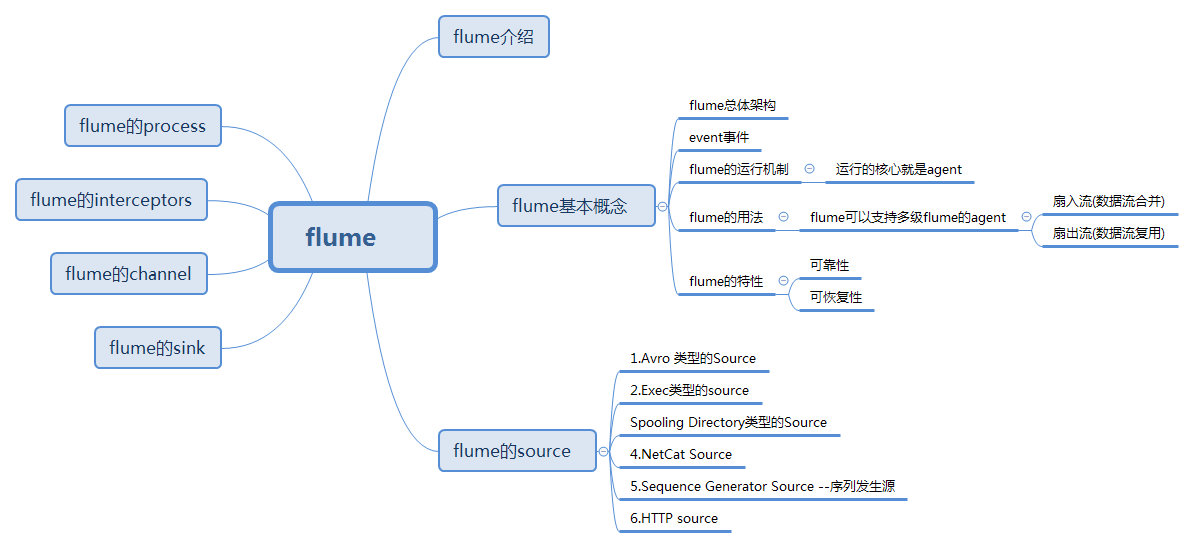
Big\_day13

flume

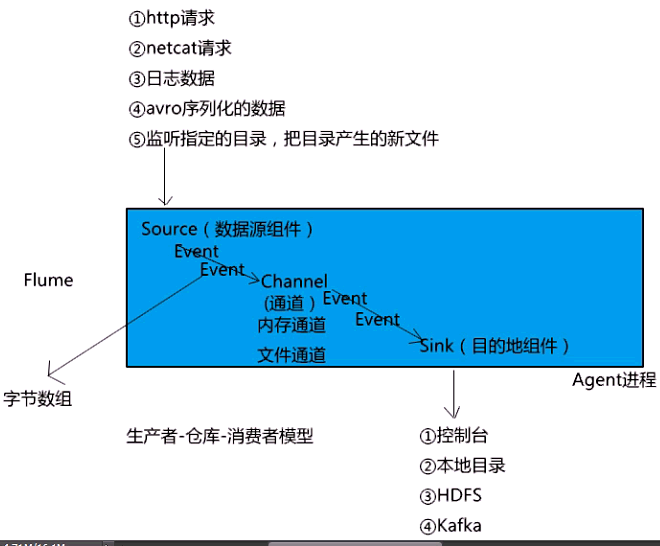
**flume介绍**



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)。

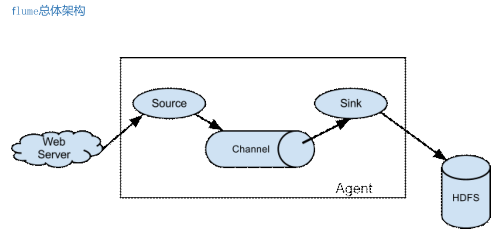
**版本**



1.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）。

2.Flume1.X版本的统称Flume-ng。不需要引入zookeerper。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。

**flume基本概念---简单来说就是收集日志的**



**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：其中event信息就是flume收集到的日记记录.

IMG_259

**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，chanel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。

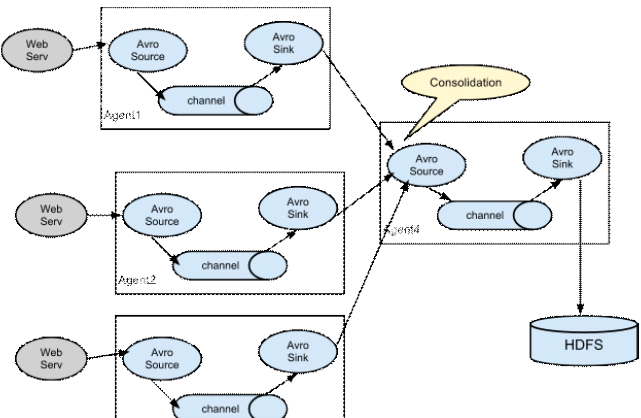
**flume的用法**

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)。

**数据流合并（扇入流）:**

就是source可以接受多个输入.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。

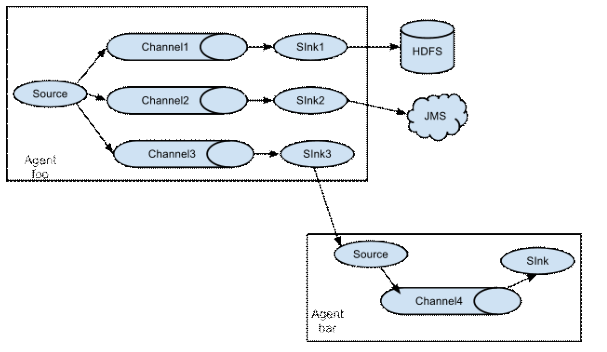


这个可在Flume中可以实现，需要配置大量第一层的agent，每一个agent都有一个avro sink，让它们都指向同一个agent的avro source（强调一下，在这样一个场景下你也可以使用thrift

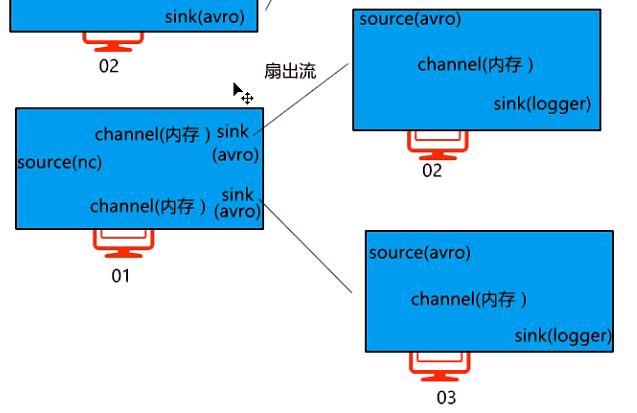
source/sink/client）。在第二层agent上的source将收到的event合并到一个channel中，event被一个sink消费到它的最终的目的地。

**数据流复用（扇出流）--**就是sink可以将数据输出多个目的地中。

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**

1.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。

2.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**

1.#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）

2.#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）

3.#描述a1的s1

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

4.#描述a1的c1

a1.channels.c1.type=memory

a1.channels.c1.capacity=1000

a1.channels.c1.transactionCapacity=100

5.#为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.启动flume:

**bin#** ./flume-ng agent -**n** a1 -**c** ../conf -**f** ../conf/template.conf **-D**flume.root.logger=INFO,console

**conf#** ../bin/flume-ng agent -**n** a1 -**c** ./ **-f** interceptor.conf2 **-D**flume.root.logger=INFO,console

-c:指定配置文件放在什么目录 -f:指定配置文件

-n:Agent的名称 -Dproperty=value 设置一个JAVA系统属性值

7.通过nc来访问:

nc localhost 8888 // http ://176.129.8.81:8888/hemo ---打印http/ accpet....

**flume的source:**

**1.Avro 类型的Source**

**监听**Avro 端口来接收外部avro客户端的事件流,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。

修改配置文件，source的type属性为 ---> a1.sources.r1.type=avro

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

IMG_263

监听到移动了文件夹

**2.Exec类型的source**

**可以将命令产生的输出作为源**

配置type:类型名称，需要是"exec"

a1.sources.r1.type=exec

a1.sources.r1.command=cat /home/software/1.txt

a1.sinks.s1.type=logger

获取到的数据作为源

**3.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。

a1.sources.r1.type=spooldir

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

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。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）

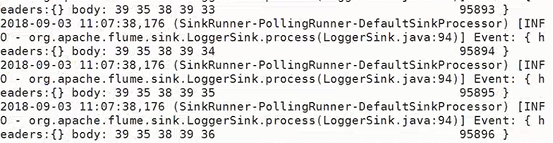
**4.NetCat Source**

**用来监听一个指定端口，并接收监听到的数据。(见入门配置-安装与配置)**

**5.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。

a1.sources.r1.type=seq



**6.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 Envent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Evnet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.

**BlobHandler**:是一种将请求中上传文件信息转化为event的处理器。

a1.sources.r1.type=http

a1.sources.r1.port=8888

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**

1.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。

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(../conf/log4j.properties)

可以通过-Dflume.root.logger=INFO,console在命令启动时手动指定log4j参数

2.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。

a1.sinks.s1.type=file\_roll

a1.sinks.s1.sink.directory=/home/work/rolldata---保存在指定的目录里面

a1.sinks.s1.sink.rollInterval=60

3.Avro Sink

是实现多级流动、扇出流(1到多) 扇入流(多到1) 的基础。

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logger sink 输出到控制台上

**3.2扇出流案例需求说明**--01机发出的数据，让02，03来接收

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

**3.3扇入案列需求说明,**02,03机收到的数据都发往01

a1.sinks.s1.type=avro

a1.sinks.s1.hostname=192.168.234.163---指向01的ip

a1.sinks.s1.port=9999

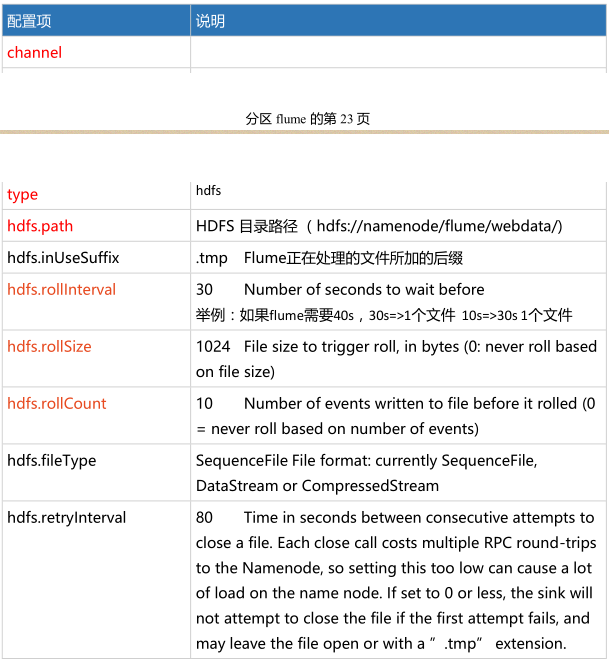
4.HDFS Sink

此Sink将事件写入到**Hadoop(伪)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haddop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。



a1.sinks.s1.type=hdfs

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

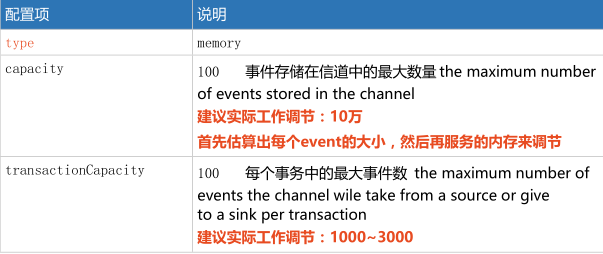
a1.sinks.s1.hdfs.fileType=DataStream

可以通过eclipse插件来查看

**flume的channel**

**1.\*\*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



IMG_267

**2.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。

**3.FileChannel**

性能比较低，但是即使程序出错数据不会丢失



a1.channels.c1.type=file

a1.channels.c1.dataDirs=/home/filechannel

**4.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**

1.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 |

a1.sources = r1

a1.channels = c1 c2 c3

a1.source.r1.selector.type = replicating(这个是默认的)

a1.source.r1.channels = c1 c2 c3

a1.source.r1.selector.optional = c3

2.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



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。

执行：

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

**flume的interceptors**

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。

**1.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。



a1.sources.r1.interceptors=i1

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

**2.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。



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

**3.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 | true |
| value | 要增加的头值 |

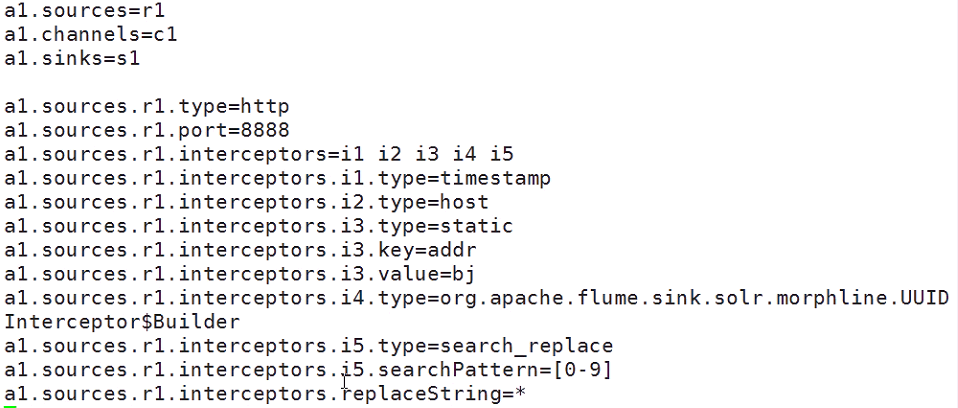
**4.UUID Interceptor**

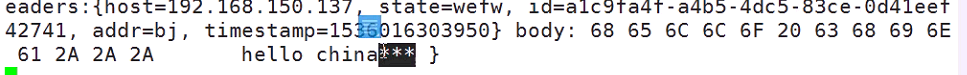
这个拦截器在所有事件头中增加一个全局一致性标志。其实就是UUID。



5..Search and Replace Interceptor

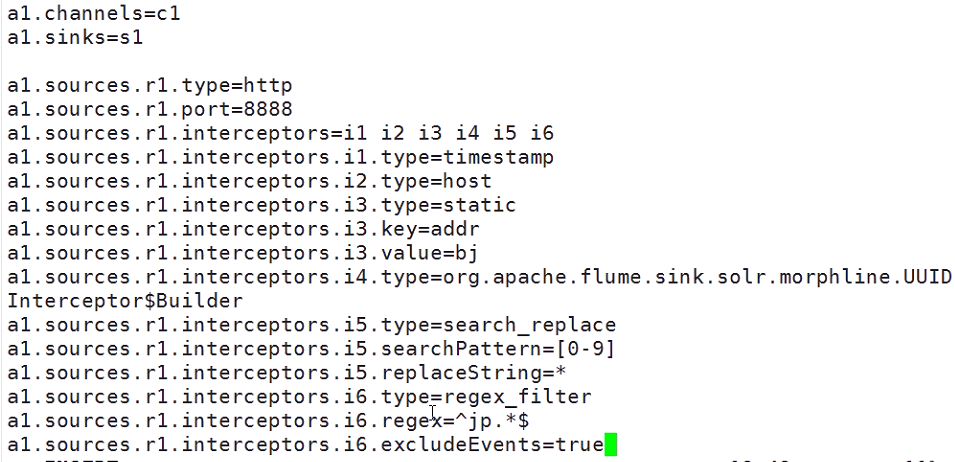
这个拦截器提供了简单的基于字符串的正则搜索和替换功能。





6.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刨除事件。



7.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。

**flume的process**

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。

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

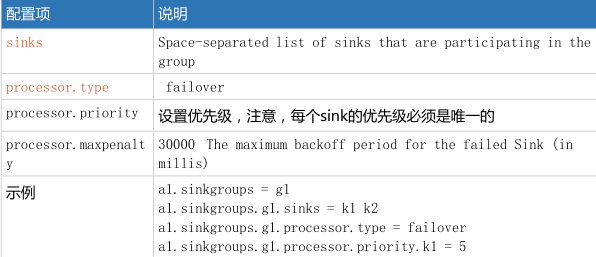
**Default Sink Processor** 只接受一个 Sink。这是默认的策略。即如果不配置Processor，用

的是这个策略。

**Failover Sink Processor** 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。



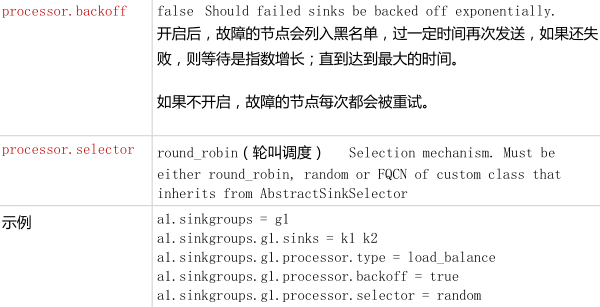
IMG_277

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**

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。



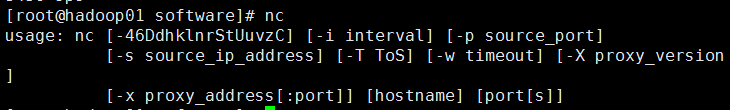


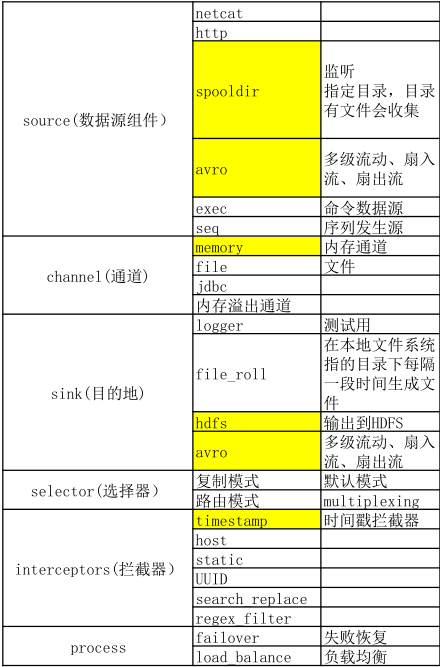
注：负载均衡模式，通道为1个。此外，在启动flume时，先启动要负载的节点02,03，再启动

负责负载均衡的01节点：

另注:

检验是否安装成功nc :





**flume事务机制**

**概述**

Flume的事务机制与可靠性保证的实现，最核心的组件是Channel（通道）。如果没有Channel组件，而紧靠Source与Sink组件是无从谈起的。

**文件通道（File Channel）**

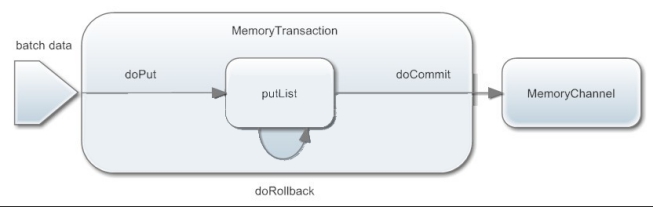
文件通道指的是将事件存储到代理（Agent）本地文件系统中的通道。虽然要比内存通道慢一些，不过它却提供了持久化的存储路径，可以应对大多数情况，它应该用在数据流中不允许出现缺口的场合。

**内存通道**

File channel虽然提供了持久化，但是其性能较差，吞吐量会受到一定的限制。相反，memory channel则牺牲可靠性换取吞吐量。当然，如果机器断电重启，则无法恢复。在实际应用中，大多数企业都是选择内存通道，因为在通过flume收集海量数据场景下，使用FileChannel所带来的性能下降是很大的甚至是无法忍受的。

**Flume内存通道事务机制**

**编程模型**



**Put事务流程**

Put事务可以分为以下阶段：

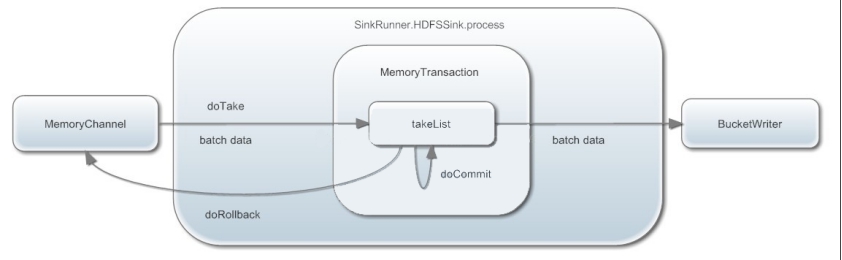
doPut:将批数据先写入临时缓冲区putList（Linkedblockingdequeue)

doCommit:检查channel内存队列是否足够合并。

doRollback:channel内存队列空间不足，回滚，等待内存通道的容量满足合并

putList就是一个临时的缓冲区，数据会先put到putList,最后由commit方法会检查channel是否有足够的缓冲区，有则合并到channel的队列。

**Take事务**



Take事务分为以下阶段：

doTake:先将数据取到临时缓冲区takeList(linkedBlockingDequeue),将数据发送到下一个节点

doCommit:如果数据全部发送成功，则清除临时缓冲区takeList

doRollback:数据发送过程中如果出现异常，rollback将临时缓冲区takeList中的数据归还给channel内存队列。