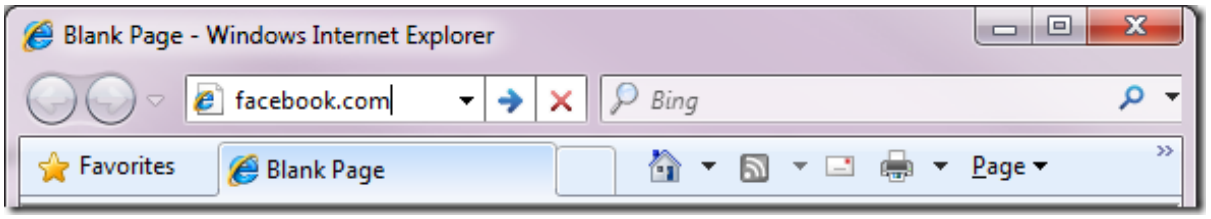
当你输入一个网址的时候，实际会发生什么?

原文:http://igoro.com/archive/what-really-happens-when-you-navigate-to-a-url/

作为一个软件开发者，你一定会对网络应用如何工作有一个完整的层次化的认知，同样这里也包括这些应用所用到的技术：像浏览器，HTTP，HTML，网络服务器，需求处理等等。

本文将更深入的研究当你输入一个网址的时候，后台到底发生了一件件什么样的事～

1. 首先嘛，你得在浏览器里输入要网址:

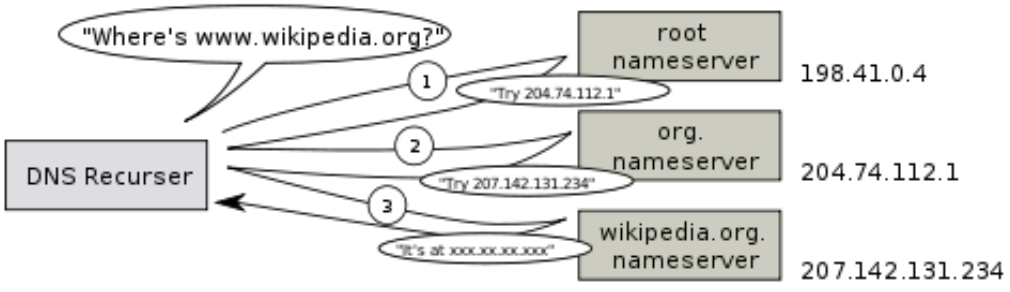


2. 浏览器查找域名的IP地址

导航的第一步是通过访问的域名找出其IP地址。DNS查找过程如下：

- 浏览器缓存 – 浏览器会缓存DNS记录一段时间。 有趣的是，操作系统没有告诉浏览器储存DNS记录的时间，这样不同浏览器会储存个自固定的一个时间（2分钟到30分钟不等）。
- 系统缓存– 如果在浏览器缓存里没有找到需要的记录，浏览器会做一个系统调用（windows里是gethostbyname）。这样便可获得系统缓存中的记录。
- 路由器缓存 – 接着，前面的查询请求发向路由器，它一般会有自己的DNS缓存。
- **ISP DNS** 缓存 – 接下来要check的就是ISP缓存DNS的服务器。在这一般都能找到相应的缓存记录。
- 递归搜索– 你的ISP的DNS服务器从跟域名服务器开始进行递归搜索，从.com顶级域名服务器到Facebook的域名服务器。一般DNS服务器的缓存中会有.com域名服务器中的域名，所以到顶级服务器的匹配过程不是那么必要了。

DNS递归查找如下图所示：



DNS有一点令人担忧，这就是像wikipedia.org 或者 facebook.com这样的整个域名看上去只是对应一个单独的IP地址。还好，有几种方法可以消除这个瓶颈：

- 循环 **DNS**是DNS查找时返回多个IP时的解决方案。举例来说，Facebook.com实际上就对应了四个IP地址。
- 负载平衡器 是以一个特定IP地址进行侦听并将网络请求转发到集群服务器上的硬件设备。 一些大型的站点一般都会使用这种昂贵的高性能负载平衡器。
- 地理 **DNS**根据用户所处的地理位置，通过把域名映射到多个不同的IP地址提高可扩展性。这样不同的服务器不能够更新同步状态，但映射静态内容的话非常好。
- **Anycast**是一个IP地址映射多个物理主机的路由技术。 美中不足，Anycast与TCP协议适应的不是很好，所以很少应用在那些方案中。

大多数DNS服务器使用Anycast来获得高效低延迟的DNS查找。

3. 浏览器给web服务器发送一个HTTP请求

因为像Facebook主页这样的动态页面，打开后在浏览器缓存中很快甚至马上就会过期，毫无疑问他们不能从中读取。

所以，浏览器将把一下请求发送到Facebook所在的服务器：

```
GET http://facebook.com/ HTTP/1.1
 Accept: application/x-ms-application, image/jpeg, application/xaml+xml, [...]
 User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; [...]
 Accept-Encoding: gzip, deflate
 Connection: Keep-Alive
 Host: facebook.com
 Cookie: datr=1265876274-[...]; locale=en_US; lsd=WW[...]; c_user=2101[...]
```

GET 这个请求定义了要读取的**URL**: “http://facebook.com/”。 浏览器自身定义 (**User-Agent**头)， 和它希望接受什么类型的相应 (**Accept** and**Accept-Encoding** 头).**Connection**头要求服务器为了后边的请求不要关闭TCP连接。

请求中也包含浏览器存储的该域名的**cookies**。可能你已经知道，在不同页面请求当中，cookies是与跟踪一个网站状态相匹配的键值。这样cookies会存储登录用户名，服务器分配的密码和一些用户设置等。Cookies会以文本文档形式存储在客户机里，每次请求时发送给服务器。

用来看原始HTTP请求及其相应的工具很多。作者比较喜欢使用fiddler，当然也有像FireBug这样其他的工具。这些软件在网站优化时会帮上很大忙。

除了获取请求，还有一种是发送请求，它常在提交表单用到。发送请求通过URL传递其参数(e.g.: http://robozzle.com/puzzle.aspx?id=85)。发送请求在请求正文头之后发送其参数。

像“http://facebook.com/”中的斜杠是至关重要的。这种情况下，浏览器能安全的添加斜杠。而像“http://example.com/folderOrFile”这样的地址，因为浏览器不清楚folderOrFile到底是文件夹还是文件，所以不能自

动添加 斜杠。这时，浏览器就不加斜杠直接访问地址，服务器会响应一个重定向，结果造成一次不必要的握手。

4. facebook服务的永久重定向响应

图中所示为Facebook服务器发回给浏览器的响应：

```
HTTP/1.1 301 Moved Permanently
 Cache-Control: private, no-store, no-cache, must-revalidate, post-check=0,
 pre-check=0
 Expires: Sat, 01 Jan 2000 00:00:00 GMT
 Location: http://www.facebook.com/
 P3P: CP="DSP LAW"
 Pragma: no-cache
 Set-Cookie: made_write_conn=deleted; expires=Thu, 12-Feb-2009 05:09:50 GMT;
 path=/; domain=.facebook.com; httponly
 Content-Type: text/html; charset=utf-8
 X-Cnection: close
 Date: Fri, 12 Feb 2010 05:09:51 GMT
 Content-Length: 0
```

服务器给浏览器响应一个301永久重定向响应，这样浏览器就会访问“http://www.facebook.com/” 而非“http://facebook.com/”。

为什么服务器一定要重定向而不是直接发会用户想看的网页内容呢？这个问题有好多有意思的答案。

其中一个原因跟搜索引擎排名有 关。你看，如果一个页面有两个地址，就像http://www.igoro.com/ 和 http://igoro.com/，搜索引擎会认为它们是两个网站，结果造成每一个的搜索链接都减少从而降低排名。而搜索引擎知道301永久重定向是 什么意思，这样就会把访问带www的和不带www的地址归到同一个网站排名下。

还有一个是用不同的地址会造成缓存友好性变差。当一个页面有好几个名字时，它可能会在缓存里出现好几次。

5. 浏览器跟踪重定向地址

现在，浏览器知道了“http://www.facebook.com/”才是要访问的正确地址，所以它会发送另一个获取请求：

```
GET http://www.facebook.com/ HTTP/1.1
 Accept: application/x-ms-application, image/jpeg, application/xaml+xml, [...]
 Accept-Language: en-US
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; [...]
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Cookie: lsd=XW[...]; c_user=21[...]; x-referer=[...]
Host: www.facebook.com
```

头信息以之前请求中的意义相同。

6. 服务器"处理"请求

服务器接收到获取请求，然后处理并返回一个响应。

这表面上看起来是一个顺向的任务，但其实这中间发生了很多有意思的东西- 就像作者博客这样简单的网站，何况像facebook那样访问量大的网站呢！

- **Web** 服务器软件
  web服务器软件（像IIS和阿帕奇）接收到HTTP请求，然后确定执行什么请求处理来处理它。请求处理就是一个能够读懂请求并且能生成HTML来进行响应的程序（像ASP.NET,PHP,RUBY...）。

  举 个最简单的例子，需求处理可以以映射网站地址结构的文件层次存储。像 http://example.com/folder1/page1.aspx这个地 址会映射/httpdocs/folder1/page1.aspx这个文件。web服务器软件可以设置成为地址人工的对应请求处理，这样 page1.aspx的发布地址就可以是 http://example.com/folder1/page1。

- 请求处理
  请求处理阅读请求及它的参数和cookies。它会读取也可能更新一些数据，并讲数据存储在服务器上。然后，需求处理会生成一个HTML响应。

所 有动态网站都面临一个有意思的难点 -如何存储数据。小网站一半都会有一个SQL数据库来存储数据，存储大量数据和/或访问量大的网站不得不找一些办法把数据库分配到多台机器上。解决方案 有：sharding （基于主键值讲数据表分散到多个数据库中），复制，利用弱语义一致性的简化数据库。

委 托工作给批处理是一个廉价保持数据更新的技术。举例来讲，Fackbook得及时更新新闻feed，但数据支持下的"你可能认识的人"功能只需要每晚更新 （作者猜测是这样的，改功能如何完善不得而知）。批处理作业更新会导致一些不太重要的数据陈旧，但能使数据更新耕作更快更简洁。

7. 服务器发回一个**HTML**响应

图中为服务器生成并返回的响应：

```
HTTP/1.1 200 OK
```

```
Cache-Control: private, no-store, no-cache, must-revalidate, post-check=0,
pre-check=0
Expires: Sat, 01 Jan 2000 00:00:00 GMT
P3P: CP="DSP LAW"
Pragma: no-cache
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
X-Cnection: close
Transfer-Encoding: chunked
Date: Fri, 12 Feb 2010 09:05:55 GMT

2b3Tn@[...]
```

整个响应大小为35kB，其中大部分在整理后以blob类型传输。

内容编码头告诉浏览器整个响应体用gzip算法进行压缩。解压blob块后，你可以看到如下期望的HTML：

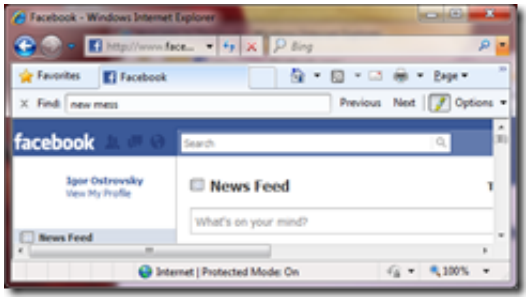
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
 lang="en" id="facebook" class=" no_js">
 <head>
 <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
 <meta http-equiv="Content-language" content="en" />
 ...
```

关于压缩，头信息说明了是否缓存这个页面，如果缓存的话如何去做，有什么cookies要去设置（前面这个响应里没有这点）和隐私信息等等。

请注意报头中把**Content-type**设置为“**text/html**”。报头让浏览器将该响应内容以HTML形式呈现，而不是以文件形式下载它。浏览器会根据报头信息决定如何解释该响应，不过同时也会考虑像URL扩展内容等其他因素。

8. 浏览器开始显示**HTML**

在浏览器没有完整接受全部HTML文档时，它就已经开始显示这个页面了：



9. 浏览器发送获取嵌入在**HTML**中的对象

在浏览器显示HTML时，它会注意到需要获取其他地址内容的标签。这时，浏览器会发送一个获取请求来重新获得这些文件。

下面是几个我们访问facebook.com时需要重获取的几个URL：

- 图片
  http://static.ak.fbcdn.net/rsrc.php/z12E0/hash/8q2anwu7.gif
  http://static.ak.fbcdn.net/rsrc.php/zBS5C/hash/7hwy7at6.gif
  ...
- **CSS** 式样表
  http://static.ak.fbcdn.net/rsrc.php/z448Z/hash/2plh8s4n.css
  http://static.ak.fbcdn.net/rsrc.php/zANE1/hash/cvtutcee.css
  ...
- **JavaScript** 文件
  http://static.ak.fbcdn.net/rsrc.php/zEMOA/hash/c8yzb6ub.js
  http://static.ak.fbcdn.net/rsrc.php/z6R9L/hash/cq2lgbs8.js
  ...

这些地址都要经历一个和HTML读取类似的过程。所以浏览器会在DNS中查找这些域名，发送请求，重定向等等...

但 不像动态页面那样，静态文件会允许浏览器对其进行缓存。有的文件可能会不需要与服务器通讯，而从缓存中直接读取。服务器的响应中包含了静态文件保存的期限 信息，所以浏览器知道要把它们缓存多长时间。还有，每个响应都可能包含像版本号一样工作的ETag头（被请求变量的实体值），如果浏览器观察到文件的版本 ETag信息已经存在，就马上停止这个文件的传输。

试着猜猜看"**fbcdn.net**"在地址中代表什么？聪明的答案是"Facebook内容分发网络"。Facebook利用内容分发网络（CDN）分发像图片，CSS表和JavaScript文件这些静态文件。所以，这些文件会在全球很多CDN的数据中心中留下备份。

静态内容往往代表站点的带宽大小，也能通过CDN轻松的复制。通常网站会使用第三方的CDN。例如，Facebook的静态文件由最大的CDN提供商Akamai来托管。

举例来讲，当你试着ping static.ak.fbcdn.net的时候，可能会从某个akamai.net服务器上获得响应。有意思的是，当你同样再ping一次的时候，响应的服务器可能就不一样，这说明幕后的负载平衡开始起作用了。

10. 浏览器发送异步（**AJAX**）请求

在Web 2.0伟大精神的指引下，页面显示完成后客户端仍与服务器端保持着联系。

以 Facebook聊天功能为例，它会持续与服务器保持联系来及时更新你那些亮亮灰灰的好友状态。为了更新这些头像亮着的好友状态，在浏览器中执行的 JavaScript代码会给服务器发送异步请求。这个异步请求发送给特定的地址，它是一个按照程式构造的获取或发送请求。还是在Facebook这个例 子中，客户端发送给 http://www.facebook.com/ajax/chat/buddy_list.php一个发布请求来获取你好友里哪个 在线的状态信息。

提起这个模式，就必须要讲讲"AJAX"-- “异步JavaScript 和 XML”，虽然服务器为什么用XML格式来进行响应也没有个一清二白的原因。再举个例子吧，对于异步请求，Facebook会返回一些JavaScript的代码片段。

除了其他，fiddler这个工具能够让你看到浏览器发送的异步请求。事实上，你不仅可以被动的做为这些请求的看客，还能主动出击修改和重新发送它们。AJAX请求这么容易被蒙，可着实让那些计分的在线游戏开发者们郁闷的了。（当然，可别那样骗人家~）

Facebook聊天功能提供了关于AJAX一个有意思的问题案例：把数据从服务器端推送到客户端。因为HTTP是一个请求-响应协议，所以聊天服务器不能把新消息发给客户。取而代之的是客户端不得不隔几秒就轮询下服务器端看自己有没有新消息。

这些情况发生时长轮询是个减轻服务器负载挺有趣的技术。如果当被轮询时服务器没有新消息，它就不理这个客户端。而当尚未超时的情况下收到了该客户的新消息，服务器就会找到未完成的请求，把新消息做为响应返回给客户端。

总结一下

希望看了本文，你能明白不同的网络模块是如何协同工作的