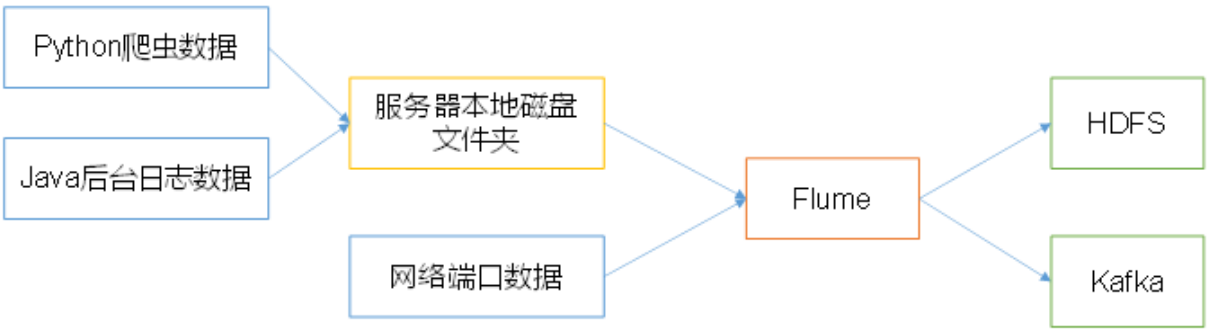
Flume

一、概述

1.Flume定义

> Flume是Cloudera提供的一个海量日志采集、传输的系统。Flume基于流式架构，灵活简单。

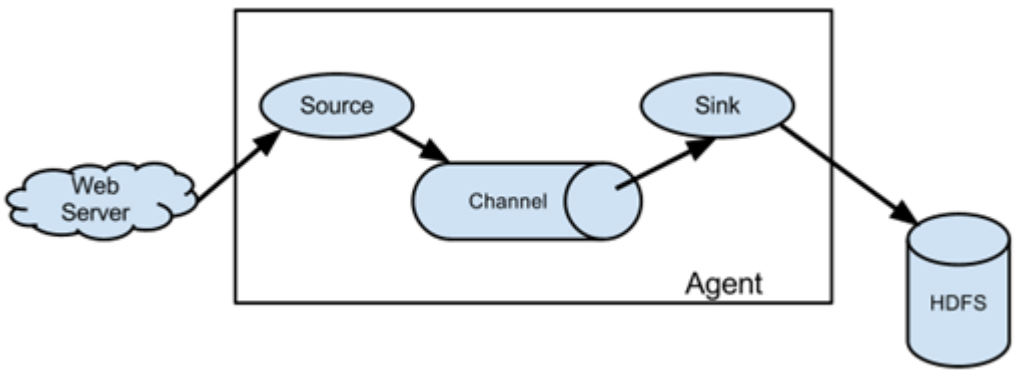


Flume最主要的作用就是，实时读取服务器本地磁盘的数据，将数据写入到HDFS。

2.Flume优点

> ① 可以和任意存储进程集成
>
> ② 输入的数据速率大于写入目的存储的速率，flume会进行缓冲，减小hdfs的压力。
>
> ③ flume中的事务基于channel，使用了两个事务模型（sender + receiver），确保消息被可靠发送。
>
> Flume使用两个独立的事务分别负责从source到channel，以及从channel到sink的事件传递。一旦事务中所有的数据全部成功提交到channel，那么source才认为该数据读取完成。同理，只有成功被sink写出去的数据，才会从channel中移除。

3.Flume组成架构



4.Flume组件

- Agent

> Agent是一个JVM进程，它以事件的形式将数据从源头送至目的地。
>
> Agent主要有3个部分组成，Source、Channel、Sink。

- Source

> Source是负责接收数据到Flume Agent的组件。

- Channel

> Channel是位于Source和Sink之间的缓冲区。因此，Channel允许Source和Sink运作在不同的速率上。Channel是线程安全的，可以同时处理几个Source的写入操作和几个Sink的读取操作。
>
> Flume自带两种Channel：Memory Channel和File Channel。
>
> Memory Channel是内存中的队列。Memory Channel在不需要关心数据丢失的情景下适用。如果需要关心数据丢失，那么Memory Channel就不应该使用，因为程序死亡、机器宕机或者重启都会导致数据丢失。
>
> File Channel将所有事件写到磁盘。因此在程序关闭或机器宕机的情况下不会丢失数据。

- Sink

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

- Event

> 传输单元，Flume数据传输的基本单元，以事件的形式将数据从源头送至目的地。 Event由可选的header和载有数据的一个byte array 构成。Header是容纳了key-value字符串对的HashMap。

Flume event

二、安装

1.安装地址

1） Flume官网地址

http://flume.apache.org

2）文档查看地址

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

3）下载地址

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

2.安装步骤

准备工作：安装JDK、并且配置环境变量

1）将apache-flume-1.9.0-bin.tar.gz上传到linux的/opt/modules目录下

2）解压apache-flume-1.9.0-bin.tar.gz到/opt/installs目录下

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

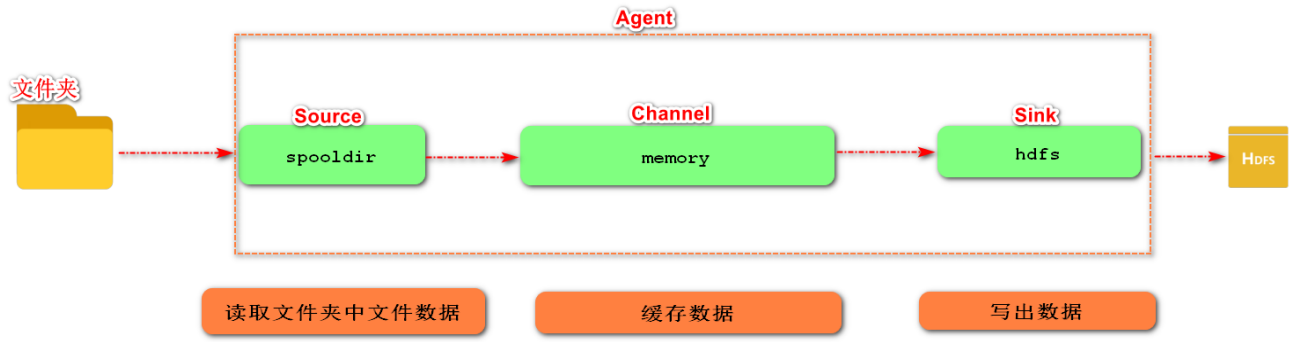
```
[root@hadoop10 conf]# pwd
/opt/installs/apache-flume-1.9.0-bin/conf
[root@hadoop10 conf]# mv flume-env.sh.template flume-env.sh
[root@hadoop10 conf]# vi flume-env.sh
export JAVA_HOME=/opt/installs/jdk1.8

[root@hadoop10 apache-flume-1.9.0-bin]# bin/flume-ng version
Flume 1.9.0
Source code repository: https://git-wip-us.apache.org/repos/asf/flume.git
Revision: d4fcab4f501d41597bc616921329a4339f73585e
Compiled by fszabo on Mon Dec 17 20:45:25 CET 2018
From source with checksum 35db629a3bda49d23e9b3690c80737f9
```

三、企业开发案例

1.实时读取目录文件到HDFS案例

1）案例需求：使用Flume监听整个目录的文件

2）需求分析：



3）实现步骤：

1. 创建spooldir-memory-hdfs.conf

```
[root@hadoop10 job]# touch spooldir-memory-hdfs.conf

# 内容如下
a1.sources = r1
a1.channels = c1
a1.sinks = k1
```

```
a1.sources.r1.type = spooldir
a1.sources.r1.spoolDir = /opt/upload
a1.sources.r1.fileSuffix = .done

a1.channels.c1.type = memory

a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = hdfs://hadoop10:9000/flume/%Y-%m-%d
a1.sinks.k1.hdfs.useLocalTimeStamp = true
#设置文件类型
a1.sinks.k1.hdfs.fileType = DataStream
#是否按照时间滚动文件夹
a1.sinks.k1.hdfs.round = true
#多少时间单位创建一个新的文件夹
a1.sinks.k1.hdfs.roundValue = 10
#重新定义时间单位
a1.sinks.k1.hdfs.roundUnit = minute
#多久生成一个新的文件  0代表禁用
a1.sinks.k1.hdfs.rollInterval = 0
#设置每个文件的滚动大小  1048576 = 1M
a1.sinks.k1.hdfs.rollSize = 1048576
#文件的滚动与 Event 数量无关
a1.sinks.k1.hdfs.rollCount = 0

a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

> 对于所有与时间相关的转义序列，Event Header中必须存在以 “timestamp”的key
>
> （除非hdfs.useLocalTimeStamp设置为true，此方法会使用TimestampInterceptor自动添加timestamp）。
>
> a1.sinks.k1.hdfs.useLocalTimeStamp = true

2. 启动监控文件夹命令

```
[root@hadoop10 apache-flume-1.9.0-bin]# bin/flume-ng agent --conf conf --name a1 --conf-file job/spooldir-memory-hdfs.conf -Dflume.root.logger=INFO,console
```

说明： 在使用Spooling Directory Source时

1) 不要在监控目录中创建并持续修改文件

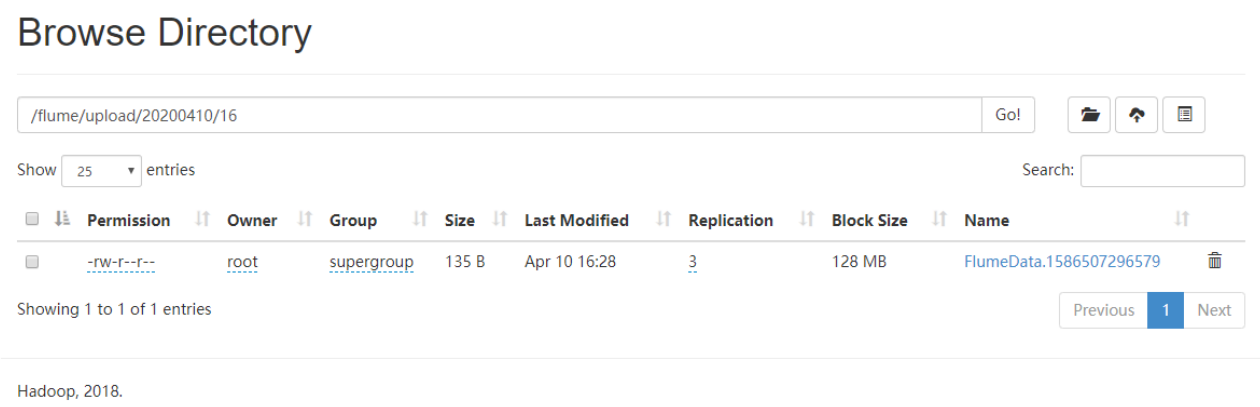
2) 上传完成的文件会以.COMPLETED结尾

3) 被监控文件夹每500毫秒扫描一次文件变动

3. 向upload文件夹中添加文件

```
[root@hadoop10 opt]# mkdir /opt/upload
[root@hadoop10 upload]# touch 1.txt
[root@hadoop10 upload]# ls
1.txt.done
```

4. 访问hdfs http://hadoop10:50070



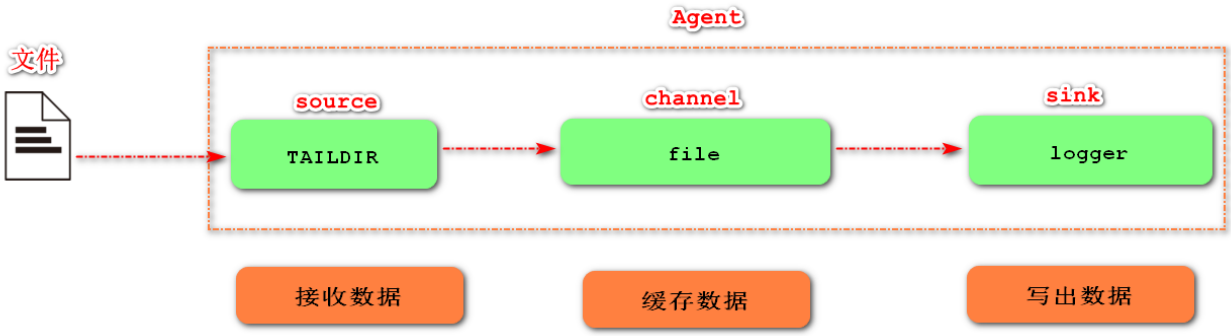
2.Taildir Source多目录断点续传

TAILDIR类型的source在读取文件完成后，会接续读取此文件，查看此文件是否有最新的文件内容，如果有最新的文件内容会对此文件的新的内容进行读取

> 备注：flume 1.7.0推出了taildirSource组件

1) 需求：监控多个目录实现断点续传

2) 需求分析:



3) 实现步骤

1. 在flume的job目录下创建一个conf文件

```
[root@hadoop10 job]# touch taildir-file-logger.conf
```

2. 在taildir-file-logger.conf文件中编写一个agent

```
a1.sources = r1
a1.channels = c1
a1.sinks = k1
```

```
a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1 f2
a1.sources.r1.filegroups.f1 = /opt/data/flume/ceshi.log
a1.sources.r1.filegroups.f2 = /opt/logs/.*log.*

a1.channels.c1.type = file

a1.sinks.k1.type = logger

a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

3. 创建指定的目录和对应的文件

```
在/opt/datas/flume/目录下创建 ceshi.log文件
在/opt目录下创建logs目录
```

4. 启动agent

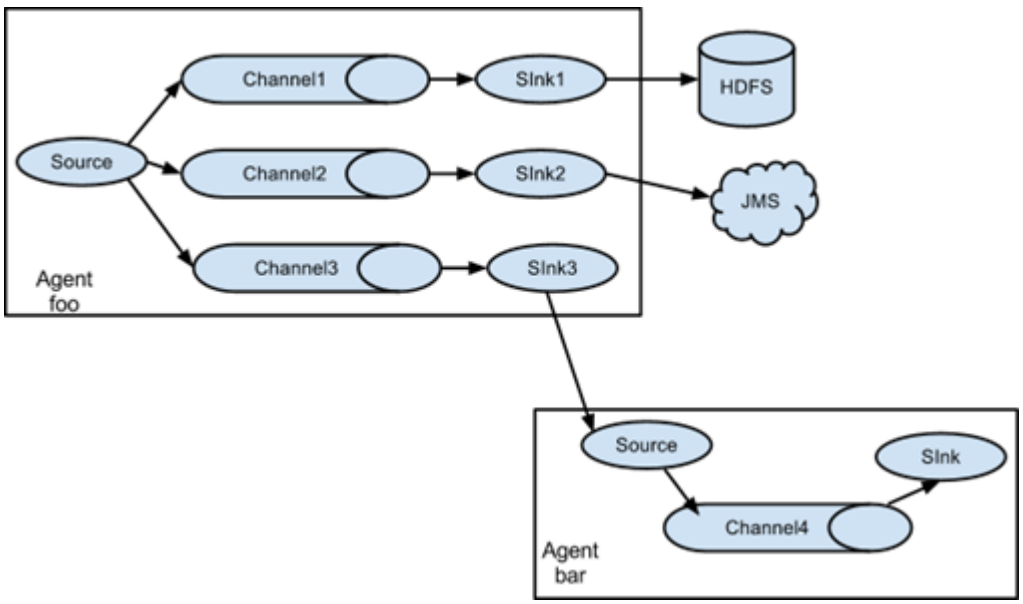
```
bin/flume-ng agent --conf conf  -name a1 --conf-file job/taildir-file-logger.conf -
Dflume.root.logger=INFO,console
```

5. 向测试.log文件中追加内容

```
echo hello world >> ceshi.log
```

3.单数据源多出口案例(选择器)

单Source多Channel、Sink 如图所示:

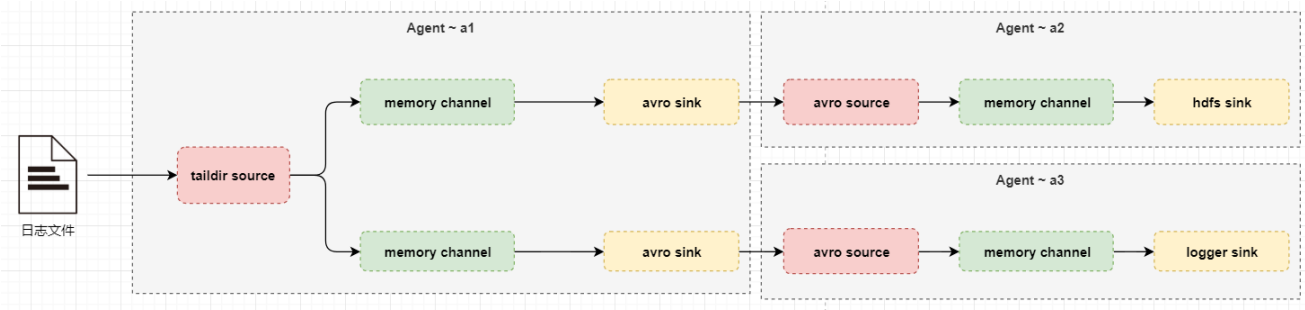


1）案例需求：使用Flume-1监控文件变动，Flume-1将变动内容传递给Flume-2，

Flume-2负责存储到HDFS。同时Flume-1将变动内容传递给Flume-3，

Flume-3负责输出到Local FileSystem。

2）需求分析：



3）实现步骤：

1. 准备工作

```
在/opt/flume/job目录下创建group1文件夹
[root@hadoop10 job]# mkdir group1
```

2. 在group1目录下,创建taildir-memory-avro.conf

```
[root@hadoop10 job]# cd group1
[root@hadoop10 group1]# touch taildir-memory-avro.conf

# 添加内容如下
a1.sources = r1
a1.channels = c1 c2
a1.sinks = k1 k2

a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/data/ceshi.log

# 将数据流复制给所有channel
a1.sources.r1.selector.type = replicating

a1.channels.c1.type = memory
a1.channels.c2.type = memory

a1.sinks.k1.type = avro
a1.sinks.k1.hostname = hadoop10
a1.sinks.k1.port = 4141

a1.sinks.k2.type = avro
a1.sinks.k2.hostname = hadoop10
a1.sinks.k2.port = 4142


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

```
a1.sinks.k1.channel = c1
a1.sinks.k2.channel = c2
```

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

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

3. 在group1目录下,创建avro-memory-hdfs.conf

```
[root@hadoop10 group1]# touch avro-memory-hdfs.conf

# 添加如下内容
a2.sources = r1
a2.channels = c1
a2.sinks = k1

a2.sources.r1.type = avro
a2.sources.r1.bind = hadoop10
a2.sources.r1.port = 4141

a2.channels.c1.type = memory

a2.sinks.k1.type = hdfs
a2.sinks.k1.hdfs.path = hdfs://hadoop10:9000/flume2/%Y-%m-%d
a2.sinks.k1.hdfs.useLocalTimeStamp = true
a2.sinks.k1.hdfs.fileType = DataStream

a2.sources.r1.channels = c1
a2.sinks.k1.channel = c1
```

4. 在group1目录下,创建avro-memory-logger.conf

```
[root@hadoop10 group1]# touch avro-memory-logger.conf

# 添加如下内容
a3.sources = r1
a3.channels = c2
a3.sinks = k1

a3.sources.r1.type = avro
a3.sources.r1.bind = hadoop10
a3.sources.r1.port = 4142

a3.channels.c2.type = memory

a3.sinks.k1.type = logger

a3.sources.r1.channels = c2
a3.sinks.k1.channel = c2
```

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

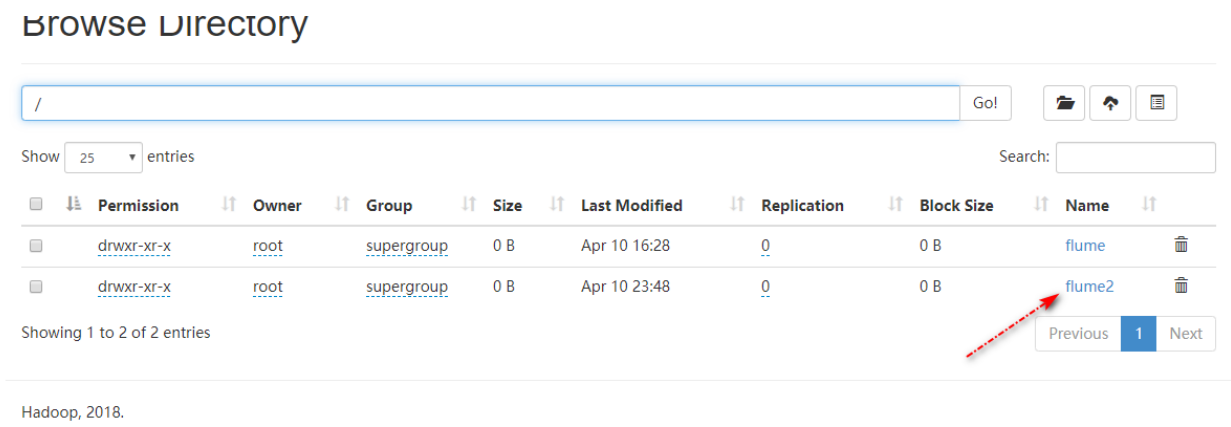
5. 执行配置文件

```
[root@hadoop10 apache-flume-1.9.0-bin]# bin/flume-ng agent --conf conf --name a3 --conf-file job/group1/avro-memory-logger.conf

[root@hadoop10 apache-flume-1.9.0-bin]# bin/flume-ng agent --conf conf --name a2 --conf-file job/group1/avro-memory-hdfs.conf

[root@hadoop10 apache-flume-1.9.0-bin]# bin/flume-ng agent --conf conf --name a1 --conf-file job/group1/exec-memory-avro.conf
```

6. 提交测试数据

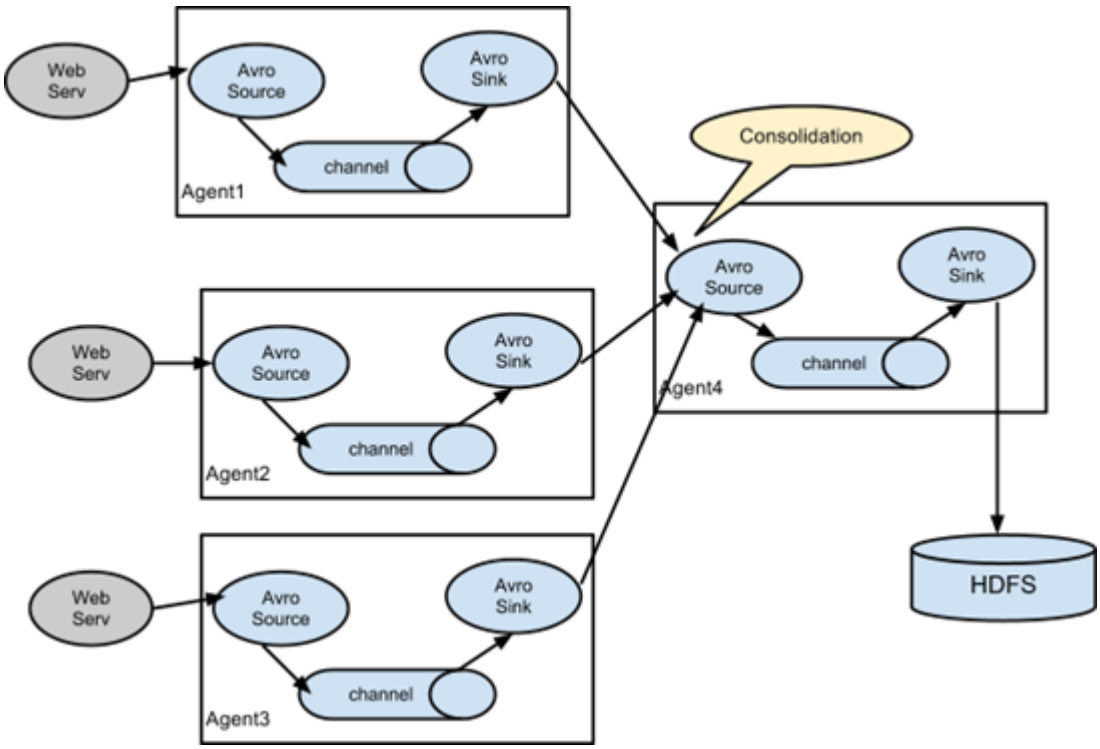
```
[root@hadoop10 opt]# echo hahaha >> a.log
```

7. 检查HDFS上数据



4.多数据源汇总案例

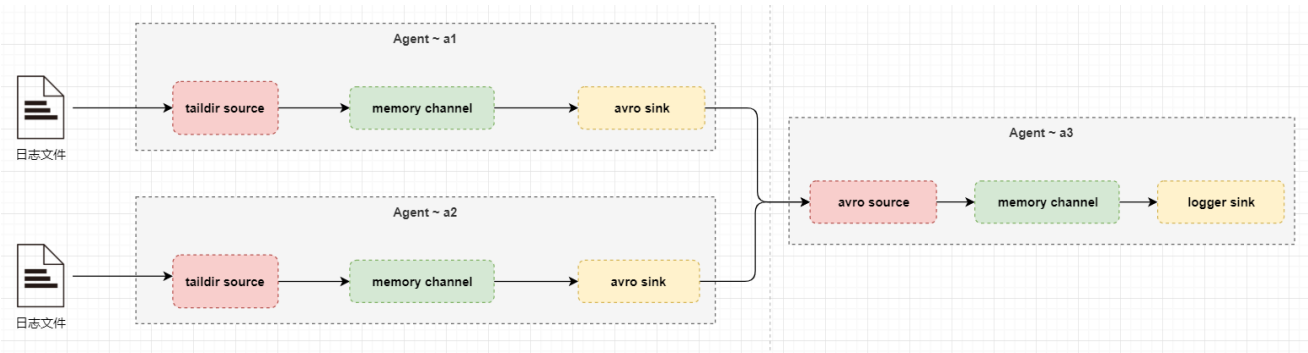
多Source汇总数据到单Flume



1）案例需求：

实现将多个agent的数据发送给一个agent

2）需求分析：



3）实现步骤：

1. 准备工作

```
在flume/job目录下创建group2文件夹
[root@hadoop10 job]# mkdir group2
```

2. 在group2目录下,创建demo6-agent1.conf

```
[root@hadoop10 group2]# touch agent1.conf


# 内容如下
```

```
a1.sources = r1
a1.channels = c1
a1.sinks = k1

a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/data/test1.log

a1.channels.c1.type = memory

a1.sinks.k1.type = avro
a1.sinks.k1.hostname = hadoop10
a1.sinks.k1.port = 6661

a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

3. 在group2目录下,创建agent2.conf

```
[root@hadoop10 group2]# touch agent2.conf

# 内容如下
a2.sources = r1
a2.channels = c1
a2.sinks = k1

a2.sources.r1.type = TAILDIR
a2.sources.r1.filegroups = f1
a2.sources.r1.filegroups.f1 = /opt/data/test2.log

a2.channels.c1.type = memory

a2.sinks.k1.type = avro
a2.sinks.k1.hostname = hadoop10
a2.sinks.k1.port = 6661

a2.sources.r1.channels = c1
a2.sinks.k1.channel = c1
```

4. 在group2目录下,创建agent3.conf

```
[root@hadoop10 group2]# touch agent3.conf

# 内容如下
a3.sources = r1
a3.channels = c1
a3.sinks = k1

a3.sources.r1.type = avro
a3.sources.r1.bind = hadoop10

a3.sources.r1.port = 6661
```

```
a3.channels.c1.type = memory

a3.sinks.k1.type = logger

a3.sources.r1.channels = c1
a3.sinks.k1.channel = c1
```

5. 执行配置文件

```
[root@hadoop10 apache-flume-1.9.0-bin]# bin/flume-ng agent --conf conf --name a3 --conf-file job/group2/agent3.conf -Dflume.root.logger=INFO,console

[root@hadoop10 apache-flume-1.9.0-bin]# bin/flume-ng agent --conf conf --name a2 --conf-file job/group2/agent2.conf -Dflume.root.logger=INFO,console

[root@hadoop10 apache-flume-1.9.0-bin]# bin/flume-ng agent --conf conf --name a1 --conf-file job/group2/agent1.conf -Dflume.root.logger=INFO,console
```

6. 使用echo命令向test1.log和test2.log追加内容

四、拦截器

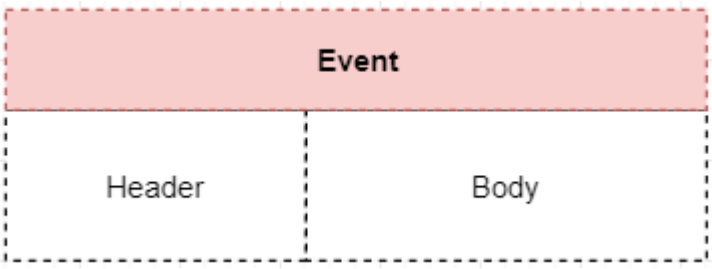
> flume通过使用Interceptors（拦截器）实现修改和过滤事件的功能。举个栗子，一个网站每天产生海量数据，但是可能会有很多数据是不完整的（缺少重要字段），或冗余的，如果不对这些数据进行特殊处理，那么会降低系统的效率。这时候拦截器就派上用场了。

1.flume内置的拦截器

先列个flume内置拦截器的表：

org.apache.flume.interceptor.Interceptor	timestamp	org.apache.flume.interceptor.TimestampInterceptor$Builder
org.apache.flume.interceptor.Interceptor	host	org.apache.flume.interceptor.HostInterceptor$Builder
org.apache.flume.interceptor.Interceptor	static	org.apache.flume.interceptor.StaticInterceptor$Builder
org.apache.flume.interceptor.Interceptor	regex_filter	org.apache.flume.interceptor.RegexFilteringInterceptor$Builder
org.apache.flume.interceptor.Interceptor	regex_extractor	org.apache.flume.interceptor.RegexFilteringInterceptor$Builder

由于拦截器一般针对Event的Header进行处理，那我先介绍一Event吧



- event是flume中处理消息的基本单元，由零个或者多个header和正文body组成。
- Header 是 key/value 形式的，可以用来制造路由决策或携带其他结构化信息(如事件的时间戳或事件来源的服务器主机名)。你可以把它想象成和 HTTP 头一样提供相同的功能——通过该方法来传输正文之外的额外信

息。

- Body是一个字节数组，包含了实际的内容。
- flume提供的不同source会给其生成的event添加不同的header

1.1 timestamp拦截器

> Timestamp Interceptor拦截器就是可以往event的header中插入关键词为timestamp的时间戳。

```
[root@hadoop10 job]# mkdir interceptors
[root@hadoop10 job]# cd interceptors/
[root@hadoop10 interceptors]# touch demo1-timestamp.conf

#文件内容如下
a1.sources = r1
a1.channels = c1
a1.sinks = k1

a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/data/test1.log

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

a1.channels.c1.type = memory

a1.sinks.k1.type = logger

a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

测试

```
[root@hadoop10 data]# echo hello >> test1.log
```

测试结果

```
2020-04-11 03:54:14,179 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:95)] Event: { headers:
{timestamp=1586548451701} body: 68 65 6C 6C 6F
```

1.2 host拦截器

> 该拦截器可以往event的header中插入关键词默认为host的主机名或者ip地址（注意是agent运行的机器的主机名或者ip地址）

```
[root@hadoop10 interceptors]# touch demo2-host.conf
```

```
#文件内容如下
a1.sources = r1
a1.channels = c1
a1.sinks = k1

a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/data/test1.log

#host interceptor
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type = host

a1.channels.c1.type = memory

a1.sinks.k1.type = logger

a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

测试

```
[root@hadoop10 data]# echo aaa >> test1.log
```

测试结果

```
2020-04-11 04:04:09,954 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:95)] Event: { headers:
{host=192.168.150.61} body: 61 61 61                                     aaa }
```

1.3 Regex Filtering Interceptor拦截器 (重要)

> Regex Filtering Interceptor拦截器用于过滤事件，筛选出与配置的正则表达式相匹配的事件。可以用于包含事件和排除事件。常用于数据清洗，通过正则表达式把数据过滤出来。

```
[root@hadoop10 interceptors]# touch demo3-regex-filtering.conf

#文件内容如下
a1.sources = r1
a1.channels = c1
a1.sinks = k1

a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/data/test1.log


#host interceptor
```

```
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type = regex_filter
#全部是数字的数据
a1.sources.r1.interceptors.i1.regex = ^[0-9]*$
#排除符合正则表达式的数据
a1.sources.r1.interceptors.i1.excludeEvents  = true

a1.channels.c1.type = memory

a1.sinks.k1.type = logger

a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

多个拦截器可以同时使用，例如:

```
# 拦截器：作用于Source，按照设定的顺序对event装饰或者过滤

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 = regex_filter
a1.sources.r1.interceptors.i3.regex = ^[0-9]*$
```

五、自定义拦截器

概述：在实际的开发中，一台服务器产生的日志类型可能有很多种，不同类型的日志可能需要发送到不同的分析系统。此时会用到 Flume 拓扑结构中的 Multiplexing 结构，Multiplexing的原理是，根据 event 中 Header 的某个 key 的值，将不同的 event 发送到不同的 Channel中，所以我们需要自定义一个 Interceptor，为不同类型的 event 的 Header 中的 key 赋予不同的值。

案例演示：我们以端口数据模拟日志，以数字（单个）和字母（单个）模拟不同类型的日志，我们需要自定义 interceptor 区分数字和字母，将其分别发往不同的分析系统（Channel）。

实现步骤

1.创建一个项目，并且引入以下依赖

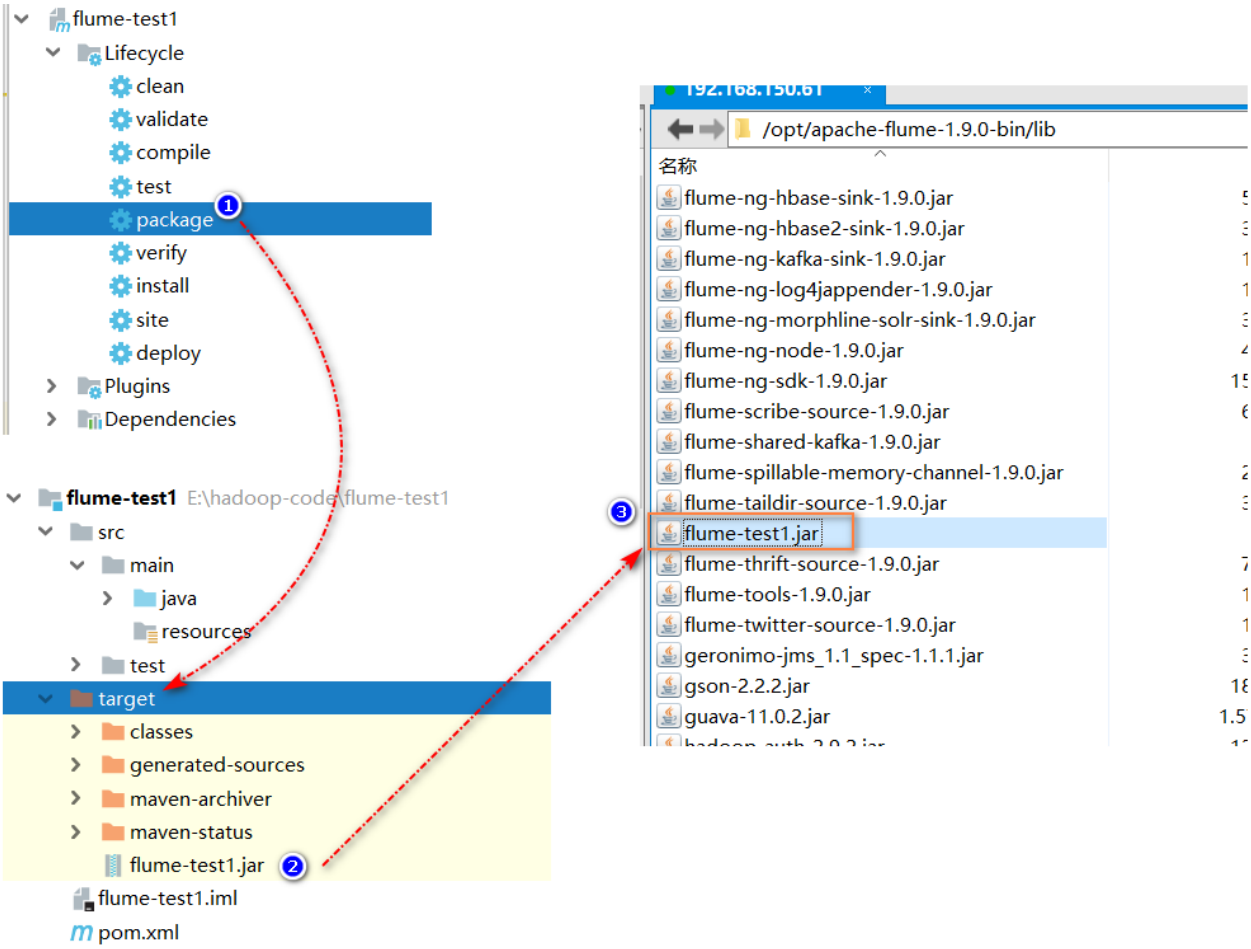
```
<dependency>
    <groupId>org.apache.flume</groupId>
    <artifactId>flume-ng-core</artifactId>
    <version>1.9.0</version>
</dependency>
```

2.自定义拦截器，实现拦截器接口

```
package com.baizhi.interceptors;

import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.interceptor.Interceptor;

import java.util.List;

public class MyInterceptor implements Interceptor {
    @Override
    public void initialize() {

    }

    @Override
    public Event intercept(Event event) {
        byte[] body = event.getBody();
        if (body[0] >= 'a' && body[0] <= 'z'){
            event.getHeaders().put("type","letter");
        }else if (body[0] >= '0' && body[0] <= '9'){
            event.getHeaders().put("type","number");
        }
        return event;
    }

    @Override
    public List<Event> intercept(List<Event> list) {
        for (Event event : list) {
            intercept(event);
        }
        return list;
    }

    @Override
    public void close() {

    }

    public static class Builder implements Interceptor.Builder{

        @Override
        public Interceptor build() {
            return new MyInterceptor();
        }
```

```
        @Override
        public void configure(Context context) {

        }
    }

}
```

3.将项目打成jar包，上传到flume安装目录的lib目录下



4.编写agent,在job目录下的interceptors目录下创建，命名为my.conf

```
a1.sources = r1
a1.channels = c1 c2
a1.sinks = k1 k2

a1.sources.r1.type = TAILDIR
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /opt/data/test1.log

a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type = com.baizhi.interceptors.MyInterceptor$Builder


a1.sources.r1.selector.type = multiplexing
```

```
a1.sources.r1.selector.header = type
a1.sources.r1.selector.mapping.letter = c1
a1.sources.r1.selector.mapping.number = c2
a1.sources.r1.selector.default  = c2

a1.channels.c1.type = memory
a1.channels.c2.type = memory

a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = hdfs://hadoop10:9000/flume/letter
a1.sinks.k1.hdfs.useLocalTimeStamp = true
a1.sinks.k1.hdfs.fileType = DataStream

a1.sinks.k2.type = hdfs
a1.sinks.k2.hdfs.path = hdfs://hadoop10:9000/flume/number
a1.sinks.k2.hdfs.useLocalTimeStamp = true
a1.sinks.k2.hdfs.fileType = DataStream

a1.sources.r1.channels = c1 c2
a1.sinks.k1.channel = c1
a1.sinks.k2.channel = c2
```

5.在/root目录下创建t1、t2文件夹

6.测试

```
[root@hadoop10 apache-flume-1.9.0-bin]# bin/flume-ng agent --conf conf --name a1 --conf-file
job/interceptors/my.conf -Dflume.roogger=INFO,console
```

六、通道选择器

> 在event进入到Channel之前，可以使用通道选择器 使指定的Event进入到指定的Channel中

Flume内置两种选择器，replicating 和 multiplexing，如果Source配置中没有指定选择器，那么会自动使用复制Channel选择器。

1.复制Channel选择器 (replicating)

特点：数据同步给多个Channel

参考：企业开发案例 - 单数据源多出口案例(选择器)

2.多路复用Channel选择器 (multiplexing)

特点：数据分流到指定Channel

参考：自定义拦截器