Flux_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。同一个人在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，有多少人是在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。

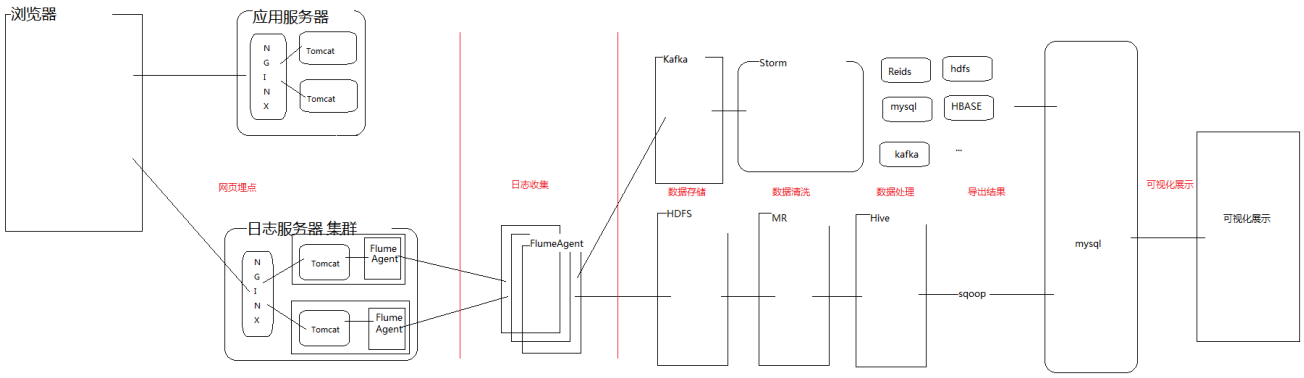
g. AvgTime

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取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



Flux_网页埋点

2018年9月15日 星期六　　上午 10:47

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，会对应在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，会对应在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：

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

      真正js代码：

```
//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);
}else{//--如果cookie ar stat uv的值不为空
```

```
}else{//--如果cookie ar_stat_uv的值不为空
        //--获取uv_id
        uv_id  = uv_str;
}
```

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：

```
从cookie中获取ss_stat
if(没有获取到ss_stat){
        创建一个ss_stat，其中包含ss_id、ss_time和ss_count
}else{
        从中获取到ss_stat,从中可以得到ss_id、ss_time和ss_count
        if(当前时间 -  ss_time > 会话超时时间){
                会话已经超时，重新生成会话信息
                创建一个ss_stat，其中包含ss_id、ss_time和ss_count
        }else{
                会话没有超时，复用之前的信息
                更新ss_stat信息，其中ss_id不变，ss_time变为当前时间，ss_count+1
        }
}
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：

```
//处理ss
//--获取cookie ar_stat_ss
var ss_stat = ar_get_cookie("ar_stat_ss");
var ss_id = "";  //sessin id
var ss_count = 0;   //session有效期内访问页面的次数
var ss_time = "";
//--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_count = 0;
        //--当前事件
        ss_time = ar_get_stm()
} else { //--如果cookie中存在ar_stat_ss
        //获取ss相关信息
        var items = ss_str.split("_");
        //--ss_id
        ss_id  = items[0];
```

```
        //--ss_count
        ss_count  = parseInt(items[1]);
        //--ss_stm
        ss_time = 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() - ss_time > 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_count = 0;
                //--当前事件
                ss_time = ar_get_stm();
        }else{//--如果会话没有超时
                //--会话id不变
                //--设置会话中的页面方位次数+1
                ss_count = ss_count + 1;
                ss_time = ar_get_stm();
        }
}
//--重新拼接cookie ar_stat_ss的值
value = ss_id+"_"+ss_count+"_"+ss_time;
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，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，会对应在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vv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，会对应在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vv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。

```
//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));
```

i. 其他信息

```
//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));



//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();
```

```
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));
```

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

```
//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;
```

从js中异步访问此地址

方式1：使用Ajax

方式2：嵌入图片

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

3. 完整JS代码

```
/**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。*/
function ar_encode(str)
{
        //进行URL编码
        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*/
```

```
/**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IE,Firefox*/
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)
{
        //获取所有cookie信息
        var co=document.cookie;

        //如果名字是个空 返回所有cookie信息
        if (name == "") {
                return co;
        }

        //名字不为空 则在所有的cookie中查找这个名字的cookie
        var mn=name+"=";
        var b,e;
        b=co.indexOf(mn);
```

```
        b=co.indexOf(nn);

        //没有找到这个名字的cookie 则返回空
        if (b < 0) {
                return "";
        }

        //找到了这个名字的cookie 获取cookie的值返回
        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
        操作符：
                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cookie消失
                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 或用户手动清空cookie为止
                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 或用户手动清空cookie为止
 * */
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() {

        //收集完日志 提交到的路径
        var dest_path   = "http://localhost:8090/LogDemo/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_stat = ar_get_cookie("ar_stat_ss");
        var ss_id = "";  //sessin id
        var ss_count = 0;   //session有效期内访问页面的次数
        var ss_time = "";
        //--如果cookie中不存在ar_stat_ss 说明是一次新的会话
        if (ss_stat == ""){
                //--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_count = 0;
                //--当前事件
                ss_time = ar_get_stm()
        } else { //--如果cookie中存在ar_stat_ss
                //获取ss相关信息
                var items = ss_stat.split("_");
                //--ss_id
                ss_id  = items[0];
                //--ss_count
                ss_count  = parseInt(items[1]);
                //--ss_stm
                ss_time = 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() - ss_time > 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_count = 0;
                        //--当前事件
                        ss_time = ar_get_stm();
                }else{//--如果会话没有超时
                        //--会话id不变
                        //--设置会话中的页面方位次数+1
                        ss_count = ss_count + 1;
                        ss_time = ar_get_stm();
                }
        }
        //--重新拼接cookie ar_stat_ss的值
        value = ss_id+"_"+ss_count+"_"+ss_time;
        ar_set_cookie("ar_stat_ss", value, 0);

        //当前地址
        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         //网页字符集
377         var charset = document.charset;
378         charset = ar_encode(String(charset));
379
380
381         //屏幕信息
382         var screen = ar_get_screen();
383         screen = ar_encode(String(screen));
384
            //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ss状态 格式为"会话id_会话次数_当前时间"
            var stat_ss = ss_id+"_"+ss_count+"_"+ss_time;
            //拼接访问地址 增加如上信息
            dest=dest_path+"url="+url+"&urlname="+urlname+"&title="+title+"&chset="+charset+"&scr="+scr
een+"&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();
    }
```

4. 开发web服务器代码

```
package cn.tedu.flux;


import java.io.IOException;
import java.net.URLDecoder;
import java.net.URLEncoder;


import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


import org.apache.log4j.Logger;


public class LogServlet extends HttpServlet {
	private static Logger logger = Logger.getLogger(LogServlet.class);
	public void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {
		String qs = URLDecoder.decode(req.getQueryString(), "utf-8");

		String kvs [] = qs.split("\\&");
		StringBuilder sb = new StringBuilder();
		for(String kv : kvs ){
			String [] arr = kv.split("=");
			String v = arr.length>=2 ? arr[1] : "";
			sb.append(v+"|");
		}
		//拼接客户端ip
		sb.append(req.getRemoteAddr());

		String line = sb.toString();
		logger.info(line);
	}


	public void doPost(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {
		doGet(req, resp);
	}


}
```

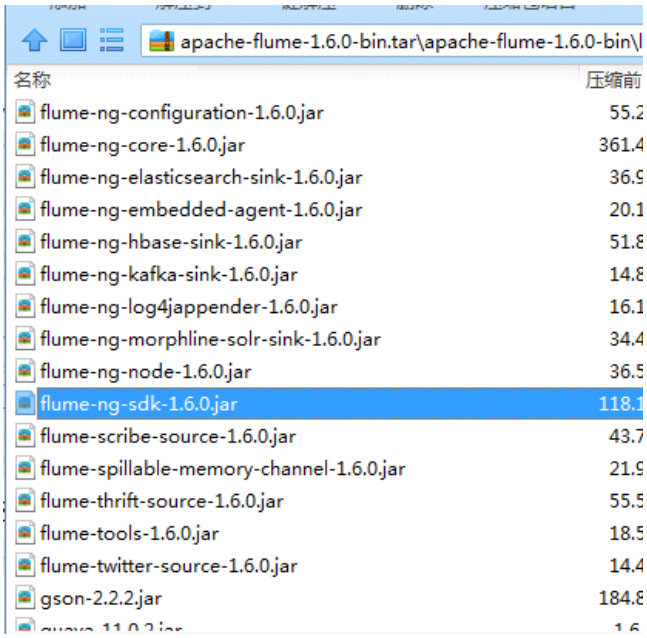
Flux_离线处理_日志收集

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Log4jApperndner相关开发包



b. 配置log4j配置文件，实现发送日志给flume

```
log4j.rootLogger = info,stdout,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.flume = org.apache.flume.clients.log4jappender.Log4jAppender
log4j.appender.flume.Hostname = hadoop01
log4j.appender.flume.Port = 44444
log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
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

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

2. 开发客户端Agent

hadoop03

```
#声明Agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1


#声明source
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.serializers = s1
a1.sources.r1.interceptors.i1.serializers.s1.name = timestamp


#声明sink
a1.sinks.k1.type = logger


#声明channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100


#绑定关系
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

3. 开发中心服务器Agent

hadoop01 hadoop02

```
#配置Agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1


#声明Source
a1.sources.r1.type = avro
a1.sources.r1.bind = 0.0.0.0
a1.sources.r1.port = 44444


#声明sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = hdfs://hadoop01:9000/flux/reportTime=%Y-%m-%d
a1.sinks.k1.hdfs.rollInterval = 30
a1.sinks.k1.hdfs.rollSize = 0
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.timeZone = GMT+8


#声明channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100


#绑定关系
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

4. 遇到的问题

a. 找不到hadoop jar包

flume中的hdfs sink需要hadoop相关jar包的支持，

要么手动将hadoop相关jar包放置到flume的lib目录下

要么在本机中解压hadoop并将hadoop路径配置为HADOOP_HOME环境变量，使flume可以自动找到这些jar。

b. 产生大量小文件

hdfs sink的滚动条件设置不合理。

修改即可

```
a1.sinks.k1.hdfs.rollInterval = 30
a1.sinks.k1.hdfs.rollSize = 0
a1.sinks.k1.hdfs.rollCount = 0
```

c. 文件内容为乱码(序列化文件无法直接查看)

hdfs sink默认产生SequenceFile文件，无法直接查看

修改即可：

```
1  a1.sinks.k1.hdfs.fileType = DataStream
```

d. 希望能够按日期分目录存储

为了支持hive的分区处理，hdfs sink在将日志写入到hdfs的过程中，希望按照日期分目录存储。

```
1  a1.sinks.k1.hdfs.path = hdfs://hadoop01:9000/flux/reportTime=%Y-%m-%d
```

并且通过拦截器在日志头中增加timestamp头

```
1  a1.sources.r1.interceptors = i1
2  a1.sources.r1.interceptors.i1.type = regex_extractor
3  a1.sources.r1.interceptors.i1.serializers = s1
4  a1.sources.r1.interceptors.i1.serializers.s1.name = timestamp
5
```

e. 生成的目录时间不正确

配置hdfs采用的时区

```
1  a1.sinks.k1.hdfs.timeZone = GMT+8
```

Flux_离线处理_数据清洗

2018年9月17日 11:27

1. 数据清洗概述

a. 存在的问题

在大数据开发中，数据本身就存在各种各样的缺陷,包括但不限于：

1）数据格式不统一
2）存在字段缺失
3）存在字段取值范围错误
4）数据需要预处理

b. 解决方案

1）数据格式不统一
编写预处理程序，将数据格式统一起来
2）存在字段缺失
丢弃数据、补充固定值、补充推测值、拿出来单独处理
无论怎样处理，根本上要遵循业务人员的意见
3）存在字段取值范围错误
根据规则检测数据范围错误的存在，进行处理
处理的手段 丢弃数据 改为固定值 改为推测值 单独处理
无论怎样检测 和 处理 ，根本上要遵循业务人员的意见
4）数据需要预处理
按照业务的预处理的需求，进行数据的预处理

c. 技术选择

数据清洗没有工具技术上的限制，什么工具技术合适就用什么工具技术
shell python java mr hive

2. 网站流量分析项目中的数据清洗

a. 清洗目标

只保留需要的字段
将会话信息拆分 为 会话编号 会话页面数 会话时间
url urlname ref uagent uvid ssid sscoutn sstime cip

b. 创建外部分区表管理已经在HDFS的流量数据

```
1  create external table flux(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_uv string,stat_ss string,cip string) partitioned by (reportTime string)  row format delimited fields terminated by '|' location '/flux';
```

c. 增加flux的分区信息

```
1  alter table flux add partition(reportTime='2018-09-17')  location '/flux/reportTime=2018-09-17';
```

d. 创建数据清洗表dataclear

```
1  create table dataclear (url string,urlname string,ref string,uagent string,uvid string,ssid string,sscoutn string,sstime string,cip string) partitioned by (reportTime string) row format delimited fields terminated by '|';
```

e. 从zebra表中导入数据到dataclear表，在这个过程中完成数据清洗

```
1  insert into dataclear partition(reportTime='2018-09-17') select url,urlname,ref,uagent,stat_uv,split(stat_ss,'_')[0],split(stat_ss,'_')[1],split(stat_ss,'_')[2],cip from flux where reportTime = '2018-09-17';
```

Flux_离线处理_数据处理

2018年9月17日 14:45

1. 利用Hive实现业务指标的计算

a. PV

访问量，一天之内访问的总量，有多少条日志就是多少个访问量。

```
select count(*) as pv from dataclear where reportTime='2018-09-17';
```

b. UV

独立访客数，一天之内用户的总数，将一天内所有日志的uvid去重后计数。

```
select count(distinct uvid) as uv from dataclear where reportTime='2018-09-17';
```

c. VV

会话总数，一天之内会话的总的数量，将一天内所有的日志的ssid去重后计数。

```
select count(distinct ssid) as vv from dataclear where reportTime='2018-09-17';
```

d. BR

跳出率，一天之内跳出的会话占总的会话的比率。一天内跳出会话的总数/会话的总数。

跳出的会话总数

```
select count(br_tab.ssid) from (select ssid from dataclear where reportTime='2018-09-17' group by ssid having count(*) = 1) as br_tab;
```

会话的总数就是vv

```
select count(distinct ssid) from dataclear where reportTime='2018-09-17';
```

计算跳出率

```
select round(br_left_tab.br_count / br_right_tab.vv_count,4) as br from   (select count(br_tab.ssid) as br_count from (select ssid from dataclear where reportTime='2018-09-17' group by ssid having count(*) = 1) as br_tab) as br_left_tab, (select count(distinct ssid) as vv_count from dataclear where reportTime='2018-09-17') as br_right_tab;
```

e. NewIP

新增IP总数，一天之内新IP的数量。

将一天所有日志的IP去重 后 检查在历史数据从未出现过的数量。

```
select count(distinct dataclear.cip) as newip from dataclear where dataclear.reportTime='2018-09-17' and dataclear.cip not in (select distinct inner_dataclear_tab.cip from dataclear as inner_dataclear_tab where datediff('2018-09-17',inner_dataclear_tab.reportTime)>0);
```

f. NewCust

新增客户总数，一天之内新用户的数量。

将一天内所有日志的uvid去重 后 检查从未在历史数据中出现过的数量。

```
select count(distinct dataclear.uvid) as newcust from dataclear where dataclear.reportTime='2018-09-17' and dataclear.uvid not in (select inner_dataclear_tab.uvid from dataclear as inner_dataclear_tab where datediff('2018-09-17',inner_dataclear_tab.reportTime)>0);
```

g. AvgTime

平均访问时长，一天之内所有会话访问时长的平均值

将一天内所有日志按照会话分组后，求会话内部最后一次访问的时间减去第一次访问的时间就是会话时长，求其平均值。

```
select avg(avgtime_tab.use_time) as avgtime from (select max(sstime) - min(sstime) as use_time from dataclear where reportTime='2018-09-17' group by ssid) as avgtime_tab;
```

h. AvgDeep

平均访问深度，一天内所有会话访问深度的平均值。

将一天内所有日志按照会话分组后，统计每个会话访问的页面去重后的总数为会话的访问深度，再求这些会话访问深度的平均值。

```
select round(avg(avgdeep_tab.deep),4) as avgdeep from (select count(distinct urlname) as deep from dataclear where reportTime='2018-09-17' group by ssid) as avgdeep_tab;
```

2. 将计算结果存入统计表 - 方案1

a. 创建tongji1表

```
create table tongji1 (reportTime string,pv int,uv int,vv int,br double,newip int,newcust int,avgtime double,avgdeep double) row format delimited fields terminated by '|';
```

b. 将计算的结果写入tongji1表

```
insert into tongji1 select
'2018-09-17',tab1.pv,tab2.uv,tab3.vv,tab4.br,tab5.newip,tab6.newcust,tab7.avgtime,tab8.avgdeep from
(select count(*) as pv from dataclear where reportTime='2018-09-17') as tab1, (select
count(distinct uvid) as uv from dataclear where reportTime='2018-09-17') as tab2, (select
count(distinct ssid) as vv from dataclear where reportTime='2018-09-17') as tab3, (select
round(br_left_tab.br_count / br_right_tab.vv_count,4) as br from   (select count(br_tab.ssid) as
br_count from (select ssid from dataclear where reportTime='2018-09-17' group by ssid having
count(*) = 1) as br_tab) as br_left_tab, (select count(distinct ssid) as vv_count from dataclear
where reportTime='2018-09-17') as br_right_tab) as tab4, (select count(distinct dataclear.cip) as
newip from dataclear where dataclear.reportTime='2018-09-17' and dataclear.cip not in (select
distinct inner_dataclear_tab.cip from dataclear as inner_dataclear_tab where
datediff('2018-09-17',inner_dataclear_tab.reportTime)>0)) as tab5, (select count(distinct
dataclear.uvid) as newcust from dataclear where dataclear.reportTime='2018-09-17' and
dataclear.uvid not in (select inner_dataclear_tab.uvid from dataclear as inner_dataclear_tab where
datediff('2018-09-17',inner_dataclear_tab.reportTime)>0)) as tab6, (select
avg(avgtime_tab.use_time) as avgtime from (select max(sstime) - min(sstime) as use_time from
dataclear where reportTime='2018-09-17' group by ssid) as avgtime_tab) as tab7, (select
round(avg(avgdeep_tab.deep),4) as avgdeep from (select count(distinct urlname) as deep from
dataclear where reportTime='2018-09-17' group by ssid) as avgdeep_tab) as tab8;
```

**这种方式通过连接查询实现 将多个查询结果插入一张tongji1表，实现了效果，但是过多的表的连接效率低下，且任意一个mr出错，整个程序要重新计算，可靠性较低。

3. 将计算结果存入统计表 - 方案2

创建过度用表tongji1_temp

create table

执行各个指标的运算，将结果存入tongji1_temp

```
insert into tongji1_temp  select '2018-09-17','pv',t1.pv from (select count(*) as pv from dataclear
where reportTime='2018-09-17') as t1;


insert into tongji1_temp  select '2018-09-17','uv',t2.uv from (select count(distinct uvid) as uv
from dataclear where reportTime='2018-09-17') as t2;


insert into tongji1_temp  select '2018-09-17','vv',t3.vv from (select count(distinct ssid) as vv
from dataclear where reportTime='2018-09-17') as t3;


insert into tongji1_temp  select '2018-09-17','br',t4.br from (select round(br_left_tab.br_count /
br_right_tab.vv_count,4) as br from   (select count(br_tab.ssid) as br_count from (select ssid from
dataclear where reportTime='2018-09-17' group by ssid having count(*) = 1) as br_tab) as
br_left_tab, (select count(distinct ssid) as vv_count from dataclear where reportTime='2018-09-17')
as br_right_tab) as t4;


insert into tongji1_temp  select '2018-09-17','newip',t5.newip from (select count(distinct
dataclear.cip) as newip from dataclear where dataclear.reportTime='2018-09-17' and dataclear.cip
not in (select distinct inner_dataclear_tab.cip from dataclear as inner_dataclear_tab where
datediff('2018-09-17',inner_dataclear_tab.reportTime)>0)) as t5;


insert into tongji1_temp  select '2018-09-17','newcust',t6.newcust from (select count(distinct
dataclear.uvid) as newcust from dataclear where dataclear.reportTime='2018-09-17' and
dataclear.uvid not in (select inner_dataclear_tab.uvid from dataclear as inner_dataclear_tab where
datediff('2018-09-17',inner_dataclear_tab.reportTime)>0)) as t6;


insert into tongji1_temp  select '2018-09-17','avgtime',t7.avgtime from (select
avg(avgtime_tab.use_time) as avgtime from (select max(sstime) - min(sstime) as use_time from
dataclear where reportTime='2018-09-17' group by ssid) as avgtime_tab) as t7;


insert into tongji1_temp  select '2018-09-17','avgdeep',t8.avgdeep from (select
round(avg(avgdeep_tab.deep),4) as avgdeep from (select count(distinct urlname) as deep from
dataclear where reportTime='2018-09-17' group by ssid) as avgdeep_tab) as t8;
```

将tongji1_temp表中的数据导入到tongji1表中：

```
insert into tongji1 select '2018-09-17',t1.pv,t2.uv,t3.vv,t4.br,t5.newip, t6.newcust, t7.avgtime,
t8.avgdeep from  (select value as pv from tongji1_temp where field='pv' and
reportTime='2018-09-17') as t1, (select value as uv from tongji1_temp where field='uv' and
reportTime='2018-09-17') as t2, (select value as vv from tongji1_temp where field='vv' and
reportTime='2018-09-17') as t3, (select value as br from tongji1_temp where field='br' and
reportTime='2018-09-17') as t4, (select value as newip from tongji1_temp where field='newip' and
reportTime='2018-09-17') as t5, (select value as newcust from tongji1_temp where field='newcust'
and reportTime='2018-09-17') as t6, (select value as avgtime from tongji1_temp where
field='avgtime' and reportTime='2018-09-17') as t7, (select value as avgdeep from tongji1_temp
where field='avgdeep' and reportTime='2018-09-17') as t8;
```

**这种方案，分别进行数据运算，存入临时表，再从临时表向tongji1表导入数据，降低了sql的复杂度，有利于提升效率，此外任意一个hql出错，只需从该hql重跑即可，之前的hql可以不用重跑。