网站统计中的数据收集原理及实现

作者 张洋 | 发布于 2012-10-24

网站统计 埋点 Web Openresty

网站数据统计分析工具是网站站长和运营人员经常使用的一种工具，比较常用的有谷歌分析、百度统计和腾讯分析等等。所有这些统计分析工具的第一步都是网站访问数据的收集。目前主流的数据收集方式基本都是基于javascript的。本文将简要分析这种数据收集的原理，并一步一步实际搭建一个实际的数据收集系统。

数据收集原理分析

简单来说，网站统计分析工具需要收集到用户浏览目标网站的行为（如打开某网页、点击某按钮、将商品加入购物车等）及行为附加数据（如某下单行为产生的订单金额等）。早期的网站统计往往只收集一种用户行为：页面的打开。而后用户在页面中的行为均无法收集。这种收集策略能满足基本的流量分析、来源分析、内容分析及访客属性等常用分析视角，但是，随着ajax技术的广泛使用及电子商务网站对于电子商务目标的统计分析的需求越来越强烈，这种传统的收集策略已经显得力不能及。

后来，Google在其产品谷歌分析中创新性的引入了可定制的数据收集脚本，用户通过谷歌分析定义好的可扩展接口，只需编写少量的javascript代码就可以实现自定义事件和自定义指标的跟踪和分析。目前百度统计、搜狗分析等产品均照搬了谷歌分析的模式。

其实说起来两种数据收集模式的基本原理和流程是一致的，只是后一种通过javascript收集到了更多的信息。下面看一下现在各种网站统计工具的数据收集基本原理。

流程概览

首先通过一幅图总体看一下数据收集的基本流程。

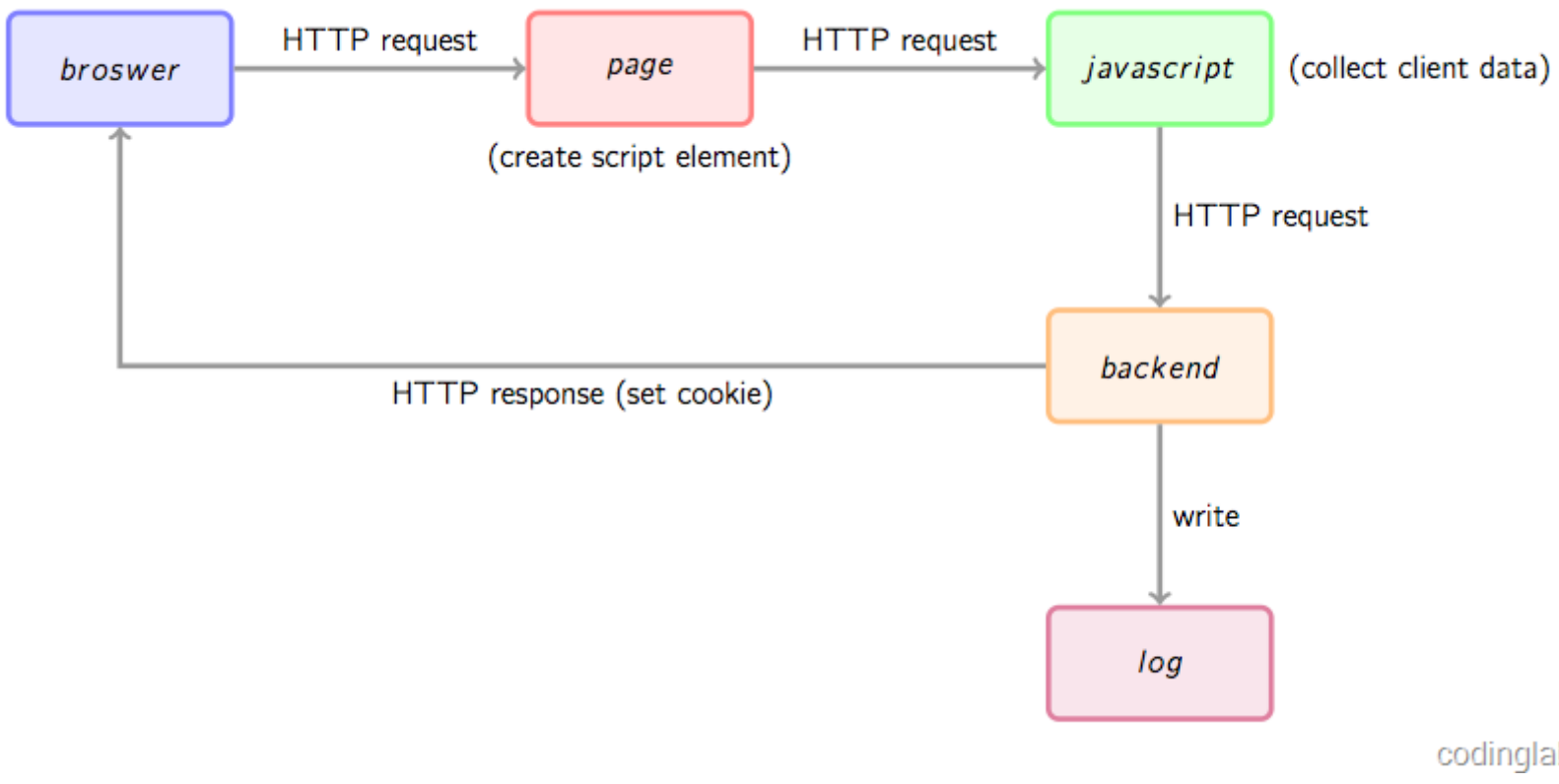

图1. 网站统计数据收集基本流程

首先，用户的行为会触发浏览器对被统计页面的一个http请求，这里姑且先认为行为就是打开网页。当网页被打开，页面中的埋点javascript片段会被执行，用过相关工具的朋友应该知道，一般网站统计工具都会要求用户在网页中加入一小段javascript代码，这个代码片段一般会动态创建一个script标签，并将src指向一个单独的js文件，此时这个单独的js文件（图1中绿色节点）会被浏览器请求到并执行，这个js往往就是真正的数据收集脚本。数据收集完成后，js会请求一个后端的数据收集脚本（图1中的backend），这个脚本一般是一个伪装成图片的动态脚本程序，可能由php、python或其它服务端语言编写，js会将收集到的数据通过http参数的方式传递给后端脚本，后端脚本解析参数并按固定格式记录到访问日志，同时可能会在http响应中给客户端种植一些用于追踪的cookie。

上面是一个数据收集的大概流程，下面以谷歌分析为例，对每一个阶段进行一个相对详细的分析。

埋点脚本执行阶段

若要使用谷歌分析（以下简称GA），需要在页面中插入一段它提供的javascript片段，这个片段往往被称为埋点代码。下面是我的博客中所放置的谷歌分析埋点代码截图：

```
<script type="text/javascript">
var _gaq = _gaq || [];
_gaq.push(['_setAccount', 'UA-35712773-1']);
_gaq.push(['_trackPageview']);

(function() {
        var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
        ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';
        var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
})();
</script>
```

codinglabs.org

图2. 谷歌分析埋点代码

其中_gaq是GA的的全局数组，用于放置各种配置，其中每一条配置的格式为：

```
1.  _gaq.push(['Action', 'param1', 'param2', ...]);
```

Action指定配置动作，后面是相关的参数列表。GA给的默认埋点代码会给出两条预置配置，_setAccount用于设置网站标识ID，这个标识ID是在注册GA时分配的。_trackPageview告诉GA跟踪一次页面访问。更多配置请参考：https://developers.google.com/analytics/devguides/collection/gajs/。实际上，这个_gaq是被当做一个FIFO队列来用的，配置代码不必出现在埋点代码之前，具体请参考上述链接的说明。

就本文来说，_gaq的机制不是重点，重点是后面匿名函数的代码，这才是埋点代码真正要做的。这段代码的主要目的就是引入一个外部的js文件（ga.js），方式是通过document.createElement方法创建一个script并根据协议（http或https）将src指向对应的ga.js，最后将这个element插入页面的dom树上。

注意ga.async = true的意思是异步调用外部js文件，即不阻塞浏览器的解析，待外部js下载完成后异步执行。这个属性是HTML5新引入的。

数据收集脚本执行阶段

数据收集脚本（ga.js）被请求后会被执行，这个脚本一般要做如下几件事：

1、通过浏览器内置javascript对象收集信息，如页面title（通过document.title）、referrer（上一跳url，通过document.referrer）、用户显示器分辨率（通过windows.screen）、cookie信息（通过document.cookie）等等一些信息。

2、解析_gaq收集配置信息。这里面可能会包括用户自定义的事件跟踪、业务数据（如电子商务网站的商品编号等）等。

3、将上面两步收集的数据按预定义格式解析并拼接。

4、请求一个后端脚本，将信息放在http request参数中携带给后端脚本。

这里唯一的问题是步骤4，javascript请求后端脚本常用的方法是ajax，但是ajax是不能跨域请求的。这里ga.js在被统计网站的域内执行，而后端脚本在另外的域（GA的后端统计脚本是http://www.google-analytics.com/__utm.gif），ajax行不通。一种通用的方法是js脚本创建一个Image对象，将Image对象的src属性指向后端脚本并携带参数，此时即实现了跨域请求后端。这也是后端脚本为什么通常伪装成gif文件的原因。通过http抓包可以看到ga.js对__utm.gif的请求：

图3. 后端脚本请求的http包

可以看到ga.js在请求__utm.gif时带了很多信息，例如utmsr=1280×1024是屏幕分辨率，utmac=UA-35712773-1是_gaq中解析出的我的GA标识ID等等。

值得注意的是，__utm.gif未必只会在埋点代码执行时被请求，如果用_trackEvent配置了事件跟踪，则在事件发生时也会请求这个脚本。

由于ga.js经过了压缩和混淆，可读性很差，我们就不分析了，具体后面实现阶段我会实现一个功能类似的脚本。

后端脚本执行阶段

GA的__utm.gif是一个伪装成gif的脚本。这种后端脚本一般要完成以下几件事情：

1、解析http请求参数的到信息。

2、从服务器（WebServer）中获取一些客户端无法获取的信息，如访客ip等。

3、将信息按格式写入log。

5、生成一副1×1的空gif图片作为响应内容并将响应头的Content-type设为image/gif。

5、在响应头中通过Set-cookie设置一些需要的cookie信息。

之所以要设置cookie是因为如果要跟踪唯一访客，通常做法是如果在请求时发现客户端没有指定的跟踪cookie，则根据规则生成一个全局唯一的cookie并种植给用户，否则Set-cookie中放置获取到的跟踪cookie以保持同一用户cookie不变（见图4）。

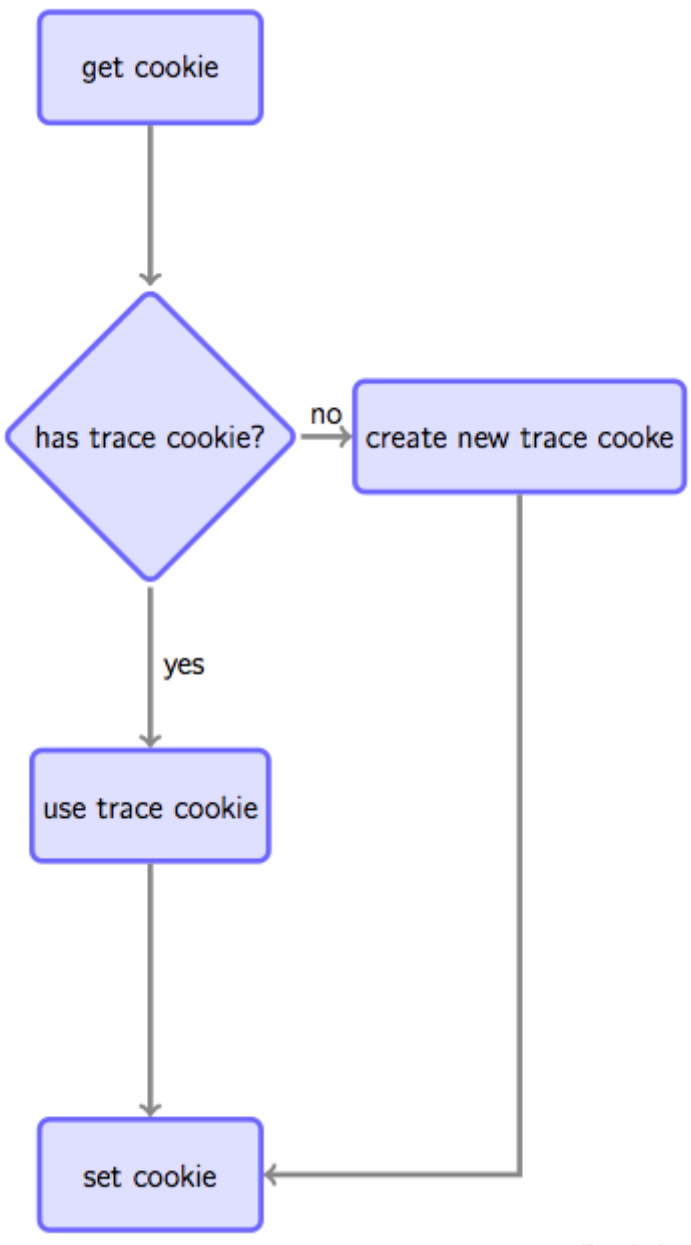

图4. 通过cookie跟踪唯一用户的原理

这种做法虽然不是完美的（例如用户清掉cookie或更换浏览器会被认为是两个用户），但是是目前被广泛使用的手段。注意，如果没有跨站跟踪同一用户的需求，可以通过js将cookie种植在被统计站点的域下（GA是这么做的），如果要全网统一定位，则通过后端脚本种植在服务端域下（我们待会的实现会这么做）。

系统的设计实现

根据上述原理，我自己搭建了一个访问日志收集系统。总体来说，搭建这个系统要做如下的事：

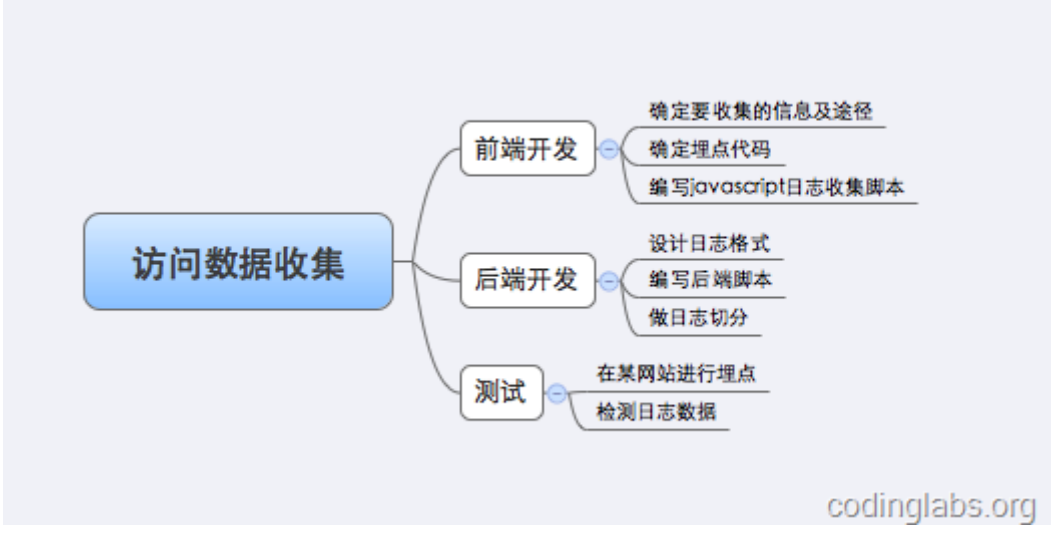

图5. 访问数据收集系统工作分解

下面详述每一步的实现。我将这个系统叫做MyAnalytics。

确定收集的信息

为了简单起见，我不打算实现GA的完整数据收集模型，而是收集以下信息。

名称	途径	备注
访问时间	web server	Nginx $msec
IP	web server	Nginx $remote_addr
域名	javascript	document.domain
URL	javascript	document.URL
页面标题	javascript	document.title
分辨率	javascript	window.screen.height & width
颜色深度	javascript	window.screen.colorDepth
Referrer	javascript	document.referrer
浏览客户端	web server	Nginx $http_user_agent
客户端语言	javascript	navigator.language
访客标识	cookie	
网站标识	javascript	自定义对象

埋点代码

埋点代码我将借鉴GA的模式，但是目前不会将配置对象作为一个FIFO队列用。一个埋点代码的模板如下：

```
<script type="text/javascript">
var _maq = _maq || [];
_maq.push(['_setAccount', '网站标识']);

(function() {
    var ma = document.createElement('script'); ma.type = 'text/javascript'; ma.async = true;
    ma.src = ('https:' == document.location.protocol ? 'https://analytics' : 'http://analytics') + '.codinglabs.org/ma.js';
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ma, s);
})();
</script>
```

这里我启用了二级域名analytics.codinglabs.org，统计脚本的名称为ma.js。当然这里有一点小问题，因为我并没有https的服务器，所以如果一个https站点部署了代码会有问题，不过这里我们先忽略吧。

前端统计脚本

我写了一个不是很完善但能完成基本工作的统计脚本ma.js：

```
(function () {
    var params = {};
    //Document对象数据
    if(document) {
        params.domain = document.domain || '';
        params.url = document.URL || '';
        params.title = document.title || '';
        params.referrer = document.referrer || '';
    }
    //window对象数据
    if(window && window.screen) {
        params.sh = window.screen.height || 0;
        params.sw = window.screen.width || 0;
        params.cd = window.screen.colorDepth || 0;
```

```
    }
    //navigator对象数据
    if(navigator) {
        params.lang = navigator.language || '';
    }
    //解析_maq配置
    if(_maq) {
        for(var i in _maq) {
            switch(_maq[i][0]) {
                case '_setAccount':
                    params.account = _maq[i][1];
                    break;
                default:
                    break;
            }
        }
    }
    //拼接参数串
    var args = '';
    for(var i in params) {
        if(args != '') {
            args += '&';
        }
        args += i + '=' + encodeURIComponent(params[i]);
    }

    //通过Image对象请求后端脚本
    var img = new Image(1, 1);
    img.src = 'http://analytics.codinglabs.org/1.gif?' + args;
})();
```

整个脚本放在匿名函数里，确保不会污染全局环境。功能在原理一节已经说明，不再赘述。其中1.gif是后端脚本。

日志格式

日志采用每行一条记录的方式，采用不可见字符^A（ascii码0x01，Linux下可通过ctrl + v ctrl + a输入，下文均用“^A”表示不可见字符0x01），具体格式如下：

时间^AIP^A域名^AURL^A页面标题^AReferrer^A分辨率高^A分辨率宽^A颜色深度^A语言^A客户端信息^A用户标识^A网站标识

后端脚本

为了简单和效率考虑，我打算直接使用nginx的access_log做日志收集，不过有个问题就是nginx配置本身的逻辑表达能力有限，所以我选用了OpenResty做这个事情。OpenResty是一个基于Nginx扩展出的高性能应用开发平台，内部集成了诸多有用的模块，其中的核心是通过ngx_lua模块集成了Lua，从而在nginx配置文件中可以通过Lua来表述业务。关于这个平台我这里不做过多介绍，感兴趣的同学可以参考其官方网站http://openresty.org/，或者这里有其作者章亦春（agentzh）做的一个非常有爱的介绍OpenResty的slide：http://agentzh.org/misc/slides/ngx-openresty-ecosystem/，关于ngx_lua可以参考：https://github.com/chaoslawful/lua-nginx-module。

首先，需要在nginx的配置文件中定义日志格式：

```
log_format tick
"$msec^A$remote_addr^A$u_domain^A$u_url^A$u_title^A$u_referrer^A$u_sh^A$u_sw^A$u_cd^A$u_lang^A$http_user_agent^A$u_utrace^A$u_account";
```

注意这里以u_开头的是我们待会会自己定义的变量，其它的是nginx内置变量。

然后是核心的两个location：

```
location /1.gif {
#伪装成gif文件
    default_type image/gif;
#本身关闭access_log，通过subrequest记录log
    access_log off;

    access_by_lua "
        -- 用户跟踪cookie名为__utrace
        local uid = ngx.var.cookie___utrace
        if not uid then
            -- 如果没有则生成一个跟踪cookie，算法为md5(时间戳+IP+客户端信息)
            uid = ngx.md5(ngx.now() .. ngx.var.remote_addr .. ngx.var.http_user_agent)
        end
        ngx.header['Set-Cookie'] = {'__utrace=' .. uid .. '; path=/'}
        if ngx.var.arg_domain then
        -- 通过subrequest到/i-log记录日志，将参数和用户跟踪cookie带过去
            ngx.location.capture('/i-log?' .. ngx.var.args .. '&utrace=' .. uid)
        end
    ";
```

```
20.
21.       #此请求不缓存
22.       add_header Expires "Fri, 01 Jan 1980 00:00:00 GMT";
23.       add_header Pragma "no-cache";
24.       add_header Cache-Control "no-cache, max-age=0, must-revalidate";
25.
26.       #返回一个1x1的空gif图片
27.       empty_gif;
28.   }
29.
30.   location /i-log {
31.       #内部location，不允许外部直接访问
32.       internal;
33.
34.       #设置变量，注意需要unescape
35.       set_unescape_uri $u_domain $arg_domain;
36.       set_unescape_uri $u_url $arg_url;
37.       set_unescape_uri $u_title $arg_title;
38.       set_unescape_uri $u_referrer $arg_referrer;
39.       set_unescape_uri $u_sh $arg_sh;
40.       set_unescape_uri $u_sw $arg_sw;
41.       set_unescape_uri $u_cd $arg_cd;
42.       set_unescape_uri $u_lang $arg_lang;
43.       set_unescape_uri $u_utrace $arg_utrace;
44.       set_unescape_uri $u_account $arg_account;
45.
46.       #打开日志
47.       log_subrequest on;
48.       #记录日志到ma.log，实际应用中最好加buffer，格式为tick
49.       access_log /path/to/logs/directory/ma.log tick;
50.
51.       #输出空字符串
52.       echo '';
53.   }
```

要完全解释这段脚本的每一个细节有点超出本文的范围，而且用到了诸多第三方ngxin模块（全都包含在OpenResty中了），重点的地方我都用注释标出来了，可以不用完全理解每一行的意义，只要大约知道这个配置完成了我们在原理一节提到的后端逻辑就可以了。

日志轮转

真正的日志收集系统访问日志会非常多，时间一长文件变得很大，而且日志放在一个文件不便于管理。所以通常要按时间段将日志切分，例如每天或每小时切分一个日志。我这里为了效果明显，每一小时切分一个日志。我是通过crontab定时调用一个shell脚本实现的，shell脚本如下：

```
1.  _prefix="/path/to/nginx"
2.  time=`date +%Y%m%d%H`
3.
4.  mv ${_prefix}/logs/ma.log ${_prefix}/logs/ma/ma-${time}.log
5.  kill -USR1 `cat ${_prefix}/logs/nginx.pid`
```

这个脚本将ma.log移动到指定文件夹并重命名为ma-{yyyymmddhh}.log，然后向nginx发送USR1信号令其重新打开日志文件。

然后再/etc/crontab里加入一行：

```
1.  59  *  *  *  * root /path/to/directory/rotatelog.sh
```

在每个小时的59分启动这个脚本进行日志轮转操作。

测试

下面可以测试这个系统是否能正常运行了。我昨天就在我的博客中埋了相关的点，通过http抓包可以看到ma.js和1.gif已经被正确请求：

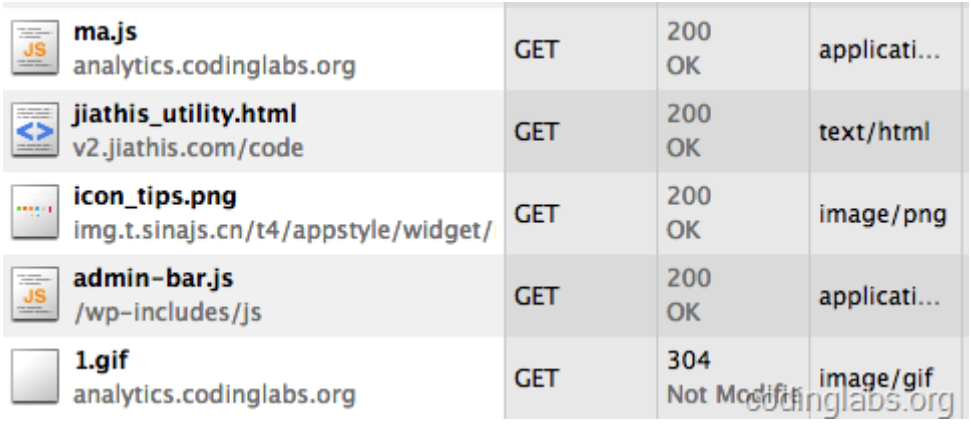

图6. http包分析ma.js和1.gif的请求

同时可以看一下1.gif的请求参数：

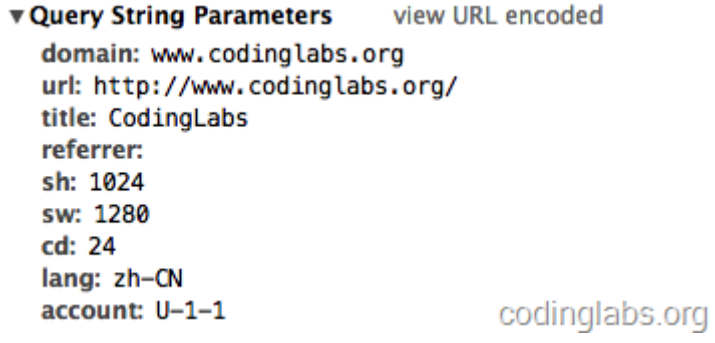

图7. 1.gif的请求参数

相关信息确实也放在了请求参数中。

然后我tail打开日志文件，然后刷新一下页面，因为没有设access log buffer， 我立即得到了一条新日志：

```
1. 1351060731.360^A0.0.0.0^Awww.codinglabs.org^Ahttp://www.codinglabs.org/^ACodingLabs^A^A1024^A1280^A24^Azh-CN^AMozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/537.4 (KHTML, like Gecko) Chrome/22.0.1229.94 Safari/537.4^A4d612be64366768d32e623d594e82678^AU-1-1
```

注意实际上原日志中的^A是不可见的，这里我用可见的^A替换为方便阅读，另外IP由于涉及隐私我替换为了0.0.0.0。

看一眼日志轮转目录，由于我之前已经埋了点，所以已经生成了很多轮转文件：

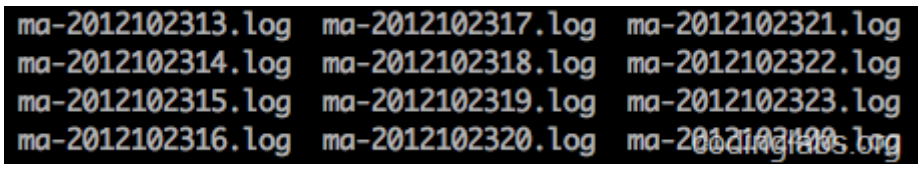

图8. 轮转日志

关于分析

通过上面的分析和开发可以大致理解一个网站统计的日志收集系统是如何工作的。有了这些日志，就可以进行后续的分析了。本文只注重日志收集，所以不会写太多关于分析的东西。

注意，原始日志最好尽量多的保留信息而不要做过多过滤和处理。例如上面的MyAnalytics保留了毫秒级时间戳而不是格式化后的时间，时间的格式化是后面的系统做的事而不是日志收集系统的责任。后面的系统根据原始日志可以分析出很多东西，例如通过IP库可以定位访问者的地域、user agent中可以得到访问者的操作系统、浏览器等信息，再结合复杂的分析模型，就可以做流量、来源、访客、地域、路径等分析了。当然，一般不会直接对原始日志分析，而是会将其清洗格式化后转存到其它地方，如MySQL或HBase中再做分析。

分析部分的工作有很多开源的基础设施可以使用，例如实时分析可以使用Storm，而离线分析可以使用Hadoop。当然，在日志比较小的情况下，也可以通过shell命令做一些简单的分析，例如，下面三条命令可以分别得出我的博客在今天上午8点到9点的访问量（PV），访客数（UV）和独立IP数（IP）：

```
1. awk -F^A '{print $1}' ma-2012102409.log | wc -l
2. awk -F^A '{print $12}' ma-2012102409.log | uniq | wc -l
3. awk -F^A '{print $2}' ma-2012102409.log | uniq | wc -l
```

其它好玩的东西朋友们可以慢慢挖掘。

参考

GA的开发者文档：https://developers.google.com/analytics/devguides/collection/gajs/

一篇关于实现nginx收日志的文章：
http://blog.linezing.com/2011/11/%E4%BD%BF%E7%94%A8nginx%E8%AE%B0%E6%97%A5%E5%BF%97

关于Nginx可以参考：http://wiki.nginx.org/Main

OpenResty的官方网站为：http://openresty.org

ngx_lua模块可参考：https://github.com/chaoslawful/lua-nginx-module

本文http抓包使用Chrome浏览器开发者工具，绘制思维导图使用Xmind，流程和结构图使用Tikz PGF