

# flume介绍

## 概述

Flume最早是Cloudera提供的日志收集系统，后贡献给Apache。所以目前是Apache下的项目，Flume支持在日志系统中定制各类数据发送方，用于收集数据。

Flume是一个高可用的，高可靠的鲁棒性（robust 健壮性），**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(source);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(sink)。

当前Flume有两个版本Flume 0.9X版本的统称Flume-og，老版本的flume需要引入zookeeper集群管理，性能也较低（单线程工作）。

Flume1.X版本的统称Flume-ng。新版本需要引入zookeeper。

由于Flume-ng经过重大重构，与Flume-og有很大不同，使用时请注意区分。

## 系统需求

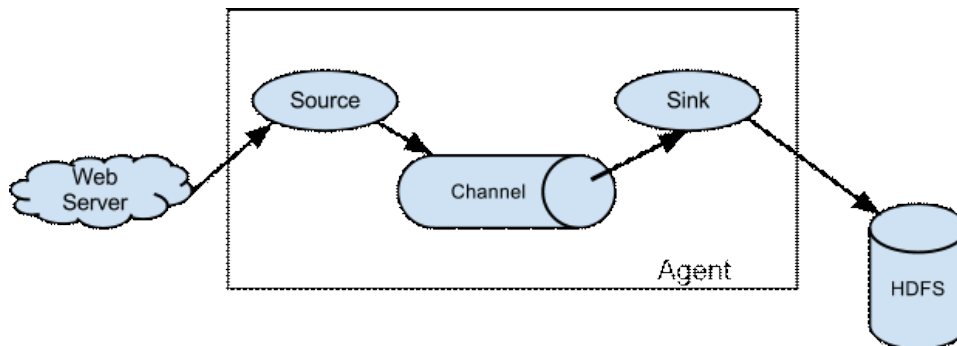
jdk1.6以上。推荐1.7或1.8

## 下载地址

[flume.apache.org](http://flume.apache.org)

# flume基本概念

## flume总体架构



flume是分布式的日志收集系统，它将各个服务器中的数据收集起来并送到指定的地方去，比如说送到图中的HDFS，简单来说flume就是收集日志的。

## event 事件

event的相关概念：flume的核心是把数据从数据源(source)收集过来，在将收集到的数据送到指定的目的地(sink)。为了保证输送的过程一定成功，在送到目的地(sink)之前，会先缓存数据(channel),待数据真正到达目的地(sink)后，flume在删除自己缓存的数据。

在整个数据的传输的过程中，流动的是event，即事务保证是在event级别进行的。那么什么是event呢？——event将传输的数据进行封装，是flume传输数据的基本单位，如果是文本文件，通常是一行记录，event也是事务的基本单位。event从source，流向channel，再到sink，本身为一个字节数组，并可携带headers(头信息)信息。event代表着一个数据的最小完整单元，从外部数据源来，向外部的目的地去。

一个完整的event包括：event headers、event body、event信息(即文本文件中的单行记

录)，如下所以：

```
2016-05-29 21:51:13,820 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.processor(LoggerSink.java:94)] Event: { headers:{file=/usr/local/datainput/file10} body: 7A 68 61 6E 67 6D 69 6E 67 zhangming }
```

其中event信息就是flume收集到的日记记录。

## flume的运行机制

flume运行的核心就是agent，agent本身是一个Java进程，

agent里面包含3个核心的组件：source—>channel—>sink, 类似生产者、仓库、消费者的架构。

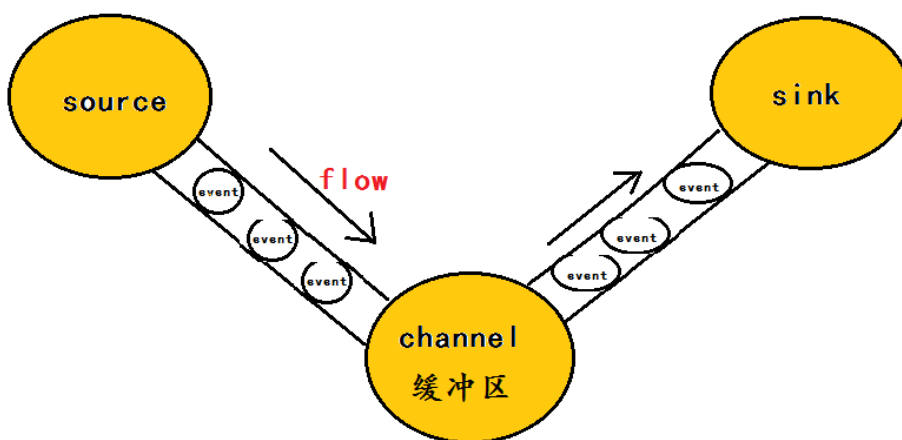
source：source组件是专门用来收集数据的，可以处理各种类型、各种格式的日志数据, 包括 avro、thrift、exec、jms、spooling directory、netcat、sequence generator、syslog、http、legacy、自定义。

channel：source组件把数据收集来以后，临时存放在channel中，即channel组件在agent中是专门用来存放临时数据的——对采集到的数据进行简单的缓存，可以存放在 memory、jdbc、file等等。

sink：sink组件是用于把数据发送到目的地的组件，目的地包括 hdfs、logger、avro、thrift、ipc、file、null、hbase、solr、自定义。

一个完整的工作流程：source不断的接收数据，将数据封装成一个一个的event，然后将event发送给channel，channel作为一个缓冲区会临时存放这些event数据，随后sink会将channel中的event数据发送到指定的地方——例如HDFS等。

注：只有在sink将channel中的数据成功发送出去之后，channel才会将临时event数据进行删除，这种机制保证了数据传输的可靠性与安全性。



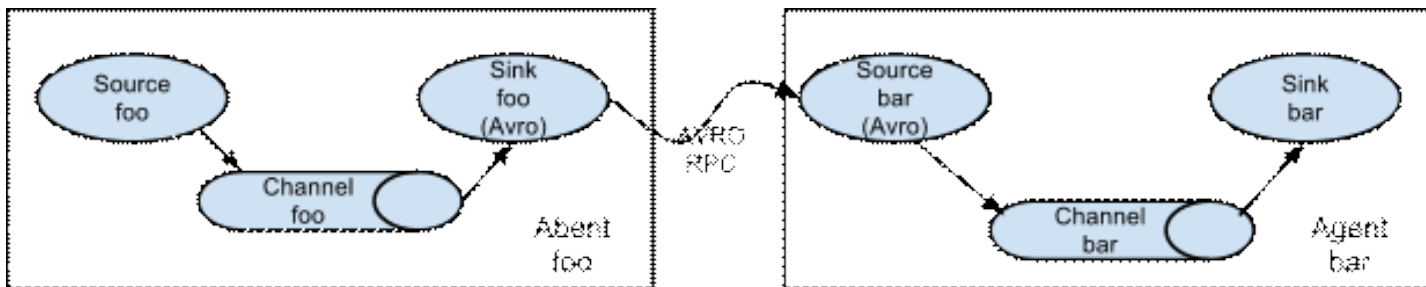
## flume的用法

flume之所以这么神奇——其原因也在于flume可以支持多级flume的agent，即flume可以前后相

继形成多级的复杂流动，例如sink可以将数据写到下一个agent的source中，这样的话就可以连成串了，可以整体处理了。

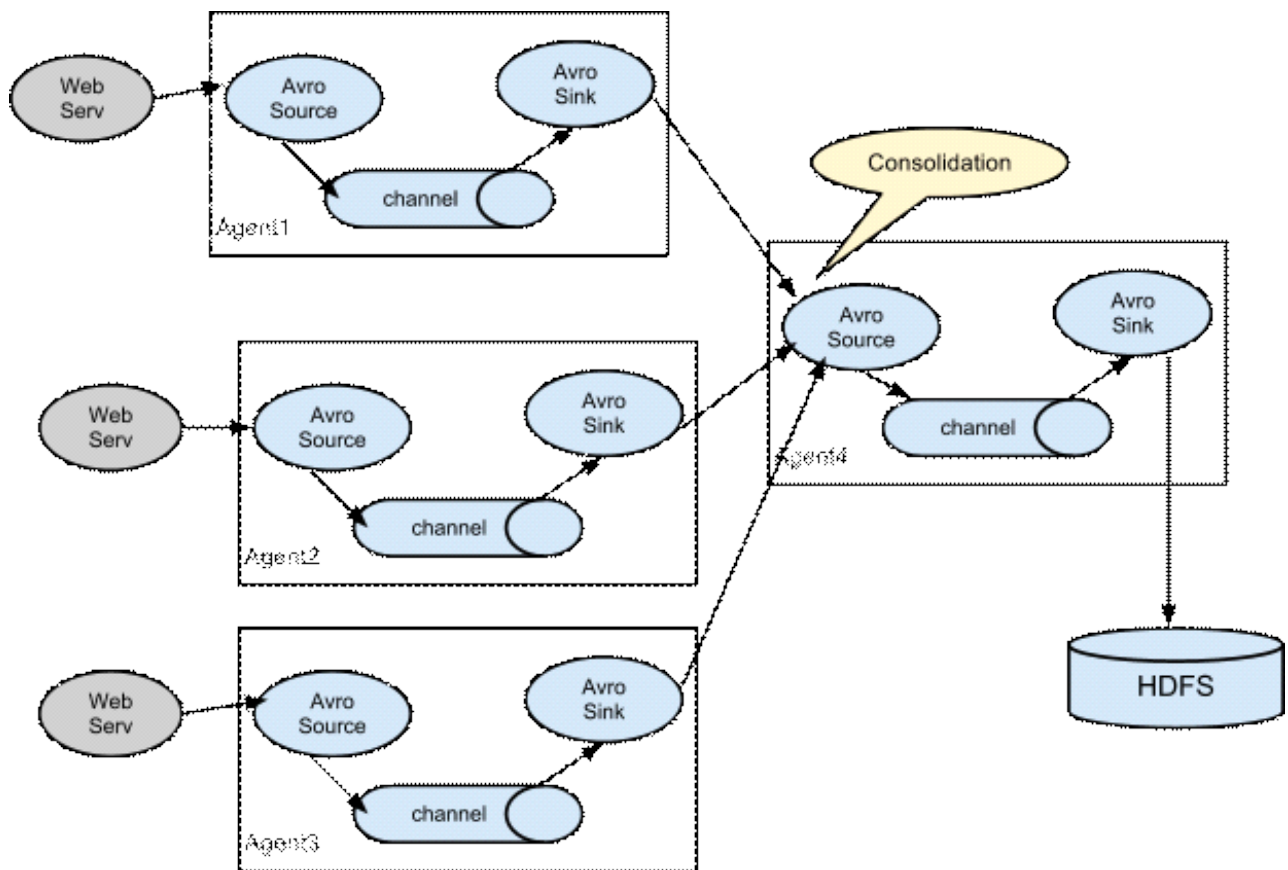
此外，flume还支持扇入(fan-in)、扇出(fan-out)。所谓扇入就是source可以接受多个输入，所谓扇出就是sink可以将数据输出多个目的地中。

### 置多个agent的数据流（多级流动）



### 数据流合并（扇入流）

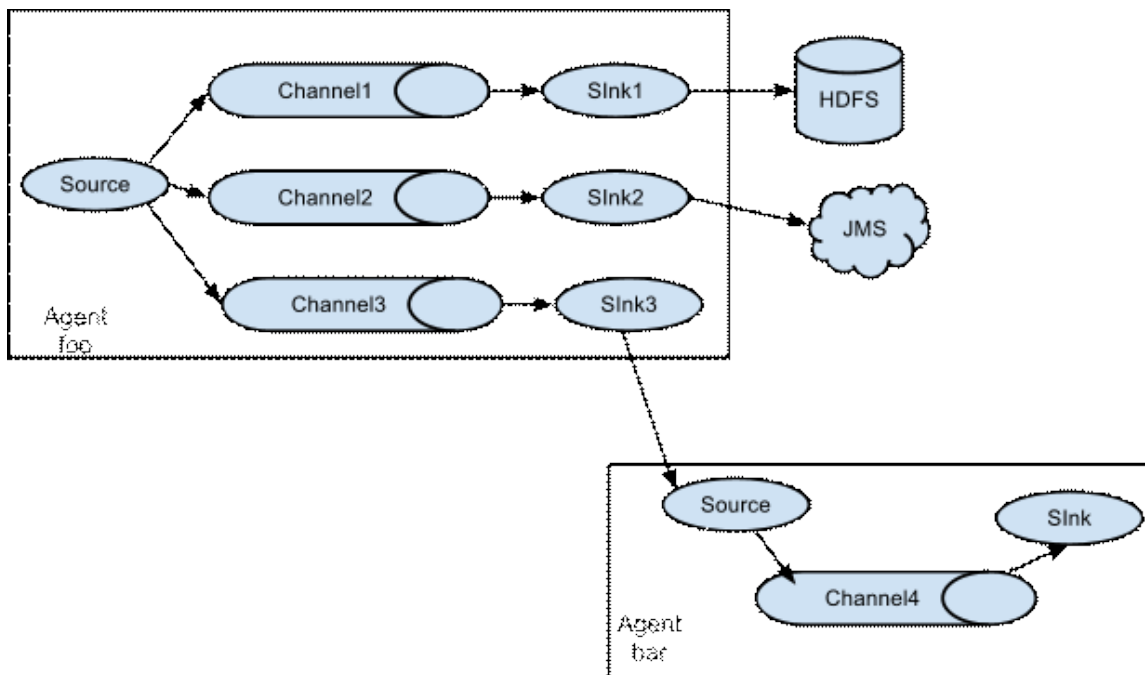
在做日志收集的时候一个常见的场景就是，大量的生产日志的客户端发送数据到少量的附属于存储子系统的消费者agent。例如，从数百个web服务器中收集日志，它们发送数据到十几个负责将数据写入HDFS集群的agent。



这个可在Flume中可以实现，需要配置大量第一层的agent，每一个agent都有一个avro sink，让它们都指向同一个agent的avro source（强调一下，在这样一个场景下你也可以使用thrift source/sink/client）。在第二层agent上的source将收到的event合并到一个channel中，event被一个sink消费到它的最终的目的地。

### 数据流复用（扇出流）

Flume支持多路输出event流到一个或多个目的地。这是靠定义一个多路数据流实现的，它可以实现复制和选择性路由一个event到一个或者多个channel。



上面的例子展示了agent foo中source扇出数据流到三个不同的channel，这个扇出可以是复制或者多路输出。在复制数据流的情况下，每一个event被发送所有的三个channel；在多路输出的情况下，一个event被发送到一部分可用的channel中，它们是根据event的属性和预先配置的值选择channel的。这些映射关系应该被填写在agent的配置文件中。

## Flume的特性

### 可靠性

事务型的数据传递，保证数据的可靠性

一个日志交给flume来处理，不会出现此日志丢失或未被处理的情况。

### 可恢复性

通道可以以内存或文件的方式实现，内存更快，但不可恢复。文件较慢但提供了可恢复性。

# flume安装和配置

## 实现步骤：

- ☐ 1.安装jdk，1.6版本以上
- ☐ 2.上传flume的安装包
- ☐ 3.解压安装
- ☐ 4.在conf目录下，创建一个配置文件，比如：template.conf（名字可以不固定,后缀也可以不固定）
- ☐ 5.配置agent组件

## 📖 相关配置：

#配置Agent a1 的组件

a1.sources=r1

a1.channels=c1 (可以配置多个，以空格隔开，名字自己定)

a1.sinks=s1 (可以配置多个，以空格隔开，名字自己定)

#描述/配置a1的r1

a1.sources.r1.type=netcat (netcat表示通过指定端口来访问)

a1.sources.r1.bind=0.0.0.0 (表示本机)

a1.sources.r1.port=44444 (指定的端口，此端口不固定，但是不要起冲突)

#描述a1的s1

a1.sinks.s1.type=logger (表示数据汇聚点的类型是logger日志)

#描述a1的c1

a1.channels.c1.type=memory

a1.channels.c1.capacity=1000

a1.channels.c1.transactionCapacity=100

#位channel 绑定 source和sink

a1.sources.r1.channels=c1 (一个source是可以对应多个通道的)

a1.sinks.s1.channel=c1 (一个sink,只能对应一个通道)

- ☐ 6.根据指定的配置文件，来启动flume

进入flume的bin目录

执行：./flume-ng agent -n a1 -c ../conf -f ../conf/template.conf -

Dflume.root.logger=INFO,console

```

HANNEL, name: channel1 started
2016-09-29 22:40:04,741 (conf-file-poller-0) [INFO - org.apache.flume.node.Applicati
on.startAllComponents(Application.java:173)] Starting Sink sink1
2016-09-29 22:40:04,742 (conf-file-poller-0) [INFO - org.apache.flume.node.Applicati
on.startAllComponents(Application.java:184)] Starting Source source1
2016-09-29 22:40:04,756 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.source.Ne
tcatSource.start(NetcatSource.java:150)] Source starting
2016-09-29 22:40:04,835 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.source.Ne
tcatSource.start(NetcatSource.java:164)] Created serverSocket:sun.nio.ch.ServerSocke
tChannelImpl[/0:0:0:0:0:0:0:0:44444]

```

如果出现上图所示，证明配置和启动成功

#### 7.通过nc来访问：

nc localhost 44444

hello flume

```

[root@localhost software]# nc localhost 8888
hello flume
OK

```

或者：

通过外部http请求访问对应的ip和端口

比如：http://192.168.234.163:44444/hello

在虚拟机这边，会出现如下提示：

```

2016-09-29 22:46:34,869 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.
apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{} body:
47 45 54 20 2F 68 65 6C 6C 6F 20 48 54 54 50 2F GET /hello HTTP/ }
2016-09-29 22:46:34,870 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.
apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{} body:
41 63 63 65 70 74 3A 20 2A 2F 2A 0D Accept: */*. }
2016-09-29 22:46:34,870 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.
apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{} body:
41 63 63 65 70 74 2D 4C 61 6E 67 75 61 67 65 3A Accept-Language: }
2016-09-29 22:46:34,870 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.

```

#### 启动命令解释

参数	描述
agent	运行一个Flume Agent
--conf, -c <conf>	指定配置文件放在什么目录
--conf-file, -f <file>	指定配置文件，这个配置文件必须在全局选项的--conf参数定义的目录下
--name, -n <name>	Agent的名称，注意：要和配置文件里的名字一致。
-Dproperty=value	设置一个JAVA系统属性值。常见的：-Dflume.root.logger=INFO,console



# flume的Source

source学习网址：  
<http://flume.apache.org/FlumeUserGuide.html>

## 一、Avro 类型的Source

监听Avro 端口来接收外部avro客户端的事件流。和netcat不同的是，avro-source接收到的是经过avro序列化后的数据，然后反序列化数据继续传输。所以，如果是avro-source的话，源数据必须是经过avro序列化后的数据。而netcat接收的是字符串格式。

**利用Avro source可以实现多级流动、扇出流、扇入流等效果。**  
另外，也可以接收通过flume提供的avro客户端发送的日志信息。

### Avro Source可配选项说明

配置项	说明
channels	指定有几个通道并为每个通道起个名字
type	如果是Avro类型，则type属性为AVRO
bind	需要监听的主机名或IP
port	要监听的端口
threads	工作线程最大线程数
selector.type	
selector.*	
interceptors	空格分隔的拦截器列表
interceptors.*	

- 实现步骤：
- 1.修改配置文件，source的type属性为avro
  - 2.根据指定的配置文件启动flume

 **格式代码，template-avro.conf为例子：**

```
#配置Agent a1 的组件
a1.sources=r1
a1.channels=c1
a1.sinks=s1

#描述/配置a1的source1
a1.sources.r1.type=avro
a1.sources.r1.bind=0.0.0.0
a1.sources.r1.port=44444

#描述sink
```

al.sinks.sl.type=logger

#描述内存channel

al.channels.cl.type=memory  
al.channels.cl.capacity=1000  
al.channels.cl.transactionCapacity=100

#位channel 绑定 source和sink

al.sources.rl.channels=c1  
al.sinks.sl.channel=c1

3.执行启动指令，如果出现如下提示，证明启动成功

```
resource.start(AvroSource.java:226)] Starting Avro source source1: { bindAddress: 0.0.0.0, port: 44445 }...  
2016-09-30 10:32:13,543 (lifecycleSupervisor-1-4) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.register(MonitoredCounterGroup.java:120)] Monitored counter group for type: SOURCE, name: source1: Successfully registered new MBean.  
2016-09-30 10:32:13,543 (lifecycleSupervisor-1-4) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.start(MonitoredCounterGroup.java:96)] Component type: SOURCE, name: source1 started  
2016-09-30 10:32:13,544 (lifecycleSupervisor-1-4) [INFO - org.apache.flume.source.AvroSource.start(AvroSource.java:253)] Avro source source1 started.
```

4.执行agent-avro客户端指令：

./flume-ng avro-client -H 0.0.0.0 -p 44444 -F ../mydata/1.txt -c ../conf/

（注：可以通过执行：./flume-ng help 来查看指令）

如果出现：

```
1:60162 disconnected.  
2016-09-30 10:52:21,928 (New I/O server boss #1 ([id: 0xc5ea8fcd, /0:0:0:0:0:0:0:4445])) [INFO - org.apache.avro.ipc.NettyServer$NettyServerAvroHandler.handleUpstream(NettyServer.java:171)] [id: 0x558b4e79, /127.0.0.1:60164 => /127.0.0.1:44445] OPEN  
2016-09-30 10:52:21,928 (New I/O worker #3) [INFO - org.apache.avro.ipc.NettyServer$NettyServerAvroHandler.handleUpstream(NettyServer.java:171)] [id: 0x558b4e79, /127.0.0.1:60164 => /127.0.0.1:44445] BOUND: /127.0.0.1:44445  
2016-09-30 10:52:21,928 (New I/O worker #3) [INFO - org.apache.avro.ipc.NettyServer$NettyServerAvroHandler.handleUpstream(NettyServer.java:171)] [id: 0x558b4e79, /127.0.0.1:60164 => /127.0.0.1:44445] CONNECTED: /127.0.0.1:60164  
2016-09-30 10:52:22,262 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{} body: 68 65 6C 6C 6F 20 66 6C 75 6D 65 hello flume }  
2016-09-30 10:52:22,408 (New I/O worker #3) [INFO - org.apache.avro.ipc.NettyServer$NettyServerAvroHandler.handleUpstream(NettyServer.java:171)] [id: 0x558b4e79, /127.0.0.1:60164 :> /127.0.0.1:44445] DISCONNECTED
```

二、Exec类型的Source

可以将命令产生的输出作为源

Exec Source 可配置选项说明

配置项	说明
channels	
type	类型名称，需要是“exec”
command	要执行的命令
selector.type	复制还是多路复用
selector.*	Depends on the selector.type value

interceptors	空格分隔的拦截器列表
interceptors.*	

### 实现步骤：

1.修改配置文件，source的type属性为avro

#### 配置代码：

#配置Agent al 的组件

al.sources=r1

al.channels=c1

al.sinks=s1

#描述/配置al的source1

**al.sources.r1.type=exec**

**al.sources.r1.command=ping 192.168.234.163**

#描述sink

al.sinks.s1.type=logger

#描述内存channel

al.channels.c1.type=memory

al.channels.c1.capacity=1000

al.channels.c1.transactionCapacity=100

#位channel 绑定 source和sink

al.sources.r1.channels=c1

al.sinks.s1.channel=c1

2.进入到bin目录执行指令：

```
./flume-ng agent --conf ../conf --conf-file ../conf/template-exec.conf --name al -
Dflume.root.logger=INFO,console
```

```
2016-01-29 09:51:52,728 (SinkRunner-PollingRunner-DefaultSinkProce
31 39 64 bytes from 19 }
2016-01-29 09:51:52,728 (SinkRunner-PollingRunner-DefaultSinkProce
31 39 64 bytes from 19 }
2016-01-29 09:51:55,728 (SinkRunner-PollingRunner-DefaultSinkProce
31 39 64 bytes from 19 }
2016-01-29 09:51:55,729 (SinkRunner-PollingRunner-DefaultSinkProce
31 39 64 bytes from 19 }
2016-01-29 09:51:55,729 (SinkRunner-PollingRunner-DefaultSinkProce
31 39 64 bytes from 19 }
```

结果为ping 当前主机返回的结果

( 也可以尝试其他的linux指令，比如 cat 1.txt 查看文件内容 )

### 三、Spooling Directory类型的 Source

将指定的文件加入到“自动搜集”目录中。flume会持续监听这个目录，把文件当做source来处理。注意：一旦文件被放到“自动收集”目录中后，便不能修改，如果修改，flume会报错。此外，也不能有重名的文件，如果有，flume也会报错。

配置项	说明
-----	----

channels	
type	类型，需要指定为“spoolDir”
spoolDir	读取文件的路径，即“搜集目录”
selector.type	replicating replicating or multiplexing
selector.*	Depends on the selector.type value
interceptors	Space-separated list of interceptors
interceptors.*	

## 实现步骤：

### 1.配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.channels=c1

a1.sinks=s1

#描述/配置a1的source1

a1.sources.r1.type=spoolDir

a1.sources.r1.spoolDir=/home/work/data

#描述sink

a1.sinks.s1.type=logger

#描述内存channel

a1.channels.c1.type=memory

a1.channels.c1.capacity=1000

a1.channels.c1.transactionCapacity=100

#位channel 绑定 source和sink

a1.sources.r1.channels=c1

a1.sinks.s1.channel=c1

### 2.创建相关的文件夹

### 3.根据指定的配置文件，启动flume

出现下图，证明flume正在监听指定的文件目录：

```

2016-09-30 14:56:26,206 (conf-file-poller-0) [INFO - org.apache.flume.node.Applicati
on.startAllComponents(Application.java:173)] Starting Sink sink1
2016-09-30 14:56:26,207 (conf-file-poller-0) [INFO - org.apache.flume.node.Applicati
on.startAllComponents(Application.java:184)] Starting Source source1
2016-09-30 14:56:26,210 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.source.Sp
oolDirectorySource.start(SpoolDirectorySource.java:78)] SpoolDirectorySource source
starting with directory: /home/work/data
2016-09-30 14:56:26,344 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumen
tation.MonitoredCounterGroup.register(MonitoredCounterGroup.java:120)] Monitored cou
nter group for type: SOURCE, name: source1: Successfully registered new MBean.
2016-09-30 14:56:26,344 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumen
tation.MonitoredCounterGroup.start(MonitoredCounterGroup.java:96)] Component type: S
OURCE, name: source1 started

```

### 4.向指定的文件目录下传送一个日志文件，发现flume的控制台打印相关的信息

此外，会发现被处理的文件，会追加一个后缀：completed，表示已处理完。  
(重名文件包括已加后缀的文件)

四、NetCat Source

一个NetCat Source用来监听一个指定端口，并接收监听到的数据。

配置项	说明
channels	
type	类型名称，需要被设置为“netcat”
port	指定要绑定到的端口号
selector.type	
selector.*	
interceptors	
interceptors.*	

配置示例：

```
#配置Agent a1 的组件
a1.sources=r1
a1.channels=c1
a1.sinks=s1

#描述/配置a1的r1
a1.sources.r1.type=netcat
a1.sources.r1.bind=0.0.0.0
a1.sources.r1.port=44444

#描述a1的s1
a1.sinks.s1.type=logger

#描述a1的c1
a1.channels.c1.type=memory
a1.channels.c1.capacity=1000
a1.channels.c1.transactionCapacity=100

#位channel 绑定 source和sink
a1.sources.r1.channels=c1
a1.sinks.s1.channel=c1
```

五、Sequence Generator Source --序列发生源

一个简单的序列发生器，不断的产生事件，值是从0开始每次递增1。主要用来测试。

可配置选项

配置项	说明
-----	----

channels	
type	seq
selector.type	
selector.*	
interceptors	
interceptors.*	
batchSize	1 递增步长，默认是1

## 配置示例

#配置Agent al 的组件

al.sources=r1

al.sinks=s1

al.channels=c1

#描述/配置al的source1

al.sources.r1.type=seq

#描述sink

al.sinks.s1.type=logger

#描述内存channel

al.channels.c1.type=memory

al.channels.c1.capacity=1000

al.channels.c1.transactionCapacity=100

#位channel 绑定 source和sink

al.sources.r1.channels=c1

al.sinks.s1.channel=c1

## 六、HTTP source

此Source接受HTTP的GET和POST请求作为Flume的事件。其中GET方式应该只用于试验。

如果想让flume正确解析Http协议信息，比如解析出请求头、请求体等信息，需要提供一个可插拔的“处理器”来将请求转换为事件对象，这个处理器必须实现HTTPSourceHandler接口。

这个处理器接受一个 HttpServletRequest对象，并返回一个Flume Event对象集合。

flume提供了一些常用的Handler（处理器）：

### JSONHandler

可以处理JSON格式的数据，并支持UTF-8 UTF-16 UTF-32字符集

该handler接受Event数组，并根据请求头中指定的编码将其转换为Flume Event

如果没有指定编码，默认编码为UTF-8.

JSON格式如下：

```
[{
  "headers": {
    "timestamp": "434324343",
    "host": "random_host.example.com"
  },
  "body": "random_body"
},
{
  "headers": {
    "namenode": "namenode.example.com",
    "datanode": "random_datanode.example.com"
  },
  "body": "really_random_body"
}]
```

## BlobHandler

BlobHandler是一种将请求中上传文件信息转化为event的处理器。

参数说明，加！为必须属性：

！ handler      –      The FQCN of this class: org.apache.flume.sink.solr.morphline.BlobHandler  
handler.maxBlobLength    100000000    The maximum number of bytes to read and buffer for a given request

配置项	说明
channels	
type	HTTP
selector.type	
selector.*	
interceptors	
interceptors.*	

## 配置示例：

#配置Agent a1 的组件

```
a1.sources=r1
a1.sinks=s1
a1.channels=c1
```

#描述/配置a1的source1

```
a1.sources.r1.type=http
a1.sources.r1.port=8888
```

#描述sink

```
a1.sinks.s1.type=logger
```

#描述内存channel

```
al.channels.cl.type=memory  
al.channels.cl.capacity=1000  
al.channels.cl.transactionCapacity=100
```

#位channel 绑定 source和sink

```
al.sources.r1.channels=c1  
al.sinks.s1.channel=c1
```

## 2.执行启动命令

## 3.执行curl 命令，模拟一次http的Post请求

```
curl -X POST -d '[{"headers":{"a":"a1","b":"b1"},"body":"hello http-flume"}]' http://0.0.0.0:8888
```

注：这种格式是固定的，因为我们用的是flume自身提供的Json格式的Handler。此外，需要包含header 和body两关键字，这样，handler在解析时才能拿到对应的数据



# flume的Sink

## 一、Logger Sink

记录指定级别（比如INFO，DEBUG，ERROR等）的日志，通常用于调试

要求，在 --conf (-c) 参数指定的目录下有log4j的配置文件

根据设计，logger sink将体内容限制为16字节，从而避免屏幕充斥着过多的内容。如果想要查看调试的完整内容，那么你应该使用其他的sink，也许可以使用file\_roll sink，它会将日志写到本地文件系统中。

### 可配置项说明

配置项	说明
channel	
type	logger
补充说明	要求必须在 --conf 参数指定的目录下有 log4j的配置文件 可以通过-Dflume.root.logger=INFO,console在命令启动时手动指定log4j参数

### 配置示例：

#配置Agent a1 的组件

```
a1.sources=r1
a1.channels=c1
a1.sinks=s1
```

#描述/配置a1的r1

```
a1.sources.r1.type=netcat
a1.sources.r1.bind=0.0.0.0
a1.sources.r1.port=44444
```

#描述a1的s1

```
a1.sinks.s1.type=logger
```

#描述a1的c1

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

#位channel 绑定 source和sink

```
a1.sources.r1.channels=c1
a1.sinks.s1.channel=c1
```

## 二、File Roll Sink

在本地系统中存储事件。

每隔指定时长生成文件保存这段时间内收集到的日志信息。

可配置参数说明

配置项	说明
channel	
type	file_roll
sink.directory	文件被存储的目录
sink.rollInterval	30 记录日志到文件里，每隔30秒生成一个新日志文件。如果设置为0，则禁止滚动，从而导致所有数据被写入到一个文件中。

 配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1

a1.channels=c1

#描述/配置a1的source1

a1.sources.r1.type=netcat

a1.sources.r1.bind=0.0.0.0

a1.sources.r1.port=8888

#描述sink

a1.sinks.s1.type=file\_roll

a1.sinks.s1.sink.directory=/home/work/rolldata

a1.sinks.s1.sink.rollInterval=60

#描述内存channel

a1.channels.c1.type=memory

a1.channels.c1.capacity=1000

a1.channels.c1.transactionCapacity=100

#位channel 绑定 source和sink

a1.sources.r1.channels=c1

a1.sinks.s1.channel=c1

☐ 创建指定的文件目录 /home/work/rolldata

☐ 启动测试

../bin/flume-ng agent -c ./ -f ./template.conf -n a1

## 三、Avro Sink

是实现多级流动、扇出流(1到多) 扇入流(多到1) 的基础。

#### 可配置项说明

配置项	说明
channel	
type	avro
hostname	The hostname or IP address to bind to
port	The port # to listen on

#### 3.1 多级流动案例需求说明：

让01机的flume通过netcat source源接收数据，然后通过avro sink 发给02机==》02机的flume利用avro source源收数据，然后通过avro sink 传给03机==》03机通过avro source源收数据，通过logger sink 输出到控制台上

( 本例中，02机的ip:192.168.234.212 || 03机的ip:192.168.234.213 )

#### 实现步骤：

- 1.准备三台虚拟机，并安装好flume（关闭每台机器的防火墙）
- 2.配置每台flume的配置文件
- 3.启动测试

#### 01机的配置示例：

#配置Agent a1 的组件

```
a1.sources=r1
a1.sinks=s1
a1.channels=c1
```

#描述/配置a1的source

```
a1.sources.r1.type=netcat
a1.sources.r1.bind=0.0.0.0
a1.sources.r1.port=8888
```

#描述sink

```
a1.sinks.s1.type=avro
a1.sinks.s1.hostname=192.168.234.212
a1.sinks.s1.port=9999
```

#描述内存channel

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

#位channel 绑定 source和sink

```
a1.sources.r1.channels=c1
a1.sinks.s1.channel=c1
```

## 02机的配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1

a1.channels=c1

#描述/配置a1的source

a1.sources.r1.type=avro

a1.sources.r1.bind=0.0.0.0

a1.sources.r1.port=9999

#描述sink

a1.sinks.s1.type=avro

a1.sinks.s1.hostname=192.168.234.213

a1.sinks.s1.port=9999

#描述内存channel

a1.channels.c1.type=memory

a1.channels.c1.capacity=1000

a1.channels.c1.transactionCapacity=100

#位channel 绑定 source和sink

a1.sources.r1.channels=c1

a1.sinks.s1.channel=c1

## 03机的配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1

a1.channels=c1

#描述/配置a1的source1

a1.sources.r1.type=avro

a1.sources.r1.bind=0.0.0.0

a1.sources.r1.port=9999

#描述sink

a1.sinks.s1.type=logger

#描述内存channel

a1.channels.c1.type=memory

a1.channels.c1.capacity=1000

a1.channels.c1.transactionCapacity=100

#位channel 绑定 source和sink

a1.sources.r1.channels=c1

```
a1.sinks.s1.channel=c1
```

### 3.2扇出流案例需求说明

01机发出的数据，让02，03来接收

#### 实现步骤：

- 1.准备三台虚拟机，并安装好flume（关闭每台机器的防火墙）
- 2.配置每台flume的配置文件
- 3.启动测试

#### 01机的配置文件

#配置Agent a1 的组件

```
a1.sources=r1
```

```
a1.sinks=s1 s2
```

```
a1.channels=c1 c2
```

#描述/配置a1的source1

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

```
a1.sources.r1.bind=0.0.0.0
```

```
a1.sources.r1.port=8888
```

#描述sink

```
a1.sinks.s1.type=avro
```

```
a1.sinks.s1.hostname=192.168.234.212
```

```
a1.sinks.s1.port=9999
```

```
a1.sinks.s2.type=avro
```

```
a1.sinks.s2.hostname=192.168.234.213
```

```
a1.sinks.s2.port=9999
```

#描述内存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
```

#位channel 绑定 source和sink

```
a1.sources.r1.channels=c1 c2
```

```
a1.sinks.s1.channel=c1
```

```
a1.sinks.s2.channel=c2
```

 02,03配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1

a1.channels=c1

#描述/配置a1的source1

a1.sources.r1.type=avro

a1.sources.r1.bind=0.0.0.0

a1.sources.r1.port=9999

#描述sink

a1.sinks.s1.type=logger

#描述内存channel

a1.channels.c1.type=memory

a1.channels.c1.capacity=1000

a1.channels.c1.transactionCapacity=100


#位channel 绑定 source和sink

a1.sources.r1.channels=c1

a1.sinks.s1.channel=c1

### 3.3 扇入案例需求说明

02,03机收到的数据都发往01

 02,03的配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1

a1.channels=c1

#描述/配置a1的source1

a1.sources.r1.type=netcat

a1.sources.r1.bind=0.0.0.0

a1.sources.r1.port=8888

#描述sink

a1.sinks.s1.type=avro

a1.sinks.s1.hostname=192.168.234.163

a1.sinks.s1.port=9999

#描述内存channel

a1.channels.c1.type=memory

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

```
#位channel 绑定 source和sink
a1.sources.r1.channels=c1
a1.sinks.s1.channel=c1
```

#### 01机的配置示例：

#配置Agent a1 的组件

```
a1.sources=r1
a1.sinks=s1
a1.channels=c1
```

#描述/配置a1的source1

```
a1.sources.r1.type=avro
a1.sources.r1.bind=0.0.0.0
a1.sources.r1.port=9999
```

#描述sink

```
a1.sinks.s1.type=logger
```

#描述内存channel

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

#位channel 绑定 source和sink

```
a1.sources.r1.channels=c1
a1.sinks.s1.channel=c1
```

## 四、HDFS Sink

此Sink将事件写入到Hadoop分布式文件系统HDFS中。

目前它支持创建文本文件和序列化文件。

对这两种格式都支持压缩。

这些文件可以分卷，按照指定的时间或数据量或事件的数量为基础。

它还通过类似时间戳或机器属性对数据进行 buckets/partitions 操作 It also buckets/partitions data by attributes like timestamp or machine where the event originated.

HDFS的目录路径可以包含将要由HDFS替换格式的转移序列用以生成存储事件的目录/文件名。

使用这个Sink要求hadoop必须已经安装好，以便Flume可以通过hadoop提供的jar包与HDFS进行通信。

### 可配置项说明

配置项	说明
channel	

type	hdfs
hdfs.path	HDFS 目录路径 ( hdfs://namenode/flume/webdata/)
hdfs.inUseSuffix	.tmp Flume正在处理的文件所加的后缀
hdfs.rollInterval	30 Number of seconds to wait before 举例：如果flume需要40s，30s=>1个文件 10s=>30s 1个文件
hdfs.rollSize	1024 File size to trigger roll, in bytes (0: never roll based on file size)
hdfs.rollCount	10 Number of events written to file before it rolled (0 = never roll based on number of events)
hdfs.fileType	SequenceFile File format: currently SequenceFile, DataStream or CompressedStream
hdfs.retryInterval	80 Time in seconds between consecutive attempts to close a file. Each close call costs multiple RPC round-trips to the Namenode, so setting this too low can cause a lot of load on the name node. If set to 0 or less, the sink will not attempt to close the file if the first attempt fails, and may leave the file open or with a ".tmp" extension.

## 配置文件

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1

a1.channels=c1

#描述/配置a1的source1

a1.sources.r1.type=netcat

a1.sources.r1.bind=0.0.0.0

a1.sources.r1.port=8888

#描述sink

a1.sinks.s1.type=hdfs

a1.sinks.s1.hdfs.path=hdfs://192.168.234.21:9000/flume

a1.sinks.s1.hdfs.fileType=DataStream

#描述内存channel

a1.channels.c1.type=memory

a1.channels.c1.capacity=1000

a1.channels.c1.transactionCapacity=100



#位channel 绑定 source和sink

a1.sources.r1.channels=c1

a1.sinks.s1.channel=c1

报错是因为flume缺少相关hadoop的依赖jar包，找到以下的jar包，放到flume的lib目录下即可。

commons-configuration-1.6.jar

hadoop-auth-2.5.2.jar

hadoop-common-2.5.2.jar

hadoop-hdfs-2.5.2.jar

hadoop-mapreduce-client-core-2.5.2.jar

但是一个一个找特别麻烦，所以解决办法是将hadoop的jar包都拷贝到flume的lib目录下：

进入到hadoop安装目录的share目录下的hadoop目录

执行：scp common/\* common/lib/\* hdfs/\* hdfs/lib/\* mapreduce/\* mapreduce/lib/\*  
tools/lib/\* 192.168.234.163:/home/software/flume/lib/

# flume的Channel

## 一、Memory Channel

事件将被存储在内存中（指定大小的队列里）

**非常适合那些需要高吞吐量且允许数据丢失的场景下**

属性说明：

配置项	说明
type	memory
capacity	100 事件存储在信道中的最大数量 the maximum number of events stored in the channel <b>建议实际工作调节：10万</b> <b>首先估算出每个event的大小，然后再服务的内存来调节</b>
transactionCapacity	100 每个事务中的最大事件数 the maximum number of events the channel will take from a source or give to a sink per transaction <b>建议实际工作调节：1000~3000</b>

配置示例略

## 二、JDBC Channel

事件会被持久化（存储）到可靠的数据库里，目前支持嵌入式Derby数据库。即source=》channel=》sink。在传输的过程中，会先把事件存到关系型数据库里。但是Derby数据库不太好用，所以JDBC Channel目前仅用于测试，不能用于生产环境。

## 三、FileChannel

性能比较低，但是即使程序出错数据不会丢失

性能会比较低下，但是即使程序出错数据不会丢失

配置项	说明
type	file
dataDirs	~/flume/file-channel/data逗号分隔的目录列表，用以存放日志文件。使用单独的磁盘上的多个目录可以提高文件通道效率。

### 配置示例:

```
a1.sources=r1
a1.channels=c1
a1.sinks=s1
```

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

```
a1.sources.r1.bind=0.0.0.0  
a1.sources.r1.port=8888
```

```
a1.sinks.s1.type=logger
```

```
a1.channels.c1.type=file  
a1.channels.c1.dataDirs=/home/filechannel
```

```
a1.sources.r1.channels=c1  
a1.sinks.s1.channel=c1
```

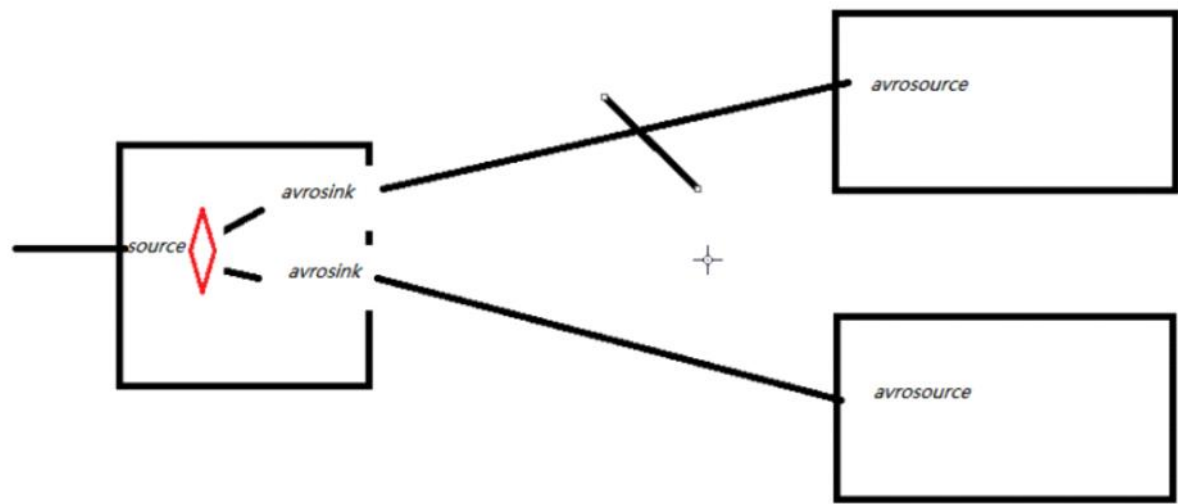
#### 四、内存溢出通道

优先把Event存到内存中，如果存不下，在溢出到文件中，目前处于测试阶段，还未能用于生产环境

# flume的Selector

## 一、Selector的复制模式

选择器可以工作在**复制**、多路复用(**路由**) 模式下  
Selector 默认是复制模式(replicating) , 即把source复制 , 然后分发给多个sink



针对每个Source , 都有一个配置参数 , 即selector.type

配置项	说明
selector.type	replicating 表示复制模式 , source的selector如果不配置 , 默认就是这种模式 针对上图 , 在复制模式下 , 当source接收到数据后 , 会复制成两份 , 分别发给两个avro sink
selector.optional	标志通道为可选

📖 配置示例 :

```
al.sources = r1
al.channels = c1 c2 c3
al.source.r1.selector.type = replicating(这个是默认的)
al.source.r1.channels = c1 c2 c3
al.source.r1.selector.optional = c3
```

## 二、多路复用模式

在这种模式下 , 用户可以指定转发的规则。selector根据规则进行数据的分发

### 可配置参数

配置项	说明
selector.type	multiplexing 表示路由模式
selector.header	指定要监测的头的名称
selector.mapping.*	匹配规则
selector.default	如果未满足匹配规则 , 则默认发往指定的通道

### 案例需求说明 :

01机利用http resource接收数据 , 根据路由规则 , 发往02 , 03机。02 , 03通过avro source接收数据 , 通过logger sink 打印数据

说明：

检测的头信息是state关键字，如果state=CN，进c1通道。如果state=US，进c2,c3通道。如果不满足以上的情况，进c4通道。

#### 配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1 s2

a1.channels=c1 c2

#描述/配置a1的source1

a1.sources.r1.type=http

a1.sources.r1.port=8888

a1.sources.r1.selector.type=multiplexing

a1.sources.r1.selector.header=state

a1.sources.r1.selector.mapping.cn=c1

a1.sources.r1.selector.mapping.us=c2

a1.sources.r1.selector.default=c2

#描述sink

a1.sinks.s1.type=avro

a1.sinks.s1.hostname=192.168.234.212

a1.sinks.s1.port=9999

a1.sinks.s2.type=avro

a1.sinks.s2.hostname=192.168.234.213

a1.sinks.s2.port=9999

#描述内存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

#位channel 绑定 source和sink

a1.sources.r1.channels=c1 c2

a1.sinks.s1.channel=c1

a1.sinks.s2.channel=c2

#### 02,03配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1

a1.channels=c1

#描述/配置a1的source1

a1.sources.r1.type=avro

```
a1.sources.r1.bind=0.0.0.0
```

```
a1.sources.r1.port=9999
```

```
#描述sink
```

```
a1.sinks.s1.type=logger
```

```
#描述内存channel
```

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

```
a1.channels.c1.capacity=1000
```

```
a1.channels.c1.transactionCapacity=100
```

```
#位channel 绑定 source和sink
```

```
a1.sources.r1.channels=c1
```

```
a1.sinks.s1.channel=c1
```

然后执行：

```
curl -X POST -d ' [{"headers":{"state":"cn"},"body":"hello china" }]' http://192.168.150.137:8888
```

# flume的Interceptors

## 概述

Flume有能力在运行阶段修改/删除Event，这是通过拦截器（Interceptors）来实现的。

拦截器需要实现org.apache.flume.interceptor.Interceptor接口。

拦截器可以修改或删除事件基于开发者在选择器中选择的任何条件。

拦截器采用了责任链模式，多个拦截器可以按指定顺序拦截。

一个拦截器返回的事件列表被传递给链中的下一个拦截器。

如果一个拦截器需要删除事件，它只需要在返回的事件集中不包含要删除的事件即可。

如果要删除所有事件，只需返回一个空列表。

## 一、Timestamp Interceptor

这个拦截器在事件头中插入以毫秒为单位的当前处理时间。

头的名字为timestamp，值为当前处理的时间戳。

如果在之前已经有这个时间戳，则保留原有的时间戳。

## 可配置项说明

配置项	说明
type	timestamp
preserveExisting	false 如果时间戳已经存在是否保留

## 配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1 s2

a1.channels=c1 c2

#描述/配置a1的source1

a1.sources.r1.type=http

a1.sources.r1.port=8888

a1.sources.r1.selector.type=multiplexing

a1.sources.r1.selector.header=state

```
al.sources.r1.selector.mapping.cn=c1
```

```
al.sources.r1.selector.mapping.us=c2
```

```
al.sources.r1.selector.default=c2
```

```
al.sources.r1.interceptors=il
```

```
al.sources.r1.interceptors.il.type=timestamp
```

```
#描述sink
```

```
al.sinks.s1.type=avro
```

```
al.sinks.s1.hostname=192.168.234.212
```

```
al.sinks.s1.port=9999
```

```
al.sinks.s2.type=avro
```

```
al.sinks.s2.hostname=192.168.234.213
```

```
al.sinks.s2.port=9999
```

```
#描述内存channel
```

```
al.channels.c1.type=memory
```

```
al.channels.c1.capacity=1000
```

```
al.channels.c1.transactionCapacity=100
```

```
al.channels.c2.type=memory
```

```
al.channels.c2.capacity=1000
```

```
al.channels.c2.transactionCapacity=100
```

```
#位channel 绑定 source和sink
```

```
al.sources.r1.channels=c1 c2
```

```
al.sinks.s1.channel=c1
```

```
al.sinks.s2.channel=c2
```

结果：

```
2016-08-26 23:43:03,524 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{state=jp, timestamp=1472280180424} body: 69 64 6F 61 6C 6C 2E 6F 72 67 5F 62 6F 64 79   idoall.org_body }
```



## 二、Host Interceptor

这个拦截器插入当前处理Agent的主机名或ip

头的名字为host或配置的名称

值是主机名或ip地址，基于配置。

参数说明：

配置参数	说明
type	host
preserveExisting	false 如果主机名已经存在是否保留
useIP	true 如果配置为true则用IP，配置为false则用主机名
hostHeader	host 加入头时使用的名称



配置示例：

#配置Agent a1 的组件

a1.sources=r1

a1.sinks=s1 s2

a1.channels=c1 c2

#描述/配置a1的source1

a1.sources.r1.type=http

a1.sources.r1.port=8888

a1.sources.r1.selector.type=multiplexing

a1.sources.r1.selector.header=state

a1.sources.r1.selector.mapping.cn=c1

a1.sources.r1.selector.mapping.us=c2

a1.sources.r1.selector.default=c2

a1.sources.r1.interceptors=i1 i2

a1.sources.r1.interceptors.i1.type=timestamp

a1.sources.r1.interceptors.i2.type=host

#描述sink

a1.sinks.s1.type=avro

a1.sinks.s1.hostname=192.168.234.212

a1.sinks.s1.port=9999

a1.sinks.s2.type=avro

a1.sinks.s2.hostname=192.168.234.213

a1.sinks.s2.port=9999

#描述内存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

#位channel 绑定 source和sink

a1.sources.r1.channels=c1 c2

a1.sinks.s1.channel=c1

a1.sinks.s2.channel=c2

结果：

```
e.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{host=127.0.0.1, state=cn, ti
```

### 三、Static Interceptor

此拦截器允许用户增加静态头信息使用静态的值到所有事件。

目前的实现中不允许一次指定多个头。

如果需要增加多个静态头可以指定多个Static interceptors

配置项	说明
type	static
preserveExisting	true
key	key 要增加的头名

value	value 要增加的头值
-------	--------------



配置示意：

```
#配置Source
a1.sources.r1.type = http
a1.sources.r1.port = 44444
a1.sources.r1.selector.type=multiplexing
a1.sources.r1.selector.header=state
a1.sources.r1.selector.mapping.cn=c1
a1.sources.r1.selector.mapping.us=c2
a1.sources.r1.selector.default=c2
a1.sources.r1.interceptors = i1 i2 i3
a1.sources.r1.interceptors.i1.type = timestamp
a1.sources.r1.interceptors.i2.type = host
a1.sources.r1.interceptors.i3.type = static
a1.sources.r1.interceptors.i3.key = addr
a1.sources.r1.interceptors.i3.value = bj
```

结果示意：

```
2016-08-26 23:46:55,176 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{host=192.168.242.111, state=jp, addr=bj, timestamp=1472280412325} body: 69 64 6F 61 6C 6C 2E 6F 72 67 5F 62 6F 64 79      idoall.org_body }
```

四.UUID Interceptor

这个拦截器在所有事件头中增加一个全局一致性标志。

其实就是UUID。

可配置说明

配置项	说明
type	org.apache.flume.sink.solr.morphline.UUIDInterceptor\$Builder
headerName	id 头名称
preserveExisting	true 如果头已经存在，是否保留
prefix	“ ” 在UUID前拼接的字符串前缀



配置示例：

```
#配置Source
al.sources.r1.type = http
al.sources.r1.port = 44444
al.sources.r1.selector.type=multiplexing
al.sources.r1.selector.header=state
al.sources.r1.selector.mapping.cn=c1
al.sources.r1.selector.mapping.us=c2
al.sources.r1.selector.default=c2
al.sources.r1.interceptors = i1 i2 i3 i4
al.sources.r1.interceptors.i1.type = timestamp
al.sources.r1.interceptors.i2.type = host
al.sources.r1.interceptors.i3.type = static
al.sources.r1.interceptors.i3.key = addr
al.sources.r1.interceptors.i3.value = bj
al.sources.r1.interceptors.i4.type = org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder
```

结果示意：

```
2016-08-26 23:48:53,212 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{host=192.168.242.111, state=jp, id=d354d2f0-14ff-4815-b382-99ae0a191926, addr=bj, timestamp=1472280528105} body: 69 64 6F 61 6C 6C 2E 6F 72 67 5F 62 6F 64 79   idoall.org_body }
```

五.Search and Replace Interceptor

这个拦截器提供了简单的基于字符串的正则搜索和替换功能。

可配置项说明

配置项	说明
type	search_replace
searchPattern	要搜索和替换的正则表达式
replaceString	要替换为的字符串
charset	UTF-8 字符集编码，默认utf-8

 配置示意：

```
#配置Source
a1.sources.r1.type = http
a1.sources.r1.port = 44444
a1.sources.r1.selector.type=multiplexing
a1.sources.r1.selector.header=state
a1.sources.r1.selector.mapping.cn=c1
a1.sources.r1.selector.mapping.us=c2
a1.sources.r1.selector.default=c2
a1.sources.r1.interceptors = i1 i2 i3 i4 i5
a1.sources.r1.interceptors.i1.type = timestamp
a1.sources.r1.interceptors.i2.type = host
a1.sources.r1.interceptors.i3.type = static
a1.sources.r1.interceptors.i3.key = addr
a1.sources.r1.interceptors.i3.value = bj
a1.sources.r1.interceptors.i4.type = org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder
a1.sources.r1.interceptors.i5.type = search_replace
a1.sources.r1.interceptors.i5.searchPattern = [0-9]
a1.sources.r1.interceptors.i5.replaceString = *
```

发送的数据示意（比如在body里写一些数字）：

```
[root@hadoop01 ~]# curl -X POST -d ' [{ "headers" : {"state":"jp"}, "body" : "abc123321def"}]' http://192.168.242.111:44444
```

结果示意：

```
2016-08-26 23:53:07,279 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.process(LoggerSink.java:94)] Event: { headers:{host=192.168.242.111, state=jp, id=ae490979-a0b2-48c7-a46f-d57979887d2c, addr=bj, timestamp=1472280780737} body: 61 62 63 2A 2A 2A 79 2A 2A 2A 64 65 66      abc***y***def }
```

## 六.Regex Filtering Interceptor

此拦截器通过解析事件体去匹配给定正则表达式来筛选事件。

所提供的正则表达式即可以用来包含或剔除事件。

属性说明：

配置项	说明
type	regex_filter
regex	”.*” 所要匹配的正则表达式
excludeEvents	false 如果是true则剔除匹配的事件，false则包含匹配的事件。



配置示意：

```

l.sources.r1.interceptors.i2.type = host
l.sources.r1.interceptors.i3.type = static
l.sources.r1.interceptors.i3.key = addr
l.sources.r1.interceptors.i3.value = bj
l.sources.r1.interceptors.i4.type = org.apache.flume.sink.solr.morphline.UUIDInterceptors$Builder
l.sources.r1.interceptors.i5.type = search_replace
l.sources.r1.interceptors.i5.searchPattern = [0-9]
l.sources.r1.interceptors.i5.replaceString = *
l.sources.r1.interceptors.i6.type = regex_filter
l.sources.r1.interceptors.i6.regex = ^jp.*$
l.sources.r1.interceptors.i6.excludeEvents = true

```

结果将过滤到以jp开头的信息，即如果发送的是以jp开头的信息，则收不到

```

[192.168.242.111:4444]
[root@hadoop01 ~]# curl -X POST -d '{"headers":{"state":"jp"},"body":"jpabc123y321def"}' http://192.168.242.111:4444
[root@hadoop01 ~]#

```

## 七.Regex Extractor Interceptor

使用指定正则表达式匹配事件，并将匹配到的组作为头加入到事件中。

它也支持插件化的序列化器用来格式化匹配到的组在加入他们作为头之前。

属性说明：

!type - 类型，必须是regex\_extractor

!regex - 要匹配的正则表达式

!serializers - Space-separated list of serializers for mapping matches to header names and serializing their values. (See example below) Flume provides built-in support for the following serializers:

org.apache.flume.interceptor.RegexExtractorInterceptorPassThroughSerializer

org.apache.flume.interceptor.RegexExtractorInterceptorMillisSerializer

serializers.<sl>.type default Must be default

(org.apache.flume.interceptor.RegexExtractorInterceptorPassThroughSerializer),

org.apache.flume.interceptor.RegexExtractorInterceptorMillisSerializer, or the

FQCN of a custom class that implements

org.apache.flume.interceptor.RegexExtractorInterceptorSerializer

serializers.<sl>.name -

serializers.\* - Serializer-specific properties

----

If the Flume event body contained 1:2:3.4foobar5 and the following configuration

was used

```
a1.sources.r1.interceptors.i1.regex = (\\d):(\\d):(\\d)
```

```
a1.sources.r1.interceptors.i1.serializers = s1 s2 s3
```

```
a1.sources.r1.interceptors.i1.serializers.s1.name = one
```

```
a1.sources.r1.interceptors.i1.serializers.s2.name = two
```

```
a1.sources.r1.interceptors.i1.serializers.s3.name = three
```

The extracted event will contain the same body but the following headers will have

been added one=>1, two=>2, three=>3

----

# flume的Process

## 概述

Sink Group允许用户将多个Sink组合成一个实体。

Flume Sink Processor 可以通过切换组内Sink用来实现负载均衡的效果，或在一个Sink故障时切换到另一个Sink。

## 📖 格式要求：

sinks - 用空格分隔的Sink集合

processor.type default 类型名称，必须是 default、failover 或

load\_balance

## Default Sink Processor

Default Sink Processor 只接受一个 Sink。这是默认的策略。即如果不配置Processor，用的是这个策略。

## Failover Sink Processor

Failover Sink Processor 维护一个sink们的优先表。确保只要一个是可用的就事件就可以被处理。

失败处理原理是，为失效的sink指定一个冷却时间，在冷却时间到达后再重新使用。

sink们可以被配置一个优先级，数字越大优先级越高。

如果sink发送事件失败，则下一个最高优先级的sink将会尝试接着发送事件。

如果没有指定优先级，则优先级顺序取决于sink们的配置顺序，先配置的默认优先级高于后配置的。

在配置的过程中，设置一个group processor，并且为每个sink都指定一个优先级。

优先级必须是唯一的。

另外可以设置maxpenalty属性指定限定失败时间。

## 可配置项说明

配置项	说明
sinks	Space-separated list of sinks that are participating in the group
processor.type	failover
processor.priority	设置优先级，注意，每个sink的优先级必须是唯一的
processor.maxpenalty	30000 The maximum backoff period for the failed Sink (in millis)
示例	al.sinkgroups = g1 al.sinkgroups.g1.sinks = k1 k2 al.sinkgroups.g1.processor.type = failover al.sinkgroups.g1.processor.priority.k1 = 5



```
al.sinkgroups.g1.processor.priority.k2 = 10
al.sinkgroups.g1.processor.maxpenalty = 10000
```

配置示意：

```
#声明Agent
a1.sources = r1
a1.sinks = k1 k2
a1.channels = c1 c2
a1.sinkgroups = g1
a1.sinkgroups.g1.sinks = k1 k2
a1.sinkgroups.g1.processor.type=failover
a1.sinkgroups.g1.processor.priority.k1=5
a1.sinkgroups.g1.processor.priority.k2=10

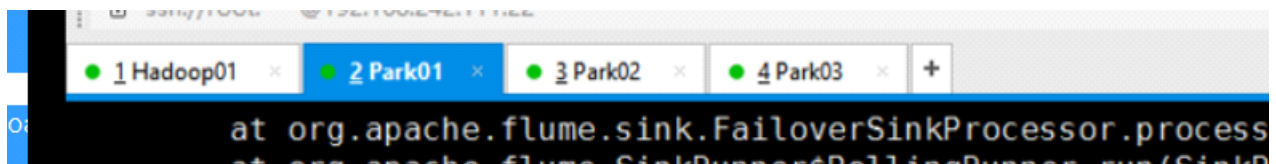
#配置Source
a1.sources.r1.type = http
a1.sources.r1.port = 44444

#配置sink
a1.sinks.k1.type=avro
a1.sinks.k1.hostname=Park02
a1.sinks.k1.port=9999

a1.sinks.k2.type=avro
a1.sinks.k2.hostname=Park03
a1.sinks.k2.port=9999

#配置channel
```

测试示意：首先向park01发送数据=》sink到 park02和park03,但是park03的优先级高，所以 park03会收到。=》当把park03关机之后，经过超时时间后，park02才会收到数据。



### Load balancing Sink Processor

Load balancing Sink processor 提供了在多个sink之间实现负载均衡的能力。

它维护了一个活动sink的索引列表。

它支持轮询 或 随机方式的负载均衡，默认值是轮询方式，可以通过配置指定。

也可以通过实现AbstractSinkSelector接口实现自定义的选择机制。

配置项	说明
processor.sinks	
processor.type	load_balance

<code>processor.backoff</code>	<p>false Should failed sinks be backed off exponentially. 开启后，故障的节点会列入黑名单，过一定时间再次发送，如果还失败，则等待是指数增长；直到达到最大的时间。</p> <p>如果不开启，故障的节点每次都会被重试。</p>
<code>processor.selector</code>	<p>round_robin ( 轮叫调度 )      Selection mechanism. Must be either round_robin, random or FQCN of custom class that inherits from AbstractSinkSelector</p>
示例	<pre>al.sinkgroups = g1 al.sinkgroups.g1.sinks = k1 k2 al.sinkgroups.g1.processor.type = load_balance al.sinkgroups.g1.processor.backoff = true al.sinkgroups.g1.processor.selector = random</pre>

## 配置示意

**注：**负载均衡模式，通道为1个。此外，在启动flume时，先启动要负载的节点02,03，再启动负责负载均衡的01节点：

## 01的配置示例：

#配置Agent a1 的组件

```
al.sources=r1
al.sinks=s1 s2
al.channels=c1
al.sinkgroups=g1
```

#描述/配置a1的source1

```
al.sources.r1.type=http
al.sources.r1.port=8888
al.sinkgroups.g1.sinks=s1 s2
al.sinkgroups.g1.processor.type=load_balance
al.sinkgroups.g1.processor.backoff=true
al.sinkgroups.g1.processor.selector=round_robin
轮叫调度算法 ( 轮询发送 )
```

#描述sink


```
al.sinks.s1.type=avro
al.sinks.s1.hostname=192.168.234.212
al.sinks.s1.port=9999
```

```
al.sinks.s2.type=avro
al.sinks.s2.hostname=192.168.234.213
al.sinks.s2.port=9999
```

#描述内存channel

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

```
#位channel 绑定 source和sink
a1.sources.r1.channels=c1
a1.sinks.s1.channel=c1
a1.sinks.s2.channel=c1
```

 02,03的配置示例：

#配置Agent a1 的组件

```
a1.sources=r1
```

```
a1.sinks=s1
```

```
a1.channels=c1
```

#描述/配置a1的source1

```
a1.sources.r1.type=avro
```

```
a1.sources.r1.bind=0.0.0.0
```

```
a1.sources.r1.port=9999
```

#描述sink

```
a1.sinks.s1.type=logger
```

#描述内存channel

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

```
a1.channels.c1.capacity=1000
```

```
a1.channels.c1.transactionCapacity=100
```

#位channel 绑定 source和sink

```
a1.sources.r1.channels=c1
```

```
a1.sinks.s1.channel=c1
```