

# Flume 简介

---

## 学生指南

## 商标产权声明

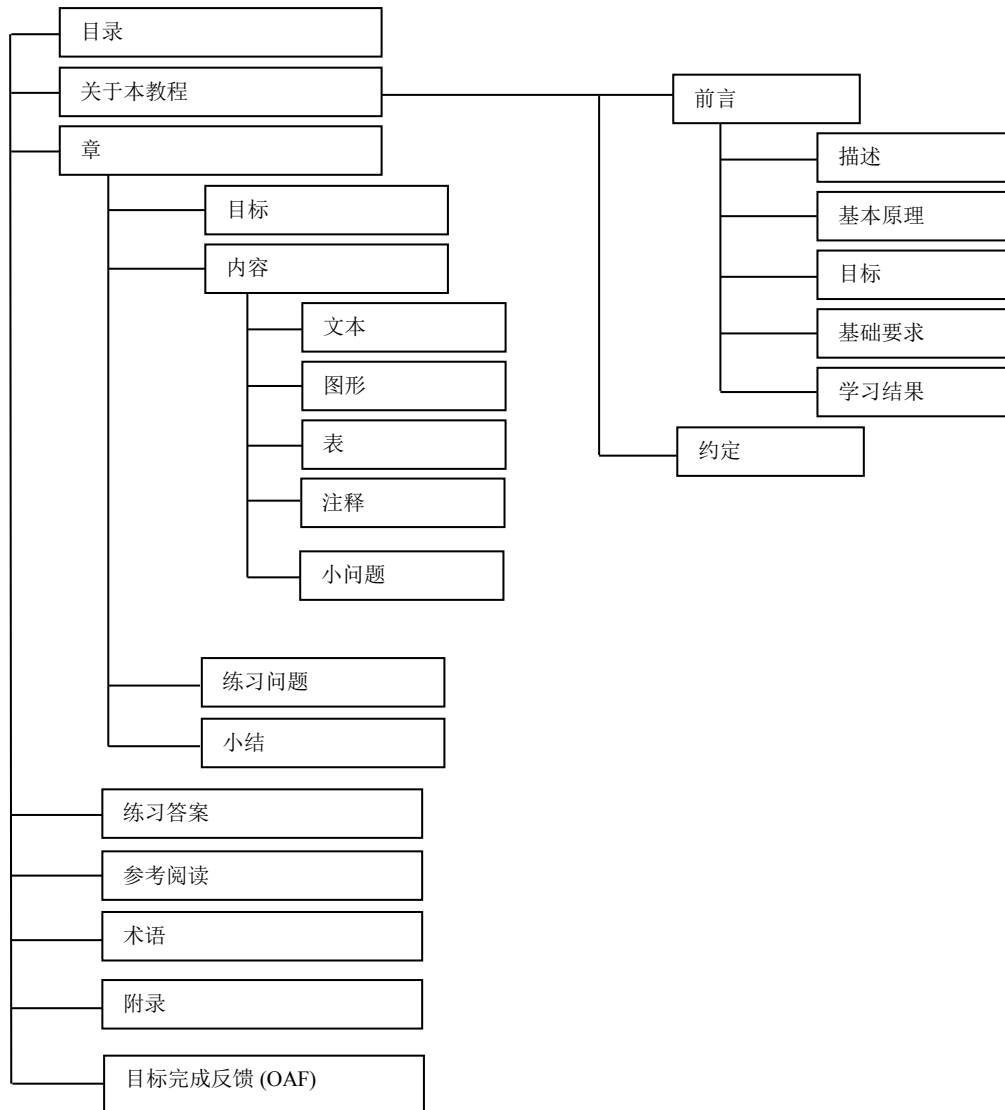
所有产品都是各自组织的注册商标。  
所有软件仅用于教育用途。

Flume 简介 SG/13-M04-V1.0  
版权 ©NIIT。保留一切权利。

未经出版商提前书面许可，禁止以任何形式或任何手段（电子、机械、影印、转录或其他方式）对本出版物的某一部分进行复制、传播或将其存储在检索系统中。

印制：Sona Printers Pvt. Ltd. B-181 Okhla Ph-1, N.D.-20 电话：26811313-4-5-6

# 教程设计 - 学生指南





# 目录

## 关于本教程

前言	i
描述	i
目标	i
基础要求	ii
学习结果	ii
约定	iii

## 第 1 章 – Flume 基础

<b>Flume 简介</b>	<b>1.4</b>
概念介绍	1.4
特点	1.4
Flume 组成组件	1.5
<b>Flume 安装配置</b>	<b>1.6</b>
软件安装	1.6
环境验证	1.7
<b>Flume 数据流模型</b>	<b>1.9</b>
数据流模型基本概念	1.9
数据源	1.9
Channel	1.11
sink	1.12
常用数据流模型	1.12
<b>Flume 整合 kafka</b>	<b>1.19</b>
<b>Flume 常见问题</b>	<b>1.27</b>
<b>练习问题</b>	<b>1.21</b>
小结	4.22

## 参考阅读

**Flume 简介-----R.3**

---

## 关于本教程





# 前言

## 描述

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 支持在日志系统中定制各类数据发送方，用于收集数据；同时，Flume 提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力。

## 目标

完成此教程后，学生将能够熟练使用以下工具：

- Flume 原理
- Flume 安装配置
- Flume 案例

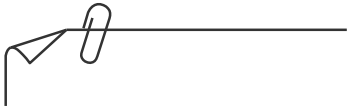
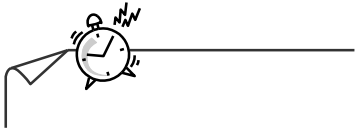

## 基础要求

想要学习此教程的学生应该具备逻辑构建和有效问题解决的基本知识。

## 学习结果

完成此教程后，学生将能够使用 相关工具配合 `hadoop`、`hbase`、`storm` 一起使用

约定

约定	表示...
	注释
	小问题
	活动的占位符



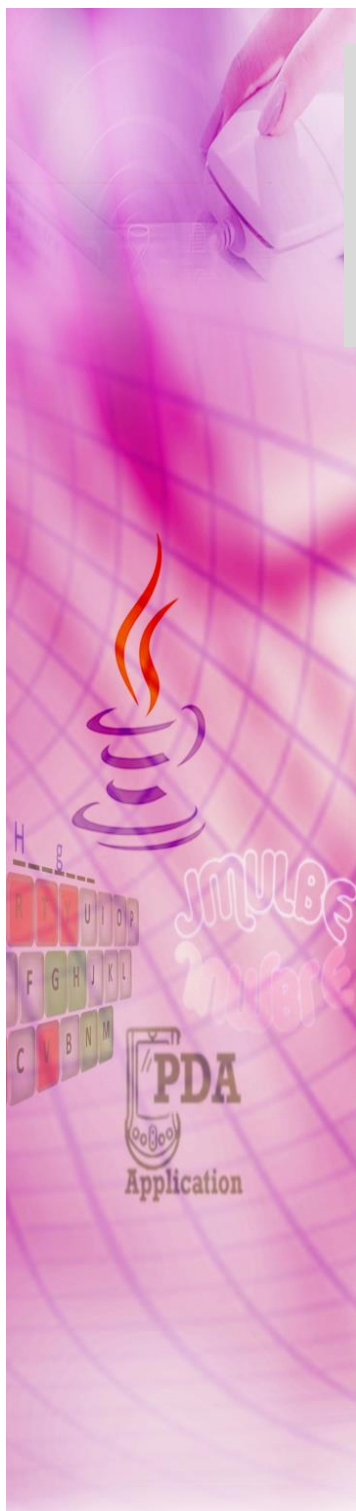
# Flume

在互联网中服务器一般都采用集群方式部署来完成，以满足高并发、高访问量的业务需求。但是问题随之而来，各业务系统记录的日志都是独立的，如果要查看各业务系统的日志必须登录每台服务器，导致复杂性增强、安全性降低。Flume 的出现刚好解决了这样的问题。

## 目标

在本章中，您将学习：

- Flume 简介
- flume 安装配置
- Flume 数据流模型
- Flume 数据源读取方式
- Flume 集成 kafka
- Flume 常见问题

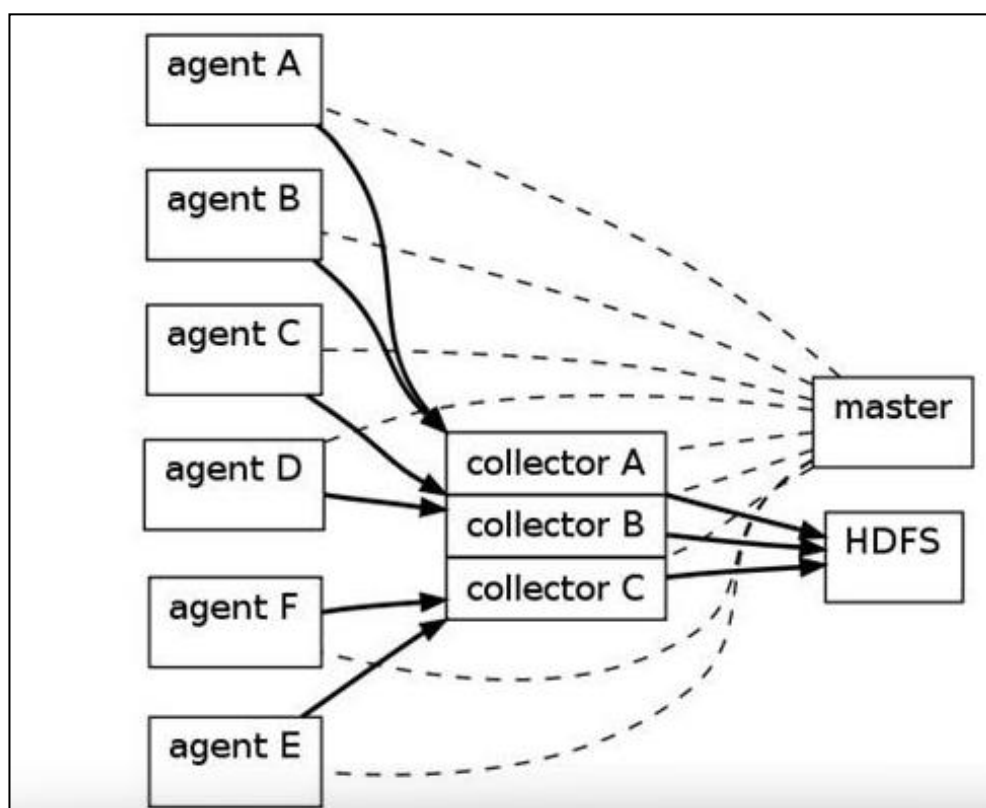


# Flume 简介

## ■ 概念

Flume 是 Cloudera 提供的日志收集系统，Flume 支持在日志系统中定制各类数据发送方，用于收集数据；同时，Flume 提供对数据进行简单处理，并写到各种 storage。Flume 是一个分布式、可靠、和高可用的海量日志采集、聚合和传输的系统。

在 Flume 中，最重要的抽象是 data flow(数据流)，data flow 描述了数据从产生，传输、处理并最终写入目标的一条路径，如下图所描述。



## ■ Flume 特点

Flume 支持可靠性、伸缩性、配置一致性、扩展性。

- ✧ **可靠性:** Flume 提供 3 中数据可靠性选项，包括 End-to-end、Store on failure 和 Best effort。其中 End-to-end 使用了磁盘日志和接受端 Ack 的方式，保证 Flume 接受到的数据会最终到达目的。Store on failure 在目的不可用的时候，数据会保持在本地硬盘。和

End-to-end 不同的是，如果是进程出现问题，Store on failure 可能会丢失部分数据。

Best effort 不做任何 QoS 保证。

- ✧ **伸缩性:** Flume 的 3 大组件: collector、master 和 storage tier 都是可伸缩的。需要注意的是，Flume 中对事件的处理不需要带状态，它的伸缩性可以很容易实现。
- ✧ **配置一致性:** Flume 利用 ZooKeeper 和 gossip，保证配置数据的一致性、高可用。同时，多 Master，保证 Master 可以管理大量的节点。
- ✧ **扩展性:** 基于 Java，用户可以为 Flume 添加各种新的功能，如通过继承 Source，用户可以实现自己的数据接入方式，实现 Sink 的子类，用户可以将数据写往特定目标，同时，通过 SinkDecorator，用户可以对数据进行一定的预处理。

## ■ Flume 组成组件

Event：一个数据单元，带有一个可选的消息头，可以是日志记录、 avro 对象等

Collector ：用于对数据进行聚合。

Flow：Event 从源点到达目的点的迁移的抽象

Client：操作位于源点处的 Event，将其发送到 Flume Agent

Agent：一个独立的 Flume 进程，负责数据收集，包含组件 Source、Channel、Sink

Source：数据源，用来消费传递到该组件的 Event，每个 Agent, Collector 都可以有一个数据源

Channel：连接 sources 和 sinks ，有点像一个队列，中转 Event 的一个临时存储，保存有

Sink：输出端，从 Channel 中读取并移除 Event，将 Event 传递到 Flow Pipeline 中的下一个 Agent（如果有的话）

# Flume 安装配置

## 软件安装

### ■ Flume 安装

下载 flume1.5

```
wget http://mirrors.cnnic.cn/apache/flume/1.5.0/apache-flume-1.5.0-bin.tar.gz
```

```
tar -zxvf apache-flume-1.5.0-bin.tar
```

### ■ 环境变量设置

```
vim /etc/profile
```

```
export FLUME_HOME=/opt/niit/tools/apache-flume-1.5.0
```

```
export FLUME_CONF_DIR=$FLUME_HOME/conf
```

```
export PATH=.:$PATH:$FLUME_HOME/bin
```

```
source /etc/profile
```

```
cd $FLUME_HOME/conf
```

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

```
vi flume-env.sh
```

```
JAVA_HOME=/opt/niit/jdk1.7.0_71
```

### ■ 验证 flume 版本

进入 flume 安装目录的 bin 目录下执行 `./flume-ng version`，如果出现下面的内容表示安装成功

```
[root@hdname bin]# ./flume-ng version
Flume 1.5.0
Source code repository: https://git-wip-us.apache.org/repos/asf/flume.git
Revision: 8633220df808c4cd0c13d1cf0320454a94f1ea97
Compiled by hshreedharan on Wed May  7 14:49:18 PDT 2014
From source with checksum a01fe726e4380ba0c9f7a7d222db961f
```

## 环境验证



## ■ 创建服务器端

在\$FLUME\_HOME/conf 目录下创建一个文件 example.conf

```
vi example.conf
```

```
# example.conf: A single-node Flume configuration
```

```
# 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
```

命令行下执行：

```
../bin/flume-ng agent --conf conf --conf-file example.conf --name a1 -
```

```
Dflume.root.logger=INFO,console
```

```
15/07/19 05:15:10 INFO sink.DefaultSinkFactory: Creating instance of sink: k1, type: logger
15/07/19 05:15:10 INFO node.AbstractConfigurationProvider: Channel c1 connected to [r1, k1]
15/07/19 05:15:10 INFO node.Application: Starting new configuration:{ sourceRunners:{r1=EventDrivenSource
Runner: { source:org.apache.flume.source.NetcatSource{name:r1,state:IDLE} }} sinkRunners:{k1=SinkRunner:
{ policy:org.apache.flume.sink.DefaultSinkProcessor@1df304 counterGroup:{ name:null counters:{} } }} chan
nels:{c1=org.apache.flume.channel.MemoryChannel{name: c1}} }
15/07/19 05:15:10 INFO node.Application: Starting Channel c1
15/07/19 05:15:10 INFO instrumentation.MonitoredCounterGroup: Monitored counter group for type: CHANNEL,
name: c1: Successfully registered new MBean.
15/07/19 05:15:10 INFO instrumentation.MonitoredCounterGroup: Component type: CHANNEL, name: c1 started
15/07/19 05:15:10 INFO node.Application: Starting Sink k1
15/07/19 05:15:10 INFO node.Application: Starting Source r1
15/07/19 05:15:10 INFO source.NetcatSource: Source starting
15/07/19 05:15:10 INFO source.NetcatSource: Created serverSocket:sun.nio.ch.ServerSocketChannelImpl[127.
0.0.1:44444]
```

## ■ 创建客户端

telnet localhost 44444

输入内容后敲回车键

```
[root@hdname ~]# telnet localhost 44444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
█
```

服务器端就会输出最终结果

```
15/07/19 05:15:10 INFO instrumentation.MonitoredCounterGroup: Component type: CHANNEL, name: c1 started
15/07/19 05:15:10 INFO node.Application: Starting Sink k1
15/07/19 05:15:10 INFO node.Application: Starting Source r1
15/07/19 05:15:10 INFO source.NetcatSource: Source starting
15/07/19 05:15:10 INFO source.NetcatSource: Created serverSocket:sun.nio.ch.ServerSocketChannelImpl[/127.
0.0.1:44444]
15/07/19 05:16:04 INFO sink.LoggerSink: Event: { headers:{} body: 68 65 6C 6C 6F 20 6E 69 69 74 0D
hello niit. }
```



## 活动 1.1: 在 Liunx 下安装 Flume

# Flume 数据流模型

## 数据流模型基本概念

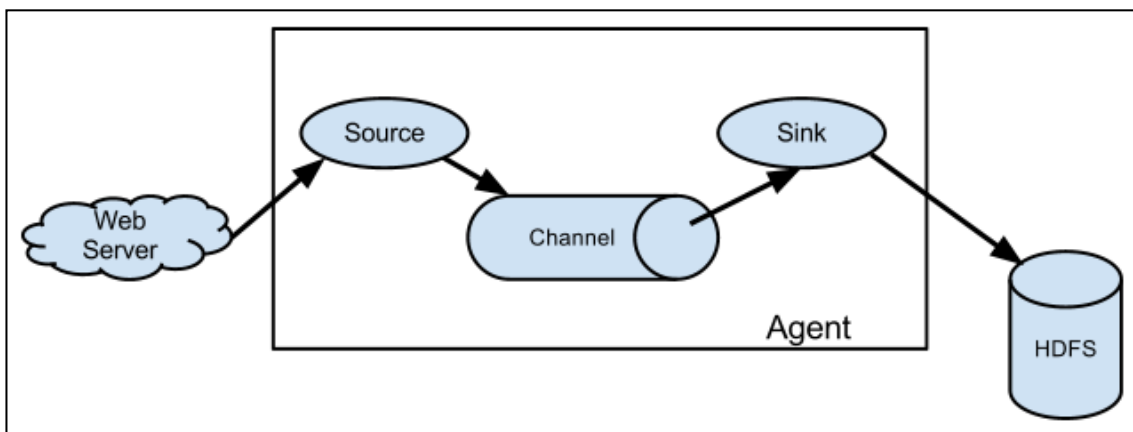
Flume 传输的数据的基本单位是 event，如果是文本文件，通常是一行记录，这也是事务的基本单位。flume 运行的核心是 agent。它是一个完整的数据收集工具，含有三个核心组件，分别是 source、channel、sink。

Event 从 Source，流向 Channel，再到 Sink，本身为一个 byte 数组，并可携带 headers 信息。Event 代表着一个数据流的最小完整单元，从外部数据源来，向外部的目的地去。

Source:完成对日志数据的收集，分成 transtion 和 event 打入到 channel 之中。

Channel:主要提供一个队列的功能，对 source 提供中的数据简单的缓存。

Sink:取出 Channel 中的数据，进行相应的存储文件系统，数据库，或者提交到远程服务器。通过这些组件，event 可以从一个地方流向另一个地方，如下图所示



## 数据源

### ■ Exec source

可通过写 Unix command 的方式组织数据，最常用的就是 `tail -F [file]`。

可以实现实时传输，但在 flume 不运行和脚本错误时，会丢数据，也不支持断点续传功能。因为没有记录上次文件读到的位置，从而没办法知道，下次再读时，从什么地方开始读。特别是

在日志文件一直在增加的时候。flume 的 source 挂了。等 flume 的 source 再次开启的这段时间内，增加的日志内容，就没办法被 source 读取到了。不过 flume 有一个 execStream 的扩展，可以自己写一个监控日志增加情况，把增加的日志，通过自己写的工具把增加的内容，传送给 flume 的 node。再传送给 sink 的 node。

■ Spooling Directory Source

监测配置的目录下新增的文件，并将文件中的数据读取出来，可实现准实时。需要注意两点：1、拷贝到 spool 目录下的文件不可以再打开编辑。2、spool 目录下不可包含相应的子目录。在实际使用的过程中，可以结合 log4j 使用，使用 log4j 的时候，将 log4j 的文件分割机制设为 1 分钟一次，将文件拷贝到 spool 的监控目录。log4j 有一个 TimeRolling 的插件，可以把 log4j 分割的文件到 spool 目录。基本实现了实时的监控。Flume 在传完文件之后，将会修改文件的后缀，变为 .COMPLETED（后缀也可以在配置文件中灵活指定）

ExecSource，SpoolSource 对比：ExecSource 可以实现对日志的实时收集，但是存在 Flume 不运行或者指令执行出错时，将无法收集到日志数据，无法何证日志数据的完整性。SpoolSource 虽然无法实现实时的收集数据，但是可以使用以分钟的方式分割文件，趋近于实时。如果应用无法实现以分钟切割日志文件的话，可以两种收集方式结合使用。

Source 类型	说明
Avro Source	支持 Avro 协议（实际上是 Avro RPC），内置支持
Thrift Source	支持 Thrift 协议，内置支持
Exec Source	Exec Source   基于 Unix 的 command 在标准输出上生产数据
JMS Source	从 JMS 系统（消息、主题）中读取数据，ActiveMQ 已经测试过
Spooling Directory Source	监控指定目录内数据变更
Twitter 1% firehose Source	通过 API 持续下载 Twitter 数据，试验性质
Netcat Source	监控某个端口，将流经端口的每一个文本行数据作为 Event 输入
Sequence Generator Source	序列生成器数据源，生产序列数据

Source 类型	说明
Syslog Sources	读取 syslog 数据，产生 Event，支持 UDP 和 TCP 两种协议
HTTP Source	基于 HTTP POST 或 GET 方式的数据源，支持 JSON、BLOB 表示形式
Legacy Sources	兼容老的 Flume OG 中 Source (0.9.x 版本)

## Channel

当前有几个 channel 可供选择，分别是 Memory Channel, JDBC Channel, File Channel, Psuedo Transaction Channel。比较常见的是前三种 channel。

- ✧ MemoryChannel 可以实现高速的吞吐，但是无法保证数据的完整性。
- ✧ MemoryRecoverChannel 在官方文档的建议上已经建议使用 FileChannel 来替换。
- ✧ FileChannel 保证数据的完整性与一致性。在具体配置 FileChannel 时，建议 FileChannel 设置的目录和程序日志文件保存的目录设成不同的磁盘，以便提高效率。

File Channel 是一个持久化的隧道 (channel)，它持久化所有的事件，并将其存储到磁盘。因此，即使 Java 虚拟机当掉，或者操作系统崩溃或重启，再或者事件没有在管道中成功地传递到下一个代理 (agent)，这一切都不会造成数据丢失。Memory Channel 是一个不稳定的隧道，其原因是由于它在内存中存储所有事件。如果 java 进程死掉，任何存储在内存的事件将会丢失。另外，内存的空间收到 RAM 大小的限制，而 File Channel 这方面是它的优势，只要磁盘空间足够，它就可以将所有事件数据存储到磁盘上。

Flume Channel 支持的类型：

Channel 类型	说明
Memory Channel	Event 数据存储于内存中
JDBC Channel	Event 数据存储于持久化存储中，当前 Flume Channel 内置支持 Derby
File Channel	Event 数据存储于磁盘文件中
Spillable Memory	Event 数据存储于内存中和磁盘上，当内存队列满了，会持久化到磁盘

Channel 类型	说明
Channel	文件（当前试验性的，不建议生产环境使用）
Pseudo Transaction Channel	测试用途
Custom Channel	自定义 Channel 实现

## sink

Sink 在设置存储数据时，可以向文件系统、数据库、hadoop 存数据，在日志数据较少时，可以将数据存储在文件系中，并且设定一定的时间间隔保存数据。在日志数据较多时，可以将相应的日志数据存储到 Hadoop 中，便于日后进行相应的数据分析。

Flume Sink 支持的类型

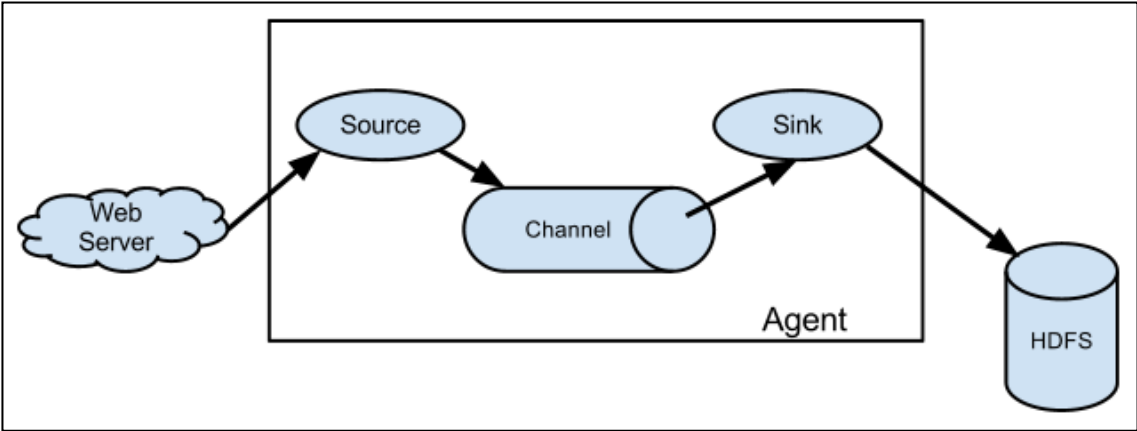
Sink 类型	说明
HDFS Sink	数据写入 HDFS
Logger Sink	数据写入日志文件
Avro Sink	数据被转换成 Avro Event，然后发送到配置的 RPC 端口上
Thrift Sink	数据被转换成 Thrift Event，然后发送到配置的 RPC 端口上
IRC Sink	数据在 IRC 上进行回放
File Roll Sink	存储数据到本地文件系统
Null Sink	丢弃到所有数据
HBase Sink	数据写入 HBase 数据库
Morphline Solr Sink	数据发送到 Solr 搜索服务器（集群）
ElasticSearch Sink	数据发送到 Elastic Search 搜索服务器（集群）

Sink 类型	说明
Kite Dataset Sink	写数据到 Kite Dataset，试验性质的
Custom Sink	自定义 Sink 实现

## 常用数据流模型

### ■ 单节点 Flume 直接写入 HDFS

数据流模型如下



配置文件定义的格式如下：

# list the sources, sinks and channels for the agent

```
<Agent>.sources = <Source>
```

```
<Agent>.sinks = <Sink>
```

```
<Agent>.channels = <Channel1> <Channel2>
```

# set channel for source

```
<Agent>.sources.<Source>.channels = <Channel1> <Channel2> ...
```

# set channel for sink

```
<Agent>.sinks.<Sink>.channel = <Channel1>
```

# properties for sources

```

<Agent>.sources.<Source>.<someProperty> = <someValue>

# properties for channels

<Agent>.channel.<Channel>.<someProperty> = <someValue>

# properties for sinks

<Agent>.sources.<Sink>.<someProperty> = <someValue>

# Define a memory channel called ch1 on agent1
agent1.channels.ch1.type = memory
agent1.channels.ch1.capacity = 100000
agent1.channels.ch1.transactionCapacity = 100000
agent1.channels.ch1.keep-alive = 30
#define source monitor a file
agent1.sources.avro-source1.type = exec
agent1.sources.avro-source1.shell = /bin/bash -c
agent1.sources.avro-source1.command = tail -n +0 -F
/home/storm/tmp/id.txt
agent1.sources.avro-source1.channels = ch1
agent1.sources.avro-source1.threads = 5
# Define a logger sink that simply logs all events it receives
# and connect it to the other end of the same channel.
agent1.sinks.log-sink1.channel = ch1
agent1.sinks.log-sink1.type = hdfs
agent1.sinks.log-sink1.hdfs.path = hdfs://192.168.1.111:8020/flumeTest
agent1.sinks.log-sink1.hdfs.writeFormat = Text
agent1.sinks.log-sink1.hdfs.fileType = DataStream
agent1.sinks.log-sink1.hdfs.rollInterval = 0
agent1.sinks.log-sink1.hdfs.rollSize = 1000000
agent1.sinks.log-sink1.hdfs.rollCount = 0
agent1.sinks.log-sink1.hdfs.batchSize = 1000
agent1.sinks.log-sink1.hdfs.txnEventMax = 1000
agent1.sinks.log-sink1.hdfs.callTimeout = 60000
agent1.sinks.log-sink1.hdfs.appendTimeout = 60000
# Finally, now that we've defined all of our components, tell
# agent1 which ones we want to activate.
agent1.channels = ch1
agent1.sources = avro-source1
agent1.sinks = log-sink1

../bin/flume-ng agent --conf ../conf/ -f flume_directHDFS.conf -n agent1 -
Dflume.root.logger=INFO,console

```

## 参数说明

✧ -n 指定 agent 名称



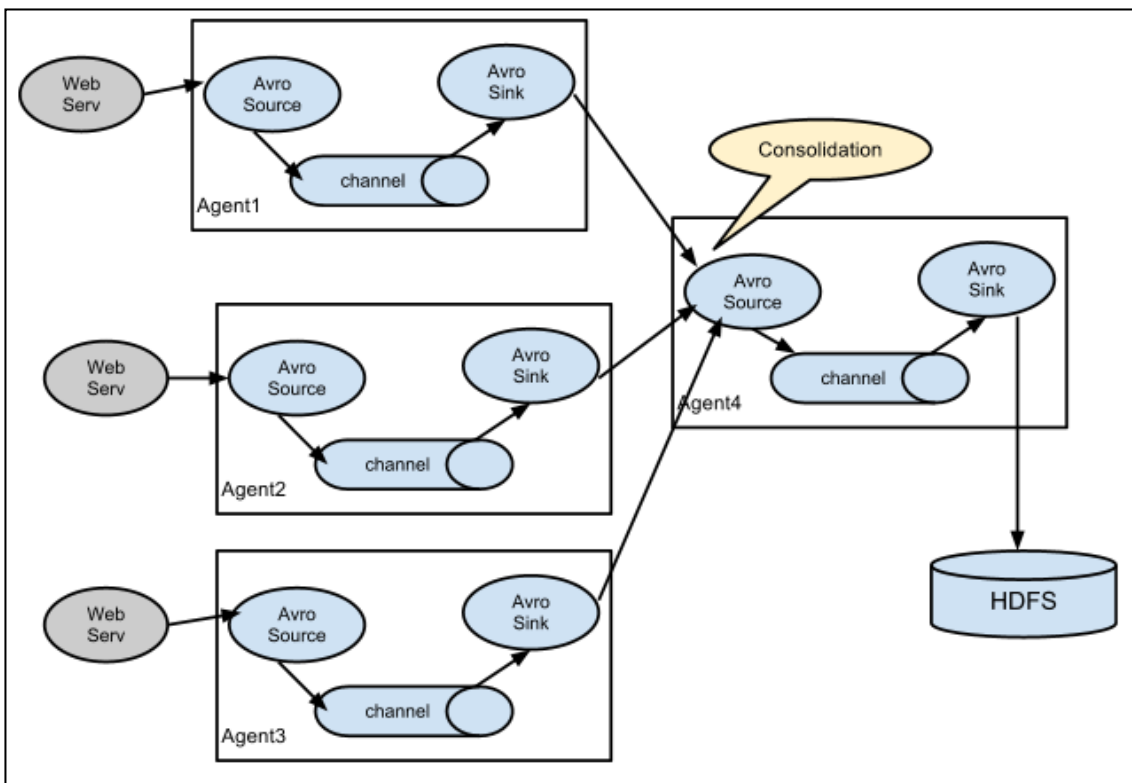
- ✧ -c 指定配置文件目录
- ✧ -f 指定配置文件
- ✧ -Dflume.root.logger=DEBUG,console 设置日志等级

说明：实际环境中有这样的需求，通过在多个 agent 端 tail 日志，发送给 collector，collector 再把数据收集，统一发送给 HDFS 存储起来，当 HDFS 文件大小超过一定的大小或者超过在规定的的时间间隔会生成一个文件。Flume 实现了两个 Trigger，SizeTriger 和 TimeTriger

- ✧ **SizeTriger**（在调用 HDFS 输出流写的同时，count 该流已经写入的大小总和，若超过一定大小，则创建新的文件和输出流，写入操作指向新的输出流，同时 close 以前的输出流）
- ✧ **TimeTriger**（开启定时器，当到达该点时，自动创建新的文件和输出流，新的写入重定向到该流中，同时 close 以前的输出流）。

## ■ 多 agent 汇聚写入 HDFS

这种模型需要分 2 步配置，首先配置 Flume Client，再配置 Flume Server



## ■ 在各个 webserv 日志机上配置 Flume Client

```
# clientMainAgent
clientMainAgent.channels = c1
clientMainAgent.sources   = s1
clientMainAgent.sinks     = k1 k2
# clientMainAgent sinks group
clientMainAgent.sinkgroups = g1
# clientMainAgent Spooling Directory Source
clientMainAgent.sources.s1.type = spooldir
clientMainAgent.sources.s1.spoolDir = /dsap/rawdata/
clientMainAgent.sources.s1.fileHeader = true
clientMainAgent.sources.s1.deletePolicy = immediate
clientMainAgent.sources.s1.batchSize = 1000
clientMainAgent.sources.s1.channels = c1
clientMainAgent.sources.s1.serializer.maxLineLength = 1048576
# clientMainAgent FileChannel
clientMainAgent.channels.c1.type = file
clientMainAgent.channels.c1.checkpointDir = /var/flume/fchannel/spool/checkpoint
clientMainAgent.channels.c1.dataDirs = /var/flume/fchannel/spool/data
clientMainAgent.channels.c1.capacity = 200000000
clientMainAgent.channels.c1.keep-alive = 30
clientMainAgent.channels.c1.write-timeout = 30
clientMainAgent.channels.c1.checkpoint-timeout = 600
# clientMainAgent Sinks
# k1 sink
clientMainAgent.sinks.k1.channel = c1
clientMainAgent.sinks.k1.type = avro
# connect to CollectorMainAgent
clientMainAgent.sinks.k1.hostname = flume115
clientMainAgent.sinks.k1.port = 41415
# k2 sink
clientMainAgent.sinks.k2.channel = c1
clientMainAgent.sinks.k2.type = avro
# connect to CollectorBackupAgent
clientMainAgent.sinks.k2.hostname = flume116
clientMainAgent.sinks.k2.port = 41415
# clientMainAgent sinks group
clientMainAgent.sinkgroups.g1.sinks = k1 k2
# load_balance type
clientMainAgent.sinkgroups.g1.processor.type = load_balance
clientMainAgent.sinkgroups.g1.processor.backoff = true
clientMainAgent.sinkgroups.g1.processor.selector = random
```

## 客户端 Agent 启动命令

```
../bin/flume-ng agent --conf ../conf/ -f flume_Consolidation.conf -n  
clientMainAgent -Dflume.root.logger=DEBUG,console
```

### ■ 在汇聚节点配置 Flume server

```
# collectorMainAgent  
collectorMainAgent.channels = c2  
collectorMainAgent.sources = s2  
collectorMainAgent.sinks = k1 k2  
# collectorMainAgent AvroSource  
collectorMainAgent.sources.s2.type = avro  
collectorMainAgent.sources.s2.bind = flume115  
collectorMainAgent.sources.s2.port = 41415  
collectorMainAgent.sources.s2.channels = c2  
# collectorMainAgent FileChannel  
collectorMainAgent.channels.c2.type = file  
collectorMainAgent.channels.c2.checkpointDir  
= /opt/var/flume/fchannel/spool/checkpoint  
collectorMainAgent.channels.c2.dataDirs =  
/opt/var/flume/fchannel/spool/data, /work/flume/fchannel/spool/data  
collectorMainAgent.channels.c2.capacity = 200000000  
collectorMainAgent.channels.c2.transactionCapacity=6000  
collectorMainAgent.channels.c2.checkpointInterval=60000  
# collectorMainAgent hdfsSink  
collectorMainAgent.sinks.k2.type = hdfs  
collectorMainAgent.sinks.k2.channel = c2  
collectorMainAgent.sinks.k2.hdfs.path = hdfs://db-cdh-cluster/flume%{dir}  
collectorMainAgent.sinks.k2.hdfs.filePrefix = k2_%{file}  
collectorMainAgent.sinks.k2.hdfs.inUsePrefix =_  
collectorMainAgent.sinks.k2.hdfs.inUseSuffix =.tmp  
collectorMainAgent.sinks.k2.hdfs.rollSize = 0  
collectorMainAgent.sinks.k2.hdfs.rollCount = 0  
collectorMainAgent.sinks.k2.hdfs.rollInterval = 240  
collectorMainAgent.sinks.k2.hdfs.writeFormat = Text  
collectorMainAgent.sinks.k2.hdfs.fileType = DataStream  
collectorMainAgent.sinks.k2.hdfs.batchSize = 6000  
collectorMainAgent.sinks.k2.hdfs.callTimeout = 60000  
collectorMainAgent.sinks.k1.type = hdfs  
collectorMainAgent.sinks.k1.channel = c2  
collectorMainAgent.sinks.k1.hdfs.path = hdfs://db-cdh-cluster/flume%{dir}  
collectorMainAgent.sinks.k1.hdfs.filePrefix = k1_%{file}  
collectorMainAgent.sinks.k1.hdfs.inUsePrefix =_  
collectorMainAgent.sinks.k1.hdfs.inUseSuffix =.tmp
```

```
collectorMainAgent.sinks.k1.hdfs.rollSize = 0
collectorMainAgent.sinks.k1.hdfs.rollCount = 0
collectorMainAgent.sinks.k1.hdfs.rollInterval = 240
collectorMainAgent.sinks.k1.hdfs.writeFormat = Text
collectorMainAgent.sinks.k1.hdfs.fileType = DataStream
collectorMainAgent.sinks.k1.hdfs.batchSize = 6000
collectorMainAgent.sinks.k1.hdfs.callTimeout = 60000
```

### 服务端启动命令

```
../bin/flume-ng agent --conf ../conf/ -f flume_Consolidation.conf -n
collectorMainAgent -Dflume.root.logger=DEBUG,console
```

上面采用的就是类似 cs 架构，各个 flume agent 节点先将各台机器的日志汇总到 Consolidation 节点，然后再由这些节点统一写入 HDFS，并且采用了负载均衡的方式，你还可以配置高可用的模式等等。

## Flume 整合 kafka

### ■ 文件拷贝

flume 和 kafka 的整合需要用到以下 jar 包，位于 kafka\_2.9.1-0.8.2.1\libs 目录下：

kafka\_2.9.1-0.8.2.1.jar

scala-library-2.9.1.jar

metrics-core-2.2.0.jar

将上述 3 个文件拷贝到 flume-1.5.0/lib 目录下

### ■ 编写 kafkasink 文件

```
import java.util.Properties;
import kafka.javaapi.producer.Producer;
import kafka.producer.KeyedMessage;
import kafka.producer.ProducerConfig;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.flume.Channel;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.EventDeliveryException;
import org.apache.flume.Transaction;
import org.apache.flume.conf.Configurable;
import org.apache.flume.sink.AbstractSink;

public class KafkaSink extends AbstractSink implements Configurable {
    private static final Log logger = LogFactory.getLog(KafkaSink.class);
    private String topic;
    private Producer<String, String> producer;
    public void configure(Context context) {
        topic = "niit_testTopic";
        Properties props = new Properties();
```

```

        props.setProperty("metadata.broker.list", "m1:9092,m2:9092,s1:9092,s2:9092");
        props.setProperty("serializer.class", "kafka.serializer.StringEncoder");
        props.put("partitioner.class", "niit.cloud.kafka.PartitionerTest");
        props.put("zookeeper.connect", "m1:2181,m2:2181,s1:2181,s2:2181/kafka");
        props.setProperty("num.partitions", "4"); //
        props.put("request.required.acks", "1");
        ProducerConfig config = new ProducerConfig(props);
        producer = new Producer<String, String>(config);
        logger.info("KafkaSink 初始化完成.");
    }

    public Status process() throws EventDeliveryException {
        Channel channel = getChannel();
        Transaction tx = channel.getTransaction();
        try {
            tx.begin();
            Event e = channel.take();
            if (e == null) {
                tx.rollback();
                return Status.BACKOFF;
            }
            KeyedMessage<String, String> data = new KeyedMessage<String,
String>(topic, new String(e.getBody()));
            producer.send(data);
            logger.info("flume 向 kafka 发送消息: " + new String(e.getBody()));
            tx.commit();
            return Status.READY;
        } catch (Exception e) {
            logger.error("Flume KafkaSinkException:", e);
            tx.rollback();
        }
    }

```

```

        return Status.BACKOFF;
    } finally {
        tx.close();
    }
}
}

```

## 配置 Agent

```

vi $FLUME_HOME/conf/kafka.conf
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# Describe/configure the source
a1.sources.r1.type = syslogtcp
a1.sources.r1.port = 5140
a1.sources.r1.host = localhost
a1.sources.r1.channels = c1
# Describe the sink
a1.sinks.k1.type = niit.cloud.flume.sink.KafkaSink
# 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

```

## 启动 Agent

```

bin/flume-ng agent -c . -f /home/hadoop/flume-1.5.0-bin/conf/kafka.conf -n a1 -
Dflume.root.logger=INFO,console

```

## ■ KafkaSpouttest

```

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import kafka.consumer.ConsumerConfig;
import kafka.consumer.ConsumerIterator;
import kafka.consumer.KafkaStream;
import kafka.javaapi.consumer.ConsumerConnector;
import backtype.storm.spout.SpoutOutputCollector;
import backtype.storm.task.TopologyContext;
import backtype.storm.topology.IRichSpout;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Values;

public class KafkaSpouttest implements IRichSpout {
    private SpoutOutputCollector collector;
    private ConsumerConnector consumer;
    private String topic;
    public KafkaSpouttest() {
    }
    public KafkaSpouttest(String topic) {
        this.topic = topic;
    }
    public void nextTuple() {
    }
    public void open(Map conf, TopologyContext context, SpoutOutputCollector
collector) {

```



```

        this.collector = collector;
    }

    public void ack(Object msgId) {
    }

    public void activate() {
consumer=kafka.consumer.Consumer.createJavaConsumerConnector(createConsumerConfig());
Map<String, Integer> topickMap = new HashMap<String, Integer>();
topickMap.put(topic, 1);
Map<String, List<KafkaStream<byte[], byte[]>>>
streamMap=consumer.createMessageStreams(topickMap);

        KafkaStream<byte[], byte[]>stream = streamMap.get(topic).get(0);
        ConsumerIterator<byte[], byte[]> it =stream.iterator();
        while(it.hasNext()){

            String value =new String(it.next().message());
SimpleDateFormat formatter = new SimpleDateFormat("yyyy 年 MM 月 dd 日 HH:mm:ss SSS");
Date curDate = new Date(System.currentTimeMillis());//获取当前时间
String str = formatter.format(curDate);
System.out.println("storm 接收到来自 kafka 的消息----->" + value);
collector.emit(new Values(value, 1, str), value);

        }
    }

    private static ConsumerConfig createConsumerConfig() {
        Properties props = new Properties();
        // 设置 zookeeper 的链接地址
        props.put("zookeeper.connect", "m1:2181,m2:2181,s1:2181,s2:2181");
        // 设置 group id
        props.put("group.id", "1");

        // kafka 的 group 消费记录是保存在 zookeeper 上的, 但这个信息在 zookeeper 上不
        是实时更新的, 需要有个间隔时间更新
    }

```

```

        props.put("auto.commit.interval.ms", "1000");
        props.put("zookeeper.session.timeout.ms", "10000");
        return new ConsumerConfig(props);
    }

    public void close() {
    }

    public void deactivate() {
    }

    public void fail(Object msgId) {
    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word", "id", "time"));
    }

    public Map<String, Object> getComponentConfiguration() {
        System.out.println("getComponentConfiguration 被调用");
        topic="niit_testTopic";
        return null;
    }
}

```

## ■ **KafkaTopologytest**

```

import java.util.HashMap;
import java.util.Map;
import backtype.storm.Config;
import backtype.storm.LocalCluster;
import backtype.storm.topology.BasicOutputCollector;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.TopologyBuilder;
import backtype.storm.topology.base.BaseBasicBolt;

```

```

import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Tuple;
import backtype.storm.tuple.Values;
import backtype.storm.utils.Utills;

public class KafkaTopologytest {
    public static void main(String[] args) {
        TopologyBuilder builder = new TopologyBuilder();
        builder.setSpout("spout", new KafkaSpouttest(""), 1);
        builder.setBolt("bolt1", new Bolt1(), 2).shuffleGrouping("spout");
        builder.setBolt("bolt2", new Bolt2(), 2).fieldsGrouping("bolt1", new Fields("word"));
        Map conf = new HashMap();
        conf.put(Config.TOPOLOGY_WORKERS, 1);
        conf.put(Config.TOPOLOGY_DEBUG, true);
        LocalCluster cluster = new LocalCluster();
        cluster.submitTopology("my-flume-kafka-storm-topology-integration", conf,
            builder.createTopology());
            Utills.sleep(1000*60*5); // local cluster test ...
            cluster.shutdown();
        }

    public static class Bolt1 extends BaseBasicBolt {
        public void execute(Tuple input, BasicOutputCollector collector) {
            try {
                String msg = input.getString(0);
                int id = input.getInteger(1);
                String time = input.getString(2);
                msg = msg+"bolt1";
                System.out.println("对消息加工第 1 次-----[arg0]:"+ msg + "---
[arg1]:"+id+"---[arg2]:"+time+"----->" +msg);
            }
        }
    }
}

```

```

        if (msg != null) {
            collector.emit(new Values(msg));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word"));
}

}

public static class Bolt2 extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<String, Integer>();
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String msg = tuple.getString(0);
        msg = msg + "bolt2";
        System.out.println("对消息加工第 2 次----->" + msg);
        collector.emit(new Values(msg, 1));
    }
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word", "count"));
    }
}

}

```

## Flume 常见问题

Redis 在很多方面与其他数据库解决方案不同：它使用内存提供主存储支持，而仅使用硬盘做持久性的存储；它的数据模型非常独特，用的是单线程。

### ■ OOM 问题

#### 问题一、flume 报错：

```
java.lang.OutOfMemoryError: GC overhead limit exceeded
```

或者：

```
java.lang.OutOfMemoryError: Java heap space
```

```
Exception in thread "SinkRunner-PollingRunner-DefaultSinkProcessor"
```

```
java.lang.OutOfMemoryError: Java heap space
```

#### 解决方案

Flume 启动时的最大堆内存大小默认是 20M，线上环境很容易 OOM，因此需要你在 flume-env.sh 中添加 JVM 启动参数

```
JAVA_OPTS="-Xms8192m -Xmx8192m -Xss256k -Xmn2g -XX:+UseParNewGC -  
XX:+UseConcMarkSweepGC -XX:-UseGCOverheadLimit"
```

然后在启动 agent 的时候一定要带上 -c conf 选项，否则 flume-env.sh 里配置的环境变量不会被加载生效

#### 问题二、小文件写入 HDFS 延时的问题

##### 解决方案：

flume 的 sink 已经实现了几种最主要的持久化触发器：

比如按大小、按间隔时间、按消息条数等等，针对你的文件过小迟迟没法写入 HDFS 持久化的问题，

如果此时还没有满足持久化的条件，比如行数还没有达到配置的阈值或者大小还没达到等等，可以调整刷新时间

```
agent1.sinks.log-sink1.hdfs.rollInterval = 20
```

##### 以下为常见的持久化触发器

```
# Number of seconds to wait before rolling current file (in 600 seconds)  
agent.sinks.sink.hdfs.rollInterval=600  
# File size to trigger roll, in bytes (256Mb)
```

```
agent.sinks.sink.hdfs.rollSize = 268435456
# never roll based on number of events
agent.sinks.sink.hdfs.rollCount = 0
# Timeout after which inactive files get closed (in seconds)
agent.sinks.sink.hdfs.idleTimeout = 3600
agent.sinks.HDFS.hdfs.batchSize = 1000
```

### 问题三、数据重复写入、丢失问题

#### 解决方案：

Flume 的 HDFSsink 在数据写入/读出 Channel 时，都有 Transaction 的保证。当 Transaction 失败时，会回滚，然后重试。但由于 HDFS 不可修改文件的内容，假设有 1 万行数据要写入 HDFS，而在写入 5000 行时，网络出现问题导致写入失败，Transaction 回滚，然后重写这 10000 条记录成功，就会导致第一次写入的 5000 行重复。这些问题是 HDFS 文件系统设计上的特性缺陷，并不能通过简单的 Bugfix 来解决。我们只能关闭批量写入，单条事务保证，或者启用监控策略，两端对数。

Memory 和 exec 的方式可能会有数据丢失，file 是 end to end 的可靠性保证的，但是性能较前两者要差。end to end、store on failure 方式 ACK 确认时间设置过短（特别是高峰时间）也有可能引发数据的重复写入。



## 活动 1.2：通过 Flume 收集日志写入 HDFS

---

## 练习问题

1、关于 redis 描述不正确的是 ( )

A、redis 是一个 key-value 存储系统

B、支持 string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和 hash(哈希类型)

C、Redis 的所有操作都是原子性的，同时 Redis 还支持对几个操作全并后的原子性执行

D、由于 redis 是单实例启动，所以只能单机部署，不支持集群模式

2、往 redis 中写入一个 key=name, value=niit 的命令是

A、put name niit

B、set name niit

C、add name niit

D、insert name niit

3、从 redis 里面取一个 key=name 的命令是：

A、get name

B、push name

C、query name

D、pop name

4、从 redis 服务器端默认监听端口是

A、3306

B、3389

C、6379

D、1521

## 小结

在本章中，您已学习了 flume 的相关概念和使用场景，如：

- ✧ Flume 简介
- ✧ flume 安装配置
- ✧ Flume 数据流模型
- ✧ Flume 数据源读取方式
- ✧ Flume 集成 kafka
- ✧ Flume 常见问题

在互联网中服务器一般都采用集群方式部署来完成，以满足高并发、高访问量的业务需求。但是问题随之而来，各业务系统记录的日志都是独立的，如果要查看各业务系统的日志必须登录每台服务器，导致复杂性增强、安全性降低。Flume 的出现刚好解决了这样的问题。



## 练习答案

---

### 第 1 章

1. D
2. D
3. D

### 第 2 章

1. C
2. D
3. C

### 第 3 章

1. C
2. B
3. A
4. C

### 第 4 章

1. D.
2. D
3. C
4. B

### 第 5 章

- 1、B
- 2、C

## 第 6 章

- 1、D
- 2、D
- 3、D
- 4、D
- 5、D
- 6、C

## 参考阅读





## Flume 简介

下表列出了《大数据生态圈工具》教程的参考资料。

编号	书名	网址
flume		<a href="http://flume.apache.org/">http://flume.apache.org/</a>

### 参考资料列表

**免责声明：**“参考阅读”部分列出的所有 URL 的准确性和适当性均在添加时经过检验。但不能保证其网站和内容长时间不变。

