# 日志收集模块

## 1 网站指标分析

### Pv 网页浏览量

或者点击量。通常是衡量一个网站的主要指标，用户对网页进行一个刷新，pv+1

### **uv 独立访客数量**

一天只能有一个uv。通过对浏览器端设置一个特殊的cookie，通过cookie来唯一表示这个用户

### **vv ：Vist View 访客的会话次数**

记录所有用户1天内访问了多少次网站，一天内可以多次访问。可以使用cookie实现，一个cookie在不指定setPath时，默认只能访问发送cookie的路径及其子孙路径。

### **Bounce Rate 跳出**率

如果一个用户进入一个网站的页面，而没有点击网站的其他页面就离开了，即为跳出。说明用户进入网站后是否会网站的内容产生了兴趣。跳出率越低越好

跳出率=只访问一个网页的会话/会话总数

### **新增独立IP**

一段时间内访问网站的ip是以前没出现过的ip，指的是公网ip，而不是网络地址协议(NAT)转换来的内网ip,所以这个ip只能在服务器端获取

### **平均会话时长**

指人们访问网站平均停留的时间。一般的统计工具只能统计到访客访问网站的倒数第二个网页离开的精确时间，所以一般访客的停留时间都小于真实时间。

### **新增独立访客**

当日独立访客中，历史上首次出现的访客为独立访客

### **访问深度**

访客的去重的访问页面数量

## 2 js埋点

我们需要收集某个第三方访站的访问情况，通用的做法是在第三方网站的相关页面嵌入一个JS脚本，这样当用户访问该页面时，页面上的JS脚本会在页面上动态加入一个<image/>标签，并且<image/>的src属性指向日志服务器下的一个透明图片的URL地址，在该URL参数上会附带第三方网站的一些信息（如被访问页面的URL、识别用户身份的cookie等）,这样通过分析日志服务器（一般是nginx或apache）的access日志文件就能获取到这些信息。这个JS埋点的脚本可以自己编写来实现，代码片断如下：

### Js代码

|  |
| --- |
| **function** *ar\_encode*(str){  **return** encodeURI(str); } */\*屏幕分辨率\*/* **function** *ar\_get\_screen*(){  **var** c=**""**;  **if**(**self**.**screen**){  c=**screen**.**width**+**"x"**+**screen**.**height**;  }  **return** c; } */\*颜色质量\*/* **function** *ar\_get\_color*(){  **var** c=**""**;  **if**(**self**.**screen**){  c=**screen**.**colorDepth**+**"-bit"**;  }  **return** c; } */\*\*返回当前的浏览器语言\*/* **function** *ar\_get\_language*() {  **var** l = **""**;  **var** n = **navigator**;   **if** (n.**language**) {  l = n.**language**.toLowerCase();  }  **else  if** (n.browserLanguage) {  l = n.browserLanguage.toLowerCase();  }   **return** l; } */\*返回浏览器类型\*/* **function** *ar\_get\_agent*(){  **var** a=**""**;  **var** n=**navigator**;  **if**(n.**userAgent**){  a= n.**userAgent**;  }  **return** a; } */\*\*方法可返回一个布尔值，该值指示浏览器是否支持并启用了Java\*/* **function** *ar\_get\_jvm\_enabled*() {  **var** j = **""**;  **var** n = **navigator**;   j = n.javaEnabled() ? 1 : 0;   **return** j; }  */\*\*返回浏览器是否支持(启用)cookie \*/* **function** *ar\_get\_cookie\_enabled*() {  **var** c = **""**;  **var** n = **navigator**;  c = n.**cookieEnabled** ? 1 : 0;   **return** c; }  */\*\*检测浏览器是否支持Flash或有Flash插件\*/* **function** *ar\_get\_flash\_ver*() {  **var** f=**""**,n=**navigator**;   **if** (n.**plugins** && n.**plugins**.**length**) {  **for** (**var** ii=0;ii<n.**plugins**.**length**;ii++) {  **if** (n.**plugins**[ii].name.indexOf(**'Shockwave Flash'**)!=-1) {  f=n.**plugins**[ii].description.split(**'Shockwave Flash '**)[1];  **break**;  }  }  }  **else  if** (**window**.ActiveXObject) {  **for** (**var** ii=10;ii>=2;ii--) {  **try** {  **var** fl=eval(**"new ActiveXObject('ShockwaveFlash.ShockwaveFlash."**+ii+**"');"**);  **if** (fl) {  f=ii + **'.0'**;  **break**;  }  }  **catch**(e) {}  }  }  **return** f; }   */\*\*匹配顶级域名\*/* **function** *ar\_c\_ctry\_top\_domain*(str) {  **var** pattern = **"/^aero$|^cat$|^coop$|^int$|^museum$|^pro$|^travel$|^xxx$|^com$|^net$|^gov$|^org$|^mil$|^edu$|^biz$|^info$|^name$|^ac$|^mil$|^co$|^ed$|^gv$|^nt$|^bj$|^hz$|^sh$|^tj$|^cq$|^he$|^nm$|^ln$|^jl$|^hl$|^js$|^zj$|^ah$|^hb$|^hn$|^gd$|^gx$|^hi$|^sc$|^gz$|^yn$|^xz$|^sn$|^gs$|^qh$|^nx$|^xj$|^tw$|^hk$|^mo$|^fj$|^ha$|^jx$|^sd$|^sx$/i"**;   **if**(str.match(pattern)){ **return** 1; }   **return** 0; } */\*\*处理域名地址\*/* **function** *ar\_get\_domain*(host) {  *//如果存在则截去域名开头的 "www."* **var** d=host.replace(/^www\./, **""**);   *//剩余部分按照"."进行split操作，获取长度* **var** ss=d.split(**"."**);  **var** l=ss.**length**;   *//如果长度为3，则为xxx.yyy.zz格式* **if**(l == 3){  *//如果yyy为顶级域名，zz为次级域名，保留所有* **if**(*ar\_c\_ctry\_top\_domain*(ss[1]) && ar\_c\_ctry\_domain(ss[2])){  }  *//否则只保留后两节* **else**{  d = ss[1]+**"."**+ss[2];  }  }  *//如果长度大于3* **else if**(l >= 3){   *//如果host本身是个ip地址，则直接返回该ip地址为完整域名* **var** ip\_pat = **"^[0-9]\*\.[0-9]\*\.[0-9]\*\.[0-9]\*$"**;  **if**(host.match(ip\_pat)){  **return** d;  }  *//如果host后两节为顶级域名及次级域名，则保留后三节* **if**(*ar\_c\_ctry\_top\_domain*(ss[l-2]) && ar\_c\_ctry\_domain(ss[l-1])) {  d = ss[l-3]+**"."**+ss[l-2]+**"."**+ss[l-1];  }  *//否则保留后两节* **else**{  d = ss[l-2]+**"."**+ss[l-1];  }  }   **return** d; }  */\*\*返回cookie信息\*/* **function** *ar\_get\_cookie*(name) {  **var** mn=name+**"="**;  **var** b,e;  **var** co=**document**.**cookie**;   **if** (mn==**"="**) {  **return** co;  }   b=co.indexOf(mn);   **if** (b < 0) {  **return ""**;  }   e=co.indexOf(**";"**, b+name.**length**);   **if** (e < 0) {  **return** co.substring(b+name.**length** + 1);  }  **else** {  **return** co.substring(b+name.**length** + 1, e);  } }  */\*\*设置cookie信息\*/* **function** *ar\_set\_cookie*(name, val, cotp) {  **var** date=**new** Date;  **var** year=date.getFullYear();  **var** hour=date.getHours();   **var** cookie=**""**;   **if** (cotp == 0) {  cookie=name+**"="**+val+**";"**;  }  **else if** (cotp == 1) {  year=year+10;  date.setYear(year);  cookie=name+**"="**+val+**";expires="**+date.toGMTString()+**";"**;  }  **else if** (cotp == 2) {  hour=hour+1;  date.setHours(hour);  cookie=name+**"="**+val+**";expires="**+date.toGMTString()+**";"**;  }   **var** d=*ar\_get\_domain*(**document**.**domain**);  **if**(d != **""**){  cookie +=**"domain="**+d+**";"**;  }  cookie +=**"path="**+**"/;"**;   **document**.**cookie**=cookie; } */\*\*返回客户端时间\*/* **function** *ar\_get\_stm*() {  **return new** Date().getTime(); }  */\*\*返回指定个数的随机数字串\*/* **function** *ar\_get\_random*(n) {  **var** str = **""**;  **for** (**var** i = 0; i < n; i ++) {  str += String(parseInt(**Math**.random() \* 10));  }  **return** str; } */\* main function \*/* **function** *ar\_main*() {   **var** dest\_path = **"http://127.0.0.1:8080/servlet/LogServlet?"**;  **var** expire\_time = 30 \* 60 \* 1000;*//会话超时时长   //处理uv  //--获取cookie ar\_stat\_uv的值* **var** uv\_str = *ar\_get\_cookie*(**"ar\_stat\_uv"**);  **var** uv\_id = **""**;  *//--如果cookie ar\_stat\_uv的值为空* **if** (uv\_str == **""**){  *//--为这个新uv配置id，为一个长度20的随机数字* uv\_id = *ar\_get\_random*(20);  *//--设置cookie ar\_stat\_uv 保存时间为10年  ar\_set\_cookie*(**"ar\_stat\_uv"**, uv\_id, 1);  }  *//--如果cookie ar\_stat\_uv的值不为空* **else**{  *//--获取uv\_id* uv\_id = uv\_str;  }   *//处理ss  //--获取cookie ar\_stat\_ss* **var** ss\_str = *ar\_get\_cookie*(**"ar\_stat\_ss"**);  **var** ss\_id = **""**; *//sessin id* **var** ss\_no = 0; *//session有效期内访问页面的次数   //--如果cookie中不存在ar\_stat\_ss 说明是一次新的会话* **if** (ss\_str == **""**){  *//--随机生成长度为10的session id* ss\_id = *ar\_get\_random*(10);  *//--session有效期内页面访问次数为0* ss\_no = 0;  *//--拼接cookie ar\_stat\_ss 值 格式为 会话编号\_会话期内访问次数\_客户端时间\_网站id* **value** = ss\_id+**"\_"**+ss\_no+**"\_"**+*ar\_get\_stm*();  *//--设置cookie ar\_stat\_ss  ar\_set\_cookie*(**"ar\_stat\_ss"**, **value**, 0);  }  *//--如果cookie中存在ar\_stat\_ss* **else** {  *//获取ss相关信息* **var** items = ss\_str.split(**"\_"**);  *//--ss\_id* **var** cookie\_ss\_id = items[0];  *//--ss\_no* **var** cookie\_ss\_no = parseInt(items[1]);  *//--ss\_stm* **var** cookie\_ss\_stm = items[2];   *//如果当前时间-当前会话上一次访问页面的时间>30分钟,虽然cookie还存在，但是其实已经超时了！仍然需要重新生成cookie* **if** (*ar\_get\_stm*() - cookie\_ss\_stm > expire\_time) {  *//--重新生成会话id* ss\_id = *ar\_get\_random*(10);  *//--设置会话中的页面访问次数为0* ss\_no = 0;  }  *//--如果会话没有超时* **else**{  *//--会话id不变* ss\_id = cookie\_ss\_id;  *//--设置会话中的页面方位次数+1* ss\_no = cookie\_ss\_no + 1;  }   *//--重新拼接cookie ar\_stat\_ss的值* **value** = ss\_id+**"\_"**+ss\_no+**"\_"**+*ar\_get\_stm*();  *ar\_set\_cookie*(**"ar\_stat\_ss"**, **value**, 0);  }    *//返回导航到当前网页的超链接所在网页的URL* **var** ref = **document**.**referrer**;  ref = *ar\_encode*(String(ref));   *//当前地址* **var** url = **document**.**URL**;  url = *ar\_encode*(String(url));   *//当前资源名* **var** urlname = **document**.**URL**.substring(**document**.**URL**.lastIndexOf(**"/"**)+1);  urlname = *ar\_encode*(String(urlname));   *//网页标题* **var** title = **document**.**title**;  title = *ar\_encode*(String(title));   *//网页字符集* **var** charset = **document**.**charset**;  charset = *ar\_encode*(String(charset));   *//屏幕信息* **var** screen = *ar\_get\_screen*();  screen = *ar\_encode*(String(screen));   *//颜色信息* **var** color =*ar\_get\_color*();  color =*ar\_encode*(String(color));   *//语言信息* **var** language = *ar\_get\_language*();  language = *ar\_encode*(String(language));   *//浏览器类型* **var** agent =*ar\_get\_agent*();  agent =*ar\_encode*(String(agent));   *//浏览器是否支持并启用了java* **var** jvm\_enabled =*ar\_get\_jvm\_enabled*();  jvm\_enabled =*ar\_encode*(String(jvm\_enabled));   *//浏览器是否支持并启用了cookie* **var** cookie\_enabled =*ar\_get\_cookie\_enabled*();  cookie\_enabled =*ar\_encode*(String(cookie\_enabled));   *//浏览器flash版本* **var** flash\_ver = *ar\_get\_flash\_ver*();  flash\_ver = *ar\_encode*(String(flash\_ver));   *//当前uv状态 格式为"会话id\_会话次数\_当前时间"* **var** stat\_ss = ss\_id+**"\_"**+ss\_no+**"\_"**+*ar\_get\_stm*();   *//拼接访问地址 增加如上信息* **dest**=dest\_path+**"url="**+url+**"&urlname="**+urlname+**"&title="**+title+**"&chset="**+charset+**"&scr="**+screen+**"&col="**+color+**"&lg="**+language+**"&je="**+jvm\_enabled+**"&ce="**+cookie\_enabled+**"&fv="**+flash\_ver+**"&cnv="**+String(**Math**.random())+**"&ref="**+ref+**"&uagent="**+agent+**"&stat\_uv="**+uv\_id+**"&stat\_ss="**+stat\_ss;  *//通过插入图片访问该地址* **document**.getElementsByTagName(**"body"**)[0].**innerHTML** += **"<img src=\""**+**dest**+**"\" border=\"0\" width=\"1\" height=\"1\" />"**;  }  **window**.onload = **function**(){  *//触发main方法  ar\_main*(); } |

### Servlet

|  |
| --- |
| @WebServlet(name = **"LogServlet"**,urlPatterns={**"/servlet/LogServlet"**}) **public class** LogServlet **extends** HttpServlet {  **private static** Logger *logger*=Logger.*getLogger*(LogServlet.**class**);  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request,response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  *//获取所有参数* String str=request.getQueryString();  *//进行url解码* str= URLDecoder.*decode*(str,**"utf-8"**);  *//将str进行处理变成将值用|分割形式* StringBuffer buffer=**new** StringBuffer();  String [] kvs=str.split(**"&"**);  **for**(String kv:kvs){  String value=kv.split(**"="**).**length**>=2?kv.split(**"="**)[1]:**""**;  buffer.append(value+**"|"**);  }  *//拼接ip* String ip=request.getRemoteAddr();  buffer.append(ip);  str=buffer.toString();  *logger*.info(str);  } } |

此外这个js埋点的脚本我们也可以考虑用第三方开源的产品，这样这些开源产品会自动帮助我们出分析结果，这样的产品有：

谷歌的Analytics ，其源码地址：<https://github.com/googleanalytics> ，

使用说明：<https://developers.google.com/analytics/devguides/collection/analyticsjs/>

腾讯云分析平台，http://mta.qq.com/

## 3 flume收集日志

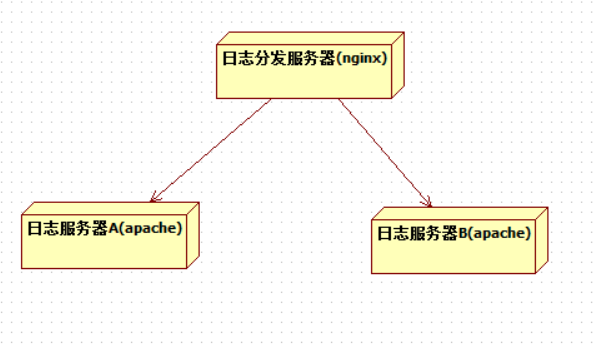
用logf4 appender把日志直接发送到flume的avro source

需要flume-ng-sdk.1.6.0Jar，logf4.jar

4

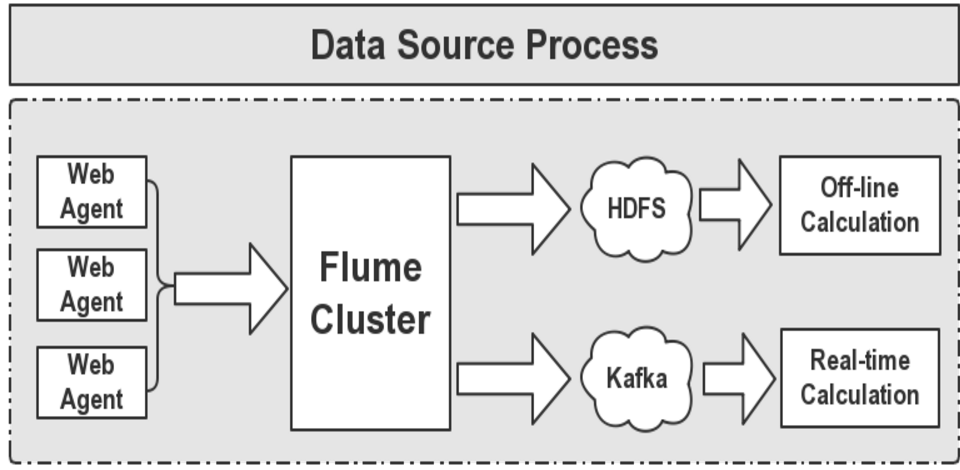
1. **日志服务器搭建**

在实际应用中，日志服务器应为多台，部署成负载均衡的方式。其中nginx作为分发器，用户的日志请求会发送到nginx之上，然后再通过其分发到一台日志服务器。日志服务器采用apache来搭建，这是考虑到apache的模块对日志的切割支持得比较好。



1. **日志收集**

Apache等Web服务器中的访问日志文件默认情况下的路径是/var/log/httpd/access\_log，可以通过/etc/httpd/conf/httpd.conf进行配置。默认情况下，apache会把用户的访问记录不断写入到access\_log文件中，这们就不方便flume对日志进行收集，所以根据实际业务需要，我们可以制定两种日志收集策略：实时收集和定时收集。



首先要先安装Flume ，下载地址：<http://www.apache.org/dyn/closer.lua/flume/1.6.0/apache-flume-1.6.0-bin.tar.gz>

下载后解压后即完成安装

### 实时收集

需要实时监控日志文件access\_log文件的变化，在linux下可以使用tail命令来达到这个目的。

flume.conf配置：

tailAgent.sources = execSrc

tailAgent.channels = memoryChannel

tailAgent.sinks = loggerSink

tailAgent.sources.execSrc.type = exec

tailAgent.sources.execSrc.channels = memoryChannel

tailAgent.sources.execSrc.command = tail -F /var/log/httpd/access\_log

tailAgent.sinks.loggerSink.type = logger

tailAgent.sinks.loggerSink.channel = memoryChannel

tailAgent.channels.memoryChannel.type = memory

tailAgent.channels.memoryChannel.capacity = 100

启动tailAgent的命令：

{$FLUME\_HOME}/bin/flume-ng agent --conf ../conf/ -f ../conf/flume.conf -Dflume.root.logger=DEBUG,console -n tailAgent

这样每当/var/log/httpd/access\_log文件中出现追加内容时，flume都能实时捕捉到

### 定时收集

需要按照定时的周期对日志进行切割，当前需求下也就是每小时需要产生一个日志文件。然后配置flume监控日志文件所在的目录。每当目录下出现新的日志文件，都会被flume的agent发送到HDFS的系统上。下面对这几个步骤依次进行分析：

* + 日志切割

apache服务器大约每响应一万次请求，access log文件会增长1M，这样这个access\_log文件就会逐渐变得很大。针对这个情况，Apache Server提供了一个Log Rotation功能，能够定期产生新的日志文件。在实际应用时Apache httpd能够通过管道的方式来调用Log Rotation进程。具体需在/etc/httpd/conf/httpd.conf文件中进行配置，

<IfModule log\_config\_module>

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined //定义日志格式和别名combined

LogFormat "%h %l %u %t \"%r\" %>s %b" common //定义日志格式和别名common

<IfModule logio\_module>

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combinedio

</IfModule>

#CustomLog "logs/access\_log" common

CustomLog "logs/access\_log" combined //CustomLog是配置日志生效的指令

</IfModule>

上面是apache的默认配置，为达到切割效果，CustomLog指定的配置要改成下面这行：

**CustomLog "|/usr/sbin/rotatelogs -c -L /etc/httpd/logs/access\_log -p /etc/httpd/logs/log-executor.sh logs/access\_log\_%Y\_%m\_%d\_%H\_%M.log 3600 480" combined**

在上述配置中3600代表秒，表示每隔一小时生成一个日志文件。480表示相对于UTC的时差的分钟数。如果省略，则假定为"0"并使用UTC时间

-c表示表示如果没有请求时也会创建空的日志文件

-p 表示在产生新的日志文件时rotatelogs会调用我们指定的程序，并且会为这个程序隐含的传两个参数，第

一个参数是生成的新生成的日志文件的名字，第二个参数是上次生成的日志文件的名字，利用这一点，

我们可以写一个脚本来实现把前一个已完成的日志文件在此时copy到一个与flume采集相关的目录中，

以便flume进行采集，我们在此处插入的脚本代码如下：

newLog=$1

oldLog=$2

sourfile="$oldLog"

destfile="/user/root/logs/"

cp $sourfile $destfile

**注意事项**：chmod o+w /var/log/httpd/ 要把日志目录的可写权限打开，这个写入是由apache组的用户在执行。

* + flume收集配置

a1.channels **=** ch-1

a1.sources **=** src-1

a1.sources.src-1.type **=** spooldir

a1.sources.src-1.channels **=** ch-1

a1.sources.src-1.spoolDir **=** /var/log/apache/flumeSpool

a1.sources.src-1.fileHeader **=** true

启动flume：

./flume-ng agent --conf ../conf --conf-file ../conf/flux.conf --name a1 -Dflume.root.logger=INFO,console

### 扇入收集的conf：

Hadoop01

|  |
| --- |
| a1.sources=r1  a1.sinks=k1  a1.channels=c1  a1.sources.r1.type=avro  a1.sources.r1.bind=0.0.0.0  a1.sources.r1.port=44444  a1.sources.r1.interceptors=i1  a1.sources.r1.interceptors.i1.type=regex\_extractor  a1.sources.r1.interceptors.i1.regex = ^.\*?\_\\d\_(.+?)\\|.\*$  a1.sources.r1.interceptors.i1.serializers = s1  a1.sources.r1.interceptors.i1.serializers.s1.name = timestamp  a1.sinks.k1.type = hdfs  a1.sinks.k1.hdfs.path = hdfs://hadoop01:9000/flux/reportTime=%Y-%m-%d  a1.sinks.k1.hdfs.fileType = DataStream  #a1.sinks.k1.hdfs.filePrefix = %Y-%m-%d-%H-%M-%S  a1.sinks.k1.hdfs.fileSuffix = .data  a1.sinks.k1.hdfs.rollInterval=0  a1.sinks.k1.hdfs.rollSize=104857600  a1.sinks.k1.hdfs.rollCount=0  a1.channels.c1.type=memory  a1.channels.c1.capacity=1000  a1.channels.c1.transactionCapacity=100  a1.sources.r1.channels=c1  a1.sinks.k1.channel=c1 |

Hadoop02，hadoop03

|  |
| --- |
| 1.sources=r1  a1.sinks=k1  a1.channels=c1  a1.sources.r1.type=avro  a1.sources.r1.bind=0.0.0.0  a1.sources.r1.port=44444  a1.sinks.k1.type=avro  a1.sinks.k1.hostname = 192.168.8.151  a1.sinks.k1.port = 44444  a1.channels.c1.type=memory  a1.channels.c1.capacity=1000  a1.channels.c1.transactionCapacity=100  a1.sources.r1.channels=c1  a1.sinks.k1.channel=c1 |

./flume-ng agent --conf ../conf --conf-file ../conf/flux.conf --name a1 -Dflume.root.logger=INFO,console

Kafka有ack fail机制，不会自动ack

### Log4j链接flume：

|  |
| --- |
| **log4j.rootLogger** = **INFO,stdout,D,flume  log4j.appender.stdout** = **org.apache.log4j.ConsoleAppender log4j.appender.stdout.Target** = **System.out log4j.appender.stdout.layout** = **org.apache.log4j.PatternLayout log4j.appender.stdout.layout.ConversionPattern** = **%m%n  log4j.appender.D** = **org.apache.log4j.DailyRollingFileAppender log4j.appender.D.File** = **/logs/log.log log4j.appender.D.Append** = **true log4j.appender.D.Threshold** = **Info log4j.appender.D.layout** = **org.apache.log4j.PatternLayout log4j.appender.D.layout.ConversionPattern** = **%m%n  log4j.appender.flume** = **org.apache.flume.clients.log4jappender.Log4jAppender log4j.appender.flume.Hostname** =**192.168.8.152 log4j.appender.flume.Port** = **44444 log4j.appender.flume.UnsafeMode** = **true** |

# Hive清洗数据命令：

|  |
| --- |
| */\*#建立数据库\*/* **create** database flux;  */\*#建立外部表\*/* **create EXTERNAL table** flux (host string,url String,urlname String,title String,chset String,scr String,col String,lg String,je String,ec String,fv String,cn String,ref String,uagent String,stat\_ss String,stat\_uv String,cip String) partitioned **by** (reportTime string) row format delimited fields terminated **by '|'** stored **as** textfile location **'/flux'**;  */\*#增加分区\*/* **ALTER TABLE** flux **add** PARTITION (reportTime=**'2016-12-02'**) location **'/flux/reportTime=2016-12-02'**;  */\*网站访问次数pv\*/* **select** *count*(*\**) **as** pv **from** flux **where** reportTime=**'2017-01-09'**;  */\*uv\*/* **select** *count*(**distinct** stat\_ss) **as** uv **from** flux **where** reportTime=**'2017-01-09'**;  */\*vv\*/* **select** *count*(**DISTINCT** split(stat\_uv,**'\_'**)[0]) **as** vv **from** flux **where** reportTime=**'2017-01-09'**;  */\*跳出率\*/* **select** round(a/b,2) **from** (**select** *count*(*\**)**as** a **from** (**select** *count*(**distinct** urlname) **as** u **from** flux **group by** split(stat\_uv,**'\_'**)[0]) **as c where** u==1) **as** tab1 **left join** (**select** *count*(**DISTINCT** split(stat\_uv,**'\_'**)[0]) **as** b **from** flux) **as** tab2;  */\*新增独立ip\*/* **select** *count*(**DISTINCT** cip) **from** flux **where** reportTime=**'2017-01-09' and** cip **not in** (**select distinct** cip **as** ip **from** (**select** *\** **from** flux) **as** newt **where** reportTime <> **'2017-01-09'**);  */\*平均时长avgtime\*/* **select** round(*avg*(tim),2) **as** avgtime **from** (**select** round((*max*(split(stat\_ss,**'\_'**)[2])-*min*(split(stat\_ss,**'\_'**)[2]))/1000,2) **as** tim **from** flux **group by** split(stat\_ss,**'\_'**)[0]) **as** timet;  */\*新增独立访客newcust\*/* **select** *count*(**distinct** stat\_ss) **as** newcust **from** flux **where** stat\_ss **not in** (**select distinct** stat\_ss **as** history **from** (**select** *\** **from** flux) **as** tnewcust **where** reportTime<>**'2016-12-02'**) **and** reportTime=**'2016-12-02'**;  */\*访问深度wiewdeep\*/* **select** round(*avg*(deep),2) **as** wd **from** (**select** *count*(**distinct** urlname) **as** deep **from** flux **group by** split(stat\_ss,**'\_'**)[0]) **as** tt;  */\*创建结果表\*/* **create table** tongji(**time date**,**pv int**,**uv int**,**vv int**,**br double**,**newip int**,**avgtime int**,**newcust int**,**viewdeep int**)row format delimited fields terminated **by '|'**; */\*把计算结果合并插入表中\*/* **insert** overwrite **table** tongji **select '2016-12-02'**,t1.pv,t2.uv,t3.vv,t4.br,t5.newip,t6.avgtime,t7.newcust,t8.viewdeep **from** (**select** *count*(*\**) **as** pv **from** flux **where** reportTime=**'2016-12-02'** ) **as** t1 **left join** (**select** *count*(**distinct** stat\_ss) **as** uv **from** flux **where** reportTime=**'2016-12-02'**) **as** t2 **left join** (**select** *count*(**distinct** split(stat\_uv,**'\_'**)[0]) **as** vv **from** flux **where** reportTime=**'2016-12-02'**) **as** t3 **left join** (**select** round(x/y,2) **as** br **from** (**select** *count*(*\**) **as** x **from** (**select** *count*(*\**) **as** countView **from** flux **group by** split(stat\_uv,**'\_'**)[0] **having** countView = 1) **as** tbr11) **as** tbr1 **left join** (**select** *count*(*\**) **as** y **from** (**select** *count*(*\**) **as** countView **from** flux **group by** split(stat\_uv,**'\_'**)[0]) **as** tbr22) **as** tbr2) **as** t4 **left join** (**select** *count*(**distinct** cip) **as** newip **from** flux **where** cip **not in** (**select distinct** cip **as** history **from** (**select** *\** **from** flux) **as** tnewip **where** reportTime<>**'2016-12-02'**) **and** reportTime=**'2016-12-02'**) **as** t5 **left join** (**select** round(*avg*(**time**),2) **as** avgtime **from**(**select** round((*max*(split(stat\_uv,**'\_'**)[2]) - *min*(split(stat\_uv,**'\_'**)[2]))/1000,0) **time from** flux **group by** split(stat\_uv,**'\_'**)[0]) **as** tavgtime) **as** t6 **left join** (**select** *count*(**distinct** stat\_ss) **as** newcust **from** flux **where** stat\_ss **not in** (**select distinct** stat\_ss **as** history **from** (**select** *\** **from** flux) **as** tnewcust **where** reportTime<>**'2016-12-02'**) **and** reportTime=**'2016-12-02'**) **as** t7 **left join**(**select** round(*avg*(deep),2) **as** viewdeep **from** (**select** *count*(**distinct** urlname) **as** deep **from** flux **where** reportTime=**'2016-12-02' group by** split(stat\_uv,**'\_'**)[0]) **as** tviewdeep) **as** t8; |

## Flume链接kafka

|  |
| --- |
| a1.sources=r1  a1.sinks=k1  a1.channels=c1  a1.sources.r1.type=avro  a1.sources.r1.bind=0.0.0.0  a1.sources.r1.port=44444  a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink  a1.sinks.k1.topic = test  a1.sinks.k1.brokerList = localhost:9092  a1.sinks.k1.requiredAcks = 1  a1.sinks.k1.batchSize = 20  a1.channels.c1.type=memory  a1.channels.c1.capacity=1000  a1.channels.c1.transactionCapacity=100  a1.sources.r1.channels=c1  a1.sinks.k1.channel=1 |

启动：

 bin/flume-ng agent --conf conf --conf-file conf/kafka.properties --name a1 -Dflume.root.logger=INFO,console

启动kafka：

启动kafka:

启动zookeeper：

bin/zookeeper-server-start.sh config/zookeeper.properties &

启动kafka：

bin/kafka-server-start.sh config/server.properties

创建topic：

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test