

尚硅谷大数据技术之 Flume

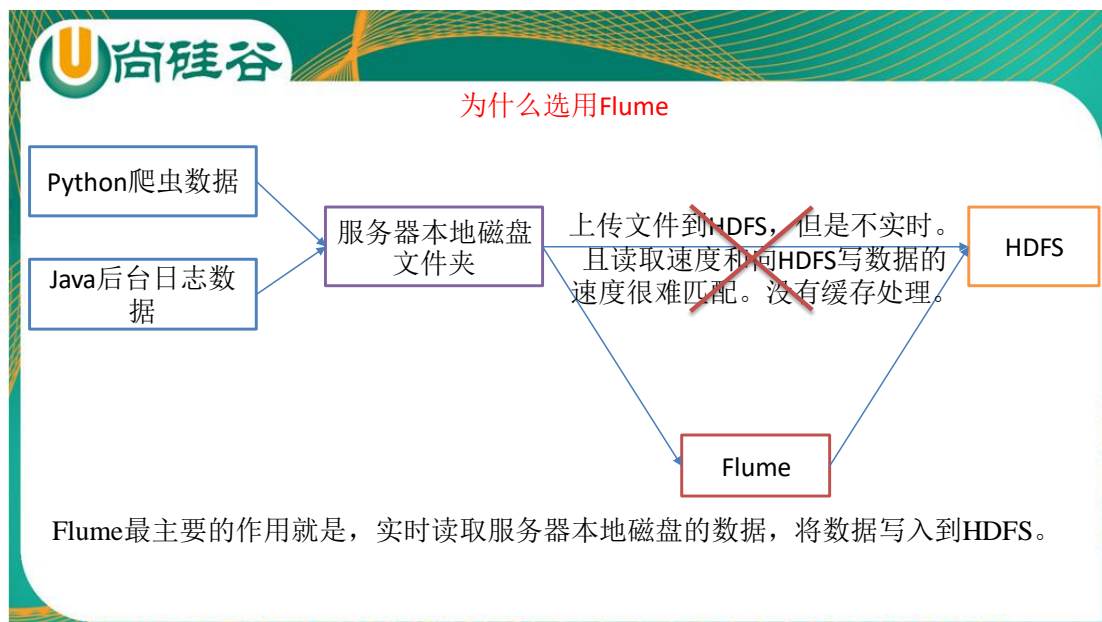
(作者：尚硅谷大数据研发部)

版本：V1.4

第 1 章 Flume 概述

1.1 Flume 概念

Flume 是 Cloudera 提供的一个高可用的，高可靠的，**分布式**的海量日志采集、聚合和传输的系统。Flume 基于流式架构，灵活简单。



1.2 Flume 组成架构

Flume 组成架构如图 1-1，图 1-2 所示：

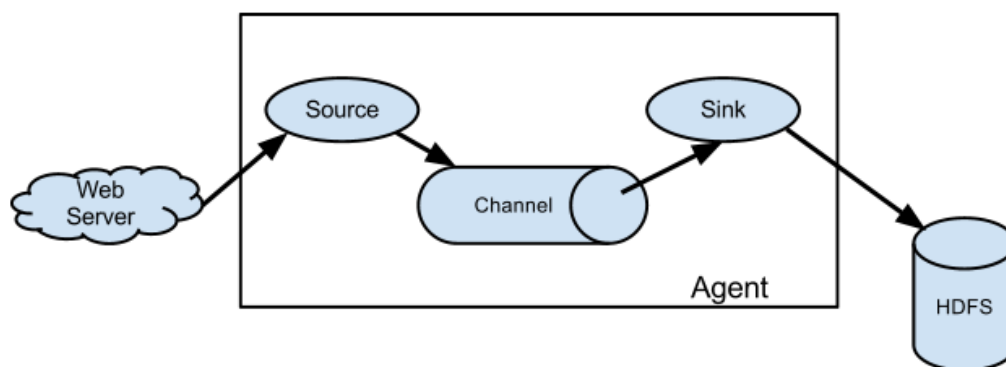


图 1-1 Flume 组成架构

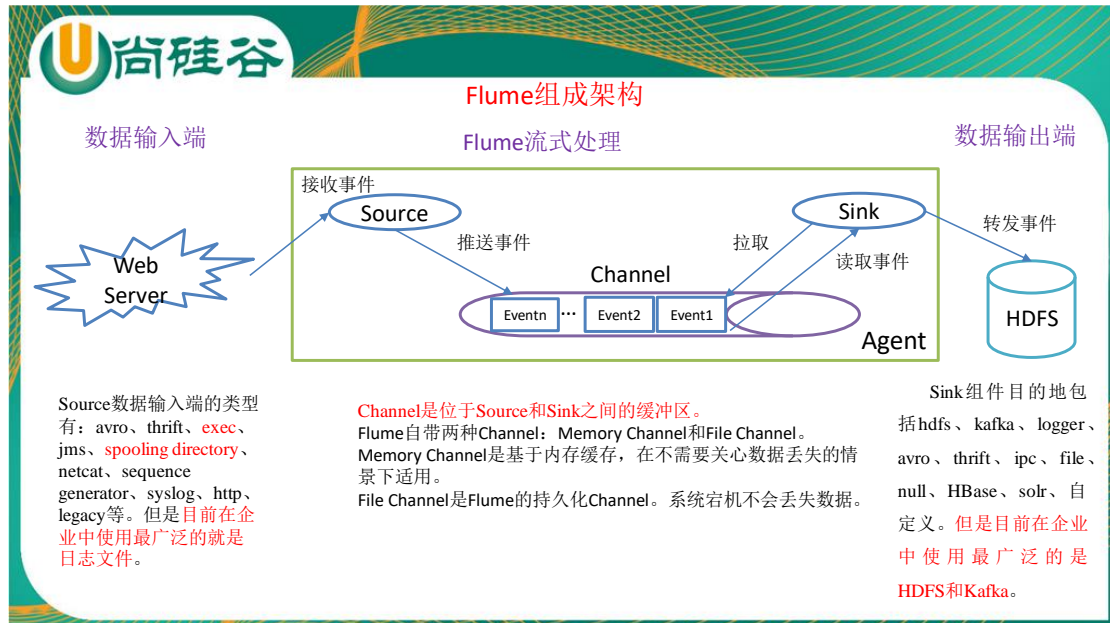


图 1-2 Flume 组成架构详解

下面我们来详细介绍一下 Flume 架构中的组件。

1.2.1 Agent

Agent 是一个 JVM 进程，它以事件的形式将数据从源头送至目的，是 Flume 数据传输的基本单元。

Agent 主要有 3 个部分组成，Source、Channel、Sink。

1.2.2 Source

Source 是负责接收数据到 Flume Agent 的组件。Source 组件可以处理各种类型、各种格式的日志数据，包括 avro、thrift、exec、jms、spooling directory、netcat、sequence generator、syslog、http、legacy。

1.2.3 Channel

Channel 是位于 Source 和 Sink 之间的缓冲区。因此，Channel 允许 Source 和 Sink 运作在不同的速率上。Channel 是线程安全的，可以同时处理几个 Source 的写入操作和几个 Sink 的读取操作。

Flume 自带两种 Channel：Memory Channel 和 File Channel。

Memory Channel 是内存中的队列。Memory Channel 在不需要关心数据丢失的情景下适用。如果需要关心数据丢失，那么 Memory Channel 就不应该使用，因为程序死亡、机器宕机或者重启都会导致数据丢失。

File Channel 将所有事件写到磁盘。因此在程序关闭或机器宕机的情况下不会丢失数据。

1.2.4 Sink

Sink 不断地轮询 Channel 中的事件且批量地移除它们，并将这些事件批量写入到存储或索引系统、或者被发送到另一个 Flume Agent。

Sink 是完全事务性的。在从 Channel 批量删除数据之前，每个 Sink 用 Channel 启动一个事务。批量事件一旦成功写出到存储系统或下一个 Flume Agent，Sink 就利用 Channel 提交事务。事务一旦被提交，该 Channel 从自己的内部缓冲区删除事件。

Sink 组件目的地包括 hdfs、logger、avro、thrift、ipc、file、null、HBase、solr、自定义。

1.2.5 Event

传输单元，Flume 数据传输的基本单元，以事件的形式将数据从源头送至目的地。

1.3 Flume 拓扑结构

Flume 的拓扑结构如图 1-3、1-4、1-5 和 1-6 所示：

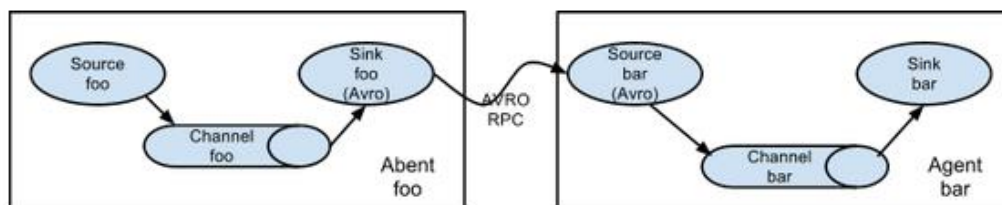


图 1-3 Flume Agent 连接

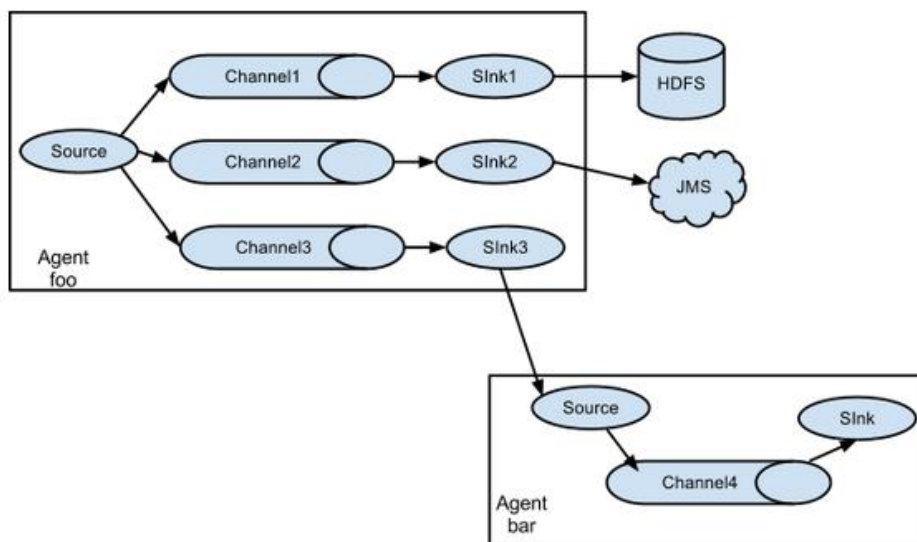


图 1-4 单 source，多 channel、sink

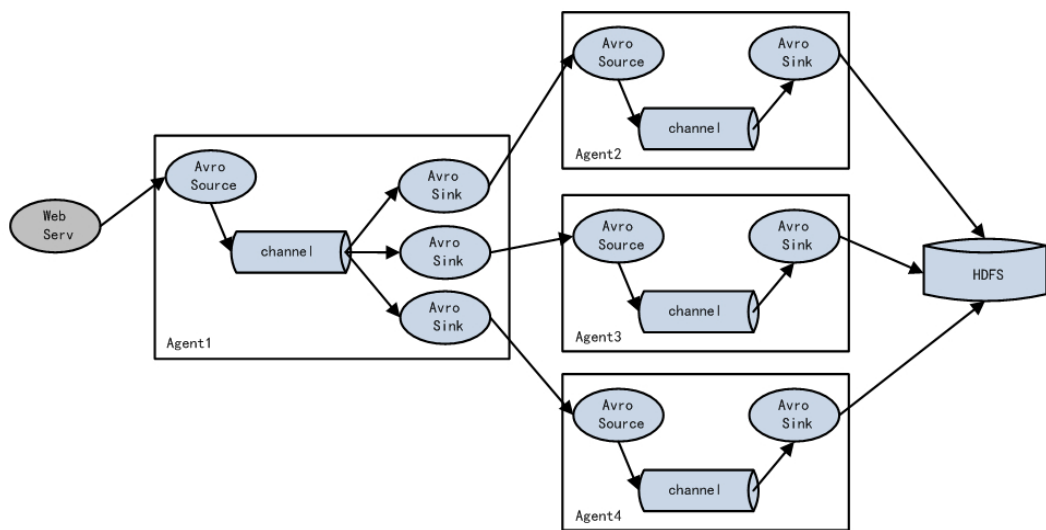


图 1-5 Flume 负载均衡

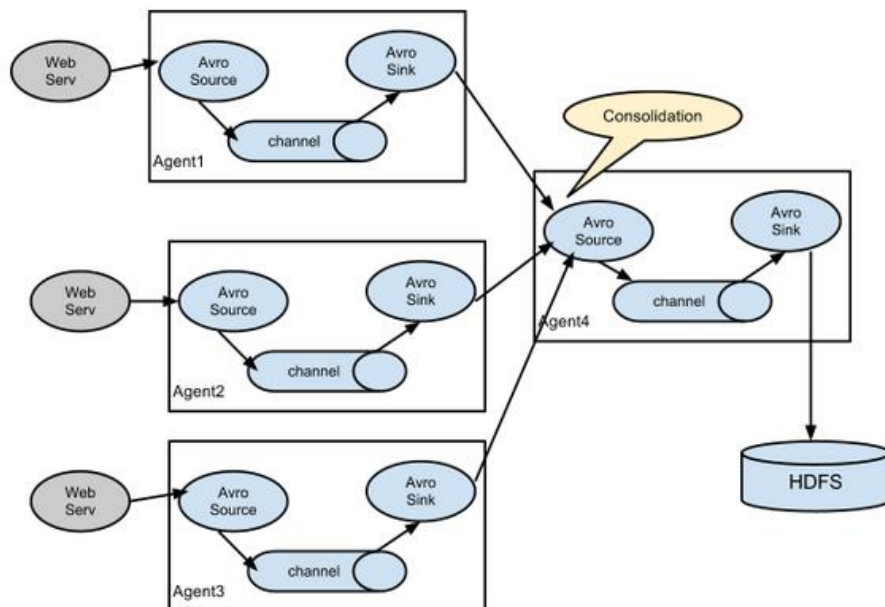
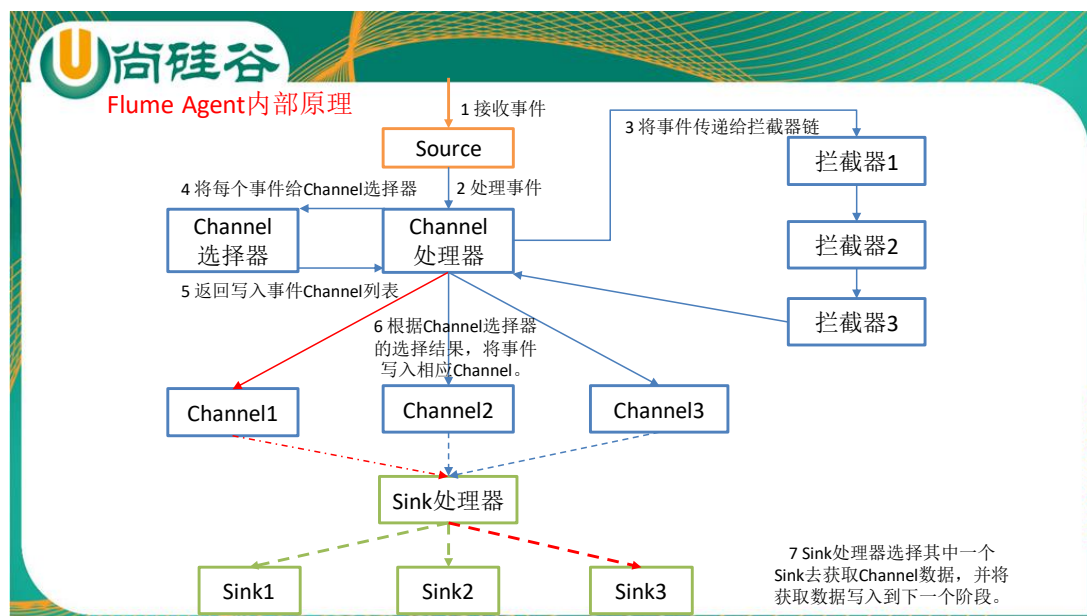


图 1-6 Flume Agent 聚合

1.4 Flume Agent 内部原理



第 2 章 快速入门

2.1 Flume 安装地址

1) Flume 官网地址

<http://flume.apache.org/>

2) 文档查看地址

<http://flume.apache.org/FlumeUserGuide.html>

3) 下载地址

<http://archive.apache.org/dist/flume/>

2.2 安装部署

1) 将 apache-flume-1.7.0-bin.tar.gz 上传到 linux 的 /opt/software 目录下

2) 解压 apache-flume-1.7.0-bin.tar.gz 到 /opt/module/ 目录下

```
[atguigu@hadoop102 software]$ tar -zxvf apache-flume-1.7.0-bin.tar.gz -C /opt/module/
```

3) 修改 apache-flume-1.7.0-bin 的名称为 flume

```
[atguigu@hadoop102 module]$ mv apache-flume-1.7.0-bin flume
```

4) 将 flume/conf 下的 flume-env.sh.template 文件修改为 flume-env.sh，并配置 flume-env.sh 文件


```
[atguigu@hadoop102 conf]$ mv flume-env.sh.template flume-env.sh
[atguigu@hadoop102 conf]$ vi flume-env.sh
export JAVA_HOME=/opt/module/jdk1.8.0_144
```

第3章 案例实操

3.1 监控端口数据官方案例

1) 案例需求: 首先, Flume 监控本机 44444 端口, 然后通过 telnet 工具向本机 44444 端口发送消息, 最后 Flume 将监听的数据实时显示在控制台。

2) 需求分析:



监听数据端口案例分析

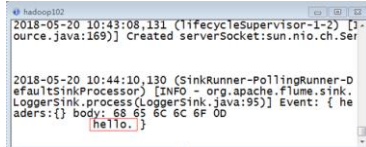
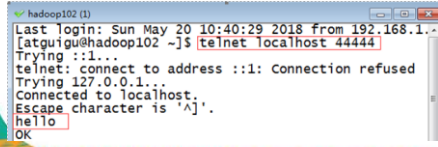
1 通过telnet工具向本机的44444端口发送数据

```
telnet localhost 44444
```

2 Flume监控本机的44444端口。通过Flume的source端读取数据。

```
bin/flume-ng agent --conf conf/ --name a1 --conf-file job/flume-telnet.conf -Dflume.root.logger==INFO,console
```

3 Flume将获取的数据通过Sink端写出到控制台



3) 实现步骤:

1. 安装 telnet 工具

在/opt/software 目录下创建 flume-telnet 文件夹。

```
[atguigu@hadoop102 software]$ mkdir flume-telnet
```

再将 rpm 软件包(、telnet-0.17-48.el6.x86_64.rpm 和 telnet-server-0.17-48.el6.x86_64.rpm)

拷入/opt/software/flume-telnet 文件夹下面。执行 RPM 软件包安装命令:

```
[atguigu@hadoop102 software]$ sudo rpm -ivh xinetd-2.3.14-40.el6.x86_64.rpm
[atguigu@hadoop102 software]$ sudo rpm -ivh telnet-0.17-48.el6.x86_64.rpm
[atguigu@hadoop102 software]$ sudo rpm -ivh telnet-server-0.17-48.el6.x86_64.rpm
```

2. 判断 44444 端口是否被占用

```
[atguigu@hadoop102 flume-telnet]$ sudo netstat -tunlp | grep 44444
```

功能描述: netstat 命令是一个监控 TCP/IP 网络的非常有用的工具, 它可以显示路由表、实际的网络连接以及每一个网络接口设备的状态信息。

基本语法: netstat [选项]

选项参数:

-t 或--tcp: 显示 TCP 传输协议的连线状况;

- u 或--udp: 显示 UDP 传输协议的连线状况;
- n 或--numeric: 直接使用 ip 地址, 而不通过域名服务器;
- l 或--listening: 显示监控中的服务器的 Socket;
- p 或--programs: 显示正在使用 Socket 的程序识别码和程序名称;

3. 创建 Flume Agent 配置文件 flume-telnet-logger.conf

在 flume 目录下创建 job 文件夹并进入 job 文件夹。

```
[atguigu@hadoop102 flume]$ mkdir job  
[atguigu@hadoop102 flume]$ cd job/
```

在 job 文件夹下创建 Flume Agent 配置文件 flume-telnet-logger.conf。

```
[atguigu@hadoop102 job]$ touch flume-telnet-logger.conf
```

在 flume-telnet-logger.conf 文件中添加如下内容。

```
[atguigu@hadoop102 job]$ vim flume-telnet-logger.conf
```

添加内容如下:

```
# Name the components on this agent  
a1.sources = r1  
a1.sinks = k1  
a1.channels = c1  
  
# Describe/configure the source  
a1.sources.r1.type = netcat  
a1.sources.r1.bind = localhost  
a1.sources.r1.port = 44444  
  
# Describe the sink  
a1.sinks.k1.type = logger  
  
# Use a channel which buffers events in memory  
a1.channels.c1.type = memory  
a1.channels.c1.capacity = 1000  
a1.channels.c1.transactionCapacity = 100  
  
# Bind the source and sink to the channel  
a1.sources.r1.channels = c1  
a1.sinks.k1.channel = c1
```

注: 配置文件来源于官方手册 <http://flume.apache.org/FlumeUserGuide.html>



配置文件解析

```

# Name the components on this agent
a1.sources = r1          r1:表示a1的输入源
a1.sinks = k1            k1:表示a1的输出目的地
a1.channels = c1         c1:表示a1的缓冲区

# Describe/configure the source
a1.sources.r1.type = netcat      表示a1的输入源类型为netcat端口类型
a1.sources.r1.bind = localhost   表示a1的监听的主机
a1.sources.r1.port = 44444       表示a1的监听的端口号

# Describe the sink
a1.sinks.k1.type = logger        表示a1的输出目的地是控制台logger类型

# Use a channel which buffers events in memory
a1.channels.c1.type = memory     表示a1的channel类型是memory内存型
a1.channels.c1.capacity = 1000   表示a1的channel总容量1000个event
a1.channels.c1.transactionCapacity = 100  表示a1的channel传输时收集到了100条event以后再去提交事务

# Bind the source and sink to the channel
a1.sources.r1.channels = c1      表示将r1和c1连接起来
a1.sinks.k1.channel = c1         表示将k1和c1连接起来

```

a1:表示agent的名称

4. 先开启 flume 监听端口

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

参数说明:

--conf conf/ : 表示配置文件存储在 conf/ 目录

--name a1 : 表示给 agent 起名为 a1

--conf-file job/flume-telnet.conf : flume 本次启动读取的配置文件是在 job 文件夹下的 flume-telnet.conf 文件。

-Dflume.root.logger==INFO,console : -D 表示 flume 运行时动态修改 flume.root.logger 参数属性值,并将控制台日志打印级别设置为 INFO 级别。日志级别包括:log、info、warn、error。

5. 使用 telnet 工具向本机的 44444 端口发送内容

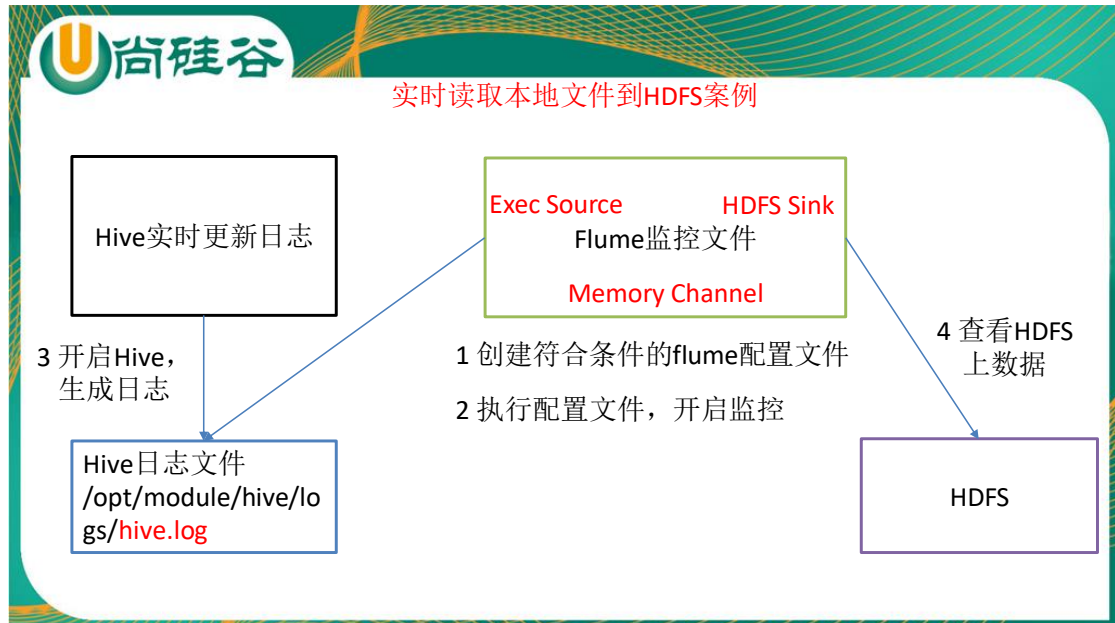
```
$ telnet localhost 44444
```

--如何关闭 telnet?

3.2 实时读取本地文件到 HDFS 案例

1) 案例需求: 实时监控 Hive 日志, 并上传到 HDFS 中

2) 需求分析:



3) 实现步骤:

1. Flume 要想将数据输出到 HDFS, 必须持有 Hadoop 相关 jar 包

将 commons-configuration-1.6.jar、

hadoop-auth-2.7.2.jar、

hadoop-common-2.7.2.jar、

hadoop-hdfs-2.7.2.jar、

commons-io-2.4.jar、

htrace-core-3.1.0-incubating.jar 拷贝到/opt/module/flume/lib 文件夹下。

提示: 标红的 jar 为 1.99 版本 flume 必须引用的 jar。其他版本可以不引用。

2. 创建 flume-file-hdfs.conf 文件

创建文件

```
[atguigu@hadoop102 job]$ touch flume-file-hdfs.conf
```

注: 要想读取 Linux 系统中的文件, 就得按照 Linux 命令的规则执行命令。由于 hive 日志在 Linux 系统中所以读取文件的类型选择: exec 即 execute 执行的意思。表示执行 Linux 命令来读取文件。

```
[atguigu@hadoop102 job]$ vim flume-file-hdfs.conf
```

添加如下内容

```
# Name the components on this agent
a2.sources = r2
a2.sinks = k2
a2.channels = c2


# Describe/configure the source
a2.sources.r2.type = exec
a2.sources.r2.command = tail -F /opt/module/hive/logs/hive.log
a2.sources.r2.shell = /bin/bash -c

# Describe the sink
a2.sinks.k2.type = hdfs
```

```
a2.sinks.k2.hdfs.path = hdfs://hadoop102:9000/flume/%Y%m%d/%H
#上传文件的前缀
a2.sinks.k2.hdfs.filePrefix = logs-
#是否按照时间滚动文件夹
a2.sinks.k2.hdfs.round = true
#多少时间单位创建一个新的文件夹
a2.sinks.k2.hdfs.roundValue = 1
#重新定义时间单位
a2.sinks.k2.hdfs.roundUnit = hour
#是否使用本地时间戳
a2.sinks.k2.hdfs.useLocalTimeStamp = true
#积攒多少个Event才flush到HDFS一次
a2.sinks.k2.hdfs.batchSize = 1000
#设置文件类型，可支持压缩
a2.sinks.k2.hdfs.fileType = DataStream
#多久生成一个新的文件
a2.sinks.k2.hdfs.rollInterval = 600
#设置每个文件的滚动大小
a2.sinks.k2.hdfs.rollSize = 134217700
#文件的滚动与Event数量无关
a2.sinks.k2.hdfs.rollCount = 0
#最小冗余数
a2.sinks.k2.hdfs.minBlockReplicas = 1

# Use a channel which buffers events in memory
a2.channels.c2.type = memory
a2.channels.c2.capacity = 1000
a2.channels.c2.transactionCapacity = 100

# Bind the source and sink to the channel
a2.sources.r2.channels = c2
a2.sinks.k2.channel = c2
```



实时读取本地文件到HDFS案例

```
# Name the components on this agent
a2.sources = r2          #定义source
a2.sinks = k2            #定义sink
a2.channels = c2         #定义channel
# Describe/configure the source
a2.sources.r2.type = exec #定义source类型为exec可执行命令的
a2.sources.r2.command = tail -F /opt/module/hive/logs/hive.log
a2.sources.r2.shell = /bin/bash -c #执行shell脚本的绝对路径

# Describe the sink
a2.sinks.k2.type = hdfs
a2.sinks.k2.hdfs.path = hdfs://hadoop102:9000/flume/%Y%m%d/%H
a2.sinks.k2.hdfs.filePrefix = logs- #上传文件的前缀
a2.sinks.k2.hdfs.round = true       #是否按照时间滚动文件夹
a2.sinks.k2.hdfs.roundValue = 1     #多少时间单位创建一个新的文件夹
a2.sinks.k2.hdfs.roundUnit = hour   #重新定义时间单位
a2.sinks.k2.hdfs.useLocalTimeStamp = true #是否使用本地时间戳
a2.sinks.k2.hdfs.batchSize = 1000    #积攒多少个Event才flush到HDFS一次
a2.sinks.k2.hdfs.fileType = DataStream #设置文件类型，可支持压缩
a2.sinks.k2.hdfs.rollInterval = 600  #多久生成一个新的文件
a2.sinks.k2.hdfs.rollSize = 134217700 #设置每个文件的滚动大小
a2.sinks.k2.hdfs.rollCount = 0       #文件的滚动与Event数量无关
a2.sinks.k2.hdfs.minBlockReplicas = 1 #最小冗余数

# Use a channel which buffers events in memory
a2.channels.c2.type = memory
a2.channels.c2.capacity = 1000
a2.channels.c2.transactionCapacity = 100

# Bind the source and sink to the channel
a2.sources.r2.channels = c2
a2.sinks.k2.channel = c2
```

3. 执行监控配置

```
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name a2 --conf-file job/flume-file-hdfs.conf
```

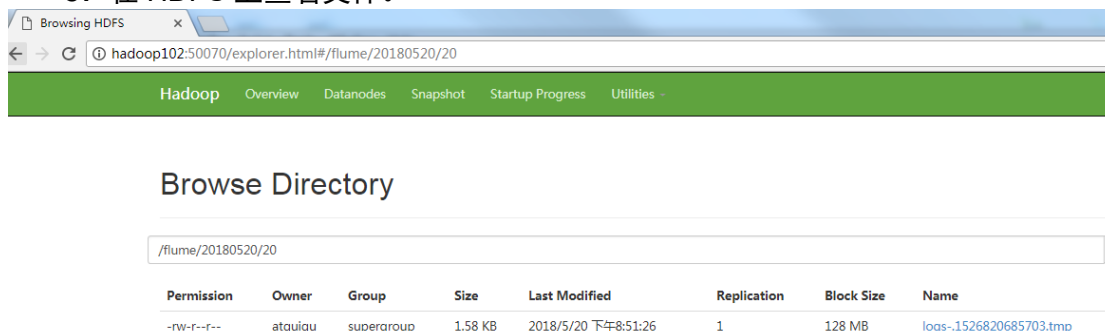
4. 开启 hadoop 和 hive 并操作 hive 产生日志

```
[atguigu@hadoop102 hadoop-2.7.2]$ sbin/start-dfs.sh
```

```
[atguigu@hadoop103 ~]$ sbin/start-yarn.sh
```

```
[atguigu@hadoop102 ~]$ bin/hive
hive (default)>
```

5. 在 HDFS 上查看文件。



Browse Directory

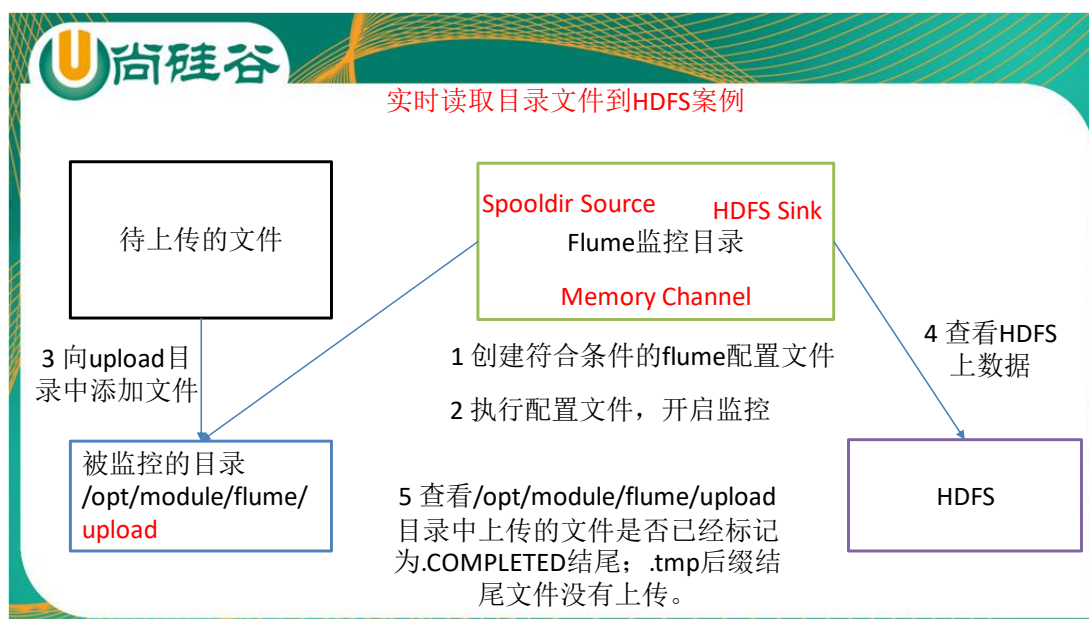
/flume/20180520/20

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	atguigu	supergroup	1.58 KB	2018/5/20 下午8:51:26	1	128 MB	logs-1526820685703.tmp

3.3 实时读取目录文件到 HDFS 案例

1) 案例需求：使用 flume 监听整个目录的文件

2) 需求分析：



3) 实现步骤：

1. 创建配置文件 flume-dir-hdfs.conf

创建一个文件

```
[atguigu@hadoop102 ~]$ touch flume-dir-hdfs.conf
```

打开文件

```
[atguigu@hadoop102 ~]$ vim flume-dir-hdfs.conf
```

添加如下内容


```
a3.sources = r3
a3.sinks = k3
a3.channels = c3
```

```
# Describe/configure the source
a3.sources.r3.type = spooldir
a3.sources.r3.spoolDir = /opt/module/flume/upload
a3.sources.r3.fileSuffix = .COMPLETED
a3.sources.r3.fileHeader = true
#忽略所有以.tmp 结尾的文件，不上传
a3.sources.r3.ignorePattern = ([^ ]*\tmp)

# Describe the sink
a3.sinks.k3.type = hdfs
a3.sinks.k3.hdfs.path = hdfs://hadoop102:9000/flume/upload/%Y%m%d/%H
#上传文件的前缀
a3.sinks.k3.hdfs.filePrefix = upload-
#是否按照时间滚动文件夹
a3.sinks.k3.hdfs.round = true
#多少时间单位创建一个新的文件夹
a3.sinks.k3.hdfs.roundValue = 1
#重新定义时间单位
a3.sinks.k3.hdfs.roundUnit = hour
#是否使用本地时间戳
a3.sinks.k3.hdfs.useLocalTimeStamp = true
#积攒多少个 Event 才 flush 到 HDFS 一次
a3.sinks.k3.hdfs.batchSize = 100
#设置文件类型，可支持压缩
a3.sinks.k3.hdfs.fileType = DataStream
#多久生成一个新的文件
a3.sinks.k3.hdfs.rollInterval = 600
#设置每个文件的滚动大小大概是 128M
a3.sinks.k3.hdfs.rollSize = 134217700
#文件的滚动与 Event 数量无关
a3.sinks.k3.hdfs.rollCount = 0
#最小冗余数
a3.sinks.k3.hdfs.minBlockReplicas = 1

# Use a channel which buffers events in memory
a3.channels.c3.type = memory
a3.channels.c3.capacity = 1000
a3.channels.c3.transactionCapacity = 100

# Bind the source and sink to the channel
a3.sources.r3.channels = c3
a3.sinks.k3.channel = c3
```



实时读取目录文件到HDFS案例

```

a3.sources = r3           #定义source
a3.sinks = k3             #定义sink
a3.channels = c3          #定义channel

# Describe/configure the source
a3.sources.r3.type = spooldir #定义source类型为目录
a3.sources.r3.spoolDir = /opt/module/flume/upload #定义监控目录
a3.sources.r3.fileSuffix = .COMPLETED #定义文件上传完, 后缀
a3.sources.r3.fileHeader = true #是否有文件头
a3.sources.r3.ignorePattern = ([^]*\\.tmp) #忽略所有以 .tmp 结尾的文件, 不上传

# Describe the sink
a3.sinks.k3.type = hdfs #sink类型为hdfs
a3.sinks.k3.hdfs.path = hdfs://hadoop102:9000/flume/upload/%Y%m%d/%H #文件上传到hdfs的路径
a3.sinks.k3.hdfs.filePrefix = upload- #上传文件到hdfs的前缀
a3.sinks.k3.hdfs.round = true #是否按时间滚动文件
a3.sinks.k3.hdfs.roundValue = 1 #多少时间单位创建一个新的文件夹
a3.sinks.k3.hdfs.roundUnit = hour #重新定义时间单位
a3.sinks.k3.hdfs.useLocalTimeStamp = true #是否使用本地时间戳
a3.sinks.k3.hdfs.batchSize = 100 #积攒多少个Event才flush到HDFS一次
a3.sinks.k3.hdfs.fileType = DataStream #设置文件类型, 可支持压缩

a3.sinks.k3.hdfs.rollInterval = 600 #多久生成新文件
a3.sinks.k3.hdfs.rollSize = 134217700 #多大生成新文件
a3.sinks.k3.hdfs.rollCount = 0 #多少event生成新文件
a3.sinks.k3.hdfs.minBlockReplicas = 1 #多少副本数

# Use a channel which buffers events in memory
a3.channels.c3.type = memory
a3.channels.c3.capacity = 1000
a3.channels.c3.transactionCapacity = 100

# Bind the source and sink to the channel
a3.sources.r3.channels = c3
a3.sinks.k3.channel = c3
                    
```

2. 启动监控文件夹命令

```
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name a3 --conf-file job/flume-dir-hdfs.conf
```

说明： 在使用 Spooling Directory Source 时

- 1) 不要在监控目录中创建并持续修改文件
- 2) 上传完成的文件会以.COMPLETED 结尾
- 3) 被监控文件夹每 600 毫秒扫描一次文件变动

3. 向 upload 文件夹中添加文件

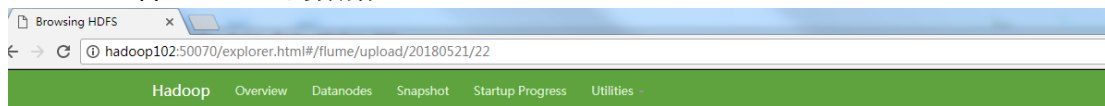
在/opt/module/flume 目录下创建 upload 文件夹

```
[atguigu@hadoop102 flume]$ mkdir upload
```

向 upload 文件夹中添加文件

```
[atguigu@hadoop102 upload]$ touch atguigu.txt
[atguigu@hadoop102 upload]$ touch atguigu.tmp
[atguigu@hadoop102 upload]$ touch atguigu.log
```

4. 查看 HDFS 上的数据



Browse Directory

/flume/upload/20180521/22								Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
-rw-r--r--	atguigu	supergroup	1 B	2018/5/21 下午10:29:43	3	128 MB	upload-1526912982563.tmp	

5. 等待 1s, 再次查询 upload 文件夹

```
[atguigu@hadoop102 upload]$ ll
总用量 0
-rw-rw-r--. 1 atguigu atguigu 0 5月 20 22:31 atguigu.log.COMPLETED
```

```
-rw-rw-r--. 1 atguigu atguigu 0 5月 20 22:31 atguigu.tmp
-rw-rw-r--. 1 atguigu atguigu 0 5月 20 22:31 atguigu.txt.COMPLETED
```

3.4 单数据源多出口案例(一)

单 Source 多 Channel、Sink 如图 7-2 所示

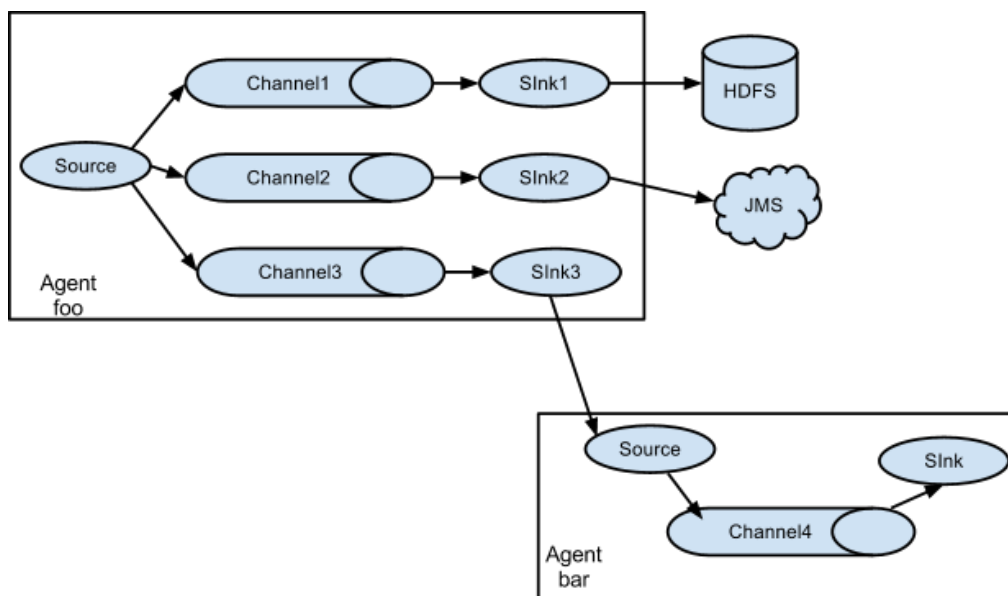
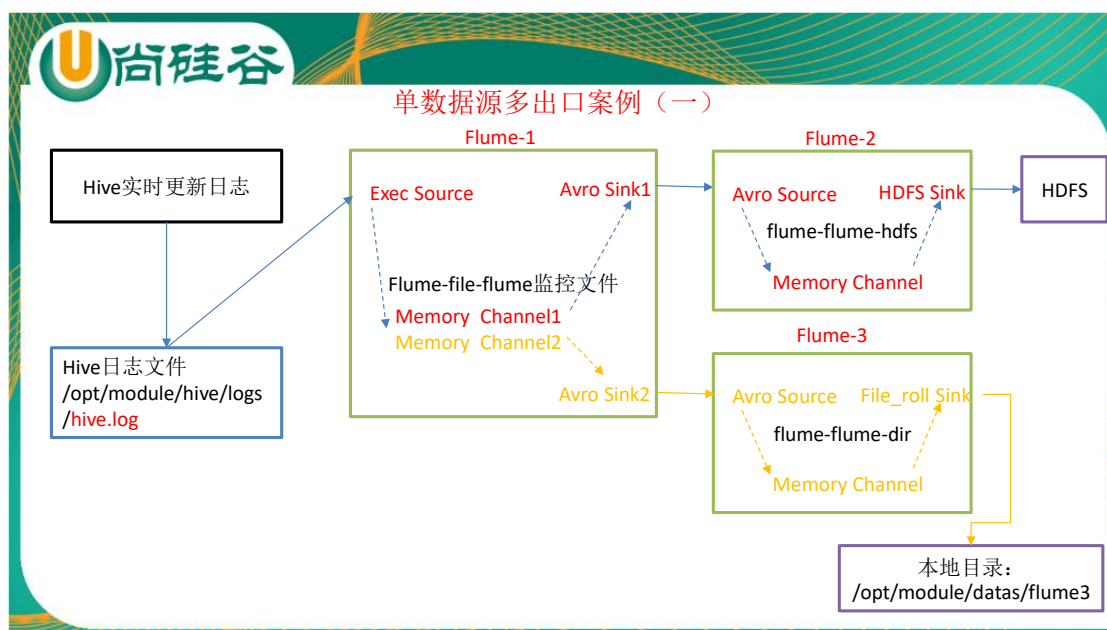


图 7-2 单 Source 多 Channel、Sink

- 1) 案例需求：使用 flume-1 监控文件变动，flume-1 将变动内容传递给 flume-2，flume-2 负责存储到 HDFS。同时 flume-1 将变动内容传递给 flume-3，flume-3 负责输出到 local filesystem。
- 2) 需求分析：



- 3) 实现步骤：

0. 准备工作

在/opt/module/flume/job 目录下创建 group1 文件夹

```
[atguigu@hadoop102 job]$ cd group1/
```

在/opt/module/datas/目录下创建 flume3 文件夹

```
[atguigu@hadoop102 datas]$ mkdir flume3
```

1. 创建 flume-file-flume.conf

配置 1 个接收日志文件的 source 和两个 channel、两个 sink，分别输送给 flume-flume-hdfs 和 flume-flume-dir。

创建配置文件并打开

```
[atguigu@hadoop102 group1]$ touch flume-file-flume.conf  
[atguigu@hadoop102 group1]$ vim flume-file-flume.conf
```

添加如下内容

```
# Name the components on this agent  
a1.sources = r1  
a1.sinks = k1 k2  
a1.channels = c1 c2  
# 将数据流复制给多个 channel  
a1.sources.r1.selector.type = replicating  
  
# Describe/configure the source  
a1.sources.r1.type = exec  
a1.sources.r1.command = tail -F /opt/module/hive/logs/hive.log  
a1.sources.r1.shell = /bin/bash -c  
  
# Describe the sink  
a1.sinks.k1.type = avro  
a1.sinks.k1.hostname = hadoop102  
a1.sinks.k1.port = 4141  
  
a1.sinks.k2.type = avro  
a1.sinks.k2.hostname = hadoop102  
a1.sinks.k2.port = 4142  
  
# Describe the channel  
a1.channels.c1.type = memory  
a1.channels.c1.capacity = 1000  
a1.channels.c1.transactionCapacity = 100  
  
a1.channels.c2.type = memory  
a1.channels.c2.capacity = 1000  
a1.channels.c2.transactionCapacity = 100  
  
# Bind the source and sink to the channel  
a1.sources.r1.channels = c1 c2  
a1.sinks.k1.channel = c1  
a1.sinks.k2.channel = c2
```

注：Avro 是由 Hadoop 创始人 Doug Cutting 创建的一种语言无关的数据序列化和 RPC 框架。

注：RPC（Remote Procedure Call）—远程过程调用，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：[尚硅谷官网](#)

2. 创建 flume-flume-hdfs.conf

配置上级 flume 输出的 source，输出是到 hdfs 的 sink。

创建配置文件并打开

```
[atguigu@hadoop102 group1]$ touch flume-flume-hdfs.conf
[atguigu@hadoop102 group1]$ vim flume-flume-hdfs.conf
```

添加如下内容

```
# Name the components on this agent
a2.sources = r1
a2.sinks = k1
a2.channels = c1

# Describe/configure the source
a2.sources.r1.type = avro
a2.sources.r1.bind = hadoop102
a2.sources.r1.port = 4141

# Describe the sink
a2.sinks.k1.type = hdfs
a2.sinks.k1.hdfs.path = hdfs://hadoop102:9000/flume2/%Y%m%d/%H
#上传文件的前缀
a2.sinks.k1.hdfs.filePrefix = flume2-
#是否按照时间滚动文件夹
a2.sinks.k1.hdfs.round = true
#多少时间单位创建一个新的文件夹
a2.sinks.k1.hdfs.roundValue = 1
#重新定义时间单位
a2.sinks.k1.hdfs.roundUnit = hour
#是否使用本地时间戳
a2.sinks.k1.hdfs.useLocalTimeStamp = true
#积攒多少个 Event 才 flush 到 HDFS 一次
a2.sinks.k1.hdfs.batchSize = 100
#设置文件类型，可支持压缩
a2.sinks.k1.hdfs.fileType = DataStream
#多久生成一个新的文件
a2.sinks.k1.hdfs.rollInterval = 600
#设置每个文件的滚动大小大概是 128M
a2.sinks.k1.hdfs.rollSize = 134217700
#文件的滚动与 Event 数量无关
a2.sinks.k1.hdfs.rollCount = 0
#最小冗余数
a2.sinks.k1.hdfs.minBlockReplicas = 1

# Describe the channel
a2.channels.c1.type = memory
a2.channels.c1.capacity = 1000
a2.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a2.sources.r1.channels = c1
a2.sinks.k1.channel = c1
```

3. 创建 flume-flume-dir.conf

配置上级 flume 输出的 source，输出是到本地目录的 sink。

创建配置文件并打开

```
[atguigu@hadoop102 group1]$ touch flume-flume-dir.conf
[atguigu@hadoop102 group1]$ vim flume-flume-dir.conf
```

添加如下内容

```
# Name the components on this agent
a3.sources = r1
a3.sinks = k1
a3.channels = c2

# Describe/configure the source
a3.sources.r1.type = avro
a3.sources.r1.bind = hadoop102
a3.sources.r1.port = 4142

# Describe the sink
a3.sinks.k1.type = file_roll
a3.sinks.k1.sink.directory = /opt/module/datas/flume3

# Describe the channel
a3.channels.c2.type = memory
a3.channels.c2.capacity = 1000
a3.channels.c2.transactionCapacity = 100

# Bind the source and sink to the channel
a3.sources.r1.channels = c2
a3.sinks.k1.channel = c2
```

提示：输出的本地目录必须是已经存在的目录，如果该目录不存在，并不会创建新的目录。

4. 执行配置文件

分别开启对应配置文件：flume-flume-dir, flume-flume-hdfs, flume-file-flume。

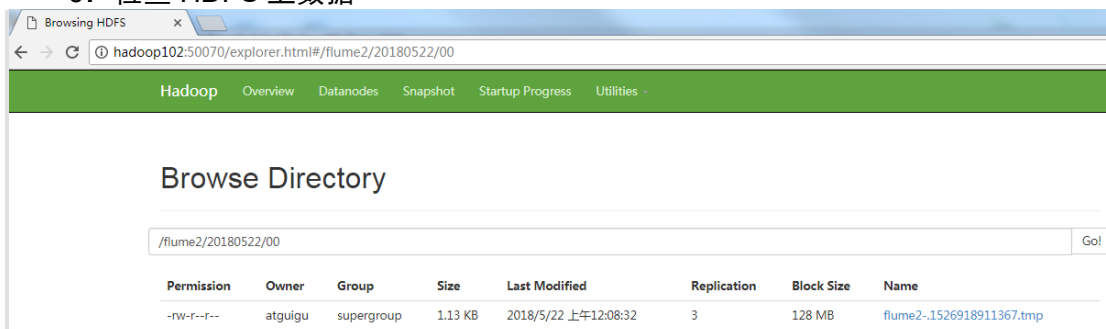
```
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name a3 --conf-file
job/group1/flume-flume-dir.conf
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name a2 --conf-file
job/group1/flume-flume-hdfs.conf
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name a1 --conf-file
job/group1/flume-file-flume.conf
```

5. 启动 hadoop 和 hive

```
[atguigu@hadoop102 hadoop-2.7.2]$ sbin/start-dfs.sh
[atguigu@hadoop103 hadoop-2.7.2]$ sbin/start-yarn.sh

[atguigu@hadoop102 hive]$ bin/hive
hive (default)>
```

6. 检查 HDFS 上数据



The screenshot shows the Hadoop HDFS Explorer interface. The breadcrumb path is /flume2/20180522/00. A table lists the contents of the directory:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	atguigu	supergroup	1.13 KB	2018/5/22 上午12:08:32	3	128 MB	flume2-1526918911367.tmp

7 检查/opt/module/datas/flume3 目录中数据

```
[atguigu@hadoop102 flume3]$ ll
总用量 8
-rw-rw-r--. 1 atguigu atguigu 5942 5月 22 00:09 1526918887550-3
```

3.5 单数据源多出口案例(二)

单 Source、Channel 多 Sink(负载均衡)如图 7-3 所示

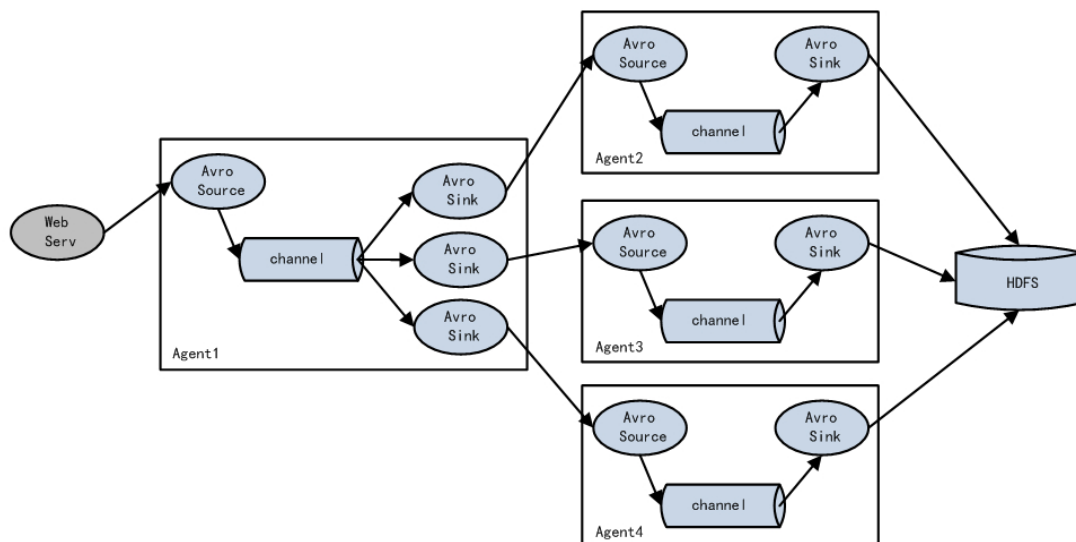
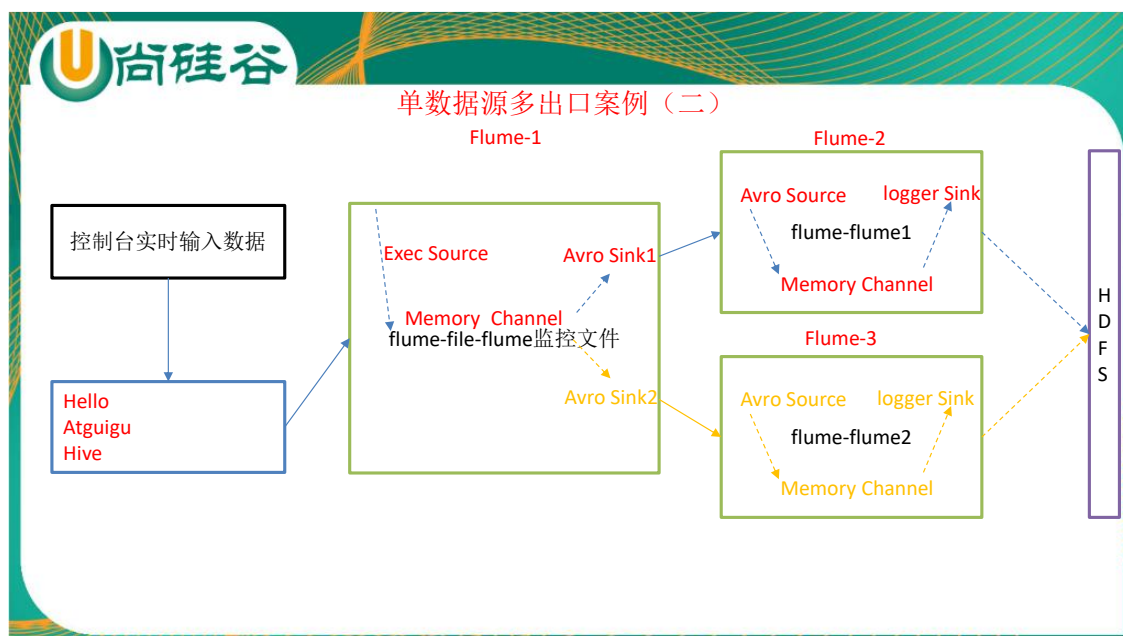


图 7-3 单 Source、Channel 多 Sink

- 1) 案例需求：使用 flume-1 监控文件变动，flume-1 将变动内容传递给 flume-2，flume-2 负责存储到 HDFS。同时 flume-1 将变动内容传递给 flume-3，flume-3 也负责存储到 HDFS
- 2) 需求分析：



- 3) 实现步骤：

0. 准备工作

在 /opt/module/flume/job 目录下创建 group2 文件夹

```
[atguigu@hadoop102 job]$ cd group1/
```

1. 创建 flume-netcat-flume.conf

配置 1 个接收日志文件的 source 和 1 个 channel、两个 sink，分别输送给 flume-flume1 和 flume-flume2。

创建配置文件并打开

```
[atguigu@hadoop102 group1]$ touch flume-netcat-flume.conf  
[atguigu@hadoop102 group1]$ vim flume-netcat-flume.conf
```

添加如下内容

```
# Name the components on this agent  
a1.sources = r1  
a1.channels = c1  
a1.sinkgroups = g1  
a1.sinks = k1 k2  
  
# Describe/configure the source  
a1.sources.r1.type = netcat  
a1.sources.r1.bind = localhost  
a1.sources.r1.port = 44444  
  
a1.sinkgroups.g1.processor.type = load_balance  
a1.sinkgroups.g1.processor.backoff = true  
a1.sinkgroups.g1.processor.selector = round_robin  
a1.sinkgroups.g1.processor.selector.maxTimeOut=10000  
  
# Describe the sink  
a1.sinks.k1.type = avro  
a1.sinks.k1.hostname = hadoop102  
a1.sinks.k1.port = 4141  
  
a1.sinks.k2.type = avro  
a1.sinks.k2.hostname = hadoop102  
a1.sinks.k2.port = 4142  
  
# Describe the channel  
a1.channels.c1.type = memory  
a1.channels.c1.capacity = 1000  
a1.channels.c1.transactionCapacity = 100  
  
# Bind the source and sink to the channel  
a1.sources.r1.channels = c1  
a1.sinkgroups.g1.sinks = k1 k2  
a1.sinks.k1.channel = c1  
a1.sinks.k2.channel = c1
```

注：Avro 是由 Hadoop 创始人 Doug Cutting 创建的一种语言无关的数据序列化和 RPC 框架。

注：RPC（Remote Procedure Call）—远程过程调用，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。

2. 创建 flume-flume1.conf

配置上级 flume 输出的 source，输出是到本地控制台。

创建配置文件并打开

```
[atguigu@hadoop102 group1]$ touch flume-flume1.conf  
[atguigu@hadoop102 group1]$ vim flume-flume1.conf
```

添加如下内容

```
# Name the components on this agent  
a2.sources = r1  
a2.sinks = k1  
a2.channels = c1  
  
# Describe/configure the source  
a2.sources.r1.type = avro  
a2.sources.r1.bind = hadoop102  
a2.sources.r1.port = 4141  
  
# Describe the sink  
a2.sinks.k1.type = logger  
  
# Describe the channel  
a2.channels.c1.type = memory  
a2.channels.c1.capacity = 1000  
a2.channels.c1.transactionCapacity = 100  
  
# Bind the source and sink to the channel  
a2.sources.r1.channels = c1  
a2.sinks.k1.channel = c1
```

3. 创建 flume-flume2.conf

配置上级 flume 输出的 source，输出是到本地控制台。

创建配置文件并打开

```
[atguigu@hadoop102 group1]$ touch flume-flume2.conf  
[atguigu@hadoop102 group1]$ vim flume-flume2.conf
```

添加如下内容

```
# Name the components on this agent  
a3.sources = r1  
a3.sinks = k1  
a3.channels = c2  
  
# Describe/configure the source  
a3.sources.r1.type = avro  
a3.sources.r1.bind = hadoop102  
a3.sources.r1.port = 4142  
  
# Describe the sink  
a3.sinks.k1.type = logger  
  
# Describe the channel  
a3.channels.c2.type = memory  
a3.channels.c2.capacity = 1000  
a3.channels.c2.transactionCapacity = 100  
  
# Bind the source and sink to the channel  
a3.sources.r1.channels = c2  
a3.sinks.k1.channel = c2
```

4. 执行配置文件

分别开启对应配置文件：flume-flume2，flume-flume1，flume-netcat-flume。

```
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name a3 --conf-file  
job/group1/flume-flume2.conf -Dflume.root.logger=INFO,console
```

```
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name a2 --conf-file job/group1/flume-flume1.conf -Dflume.root.logger=INFO,console
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name a1 --conf-file job/group1/flume-netcat-flume.conf
```

5. 使用 telnet 工具向本机的 44444 端口发送内容

```
$ telnet localhost 44444
```

6. 查看 flume2 及 flume3 的控制台打印日志

3.6 多数据源汇总案例

多 Source 汇总数据到单 Flume 如图 7-4 所示

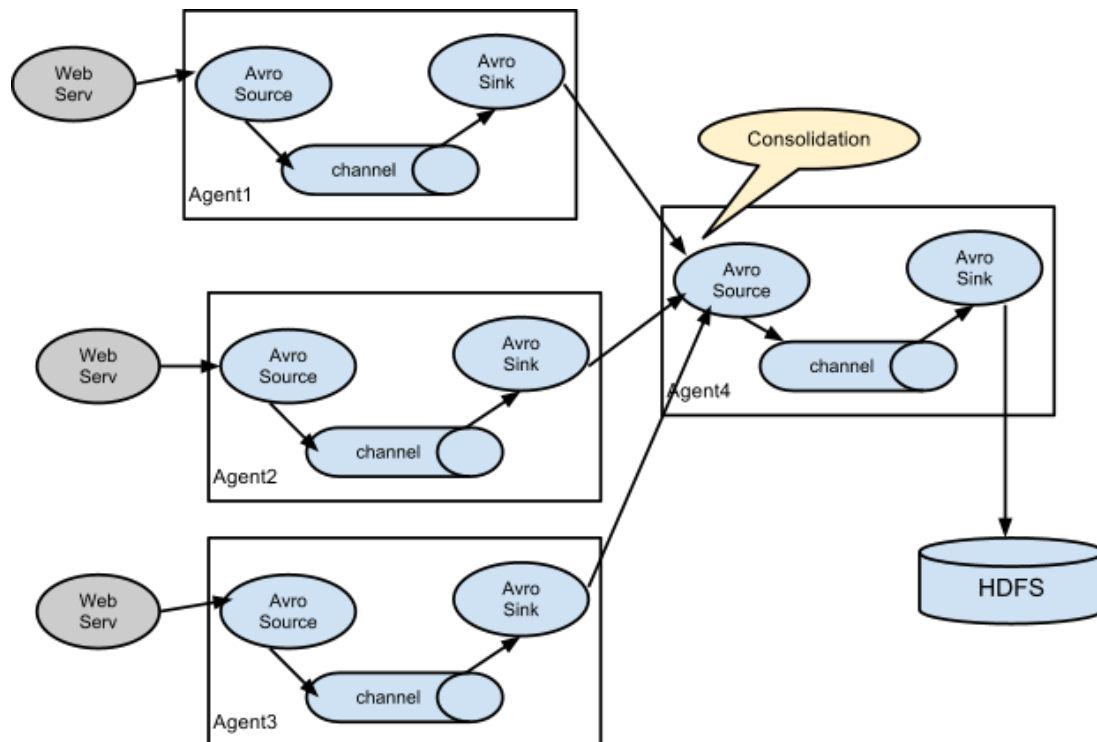


图 7-4 多 Flume 汇总数据到单 Flume

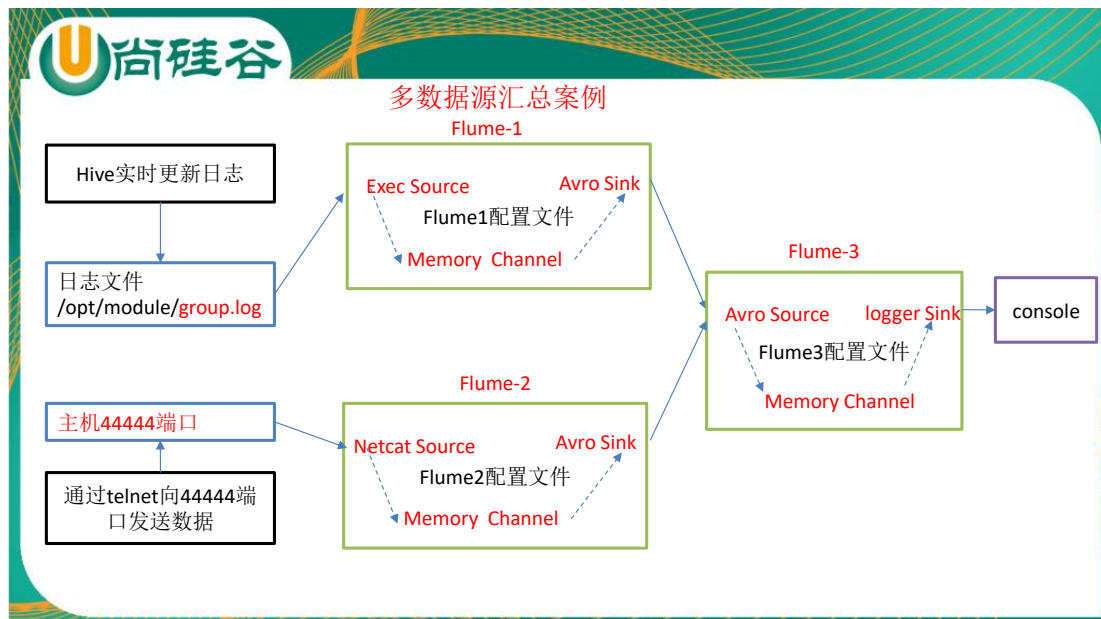
1) 案例需求:

hadoop103 上的 flume-1 监控文件 hive.log,

hadoop104 上的 flume-2 监控某一个端口的数据流,

flume-1 与 flume-2 将数据发送给 hadoop102 上的 flume-3, flume-3 将最终数据打印到控制台

2) 需求分析:



3) 实现步骤:

0. 准备工作

分发 flume

```
[atguigu@hadoop102 module]$ xsync flume
```

在 hadoop102、hadoop103 以及 hadoop104 的 /opt/module/flume/job 目录下创建一个 group2 文件夹

```
[atguigu@hadoop102 job]$ mkdir group2
[atguigu@hadoop103 job]$ mkdir group2
[atguigu@hadoop104 job]$ mkdir group2
```

1. 创建 flume1.conf

配置 source 用于监控 hive.log 文件，配置 sink 输出数据到下一级 flume。

在 hadoop103 上创建配置文件并打开

```
[atguigu@hadoop103 group2]$ touch flume1.conf
[atguigu@hadoop103 group2]$ vim flume1.conf
```

添加如下内容

```
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# Describe/configure the source
a1.sources.r1.type = exec
a1.sources.r1.command = tail -F /opt/module/group.log
a1.sources.r1.shell = /bin/bash -c

# Describe the sink
a1.sinks.k1.type = avro
a1.sinks.k1.hostname = hadoop102
a1.sinks.k1.port = 4141

# Describe the channel
```



```
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

2. 创建 flume2.conf

配置 source 监听端口 44444 数据流，配置 sink 数据到下一级 flume:

在 hadoop104 上创建配置文件并打开

```
[atguigu@hadoop104 group2]$ touch flume2.conf
[atguigu@hadoop104 group2]$ vim flume2.conf
```

添加如下内容

```
# Name the components on this agent
a2.sources = r1
a2.sinks = k1
a2.channels = c1

# Describe/configure the source
a2.sources.r1.type = netcat
a2.sources.r1.bind = hadoop104
a2.sources.r1.port = 44444

# Describe the sink
a2.sinks.k1.type = avro
a2.sinks.k1.hostname = hadoop102
a2.sinks.k1.port = 4141

# Use a channel which buffers events in memory
a2.channels.c1.type = memory
a2.channels.c1.capacity = 1000
a2.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a2.sources.r1.channels = c1
a2.sinks.k1.channel = c1
```

3. 创建 flume3.conf

配置 source 用于接收 flume1 与 flume2 发送过来的数据流，最终合并后 sink 到控制台。

在 hadoop102 上创建配置文件并打开

```
[atguigu@hadoop102 group2]$ touch flume3.conf
[atguigu@hadoop102 group2]$ vim flume3.conf
```

添加如下内容

```
# Name the components on this agent
a3.sources = r1
a3.sinks = k1
a3.channels = c1

# Describe/configure the source
a3.sources.r1.type = avro
a3.sources.r1.bind = hadoop102
a3.sources.r1.port = 4141

# Describe the sink
```

```
# Describe the sink
a3.sinks.k1.type = logger

# Describe the channel
a3.channels.c1.type = memory
a3.channels.c1.capacity = 1000
a3.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a3.sources.r1.channels = c1
a3.sinks.k1.channel = c1
```

4. 执行配置文件

分别开启对应配置文件：flume3.conf, flume2.conf, flume1.conf。

```
[atguigu@hadoop102 flume]$ bin/flume-ng agent --conf conf/ --name a3 --conf-file
job/group2/flume3.conf -Dflume.root.logger=INFO,console
[atguigu@hadoop104 flume]$ bin/flume-ng agent --conf conf/ --name a2 --conf-file
job/group2/flume2.conf
[atguigu@hadoop103 flume]$ bin/flume-ng agent --conf conf/ --name a1 --conf-file
job/group2/flume1.conf
```

5. 在 hadoop103 上向/opt/module 目录下的 group.log 追加内容

```
[atguigu@hadoop103 module]$ echo 'hello' > group.log
```

6. 在 hadoop104 上向 44444 端口发送数据

```
[atguigu@hadoop104 flume]$ telnet hadoop104 44444
```

7. 检查 hadoop102 上数据

```
2018-06-12 10:28:44,097 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink
.process(LoggerSink.java:95)] Event: { headers:{} body: 68 65 6C 6C 6F }
2018-06-12 10:28:48,479 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink
.process(LoggerSink.java:95)] Event: { headers:{} body: 68 65 6C 6C 6F }
```

第 4 章 Flume 监控之 Ganglia

4.1 Ganglia 的安装与部署

1) 安装 httpd 服务与 php

```
[atguigu@hadoop102 flume]$ sudo yum -y install httpd php
```

2) 安装其他依赖

```
[atguigu@hadoop102 flume]$ sudo yum -y install rrdtool perl-rrdtool rrdtool-devel
[atguigu@hadoop102 flume]$ sudo yum -y install apr-devel
```

3) 安装 ganglia

```
[atguigu@hadoop102 flume]$ sudo rpm -Uvh
http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
[atguigu@hadoop102 flume]$ sudo yum -y install ganglia-gmetad
[atguigu@hadoop102 flume]$ sudo yum -y install ganglia-web
[atguigu@hadoop102 flume]$ sudo yum install -y ganglia-gmond
```

4) 修改配置文件 ganglia.conf

```
[atguigu@hadoop102 flume]$ sudo vim /etc/httpd/conf.d/ganglia.conf
```

修改为红颜色的配置：

```
# Ganglia monitoring system php web frontend
Alias /ganglia /usr/share/ganglia
<Location /ganglia>
    Order deny,allow
```

```
Deny from all
Allow from all
# Allow from 127.0.0.1
# Allow from ::1
# Allow from .example.com
</Location>
```

5) 修改配置文件 gmetad.conf

```
[atguigu@hadoop102 flume]$ sudo vim /etc/ganglia/gmetad.conf
```

修改为:

```
data_source "hadoop102" 192.168.9.102
```

6) 修改配置文件 gmond.conf

```
[atguigu@hadoop102 flume]$ sudo vim /etc/ganglia/gmond.conf
```

修改为:

```
cluster {
    name = "hadoop102"
    owner = "unspecified"
    latlong = "unspecified"
    url = "unspecified"
}
udp_send_channel {
    #bind_hostname = yes # Highly recommended, soon to be default.
    # This option tells gmond to use a source address
    # that resolves to the machine's hostname. Without
    # this, the metrics may appear to come from any
    # interface and the DNS names associated with
    # those IPs will be used to create the RRDs.

    # mcast_join = 239.2.11.71
    host = 192.168.9.102
    port = 8649
    ttl = 1
}
udp_rcv_channel {
    # mcast_join = 239.2.11.71
    port = 8649
    bind = 192.168.9.102
    retry_bind = true
    # Size of the UDP buffer. If you are handling lots of metrics you really
    # should bump it up to e.g. 10MB or even higher.
    # buffer = 10485760
}
```

7) 修改配置文件 config

```
[atguigu@hadoop102 flume]$ sudo vim /etc/selinux/config
```

修改为:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#   enforcing - SELinux security policy is enforced.
#   permissive - SELinux prints warnings instead of enforcing.
#   disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of these two values:
#   targeted - Targeted processes are protected,
#   mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

尖叫提示: selinux 本次生效关闭必须重启, 如果此时不想重启, 可以临时生效之:

```
[atguigu@hadoop102 flume]$ sudo setenforce 0
```

5) 启动 ganglia

```
[atguigu@hadoop102 flume]$ sudo service httpd start
[atguigu@hadoop102 flume]$ sudo service gmetad start
[atguigu@hadoop102 flume]$ sudo service gmond start
```

6) 打开网页浏览 ganglia 页面

<http://192.168.9.102/ganglia>

尖叫提示：如果完成以上操作依然出现权限不足错误，请修改/var/lib/ganglia 目录的权限：

```
[atguigu@hadoop102 flume]$ sudo chmod -R 777 /var/lib/ganglia
```

4.2 操作 Flume 测试监控

1) 修改/opt/module/flume/conf 目录下的 flume-env.sh 配置：

```
JAVA_OPTS="-Dflume.monitoring.type=ganglia
-Dflume.monitoring.hosts=192.168.9.102:8649
-Xms100m
-Xmx200m"
```

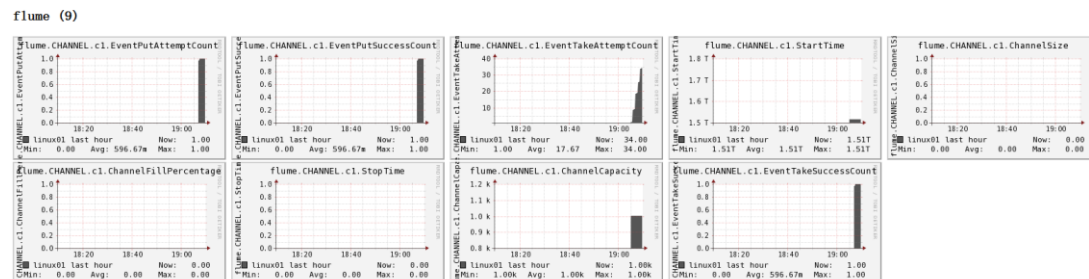
2) 启动 flume 任务

```
[atguigu@hadoop102 flume]$ bin/flume-ng agent \
--conf conf/ \
--name a1 \
--conf-file job/flume-telnet-logger.conf \
-Dflume.root.logger==INFO,console \
-Dflume.monitoring.type=ganglia \
-Dflume.monitoring.hosts=192.168.9.102:8649
```

3) 发送数据观察 ganglia 监测图

```
[atguigu@hadoop102 flume]$ telnet localhost 44444
```

样式如图：



图例说明：

字段（图表名称）	字段含义
EventPutAttemptCount	source 尝试写入 channel 的事件总数量
EventPutSuccessCount	成功写入 channel 且提交的事件总数量
EventTakeAttemptCount	sink 尝试从 channel 拉取事件的总数量。这不意味着每次事件都被返回，因为 sink 拉取的时候 channel 可能没有任何数据。
EventTakeSuccessCount	sink 成功读取的事件的总数量
StartTime	channel 启动的时间（毫秒）
StopTime	channel 停止的时间（毫秒）
ChannelSize	目前 channel 中事件的总数量
ChannelFillPercentage	channel 占用百分比
ChannelCapacity	channel 的容量

第 5 章 自定义 MySQLSource

5.1 自定义 Source 说明

Source 是负责接收数据到 Flume Agent 的组件。Source 组件可以处理各种类型、各种格式的日志数据，包括 avro、thrift、exec、jms、spooling directory、netcat、sequence generator、syslog、http、legacy。官方提供的 source 类型已经很多，但是有时候并不能满足实际开发当中的需求，此时我们就需要根据实际需求自定义某些 source。

如：实时监控 MySQL，从 MySQL 中获取数据传输到 HDFS 或者其他存储框架，所以此时需要我们自己实现 MySQLSource。

官方也提供了自定义 source 的接口：

官网说明：<https://flume.apache.org/FlumeDeveloperGuide.html#source>

5.3 自定义 MySQLSource 组成

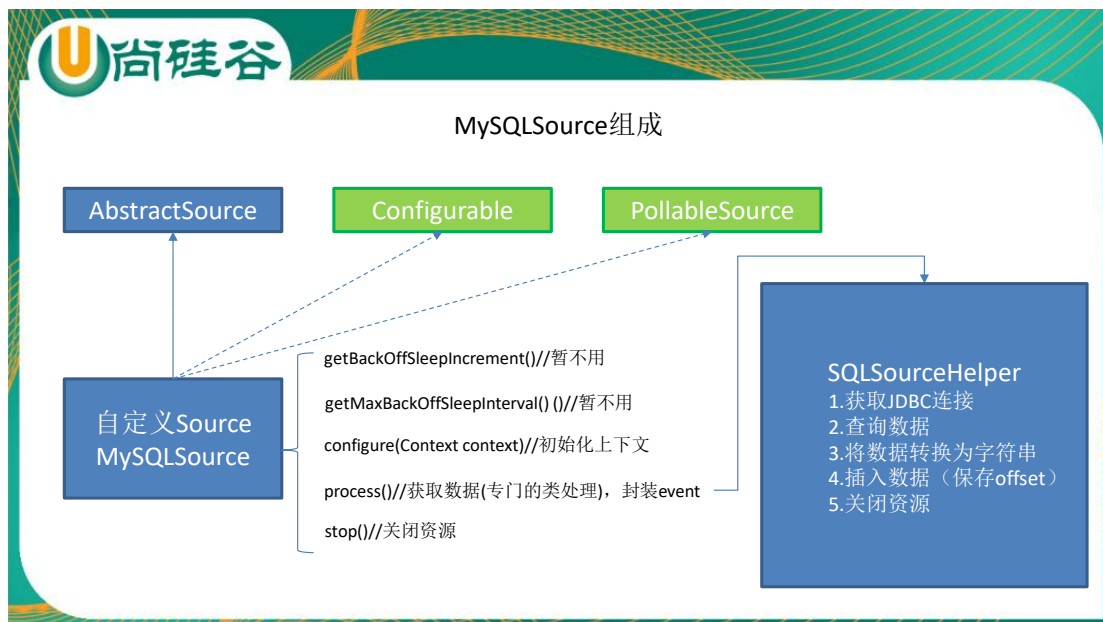


图 6-1 自定义 MySQLSource 组成

5.2 自定义 MySQLSource 步骤

根据官方说明自定义 mysqlsource 需要继承 AbstractSource 类并实现 Configurable 和 PollableSource 接口。

实现相应方法：

```
getBackOffSleepIncrement()//暂不用  
getMaxBackOffSleepInterval()//暂不用  
configure(Context context)//初始化 context
```

process()//获取数据（从 mysql 获取数据，业务处理比较复杂，所以我们定义一个专门的类——SQLSourceHelper 来处理跟 mysql 的交互），封装成 event 并写入 channel，这个方法被循环调用

```
stop()//关闭相关的资源
```

5.4 代码实现

5.4.1 导入 pom 依赖

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

5.4.2 添加配置信息

在 classpath 下添加 jdbc.properties 和 log4j.properties

jdbc.properties:

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

log4j.properties:

```
#-----console-----
log4j.rootLogger=info,myconsole,myfile
log4j.appender.myconsole=org.apache.log4j.ConsoleAppender
log4j.appender.myconsole.layout=org.apache.log4j.SimpleLayout
#log4j.appender.myconsole.layout.ConversionPattern=%d [%t] %-5p [%c] - %m%n

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

5.4.3 SQLSourceHelper

1) 属性说明:

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

2) 方法说明:

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可百度访问：尚硅谷官网

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

3) 代码实现:

```
package com.atguigu.source;

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

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

public class SQLSourceHelper {

    private static final Logger LOG = LoggerFactory.getLogger(SQLSourceHelper.class);

    private int runQueryDelay, //两次查询的时间间隔
        startFrom, //开始 id
        currentIndex, //当前 id
        recordSixe = 0, //每次查询返回结果的条数
        maxRow; //每次查询的最大条数

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

    //上下文，用来获取配置文件
    private Context context;

    //为定义的变量赋值（默认值），可在 flume 任务的配置文件中修改
    private static final int DEFAULT_QUERY_DELAY = 10000;
    private static final int DEFAULT_START_VALUE = 0;
    private static final int DEFAULT_MAX_ROWS = 2000;
    private static final String DEFAULT_COLUMNS_SELECT = "*";
    private static final String DEFAULT_CHARSET_RESULTSET = "UTF-8";

    private static Connection conn = null;
    private static PreparedStatement ps = null;
```



```
private static String connectionURL, connectionUserName, connectionPassword;

//加载静态资源
static {
    Properties p = new Properties();
    try {

p.load(SQLSourceHelper.class.getClassLoader().getResourceAsStream("jdbc.properties"));
        connectionURL = p.getProperty("dbUrl");
        connectionUserName = p.getProperty("dbUser");
        connectionPassword = p.getProperty("dbPassword");
        Class.forName(p.getProperty("dbDriver"));
    } catch (IOException | ClassNotFoundException e) {
        LOG.error(e.toString());
    }
}

//获取 JDBC 连接
private static Connection InitConnection(String url, String user, String pw) {
    try {
        Connection conn = DriverManager.getConnection(url, user, pw);
        if (conn == null)
            throw new SQLException();
        return conn;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}

//构造方法
SQLSourceHelper(Context context) throws ParseException {
    //初始化上下文
    this.context = context;

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

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

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

//校验相应的配置信息（表，查询语句以及数据库连接的参数）
private void checkMandatoryProperties() {
    if (table == null) {
```

```
        throw new ConfigurationException("property table not set");
    }
    if (connectionURL == null) {
        throw new ConfigurationException("connection.url property not set");
    }
    if (connectionUserName == null) {
        throw new ConfigurationException("connection.user property not set");
    }
    if (connectionPassword == null) {
        throw new ConfigurationException("connection.password property not set");
    }
}

//构建 sql 语句
private String buildQuery() {
    String sql = "";
    //获取当前 id
    currentIndex = getStatusDBIndex(startFrom);
    LOG.info(currentIndex + "");
    if (customQuery == null) {
        sql = "SELECT " + columnsToSelect + " FROM " + table;
    } else {
        sql = customQuery;
    }
    StringBuilder execSql = new StringBuilder(sql);
    //以 id 作为 offset
    if (!sql.contains("where")) {
        execSql.append(" where ");
        execSql.append("id").append(">").append(currentIndex);
        return execSql.toString();
    } else {
        int length = execSql.toString().length();
        return execSql.toString().substring(0, length
String.valueOf(currentIndex).length()) + currentIndex;
    }
}

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

```
// 重新连接
conn = InitConnection(connectionURL, connectionUserName, connectionPassword);
}
return null;
}

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

//更新 offset 元数据状态，每次返回结果集后调用。必须记录每次查询的 offset 值，为程序中断续跑数据时使用，以 id 为 offset
void updateOffset2DB(int size) {
    //以 source_tab 做为 KEY，如果不存在则插入，存在则更新（每个源表对应一条记录）
    String sql = "insert into flume_meta(source_tab,currentIndex) VALUES('"
        + this.table
        + "','" + (recordSixe += size)
        + "','" + "on DUPLICATE key update"
        + "source_tab=values(source_tab),currentIndex=values(currentIndex)";
    LOG.info("updateStatus Sql:" + sql);
    execSql(sql);
}

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

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

```
}

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

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

int getCurrentIndex() {
    return currentIndex;
}

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

int getRunQueryDelay() {
    return runQueryDelay;
}

String getQuery() {
    return query;
}

String getConnectionURL() {
    return connectionURL;
}

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

Context getContext() {
    return context;
}

public String getConnectionUserName() {
    return connectionUserName;
}

public String getConnectionPassword() {
    return connectionPassword;
}
```

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

5.4.4 MySQLSource

代码实现：

```
package com.atguigu.source;  
  
import org.apache.flume.Context;  
import org.apache.flume.Event;  
import org.apache.flume.EventDeliveryException;  
import org.apache.flume.PollableSource;  
import org.apache.flume.conf.Configurable;  
import org.apache.flume.event.SimpleEvent;  
import org.apache.flume.source.AbstractSource;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.text.ParseException;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
  
public class SQLSource extends AbstractSource implements Configurable, PollableSource {  
  
    //打印日志  
    private static final Logger LOG = LoggerFactory.getLogger(SQLSource.class);  
    //定义 sqlHelper  
    private SQLSourceHelper sqlSourceHelper;  
  
    @Override  
    public long getBackOffSleepIncrement() {  
        return 0;  
    }  
  
    @Override  
    public long getMaxBackOffSleepInterval() {  
        return 0;  
    }  
  
    @Override  
    public void configure(Context context) {  
        try {  
            //初始化  
            sqlSourceHelper = new SQLSourceHelper(context);  
        } catch (ParseException e) {  
            e.printStackTrace();  
        }  
    }  
  
    @Override  
    public Status process() throws EventDeliveryException {  
        try {  
            //查询数据表  
            List<List<Object>> result = sqlSourceHelper.executeQuery();  
            //存放 event 的集合  
            List<Event> events = new ArrayList<>();  
            //存放 event 头集合
```

```
HashMap<String, String> header = new HashMap<>();
//如果有返回数据，则将数据封装为 event
if (!result.isEmpty()) {
    List<String> allRows = sqlSourceHelper.getAllRows(result);
    Event event = null;
    for (String row : allRows) {
        event = new SimpleEvent();
        event.setBody(row.getBytes());
        event.setHeaders(header);
        events.add(event);
    }
    //将 event 写入 channel
    this.getChannelProcessor().processEventBatch(events);
    //更新数据表中的 offset 信息
    sqlSourceHelper.updateOffset2DB(result.size());
}
//等待时长
Thread.sleep(sqlSourceHelper.getRunQueryDelay());
return Status.READY;
} catch (InterruptedException e) {
    LOG.error("Error procesing row", e);
    return Status.BACKOFF;
}
}

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

5.5 测试

5.5.1 jar 包准备

1) 将 mysql 驱动包也放入 flume 的 lib 目录下

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

2) 打包项目并将 jar 包放入 flume 的 lib 目录下

5.5.2 配置文件准备

1) 创建配置文件并打开

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

2) 添加如下内容

```
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1
```

```
# Describe/configure the source
a1.sources.r1.type = com.atguigu.source.SQLSource
a1.sources.r1.connection.url = jdbc:mysql://192.168.9.102:3306/ct
a1.sources.r1.connection.user = root
a1.sources.r1.connection.password = 000000
a1.sources.r1.table = wlslog
a1.sources.r1.columns.to.select = *
a1.sources.r1.incremental.column.name = id
a1.sources.r1.incremental.value = 0
a1.sources.r1.run.query.delay=5000

# Describe the sink
a1.sinks.k1.type = logger

# Describe the channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

5.5.3 mysql 表准备

1) 创建 mysqlsource 数据库

```
CREATE DATABASE mysqlsource;
```

2) 在 mysqlsource 数据库下创建数据表 student 和元数据表 flume_meta

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

3) 向数据表中添加数据

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

5.5.4 测试并查看结果

1) 任务执行

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

2) 结果展示，如图 6-2 所示：


```

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

```

图 6-2 结果展示

第 6 章 扩展

6.1 常见正则表达式语法

元字符	描述
^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配“\n”或“\r”之后的位置。
\$	匹配输入字符串的结束位置。如果设置了 RegExp 对象的 Multiline 属性，\$ 也匹配“\n”或“\r”之前的位置。
*	匹配前面的子表达式任意次。例如，zo*能匹配“z”，“zo”以及“zoo”。*等价于{0,}。
+	匹配前面的子表达式一次或多次(大于等于 1 次)。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。
[a-z]	<p>字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。</p> <p>注意:只有连字符在字符组内部时,并且出现在两个字符之间时,才能表示字符的范围;如果出字符组的开头,则只能表示连字符本身。</p>

7.2 练习

案例需求:

- 1) flume-1 监控 hive.log 日志, flume-1 的数据传送给 flume-2, flume-2 将数据追加到本

地文件，同时将数据传输到 flume-3。

2) flume-4 监控本地另一个自己创建的文件 any.txt，并将数据传送给 flume-3。

3) flume-3 将汇总数据写入到 HDFS。

请先画出结构图，再开始编写任务脚本。