第二部分

HTTP结构

第二部分的6章主要介绍了HTTP服务器、代理、缓存、网关和机器人应用程序，这些都是Web系统架构的构造模块。

- 第5章概述了Web服务器结构。
- 第6章详细介绍了HTTP代理服务器，它们是连接HTTP客户端的中间服务器，是HTTP服务和控制的平台。
- 第7章深入研究了Web的缓存机制。缓存是通过对常用文档进行本地复制来提高性能、减少流量的设备。
- 第8章介绍了一些应用程序，通过这些程序，HTTP就可以与使用不同协议（比如SSL加密协议）的软件进行互操作了。
- 第9章介绍了Web客户端，结束了HTTP架构之旅。
- 第10章涵盖了HTTP未来发展的一些主题，特别介绍了HTTP-NG技术。

107 ~ 108

第5章

Web服务器

Web 服务器每天会分发出数十亿的 Web 页面。这些页面可以告诉你天气情况，装载在线商店的购物车，还能帮你找到许久未联系的高中同学。Web 服务器是万维网的骨干。本章将介绍以下话题。

- 对多种使用不同类型软硬件的 Web 服务器进行调查。
- 介绍如何用 Perl 编写简单的诊断性 Web 服务器。
- 一步一步地解释 Web 服务器是如何处理 HTTP 事务的。

为了对问题进行具体的说明，例子中使用了 Apache Web 服务器及其配置选项。

5.1 各种形状和尺寸的Web服务器

Web 服务器会对 HTTP 请求进行处理并提供响应。术语“Web 服务器”可以用来表示 Web 服务器的软件，也可以用来表示提供 Web 页面的特定设备或计算机。

Web 服务器有着不同的风格、形状和尺寸。有普通的 10 行 Perl 脚本的 Web 服务器、50MB 的安全商用引擎以及极小的卡上服务器。但不管功能有何差异，所有的 Web 服务器都能够接收请求资源的 HTTP 请求，将内容回送给客户端（参见图 1-5）。

5.1.1 Web服务器的实现

Web 服务器实现了 HTTP 和相关的 TCP 连接处理。负责管理 Web 服务器提供的资源，以及对 Web 服务器的配置、控制及扩展方面的管理。

Web 服务器逻辑实现了 HTTP 协议、管理着 Web 资源，并负责提供 Web 服务器的
109 管理功能。Web 服务器逻辑和操作系统共同负责管理 TCP 连接。底层操作系统负责管理底层计算机系统的硬件细节，并提供了 TCP/IP 网络支持、负责装载 Web 资源的文件系统以及控制当前计算活动的进程管理功能。

Web 服务器有各种不同的形式。

- 可以在标准的计算机系统上安装并运行通用的软件 Web 服务器。
- 如果不想那么麻烦地去安装软件，可以买一台 Web 服务器设备，通常会是一台安装在时髦机架上的计算机，里面的软件会预装并配置好。
- 随着微处理器奇迹般地出现，有些公司甚至可以在少量计算机芯片上实现嵌入式 Web 服务器，使其成为完美的（便携式）消费类设备管理控制台。

我们分别来看看这些实现方式。

5.1.2 通用软件Web服务器

通用软件 Web 服务器都运行在标准的、有网络功能的计算机系统上。可以选择开源软件（比如 Apache 或 W3C 的 Jigsaw）或者商业软件（比如微软和 iPlanet 的 Web 服务器）。基本上所有的计算机和操作系统中都有可用的 Web 服务器软件。

尽管不同类型的 Web 服务器程序有数万个（包括定制的和特殊用途的 Web 服务器），但大多数 Web 服务器软件都来自少数几个组织。

2002 年 2 月，Netcraft 调查（http://www.netcraft.com/survey/）显示有三家厂商主宰了公共因特网 Web 服务器市场（参见图 5-1）。

- 免费的 Apache 软件占据了所有因特网 Web 服务器中大约 60% 的市场。
- 微软的 Web 服务器占据了另外 30%。
- Sun 的 iPlanet 占据了另外 3%。

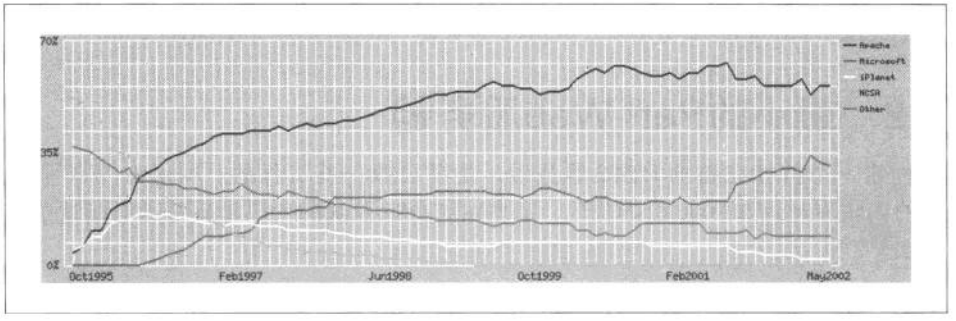


图 5-1 Netcraft 的自动化调查估计的 Web 服务器市场份额

110

但这些数据也不能尽信，通常大家都认为 Netcraft 调查会夸大 Apache 软件的优势。首先，在调查计算服务器的时候没有考虑其流行程度。各大 ISP 的代理服务器访问研究表明，Apache 服务器提供的页面数量远小于 60%，但仍然超过了微软和 Sun 的 iPlanet。然而，据说微软和 iPlanet 服务器在公司企业中要比 Apache 更受欢迎。

5.1.3 Web服务器设备

Web 服务器设备（Web server appliance）是预先打包好的软硬件解决方案。厂商会在他们选择的计算机平台上预先安装好软件服务器，并将软件配置好。下面是一些 Web 服务器设备的例子：

- Sun/Cobalt RaQ Web 设备（http://www.cobalt.com）；
- 东芝的 Magnia SG10（http://www.toshiba.com）；

- IBM 的 Whistle Web 服务器设备（http://www.whistle.com）。

应用解决方案不再需要安装及配置软件，通常可以极大地简化管理工作。但是，Web 服务器通常不太灵活，特性不太丰富，而且服务器硬件也不太容易重用或升级。

5.1.4 嵌入式Web服务器

嵌入式服务器（embeded server）是要嵌入到消费类产品（比如打印机或家用设备）中去的小型 Web 服务器。嵌入式 Web 服务器允许用户通过便捷的 Web 浏览器接口来管理其消费者设备。

有些嵌入式 Web 服务器甚至可以在小于一平方英寸的空间内实现，但通常只能提供最小特性功能集。下面是两种非常小的嵌入式 Web 服务器实例：

- IPic 火柴头大小的 Web 服务器（http://www-ccs.cs.umass.edu/~shri/iPic.html）；
- NetMedia SitePlayer SP1 以太网 Web 服务器（http://www.siteplayer.com）。

5.2 最小的Perl Web服务器

要构建一个特性完备的 HTTP 服务器，是需要做一些工作的。Apache Web 服务器的内核有超过 50 000 行的代码，那些可选处理模块的代码量更是远远超过这个数字。

这个软件所要做的就是支持 HTTP/1.1 的各种特性：丰富的资源支持、虚拟主机、访问控制、日志记录、配置、监视和性能特性。在这里，可以用少于 30 行的 Perl
111 代码来创建一个最小的可用 HTTP 服务器。我们来看看这是怎么实现的。

例 5-1 显示了一个名为 type-o-serve 的小型 Perl 程序。这个程序是个很有用的诊断工具，可以用来测试与客户端和代理的交互情况。与所有 Web 服务器一样，type-o-serve 会等待 HTTP 连接。只要 type-o-serve 收到了请求报文，就会将报文打印在屏幕上，然后等待用户输入（或粘贴）一条响应报文，并将其回送给客户端。通过这种方式，type-o-serve 假扮成一台 Web 服务器，记录下确切的 HTTP 请求报文，并允许用户回送任意的 HTTP 响应报文。

这个简单的 type-o-serve 实用程序并没有实现大部分的 HTTP 功能，但它是一种很有用的工具，产生服务器响应报文的方式与 Telnet 产生客户端请求报文的方式相同（参见例 5-1）。可以从 http://www.http-guide.com/tools/type-o-serve.pl 上下载 type-o-serve 程序。

例 5-1 type-o-serve——用于 HTTP 调试的最小型 Perl Web 服务器

```
#!/usr/bin/perl

use Socket;
use Carp;
use FileHandle;

# (1) use port 8080 by default, unless overridden on command line
$port = (@ARGV ? $ARGV[0] : 8080);

# (2) create local TCP socket and set it to listen for connections
$proto = getprotobyname('tcp');
socket(S, PF_INET, SOCK_STREAM, $proto) || die;
setsockopt(S, SOL_SOCKET, SO_REUSEADDR, pack("l", 1)) || die;
bind(S, sockaddr_in($port, INADDR_ANY)) || die;
listen(S, SOMAXCONN) || die;

# (3) print a startup message
printf("   <<<Type-O-Serve Accepting on Port %d>>>\n\n",$port);

while (1)
{
    # (4) wait for a connection C
    $cport_caddr = accept(C, S);
    ($cport,$caddr) = sockaddr_in($cport_caddr);
    C->autoflush(1);

    # (5) print who the connection is from
    $cname = gethostbyaddr($caddr,AF_INET);
    printf("   <<<Request From '%s'>>>\n",$cname);

    # (6) read request msg until blank line, and print on screen
    while ($line = <C>)
    {
        print $line;
        if ($line =~ /^\r/) { last; }
    }

    # (7) prompt for response message, and input response lines,
    #     sending response lines to client, until solitary "."
    printf("   <<<Type Response Followed by '.'>>>\n");

    while ($line = <STDIN>)
    {
        $line =~ s/\r//;
        $line =~ s/\n//;
        if ($line =~ /^\./) { last; }
        print C $line . "\r\n";
    }
    close(C);
}
```

112

图 5-2 显示了 Joe 的五金商店的管理员是如何用 type-o-serve 来测试 HTTP 通信的。

- 首先，管理员启动了 type-o-serve 诊断服务器，在一个特定的端口上监听。由于 Joe 的五金商店已经有一个产品化的 Web 服务器在监听 80 端口了，所以管理员用下面这条命令在端口 8080(可以选择任意未用端口)上启动了 type-o-serve 服务：

```
% type-o-serve.pl 8080
```

- 只要 type-o-serve 开始运行了，就可以将浏览器指向这个 Web 服务器。在图 5-2 中，浏览器指向了 http://www.joes-hardware.com:8080/foo/bar/blah.txt。
- type-o-serve 程序收到来自浏览器的 HTTP 请求报文，并将 HTTP 请求报文的内容打印在屏幕上。然后 type-o-serve 诊断工具会等待用户输入一条简单的响应报文，后面跟着只有一个句号的空行。
- type-o-serve 将 HTTP 响应报文回送给浏览器，浏览器会显示响应报文的主体。

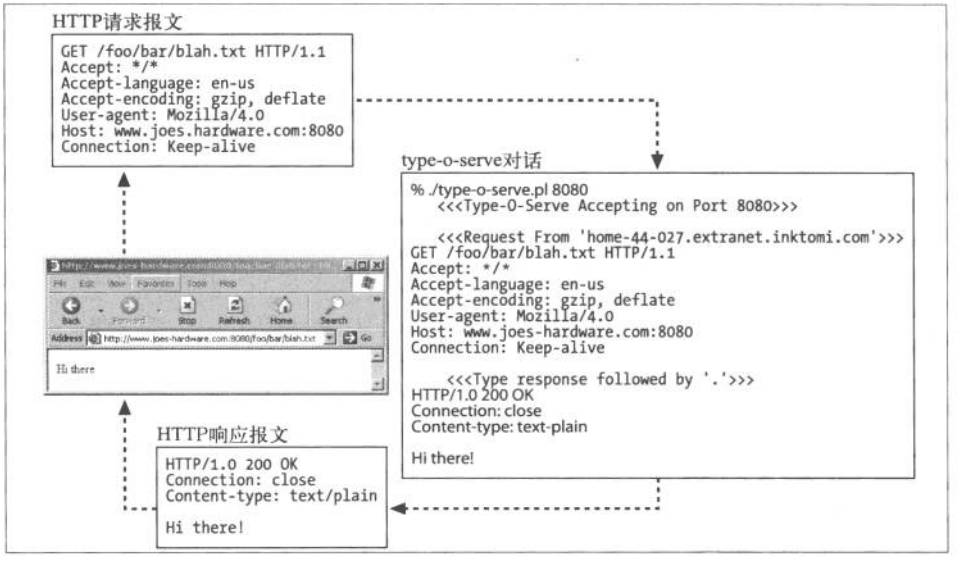


图 5-2 type-o-serve 实用程序让用户输入服务器响应，将其回送给客户端

5.3 实际的Web服务器会做些什么

例 5-1 显示的 Perl 服务器是一个 Web 服务器的小例子。最先进的商用 Web 服务器要比它复杂得多，但它们确实执行了几项同样的任务，如图 5-3 所示。

(1) 建立连接——接受一个客户端连接，或者如果不希望与这个客户端建立连接，就将其关闭。

(2) 接收请求——从网络中读取一条 HTTP 请求报文。
(3) 处理请求——对请求报文进行解释，并采取行动。
(4) 访问资源——访问报文中指定的资源。
(5) 构建响应——创建带有正确首部的 HTTP 响应报文。
(6) 发送响应——将响应回送给客户端。
(7) 记录事务处理过程——将与已完成事务有关的内容记录在一个日志文件中。 113

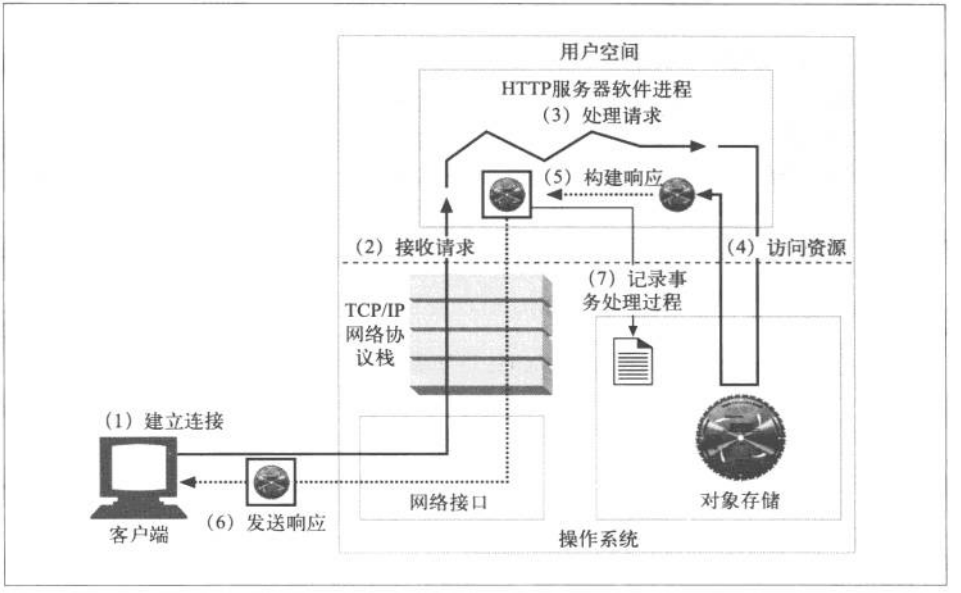


图 5-3 基本 Web 服务器请求的步骤 114

接下来的 7 个小节重点说明了 Web 服务器是怎样实现这些基本任务的。

5.4 第一步——接受客户端连接

如果客户端已经打开了一条到服务器的持久连接，可以使用那条连接来发送它的请求。否则，客户端需要打开一条新的到服务器的连接（回顾第 4 章，复习一下 HTTP 的连接管理技术）。

5.4.1 处理新连接

客户端请求一条到 Web 服务器的 TCP 连接时，Web 服务器会建立连接，判断连接的另一端是哪个客户端，从 TCP 连接中将 IP 地址解析出来。[1] 一旦新连接建立起来

并被接受，服务器就会将新连接添加到其现存 Web 服务器连接列表中，做好监视连接上数据传输的准备。

Web 服务器可以随意拒绝或立即关闭任意一条连接。有些 Web 服务器会因为客户端 IP 地址或主机名是未认证的，或者因为它是已知的恶意客户端而关闭连接。Web 服务器也可以使用其他识别技术。

5.4.2 客户端主机名识别

可以用“反向 DNS”对大部分 Web 服务器进行配置，以便将客户端 IP 地址转换成客户端主机名。Web 服务器可以将客户端主机名用于详细的访问控制和日志记录。但要注意的是，主机名查找可能会花费很长时间，这样会降低 Web 事务处理的速度。很多大容量 Web 服务器要么会禁止主机名解析，要么只允许对特定内容进行解析。

可以用配置指令 HostnameLookups 启用 Apache 的主机查找功能。比如，例 5-2 中的 Apache 配置指令就只打开了 HTML 和 CGI 资源的主机名解析功能。

例 5-2 配置 Apache，为 HTML 和 CGI 资源查找主机名

```
HostnameLookups off
<Files ~ "\.(html|htm|cgi)$">
    HostnameLookups on
</Files>
```

5.4.3 通过ident确定客户端用户

有些 Web 服务器还支持 IETF 的 ident 协议。服务器可以通过 ident 协议找到发起
115 HTTP 连接的用户名。这些信息对 Web 服务器的日志记录特别有用——流行的通用日志格式（Common Log Format）的第二个字段中就包含了每条 HTTP 请求的 ident 用户名。[2]

如果客户端支持 ident 协议，就在 TCP 端口 113 上监听 ident 请求。图 5-4 说明了 ident 协议是如何工作的。在图 5-4a 中，客户端打开了一条 HTTP 连接。然后，服务器打开自己到客户端 ident 服务器端口（113）的连接，发送一条简单的请求，询问与（由客户端和服务器端口号指定的）新连接相对应的用户名，并从客户端解析出包含用户名的响应。

注 1：不同的操作系统在对 TCP 连接进行操作时会使用不同的接口和数据结构。在 Unix 环境下，TCP 连接是由一个套接字表示的，可以用 getpeername 调用从套接字中获取客户端的 IP 地址。

注 2：这个通用日志格式的 ident 字段被称为“rfc931”，这是根据定义 ident 协议的过时 RFC 版本（更新过的 ident 规范记录在 RFC1413 中）命名的。

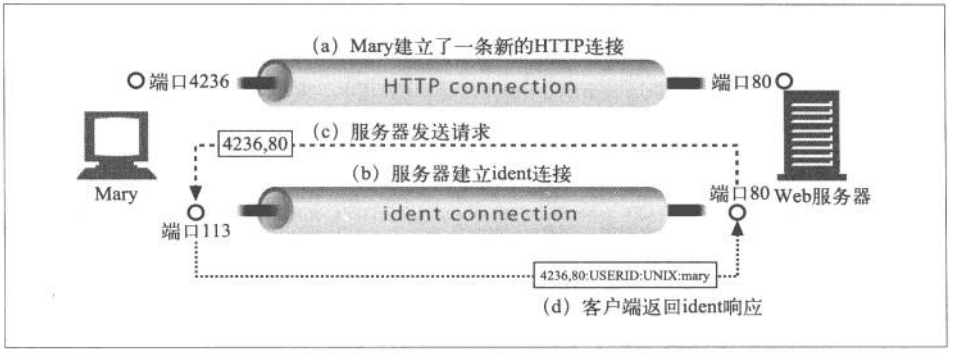


图 5-4　使用 ident 协议来确定 HTTP 的客户端用户名

ident 在组织内部可以很好地工作，但出于多种原因，在公共因特网上并不能很好地工作，原因包括：

- 很多客户端 PC 没有运行 ident 识别协议守护进程软件；
- ident 协议会使 HTTP 事务处理产生严重的时延；
- 很多防火墙不允许 ident 流量进入；
- ident 协议不安全，容易被伪造；
- ident 协议也不支持虚拟 IP 地址；
- 暴露客户端的用户名还涉及隐私问题。

可以通过 Apache 的 IdentityCheck on 指令告知 Apache Web 服务器使用 ident 查找功能。如果没有 ident 信息可用，Apache 会用连字符（-）来填充 ident 日志字段。由于没有 ident 信息可用，在使用通用日志格式的日志文件中，第二个字段通常都是连字符。

5.5　第二步——接收请求报文

连接上有数据到达时，Web 服务器会从网络连接中读取数据，并将请求报文中的内容解析出来（参见图 5-5）。

116

解析请求报文时，Web 服务器会：

- 解析请求行，查找请求方法、指定的资源标识符（URI）以及版本号，[3] 各项之间由一个空格分隔，并以一个回车换行（CRLF）序列作为行的结束；[4]

注 3：HTTP 的初始版本 HTTP/0.9 并不支持版本号。有些 Web 服务器也支持没有版本号的情况，会将报文作为 HTTP/0.9 请求进行解析。

注 4：很多客户端会错误地将 LF 作为行结束的终止符发送，所以很多 Web 服务器都支持将 LF 或 CRLF 作为行结束序列使用。

- 读取以 CRLF 结尾的报文首部；
- 检测到以 CRLF 结尾的、标识首部结束的空行（如果有的话）；
- 如果有的话（长度由 Content-Length 首部指定），读取请求主体。

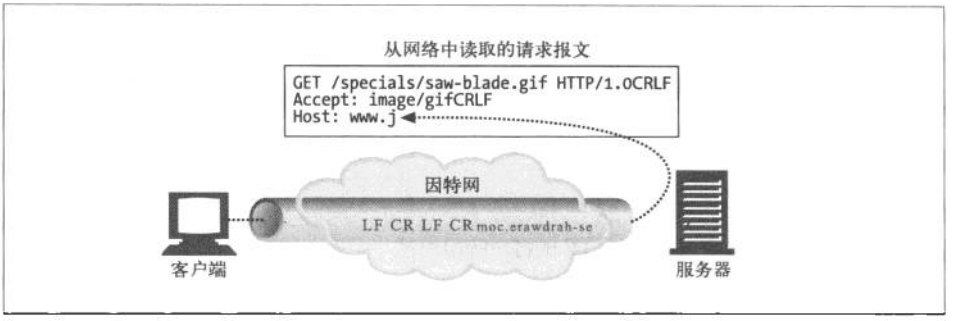


图 5-5　从连接中读取请求报文

解析请求报文时，Web 服务器会不定期地从网络上接收输入数据。网络连接可能随时都会出现延迟。Web 服务器需要从网络中读取数据，将部分报文数据临时存储在内存中，直到收到足以进行解析的数据并理解其意义为止。

5.5.1　报文的内部表示法

有些 Web 服务器还会用便于进行报文操作的内部数据结构来存储请求报文。比如，数据结构中可能包含有指向请求报文中各个片段的指针及其长度，这样就可以将这些首部存放在一个快速查询表中，以便快速访问特定首部的具体值了（参见图 5-6）。

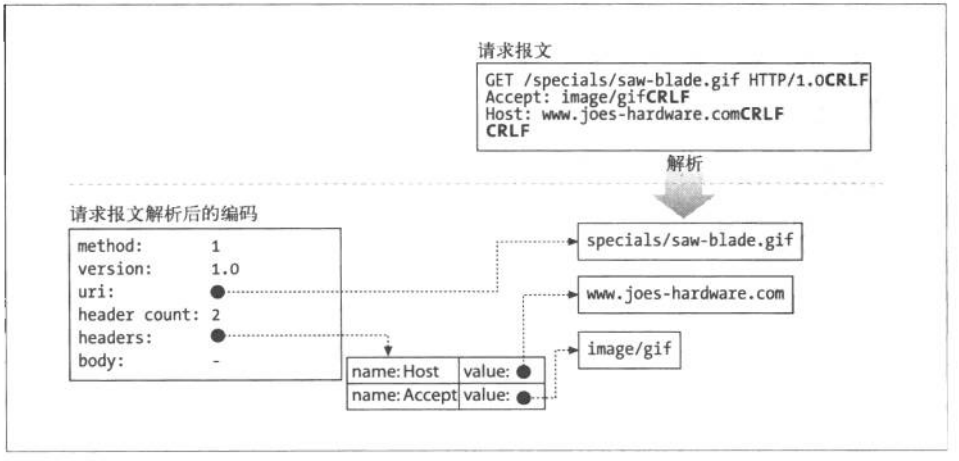


图 5-6　将请求报文解析为便捷的内部表示形式

5.5.2 连接的输入/输出处理结构

高性能的 Web 服务器能够同时支持数千条连接。这些连接使得服务器可以与世界各地的客户端进行通信，每个客户端都向服务器打开了一条或多条连接。某些连接可能在快速地向 Web 服务器发送请求，而其他一些连接则可能在慢慢发送，或者不经常发送请求，还有一些可能是空闲的，安静地等待着将来可能出现的动作。 117

因为请求可能会在任意时刻到达，所以 Web 服务器会不停地观察有无新的 Web 请求。不同的 Web 服务器结构会以不同的方式为请求服务，如图 5-7 所示。

- *单线程 Web 服务器*（参见图 5-7a）
  单线程的 Web 服务器一次只处理一个请求，直到其完成为止。一个事务处理结束之后，才去处理下一条连接。这种结构易于实现，但在处理过程中，所有其他连接都会被忽略。这样会造成严重的性能问题，只适用于低负荷的服务器，以及 type-o-serve 这样的诊断工具。

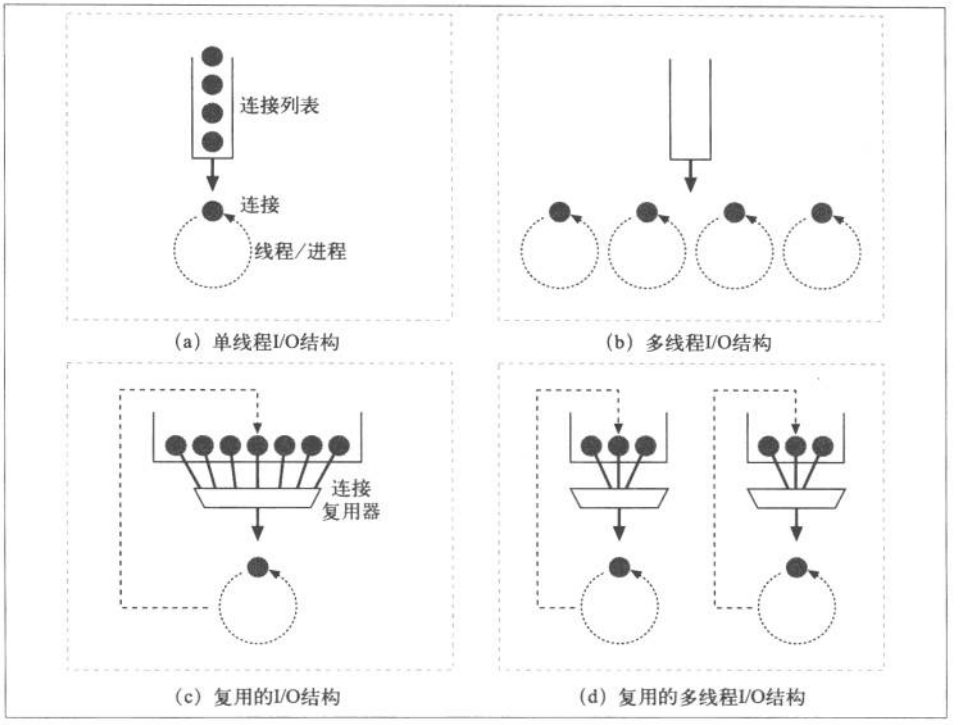
- *多进程及多线程 Web 服务器*（参见图 5-7b）
  多进程和多线程 Web 服务器用多个进程，或更高效的线程同时对请求进行处理。[5] 可以根据需要创建，或者预先创建一些线程 / 进程。[6] 有些服务器会为每条连接分配一个线程 / 进程，但当服务器同时要处理成百、上千，甚至数以万计的连接时，需要的进程或线程数量可能会消耗太多的内存或系统资源。因此，很多多线程 Web 服务器都会对线程 / 进程的最大数量进行限制。 118

- *复用 I/O 的服务器*（参见图 5-7c）
  为了支持大量的连接，很多 Web 服务器都采用了复用结构。在复用结构中，要同时监视所有连接上的活动。当连接的状态发生变化时（比如，有数据可用，或出现错误时），就对那条连接进行少量的处理；处理结束之后，将连接返回到开放连接列表中，等待下一次状态变化。只有在有事情可做时才会对连接进行处理；在空闲连接上等待的时候并不会绑定线程和进程。

- *复用的多线程 Web 服务器*（参见图 5-7d）
  有些系统会将多线程和复用功能结合在一起，以利用计算机平台上的多个 CPU。多个线程（通常是一个物理处理器）中的每一个都在观察打开的连接（或打开的连接中的一个子集），并对每条连接执行少量的任务。

注 5：进程是一个独立的程序控制流，有自己的变量集。线程是一种更快、更高效的进程版本。单个程序可以通过线程和进程同时处理多件事情。为了便于解释，我们将线程和进程当作是可以互换的概念。但由于性能的不同，很多高性能服务器既是多进程的，又是多线程的。

注 6：会预先创建一些线程的系统被称为“工作池”系统，因为池中会有一组线程在等待工作。



119 图 5-7 Web 服务器输入 / 输出结构

5.6 第三步——处理请求

一旦 Web 服务器收到了请求，就可以根据方法、资源、首部和可选的主体部分来对请求进行处理了。

有些方法（比如 POST）要求请求报文中必须带有实体主体部分的数据。其他一些方法（比如 OPTIONS）允许有请求的主体部分，也允许没有。少数方法（比如 GET）禁止在请求报文中包含实体的主体数据。

这里我们并不对请求的具体处理方式进行讨论，因为本书其余大多数章节都在讨论这个问题。

5.7 第四步——对资源的映射及访问

Web 服务器是资源服务器。它们负责发送预先创建好的内容，比如 HTML 页面或 JPEG 图片，以及运行在服务器上的资源生成程序所产生的动态内容。

在 Web 服务器将内容传送给客户端之前，要将请求报文中的 URI 映射为 Web 服务器上适当的内容或内容生成器，以识别出内容的源头。

5.7.1 docroot

Web 服务器支持各种不同类型的资源映射，但最简单的资源映射形式就是用请求 URI 作为名字来访问 Web 服务器文件系统中的文件。通常，Web 服务器的文件系统中会有一个特殊的文件夹专门用于存放 Web 内容。这个文件夹被称为*文档的根目录*（document root，或 docroot）。Web 服务器从请求报文中获取 URI，并将其附加在文档根目录的后面。

在图 5-8 中，有一条对 /specials/saw-blade.gif 的请求到达。这个例子中 Web 服务器的文档根目录为 /usr/local/httpd/files。Web 服务器会返回文件 /usr/local/httpd/files/specials/saw-blade.gif。

图 5-8 将请求 URI 映射为本地 Web 服务器上的资源

120

在配置文件 httpd.conf 中添加一个 DocumentRoot 行就可以为 Apache Web 服务器设置文档的根目录了：

```
DocumentRoot /usr/local/httpd/files
```

服务器要注意，不能让相对 URL 退到 docroot 之外，将文件系统的其余部分暴露出来。比如，大多数成熟的 Web 服务器都不允许这样的 URI 看到 Joe 的五金商店文档根目录上一级的文件：

```
http://www.joes-hardware.com/../
```

1. 虚拟托管的docroot

虚拟托管的 Web 服务器会在同一台 Web 服务器上提供多个 Web 站点，每个站点在服务器上都有自己独有的文档根目录。虚拟托管 Web 服务器会根据 URI 或 Host 首

部的 IP 地址或主机名来识别要使用的正确文档根目录。通过这种方式，即使请求 URI 完全相同，托管在同一 Web 服务器上的两个 Web 站点也可以拥有完全不同的内容了。

图 5-9 中的服务器托管了两个站点：www.joes-hardware.com 和 www.marys-antiques.com。服务器可以通过 HTTP 的 Host 首部，或根据不同的 IP 地址来区分不同的 Web 站点。

- 当请求 A 到达时，服务器会获取文件 /docs/joe/index.html。
- 当请求 B 到达时，服务器会获取文件 /docs/mary/index.html。

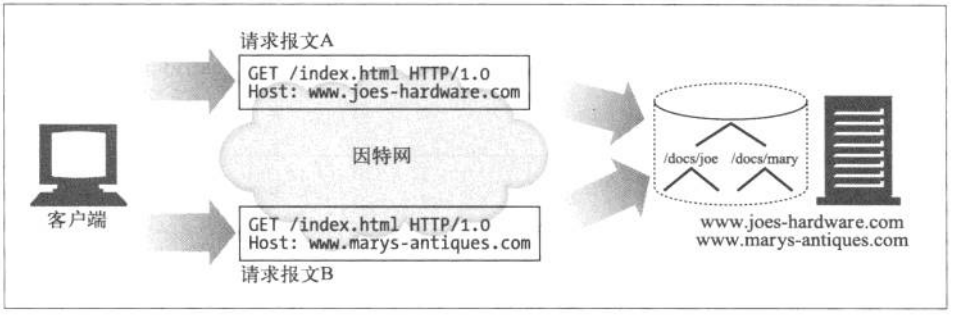


图 5-9　虚拟托管的请求会使用不同的文档根目录

对大多数 Web 服务器来说，配置虚拟托管的文档根目录是很简单的。对常见的 Apache Web 服务器来说，需要为每个虚拟 Web 站点配置一个 VirtualHost 块，而且每个虚拟服务器都要包含 DocumentRoot（例 5-3）。

例 5-3　Apache Web 服务器虚拟主机的 docroot 配置

```
<VirtualHost www.joes-hardware.com>
  ServerName www.joes-hardware.com
  DocumentRoot /docs/joe
  TransferLog /logs/joe.access_log
  ErrorLog /logs/joe.error_log
</VirtualHost>

<VirtualHost www.marys-antiques.com>
  ServerName www.marys-antiques.com
  DocumentRoot /docs/mary
  TransferLog /logs/mary.access_log
  ErrorLog /logs/mary.error_log
</VirtualHost>
    ...
```

更多与虚拟托管有关的信息可以参考 18.2 节。

2. 用户的主目录docroot

Docroot 的另一种常见应用是在 Web 服务器上为人们提供私有的 Web 站点。通常会把那些以斜杠和波浪号（/~）开始，后面跟着用户名的 URI 映射为此用户的私有文档根目录。私有 docroot 通常都是用户主目录下那个名为 public_html 的目录，但也可将其配置为其他值（参见图 5-10）。

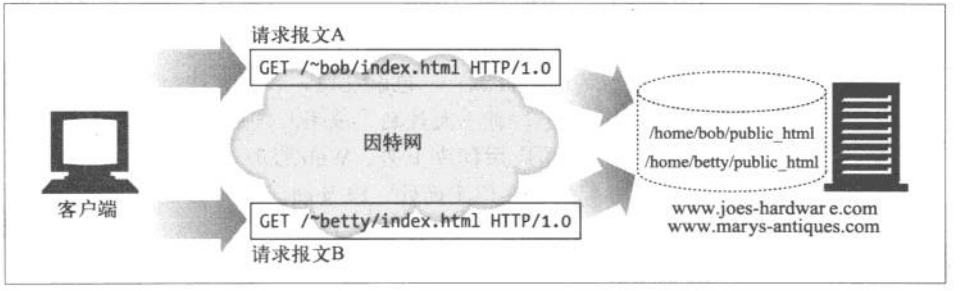


图 5-10　不同用户有不同的 docroot

5.7.2　目录列表

Web 服务器可以接收对目录 URL 的请求，其路径可以解析为一个目录，而不是文件。我们可以对大多数 Web 服务器进行配置，使其在客户端请求目录 URL 时采取不同的动作。

- 返回一个错误。
- 不返回目录，返回一个特殊的默认“索引文件”。
- 扫描目录，返回一个包含目录内容的 HTML 页面。

大多数 Web 服务器都会去查找目录中一个名为 index.html 或 index.htm 的文件来代表此目录。如果用户请求的是一个目录的 URL，而且这个目录中有一个名为 index.html（或 index.htm）的文件，服务器就会返回那个文件的内容。 122

在 Apache Web 服务器上，可以用配置指令 `DirectoryIndex` 来配置要作为默认目录文件使用的文件名集合。指令 `DirectoryIndex` 会按照优先顺序列出所有可以作为目录索引文件使用的文件名。下列配置行会使 Apache 在收到一个目录 URL 请求时，在目录中搜索命令中列出来的任意一个文件：

```
DirectoryIndex index.html index.htm home.html home.htm index.cgi
```

如果用户请求目录 URI 时，没有提供默认的索引文件，而且没有禁止使用目录索引，很多 Web 服务器都会自动返回一个 HTML 文件，此文件中会列出那个目录里

的文件名，以及每个文件的大小和修改日期，还包括到每个文件的 URI 链接。使用这个文件列表可能会很方便，但有些好事者也可以通过它在 Web 服务器上找到一些通常找不到的东西。

可以通过以下 Apache 指令禁止自动生成目录索引文件：

```
Options -Indexes
```

5.7.3 动态内容资源的映射

Web 服务器还可以将 URI 映射为动态资源——也就是说，映射到按需动态生成内容的程序上去（参见图 5-11）。实际上，有一大类名为*应用程序服务器*的 Web 服务器会将 Web 服务器连接到复杂的后端应用程序上去。Web 服务器要能够分辨出资源什么时候是动态的，动态内容生成程序位于何处，以及如何运行那个程序。大多数 Web 服务器都提供了一些基本的机制以识别和映射动态资源。

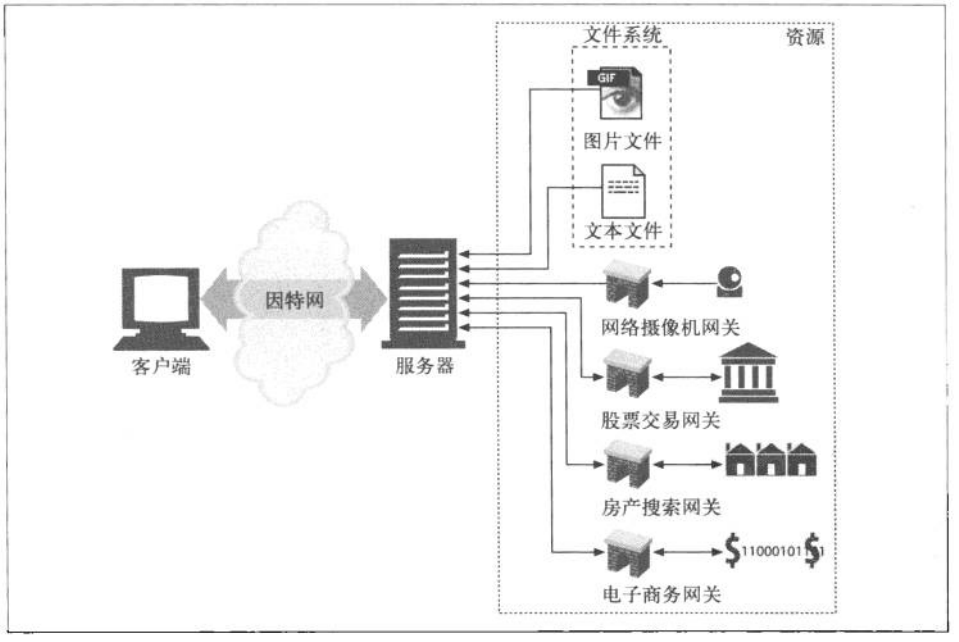


图 5-11 Web 服务器可以提供静态资源和动态资源

Apache 允许用户将 URI 路径名组件映射为可执行文件目录。服务器收到一条带有可执行路径组件的对 URI 的请求时，会试着去执行相应服务器目录中的程序。例如，下列 Apache 配置指令就表明所有路径以 /cgi-bin/ 开头的 URI 都应该执行在目录 /usr/local/etc/httpd/cgi-programs/ 中找到的相应文件：

```
ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-programs/
```

Apache 还允许用户用一个特殊的文件扩展名来标识可执行文件。通过这种方式就可以将可执行脚本放在任意目录中了。下面的 Apache 配置指令说明要执行所有以 .cgi 结尾的 Web 资源：

```
AddHandler cgi-script .cgi
```

CGI 是早期出现的一种简单、流行的服务端应用程序执行接口。现代的应用程序服务器都有更强大更有效的服务端动态内容支持机制，包括微软的动态服务器页面（Active Server Page）和 Java servlet。 123

5.7.4 服务器端包含项

很多 Web 服务器还提供了对服务器端包含项（SSI）的支持。如果某个资源被标识为存在服务器端包含项，服务器就会在将其发送给客户端之前对资源内容进行处理。

要对内容进行扫描，以查找（通常包含在特定 HTML 注释中的）特定的模板，这些模板可以是变量名，也可以是嵌入式脚本。可以用变量的值或可执行脚本的输出来取代特定的模板。这是创建动态内容的一种简便方式。

5.7.5 访问控制

Web 服务器还可以为特定资源进行访问控制。有请求到达，要访问受控资源时，Web 服务器可以根据客户端的 IP 地址进行访问控制，也可以要求输入密码来访问资源。

更多与 HTTP 认证有关的信息请参见第 12 章。 124

5.8 第五步——构建响应

一旦 Web 服务器识别出了资源，就执行请求方法中描述的动作，并返回响应报文。响应报文中包含有响应状态码、响应首部，如果生成了响应主体的话，还包括响应主体。3.4 节详细介绍了 HTTP 响应代码。

5.8.1 响应实体

如果事务处理产生了响应主体，就将内容放在响应报文中回送过去。如果有响应主体的话，响应报文中通常包括：

- 描述了响应主体 MIME 类型的 `Content-Type` 首部；

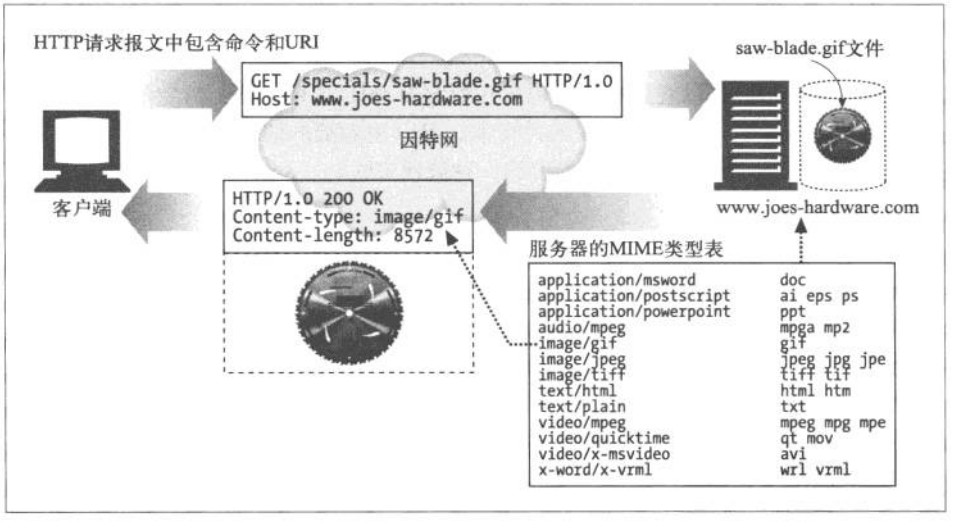
- 描述了响应主体长度的 Content-Length 首部；
- 实际报文的主体内容。

5.8.2 MIME类型

Web 服务器要负责确定响应主体的 MIME 类型。有很多配置服务器的方法可以将 MIME 类型与资源关联起来。

- **MIME 类型（mime.types）**

  Web 服务器可以用文件的扩展名来说明 MIME 类型。Web 服务器会为每个资源扫描一个包含了所有扩展名的 MIME 类型的文件，以确定其 MIME 类型。这种基于扩展名的类型相关是最常见的，参见图 5-12。



125 图 5-12 Web 服务器用 MIME 类型文件来设置资源输出的 Content-type 首部

- **魔法分类（Magic typing）**

  Apache Web 服务器可以扫描每个资源的内容，并将其与一个已知模式表（被称为魔法文件）进行匹配，以决定每个文件的 MIME 类型。这样做可能比较慢，但很方便，尤其是文件没有标准扩展名的时候。

- **显式分类（Explicit typing）**

  可以对 Web 服务器进行配置，使其不考虑文件的扩展名或内容，强制特定文件或目录内容拥有某个 MIME 类型。

- *类型协商*

  有些 Web 服务器经过配置，可以以多种文档格式来存储资源。在这种情况下，可以配置 Web 服务器，使其可以通过与用户的协商来决定使用哪种格式（及相关的 MIME 类型）“最好”。这个问题将在第 17 章讨论。

  还可以通过配置 Web 服务器，将特定的文件与 MIME 类型相关联。

5.8.3 重定向

Web 服务器有时会返回重定向响应而不是成功的报文。Web 服务器可以将浏览器重定向到其他地方来执行请求。重定向响应由返回码 3XX 说明。`Location` 响应首部包含了内容的新地址或优选地址的 URI。重定向可用于下列情况。

- *永久删除的资源*

  资源可能已经被移动到了新的位置，或者被重新命名，有了一个新的 URL。Web 服务器可以告诉客户端资源已经被重命名了，这样客户端就可以在从新地址获取资源之前，更新书签之类的信息了。状态码 301 Moved Permanently 就用于此类重定向。

- *临时删除的资源*

  如果资源被临时移走或重命名了，服务器可能希望将客户端重定向到新的位置上去。但由于重命名是临时的，所以服务器希望客户端将来还可以回头去使用老的 URL，不要对书签进行更新。状态码 303 See Other 以及状态码 307 Temporary Redirect 就用于此类重定向。

- *URL 增强*

  服务器通常用重定向来重写 URL，往往用于嵌入上下文。当请求到达时，服务器会生成一个新的包含了嵌入式状态信息的 URL，并将用户重定向到这个新的 URL 上去。[7] 客户端会跟随这个重定向信息，重新发起请求，但这次的请求会包含完整的、经过状态增强的 URL。这是在事务间维护状态的一种有效方式。状态码 303 See Other 和 307 Temporary Redirect 用于此类重定向。 126

- *负载均衡*

  如果一个超载的服务器收到一条请求，服务器可以将客户端重定向到一个负载不太重的服务器上去。状态码 303 See Other 和 307 Temporary Redirect 可用于此类重定向。

- *服务器关联*

  Web 服务器上可能会有某些用户的本地信息；服务器可以将客户端重定向到包含了那个客户端信息的服务器上去。状态码 303 See Other 和 307 Temporary Redirect 可用于此类重定向。

注 7：有时会将这些经过扩展和状态增强的 URL 称为“胖 URL”。

- 规范目录名称
  客户端请求的URI是一个不带尾部斜线的目录名时，大多数Web服务器都会将客户端重定向到一个加了斜线的URI上，这样相对链接就可以正常工作了。

5.9 第六步——发送响应

Web服务器通过连接发送数据时也会面临与接收数据一样的问题。服务器可能有很多条到各个客户端的连接，有些是空闲的，有些在向服务器发送数据，还有一些在向客户端回送响应数据。

服务器要记录连接的状态，还要特别注意对持久连接的处理。对非持久连接而言，服务器应该在发送了整条报文之后，关闭自己这一端的连接。

对持久连接来说，连接可能仍保持打开状态，在这种情况下，服务器要特别小心，要正确地计算Content-Length首部，不然客户端就无法知道响应什么时候结束了（参见第4章）。

5.10 第七步——记录日志

最后，当事务结束时，Web服务器会在日志文件中添加一个条目，来描述已执行的事务。大多数Web服务器都提供了几种日志配置格式。更多细节请参见第21章。

5.11 更多信息

更多有关Apache、Jigsaw和ident的信息，参见以下参考资源

- *Apache:The Definitive Guide*[8]（《Apache权威指南》）
  Ben Laurie和Peter Laurie编写，O'Reilly & Associates公司出版。
- *Professional Apache*
  Peter Wainwright编写，Wrox公司出版。
- http://www.w3c.org/Jigsaw/
  Jigsaw——W3C的服务器W3C联盟Web站点。
- http://www.ietf.org/rfc/rfc1413.txt
  RFC 1413，M. St. Johns编写的“Identification Protocol”（“标识协议”）。

注8：本书影印版由中国电力出版社出版。（编者注）