.sinks.k1.type = file_roll  # 配置 Sink 的数据目的地类型
a1.sinks.k1.sink.directory = /usr/test/flume-test/1/output # 根据数据目的地类型，配置目的地位置

a1.channels.c1.type = memory  # 配置 Channel 类型为 Memory
a1.channels.c1.capacity = 1000  # 指定缓存的最大值
a1.channels.c1.transactionCapacity = 100  # 指定缓存在每次事务中，读 Source 或写 Sink 的最大值

# 把 Source 和 Sink 绑定到 Channel 上
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

执行以下命令启动 Flume，此时，进程不会退出，根据文件中的 source 类型，会一直监控目录“1”下的子目录“input”的变化。

```
[root@node1 1]# flume-ng agent -c conf/ -f file-logger.conf -n a1 -Dflume.root.logger=INFO,console
```

以上命令中的参数说明如下：“flume-ng”表示启动一个 flume agent；“-c”指定 flume 的配置；“-f”指定采集方案文件；“-n”指定启动的 agent 的名称，该名称必须与采集方案文件中写的名称保持一致；“-Dflume.root.logger”指定日志信息展示的类型与方式。

接下来，使用 SecureFX 工具将日志素材文件 access.log 复制到“input”目录。

（1）在 SecureCRT 的“node1 节点”窗口（见图 1.5），单击工具栏上的 SecureFX 图标。

图 1.5　单击 SecureFX 图标

（2）在打开的“SecureFX”窗口的左边（见图 1.6），选择本地的“access.log”文件，在窗口右边，选择“node1”中的“/usr/test/flume-test/1/input”目录。

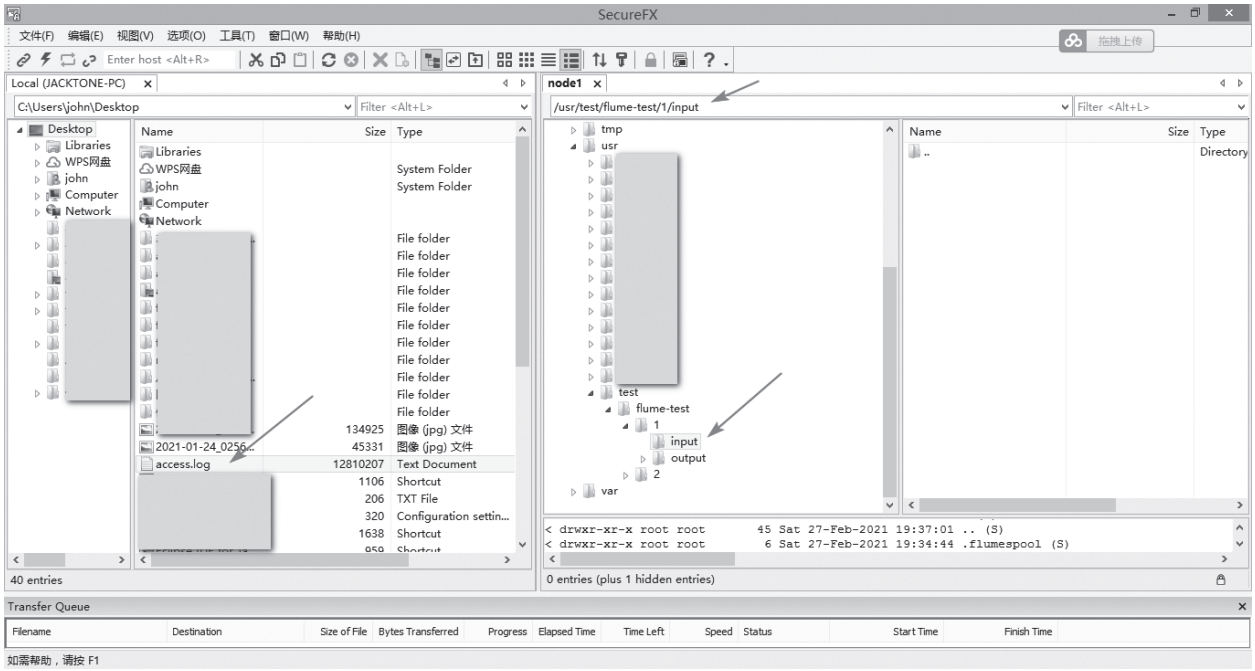


图 1.6　找到素材文件以及目的地目录

（3）将窗口左边选择的“access.log”文件（见图 1.7）拖动到最右边的空白窗口中。

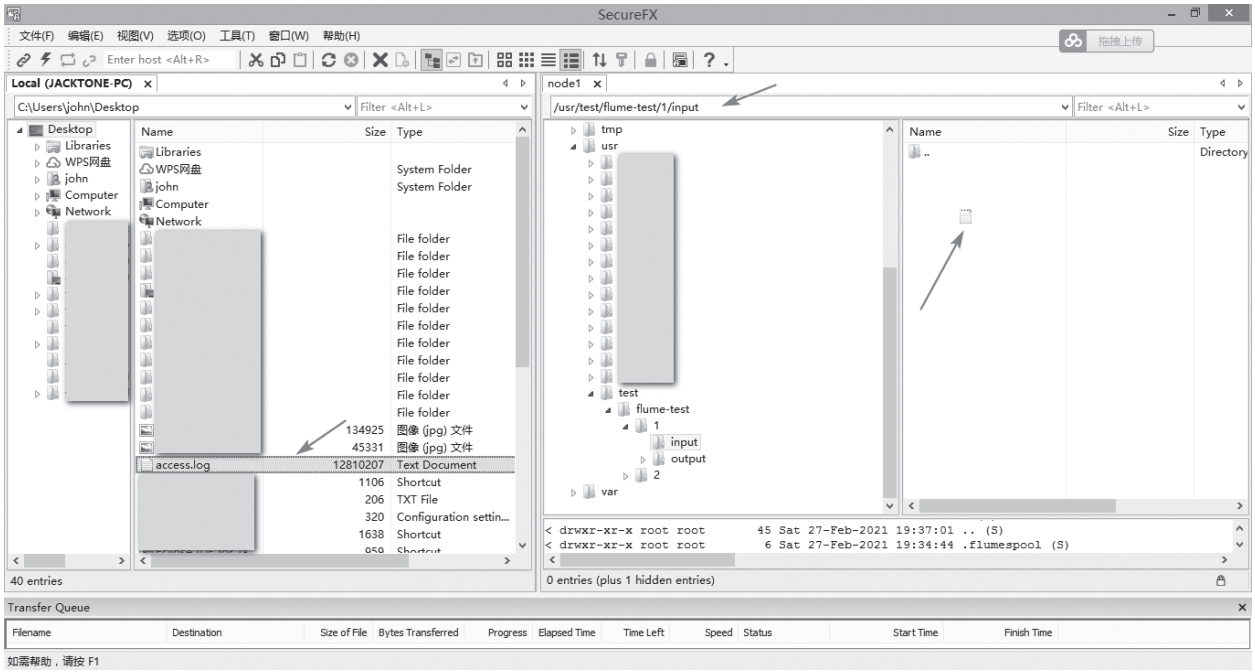


图 1.7　复制日志素材文件 access.log

（4）释放鼠标，完成文件的复制（见图 1.8）。

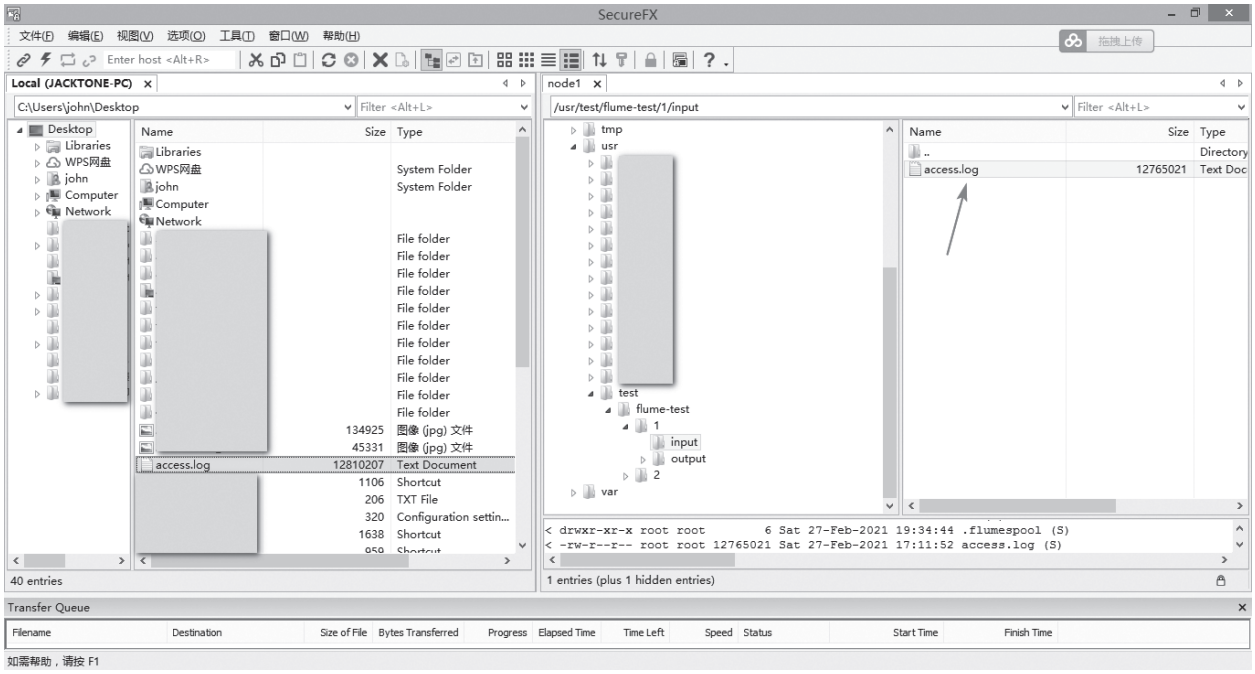


图 1.8　复制文件到“input”目录后的效果

复制后，Flume 采集进程会监控到“input”目录的变化，开始采集。采集后，输出如下反馈信息。

```
[root@node1 1]# INFO avro.ReliableSpoolingFileEventReader: Preparing to move file /usr/test/flume-test/1/input/access.log to /usr/test/flume-test/1/input/access.log.COMPLETED
```

从反馈信息中可以查看到“access.log”文件名后添加了一个尾部，整个文件名称变成了“access.log.COMPLETED”，而采集到的数据出现在“output”子目录中。最后，按“Ctrl+C”组合键可退出 Flume 进程。

提示：

凡是文件名后添加了“.COMPLETED”标记的，表明 Flume 已完成对该文件内容的采集。

2. 单点采集数据到 HDFS

在主节点 node1 中，将 hadoop 的“share/hadoop/common/lib”子目录中的“guava-27.0-jre.jar”文件复制到 flume 的子目录“lib”中。

```
[root@node1 1]# cp /usr/app/hadoop-3.3.1/share/hadoop/common/lib/guava-27.0-jre.jar /usr/app/apache-flume-1.9.0-bin/lib/
```

将 flume 子目录“lib”中的“guava-11.0.2.jar”文件重命名。

```
[root@node1 1]# mv /usr/app/apache-flume-1.9.0-bin/lib/guava-11.0.2.jar /usr/app/apache-flume-1.9.0-bin/lib/guava-11.0.2.jar.bak
```

在所有节点上启动 ZooKeeper。

```
[root@node1 1]# zkServer.sh start
```

在主节点 node1 上启动 HDFS。

```
[root@node1 1]# start-dfs.sh
```

在主节点 node1 上启动 YARN。

```
[root@node1 1]# start-yarn.sh
```

进入“flume-test”目录。

```
[root@node1 1]# cd /usr/test/flume-test
```

在“flume-test”目录下，一次创建好两级子目录“2/input”。

```
[root@node1 flume-test]# mkdir -p 2/input
```

进入目录“2”。

```
[root@node1 flume-test]# cd 2
```

创建新文件“hdfs-logger.conf”，编写采集方案。

```
[root@node1 flume-test]# vi hdfs-logger.conf
```

向文件中写入方案。

```
# 定义 Agent 中各组件名称，Agent 名为 a1
# sources 名为 r1
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# 配置 Sources
a1.sources.r1.type = spooldir  # 数据源类型
a1.sources.r1.spoolDir = /usr/test/flume-test/2/input

# 配置 Sink
a1.sinks.k1.type =  hdfs  # 采集后的数据目的地类型
# 指定采集后的数据存放到 hdfs 的路径，此时的节点 node1 应处于 Active 状态
# 用时间格式符产生路径名称，用以区分采集后的数据
a1.sinks.k1.hdfs.path = hdfs://node1:9820/flume/2/data/%Y-%m-%d/%H%M
# 设置间隔多少秒滚动生成一个 HDFS 文件，0 表示不按时间长度滚动生成文件，默认值是 30 秒
a1.sinks.k1.hdfs.rollInterval = 0
# 数据达到多少字节时滚动生成一个新的 HDFS 文件，如果是 0，则不根据大小滚动生成文件，默认值是 1024
a1.sinks.k1.hdfs.rollSize = 10240000
a1.sinks.k1.hdfs.rollCount = 0  # 0 表示不基于事件数量值滚动生成文件，默认值是 10
# 设置无响应状态下的超时时间（秒），默认值是 0，表示不关闭无响应状态下的文件
a1.sinks.k1.hdfs.idleTimeout = 3
# 输出的文件格式，DataStream 表示不压缩，SequenceFile 表示序列格式，CompressedStream 表示压缩格式，但需要额外设置 hdfs.codeC 属性，指定压缩类型
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.round = true  # 时间戳是否应该向下舍入。向下舍入时，会四舍五入到该值的最高倍数，小于当前时间
```

```
a1.sinks.k1.hdfs.roundValue = 10
a1.sinks.k1.hdfs.roundUnit = minute  # 舍入值的单位，可以是 second、minute 或 hour
a1.sinks.k1.hdfs.useLocalTimeStamp = true  # 使用本地时间戳

# 配置 Channel
a1.channels.c1.type = memory  # 缓存类型
a1.channels.c1.capacity = 1000  # 缓存大小
a1.channels.c1.transactionCapacity = 100  # 事务缓存大小

# 把 Source 和 Sink 绑定到 Channel 上
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

启动 Flume。

```
[root@node1 2]# flume-ng agent -c conf/ -f hdfs-logger.conf -n a1 -Dflume.root.logger=INFO,console
```

然后在集群中的另一个节点终端上，以节点 node2 为例，执行“ssh”命令登录到主节点 node1。

```
[root@node2 ~]# ssh node1
```

进入子目录“1”下的“input”目录。

```
[root@node1 ~]# cd /usr/test/flume-test/2/input
```

在当前目录，采取以下命令的方式为“input”目录动态创建数据素材文件。

```
[root@node1 input]# echo "hello world" > test.log
```

创建后，执行以下命令，可以退出“ssh”的登录。

```
[root@node1 input]# exit
```

Flume 采集进程输出采集完成的提示信息。

```
[root@node1 2]# INFO avro.ReliableSpoolingFileEventReader: Preparing to move file /usr/test/flume-test/2/input/access.log to /usr/test/flume-test/2/input/access.log.COMPLETED
```

在浏览器的地址栏中输入 HDFS 的地址“node1:9870”后，按“Enter”键，在页面的菜单栏，选择“Utilities”菜单列表（见图 1.9），选择第一个子菜单项“Browse the file system”，使页面跳转到“Browse Directory”页面。

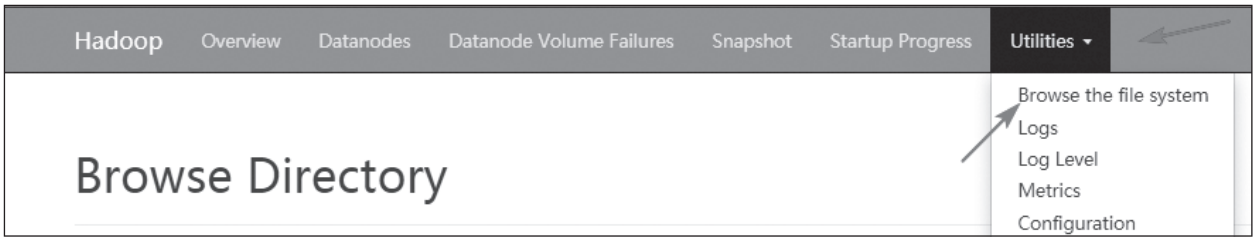


图 1.9　HDFS 的 Web 页面菜单栏

在“Browse Directory”页面的输入框输入 HDFS 的根目录“/”符（见图 1.10），单击“Go!”按钮。在页面刷新后，会显示 HDFS 当前根目录下的所有子目录。

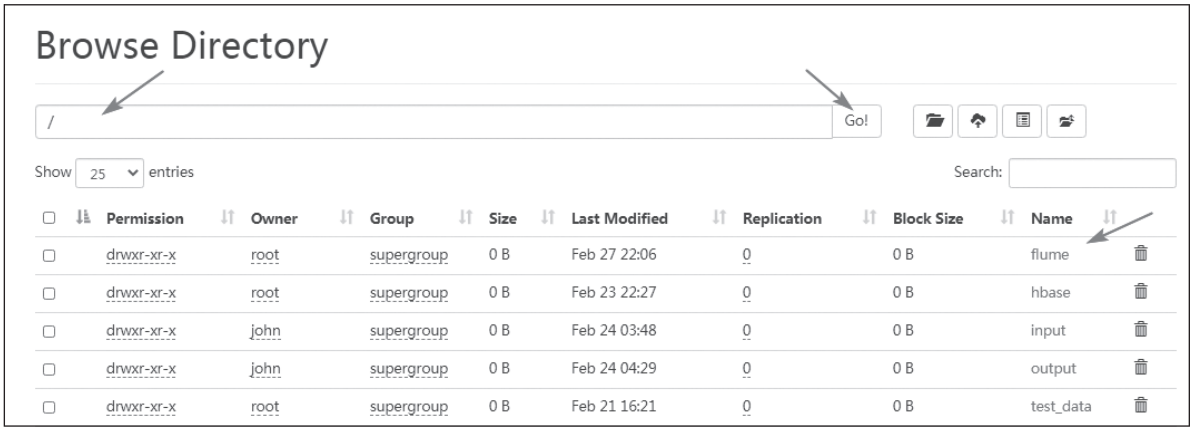


图 1.10　在 HDFS 的 Web 页面进行目录浏览

依次进入“flume”→“2”→“data”→“日期”→“时间”目录，最后在“时间”子目录中（见图 1.11）会显示存放 Flume 采集结果的两个文件。

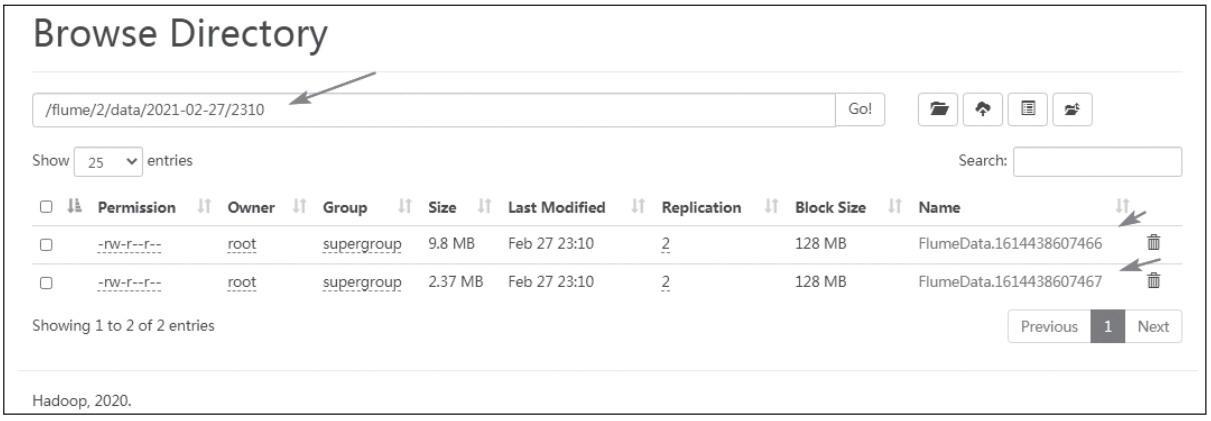


图 1.11　Flume 采集的结果

单击其中一个文件后，单击弹出窗口左上角的“Download”菜单（见图 1.12），可以将此文件下载到本地，用记事本程序查看文件内容；或者单击弹出窗口中间的“Head the file(first 32K)”菜单，可以在当前窗口中的下方区域查看文件的前 32K 大小内容；或单击弹出窗口的“Tail the file(last 32K)”菜单，在窗口中的下方区域查看文件的后 32K 大小内容。

3. 采集持续增加的内容，保存到 HDFS

进入“flume-test”目录。

```
[root@node1 2]# cd /usr/test/flume-test
```

在“flume-test”目录下，一次创建好两级子目录“3/input”。

```
[root@node1 flume-test]# mkdir -p 3/input
```

进入目录“3”。

```
[root@node1 flume-test]# cd 3
```

将原来目录“2”下的配置采集方案复制到目录“3”中，并重命名。

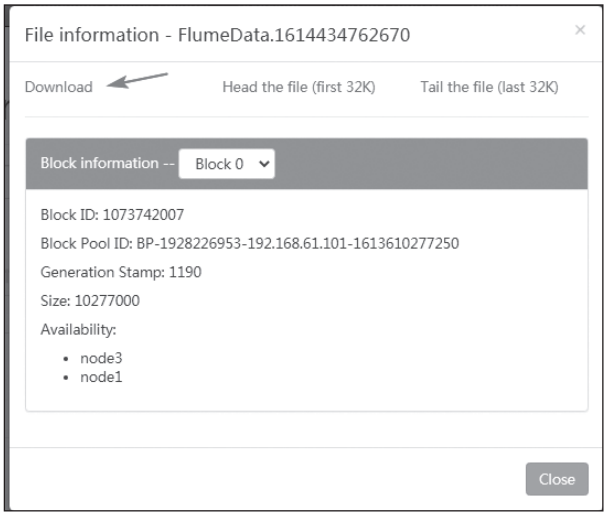


图 1.12　文件信息窗口

```
[root@node1 3]# cp ../2/hdfs-logger.conf  increase-logger.conf
```

打开“increase-logger.conf”文件。

```
[root@node1 3]# vi increase-logger.conf
```

在文件中修改以下内容（注：以下只列出了被修改处代码）。

```
# 配置 Sources
a1.sources.r1.type = exec  # 数据源类型
#a1.sources.r1.spoolDir = /usr/test/flume-test/3/input
# 指定 shell 命令，以及持续监测的目标文件
a1.sources.r1.command = tail -F /usr/test/flume-test/3/input/test.log

# 用日期时间来区分是哪个时段的采集数据
a1.sinks.k1.hdfs.path = hdfs://node1 的主机名 :9820/flume/3/data/%Y-%m-%d/%H%M
a1.sinks.k1.hdfs.rollInterval = 600
a1.sinks.k1.hdfs.rollSize = 0
a1.sinks.k1.hdfs.rollCount = 10000
a1.sinks.k1.hdfs.filePrefix = transaction_log  # 给文件名设置前缀

# 配置 channel
a1.channels.c1.type = file # 缓存类型
# 设置 channel 检查点文件的存放目录
a1.channels.c1.checkpointDir = /usr/test/flume-test/3/dataCheckpointDir
a1.channels.c1.dataDirs = /usr/test/flume-test/3/data # 设置 channel 日志文件的存放目录

#a1.channels.c1.capacity = 1000  # 缓存大小
#a1.channels.c1.transactionCapacity = 100  # 事务缓存大小
```

启动 Flume。

```
[root@node1 3]#  flume-ng agent -c conf/ -f increase-logger.conf -n a1 -Dflume.root.logger=INFO,console
```

另开启一个主节点 node1 终端，进入“3”目录的子目录“input”，创建“test.log”文件。

```
[root@node1 ~]# cd /usr/test/flume-test/3/input
[root@node1 ~]# vi test.log
```

向“test.log”文件中写入新内容，写入后关闭。反复操作几次打开文件、写入新内容、关闭过程。最后，登录 HDFS 页面，检测结果。

4. 对持续增加内容的第二种采集方式

进入“flume-test”目录。

```
[root@node1 3]#  cd /usr/test/flume-test
```

在“flume-test”目录下，一次创建好两级子目录“4/input”。

```
[root@node1 flume-test]#  mkdir -p 4/input
```

进入目录“4”。

```
[root@node1 flume-test]# cd 4
```

将原来目录“2”下的配置采集方案复制到目录“4”，并重命名。

```
[root@node1 4]# cp ../2/hdfs-logger.conf  tail-logger.conf
```

打开“tail-logger.conf”文件。

```
[root@node1 4]# vi tail-logger.conf
```

在文件中修改以下内容（注：以下只列出了被修改处代码）。

```
# 配置 Sources
a1.sources.r1.type = TAILDIR # 数据源类型
#a1.sources.r1.spoolDir = /usr/test/flume-test/4/input
# 以 .json 格式记录每次查询新增内容时的最后位置，以及查询的绝对路径和节点
a1.sources.r1.positionFile = /usr/test/flume-test/4/record/taildir_position.json
# 如果要监控多个文件，可以设置文件组，按组的方式，管理文件
a1.sources.r1.filegroups = f1 f2
a1.sources.r1.filegroups.f1 = /usr/test/flume-test/4/input/test.txt   # 指定组 1 的目标文件
# 指定组 2 的目标文件，文件名中采用了通配符
a1.sources.r1.filegroups.f2 = /usr/test/flume-test/4/input/.*log.*

a1.sinks.k1.hdfs.path = hdfs://node1:9820/flume/4/data/%Y-%m-%d/%H%M
# 配置 Channel 类型为 Memory
#a1.channels.c1.type = memory  # 缓存类型
#a1.channels.c1.capacity = 1000  # 缓存大小
#a1.channels.c1.transactionCapacity = 100  # 事务缓存大小
a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /usr/test/flume-test/4/dataCheckpointDir
a1.channels.c1.dataDirs = /usr/test/flume-test/4/data
```

启动 Flume。

```
[root@node1 4]# flume-ng agent -c conf/ -f tail-logger.conf -n a1 -Dflume.root.logger=INFO,console
```

另开启一个主节点 node1 终端，进入“4”目录的子目录“input”，创建“test.log”文件。并将“test.log”文件多次打开、关闭，并在打开后，添加不同的新内容。

```
[root@node1 ~]# cd /usr/test/flume-test/4/input
[root@node1 ~]# vi test.log
```

在“input”目录下，再创建一个“test.txt”文件。

```
[root@node1 ~]# vi test.txt
```

像“test.log”文件操作一样，也多次打开、关闭“test.txt”文件，并在打开后，添加不同的新内容。最后，登录 HDFS 页面，检测结果。

1.3.3 Flume 集群的搭建

将主节点 node1 上的环境变量文件复制到另外两个节点。

```
[root@node1 ~]# scp /etc/profile root@node2:/etc/
[root@node1 ~]# scp /etc/profile root@node3:/etc/
```

将主节点 node1 上安装好的 Flume 复制到另外两个节点。

```
[root@node1 ~]# scp -r /usr/app/apache-flume-1.9.0-bin/ root@node2:/usr/app/
[root@node1 ~]# scp -r /usr/app/apache-flume-1.9.0-bin/ root@node3:/usr/app/
```

在另外两个节点上，执行下列命令，使更新后的环境变量生效。

```
source /etc/profile
```

1.3.4 Flume 集群模式下的数据采集

场景：集群中三个节点分工负责，共同完成数据的采集。

集群采集模型如图 1.13 所示，在这个模型中，节点 node1 负责采集数据，节点 node1 与节点 node2、节点 node3 之间采用 avro 协议来传递消息；节点 node2、节点 node3 负责从节点 node1 接收数据，并将数据传递给数据目的地；由于节点 node1 需要对接 node2、node3 两个节点，因此在节点 node1 上需要设置两个 sink，组成一个 sinkgroup，两个 sink 之间的工作协同由 sinkgroup 负责完成。

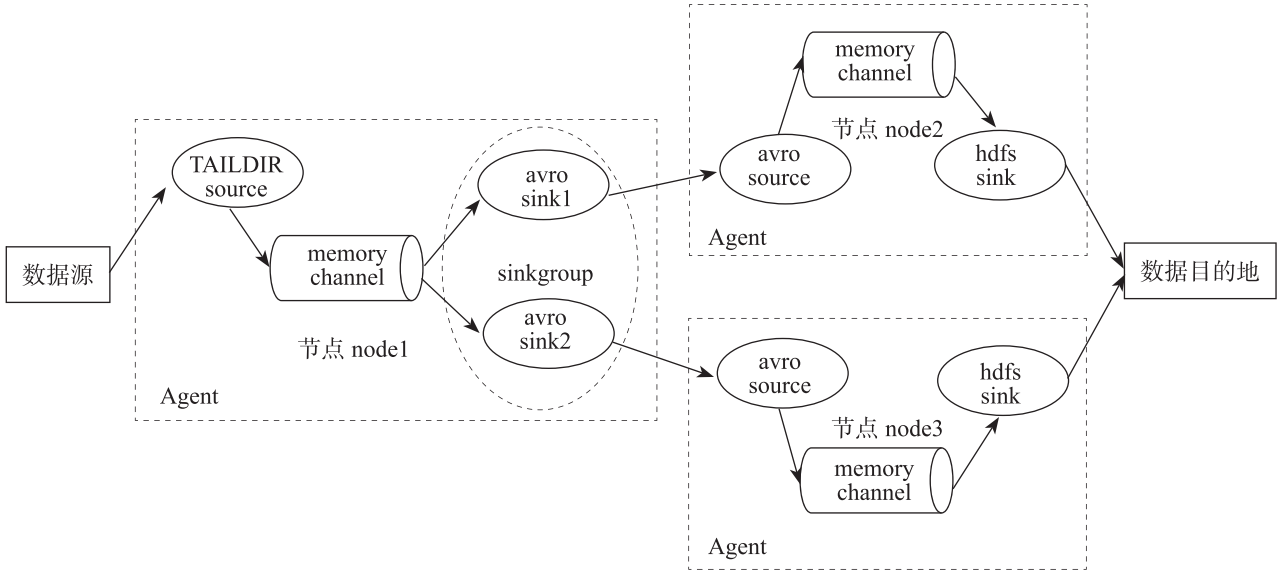


图 1.13　集群采集模型

sinkgroup 通过 sinks 属性来设置 sinkgroup 的成员，通过 processor 属性的设置，来实现多个 sink 之间的工作协调，从而达到提升 Flume 集群工作的可靠性目的。sinkgroup processor 的常用类型功能及配套属性如表 1.4 所示。

表 1.4　sinkgroup processor 的常用类型、功能及配套属性

序号	类型	功能	配套属性	功能
1	Load_balance	用于 sinkgroup 成员间工作压力的负载均衡	backoff	布尔类型，true 值表示将有故障的成员列入禁用黑名单
			selector	用于设置成员间的协调机制，其值有三种选项： （1）round_robin 表示采用轮询的方式； （2）random 表示采用随机的方式； （3）AbstractSinkSelector 表示自定义的方式
			maxTimeOut	用于设置故障成员被禁用的最长时间，毫秒级

续表

序号	类型	功能	配套属性	功能
2	failover	对有故障的 sinkgroup 成员进行管理与维护	priority	用于设置成员的优先级，数值越大，优先级越高。优先选择优先级最高的成员处理数据
			maxpenalty	用于设置故障成员的最大重试间隔时间，以毫秒为时间单位

在创建采集方案前，先完成以下准备工作。在 node2、node3 两节点上创建以下新目录。

```
mkdir -p /usr/test/flume-test
```

在所有三个节点上，进入各自的“flume-test”目录。

```
cd /usr/test/flume-test
```

在所有三个节点上，在“flume-test”目录下新建一个“group01”目录。

```
 mkdir group01
```

在所有三个节点上，进入各自的“group01”目录。

```
cd group01
```

在所有三个节点的“group01”目录下，创建采集方案文件。

```
vi flume-group.conf
```

1. 主节点的设置

在主节点 node1 的“flume-group.conf”文件中写入以下代码。

```
# 配置负载均衡模式下的一级采集方案
a1.sources = r1
# 配置两个 sink
a1.sinks = k1 k2
a1.channels = c1
# 配置 sources 组件
a1.sources.r1.type = TAILDIR
# 以 .json 格式记录每次查询新增内容时的最后位置，以及查询的绝对路径和节点
a1.sources.r1.positionFile = /usr/test/flume-test/group01/record/taildir_position.json
a1.sources.r1.filegroups = f1  # 为了简化，只设置 1 个文件组
a1.sources.r1.filegroups.f1 = /usr/test/flume-test/group01/input/test.txt # 指定组 1 的采集文件
# 设置 channels
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

a1.sinks.k1.channel = c1  # 将第 1 个 sink 绑定到 node2 节点
a1.sinks.k1.type = avro  # 设置 sink 的类型为 avro 消息
a1.sinks.k1.hostname = node2  # 指定接收 avro 消息的主机名
a1.sinks.k1.port = 8888  # 指定 avro 通信的端口号

a1.sinks.k2.channel = c1   # 将第 2 个 sink 绑定到 node3 节点
```

```
a1.sinks.k2.type = avro
a1.sinks.k2.hostname = node3
a1.sinks.k2.port = 8888

a1.sinkgroups = g1   # 设置集群内的负载均衡方案
a1.sinkgroups.g1.sinks = k1 k2  # 设置 sinkgroup 的成员
# 设置 processor 为 load_balance，实现成员间的负载均衡
a1.sinkgroups.g1.processor.type = load_balance
a1.sinkgroups.g1.processor.backoff = true
a1.sinkgroups.g1.processor.selector = random
a1.sinkgroups.g1.processor.maxTimeOut = 10000
```

2. 从节点的设置

在节点 node2 的“flume-group.conf”文件中写入以下代码。

```
# 配置负载均衡模式下的二级采集方案的一个 sink 分支
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# 配置 sources 组件
a1.sources.r1.type = avro   # 设置 source 的类型为 avro 消息
a1.sources.r1.bind = node2   # 绑定本机主机名
a1.sources.r1.port = 8888   # 指定 avro 通信的端口号
a1.sinks.k1.type = hdfs   # 数据的目的地类型
# 用日期时间来区分是哪个时段的采集数据，此时的 node1 节点应该是 Active 状态
a1.sinks.k1.hdfs.path = hdfs://node1:9820/flume/group01/data/%Y-%m-%d/%H%M
a1.sinks.k1.hdfs.rollInterval = 0   # 间隔多少秒生成一个 HDFS 文件
a1.sinks.k1.hdfs.rollSize = 10240000   # 数据量达到多少时生成一个 HDFS 文件
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.idleTimeout = 3
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.round = true
a1.sinks.k1.hdfs.roundValue = 10

a1.sinks.k1.hdfs.roundUnit = minute
a1.sinks.k1.hdfs.useLocalTimeStamp = true
# 缓存类型
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
# 把 Source 和 Sink 绑定到 Channel 上
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

将节点 node2 的 flume-group.conf 文件中的代码复制到节点 node3 的 flume-group.conf 文件中（或采用文件传递的方式，覆盖节点 node3 上的同名文件），在节点 node3 的 flume-group.conf 文件代码中，对主机名进行修改（此处省略了其他不需要修改的代码），如下所示。

```
# 配置 sources 组件
......
a1.sources.r1.bind = node3
```

3. 启动 Flume 集群与数据模拟载入

1）启动 Flume 集群

节点 node2、节点 node3 的 Flume Agent 启动应先于主节点 node1。在节点 node2、节点 node3 的“group01”目录下，分别启动 Flume Agent，然后启动主节点 node1 上的 Flume Agent（三个节点上所用命令的内容一致）。

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

2）模拟数据的动态载入

另开启一个主节点 node1 的终端，进入“group01”目录。

```
[root@node1 ~]# cd /usr/test/flume-test/group01
```

在“group01”目录下，创建“input”子目录。

```
[root@node1 group01]# mkdir input
```

进入“input”子目录。

```
[root@node1 group01]# cd input
```

在当前目录下，创建“load_data.sh”文件。

```
[root@node1 input]# vi load_data.sh
```

向文件中写入以下 shell 脚本代码。

```
#!/bin/bash
i=1
while true
do
echo $i.....`date +"%Y-%m-%d %H:%M:%S"` >> ./test.txt
let i++
sleep 1
done
```

保存并退出“load_data.sh”文件后，执行下列命令，修改文件的权限。

```
[root@node1 input]# chmod u+x load_data.sh
```

最后执行以下命令，运行此脚本程序。

```
[root@node1 input]# ./load_data.sh
```

3）退出并观察采集结果

按以下顺序退出各节点上运行的采集程序：

（1）在主节点 node1 的生产模拟数据的窗口，按“Ctrl+C”组合键，退出数据动态生产状态。

（2）在 node2、node3 两节点上，按“Ctrl+C”组合键，退出数据接收状态。

（3）最后，在主节点 node1 的数据采集窗口上，按“Ctrl+C”组合键，退出数据的采集状态。

然后登录 HDFS 的 Web 页面，按采集方案中设置的数据存储目的地路径，查看采集后的结果。

1.3.5 复杂场景下的 Flume 集群数据采集

1. 复杂场景分析

服务器 node2、服务器 node3 实时产生日志数据，日志数据有 admin.log 文件、user.log 文件、orders.log 文件，现在需要将服务器 node2、服务器 node3 上产生的这 3 类数据采集到服务器 node1 上，存于 HDFS，目录格式如以下（按照类型 + 日期的格式分别存储）。

```
/flume/complex01/logs-data/admin/20210301/**
/flume/complex01/logs-data/user/20210301/**
/flume/complex01/logs-data/orders/20210301/**
```

2. 采集方案设计

为实现上面的场景需求，将设计节点 node2、节点 node3 负责采集产生的日志数据，节点 node1 负责汇总数据到 HDFS，并借助 Flume 的 interceptor（拦截器）对数据进行标记，根据标记进行数据的归类。采集模型如图 1.14 所示。

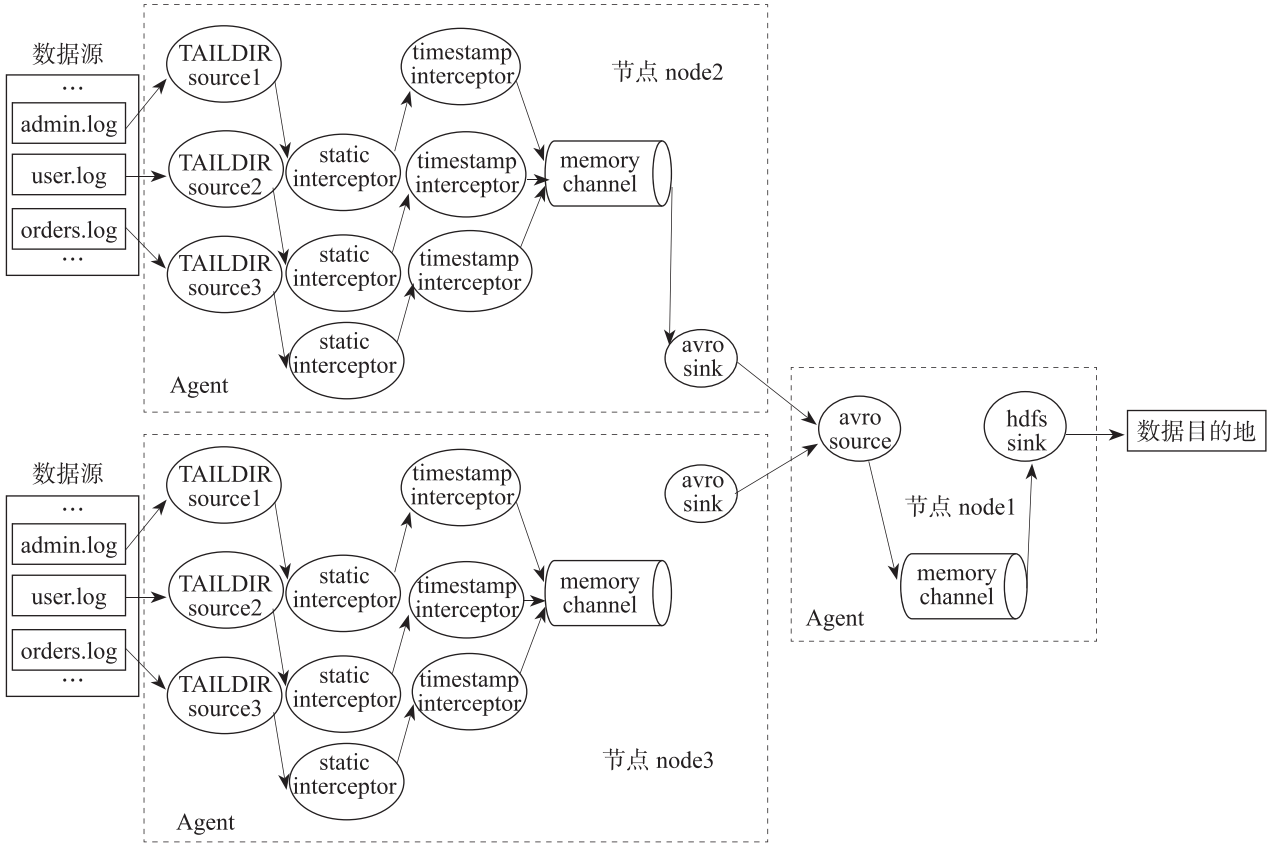


图 1.14　采集模型

Flume 的 interceptor 主要用于修改或删除 event 的 header 信息（event 即事件，Flume 传输的基本单位，参见前面 1.2 节中对 event 的讲解），interceptor 的常用类型功能及配套属性如表 1.5 所示。

表 1.5　interceptor 的常用类型功能及配套属性

序号	类型	功能	配套属性	功能
1	static	用于向所有 event 追加带有静态 value 的静态 header	key	用于设置追加到 event 的静态 header 的键名
			value	用于设置追加到 event 的静态 header 的值
			preserveExisting	布尔类型，用于设置是否保留已配置的静态 header。默认为 true
2	timestamp	用于在 event 的 header 中插入它处理事件的时间（毫秒）	headerName	用于设置插入 event 的 header 中的键名为 timestamp
			preserveExisting	布尔类型，用于设置是否保留已存在的时间戳。默认为 false
3	search_replace	用于使用正则表达式进行简单字符串的搜索和替换，与 Java matcher.replaceAll（）方法中使用的规则相同	searchPattern	用于设置正则表达式
			replaceString	用于设置要被替换的字符串
			charset	用于指定 event body 的字符集编码，默认值为 UTF-8
4	regex_filter	用于将 event 的 body 解释为文本，并使用正则表达式来匹配文本内容，达到对 event 的选择或过滤	regex	用于设置正则表达式
			excludeEvents	布尔类型，当为 true 时，进行 event 的过滤；当为 false 时，则进行选择。默认为 false
5	host	用于在 event 的 header 中插入当前 agent 所在节点的 IP 地址或主机名	preserveExisting	布尔类型，用于设置是否保留已存在的 host 头。默认为 false
			useIP	布尔类型，当为 true 时，插入 IP 地址；当为 false 时，插入主机名。默认为 true
			hostHeader	用于设置插入 event 的 header 中的键名为 host

更多 interceptor 类型及属性，请查阅 Flume 的官方文档。

采集方案中可以设置多个 interceptor，多个 interceptor 的执行顺序将根据采集方案中代码编写时给出的先后次序执行。

提示：

如果采集方案中，设置了 interceptor，那么 event header 中的默认时间戳会被清除，因此，需要在 interceptor 中重新配置 timestamp。

3. 代码实现

在创建采集方案前，在所有节点上，进入“flume-test”目录。

```
cd /usr/test/flume-test
```

在所有节点的“flume-test”目录下，创建新目录“complex01”。

```
mkdir complex01
```

在所有节点上，进入“complex01”目录。

```
cd complex01
```

1）在节点 node2 和节点 node3 编写采集方案

在新建的“complex01”目录下，创建“collect-logs.conf”文件。

```
vi collect-logs.conf
```

向文件中写入以下代码内容。

```
# 配置 Agent 组件，编写第一级日志采集方案
# 配置 3 个 Source，用于对应采集三类日志数据文件
a1.sources = r1 r2 r3
a1.sinks = k1
a1.channels  = c1

# 配置第 1 个 sources 组件
a1.sources.r1.type = TAILDIR
# 以 json 格式记录每次查询新增内容时的最后位置
a1.sources.r1.positionFile = /usr/test/flume-test/complex01/record/taildir_position_admin.json
a1.sources.r1.filegroups = f1 # 为了简化，只设置 1 个文件组
a1.sources.r1.filegroups.f1 = /usr/test/flume-test/complex01/input/admin.log # 指定组 1 的目标文件
# 为第 1 个 source 配置两个拦截器，名称分别为 i1 和 i2，名称之间用英文的空格隔开
a1.sources.r1.interceptors = i1 i2
a1.sources.r1.interceptors.i1.type = static  # 第 1 个拦截器的类型为 static
# 设置键名为 type，注意这个键名 type 会在下游的 Agent 中作为变量形式 %{type} 被调用
a1.sources.r1.interceptors.i1.key = type
a1.sources.r1.interceptors.i1.value = admin
# 第 1 个拦截器的类型为 timestamp，会在下游的 Agent 中作为变量形式 %Y-%m-%d 被调用
a1.sources.r1.interceptors.i2.type = timestamp

# 配置第 2 个 sources 组件
a1.sources.r2.type = TAILDIR
# 以 .json 格式记录每次查询新增内容时的最后位置
a1.sources.r2.positionFile = /usr/test/flume-test/complex01/record/taildir_position_user.json
a1.sources.r2.filegroups = f2  # 为了简化，只设置 1 个文件组
a1.sources.r2.filegroups.f2 = /usr/test/flume-test/complex01/input/user.log  # 指定组 2 的目标文件

# 为第 2 个 source 配置两个拦截器，名称分别为 i1 和 i2，名称之间用英文的空格间隔
a1.sources.r2.interceptors = i1 i2
a1.sources.r2.interceptors.i1.type = static
a1.sources.r2.interceptors.i1.key = type
a1.sources.r2.interceptors.i1.value = user
a1.sources.r2.interceptors.i2.type = timestamp
# 配置第 3 个 sources 组件
a1.sources.r3.type = TAILDIR
# 以 .json 格式记录每次查询新增内容时的最后位置
a1.sources.r3.positionFile = /usr/test/flume-test/complex01/record/taildir_position_orders.json
a1.sources.r3.filegroups = f3  # 为了简化，只设置 1 个文件组
```

```
a1.sources.r3.filegroups.f3 = /usr/test/flume-test/complex01/input/orders.log  # 指定组 3 的目标文件

# 为第 3 个 source 配置两个拦截器
a1.sources.r3.interceptors = i1 i2
a1.sources.r3.interceptors.i1.type = static
a1.sources.r3.interceptors.i1.key = type
a1.sources.r3.interceptors.i1.value = orders
a1.sources.r3.interceptors.i2.type = timestamp

# 设置 channels
a1.channels.c1.type = memory
a1.channels.c1.capacity = 2000000
a1.channels.c1.transactionCapacity = 100000

# 设置将 sink 绑定到 node1 节点
a1.sinks.k1.channel = c1
a1.sinks.k1.type = avro
a1.sinks.k1.hostname = node1
a1.sinks.k1.port = 8888

# 将 sources 与 channel 进行关联绑定
a1.sources.r1.channels = c1
a1.sources.r2.channels = c1
a1.sources.r3.channels = c1
```

在“complex01”目录下，创建“input”子目录。

```
mkdir input
```

进入“input”子目录。

```
cd input
```

在当前目录下，创建“load_data.sh”文件。

```
vi load_data.sh
```

向文件中写入以下代码内容。

```
#!/bin/bash
i=1
while true
do
echo $i.....`date +"%Y-%m-%d %H:%M:%S"`.....admin >> ./admin.log
echo $i.....`date +"%Y-%m-%d %H:%M:%S"`.....user >> ./user.log
echo $i.....`date +"%Y-%m-%d %H:%M:%S"`.....orders >> ./orders.log
let i++
sleep 1
done
```

保存并退出文件后，执行下列命令，修改文件的权限。

```
chmod u+x load_data.sh
```

提示：

在 node2 和 node3 两节点上，都需要完成以上的操作步骤内容。

2）在节点 node1 上编写采集方案

在新建的“complex01”目录下，创建“collect-logs.conf”文件。

```
vi collect-logs.conf
```

向文件中写入以下代码内容。

```
# 配置 Agent 组件，编写第二级日志收集汇总方案
a1.sources = r1
a1.sinks = k1
a1.channels  = c1
# 配置 sources 组件，对接 node2 和 node3 两节点的 avro 消息
a1.sources.r1.type = avro
a1.sources.r1.bind = node1
a1.sources.r1.port = 8888

# 设置 channels
a1.channels.c1.type = memory
a1.channels.c1.capacity = 20000
a1.channels.c1.transactionCapacity = 10000

# 配置 sink
a1.sinks.k1.type = hdfs  # 指定数据目的地类型
# 变量｛type｝来至对上游 Agent 中 static 拦截器键名的调用，用于分类采集的数据
# 此时的 node1 节点应是 Active 状态
# %Y-%m-%d 来至对上游 Agent 中 timestamp 拦截器时间戳的调用，用于按时间段分类采集的数据
a1.sinks.k1.hdfs.path = hdfs://node1:9820/flume/complex01/logs-data/%{type}/%Y-%m-%d
a1.sinks.k1.hdfs.rollInterval = 0              # 不按时间周期滚动生成 HDFS 文件
a1.sinks.k1.hdfs.rollSize = 10485760           # 按数据量大小滚动生成 HDFS 文件，以字节为单位
a1.sinks.k1.hdfs.rollCount = 0                 # 不按事件数量值滚动生成 HDFS 文件
a1.sinks.k1.hdfs.filePrefix = events           # 设置文件前缀
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.writeFormat = Text
a1.sinks.k1.hdfs.threadsPoolSize = 10          # 每个 HDFS sink 可操作的线程数

# 将 sources、sink 与 channel 进行关联绑定
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

提示：

hdfs.callTimeout 属性在 Flume 1.9 版本中已被弃用，因此在配置 sink 时，不需要设置该属性。

4. 启动 Flume 采集集群

在所有节点启动 ZooKeeper。

```
zkServer.sh start
```

在主节点 node1 上执行以下命令，启动 HDFS 集群。

```
[root@node1 ~]# start-dfs.sh
```

在主节点 node1 上执行以下命令，启动 YARN 集群。

```
[root@node1 ~]# start-yarn.sh
```

在所有节点上，进入“complex01”子目录。

```
cd /usr/test/flume-test/complex01
```

在所有节点上，执行以下命令，启动“collect-logs.conf”采集方案。操作时，严格按主节点 node1 →从节点 node2 →从节点 node3 的先后顺序，分别启动各自的“collect-logs.conf”采集方案文件，等上一个节点的采集方案启动后，再启动下一个节点上的采集方案。

```
flume-ng agent -c conf/ -f collect-logs.conf -n a1 -Dflume.root.looger=INFO,console
```

主节点 node1 的采集方案成功启动后的反馈信息如图 1.15 所示。

```
2021-07-06 10:41:48,375 INFO node.Application: Starting Channel c1
2021-07-06 10:41:48,376 INFO node.Application: Waiting for channel: c1 to start. Sleeping for 500 ms
2021-07-06 10:41:48,856 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: CHANNEL, name: c1
: Successfully registered new MBean.
2021-07-06 10:41:48,857 INFO instrumentation.MonitoredCounterGroup: Component type: CHANNEL, name: c1 started
2021-07-06 10:41:48,878 INFO node.Application: Starting Sink k1
2021-07-06 10:41:48,881 INFO node.Application: Starting Source r1
2021-07-06 10:41:48,883 INFO source.AvroSource: Starting Avro source r1: { bindAddress: node1, port: 8888 }...
2021-07-06 10:41:48,903 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: SINK, name: k1: S
uccessfully registered new MBean.
2021-07-06 10:41:48,903 INFO instrumentation.MonitoredCounterGroup: Component type: SINK, name: k1 started
2021-07-06 10:41:49,839 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: SOURCE, name: r1:
 Successfully registered new MBean.
2021-07-06 10:41:49,839 INFO instrumentation.MonitoredCounterGroup: Component type: SOURCE, name: r1 started
2021-07-06 10:41:49,844 INFO source.AvroSource: Avro source r1 started.
```

图 1.15　node1 采集方案成功启动后的反馈信息

从节点 node2 的采集方案成功启动后的反馈信息如图 1.16 所示，从节点 node3 的反馈信息与从节点 node2 反馈的信息类似，不再列举。

```
2021-07-09 04:53:44,357 INFO api.NettyAvroRpcClient: Using default maxIOWorkers
2021-07-09 04:53:44,417 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: SOURCE, name: r1:
 Successfully registered new MBean.
2021-07-09 04:53:44,417 INFO instrumentation.MonitoredCounterGroup: Component type: SOURCE, name: r1 started
2021-07-09 04:53:44,461 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: SOURCE, name: r3:
 Successfully registered new MBean.
2021-07-09 04:53:44,461 INFO instrumentation.MonitoredCounterGroup: Component type: SOURCE, name: r3 started
2021-07-09 04:53:52,288 INFO sink.AbstractRpcSink: Rpc sink k1 started.
```

图 1.16　node2 采集方案成功启动后的反馈信息

在从节点 node2、从节点 node3 上的采集方案成功启动后，主节点 node1 会有新的反馈信息返回（见图 1.17），表明从节点 node2、从节点 node3 已经分别与主节点 node1 建立连接。

```
2021-07-06 10:50:22,474 INFO ipc.NettyServer: [id: 0xac61ebcb, /192.168.61.102:39900 => /192.168.61.101:8888] OPEN
2021-07-06 10:50:22,475 INFO ipc.NettyServer: [id: 0xac61ebcb, /192.168.61.102:39900 => /192.168.61.101:8888] BOUND: /1
92.168.61.101:8888
2021-07-06 10:50:22,475 INFO ipc.NettyServer: [id: 0xac61ebcb, /192.168.61.102:39900 => /192.168.61.101:8888] CONNECTED
: /192.168.61.102:39900
2021-07-06 10:50:31,443 INFO ipc.NettyServer: [id: 0x19d0ad77, /192.168.61.103:48600 => /192.168.61.101:8888] OPEN
2021-07-06 10:50:31,455 INFO ipc.NettyServer: [id: 0x19d0ad77, /192.168.61.103:48600 => /192.168.61.101:8888] BOUND: /1
92.168.61.101:8888
2021-07-06 10:50:31,455 INFO ipc.NettyServer: [id: 0x19d0ad77, /192.168.61.103:48600 => /192.168.61.101:8888] CONNECTED
: /192.168.61.103:48600
```

图 1.17　node2、node3 已经分别与 node1 建立连接

再为 node2、node3 两节点各开一个新终端窗口，然后进入各自的“input”子目录。

```
cd /usr/test/flume-test/complex01/input
```

在节点 node2 和节点 node3 上运行脚本程序，生成模拟数据。

```
./load_data.sh
```

运行脚本程序后，原节点 node2 的采集方案窗口会有新的反馈信息返回（见图 1.18），表明节点 node2 的 Flume 已经监测到数据的变化，开始了数据的采集，节点 node3 的反馈信息与节点 node2 反馈的信息类似，不再列举。

```
2021-07-09 04:58:39,572 INFO taildir.ReliableTaildirEventReader: Opening file: /usr/test/flume-test/complex01/input/user.log, inode: 12737885, pos: 0
2021-07-09 04:58:39,577 INFO taildir.ReliableTaildirEventReader: Opening file: /usr/test/flume-test/complex01/input/admin.log, inode: 12737877, pos: 0
2021-07-09 04:58:39,577 INFO taildir.ReliableTaildirEventReader: Opening file: /usr/test/flume-test/complex01/input/orders.log, inode: 12737895, pos: 0
```

图 1.18　node2 的 Flume 已经监测到数据的变化

在从节点 node2、从节点 node3 开始采集数据后，主节点 node1 也会有新的反馈信息返回（见图 1.19），表明主节点 node1 的 Flume 在 HDFS 上分别创建了三个存放采集结果的目录。

```
2021-07-06 10:55:15,363 INFO hdfs.HDFSDataStream: Serializer = TEXT, UseRawLocalFileSystem = false
2021-07-06 10:55:15,881 INFO hdfs.BucketWriter: Creating hdfs://node1:9820/flume/complex01/logs-data/user/2021-07-06/events.1625540115364.tmp
2021-07-06 10:55:21,519 INFO hdfs.HDFSDataStream: Serializer = TEXT, UseRawLocalFileSystem = false
2021-07-06 10:55:21,637 INFO hdfs.BucketWriter: Creating hdfs://node1:9820/flume/complex01/logs-data/orders/2021-07-06/events.1625540121520.tmp
2021-07-06 10:55:21,744 INFO hdfs.HDFSDataStream: Serializer = TEXT, UseRawLocalFileSystem = false
2021-07-06 10:55:21,814 INFO hdfs.BucketWriter: Creating hdfs://node1:9820/flume/complex01/logs-data/admin/2021-07-06/events.1625540121744.tmp
```

图 1.19　node1 的新反馈信息

5. 退出并观察采集结果

按以下顺序退出各节点上运行的采集程序。

（1）在节点 node2、节点 node3 的模拟数据生成窗口，按“Ctrl+C”组合键，退出数据动态生成状态。

（2）在节点 node2、节点 node3 的原窗口，按“Ctrl+C”组合键，退出采集状态。

（3）在主节点 node1 上，按“Ctrl+C”组合键，退出分类汇总状态。

然后登录 HDFS 页面，检测采集结果（见图 1.20），在 HDFS 目录下，产生了三个对应的子目录。

图 1.20　存放三个结果数据所对应的子目录

以“admin”子目录为例，其采集后的结果数据文件如图 1.21 所示。

图 1.21　采集后的结果数据文件

单击该文件，在弹出的新窗口中，单击窗口中间的“Head the file(first 32K)”链接（见图 1.22），在窗口下方，展示了文件中的内容，结果显示文件中只有从 admin.log 数据源中采集到的数据，说明设计的采集方案达到了场景的需求。其他目录下的采集结果不再列举，请在项目完成后自行查看。

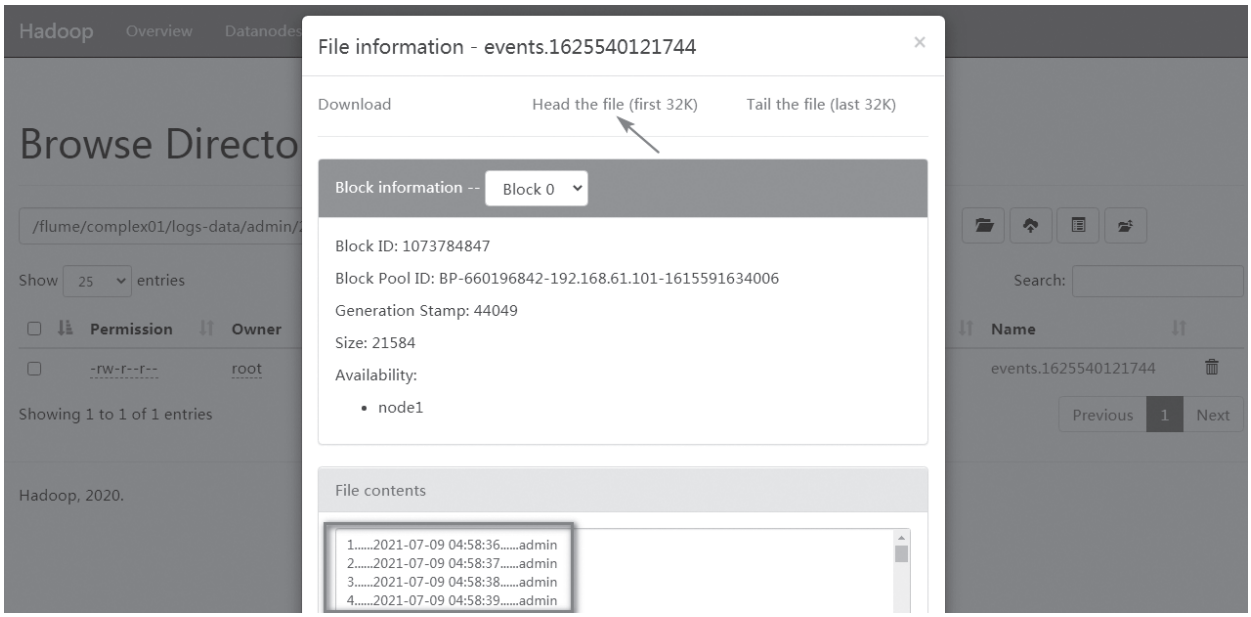


图 1.22　查看文件中的数据

6. 解决出现的问题

1）问题 1

如果所有程序启动后，在主节点 node1 的窗口中出现以下信息内容。

```
[root@node1 complex01]# org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.ipc.StandbyException): Operation category READ is not supported in state standby. Visit https://s.apache.org/sbnn-error
```

这是因为节点 node1 的 NameNode 是 standby 状态造成的，解决步骤如下：

（1）在节点 node2、节点 node3 的模拟数据生成窗口，按“Ctrl+C”组合键，退出数据动态生成状态；

（2）在节点 node2、节点 node3 的原窗口，按“Ctrl+C”组合键，退出采集状态；

（3）在主节点 node1 上，按“Ctrl+C”组合键，退出分类汇总状态；

（4）在节点 node2 上，执行以下命令，停掉节点 node2 的 NameNode；

```
[root@node2 complex01]# hdfs --daemon stop namenode
```

（5）待节点 node2 上的 NameNode 停掉后，在节点 node2 上执行以下命令查看节点 node1 的 NameNode 状态；

```
[root@node2 complex01]#  hdfs haadmin -getServiceState nn1
```

（6）待节点 node1 的 NameNode 状态返回结果显示为 Active 后，则在节点 node2 上，执行下列命令，重新启动节点 node2 的 NameNode；

```
[root@node2 complex01]#  hdfs --daemon start namenode
```

（7）按 1.3.5 小节“4. 启动 Flume 采集集群”的步骤，重新测试。

2）问题 2

如果在节点 node2 或节点 node3 启动采集方案后，反馈报错信息（见图 1.23）。

```
2021-07-09 04:47:02,835 INFO taildir.ReliableTaildirEventReader: Updating position from position file: /usr/test/flume-
test/complex01/record/taildir_position_orders.json
2021-07-09 04:47:02,842 INFO taildir.ReliableTaildirEventReader: Updating position from position file: /usr/test/flume-
test/complex01/record/taildir_position_admin.json
2021-07-09 04:47:02,843 INFO taildir.ReliableTaildirEventReader: Updating position from position file: /usr/test/flume-
test/complex01/record/taildir_position_user.json
2021-07-09 04:47:02,867 ERROR taildir.ReliableTaildirEventReader: Failed loading positionFile: /usr/test/flume-test/com
plex01/record/taildir_position_orders.json
java.io.EOFException: End of input at line 1 column 1
        at com.google.gson.stream.JsonReader.nextNonWhitespace(JsonReader.java:954)
        at com.google.gson.stream.JsonReader.nextValue(JsonReader.java:771)
        at com.google.gson.stream.JsonReader.peek(JsonReader.java:367)
        at com.google.gson.stream.JsonReader.expect(JsonReader.java:337)
        at com.google.gson.stream.JsonReader.beginArray(JsonReader.java:306)
        at org.apache.flume.source.taildir.ReliableTaildirEventReader.loadPositionFile(ReliableTaildirEventReader.java:
111)
        at org.apache.flume.source.taildir.ReliableTaildirEventReader.<init>(ReliableTaildirEventReader.java:96)
        at org.apache.flume.source.taildir.ReliableTaildirEventReader.<init>(ReliableTaildirEventReader.java:49)
        at org.apache.flume.source.taildir.ReliableTaildirEventReader$Builder.build(ReliableTaildirEventReader.java:355
)
        at org.apache.flume.source.taildir.TaildirSource.start(TaildirSource.java:105)
```

图 1.23　因颠倒了启动顺序，而导致的位置索引记录文件报错

说明在启动采集方案时，没有严格按前面启动 Flume 采集集群中所要求的启动顺序，启动各节点上的采集方案。解决步骤如下：

首先回到报错的节点窗口，按“Ctrl+C”组合键，退出采集状态，然后在“complex01”目录下，执行以下命令，删除掉该目录的子目录“record”；

```
rm -rf record
```

成功删除“record”目录后，重新执行以下命令，启动采集方案即可。

```
flume-ng agent -c conf/ -f collect-logs.conf -n a1 -Dflume.root.looger=INFO,console
```

拓展练习

1. 在 Hadoop 集群中搭建并配置 Flume 集群。

2. 将素材文件 access.log、info.log 分别放置于节点 node1、节点 node2 中，编写 Flume 采集脚本文件，实现对 access.log、info.log 文件内容的采集，并将采集的数据汇总于节点 node3，归类存到 HDFS 中。