Day20_数据采集 & flume

Hadoop Day20 数据采集 flume

大数据数据采集

数据来源形式	数据来源格式	文件传输
网络爬虫	html文档	flume、kafka
日志数据	log文件，日志流	flume、kafka
业务数据	关系型数据库	sqoop
传感数据	数据流	kafka

flume

> Flume是Cloudera提供的一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统，Flume支持在日志系统中定制各类数据发送方，用于收集数据；同时，Flume提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力。

对于flume的学习和其他的不太一样，我们需要依赖官网上的文档进行开发，官网用户手册http://flume.apache.org/FlumeUserGuide.html和官网开发手册http://flume.apache.org/FlumeDeveloperGuide.html
flume数据采集的数据来源:

1. log文件
2. 网路端口数据
3. 消息队列数据

flume数据发送来源

1. hdfs
2. hive
3. hbase
4. 网络端口
5. 消息队列

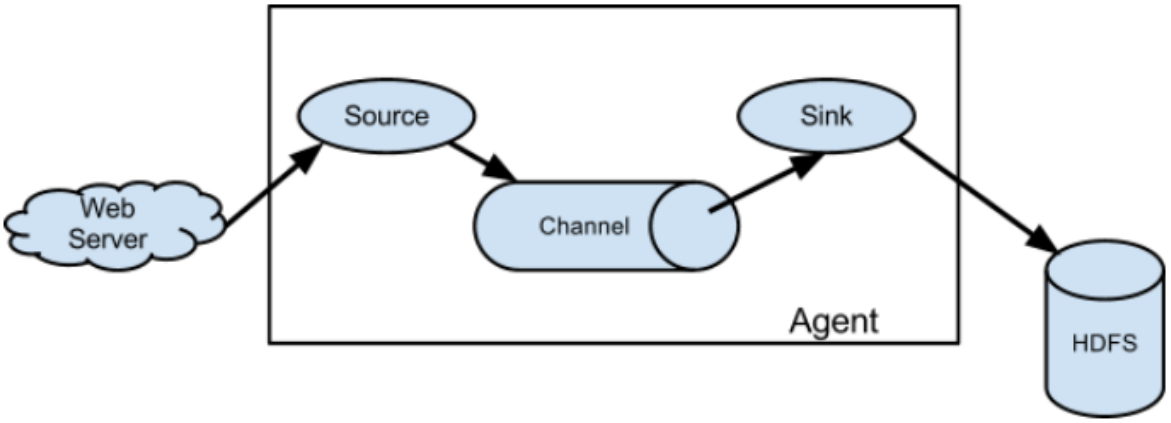
flume-ng将采集的过程交给用户开发agent来直接指定，agent中有三个组件Source、Channel(相当于缓冲区)、sink(目的是从channel中取数据)

- Event：一个数据单元，带有一个可选的消息头
- Flow：Event从源点到达目的点的迁移的抽象
- Client：操作位于源点处的Event，将其发送到Flume Agent
- Agent：一个独立的Flume进程，包含组件Source、Channel、Sink
- Source：用来消费传递到该组件的Event
- Channel：中转Event的一个临时存储，保存有Source组件传递过来的Event
- Sink：从Channel中读取并移除Event，将Event传递到Flow Pipeline中的下一个Agent（如果有的话）

Flume 的核心是把数据从数据源收集过来，再送到目的地。为了保证输送一定成功，在送到目的地之前，会先缓存数据，待数据真正到达目的地后，删除自己缓存的数据。

Flume 传输的数据的基本单位是 Event，如果是文本文件，通常是一行记录，这也是事务的基本单位。Event 从 Source，流向 Channel，再到 Sink，本身为一个 byte 数组，并可携带 headers 信息。Event 代表着一个数据流的最小完整单元，从外部数据源来，向外部的目的地去。

Flume 运行的核心是 Agent。它是一个完整的数据收集工具，含有三个核心组件，分别是 source、channel、sink。通过这些组件，Event 可以从一个地方流向另一个地方，如下图所示。



- source 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。
- channel 是一个存储地，接收 source 的输出，直到有 sink 消费掉 channel 中的数据。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，可以自动重启，不会造成数据丢失，因此很可靠。
- sink 会消费 channel 中的数据，然后送给外部源或者其他 source。如数据可以写入到 HDFS 或者 HBase 中。

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

a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444

a1.sinks.s1.type = logger

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

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

netcat:source的一种类型，网络抓取
bind：资源来源

flume的整个数据采集过程，数据是被封装到Event里面进行传输的

采集日志log文件，把结果存放到hdfs上

log生成的方式

1. 滚动生成
    - 按时间滚动
    - 按文件大小滚动

当有新日志文件产生的时候，把刚写完的日志系统拷贝搭配spoolingdirectory

spooling Source hdfs sink

source ：Spooling Directory Source
channel：memory
sink ： hdfs sink

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

a1.sources.r1.type = spooldir
a1.sources.r1.spoolDir = /root/tmp
a1.sources.r1.fileHeader = true

a1.sinks.s1.type = hdfs
a1.sinks.s1.hdfs.path = hdfs://master:9000/flumelog/%Y%m%d
a1.sinks.s1.hdfs.fileSuffix = .log
a1.sinks.s1.hdfs.rollInterval = 0
a1.sinks.s1.hdfs.rollSize = 0
a1.sinks.s1.hdfs.rollCount = 100
a1.sinks.s1.hdfs.fileType = DataStream
a1.sinks.s1.hdfs.writeFormat = Text
a1.sinks.s1.hdfs.useLocalTimeStamp = true

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

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

执行操作`flume-ng agent -c conf -f flume_nc_to_hdfs.conf --name a1 -Dflume.root.logger=INFO.console`

avro source logger sink

> avro文件作为数据采集的对象，logger作为消费者消费的格式
> 现在我们模拟出avro文件的格式发送数据到agent中，下面的java程序就是模拟代码

```
import java.nio.charset.Charset;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

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;

// 连接avro的flume source，又发送event到flume agent
public class FlumeClient {
    private RpcClient flumeClient;
    private String hostname;
    private int port;

    public FlumeClient() {
        super();
    }

    // 初始化连接
    public FlumeClient(String hostname, int port) {
        this.hostname = hostname;
        this.port = port;
        this.flumeClient = RpcClientFactory.getDefaultInstance(host
name, port);
    }

    // 将event消息发送到avro source
    public void sendEvent(String msg){
        // 构建event的header
        Map<String,String> headers = new HashMap<>();
        headers.put("timestamp", String.valueOf(new Date().getTim
e()));
        // 构建event
        Event event = EventBuilder.withBody(msg, Charset.forName("U
TF-8"), headers);

        try {
            flumeClient.append(event);
        } catch (EventDeliveryException e) {
            e.printStackTrace();
            // 单条发送失败重新连接
            flumeClient.close();
            flumeClient = null;
            flumeClient = RpcClientFactory.getDefaultInstance(hostn
ame, port);
```

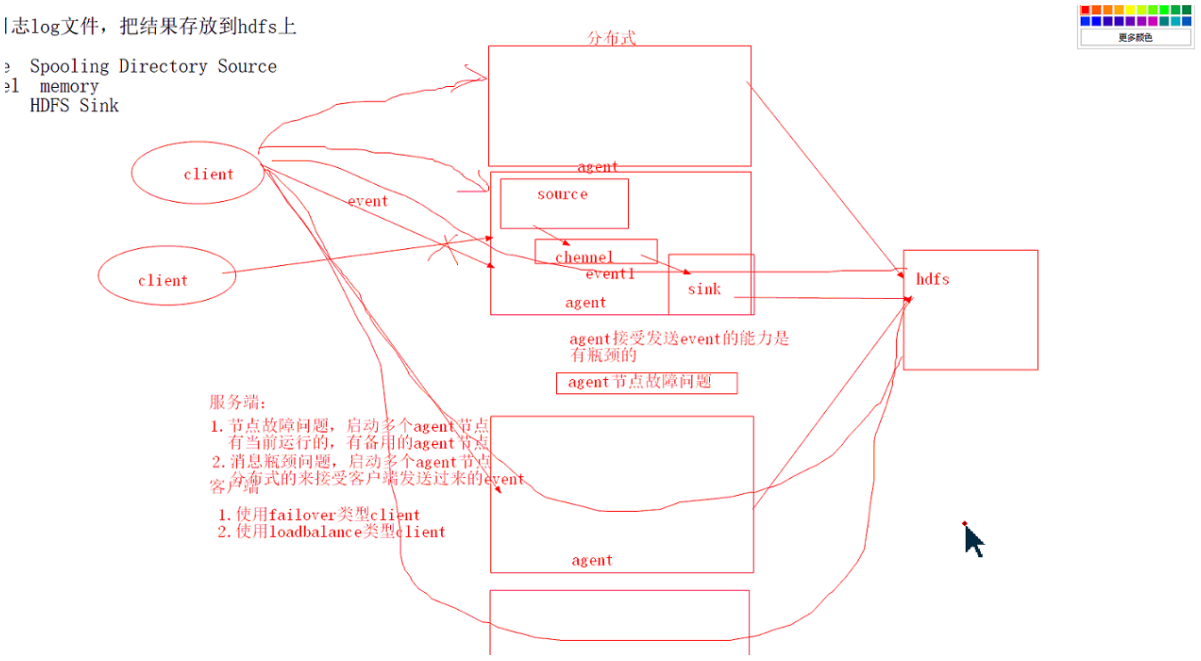
```
        }
    }
    // 发送flume连接
    public void cleanUp(){
        flumeClient.close();
    }

    public static void main(String[] args) {
        FlumeClient flumeClient = new FlumeClient("master",8888);
        String bMsg = "fromjava-msg";
        for(int i = 0; i < 100; i++){
            flumeClient.sendEvent(bMsg + i);
        }
        flumeClient.cleanUp();
    }
}
```

> 下面是avro—> logger的配置文件

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

a1.sources.r1.type = avro
a1.sources.r1.bind = master
a1.sources.r1.port = 8888

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

a1.sinks.s1.type = logger

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

flume节点故障和消息瓶颈问题

flume 节点在运行的时候可能会挂掉，导致我们的数据无法正常的进行采集，在生活中我们是不愿意出现这种现象的。我们采用分布式的方式来解决节点故障问题，同时打开多个节点，如果出现节点挂掉现象，其他的节点随时待命接收数据，处理日志信息。flume在运行的时候也会出现channel中出现了很多的event，但是sink消费的比较缓慢，慢慢的会出现channel溢出的现象，这也不是我们想要看到的现象，对于这个问题我们也采用分布式进行解决。我们可以将event发送到多个节点上，多个节点同时读取数据，这样就不会出现channel溢出的现象。下面以程序的方式说明这两个问题。



- 简要解决方案

服务端：

1. 节点故障问题，启动多个agent节点，由当前运行的，有备用的agent节点
2. 消息瓶颈问题，启动多个agent节点，分布式的来接受客户端发送的event

客户端

1. 使用failover类型client
2. 使用loadbalance类型client

节点故障问题

节点故障问题，服务端采用多个节点同时运行的方式进行，有当前节点运行，其他节点备用;客户端采用failover类型

- 客户端

创建fileover_client.conf文件，这里我们配置了连个节点，master是主节点 ，slaver1是备用节点

```
client.type = default_failover

hosts = h1 h2

hosts.h1 = master:8888
hosts.h2 = slaver1:8888

max-attempts = 2
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.charset.Charset;
import java.util.Properties;

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;

public class FileoverClient {

    private Properties properties;
    private RpcClient failoverClient;

    // 初始化fileover
    public FileoverClient() throws Exception {
        this.properties = new Properties();
        InputStream inputStream = new FileInputStream(new File("sr
c/main/resources/fileover_client.conf"));
        properties.load(inputStream);
        this.failoverClient = RpcClientFactory.getInstance(properti
es);
    }

    // 发送消息
    public void sendEvent(String msg){
        Event event = EventBuilder.withBody(msg, Charset.forName("U
TF-8"));
        try {
            failoverClient.append(event);
        } catch (EventDeliveryException e) {
            e.printStackTrace();
        }
    }
    public void close(){
        failoverClient.close();
    }

    public static void main(String[] args) throws Exception {
        FileoverClient fileoverClient = new FileoverClient();
        String msg = "message_";
        for(int i = 0; i < 100; i++){
            fileoverClient.sendEvent(msg+i);
            Thread.sleep(1000);
        }
```

```
            fileoverClient.close();
        }
    }
```

- 服务端
  服务端运行多个flume程序就可以了，记住启动备用节点

消息瓶颈问题

> 消息瓶颈问题，服务端启动多个agent节点，分布式的来接受客户端发送的event。
> 客户端使用loadbalance类型

- 配置文件

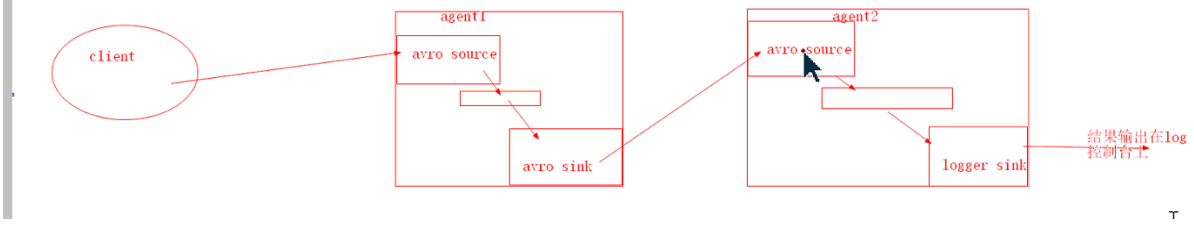
```
client.type = default_loadbalance

hosts = h1 h2

hosts.h1 = master:8888
hosts.h2 = slaver1:8888

host-selector = random
```

- 发送程序，我们这里采用随机发送

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.Charset;
import java.util.Properties;

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;

public class LoadBalanceClient {
    private RpcClient lbClient;
    private Properties properties;
    public LoadBalanceClient() throws Exception {
        this.properties = new Properties();
        InputStream inputStream = new FileInputStream(new File("sr
c/main/resources/load_balance.conf"));
        properties.load(inputStream);
        this.lbClient = RpcClientFactory.getInstance(properties);
    }

    public void sendEvent(String msg){
        Event event = EventBuilder.withBody(msg, Charset.forName("U
TF-8"));
        try {
            lbClient.append(event);
        } catch (EventDeliveryException e) {
            e.printStackTrace();
        }
    }

    public void close(){
        lbClient.close();
    }

    public static void main(String[] args) throws Exception {
        LoadBalanceClient loadBalanceClient = new LoadBalanceClien
t();
        String msg = "lbmsg_";
        for(int i = 0; i < 100; i++){
            loadBalanceClient.sendEvent(msg + i);
        }
        loadBalanceClient.close();
    }
}
```

多个节点串行化操作

> 需求分析：我们的需求是客户端以avro的方式将数据发送出去。节点一以avro的方式输出，然后将输出的avro文件发送给第二个节点，第二个节点将数据打印到控制台。当然，你也可以将数据保存到hdfs上或者hive等等



1. 书写第一个节点的服务端程序

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

a1.sources.r1.type = avro
a1.sources.r1.bind = master
a1.sources.r1.port = 9999

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

a1.sinks.s1.type = avro
a1.sinks.s1.hostname = slaver1
a1.sinks.s1.port = 9998

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

2. 书写第二个服务端程序

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

a1.sources.r1.type = avro
a1.sources.r1.bind = slaver1
a1.sources.r1.port = 9998

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

a1.sinks.s1.type = logger

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

3. 使用代码模拟产生avro文件，将文件发送到第一个节点上

```
import java.nio.charset.Charset;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

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;

// 连接avro的flume source，又发送event到flume agent
public class FlumeClient {
    private RpcClient flumeClient;
    private String hostname;
    private int port;

    public FlumeClient() {
        super();
    }

    // 初始化连接
    public FlumeClient(String hostname, int port) {
        this.hostname = hostname;
        this.port = port;
        this.flumeClient = RpcClientFactory.getDefaultInstance(host
name, port);
    }

    // 将event消息发送到avro source
    public void sendEvent(String msg){
        // 构建event的header
        Map<String,String> headers = new HashMap<>();
        headers.put("timestamp", String.valueOf(new Date().getTim
e()));
        // 构建event
        Event event = EventBuilder.withBody(msg, Charset.forName("U
TF-8"), headers);

        try {
            flumeClient.append(event);
        } catch (EventDeliveryException e) {
            e.printStackTrace();
            // 单条发送失败重新连接
            flumeClient.close();
            flumeClient = null;
            flumeClient = RpcClientFactory.getDefaultInstance(hostn
ame, port);
```

```
            }
        }
        // 发送flume连接
        public void cleanUp(){
            flumeClient.close();
        }

        public static void main(String[] args) {
            FlumeClient flumeClient = new FlumeClient("master",9999);
            String bMsg = "fromjava-msg";
            for(int i = 0; i < 100; i++){
                flumeClient.sendEvent(bMsg + i);
            }
            flumeClient.cleanUp();
        }
    }
```

4. 如果在第二个节点上的控制台上看到我们发送的数据，说明我们是成功的