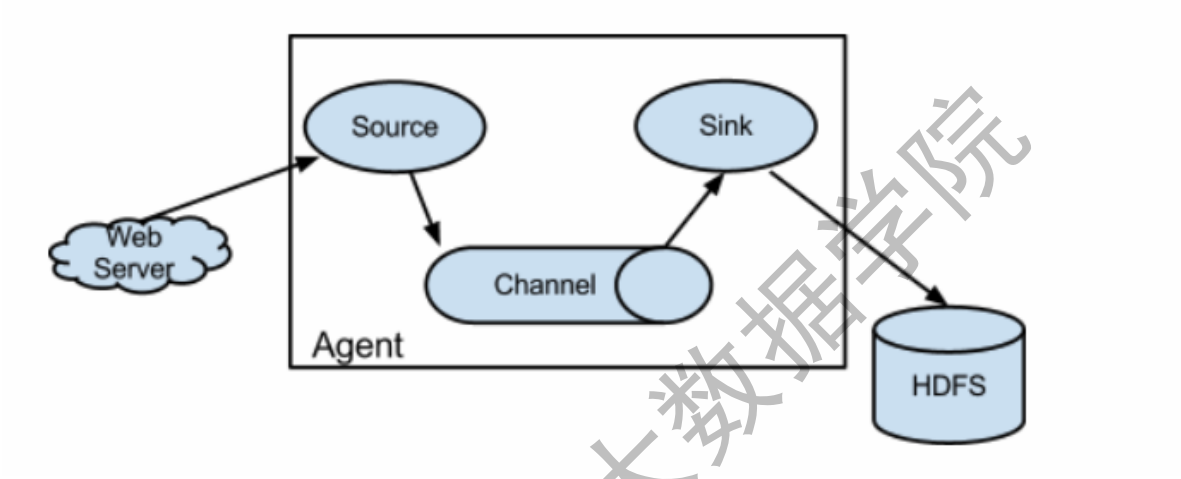
大数据组件 Flume

2.12.1 flume简介-基础知识

Flume 初始的发行版本目前被统称为 Flume OG（original generation），属于 cloudera。但随着 Flume 功能的扩展，Flume OG 代码工程臃肿、核心组件设计不合理、核心配置不标准等缺点暴露出来，为了解决这些问题，2011 年 10 月 22 号，cloudera 完成了 Flume-728，对 Flume 进行了里程碑式的改动：重构核心组件、核心配置以及代码架构，重构后的版本统称为 Flume NG（next generation）；纳入apache下也是促使其改动的一大原因，cloudera Flume 改名为 Apache Flume。



flume的特点如下：

flume是一个分布式、高可靠、高可用的服务，能够有效的收集、聚合、移动大量的日志数据。

1、它有一个简单、灵活的基于流的数据流结构。

2、具有故障转移机制和负载均衡机制。

3、使用了一个简单的可扩展的数据模型（source、channel、sink）。

4、声明式配置，可以动态更新配置（配置修改后，不用重启服务即可生效）

flume-ng处理数据有两种方式：avro-client、agent。

avro-client：一次性将数据传输到指定的avro服务的客户端。

agent：一个持续传输数据的服务。

Agent主要组件包含：Source 、Channel、Sink

数据在组件传输的单位是Event

2.12.2 flume安装与测试

1、下载：http://apache.communilink.net/flume/1.8.0/apache-flume-1.8.0-bin.tar.gz

2、解压缩：tar zvxf apache-flume-1.8.0-bin.tar.gz

3、修改配置

$ cp conf/flume-env.sh.template conf/flume-env.sh

在conf/flume-env.sh配置JAVA_HOME

创建配置文件example.conf 参考 conf/flume-conf.properties.template

注意：export JAVA_OPTS

4、启动agent

bin/flume-ng agent --conf conf/ --conf-file conf/example.conf

--name a1 -Dflume.monitoring.type=http Dflume.monitoring.port=34343 -Dflume.root.logger=INFO,console &

配置文件example.conf 内容：

生命agent名字a1,声明sources 包含r1 sink:k1 channel:c1

a1.sources = r1

a1.sinks = k1

a1.channels = c1

配置source

a1.sources.r1.type = exec

a1.sources.r1.command = tail -F /var/log/secure

配置sink

a1.sinks.k1.type = logger

配置channel

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

绑定source与sink于channel

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

对于以上配置文件中，其核心的组件如下

1、source

主要作用：从Client收集数据，传递给Channel。可以接收外部源发送过来的数据。不同的 source，可以接受不同的数据格式。 比如有目录池(spooling directory)数据源，可以监控指定文件夹中的新文件变化，如果目录中有文件产生，就会立刻读取其内容。

常见采集的数据类型：

Exec Source、Avro Source、NetCat Source、Spooling Directory Source、Kafka Source等

不同source的具体作用：

AvroSource：监听一个avro服务端口，采集Avro数据序列化后的数据；

Thrift Source：监听一个Thrift 服务端口，采集Thrift数据序列化后的数据；

Exec Source：基于Unix的command在标准输出上采集数据；

JMS Source：Java消息服务数据源，Java消息服务是一个与具体平台无关的API，这是支持jms规范的数据源采集；

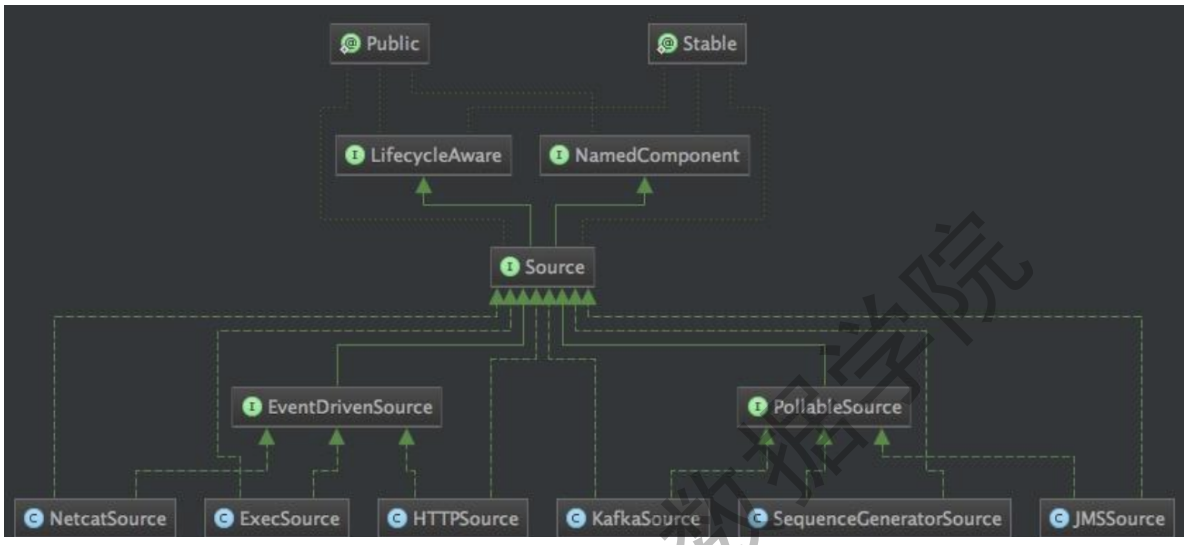
Spooling Directory Source：通过文件夹里的新增的文件作为数据源的采集；

Kafka Source：从kafka服务中采集数据。

NetCat Source： 绑定的端口（tcp、udp），将流经端口的每一个文本行数据作为Event输入

HTTP Source：监听HTTP POST和 GET产生的数据的采集

Source提供了两种机制： PollableSource（轮询拉取）和EventDrivenSource（事件驱动）：



上图展示的Source继承关系类图。

通过类图我们可以看到NetcatSource，ExecSource和HttpSource属于事件驱动模型。KafkaSource，SequenceGeneratorSource和JmsSource属于轮询拉取模型。

2、channel

Channel：一个数据的存储池，中间通道抑或是缓存队列。

主要作用：Channel用于连接Source和Sink，Source将日志信息发送到Channel，Sink从Channel消费日志信息；Channel是中转日志信息的一个临时存储，保存有Source组件传递过来的日志信息。Channel中的数据直到进入到下一个channel中或者进入终端才会被删除。当sink写入失败后，可以自动重写，不会造成数据丢失，因此很可靠。

channel的类型很多比如:内存中、jdbc数据源中、文件形式存储等。

常见采集的数据类型：

Memory Channel、File Channel、JDBC Channel、KafkaChannel、Spillable Memory Channel等

不同Channel具体作用：

Memory Channel：使用内存作为数据的存储。

JDBC Channel：使用jdbc数据源来作为数据的存储。

Kafka Channel：使用kafka服务来作为数据的存储。

File Channel：使用文件来作为数据的存储。

Spillable Memory Channel：使用内存和文件作为数据的存储，即：先存在内存中，如果内存中数据达到阀值则flush到文件中。

3、sink

Sink：数据的最终的目的地。

主要作用：接受channel写入的数据以指定的形式表现出来（或存储或展示）。

sink的表现形式很多比如:打印到控制台、hdfs上、avro服务中、文件中等。

常见采集的数据类型：

HDFS Sink、Hive Sink、Logger Sink、Avro Sink、Thrift Sink、File Roll Sink、HBaseSink、Kafka Sink等

不同Sink具体作用：

HDFS Sink：将数据传输到hdfs集群中。

Hive Sink：将数据传输到hive的表中。

Logger Sink：将数据作为日志处理（根据flume中的设置的日志的级别显示）。

Avro Sink：数据被转换成Avro Event，然后发送到指定的服务端口上。

Thrift Sink：数据被转换成Thrift Event，然后发送到指定的的服务端口上。

IRC Sink：数据向指定的IRC服务和端口中发送。

File Roll Sink：数据传输到本地文件中。

Null Sink：取消数据的传输，即不发送到任何目的地。

HBaseSink：将数据发往hbase数据库中。

MorphlineSolrSink：数据发送到Solr搜索服务器（集群）。

ElasticSearchSink：数据发送到Elastic Search搜索服务器（集群）。

Kafka Sink：将数据发送到kafka服务中。（注意依赖类库）

HDFSSink需要有hdfs的配置文件和类库。一般采取多个sink汇聚到一台采集机器负责推送到hdfs。

4、event

含义：event是Flume NG传输的数据的基本单位，也是事务的基本单位。

在文本文件，通常是一行记录就是一个event。

网络消息传输系统中，一条消息就是一个event。

结构：event里有header、body

Event里面的header类型：Map<String, String>

我们可以在source中自定义header的key：value，在某些channel和sink中使用header。

2.12.3 flume部署方式

2.12.4 flume source相关配置及测试

```
#example.conf：单节点Flume配置
    #命名Agent a1的组件
    a1.sources  =  r1
    a1.sinks  =  k1
    a1.channels  =  c1

    #描述/配置Source
```

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

    #描述Sink
    a1.sinks.k1.type  =  logger

    #描述内存Channel
    a1.channels.c1.type  =  memory
    a1.channels.c1.capacity  =  1000
    a1.channels.c1.transactionCapacity  =  100

    #为Channle绑定Source和Sink
    a1.sources.r1.channels  =  c1
    a1.sinks.k1.channel  =  c1
```

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

在windows中通过telnet命令连接flume所在机器的44444端口发送数据。

2.12.5 flume sink相关配置及测试

Flume常用Sinks有Log Sink，HDFS Sink，Avro Sink，Kafka Sink，当然也可以自定义Sink。

Logger Sink

Logger Sink以INFO 级别的日志记录到log日志中，这种方式通常用于测试。

```
Property Name          Default          Description
channel@                 –
type@                    –              类型指定：logger
maxBytesToLog            16             能够记录的最大Event Body字节数
```

HDFS Sink

Sink数据到HDFS，目前支持text 和 sequence files两种文件格式，支持压缩，并可以对数据进行分区，分桶存储。

```
Name                   Default            Description
channel@                 –
type@                    –                指定类型：hdfs
hdfs.path@               –                HDFS的路径，eg
hdfs://namenode/flume/webdata/
hdfs.filePrefix        FlumeData          保存数据文件的前缀名
hdfs.fileSuffix          –                保存数据文件的后缀名
hdfs.inUsePrefix         –                临时写入的文件前缀名
hdfs.inUseSuffix        .tmp              临时写入的文件后缀名
hdfs.rollInterval        30                间隔多长将临时文件滚动成最终目标文件，单
位：秒，

                                           如果设置成0，则表示不根据时间来滚动文件
hdfs.rollSize            1024              当临时文件达到多少（单位：bytes）时，滚动
成目标文件，

                                           如果设置成0，则表示不根据临时文件大小来滚
动文件
```

hdfs.rollCount	10	当 events 数据达到该数量时候，将临时文件滚动成目标文件，
		如果设置成0，则表示不根据events数据来滚动文件
hdfs.idleTimeout	0	当目前被打开的临时文件在该参数指定的时间（秒）内，
		没有任何数据写入，则将该临时文件关闭并重命名成目标文件
hdfs.batchSize	100	每个批次刷新到 HDFS 上的 events 数量
hdfs.codeC	–	文件压缩格式，包括：gzip, bzip2, lzo, lzop, snappy
hdfs.fileType	SequenceFile	文件格式，包括：SequenceFile, DataStream,CompressedStre，
		当使用DataStream时候，文件不会被压缩，不需要设置hdfs.codeC;
		当使用CompressedStream时候，必须设置一个正确的hdfs.codeC值；
hdfs.maxOpenFiles	5000	最大允许打开的HDFS文件数，当打开的文件数达到该值，
		最早打开的文件将会被关闭
hdfs.minBlockReplicas	–	HDFS副本数，写入 HDFS 文件块的最小副本数。
		该参数会影响文件的滚动配置，一般将该参数配置成1，才可以按照配置正确滚动文件
hdfs.writeFormat	Writable	写 sequence 文件的格式。包含：Text, Writable（默认）
hdfs.callTimeout	10000	执行HDFS操作的超时时间（单位：毫秒）
hdfs.threadsPoolSize	10	hdfs sink 启动的操作HDFS的线程数
hdfs.rollTimerPoolSize	1	hdfs sink 启动的根据时间滚动文件的线程数
hdfs.kerberosPrincipal	–	HDFS安全认证kerberos配置
hdfs.kerberosKeytab	–	HDFS安全认证kerberos配置
hdfs.proxyUser		代理用户
hdfs.round	false	是否启用时间上的”舍弃”
hdfs.roundValue	1	时间上进行“舍弃”的值
hdfs.roundUnit	second	时间上进行”舍弃”的单位，包含：second,minute,hour
hdfs.timeZone	Local Time	时区。
hdfs.useLocalTimeStamp	false	是否使用当地时间
hdfs.closeTries 0	Number	hdfs sink 关闭文件的尝试次数；
		如果设置为1，当一次关闭文件失败后，hdfs sink将不会再次尝试关闭文件，
		这个未关闭的文件将会一直留在那，并且是打开状态；
		设置为0，当一次关闭失败后，hdfs sink会继续尝试下一次关闭，直到成功
hdfs.retryInterval	180	hdfs sink 尝试关闭文件的时间间隔，
		如果设置为0，表示不尝试，相当于于将hdfs.closeTries设置成1
serializer	TEXT	序列化类型
serializer.*		

2.12.6 flume selector 相关配置与案例分析

2.12.7 flume Sink Processors相关配置和案例分析

接收组（Sink groups）允许用户将多个接收器分组到一个实体中。 接收器处理器（Sink processors）可用于在组内的所有接收器上提供负载均衡功能，或在临时故障（temporal failure）的情况下实现从一个接收器到另一个接收器的故障转移。

Property Name	Default	Description
sinks	–	以空格分隔的参与组的接收器列表
processor.type	default	组件类型名称需要是default，failover或load_balance

```
a1.sinkgroups = g1
a1.sinkgroups.g1.sinks = k1 k2
a1.sinkgroups.g1.processor.type = load_balance
```

Default Sink Processor

默认接收器只接受一个接收器。 用户不必为单个接收器创建处理器（接收器组）。 相反，用户可以遵循本用户指南中上面解释的源 - 通道 - 接收器模式。

Failover Sink Processor

故障转移接收器维护一个优先级的接收器列表，保证只要有一个可用的事件将被处理（传递）。

故障转移机制的工作原理是将故障接收器降级到池中，在池中为它们分配一个冷却期，在重试之前随顺序故障而增加。 一旦接收器成功发送事件后，它将恢复到实时池。 接收器优先级与之相关，数量越大，优先级越高。 如果在发送事件时接收器发生故障，则应尝试下一个具有最高优先级的接收器以发送事件。 例如，在优先级为80的接收器之前激活优先级为100的接收器。如果未指定优先级，则根据配置中指定接收器的顺序确定thr优先级。

要进行配置，请将接收器组处理器设置为故障转移 `failover` 并为所有单个接收器设置优先级。 所有指定的优先级必须是唯一的 此外，可以使用 `maxpenalty` 属性设置故障转移的时间上限（以毫秒为单位）。

Property Name	Default	Description
sinks	–	以空格分隔的参与组的接收器列表
processor.type	default	The component type name, needs to be `failover`
processor.priority.	-	优先值。 `<sinkName>` 必须是与当前接收器组关联的接收器实例之一。较高优先级值Sink较早被激活。 绝对值越大表示优先级越高
processor.maxpenalty	30000	组件类型名称需要是default，failover或load_balance

```
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
a1.sinkgroups.g1.processor.maxpenalty = 10000
```

负载均衡接收处理器提供了在多个接收器上进行负载均衡流量的功能。 它维护一个索引的活动接收器列表，必须在其上分配负载。 实现支持使用`round_robin`或随机选择机制（random selection）分配负载。默认`round_robin`类型，但可以通过配置覆盖。 通过从继承`AbstractSinkSelector`的实现自定义选择机制。

调用时，选择器使用其配置的选择机制选择下一个接收器并调用它。 对于`round_robin`和`random`如果所选接收器无法传递事件，则处理器通过其配置的选择机制选择下一个可用接收器。 此实现不会将失败的接收器列入黑名单，而是继续乐观地尝试每个可用的接收器。 如果所有接收器调用都导致失败，则选择器将故障传播到接收器运行器。

如果启用了`backoff`，则接收器处理器会将失败的接收器列入黑名单，将其删除以供给定超时的选择。当超时结束时，如果接收器仍然没有响应，则超时会呈指数级增加，以避免在无响应的接收器上长时间等待时卡住。 在禁用此功能的情况下，在循环中，所有失败的接收器负载将被传递到下一个接收器中，因此不均衡

Property Name	Default	Description
sinks	–	以空格分隔的参与组的接收器列表
processor.type	default	组件类型名称需要为`load_balance`
processor.backoff	false	失败的接收器是否会以指数方式退回。
processor.selector	`round_robin`	选择机制。 必须是round_robin，random或自定义类的FQCN，它继承自AbstractSinkSelector

```
a1.sinkgroups = g1
a1.sinkgroups.g1.sinks = k1 k2
a1.sinkgroups.g1.processor.type = load_balance
a1.sinkgroups.g1.processor.backoff = true
a1.sinkgroups.g1.processor.selector = random
```

2.12.8 flume Interceptors相关配置和案例分析(选讲)

Flume中的拦截器（interceptor），用户Source读取events发送到Sink的时候，在events header中加入一些有用的信息，或者对events的内容进行过滤，完成初步的数据清洗。这在实际业务场景中非常有用，Flume-ng 1.6中目前提供了以下拦截器：Timestamp Interceptor；

- Host Interceptor；
- Static Interceptor；
- UUID Interceptor；
- Morphline Interceptor；
- Search and Replace Interceptor；
- Regex Filtering Interceptor；
- Regex Extractor Interceptor；

Timestamp Interceptor

时间戳拦截器，将当前时间戳（毫秒）加入到events header中，key名字为：timestamp，值为当前时间戳。用的不是很多。比如在使用HDFS Sink时候，根据events的时间戳生成结果文件，hdfs.path = hdfs://cdh5/tmp/dap/%Y%m%d

hdfs.filePrefix = log*%Y%m%d*%H

会根据时间戳将数据写入相应的文件中。

但可以用其他方式代替（设置useLocalTimeStamp = true）。

Host Interceptor

主机名拦截器。将运行Flume agent的主机名或者IP地址加入到events header中，key名字为：host（也可自定义）。

根据上面的Source，拦截器的配置如下：

```
## source 拦截器
agent_lxw1234.sources.sources1.interceptors = i1
agent_lxw1234.sources.sources1.interceptors.i1.type = host
agent_lxw1234.sources.sources1.interceptors.i1.useIP = false
agent_lxw1234.sources.sources1.interceptors.i1.hostHeader = agentHost

# sink 1 配置
agent_lxw1234.sinks.sink1.type = hdfs
agent_lxw1234.sinks.sink1.hdfs.path = hdfs://cdh5/tmp/lxw1234/%Y%m%d
agent_lxw1234.sinks.sink1.hdfs.filePrefix = lxw1234_%{agentHost}
agent_lxw1234.sinks.sink1.hdfs.fileSuffix = .log
agent_lxw1234.sinks.sink1.hdfs.fileType = DataStream
agent_lxw1234.sinks.sink1.hdfs.useLocalTimeStamp = true
agent_lxw1234.sinks.sink1.hdfs.writeFormat = Text
agent_lxw1234.sinks.sink1.hdfs.rollCount = 0
agent_lxw1234.sinks.sink1.hdfs.rollSize = 0
agent_lxw1234.sinks.sink1.hdfs.rollInterval = 600
agent_lxw1234.sinks.sink1.hdfs.batchSize = 500
agent_lxw1234.sinks.sink1.hdfs.threadsPoolSize = 10
agent_lxw1234.sinks.sink1.hdfs.idleTimeout = 0
agent_lxw1234.sinks.sink1.hdfs.minBlockReplicas = 1
agent_lxw1234.sinks.sink1.channel = fileChannel
```

该配置用于将source的events保存到HDFS上hdfs://cdh5/tmp/lxw1234的目录下，文件名为lxw1234_<主机名>.log

Static Interceptor

静态拦截器，用于在events header中加入一组静态的key和value。

根据上面的Source，拦截器的配置如下：

```
source 拦截器
agent_lxw1234.sources.sources1.interceptors = i1
agent_lxw1234.sources.sources1.interceptors.i1.type = static
agent_lxw1234.sources.sources1.interceptors.i1.preserveExisting = true
agent_lxw1234.sources.sources1.interceptors.i1.key = static_key
agent_lxw1234.sources.sources1.interceptors.i1.value = static_value

# sink 1 配置
agent_lxw1234.sinks.sink1.type = hdfs
agent_lxw1234.sinks.sink1.hdfs.path = hdfs://cdh5/tmp/lxw1234
agent_lxw1234.sinks.sink1.hdfs.filePrefix = lxw1234_%{static_key}
agent_lxw1234.sinks.sink1.hdfs.fileSuffix = .log
agent_lxw1234.sinks.sink1.hdfs.fileType = DataStream
agent_lxw1234.sinks.sink1.hdfs.useLocalTimeStamp = true
agent_lxw1234.sinks.sink1.hdfs.writeFormat = Text
agent_lxw1234.sinks.sink1.hdfs.rollCount = 0
agent_lxw1234.sinks.sink1.hdfs.rollSize = 0
agent_lxw1234.sinks.sink1.hdfs.rollInterval = 600
agent_lxw1234.sinks.sink1.hdfs.batchSize = 500
```

```
agent_lxw1234.sinks.sink1.hdfs.threadsPoolSize = 10
agent_lxw1234.sinks.sink1.hdfs.idleTimeout = 0
agent_lxw1234.sinks.sink1.hdfs.minBlockReplicas = 1
agent_lxw1234.sinks.sink1.channel = fileChannel
```

UUID Interceptor

UUID拦截器，用于在每个events header中生成一个UUID字符串，例如：b5755073-77a9-43c1-8fad-b7a586fc1b97。生成的UUID可以在sink中读取并使用。根据上面的source，拦截器的配置如下：

```
## source 拦截器
agent_lxw1234.sources.sources1.interceptors = i1
agent_lxw1234.sources.sources1.interceptors.i1.type =
org.apache.flume.sink.solr.morphline.UUIDInterceptor$Builder
agent_lxw1234.sources.sources1.interceptors.i1.headerName = uuid
agent_lxw1234.sources.sources1.interceptors.i1.preserveExisting = true
agent_lxw1234.sources.sources1.interceptors.i1.prefix = UUID_

# sink 1 配置
agent_lxw1234.sinks.sink1.type = logger
agent_lxw1234.sinks.sink1.channel = fileChannel
```

2.12.9 flume AVRO Client开发(选讲)

由于在实际工作中，数据的生产方式极具多样性，Flume 虽然包含了一些内置的机制来采集数据，但是更多的时候用户更希望能将应用程序和flume直接相通。所以这边运行用户开发应用程序，通过IPC或者RPC连接flume并往flume发送数据。

RPC client interface

Flume的RpcClient实现了Flume的RPC机制。用户的应用程序可以很简单的调用Flume Client SDK的append(Event) 或者appendBatch(List) 方法发送数据，不用担心底层信息交换的细节。用户可以提供所需的event通过直接实现Event接口，例如可以使用简单的方便的实现SimpleEvent类或者使用EventBuilder的writeBody()静态辅助方法。

自Flume 1.4.0起，Avro是默认的RPC协议。NettyAvroRpcClient和ThriftRpcClient实现了RpcClient接口。实现中我们需要知道我们将要连接的目标flume agent的host和port用于创建client实例，然后使用RpcClient发送数据到flume agent。

官网给了一个Avro RPCclients的例子，这边直接拿来做实际测试例子。

这里我们把client.init("host.example.org",41414);

改成 client.init("192.168.233.128",50000); 与我们的主机对接

```
import org.apache.flume.Event;
import org.apache.flume.EventDeliveryException;
import org.apache.flume.api.RpcClient;
import org.apache.flume.api.RpcClientFactory;
import org.apache.flume.event.EventBuilder;
import java.nio.charset.Charset;

public class MyApp {
  public static voidmain(String[] args) {
   MyRpcClientFacade client = new MyRpcClientFacade();
   // Initializeclient with the remote Flume agent's host and port
//client.init("host.example.org",41414);
```

```
client.init("192.168.233.128",50000);

   // Send 10events to the remote Flume agent. That agent should be
   // configured tolisten with an AvroSource.
   String sampleData = "Hello Flume!";
   for (int i =0; i < 10; i++) {
     client.sendDataToFlume(sampleData);
   }

   client.cleanUp();
  }
}

class MyRpcClientFacade {
  private RpcClient client;
  private String hostname;
  private int port;

  public void init(String hostname, int port) {
   // Setup the RPCconnection
   this.hostname = hostname;
   this.port = port;
   this.client = RpcClientFactory.getDefaultInstance(hostname, port);
   // Use thefollowing method to create a thrift client (instead of the above
line):
    // this.client = RpcClientFactory.getThriftInstance(hostname, port);
  }

  public void sendDataToFlume(String data) {
   // Create aFlume Event object that encapsulates the sample data
   Event event = EventBuilder.withBody(data, Charset.forName("UTF-8"));

   // Send theevent
   try {
     client.append(event);
   } catch (EventDeliveryException e) {
     // clean up andrecreate the client
     client.close();
     client = null;
     client = RpcClientFactory.getDefaultInstance(hostname, port);
     // Use thefollowing method to create a thrift client (instead of the above
line):
     // this.client =RpcClientFactory.getThriftInstance(hostname, port);
   }
  }

  public void cleanUp() {
   // Close the RPCconnection
   client.close();
  }

}
```

下面是代理配置：

```
#配置文件：avro_client_case20.conf
```

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

# Describe/configure the source
a1.sources.r1.type = avro
a1.sources.r1.port = 50000
a1.sources.r1.host = 192.168.233.128
a1.sources.r1.channels = c1

# Describe the sink
a1.sinks.k1.channel = c1
a1.sinks.k1.type = logger

# Use a channel which buffers events inmemory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
```

这里要注意下，之前说了，在接收端需要AvroSource或者Thrift Source来监听接口。所以配置代理的时候要把a1.sources.r1.type 写成avro或者thrift

flume-ng agent -c conf -f conf/avro_client_case20.conf-n a1 -Dflume.root.logger=INFO,console

启动成功后

在eclipse 里运行JAVA程序，当然也可以打包后在服务器上运行JAVA程序。

这里要说明下，开发代码中client.append(event)不仅仅可以发送一条数据，也可以发送一个List(string) 的数据信息，也就是批量发送。

Failover Client

这个类包封装了Avro RPCclient的类默认提供故障处理能力。hosts采用空格分开host:port所代表的flume agent，构成一个故障处理组。这Failover RPC Client目前不支持thrift。如果当前选择的host agent有问题，这个failover client会自动负载到组中下一个host中。

下面是官网开发例子：

```
// Setup properties for the failover
Properties props = new Properties();
props.put("client.type", "default_failover");

// List of hosts (space-separated list of user-chosen host aliases)
props.put("hosts", "h1 h2 h3");

// host/port pair for each host alias
String host1 = "host1.example.org:41414";
String host2 = "host2.example.org:41414";
String host3 = "host3.example.org:41414";
props.put("hosts.h1", host1);
props.put("hosts.h2", host2);
props.put("hosts.h3", host3);

// create the client with failover properties
RpcClient client = RpcClientFactory.getInstance(props);
```

下面是测试的开发例子

```
import org.apache.flume.Event;
import org.apache.flume.EventDeliveryException;
import org.apache.flume.api.RpcClient;
import org.apache.flume.api.RpcClientFactory;
import org.apache.flume.event.EventBuilder;

import java.nio.charset.Charset;
import java.util.Properties;

public class Failover_Client {
    public static void main(String[] args) {
        MyRpcClientFacade2 client = new MyRpcClientFacade2();
        // Initialize client with the remote Flume agent's host and port
        client.init();

        // Send 10 events to the remote Flume agent. That agent should be
        // configured to listen with an AvroSource.
        String sampleData = "Hello Flume!";
        for (int i = 0; i < 10; i++) {
          client.sendDataToFlume(sampleData);
        }

        client.cleanUp();
      }
    }

    class MyRpcClientFacade2 {
      private RpcClient client;
      private String hostname;
      private int port;

      public void init() {
        // Setup the RPC connection
        // Use the following method to create a thrift client (instead of the
above line):
        // this.client = RpcClientFactory.getThriftInstance(hostname, port);
     // Setup properties for the failover
        Properties props = new Properties();
        props.put("client.type", "default_failover");

        // List of hosts (space-separated list of user-chosen host aliases)
        props.put("hosts", "h1 h2 h3");

        // host/port pair for each host alias
        String host1 = "192.168.233.128:50000";
        String host2 = "192.168.233.128:50001";
        String host3 = "192.168.233.128:50002";
        props.put("hosts.h1", host1);
        props.put("hosts.h2", host2);
        props.put("hosts.h3", host3);

        // create the client with failover properties
        client = RpcClientFactory.getInstance(props);
      }
```

```
    public void sendDataToFlume(String data) {
      // Create a Flume Event object that encapsulates the sample data
      Event event = EventBuilder.withBody(data, Charset.forName("UTF-8"));

      // Send the event
      try {
        client.append(event);
      } catch (EventDeliveryException e) {
        // clean up and recreate the client
        client.close();
        client = null;
        client = RpcClientFactory.getDefaultInstance(hostname, port);
        // Use the following method to create a thrift client (instead of the
above line):
        // this.client = RpcClientFactory.getThriftInstance(hostname, port);
      }
    }

    public void cleanUp() {
      // Close the RPC connection
      client.close();
    }
}
```

这边代码设三个host用于故障转移，这里偷懒，用同一个主机的3个端口模拟。代码还是将Hello Flume发送10遍给第一个flume代理，当第一个代理故障的时候，则发送给第二个代理，以顺序进行故障转移。

下面是代理配置沿用之前的那个，并对配置文件进行拷贝，

cp avro_client_case20.conf avro_client_case21.conf

cp avro_client_case20.conf avro_client_case22.conf

分别修改avro_client_case21.conf与avro_client_case22.conf中的

a1.sources.r1.port= 50001 与a1.sources.r1.port = 50002

敲命令

flume-ng agent -c conf -f conf/avro_client_case20.conf-n a1 -Dflume.root.logger=INFO,console

flume-ng agent -c conf -f conf/avro_client_case21.conf-n a1 -Dflume.root.logger=INFO,console

flume-ng agent -c conf -f conf/avro_client_case22.conf-n a1 -Dflume.root.logger=INFO,console

启动成功后

在eclipse 里运行JAVA程序Failover_Client.java，当然也可以打包后在服务器上运行JAVA程序。

我们可以看到第一个代理终端收到了，数据而其他2个终端没有数据。

然后我们把第一个终端的进程关掉，再运行一遍client程序，然后会发现这个时候是发生到第二个终端中。当第二个终端也关闭的时候，再发送数据，则是发送到最后一个终端。这里我们可以看到，故障转移的代理主机转移是采用顺序序列的。

2.12.10 flume 和kafka 的整合(后期讲)