Broadview
www.broadview.com.cn

深入分析 Java Web 技术内幕

许令波◎著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

第 1 章

深入 Web 请求过程

随着 Web 2.0 时代的到来，互联网的网络架构已经从传统的 C/S 架构转变到更加方便快捷的 B/S 架构，B/S 架构大大简化了用户使用网络应用的难度，这种人人都能上网、人人都能使用网络上提供的服务的方法也进一步推动了互联网的繁荣。

B/S 架构带来了两方面好处：

- 客户端使用统一的浏览器（Browser）。由于浏览器具有统一性，它不需要特殊的配置和网络连接，有效地屏蔽了不同服务提供商提供给用户使用服务的差异性。另外，最重要的一点是，浏览器的交互特性使得用户使用它非常简便，且用户行为的可继承性非常强，也就是用户只要学会了上网，不管使用的是哪个应用，一旦学会了，在使用其他互联网服务时同样具有了使用经验，因为它们都基于同样的浏览器操作界面。
- 服务端（Server）基于统一的 HTTP 协议。和传统的 C/S 架构使用自定义的应用层协议不同，B/S 架构使用的都是统一的 HTTP 协议。使用统一的 HTTP 协议也为服务提供商简化了开发模式，使得服务器开发者可以采用相对规范的开发模式，这样可以大大节省开发成本。由于使用统一的 HTTP 协议，所以基于 HTTP

协议的服务器就有很多，如 Apache、IIS、Nginx、Tomcat、JBoss 等，这些服务器可以直接拿来使用，不需要服务开发者单独来开发。不仅如此，连开发服务的通用框架都不需要单独开发，服务开发者只需要关注提供服务的应用逻辑，其他一切平台和框架都可以直接拿来使用，所以 B/S 架构同样简化了服务器提供者的开发，从而出现了越来越多的互联网服务。

B/S 网络架构不管对普通用户的使用还是对服务的开发都带来了好处，为互联网的主要参与者、服务使用者和服务开发者降低了学习成本。但是作为互联网应用的开发者，我们还是要清楚，从用户在浏览器里单击某个链接开始，到我们的服务返回结果给浏览器为止，这个过程中到底发生了什么、这其中还需要哪些技术来配合。

所以本章将为你描述这一过程的工作原理，它将涉及浏览器的基本行为和 HTTP 协议的解析过程、DNS 如何解析到对应的 IP 地址、CDN 又是如何工作和设计的，以及浏览器如何渲染出返回的结果等。

1.1 B/S 网络架构概述

B/S 网络架构从前端到后端都得到了简化，都基于统一的应用层协议 HTTP 来交互数据，与大多数传统 C/S 互联网应用程序采用的长连接的交互模式不同，HTTP 协议采用无状态的短连接的通信方式，通常情况下，一次请求就完成了一次数据交互，通常也对应一个业务逻辑，然后这次通信连接就断开了。采用这种方式是为了能够同时服务更多的用户，因为当前互联网应用每天都会处理上亿的用户请求，不可能每个用户访问一次后就一直保持住这个连接。

基于 HTTP 协议本身的特点，目前的 B/S 网络架构大都采用类似于图 1-1 所示的架构设计，既要满足海量用户的访问请求，又要保持用户请求的快速响应，所以现在的网络架构也越来越复杂。

当一个用户在浏览器里输入 www.taobao.com 这个 URL 时，将会发生很多操作。首先它会请求 DNS 把这个域名解析成对应的 IP 地址，然后根据这个 IP 地址在互联网上找到对应的服务器，向这个服务器发起一个 get 请求，由这个服务器决定返回默认的数据资源给访问的用户。在服务器端实际上还有很复杂的业务逻辑：服务器可能有很多台，到底指定哪台服务器来处理请求，这需要一个负载均衡设备来平均分配所有用户的请求；还有请求

的数据是存储在分布式缓存里还是一个静态文件中，或是在数据库里；当数据返回浏览器时，浏览器解析数据发现还有一些静态资源（如 CSS、JS 或者图片）时又会发起另外的 HTTP 请求，而这些请求很可能会在 CDN 上，那么 CDN 服务器又会处理这个用户的请求，大体上一个用户请求会涉及这么多的操作。每一个细节都会影响这个请求最终是否会成功。

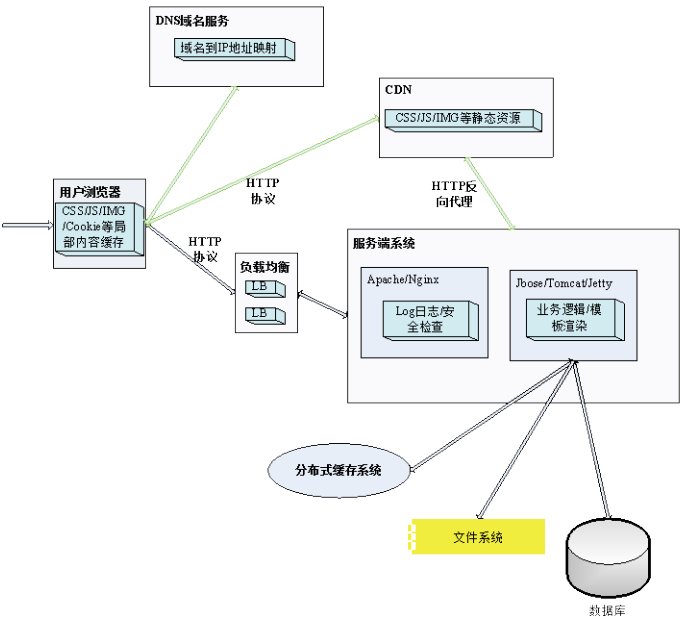

图 1-1　CDN 架构图

不管网络架构如何变化，但是始终有一些固定不变的原则需要遵守。

- 互联网上所有资源都要用一个 URL 来表示。URL 就是统一资源定位符，如果你要发布一个服务或者一个资源到互联网上，让别人能够访问到，那么你首先必须要有一个世界上独一无二的 URL。不要小看这个 URL，它几乎包含了整个互联

网的架构精髓。

- 必须基于 HTTP 协议与服务端交互。不管你要访问的是国内的还是国外的数据、是文本数据还是流媒体，都必须按照套路出牌，也就是都得采用统一打招呼的方式，这样人家才会明白你要的是什么。
- 数据展示必须在浏览器中进行。当你获取到数据资源后，必须在浏览器上才能恢复它的容貌。

只要满足上面的几点，一个互联网应用基本上就能正确地运转起来了，当然这里面还有好多细节，这些细节在后面将分别进行详细讲解。

1.2 如何发起一个请求

如何发起一个 HTTP 请求？这个问题似乎既简单又复杂，简单是指当你在浏览器里输入一个 URL 时，按回车键后这个 HTTP 请求就发起了，很快你就会看到这个请求的返回结果。复杂是指能否不借助浏览器也能发起请求，这里的“不借助”有两层含义，一是指能不能自己组装一个符合 HTTP 协议的数据包，二是处理浏览器还有哪些方式也能简单地发起一个 HTTP 请求。下面就按照这两层含义来解释如何发起一个 HTTP 请求。

如何发起一个 HTTP 请求和如何建立一个 Socket 连接区别不大，只不过 outputStream.write 写的二进制字节数据格式要符合 HTTP 协议。浏览器在建立 Socket 连接之前，必须根据地址栏里输入的 URL 的域名 DNS 解析出 IP 地址，再根据这个 IP 地址和默认 80 端口与远程服务器建立 Socket 连接，然后浏览器根据这个 URL 组装成一个 get 类型的 HTTP 请求头，通过 outputStream.write 发送到目标服务器，服务器等待 inputStream.read 返回数据，最后断开这个连接。

当然，不同浏览器在如何使用这个已经建立好的连接，以及根据什么规则来管理连接，有各种不同的实现方法。一句话，发起一个 HTTP 请求的过程就是建立一个 Socket 通信的过程。

既然发起一个 HTTP 连接本质上就是建立一个 Socket 连接，那么我们完全可以模拟浏览器来发起 HTTP 请求，这很好实现，也有很多方法实现，如 HttpClient 就是一个开源的通过程序实现的处理 HTTP 请求的工具包。当然如果你对 HTTP 协议的数据结构非常熟悉，

你完全可以自己再实现另外一个 HttpClient，甚至可以自己写个简单的浏览器。

下面是一个基本的 HttpClient 的调用示例：

```
    HttpClient httpClient = createHttpClient();
           PostMethod postMethod;
           String domainName = Switcher.domain;
           postMethod = new PostMethod(domainName);
           postMethod.addRequestHeader("Content-Type", "application/x-www-form-
urlencoded; charset=GBK");
           for (FilterData filterData : filterDatas) {
               postMethod.addParameter("ip", filterData.ip);
               postMethod.addParameter("count", String.valueOf(filterData.count));
           }
           try {
               httpClient.executeMethod(postMethod);
               postMethod.getResponseBodyAsString();
           } catch (Exception e) {
               logger.error(e);
           }
```

处理 Java 中使用非常普遍的 HttpClient 还有很多类似的工具，如 Linux 中的 curl 命令，通过 curl + URL 就可以简单地发起一个 HTTP 请求，非常方便。

例如，curl "http://item.taobao.com/item.htm?id=1264" 可以返回这个页面的 HTML 数据，如图 1-2 所示。

```
[junshan@v101055.sqa.cm4 admin]$ curl "http://item.taobao.com/item.htm?id=12643598611" >/dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  149k    0  149k    0     0  3819k      0 --:--:-- --:--:-- --:--:-- 48.3M
```

图 1-2　HTTP 请求返回的 HTML 数据

也可以查看这次访问的 HTTP 协议头的信息，加上-I 选项，如图 1-3 所示。

```
[junshan@v101055.sqa.cm4 admin]$ curl "http://item.taobao.com/item.htm?id=12643598611" -I
HTTP/1.1 200 OK
Date: Sat, 25 Feb 2012 08:33:06 GMT
Server: Apache
at_autype: 5_33483998
at_cat: item_50010850
X-Category: /cat/16
at_nick: %E5%A4%A9%E9%B9%85%E7%BA%B5%E6%A8%AA
at_itemId: 12643598611
at_isb: 0
Set-Cookie: cookie2=f050a98bd189c08c09e46c6d934b60b3;Domain=.taobao.com;Path=/;HttpOnly
Set-Cookie: _tb_token_=ee5ea36ee3366;Domain=.taobao.com;Path=/;HttpOnly
Set-Cookie: t=06f338afdea3d0c1f0306defa65fe4d3; Domain=.taobao.com; Expires=Fri, 25-May-2012 08:33:06 GMT; Path=/
Set-Cookie: v=0; Domain=.taobao.com; Path=/
Content-Language: zh-CN
Vary: Accept-Encoding
Connection: close
Content-Type: text/html;charset=GBK
```

图 1-3　HTTP 协议头的信息

还可以在访问这个 URL 时增加 HTTP 头，通过-HI 选项实现，如图 1-4 所示。

图 1-4　访问 URL 时增加 HTTP 头

因为缺少 Cookie 信息，所以上面的访问返回 302 状态码，必须增加 Cookie 才能正确访问该链接，如下所示：

```
    [junshan@v101055.sqa.cm4 admin]$ curl -I "http://switch.taobao.com:9999/
repository.htm" -H "Cookie:cna=sd0/BjeZulwCAfIdAHkzZZqC; _t_track=121.0.29.
242.1320938379988839;"
    HTTP/1.1 200 OK
    Date: Sat, 25 Feb 2012 08:41:20 GMT
    Server: Apache
    Last-Modified: Thu, 01 Jan 1970 00:00:00 GMT
    Vary: Accept-Encoding
    Content-Type: text/html;charset=GBK
```

1.3　HTTP 协议解析

B/S 网络架构的核心是 HTTP 协议，掌握 HTTP 协议对一个从事互联网工作的程序员来说非常重要，也许你已经非常熟悉 HTTP 协议，这里除了简单介绍 HTTP 协议的基本知识外，还将侧重介绍实际使用中的一些心得，后面将以实际使用的场景为例进行介绍。

要理解 HTTP 协议，最重要的就是要熟悉 HTTP 协议中的 HTTP Header，HTTP Header 控制着互联网上成千上万的用户的数据的传输。最关键的是，它控制着用户浏览器的渲染行为和服务器的执行逻辑。例如，当服务器没有用户请求的数据时就会返回一个 404 状态码，告诉浏览器没有要请求的数据，通常浏览器就会展示一个非常不愿意看到的该页面不存在的错误信息。

常见的 HTTP 请求头和响应头分别如表 1-1 和表 1-2 所示，常见的 HTTP 状态码如表 1-3 所示。

表 1-1　常见的 HTTP 请求头

请　求　头	说　　明
Accept-Charset	用于指定客户端接受的字符集
Accept-Encoding	用于指定可接受的内容编码，如 Accept-Encoding:gzip.deflate
Accept-Language	用于指定一种自然语言，如 Accept-Language:zh-cn
Host	用于指定被请求资源的 Internet 主机和端口号，如 Host:www.taobao.com
User-Agent	客户端将它的操作系统、浏览器和其他属性告诉服务器
Connection	当前连接是否保持，如 Connection: Keep-Alive

表 1-2　常见的 HTTP 响应头

响　应　头	说　　明
Server	使用的服务器名称，如 Server: Apache/1.3.6 (Unix)
Content-Type	用来指明发送给接收者的实体正文的媒体类型，如 Content-Type:text/html;charset=GBK
Content-Encoding	与请求报头 Accept-Encoding 对应，告诉浏览器服务端采用的是什么压缩编码
Content-Language	描述了资源所用的自然语言，与 Accept-Language 对应
Content-Length	指明实体正文的长度，用以字节方式存储的十进制数字来表示
Keep-Alive	保持连接的时间，如 Keep-Alive: timeout=5, max=120

表 1-3　常见的 HTTP 状态码

状　态　码	说　　明
200	客户端请求成功
302	临时跳转，跳转的地址通过 Location 指定
400	客户端请求有语法错误，不能被服务器识别
403	服务器收到请求，但是拒绝提供服务
404	请求的资源不存在
500	服务器发生不可预期的错误

要看一个 HTTP 请求的请求头和响应头，可以通过很多浏览器插件来看，在 Firefox 中有 Firebug 和 HttpFox，Chrome 自带的开发工具也可以看到每个请求的请求头信息（可用 F12 快捷键打开），IE 自带的调试工具也有类似的功能。

1.3.1 查看 HTTP 信息的工具

有时候我们需要知道一个 HTTP 请求到底返回什么数据，或者没有返回数据时想知道是什么原因造成的，这时我们就需要借助一些工具来查询这次请求的详细信息。

在 Windows 下现在主流的浏览器都有很多工具来查看当前请求的详细 HTTP 协议信息，如在 Firefox 浏览器下，使用最多的是 Firebug，如图 1-5 所示。

图 1-5 Firefox 浏览器下 HTTP 协议的信息

还有一个 HttpFox 工具提供的信息更全，如图 1-6 所示，所有 HTTP 相关信息都可以一目了然。

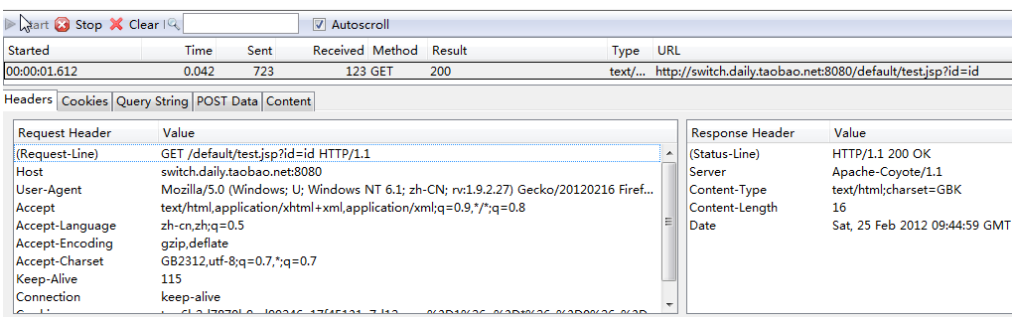

图 1-6 HttpFox 工具显示的 HTTP 协议的信息

Chrome 浏览器下也有一些类似的工具，如 Google 自带的调试工具，同样可以查看到这次请求的相关信息，如图 1-7 所示。

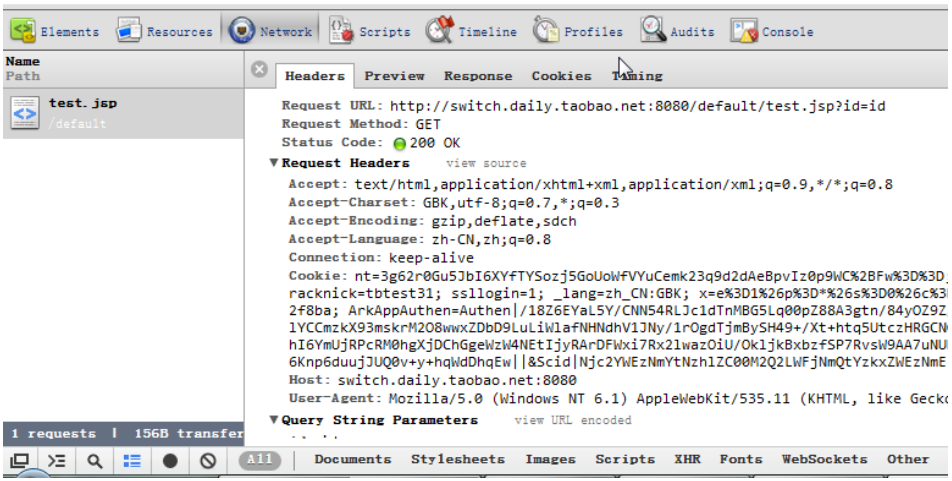

图 1-7　Google 自带的调试工具显示的 HTTP 协议的信息

Chrome 下也有类似的 Firebug 工具，但是还不够完善。

IE 从 7.0 版本开始也提供了类似的 HTTP 调试工具，如自带的开发人员工具可以通过 F12 键打开，HttpFox 插件也有 IE 版本，读者可以试着安装一下。

1.3.2　浏览器缓存机制

浏览器缓存是一个比较复杂但是又比较重要的机制，在我们浏览一个页面时发现有异常的情况下，通常考虑的就是是不是浏览器做了缓存，所以一般的做法就是按 Ctrl+F5 组合键重新请求一次这个页面，重新请求的页面肯定是最新的页面。为什么重新请求就一定能够请求到没有缓存的页面呢？首先是在浏览器端，如果是按 Ctrl+F5 组合键刷新页面，那么浏览器会直接向目标 URL 发送请求，而不会使用浏览器缓存的数据；其次即使请求发送到服务端，也有可能访问到的是缓存的数据，比如，在我们的应用服务器的前端部署一个缓存服务器，如 Varnish 代理，那么 Varnish 也可能直接使用缓存数据。所以为了保证用户能够看到最新的数据，必须通过 HTTP 协议来控制。

当我们使用 Ctrl+F5 组合键刷新一个页面时，在 HTTP 的请求头中会增加一些请求头，

它告诉服务端我们要获取最新的数据而不是缓存。

如图 1-8 所示，这次请求没有发送到服务端，使用的是浏览器缓存数据，按 Ctrl+F5 组合键刷新后，如图 1-9 所示。

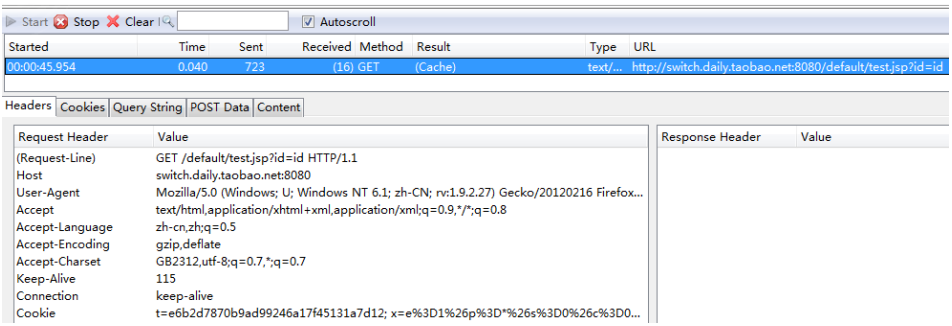

图 1-8　HTTP 的请求头返回缓冲数据

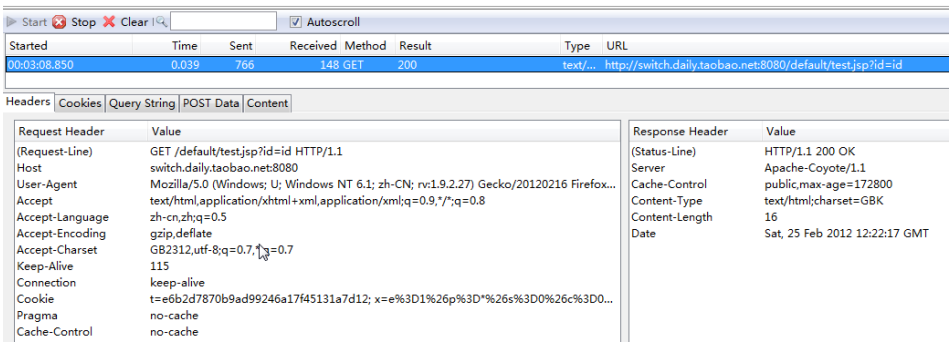

图 1-9　按 Ctrl+F5 组合键刷新页面时 HTTP 的请求头返回数据

这次请求时从服务端返回的数据，最重要的是在请求头中增加了两个请求项 Pragma:no-cache 和 Cache-Control:no-cache。为什么增加了这两项配置项，它们有什么作用？

1．Cache-Control/Pragma

这个 HTTP Head 字段用于指定所有缓存机制在整个请求/响应链中必须服从的指令，如果知道该页面是否为缓存，不仅可以控制浏览器，还可以控制和 HTTP 协议相关的缓存或代理服务器。HTTP Head 字段有一些可选值，这些值及其说明如表 1-4 所示。

表 1-4　HTTP Head 字段的可选值

可　选　值	说　　明
Public	所有内容都将被缓存，在响应头中设置
Private	内容只缓存到私有缓存中，在响应头中设置
no-cache	所有内容都不会被缓存，在请求头和响应头中设置
no-store	所有内容都不会被缓存到缓存或 Internet 临时文件中，在响应头中设置
must-revalidation/proxy-revalidation	如果缓存的内容失效，请求必须发送到服务器/代理以进行重新验证，在请求头中设置
max-age=xxx	缓存的内容将在 xxx 秒后失效，这个选项只在 HTTP 1.1 中可用，和 Last-Modified 一起使用时优先级较高，在响应头中设置

Cache-Control 请求字段被各个浏览器支持得较好，而且它的优先级也比较高，它和其他一些请求字段（如 Expires）同时出现时，Cache-Control 会覆盖其他字段。

Pragma 字段的作用和 Cache-Control 有点类似，它也是在 HTTP 头中包含一个特殊的指令，使相关的服务器来遵守，最常用的就是 Pragma:no-cache，它和 Cache-Control:no-cache 的作用是一样的。

2．Expires

Expires 通常的使用格式是 Expires: Sat, 25 Feb 2012 12:22:17 GMT，后面跟着一个日期和时间，超过这个时间值后，缓存的内容将失效，也就是浏览器在发出请求之前检查这个页面的这个字段，看该页面是否已经过期了，过期了就重新向服务器发起请求。

3．Last-Modified/Etag

Last-Modified 字段一般用于表示一个服务器上的资源的最后修改时间，资源可以是静态（静态内容自动加上 Last-Modified 字段）或者动态的内容（如 Servlet 提供了一个 getLastModified 方法用于检查某个动态内容是否已经更新），通过这个最后修改时间可以判断当前请求的资源是否是最新的。

一般服务端在响应头中返回一个 Last-Modified 字段，告诉浏览器这个页面的最后修改时间，如 Last-Modified: Sat, 25 Feb 2012 12:55:04 GMT，浏览器再次请求时在请求头中增加一个 If-Modified-Since: Sat, 25 Feb 2012 12:55:04 GMT 字段，询问当前缓存的页面是

否是最新的，如果是最新的就返回 304 状态码，告诉浏览器是最新的，服务器也不会传输新的数据。

与 Last-Modified 字段有类似功能的还有一个 Etag 字段，这个字段的作用是让服务端给每个页面分配一个唯一的编号，然后通过这个编号来区分当前这个页面是否是最新的。这种方式比使用 Last-Modified 更加灵活，但是在后端的 Web 服务器有多台时比较难处理，因为每个 Web 服务器都要记住网站的所有资源，否则浏览器返回这个编号就没有意义了。

1.4 DNS 域名解析

我们知道互联网都是通过 URL 来发布和请求资源的，而 URL 中的域名需要解析成 IP 地址才能与远程主机建立连接，如何将域名解析成 IP 地址就属于 DNS 解析的工作范畴。

可以毫不夸张地说，虽然我们平时上网感觉不到 DNS 解析的存在，但是一旦 DNS 解析出错，可能会导致非常严重的互联网灾难。目前世界上的整个互联网有几个 DNS 根域名服务器，任何一台根服务器坏掉后果都会非常严重。

1.4.1 DNS 域名解析过程

图 1-10 是 DNS 域名解析的主要请求过程实例图。

如图 1-10 所示，当一个用户在浏览器中输入 www.abc.com 时，DNS 解析将会有将近 10 个步骤，这个过程大体描述如下。

当用户在浏览器中输入域名并按下回车键后，第 1 步，浏览器会检查缓存中有没有这个域名对应的解析过的 IP 地址，如果缓存中有，这个解析过程就将结束。浏览器缓存域名也是有限制的，不仅浏览器缓存大小有限制，而且缓存的时间也有限制，通常情况下为几分钟到几小时不等，域名被缓存的时间限制可以通过 TTL 属性来设置。这个缓存时间太长和太短都不好，如果缓存时间太长，一旦域名被解析到的 IP 有变化，会导致被客户端缓存的域名无法解析到变化后的 IP 地址，以致该域名不能正常解析，这段时间内有可能会有一部分用户无法访问网站。如果时间设置太短，会导致用户每次访问网站都要重新解析一次域名。

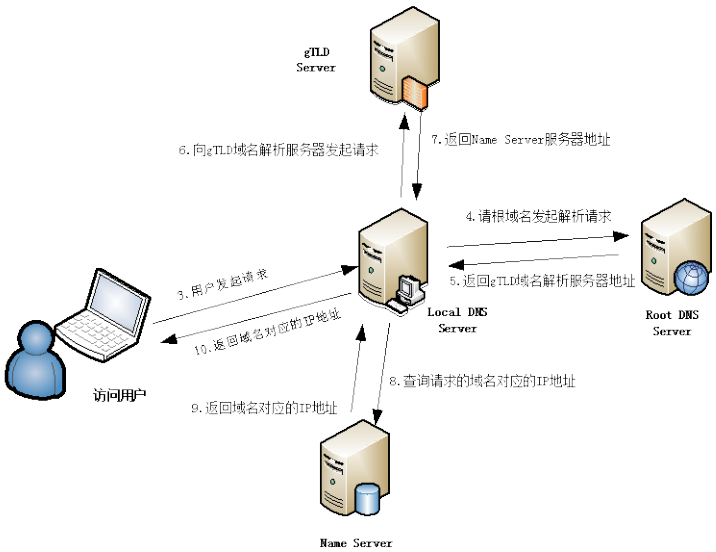

图 1-10　DNS 域名解析

第 2 步，如果用户的浏览器缓存中没有，浏览器会查找操作系统缓存中是否有这个域名对应的 DNS 解析结果。其实操作系统也会有一个域名解析的过程，在 Windows 中可以通过 C:\Windows\System32\drivers\etc\hosts 文件来设置，你可以将任何域名解析到任何能够访问的 IP 地址。如果你在这里指定了一个域名对应的 IP 地址，那么浏览器会首先使用这个 IP 地址。例如，我们在测试时可以将一个域名解析到一台测试服务器上，这样不用修改任何代码就能测试到单独服务器上的代码的业务逻辑是否正确。正是因为有这种本地 DNS 解析的规程，所以黑客就有可能通过修改你的域名解析来把特定的域名解析到它指定的 IP 地址上，导致这些域名被劫持。

这导致早期的 Windows 版本中出现过很严重的问题，而且对于一般没有太多电脑知识的用户来说，出现问题后很难发现，即使发现也很难自己解决，所以 Windows 7 中将 hosts 文件设置成了只读的，防止这个文件被轻易修改。

在 Linux 中这个配置文件是/etc/named.conf，修改这个文件可以达到同样的目的，当

解析到这个配置文件中的某个域名时，操作系统会在缓存中缓存这个解析结果，缓存的时间同样是受这个域名的失效时间和缓存的空间大小控制的。

前面这两个步骤都是在本机完成的，所以在图 1-10 中没有表示出来。到这里还没有涉及真正的域名解析服务器，如果在本机中仍然无法完成域名的解析，就会真正请求域名服务器来解析这个域名了。

第 3 步，如何、怎么知道域名服务器呢？在我们的网络配置中都会有“DNS 服务器地址”这一项，这个地址就用于解决前面所说的如果两个过程无法解析时要怎么办，操作系统会把这个域名发送给这里设置的 LDNS，也就是本地区的域名服务器。这个 DNS 通常都提供给你本地互联网接入的一个 DNS 解析服务，例如你是在学校接入互联网，那么你的 DNS 服务器肯定在你的学校，如果你是在一个小区接入互联网的，那这个 DNS 就是提供给你接入互联网的应用提供商，即电信或者联通，也就是通常所说的 SPA，那么这个 DNS 通常也会在你所在城市的某个角落，通常不会很远。在 Windows 下可以通过 ipconfig 查询这个地址，如图 1-11 所示。

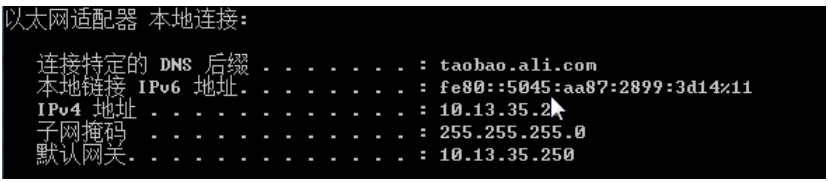

图 1-11　在 Windows 中查询 DNS Server

在 Linux 下可以通过如下方式查询配置的 DNS Server，如图 1-12 所示。

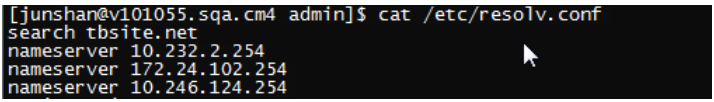

图 1-12　在 Linux 中下查询 DNS Server

这个专门的域名解析服务器性能都会很好，它们一般都会缓存域名解析结果，当然缓存时间是受域名的失效时间控制的，一般缓存空间不是影响域名失效的主要因素。大约 80%的域名解析都到这里就已经完成了，所以 LDNS 主要承担了域名的解析工作。

第 4 步，如果 LDNS 仍然没有命中，就直接到 Root Server 域名服务器请求解析。

第 5 步，根域名服务器返回给本地域名服务器一个所查询域的主域名服务器（gTLD Server）地址。gTLD 是国际顶级域名服务器，如.com、.cn、.org 等，全球只有 13 台左右。

第 6 步，本地域名服务器（Local DNS Server）再向上一步返回的 gTLD 服务器发送请求。

第 7 步，接受请求的 gTLD 服务器查找并返回此域名对应的 Name Server 域名服务器的地址，这个 Name Server 通常就是你注册的域名服务器，例如你在某个域名服务提供商申请的域名，那么这个域名解析任务就由这个域名提供商的服务器来完成。

第 8 步，Name Server 域名服务器会查询存储的域名和 IP 的映射关系表，正常情况下都根据域名得到目标 IP 记录，连同一个 TTL 值返回给 DNS Server 域名服务器。

第 9 步，返回该域名对应的 IP 和 TTL 值，Local DNS Server 会缓存这个域名和 IP 的对应关系，缓存的时间由 TTL 值控制。

第 10 步，把解析的结果返回给用户，用户根据 TTL 值缓存在本地系统缓存中，域名解析过程结束。

在实际的 DNS 解析过程中，可能还不止这 10 个步骤，如 Name Server 也可能有多级，或者有一个 GTM 来负载均衡控制，这都有可能会影响域名解析的过程。

1.4.2　跟踪域名解析过程

在 Linux 和 Windows 下都可以用 nslookup 命令来查询域名的解析结果，如图 1-13 所示。

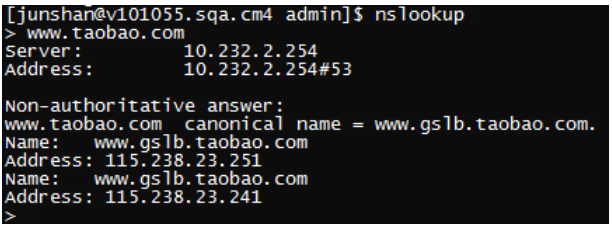

图 1-13　用 nslookup 查询域名解析结果

在 Linux 系统中还可以使用 dig 命名来查询 DNS 的解析过程，如下所示：

```
[junshan@v101055.sqa.cm4 admin]$ dig www.taobao.com

; <<>> DiG 9.3.6-P1-RedHat-9.3.6-4.P1.el5 <<>> www.taobao.com
;; global options:  printcmd
;; Got answer:
```

```
    ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16903
    ;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 3, ADDITIONAL: 3

    ;; QUESTION SECTION:
    ;www.taobao.com.                    IN     A

    ;; ANSWER SECTION:
    www.taobao.com.        1542    IN     CNAME   www.gslb.taobao.com.
    www.gslb.taobao.com.    130     IN     A       115.238.23.251
    www.gslb.taobao.com.    130     IN     A       115.238.23.241

    ;; AUTHORITY SECTION:
    gslb.taobao.com.       70371   IN     NS      gslbns3.taobao.com.
    gslb.taobao.com.       70371   IN     NS      gslbns1.taobao.com.
    gslb.taobao.com.       70371   IN     NS      gslbns2.taobao.com.

    ;; ADDITIONAL SECTION:
    gslbns1.taobao.com.     452     IN     A       121.0.23.218
    gslbns2.taobao.com.     452     IN     A       115.124.17.70
    gslbns3.taobao.com.     452     IN     A       110.75.3.193

    ;; Query time: 5 msec
    ;; SERVER: 10.232.2.254#53(10.232.2.254)
    ;; WHEN: Sun Feb 12 19:19:05 2012
    ;; MSG SIZE  rcvd: 201
```

结果的第 1 行输出了当前 Linux 的版本号，第 2 行说明可以增加可选参数 printcmd，如果加上 printcmd，打印出来的结果如下：

```
    ;; Got answer:
    ;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 58602
    ;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

    ;; QUESTION SECTION:
    ;printcmd.                  IN     A

    ;; AUTHORITY SECTION:
    .                       10800   IN      SOA     a.root-servers.net. nstld.
verisign-grs.com. 2012021200 1800 900 604800 86400

    ;; Query time: 208 msec
```

```
;; SERVER: 10.232.2.254#53(10.232.2.254)
;; WHEN: Sun Feb 12 19:20:59 2012
;; MSG SIZE  rcvd: 101
```

“QUESTION SECTION”部分表示当前查询的域名是一个 A 记录，“ANSWER SECTION”部分返回了这个域名由 CNAME 到 www.gslb.taobao.com，返回了这个域名对应的 IP 地址。

还可通过增加+trace 参数跟踪这个域名的解析过程，如下所示：

```
[junshan@v101055.sqa.cm4 admin]$ dig www.taobao.com +trace

; <<>> DiG 9.3.6-P1-RedHat-9.3.6-4.P1.el5 <<>> www.taobao.com +trace
;; global options:  printcmd
.                   449398  IN      NS      k.root-servers.net.
.                   449398  IN      NS      l.root-servers.net.
.                   449398  IN      NS      m.root-servers.net.
.                   449398  IN      NS      a.root-servers.net.
.                   449398  IN      NS      b.root-servers.net.
.                   449398  IN      NS      c.root-servers.net.
.                   449398  IN      NS      d.root-servers.net.
.                   449398  IN      NS      e.root-servers.net.
.                   449398  IN      NS      f.root-servers.net.
.                   449398  IN      NS      g.root-servers.net.
.                   449398  IN      NS      h.root-servers.net.
.                   449398  IN      NS      i.root-servers.net.
.                   449398  IN      NS      j.root-servers.net.
;; Received 272 bytes from 10.232.2.254#53(10.232.2.254) in 0 ms

com.                172800  IN      NS      a.gtld-servers.net.
com.                172800  IN      NS      b.gtld-servers.net.
com.                172800  IN      NS      c.gtld-servers.net.
com.                172800  IN      NS      d.gtld-servers.net.
com.                172800  IN      NS      e.gtld-servers.net.
com.                172800  IN      NS      f.gtld-servers.net.
com.                172800  IN      NS      g.gtld-servers.net.
com.                172800  IN      NS      h.gtld-servers.net.
com.                172800  IN      NS      i.gtld-servers.net.
com.                172800  IN      NS      j.gtld-servers.net.
com.                172800  IN      NS      k.gtld-servers.net.
com.                172800  IN      NS      l.gtld-servers.net.
```

```
com.                    172800  IN      NS      m.gtld-servers.net.
;; Received 492 bytes from 193.0.14.129#53(k.root-servers.net) in 607 ms

taobao.com.             172800  IN      NS      ns1.taobao.com.
taobao.com.             172800  IN      NS      ns2.taobao.com.
taobao.com.             172800  IN      NS      ns3.taobao.com.
;; Received 134 bytes from 192.5.6.30#53(a.gtld-servers.net) in 250 ms

www.taobao.com.         1800    IN      CNAME   www.gslb.taobao.com.
gslb.taobao.com.        86400   IN      NS      gslbns2.taobao.com.
gslb.taobao.com.        86400   IN      NS      gslbns3.taobao.com.
gslb.taobao.com.        86400   IN      NS      gslbns1.taobao.com.
;; Received 169 bytes from 110.75.1.19#53(ns1.taobao.com) in 0 ms
```

上面清楚地显示了整个域名是如何发起和解析的，从根域名(.)到 gTLD Server(.com.)再到 Name Server（taobao.com.）的整个过程都显示出来了。还可以看出 DNS 的服务器有多个备份，可以从任何一台查询到解析结果。

1.4.3 清除缓存的域名

我们知道 DNS 域名解析后会缓存解析结果，其中主要在两个地方缓存结果，一个是 Local DNS Server，另外一个是用户的本地机器。这两个缓存都是 TTL 值和本机缓存大小控制的，但是最大缓存时间是 TTL 值，基本上 Local DNS Server 的缓存时间就是 TTL 控制的，很难人工介入，但是我们的本机缓存可以通过如下方式清除。

在 Windows 下可以在命令行模式下执行 ipconfig /flushdns 命令来刷新缓存，如图 1-14 所示。

图 1-14 在 Windows 下刷新 DNS 缓存

在 Linux 下可以通过/etc/init.d/nscd restart 来清除缓存，如图 1-15 所示。

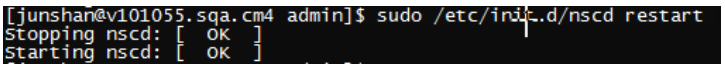

图 1-15 在 Linux 下清除 DNS 缓存

重启依然是解决很多问题的第一选择。

在 Java 应用中 JVM 也会缓存 DNS 的解析结果，这个缓存是在 InetAddress 类中完成的，而且这个缓存时间还比较特殊，它有两种缓存策略：一种是正确解析结果缓存，另一种是失败的解析结果缓存。这两个缓存时间由两个配置项控制，配置项是在 %JAVA_HOME%\lib\security\java.security 文件中配置的。两个配置项分别是 networkaddress.cache. ttl 和 networkaddress.cache.negative.ttl，它们的默认值分别是-1（永不失效）和 10（缓存 10 秒）。

要修改这两个值同样有几种方式，分别是：直接修改 java.security 文件中的默认值、在 Java 的启动参数中增加-Dsun.net.inetaddr.ttl=xxx 来修改默认值、通过 InetAddress 类动态修改。

在这里还要特别强调一下，如果我们需要用 InetAddress 类解析域名时，一定要是单例模式，不然会有严重的性能问题，如果每次都创建 InetAddress 实例，每次都要进行一次完整的域名解析，非常耗时，这点要特别注意。

1.4.4　几种域名解析方式

域名解析记录主要分为 A 记录、MX 记录、CNAME 记录、NS 记录和 TXT 记录。

- A 记录，A 代表的是 Address，用来指定域名对应的 IP 地址，如将 item.taobao.com 指定到 115.238.23.241，将 switch.taobao.com 指定到 121.14.24.241。A 记录可以将多个域名解析到一个 IP 地址，但是不能将一个域名解析到多个 IP 地址。
- MX 记录，表示的是 Mail Exchange，就是可以将某个域名下的邮件服务器指向自己的 Mail Server，如 taobao.com 域名的 A 记录 IP 地址是 115.238.25.245，如果 MX 记录设置为 115.238.25.246，是 xxx@taobao.com 的邮件路由，DNS 会将邮件发送到 115.238.25.246 所在的服务器，而正常通过 Web 请求的话仍然解析到 A 记录的 IP 地址。
- CNAME 记录，全称是 Canonical Name（别名解析）。所谓的别名解析就是可以为一个域名设置一个或者多个别名。如将 taobao.com 解析到 xulingbo.net，将 srcfan.com 也解析到 xulingbo.net。其中 xulingbo.net 分别是 taobao.com 和 srcfan.com 的别名。前面的跟踪域名解析中的“www.taobao.com. 1542 IN CNAME www.gslb.taobao.com”就是 CNAME 解析。

- NS 记录，为某个域名指定 DNS 解析服务器，也就是这个域名有指定的 IP 地址的 DNS 服务器去解析，前面的“gslb.taobao.com. 86400 IN NS gslbns2.taobao.com.”就是 NS 解析。
- TXT 记录，为某个主机名或域名设置说明，如可以为 xulingbo.net 设置 TXT 记录为“君山的博客|许令波”这样的说明。

1.5 CDN 工作机制

CDN 也就是内容分布网络（Content Delivery Network），它是构筑在现有 Internet 上的一种先进的流量分配网络。其目的是通过在现有的 Internet 中增加一层新的网络架构，将网站的内容发布到最接近用户的网络“边缘”，使用户可以就近取得所需的内容，提高用户访问网站的响应速度。有别于镜像，它比镜像更智能，可以做这样一个比喻：CDN = 镜像（Mirror）+ 缓存（Cache）+ 整体负载均衡（GSLB）。因而，CDN 可以明显提高 Internet 中信息流动的效率。

目前 CDN 都以缓存网站中的静态数据为主，如 CSS、JS、图片和静态页面等数据。用户在从主站服务器请求到动态内容后再从 CDN 上下载这些静态数据，从而加速网页数据内容的下载速度，如淘宝有 90%以上的数据都是由 CDN 来提供的。

通常来说 CDN 要达到以下几个目标。

- 可扩展（Scalability）。性能可扩展性：应对新增的大量数据、用户和事务的扩展能力；成本可扩展性：用低廉的运营成本提供动态的服务能力和高质量的内容分发。
- 安全性（Security）。强调提供物理设备、网络、软件、数据和服务过程的安全性，（趋势）减少因为 DDoS 攻击或者其他恶意行为造成商业网站的业务中断。
- 可靠性、响应和执行（Reliability、Responsiveness 和 Performance）。服务可用性，能够处理可能的故障和用户体验的下降，通过负载均衡及时提供网络的容错机制。

1.5.1 CDN 架构

通常的 CDN 架构如图 1-16 所示。

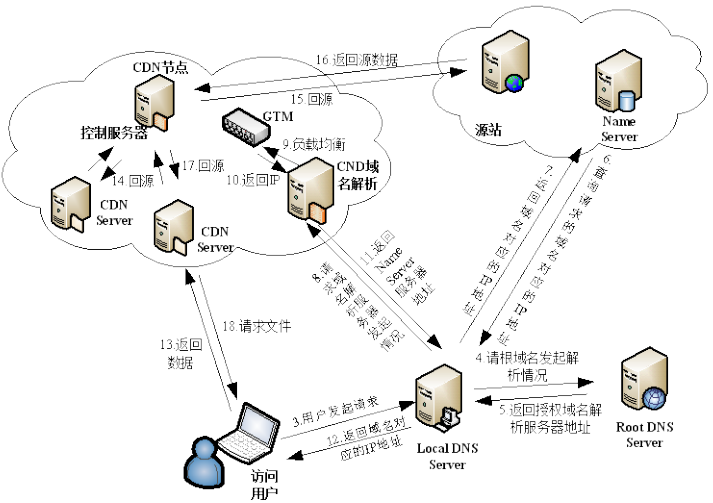

图 1-16　CDN 架构

如图 1-16 所示，一个用户访问某个静态文件（如 CSS 文件），这个静态文件的域名假如是 cdn.taobao.com，那么首先要向 Local DNS 服务器发起请求，一般经过迭代解析后回到这个域名的注册服务器去解析，一般每个公司都会有一个 DNS 解析服务器。这时这个 DNS 解析服务器通常会把它重新 CNAME 解析到另外一个域名，而这个域名最终会被指向 CDN 全局中的 DNS 负载均衡服务器，再由这个 GTM 来最终分配是哪个地方的访问用户，返回给离这个访问用户最近的 CDN 节点。

拿到 DNS 解析结果，用户就直接去这个 CDN 节点访问这个静态文件了，如果这个节点中所请求的文件不存在，就会再回到源站去获取这个文件，然后再返回给用户。

1.5.2　负载均衡

负载均衡（Load Balance）就是对工作任务进行平衡、分摊到多个操作单元上执行，如图片服务器、应用服务器等，共同完成工作任务。它可以提高服务器响应速度及利用效

率，避免软件或者硬件模块出现单点失效，解决网络拥塞问题，实现地理位置无关性，为用户提供较一致的访问质量。

通常有三种负载均衡架构，分别是链路负载均衡、集群负载均衡和操作系统负载均衡。所谓链路负载均衡也就是前面提到的通过 DNS 解析成不同的 IP，然后用户根据这个 IP 来访问不同的目标服务器，如图 1-17 所示。

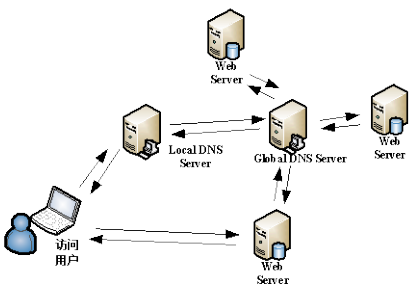

图 1-17　链路负载均衡

负载均衡是由 DNS 的解析来完成的，用户最终访问哪个 Web Server 是由 DNS Server 来控制的，在这里就是由 Global DNS Server 来动态解析域名服务。这种 DNS 解析的优点是用户会直接访问目标服务器，而不需要经过其他的代理服务器，通常访问速度会更快。但是也有缺点，由于 DNS 在用户本地和 Local DNS Server 都有缓存，一旦某台 Web Server 挂掉，那么很难及时更新用户的域名解析结构。如果用户的域名没有及时更新，那么用户将无法访问这个域名，带来的后果非常严重。

集群负载均衡是另外一种常见的负载均衡方式，它一般分为硬件负载均衡和软件负载均衡。硬件负载均衡一般使用一台专门硬件设备来转发请求，如图 1-18 所示。

硬件负载均衡的关键就是这台价格非常昂贵的设备，如 F5，通常为了安全需要一主一备。它的优点很显然就是性能非常好，缺点就是非常贵，一般公司是用不起的，还有就是当访问量陡然增大超出服务极限时，不能进行动态扩容。

软件负载均衡是使用最普遍的一种负载方式，它的特点是使用成本非常低，直接使用廉价的 PC 就可以搭建。缺点就是一般一次访问请求要经过多次代理服务器，会增加网络

延时。它的架构通常如图 1-19 所示。

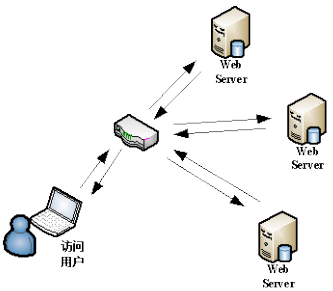

图 1-18　硬件负载均衡

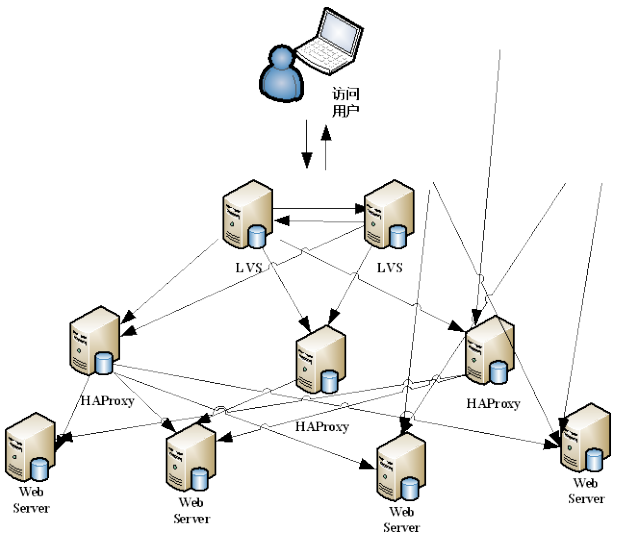

图 1-19　软件负载均衡

图 1-19 中上面两台是 LVS，使用四层负载均衡，也就是在网络层利用 IP 地址进行地址转发。下面三台使用 HAProxy 进行七层负载，也就是可以根据访问用户的 HTTP 请求头来进行负载均衡，如可以根据不同的 URL 来将请求转发到特定机器或者根据用户的 Cookie 信息来指定访问的机器。

最后一种是操作系统负载均衡，就是利用操作系统级别的软中断或者硬件中断来达到负载均衡，如可以设置多队列网卡等来实现。

这几种负载均衡不仅在 CDN 的集群中能使用，而且在 Web 服务或者分布式数据集群中同样也能使用，但是在这些地方后两种使用得要多一点。

1.6 总结

本章主要介绍了前端的一些基本知识，包括在用户端发起一个请求时，这个请求都经过了哪些服务单元，进行了哪些处理。本章可以帮助我们对 B/S 网络架构有个整体的认识。