

网站流量分析项目概述

2018年9月15日 星期六 上午 9:07

1. 业务背景

网站流量统计是改进网站服务的重要手段之一，通过获取用户在网站的行为数据，进行分析,得到有价值的信息。可以基于这些数据对网站进行改进。

按照在线情况分析：

来访时间、访客地域、来路页面、当前停留页面等，这些功能对企业实时掌握自身网站流量有很大的帮助。

按照时段分析：

按照任意的时间段 和任意的时间粒度来进行分析，比如小时段分布，日访问量分布，对于企业了解用户浏览网页的时间段有一个很好的分析。

按来源分析

来源分析提供来路域名带来的来访次数、IP、独立访客、新访客、新访客浏览次数、站内总浏览次数等数据。这个数据可以直接让企业了解推广成效的来路，从而分析出那些网站投放的广告效果更明显。

2. 业务需求

a. PV

PV(page view), 访问量 也叫 点击量，即一天之内整个网站中的页面被访问的次数。对同一个页面的重复访问记为不同的PV。

b. UV

UV(unique visitor),独立访客数，即一天之内访问网站的人数。同一天之内访问网站多次，也只能计算一个UV。

c. VV

VV (Visit View), 会话总数，即一天之内会话的总的数量。所谓的一次会话，指的是为了实现某些功能，浏览器开发网站网站，从访问第一个页面开始，会话开始，直到访问最后一个页面结束，关闭所有页面，会话结束。会话可以认为在访问第一个页面时开始，访问所有页面完成并关闭，或，超过指定时长没有后续访问 都认为会话结束。

d. BR

BR(Bounce Rate)跳出率，一天之内跳出的会话总数占有所有会话总数的比率。所谓跳出只的是一个会话中只访问过一个页面会话就结束了，这就称之为该会话跳出了。跳出的会话占全部会话的比率，称之为跳出率。这个指标在评价行推广活动的效果时非常的有用。

e. NewIP

NewIp,新增ip总数,一天之内访问网站的所有IP去重后，检查有多少是在历史数据中从未出现过的，这些IP计数，就是新增的IP总数，这个指标可以一定程度上体现网站新用户增长的

情况。

f. NewCust

NewCust，新增独立访客数，一天之内访问网站的人中，有多少人是历史记录中从来没有出现过的。这个指标可以从另一个角度体现网站用户增长的情况。

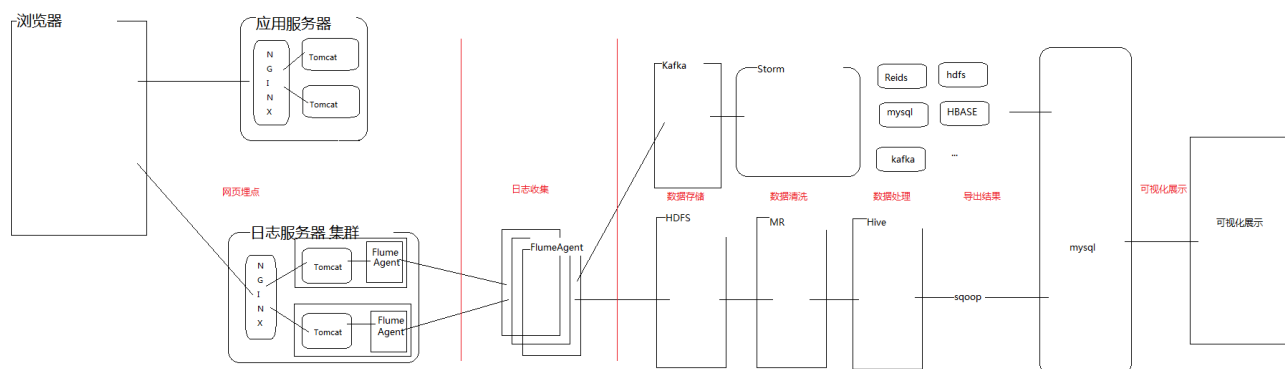
g. AvgTime

AvgTime,平均访问时长，所谓一个会话的访问时长，是指，一个会话结束的时间减去一个会话开始时间得到的会话经历的时长。将一天之内所有会话的访问时长求平均值 就是 平均访问时长。这个指标可以体现出网站对用户的粘性大小。

h. AvgDeep

AvgDeep，平均访问深度，所谓一个会话的访问深度，是指，一个会话中访问的所有资源地址去重后计数得到的指标。将一天之内所有会话的访问深度求平均值 就是 平均访问深度。这个指标可以体现出网站对用户的粘性大小。

3. 技术架构



1. 网页埋点

开发js文件，要求应用服务器的所有页面都引入这个js文件，在页面被访问时触发。
触发的js收集用户相关行为数据信息，提交到日志服务器中。

2. js编写

a. pv计算

访问量，一天之内访问网站的次数。

浏览器对应用服务器的一次访问，会对应日志服务器中的一条日志。

则想要计算pv，**并不需要其他额外的值，只需要在日志服务器中统计当前日志的数量即可得到pv。**

b. uv计算

独立访客数，一天之内访问网站的人数。

浏览器对应用服务器的一次访问，会对应日志服务器中的一条日志。

想要计算uv，**需要在这些日志中标识出每条日志是属于哪个用户的，之后，对于这一天的所有日志中的用户，去重计数就可以得到uv。**

如何标识一个用户？

在js中，为每个用户指定独一无二的编号uvid来唯一标识，并将这个uvid存储在浏览器cookie中，后续再有访问日志时，复用cookie中保存的uvid，这样这个用户无论访问多少次，都有相同的uvid，标识这些访问来自于同一个用户。

伪代码：

```
1 从cookie中获取uvid
2  if (没有获取到uvid){
3      创建一个uvid
4      作为当前用户唯一标识，后续提交给服务器
5      将此uvid保存到cookie中，保存十年，以便后续使用
6  }else{
7      从中获取uvid
8      作为当前用户的唯一标识，后续提交给服务器
9  }
```

真正js代码：

```
1  //处理uv
2  //--获取cookie ar_stat_uv的值
3  var uv_str = ar_get_cookie("ar_stat_uv");
4  var uv_id = "";
5  //--如果cookie ar_stat_uv的值为空
6  if (uv_str == ""){
7      //--为这个新uv配置id，为一个长度20的随机数字
8      uv_id = ar_get_random(20);
9      //--设置cookie ar_stat_uv 保存时间为10年
10     ar_set_cookie("ar_stat_uv", uv_id, 1);
11 }else{ //--如果cookie ar_stat_uv的值不为空
```

```

11 }else{/--如果cookie ar_stat_uv的值不为空
12     //--获取uv_id
13     uv_id = uv_str;
14 }

```

c. vv计算

会话总数，一天之内产生的会话的总的数量。

浏览器对应用服务器的一次访问，会对应在日志服务器中的一条日志。

想要计算vv，需要在这些日志中标识出每条日志是属于哪个会话的，之后，对于这一天的所有日志中的会话，去重计数就可以得到vv。

如何标识一个会话？

在js中，为每个会话指定独一无二的编号ss_id来唯一标识，并将这个ss_id存储在浏览器cookie中，后续同一个会话再有访问日志时，复用cookie中保存的ss_id，这样这个会话无论访问多少次，都有相同的ss_id，标识这些访问来自于同一个会话。

会话开始时 生成ss_id存入cookie，整个会话中复用这个ss_id标识会话。

而在会话结束时 - 浏览器关闭 或 会话超时，则应该丢弃之前的ss_id，后续访问应该重新生成ss_id标识为另一个新的会话。

伪代码：

```

1 从cookie中获取ss_stat
2  if (没有获取到ss_stat){
3      创建一个ss_stat，其中包含ss_id、ss_time和ss_count
4  }else{
5      从中获取到ss_stat,从中可以得到ss_id、ss_time和ss_count
6      if (当前时间 - ss_time > 会话超时时间){
7          会话已经超时，重新生成会话信息
8          创建一个ss_stat，其中包含ss_id、ss_time和ss_count
9      }else{
10         会话没有超时，复用之前的信息
11         更新ss_stat信息，其中ss_id不变，ss_time变为当前时间，ss_count+1
12     }
13 }
14
15 这些信息作为这个会话的唯一标识，后续提交给服务器
    将这个ss_stat保存在cookie中,以便后续使用，不指定保存时长，默认存在浏览器内存中，关闭时销毁

```

真正js代码：

```

1  //处理ss
2  //--获取cookie ar_stat_ss
3  var ss_stat = ar_get_cookie("ar_stat_ss");
4  var ss_id = ""; //session id
5  var ss_count = 0; //session有效期内访问页面的次数
6  var ss_time = "";
7  //--如果cookie中不存在ar_stat_ss 说明是一次新的会话
8  if (ss_str == ""){
9      //--随机生成长度为10的session id
10     ss_id = ar_get_random(10);
11     //--session有效期内页面访问次数为0
12     ss_count = 0;
13     //--当前事件
14     ss_time = ar_get_stm()
15 } else { //--如果cookie中存在ar_stat_ss
16     //获取ss相关信息
17     var items = ss_str.split("_");
18     //--ss_id
19     ss_id = items[0];

```

```

20         //--ss_count
21         ss_count = parseInt(items[1]);
22         //--ss_stm
23         ss_time = items[2];
24
25
26         //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,
27         //但是其实已经超时了!仍然需要重新生成cookie
28         if (ar_get_stm() - ss_time > expire_time) {
29             //--重新生成会话id
30             ss_id = ar_get_random(10);
31             //--设置会话中的页面访问次数为0
32             ss_count = 0;
33             //--当前事件
34             ss_time = ar_get_stm();
35         } else { //--如果会话没有超时
36             //--会话id不变
37             //--设置会话中的页面方位次数+1
38             ss_count = ss_count + 1;
39             ss_time = ar_get_stm();
40         }
41     }
42     //--重新拼接cookie ar_stat_ss的值
43     value = ss_id+"_"+ss_count+"_"+ss_time;
44     ar_set_cookie("ar_stat_ss", value, 0);

```

d. BR跳出率

跳出率，一天之内跳出的会话总数 占 所有会话总数的比率

浏览器对应用服务器的一次访问，会对应应在日志服务器中的一条日志。

想要计算br，需要算出一天内跳出的会话总数 和 总的会话数 求其比值，其中总的会话数就是上面的vv，而跳出的会话总数，需要将今天一天的所有的日志 按照ss_id进行分组，计算组内日志的数量，筛选出日志数量为1的会话技术，就是BR跳出率。

其中需要会话编号ss_id ss_time,之前在计算vv时已经有了，不需要额外从客户端收集其他信息。

e. NewIp新增IP总数

新增IP总数，一天之内所有的ip去重后，在历史数据中从未出现过的IP的数量。

浏览器对应用服务器的一次访问，会对应应在日志服务器中的一条日志。

想要计算新增IP总数，需要得到客户端IP，以便于在服务器判断是否为新IP。

想要获取客户端IP，可以在浏览器端获取，或在服务端获取。其中在浏览器端获取时，获取比较困难，且通常得到的时内网IP，不好用，所有可以在服务器端来获取，JS中不需要获取客户端的额外信息。

f. NewCust新增客户总数

新增IP总数，一天之内所有的uvid去重后，在历史数据中从未出现过的uvid的数量。

浏览器对应用服务器的一次访问，会对应应在日志服务器中的一条日志。

想要计算新增客户总数，需要获取一天内所有访问记录中的uvid，取重后，得到在历史数据中从未出现过的uvid的数量，就是NewCust。

需要客户端在每条日志中记录当前客户端uvid，这个信息在计算uv的过程已经有了，JS中不需要获取客户端额外的信息。

g. AvgTime平均访问时长

平均访问时长，一天内所有会话时长的平均值。

浏览器对应用服务器的一次访问，会对应日志服务器中的一条日志。

想要计算平均访问时长，需要得到每个会话最后一个页面访问的时间减去这个会话第一个页面访问的时间，再求这些会话访问时长的平均值。

需要客户端在提交的日志中，包含ssid，sstime，这几个信息在计算wv的过程中已经有了，JS中不需要获取客户端额外的信息。

h. AvgDeep平均访问深度

平均访问深度，一天内所有会话访问深度的平均值。

浏览器对应用服务器的一次访问，会对应日志服务器中的一条日志。

想要计算平均访问深度，需要得到每个会话中访问的所有地址去重后计数，再求这些会话访问深度的平均值。

需要客户端在提交的日志中，包含ssid，url，其中ssid在计算wv的过程中已经有了，JS中还需要额外获取每个页面的URL地址信息。

```
1 //当前地址
2 var url = document.URL;
3 url = ar_encode(String(url));
4 //当前资源名
5 var urlname = document.URL.substring(document.URL.lastIndexOf("/") + 1);
6 urlname = ar_encode(String(urlname));
```

i. 其他信息

```
1 //返回导航到当前网页的超链接所在网页的URL
2 var ref = document.referrer;
3 ref = ar_encode(String(ref));
4
5
6
7 //网页标题
8 var title = document.title;
9 title = ar_encode(String(title));
10
11
12 //网页字符集
13 var charset = document.charset;
14 charset = ar_encode(String(charset));
15
16
17 //屏幕信息
18 var screen = ar_get_screen();
19 screen = ar_encode(String(screen));
20
21
22 //颜色信息
23 var color = ar_get_color();
24 color = ar_encode(String(color));
25
26 //语言信息
27 var language = ar_get_language();
28 language = ar_encode(String(language));
29
30
31 //浏览器类型
32 var agent = ar_get_agent();
```

```

32 var agent = ar_get_agent();
33 agent = ar_encode(String(agent));
34
35 //浏览器是否支持并启用了java
36 var jvm_enabled = ar_get_jvm_enabled();
37 jvm_enabled = ar_encode(String(jvm_enabled));
38
39 //浏览器是否支持并启用了cookie
40 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));

```

j. 组织以上信息，提交给日志服务器

将以上信息拼接到日志服务器的地址之后，作为请求参数发送

```

1 //拼接访问地址 增加如上信息
2 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;

```

从js中异步访问此地址

方式1：使用Ajax

方式2：嵌入图片

```

1 document.getElementsByTagName("body")[0].innerHTML += "<img src='"+dest+"' border='"+0"'
  width='1' height='1' />";

```

3. 完整JS代码

```

1 /**函数可对字符串进行编码，这样就可以在所有的计算机上读取该字符串。*/
2 function ar_encode(str)
3 {
4     //进行URL编码
5     return encodeURIComponent(str);
6 }
7
8
9
10 /**屏幕分辨率*/
11 function ar_get_screen()
12 {
13     var c = "";
14
15     if (self.screen) {
16         c = screen.width+"x"+screen.height;
17     }
18
19     return c;
20 }
21
22
23
24 /**颜色质量*/
25 function ar_get_color()
26 {
27     var c = "";
28
29     if (self.screen) {
30         c = screen.colorDepth+"-bit";
31     }
32
33     return c;
34 }
35
36
37
38

```

```

39  /**返回当前的浏览器语言*/
40  function ar_get_language()
41  {
42      var l = "";
43      var n = navigator;
44
45      if (n.language) {
46          l = n.language.toLowerCase();
47      }
48      else
49      if (n.browserLanguage) {
50          l = n.browserLanguage.toLowerCase();
51      }
52
53      return l;
54  }
55
56
57  /**返回浏览器类型IE,Firefox*/
58  function ar_get_agent()
59  {
60      var a = "";
61      var n = navigator;
62
63      if (n.userAgent) {
64          a = n.userAgent;
65      }
66
67      return a;
68  }
69
70
71
72
73  /**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*/
74  function ar_get_jvm_enabled()
75  {
76      var j = "";
77      var n = navigator;
78
79      j = n.javaEnabled() ? 1 : 0;
80
81      return j;
82  }
83
84
85  /**返回浏览器是否支持(启用) cookie */
86  function ar_get_cookie_enabled()
87  {
88      var c = "";
89      var n = navigator;
90      c = n.cookieEnabled ? 1 : 0;
91
92      return c;
93  }
94
95
96
97  /**检测浏览器是否支持Flash或有Flash插件*/
98  function ar_get_flash_ver()
99  {
100     var f="",n=navigator;
101
102
103     if (n.plugins && n.plugins.length) {
104         for (var ii=0;ii<n.plugins.length;ii++) {
105             if (n.plugins[ii].name.indexOf('Shockwave Flash')!=-1) {
106                 f=n.plugins[ii].description.split('Shockwave Flash ')[1];
107                 break;
108             }
109         }
110     }
111     else
112     if (window.ActiveXObject) {
113         for (var ii=10;ii>=2;ii--) {
114             try {
115                 var fl=eval("new
116 ActiveXObject('ShockwaveFlash.ShockwaveFlash."+ii+"');");
117                 if (fl) {
118                     f=ii + '.0';
119                     break;
120                 }
121             } catch (e) {}
122         }
123     }

```



```

124         },
125         return f;
126     }
127
128
129     /**匹配顶级域名*/
130     function ar_c_etry_top_domain(str)
131     {
132         var pattern = "/^aero$|^cat$|^coop$|^int$|^museum$|^pro$|^travel$|^xxx$|^com$|^net$|^gov
133 $|^org$|^mil$|^edu$|^biz$|^info$|^name$|^ac$|^mil$|^co$|^ed$|^gv$|^nt$|^bj$|^hz$|^sh$|^tj$|^cq$|^he
134 $|^nm$|^ln$|^jl$|^hl$|^js$|^zj$|^ah$|^hb$|^hn$|^gd$|^gx$|^hi$|^sc$|^gz$|^yn$|^xz$|^sn$|^gs$|^qh
135 $|^nx$|^xj$|^tw$|^hk$|^mo$|^fj$|^ha$|^jx$|^sd$|^sx$/i";
136
137         if(str.match(pattern)){ return 1; }
138
139         return 0;
140     }
141
142     /**处理域名地址*/
143     function ar_get_domain(host)
144     {
145         //如果存在则截去域名开头的 "www."
146         var d=host.replace(/^(www\./, "");
147
148
149         //剩余部分按照"."进行split操作，获取长度
150         var ss=d.split(".");
151         var l=ss.length;
152
153
154         //如果长度为3，则为xxx.yyy.zz格式
155         if(l == 3){
156             //如果yyy为顶级域名，zz为次级域名，保留所有
157             if(ar_c_etry_top_domain(ss[1]) && ar_c_etry_domain(ss[2])){
158                 //否则只保留后两节
159                 else{
160                     d = ss[1]+"."+ss[2];
161                 }
162             }
163         }
164
165         //如果长度大于3
166         else if(l >= 3){
167
168             //如果host本身是个ip地址，则直接返回该ip地址为完整域名
169             var ip_pat = "[0-9]*\.[0-9]*\.[0-9]*\.[0-9]*";
170             if(host.match(ip_pat)){
171                 return d;
172             }
173
174             //如果host后两节为顶级域名及次级域名，则保留后三节
175             if(ar_c_etry_top_domain(ss[l-2]) && ar_c_etry_domain(ss[l-1])) {
176                 d = ss[l-3]+"."+ss[l-2]+"."+ss[l-1];
177             }
178
179             //否则保留后两节
180             else{
181                 d = ss[l-2]+"."+ss[l-1];
182             }
183         }
184
185         return d;
186     }
187
188     /**返回cookie信息*/
189     function ar_get_cookie(name)
190     {
191         //获取所有cookie信息
192         var co=document.cookie;
193
194
195         //如果名字是个空 返回所有cookie信息
196         if (name == "") {
197             return co;
198         }
199
200         //名字不为空 则在所有的cookie中查找这个名字的cookie
201         var mn=name+"=";
202         var b,e;
203         b=co.indexOf(mn);

```

```

202         b=co.indexOf(mn);
203
204         //没有找到这个名字的cookie 则返回空
205         if (b < 0) {
206             return "";
207         }
208
209
210         //找到了这个名字的cookie 获取cookie的值返回
211         e=co.indexOf(";", b+name.length);
212         if (e < 0) {
213             return co.substring(b+name.length + 1);
214         }
215         else {
216             return co.substring(b+name.length + 1, e);
217         }
218     }
219 }
220
221 /**
222     设置cookie信息
223     操作符：
224         0 表示不设置超时时间 cookie是一个会话级别的cookie cookie信息保存在浏览器内存当中 浏览器关闭时
225         cookie消失
226         1 表示设置超时时间为10年以后 cookie会一直保存在浏览器的临时文件夹里 直到超时时间到来 或用户手动清
227         空cookie为止
228         2 表示设置超时时间为1个小时以后 cookie会一直保存在浏览器的临时文件夹里 直到超时时间到来 或用户手动
229         清空cookie为止
230     */
231
232     function ar_set_cookie(name, val, cotp)
233     {
234         var date=new Date;
235         var year=date.getFullYear();
236         var hour=date.getHours();
237
238         var cookie="";
239
240         if (cotp == 0) {
241             cookie=name+"="+val+";";
242         }
243         else if (cotp == 1) {
244             year=year+10;
245             date.setYear(year);
246             cookie=name+"="+val+";expires="+date.toGMTString()+";";
247         }
248         else if (cotp == 2) {
249             hour=hour+1;
250             date.setHours(hour);
251             cookie=name+"="+val+";expires="+date.toGMTString()+";";
252         }
253
254         var d=ar_get_domain(document.domain);
255         if(d != ""){
256             cookie += "domain="+d+";";
257         }
258         cookie += "path="/" + ";";
259
260         document.cookie=cookie;
261     }
262
263
264
265
266
267
268
269     /**返回客户端时间*/
270     function ar_get_stm()
271     {
272         return new Date().getTime();
273     }
274
275
276
277     /**返回指定个数的随机数字串*/
278     function ar_get_random(n) {
279         var str = "";
280         for (var i = 0; i < n; i++) {
281             str += String(parseInt(Math.random() * 10));
282         }
283         return str;

```

```

283 }
284
285 /* main function */
286 function ar_main() {
287
288     //收集完日志 提交到的路径
289     var dest_path = "http://localhost:8090/LogDemo/servlet/LogServlet?";
290     var expire_time = 30 * 60 * 1000; //会话超时时长
291
292     //处理uv
293     //--获取cookie ar_stat_uv的值
294     var uv_str = ar_get_cookie("ar_stat_uv");
295     var uv_id = "";
296     //--如果cookie ar_stat_uv的值为空
297     if (uv_str == "") {
298         //--为这个新uv配置id, 为一个长度20的随机数字
299         uv_id = ar_get_random(20);
300         //--设置cookie ar_stat_uv 保存时间为10年
301         ar_set_cookie("ar_stat_uv", uv_id, 1);
302     }
303     //--如果cookie ar_stat_uv的值不为空
304     else {
305         //--获取uv_id
306         uv_id = uv_str;
307     }
308
309     //处理ss
310     //--获取cookie ar_stat_ss
311     var ss_stat = ar_get_cookie("ar_stat_ss");
312     var ss_id = ""; //session id
313     var ss_count = 0; //session有效期内访问页面的次数
314     var ss_time = "";
315     //--如果cookie中不存在ar_stat_ss 说明是一次新的会话
316     if (ss_stat == "") {
317         //--随机生成长度为10的session id
318         ss_id = ar_get_random(10);
319         //--session有效期内页面访问次数为0
320         ss_count = 0;
321         //--当前事件
322         ss_time = ar_get_stm();
323     } else { //--如果cookie中存在ar_stat_ss
324         //获取ss相关信息
325         var items = ss_stat.split("_");
326         //--ss_id
327         ss_id = items[0];
328         //--ss_count
329         ss_count = parseInt(items[1]);
330         //--ss_stm
331         ss_time = items[2];
332
333         //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
334         //重新生成cookie
335         if (ar_get_stm() - ss_time > expire_time) {
336             //--重新生成会话id
337             ss_id = ar_get_random(10);
338             //--设置会话中的页面访问次数为0
339             ss_count = 0;
340             //--当前事件
341             ss_time = ar_get_stm();
342         } else { //--如果会话没有超时
343             //--会话id不变
344             //--设置会话中的页面方位次数+1
345             ss_count = ss_count + 1;
346             ss_time = ar_get_stm();
347         }
348     }
349     //--重新拼接cookie ar_stat_ss的值
350     value = ss_id + "_" + ss_count + "_" + ss_time;
351     ar_set_cookie("ar_stat_ss", value, 0);
352
353     //当前地址
354     var url = document.URL;

```

```

359     var url = document.URL;
360     url = ar_encode(String(url));
361
362     //当前资源名
363     var urlname = document.URL.substring(document.URL.lastIndexOf("/") + 1);
364     urlname = ar_encode(String(urlname));
365
366     //返回导航到当前网页的超链接所在网页的URL
367     var ref = document.referrer;
368     ref = ar_encode(String(ref));
369
370
371
372     //网页标题
373     var title = document.title;
374     title = ar_encode(String(title));
375
376
377     //网页字符集
378     var charset = document.charset;
379     charset = ar_encode(String(charset));
380
381
382     //屏幕信息
383     var screen = ar_get_screen();
384     screen = ar_encode(String(screen));
385
386
387     //颜色信息
388     var color = ar_get_color();
389     color = ar_encode(String(color));
390
391
392     //语言信息
393     var language = ar_get_language();
394     language = ar_encode(String(language));
395
396
397     //浏览器类型
398     var agent = ar_get_agent();
399     agent = ar_encode(String(agent));
400
401
402     //浏览器是否支持并启用了java
403     var jvm_enabled = ar_get_jvm_enabled();
404     jvm_enabled = ar_encode(String(jvm_enabled));
405
406
407     //浏览器是否支持并启用了cookie
408     var cookie_enabled = ar_get_cookie_enabled();
409     cookie_enabled = ar_encode(String(cookie_enabled));
410
411
412     //浏览器flash版本
413     var flash_ver = ar_get_flash_ver();
414     flash_ver = ar_encode(String(flash_ver));
415
416
417     //当前ss状态 格式为"会话id_会话次数_当前时间"
418     var stat_ss = ss_id + "_" + ss_count + "_" + ss_time;
419
420     //拼接访问地址 增加如上信息
421     dest = dest_path + "url=" + url + "&urlname=" + urlname + "&title=" + title + "&charset=" + charset + "&scr=" + screen + "&col=" + color + "≶=" + 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;
422
423
424     //通过插入图片访问该地址
425     document.getElementsByTagName("body")[0].innerHTML += "<img src=\"" + dest + "\" border=\"" + 0 + "\" width=\"" + 1 + "\" height=\"" + 1 + "\" />";
426
427 }
428
429 window.onload = function() {
430     //触发main方法
431     ar_main();
432 }

```

4. 开发web服务器代码

```
1  package cn.tedu.flux;
2
3
4  import java.io.IOException;
5  import java.net.URLDecoder;
6  import java.net.URLEncoder;
7
8  import javax.servlet.ServletException;
9  import javax.servlet.http.HttpServlet;
10 import javax.servlet.http.HttpServletRequest;
11 import javax.servlet.http.HttpServletResponse;
12
13
14 import org.apache.log4j.Logger;
15
16
17 public class LogServlet extends HttpServlet {
18     private static Logger logger = Logger.getLogger(LogServlet.class);
19     public void doGet(HttpServletRequest req, HttpServletResponse resp)
20         throws ServletException, IOException {
21         String qs = URLDecoder.decode(req.getQueryString(), "utf-8");
22
23         String kvs [] = qs.split("&");
24         StringBuilder sb = new StringBuilder();
25         for(String kv : kvs ){
26             String [] arr = kv.split("=");
27             String v = arr.length>=2 ? arr[1] : "";
28             sb.append(v+"|");
29         }
30         //拼接客户端ip
31         sb.append(req.getRemoteAddr());
32
33         String line = sb.toString();
34         logger.info(line);
35     }
36
37     public void doPost(HttpServletRequest req, HttpServletResponse resp)
38         throws ServletException, IOException {
39         doGet(req, resp);
40     }
41 }
```

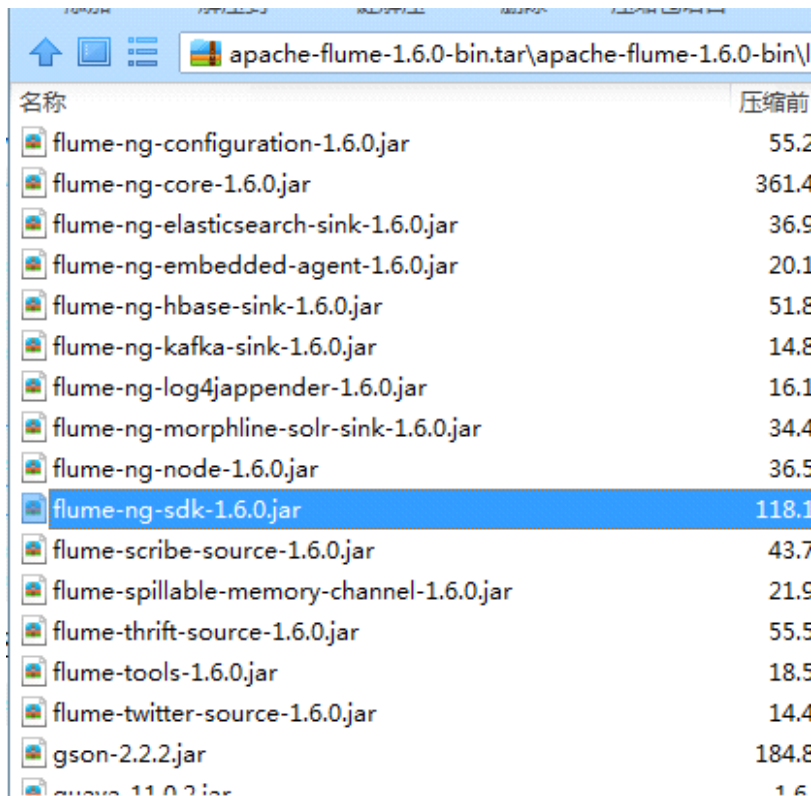
Flume日志收集的开发

2018年9月15日 星期六 下午 4:01

1. 发送日志到Flume

在日志服务器中，通过Log4jAppender将日志发往flume客户端

a. 在日志服务器应用中导入Log4jAppender相关开发包



名称	压缩前
flume-ng-configuration-1.6.0.jar	55.2
flume-ng-core-1.6.0.jar	361.4
flume-ng-elasticsearch-sink-1.6.0.jar	36.9
flume-ng-embedded-agent-1.6.0.jar	20.1
flume-ng-hbase-sink-1.6.0.jar	51.8
flume-ng-kafka-sink-1.6.0.jar	14.8
flume-ng-log4jappender-1.6.0.jar	16.1
flume-ng-morphline-solr-sink-1.6.0.jar	34.4
flume-ng-node-1.6.0.jar	36.5
flume-ng-sdk-1.6.0.jar	118.1
flume-scribe-source-1.6.0.jar	43.7
flume-spillable-memory-channel-1.6.0.jar	21.9
flume-thrift-source-1.6.0.jar	55.5
flume-tools-1.6.0.jar	18.5
flume-twitter-source-1.6.0.jar	14.4
gson-2.2.2.jar	184.8
guava-11.0.2.jar	1.6

b. 配置log4j配置文件，实现发送日志给flume

```
1 log4j.rootLogger = info,stdout,flume
2
3
4 log4j.appender.stdout = org.apache.log4j.ConsoleAppender
5 log4j.appender.stdout.Target = System.out
6 log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
7 log4j.appender.stdout.layout.ConversionPattern = %m%n
8
9
10 log4j.appender.flume =
11 org.apache.flume.clients.log4jappender.Log4jAppender
12 log4j.appender.flume.Hostname = hadoop01
13 log4j.appender.flume.Port = 44444
14 log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
15 log4j.appender.stdout.layout.ConversionPattern = %m%n
```

c. 在日志服务器的LogServlet中，通过log4j来发送日志

```
1 logger.info(line);
```

2. 开发客户端Agent

hadoop03

```
1  #声明Agent
2  a1.sources = r1
3  a1.sinks = k1 k2
4  a1.channels = c1
5
6
7  #声明source
8  a1.sources.r1.type = avro
9  a1.sources.r1.bind = 0.0.0.0
10 a1.sources.r1.port = 44444
11
12 #声明sink
13 a1.sinks.k1.type = avro
14 a1.sinks.k1.hostname = hadoop01
15 a1.sinks.k1.port = 44444
16
17
18 a1.sinks.k2.type = avro
19 a1.sinks.k2.hostname = hadoop02
20 a1.sinks.k2.port = 44444
21
22
23 a1.sinkgroups = g1
24 a1.sinkgroups.g1.sinks = k1 k2
25 a1.sinkgroups.g1.processor.type = load_balance
26 a1.sinkgroups.g1.processor.backoff = true
27 a1.sinkgroups.g1.processor.selector = random
28
29
30 #声明channel
31 a1.channels.c1.type = memory
32 a1.channels.c1.capacity = 1000
33 a1.channels.c1.transactionCapacity = 100
34
35
36 #绑定关系
37 a1.sources.r1.channels = c1
38 a1.sinks.k1.channel = c1
39 a1.sinks.k2.channel = c1
```

3. 开发中心服务器Agent

hadoop01 hadoop02

```
1  #配置Agent
2  a1.sources = r1
3  a1.sinks = k1
4  a1.channels = c1
5
6
7  #声明Source
8  a1.sources.r1.type = avro
9  a1.sources.r1.bind = 0.0.0.0
10 a1.sources.r1.port = 44444
11
12 #声明sink
13 a1.sinks.k1.type = hdfs
14 a1.sinks.k1.hdfs.path = hdfs://hadoop01:9000/flux
15 a1.sinks.k1.hdfs.rollInterval = 30
16 a1.sinks.k1.hdfs.rollSize = 0
17 a1.sinks.k1.hdfs.rollCount = 0
18 a1.sinks.k1.hdfs.timeZone = GMT+8
```

```
19  
20  
21 #声明channel  
22 a1.channels.c1.type = memory  
23 a1.channels.c1.capacity = 1000  
24 a1.channels.c1.transactionCapacity = 100  
25  
26  
    #绑定关系  
    a1.sources.r1.channels = c1  
    a1.sinks.k1.channel = c1
```