CHAPTER

第 1 章

Web 服务器概述

在本章中，我们将带领你去了解 Web 服务器的发展情况。首先对 Internet 和 WWW 的发展历史做简要的概述，重点介绍 Web 服务器的发展过程，其中尤其关注 Apache 服务器的发展历程。

其次，我们将重点介绍 Web 服务器的工作原理。从三个方面进行简单概要的介绍：HTTP 协议、HTML 标记语言，以及 Web 服务器端原理。

1.1 WWW 概述

WWW 英文全称 World Wide Web，也称万维网，它是目前 Internet 上最为流行的技术。WWW 给用户带来的最大便利就是任何时候只要用户能够访问 Internet 并且安装了 IE 等浏览器（通常会随操作系统一起提供），就可以使用 WWW。目前 WWW 已经成为人类最大的信息资源中心。

在了解 WWW 之前，我们有必要回顾 WWW 的发展历史。

本章从两个重要的方面来讨论 WWW 的发展历史：首先是整个 Internet 的发展历史，它是整个网络的基础；其次是 HTTP 协议，它是构成整个 Web 的协议基础。

图 1-1 展示了在整个 WWW 发展史上的一些重要的里程碑：

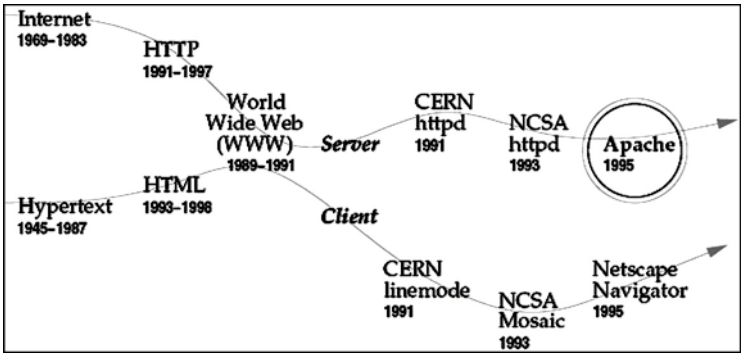

图 1-1 WWW 发展历史上的一些重要的里程碑

下面我们将结合图 1-1 简要介绍整个 WWW 的发展历史。

1.1.1 Internet 概述

1957 年，前苏联发射了第一颗人造地球卫星 Sputnik。针对此事，美国在国防部内部组成了高级研究计划署（**A**dvanced **R**esearch **P**roject **A**gency，ARPA），从而确保美国在军事应用方面的科技领先优势。1969 年，美国国防部构建了 ARPANET 网络（见图 1-2），从而进

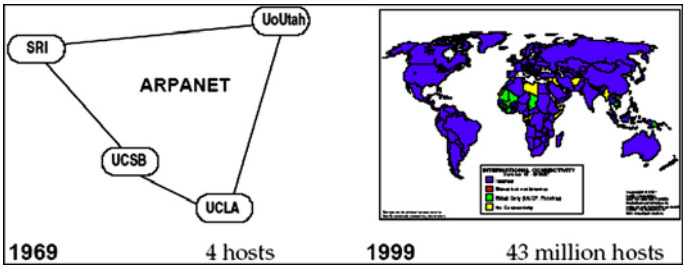

图 1-2 初始的 ARPANET 网络

一步推进了网络的研究，并创建了 4 个最初的网络节点：洛杉矶的加利福尼亚大学（UCLA）、斯坦福研究院（SRI）、圣巴巴拉的加利福尼亚大学分校，以及犹他州立大学。

最初的 ARPANET 网络包括了一条 50Kbps 的专线并使用网络控制协议（**N**etwork **C**ontrol **P**rotocol，NCP）进行通信，这是第一个主机对主机的通信协议。随后，越来越多的计算机加入到 ARPANET 网络中，用于讨论和归档 ARPANET 使用的协议和软件的几百个 RFC（Request For Comments）和 Internet 草案也随之产生了。1974 年，Vint Cerf 和 Bob Kahn 发布了基于包技术的网络互联协议，该协议详细描述了 TCP（**T**ransmission **C**ontrol **P**rogram，传输控制程序）的设计细节。1984 年，TCP 协议又被分割为两种协议：传输控制协议（TCP）和 Internet 协议（IP）。

1982 年，美国国防部对外宣称 TCP 和 IP（目前统称为 TCP/IP）这两个协议为它的官方协议，国防部的声明形成了最早的 Internet 定义，它通常是指一系列相互连接的网络，尤其是使用 TCP/IP 协议的网络。

1983 年，ARPANET 也从最初的 NCP 协议切换到 TCP/IP 协议，这样，最早的 Internet 就诞生了。在随后的几年中，网络经历了爆炸式的发展：1984 年，全球接入 Internet 的计算机数目超过了 1 000 台；1987 年，这个数目达到了 10 000 台；到了 1989 年，则达到了 100 000 台；1992 年为 1 000 000 台；1996 年为 10 000 000 台；1999 年的时候就已经达到 43 000 000 台。今天（2009 年），全球接入 Internet 的计算机数目更是达到了上百亿台。这种快速增长的趋势目前仍然在继续。

1.1.2 超文本的概念

与 Internet 的诞生相比，超文本的概念出现得更早，它最早可以追溯到 1945 年。范尼瓦·布什于 1945 年发表了文章《按照我们的想象（As We May Think）》呼唤在有思维的人和所有的知识之间建立一种新的关系。为此他提出创建一种被称为“memex”的设备，在其中每个人都能把他所拥有的各种信息（比如图书）保存起来。由于条件所限，范尼瓦·布什的思想在当时并没有变成现实，但是他的思想在此后的 50 多年中产生了巨大影响。人们普遍认为超文本的概念源于范尼瓦·布什。

尽管超文本的思想很久之前就存在，但是超文本这个词汇一直到 1965 年才被 Xanadu 项目的 Ted Nelson 提出来。后来，超文本一词得到世界的公认，成为这种非线性信息管理技术的专用词汇。1999 年国际超文本大会设立的新人奖即以 Nelson 的名字命名。

后来，布朗大学（位于 Rhode Island）的 Andries van Dam 于 1967 年开发了超文本编辑系统（**H**ypertext **E**diting **S**ystem，HES）及文件获取和编辑系统（**F**ile **R**etrieval and **E**diting **S**ystem，FRESS）——这是最早的真正超文本应用。1968 年，Douglas C. Engelbart（大多数人知道的就是他于 1963 年发明了计算机鼠标）在加利福尼亚州的圣弗朗西斯科举办的 FJCC（**F**all **J**oint **C**omputer **C**onference）会议的一次多媒体会议中演示了他的 NLS，这是一种在

线系统（o**n**L**i**ne **S**ystem），后来命名为 Augment 系统（Augment System）。NLS 中已经具备了若干超文本的特性。此外，Douglas C. Engelbart 还发明了鼠标、多窗口、图文组合文件等，甚至可以说是他发明了超文本。1999 年国际超文本大会设立的最佳论文奖即以 Engelbart 的名字命名。

在这次先锋性的事件之后，许多类似的系统相继涌现出来，这些系统无不深受超文本主义的影响，包括 1975 年卡耐基·梅隆大学的 ZOG，1978 年 MIT 大学的 Andy Lippman 的白杨树镇电影地图（Aspen Movie Map）系统，1984 年 Telos 的 Filevision，1985 年 Janet Walker 的工作站连接手册（Symbolics Document Examiner），1985 年布朗大学的 Norman Meyrowitz 的 Intermedia，等等。

1987 年，Apple 公司推出了 HyperCard（超文本卡），HyperCard 是 20 世纪 80 年代末期世界上最流行的超文本系统。从 1987 年到 1992 年，Apple 公司随每一台销售出去的机器赠送一套 HyperCard。HyperCard 的流行使超文本的基本概念得到了普及，结束了超文本仅仅作为研究主题的状况，被广泛接受为一种新技术，并且在应用开发特别是教育系统的开发方面起到举足轻重的作用。Hyper-Card 被认为是计算机历史上的里程碑，以及教育软件的范例。

第一次国际超文本技术研讨会于 1987 年 11 月 13 日至 15 日在美国的北卡罗来纳州的查布尔希尔（Chapel Hill）召开。这个会议的召开标志着超文本已经受到广泛的关注，正在形成一个新的领域。

1.1.3 WWW 的历史

早在 1965 年，Ted Nelson 就率先提出了“超文本”的概念，并在 1967 年把相应的实现计划命名为“Xanadu”。该项目于 1987 年才算完成，只不过当时只能运行在一个 SUN 工作站上，而且功能非常简单粗糙，同时该项目被重新命名为“Conklin”。

1989 年 3 月，CERN（欧洲粒子物理实验室）的 Tim Berners-Lee（Tim B.L）写了一篇名为“信息管理的建议（Information Management:A Proposal）”的文章，其中他试图给出“How will we ever keep track of large projects?”问题的答案。

在得到 Tim Berners-Lee 的老板 Mike Sendall 的同意之后，Tim Berners-Lee 开始基于 NeXTStep 开发环境开发一个超文本图形的浏览器和编辑器。Tim Berners-Lee 将这个程序命名为“WorldWideWeb”，稍后又将其重新命名为“Nexus”，从而避免应用程序和抽象信息空间之间的相互混淆。两个月之后，项目开发完成，这个“WorldWideWeb”也迅速成为第一个超文本系统，同时公众也开始接受“WWW”。

欧洲粒子物理实验室（即 CERN）的 Tim Berners-Lee 受到 Nelson 的影响，提出一项计划，目的是使科学家们能够很容易地查询同行的文章。该项目从 1990 年 10 月开始到同年 12 月完成。项目的最终成果就是形成命令行方式的浏览器和 NeXTStep 浏览器，该浏览器能够浏览超文本文件及 CERN 的 USENET。1992 年 7 月，WWW 在 CERN 内部被广泛使用。直到 1993 年 1 月，全世界共有 30 台 WWW 服务器，并有多个浏览器版本发行。之后，超

文本概念及它在 Internet 上的使用得到了进一步的发展。

整个 WWW 可以从四个方面进行描述：HTML 标记语言、HTTP 通信协议、WWW 客户端的发展，以及 WWW 服务器端的发展。本章我们只简要了解 WWW 客户端和服务器端的发展，关于 HTML 和 HTTP 协议相关的内容可以参考相应的 RFC 文档。

HTML 标记语言

HTML 作为定义万维网的基本规则之一，最初由 Tim Berners-Lee 于 1989 年在 CERN 研制出来。HTML 的设计者是这样考虑的：HTML 格式将允许科学家们透明地共享网络上的信息，即使这些科学家使用的计算机差别很大。因此，这种格式必须具备如下几个特点。

- 独立于平台，即独立于计算机硬件和操作系统。这个特性对各种系统是至关重要的，因为在这个特性中，文档可以在具有不同性能（即字体、图形和颜色差异）的计算机上以相似的形式显示文档内容。
- 超文本。允许文档中的任何文字或词组参照另一文档，这个特性将允许用户在不同计算机中的文档之间及文档内部漫游。
- 精确的结构化文档。该特性将允许某些高级应用，如 HTML 文档和其他格式文档间互相转换，以及搜索文本数据库。

Tim Berners-Lee 选择使用标准通用标记语言（**S**tandard **G**eneralized **M**arkup **L**anguage，SGML）作为 HTML 的开发模板。作为一种当时刚刚出现的国际标准，标准通用标记语言具有结构化和独立于平台的优点。

自 1989 年以来，HTML 及万维网的使用和发展有了巨大的变化。当 NCSA（**N**ational **C**enter for **S**upercomputing **A**pplications，美国国家超级计算机应用中心）在 1993 年初首次构建 Mosaic 浏览器时，NCSA 的科学家们把自己需要的特性添加到 HTML 中，包括直接插入图形。在允许人们把位图、照片和图表放入文档中以后，万维网的规模和使用出现了爆炸性的增长，第二年，HTML 的发展很快。HTML 的新标记不时地被一个又一个的浏览器引入，有一些新标记流行起来，而有一些又消失了。有些增加部分设计得很糟，很多甚至不遵从 SGML 规范。

到了 1994 年，HTML 几乎在以失控的状态发展。在 IETF（**I**nternet **E**ngineering **T**ask **F**orce）的主持下，1995 年 11 月在瑞士日内瓦举行的第一次 WWW 会议上成立了一个 HTML 工作小组。它的主要任务是把 HTML 形式化为一种 SGML DTD，并称之为 HTML Level 2（HTML 2.0，由本尼斯李最初设计的 HTML 被定义为 Level 1）。标准化之后，HTML 就可以被安全地扩展到将来的各个级别的版本，从而利用了 SGML 的实质性能和它的格式化结构。

尽管有关各方从来没有取得完全一致的共识，但万维网联盟 HTML 工作组（World Wide Web Consortium's HTML Working Group）还是集中了 1996 年的万维网发展的成果，产生了 HTML 3.2 版本。目前 HTML 已经发展到了 4.0 版本。

客户端的发展

客户端的发展受制于两方面的因素：超文本标记语言（HTML），以及呈现给终端用户使用的浏览器，只有通过浏览器才能将 HTML 数据显示在桌面上。

1990 年 11 月，一位来自莱斯特工艺学院（Leicester Polytechnic，现德蒙特福德大学 De Montfort University）数学系的实习生妮可拉 • 派罗（Nicola Pellow）加入了 Tim Berners-Lee 的工作组，共同参与首个浏览器 WWW 的开发。

1991 年 3 月，世界上首个浏览器 WWW 问世，但是这个浏览器只能运行在 NeXT 平台下。几乎在同时，为了解决跨平台使用的问题，Nicola Pellow 以 WWW 浏览器为基础开发了一款命令行式浏览器，并且给它取了个简单的名字 Line-Mode Browser。这个名字如果翻译成中文就是“命令行浏览器”，所以也可以说她干脆没有取名字。在这之后，一系列的浏览器蜂拥而出，如表 1-1 所示。

表 1-1

浏览器名称	推出时间	特点
Erwise	1992 年 4 月 29 日	首个非 NeXT 平台浏览器、首个 Unix 平台浏览器、首个诞生在 CERN 以外的浏览器
ViolaWWW	1992 年 5 月	首个流行浏览器、第二个 Unix 平台浏览器
Midas	1992 年夏	
MacWWW	1992 年	首个 Mac 平台下的浏览器
Libwww	1992 年底	

但是这些浏览器的影响远不如 Mosaic。1993 年 2 月伊利诺伊大学 Urbana-Champaign 分校的国家超级计算应用中心 NCSA（**N**ational **C**enter for **S**upercomputing **A**pplication） 推出了 Mosaic 浏览器的第一个稳定的版本。当时命名为 NCSA Mosaic for X 0.10 available ，这便是世界上第一个图形化的浏览器。9 月，NCSA 的 Aleksandar Totic 完成了 Macintosh 版本的开发，这也使 Mosaic 成为了第一个跨平台的浏览器。Mosaic 浏览器在继承前人成果的基础上，不仅能够处理基本的文档共享和链接，而且还支持多种网络协议及 MIME 协议，用户在浏览器中就可以直接浏览图形甚至声音。因此，Mosaic 一经发布出来就立即成为互联网上使用最多的应用程序。基于 Mosaic 浏览器，Netscape 公司发布了著名的网景浏览器 Netscape Navigator，它曾经占据了半数以上的桌面。到了现在，大部分人都知道占有率第一的已经不是 Netscape Navigator，而是 IE。Mosaic 则转向了全部开源，目前基于 Mosaic 浏览器的 FireFox 浏览器已经让 IE 腹背受敌。

服务器的发展

与客户端的精彩万千、丰富有趣的改进相比，服务器端的变化要平淡得多。整个服务器的发展历史基本上就是 Apache 发展的历史。

Tim Berners-Lee 在 1991 年开始编写 CERN HTTP 服务器，1993 年和 1994 年 Ari Luotonen 及 Henrik F. Nielsen 也加入了进来，这是世界上第一个真正的 Web 服务器。1993 年，Tony Sanders 使用 Perl 编写了一个 Web 服务器并取名为 Plexus，与此同时，美国国家超级计算机应用中心（NCSA）的 Robert McCool 使用 C 语言编写了一个完整的应用程序 NCSA httpd，这是世界上第一个具备完整的 Web 服务器功能的 Web 服务器。

在后来的两年中，NCSA httpd 服务器变得非常流行。但在 McCool 于 1994 年离开了 NCSA 之后，httpd 的开发和维护工作就被停止了。而人们对 httpd 的需求却在继续，为了满足自己的需要，不同的人都会在 httpd 中打上自己的补丁。后来，一些人为了能够相互之间交换补丁，便开始有意收集这些补丁。

1995 年，Brian Behlendorf 及其他的一些开发者形成了一个小组开始着手收集 NCSA 服务器的各种补丁并保存到一个中央库中。最初的 Apache 工作组使用 NCSA 服务器作为基础，并且努力对其进行改善，一方面，他们对所有的已经公开的漏洞和缺陷提供补丁方案，另一方面，他们也增加了许多看起来很有用的增强功能。当在他们的服务器上测试了这些工作之后，1995 年的 4 月，他们发布了第一次的官方版本，版本号为 0.6.2，同时取名为 Apache，意思为这个服务是一个打了各种补丁的服务器（**A PAtCHy sErver**）。与此同时，NCSA 也重新开始了它的服务器开发工作，为了确保在两个产品中获得尽可能好的质量，NCSA 的两个开发人员也加入到了该开发团队中，以共享思想和补丁程序。

与此同时，这个小组对外的正式名称为 Apache 组织，不过他们与一般的组织并不相同，他们没有正规的组织结构，也从来不见面，他们之间仅仅是通过电子邮件进行沟通，而且工作仅仅是在他们空闲的时间，并且完全是自愿的，没有任何回报的。第二年，Apache 就超过 NCSA 成为 Internet 上最广泛使用的 Web 服务器，目前为止它的市场占有率达到 60%左右，而微软的 IIS 也仅仅为 30%，其余各种 Web 服务器共占 10%的市场份额。另外，在 UNIX 市场上，Apache 占有绝对的优势，占有率达到 95%左右，而 IIS 则仅局限于 Windows 市场上。

尽管 Apache 的 0.6.2 版本获得了非常大的成功，但是所有的开发人员都意识到这个服务器的设计不可能长久地被使用。因此他们采取了两手并行的策略：一方面，对新的版本 0.7.x 继续进行开发，它们基于 0.6.2 的体系结构；另一方面，Robert Thau 开始负责设计新的体系结构，这个新的服务器体系结构中包含了众多的现在众所皆知的特性，比如模块化结构、基于内存池的分配策略等。开发小组在 1995 年 7 月转移到这个新的代码基址，并且为其增加了所有已经添加到 0.7.x 分支的特性。1995 年 8 月，这个版本以 0.8.8 发布。在将这个代码基址移植到更多的平台，编写了一组新的文档而且以更为标准化的模块形式增加了一些特性之后，Apache 的 1.0 版本在 1995 年 12 月发布。

在 1996 年 7 月，Apache 1.1 版本发布，在该版本中包含了大量的性能改善，并增加了一些新的特性，比如缓存代理模块。尽管这个模块并没有得到很好的维护，其中的一个问题直到 2.0 版本才得到彻底的解决，不过很多用户还是将 Apache 作为简单的代理服务器使用。这个版本中还包含了使用一个服务器实例侦听多个端口的能力。

下一个版本就是 Apache 1.2，它包含了对 HTTP/1.1，以及作为特定用户运行的 CGI 脚本的支持。在这个版本之后，Apache 工作组开始讨论 2.0 的新的特性集合。

不过 Apache 1.2 并不是最终的 1.x 版本。1998 年 5 月，Apache 组织发布了 1.3 版本。与前几个版本相比，1.3 最大的变化就是增加了对 Windows 平台的支持，遗憾的是，1.3 系列中并没有对 Windows 移植进行足够的优化，因此 Apache 在 Windows 下的运行性能并不高。不过对于 Unix 平台而言，Apache 1.3 是相当稳定的版本，即使到现在，Apache 1.3 也还有相当高的使用率。

2000 年的时候，推出了 Apache 2.0 版本，该版本的推出历时三年，整个体系结构发生了巨大的变化，引入了许多新的特征。Apache 版本更新的速度非常地缓慢，直到八年后的今天，它的最新版本才是 Apache 2.3。到目前为止，Apache 2.0 系列已经渐趋稳定，使用率正在逐步提高。

在 2000 年 3 月的 Apache 会议上，Apache 2.0 Alpha 1 版本发布，Apache 2.0 对整个服务器的体系结进行了完全重新的设计。Apache 2.0 中将操作系统相关的细节封装到适配层 APR（第 3 卷详细讨论）中，这样与 Apache 2.0 以前的版本（主要是 Apache 1.3）相比，2.0 版本更加容易移植到不同的操作系统平台中。更加令人惊讶的是，Apache 的模块化体系结构如此之强，以至于 Apache 2.0 可以作为一个通用的服务器平台而出现，而不仅仅是作为 Web 服务器，实现这种功能只需要在 Apache 核心模块上添加特定的模块即可。

1.2 HTTP 服务器

1.2.1 HTTP 服务器简介

如果你要了解一个软件产品，最好的主意就是了解该软件能够实现什么样的功能。对服务器应用程序而言，通常要了解四方面的内容：客户端、服务器端、客户端和服务器端的协议及服务器端能够提供的资源。这一点对于 HTTP 服务器也不例外。

整个 Web 可以分为两个重要的组成部分：客户端和服务器端。当客户端需要请求特定的 URI 时，它将与服务器建立连接，并通过 HTTP 协议发送请求至 Web 服务器。Web 服务器接收到客户端的请求之后将生成响应内容，同时将该响应内容通过连接返回给客户端。浏览器和服务器之间的通信过程可以用图 1-3 进行说明。

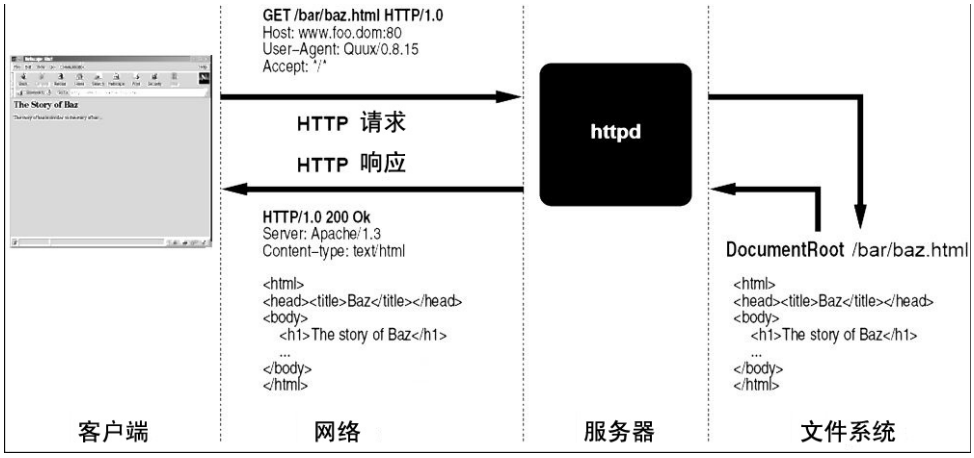

图 1-3 浏览器/服务器通信过程示意图

1.2.2 HTTP 服务器功能

正常情况下，一个 HTTP 服务器会等待浏览器发送的请求，并根据 HTTP 协议进行响应。客户端总是请求某个特定的文档。服务器将检查该请求，同时将客户端请求的文档映射到服务器本地的文件系统中，或者将该请求转发给一个特定的应用程序，并由该应用程序负责对请求进行处理，生成响应内容。一旦处理完毕，服务器将把处理结果返回给客户端。

图 1-4 的左侧部分演示了一个非常简单的系统结构：用户与浏览器打交道，浏览器接受用户的输入和数据，然后将请求发送到 HTTP 请求服务器，而请求服务器则从磁盘上读取客户端需要的文件，再发送给客户端。

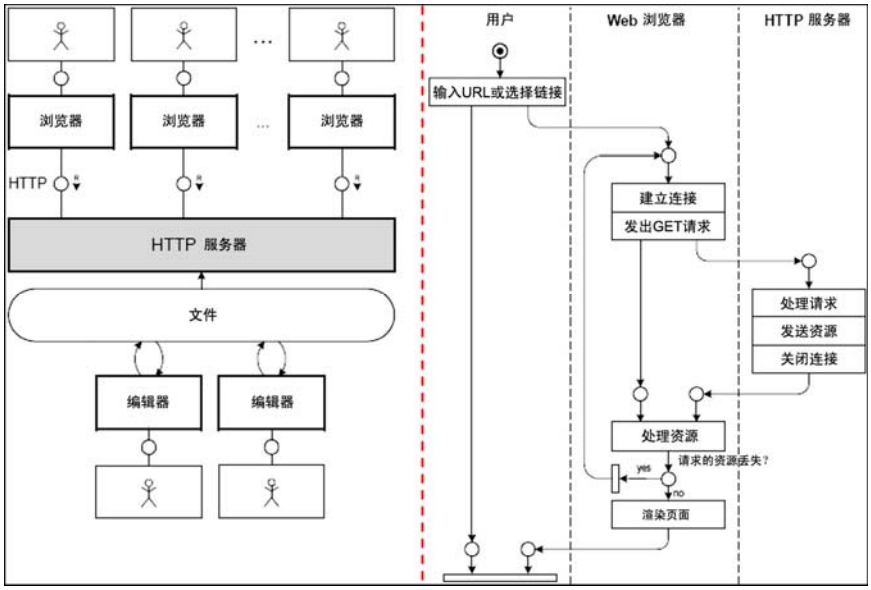

图 1-4 HTTP 服务器流程

图 1-4 的右侧部分描述了系统中发生的内容：用户在浏览器中输入 URL 或点击某个超链接，浏览器从输入的 URL 或超链接中获取服务器地址，然后与服务器建立 TCP/IP 连接。使用该连接，浏览器可以向 HTTP 服务器发送一个 GET 请求，服务器接收到该 URL 地址后从 URL 中提取出请求的资源名称。

HTTP 服务器读取请求并对请求进行处理，使用打开的 TCP/IP 连接，服务器在它的响应中发送客户端需要的资源，发送完毕后服务器将关闭该连接。浏览器接收到服务器返回的数据后，它将检测这些数据。HTML 文档中可以包含资源链接（比如图片、Flash 或 Java applets）。对于这些资源，客户端必须再次向服务器发送请求获取。一旦所有的内容都下载完毕后，浏览器就可以将这个 HTML 展现在浏览器中。

图 1-5 描述了 HTTP 服务器更多的细节：初始化之后，服务器就进入了请求——响应循环阶段，它将等待一个客户端请求，然后对该请求进行检查，将请求的 URL 映射为特定的文件，如果该文件存在，则将该文件发送返回给客户端，如果不存在，则返回一个错误消息。

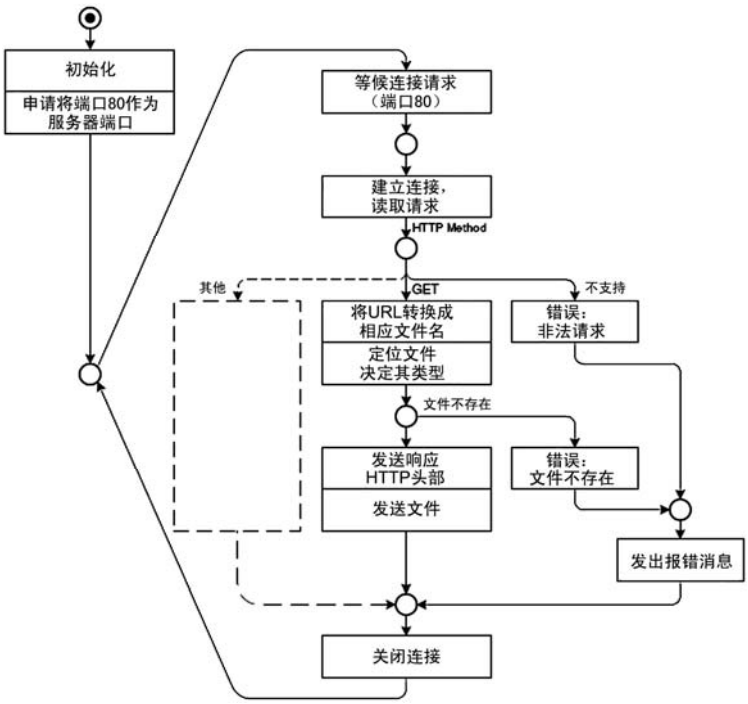

图 1-5 HTTP 服务器内部处理流程

图 1-5 所演示的只是一个最最简单的 Web 服务器，如果要实现对应的功能，不会超过 200 行代码。一个有用的 Web 服务器通常会更复杂，它们还会包含以下一些额外功能。

- 完整地实现 HTTP/0.9、HTTP/1.0 及 HTTP/1.1 协议支持。
- 能够处理并发请求，支持多进程或多线程。
- 提供开发接口，允许开发人员自行增加额外的功能。
- 服务器安全机制的实现。
- 动态内容生成，允许服务器通过 CGI、脚本语言（比如 perl）和服务器端（包含 SSI）生成动态的 Web 页面。
- 支持虚拟主机。
- 支持代理功能。
- 允许根据 MIME 类型协商选择合适的返回资源。

从管理的角度而言，HTTP 服务器还需要额外的要求，比如：

- 健壮性和稳定性，支持 7×24 小时连续运行。
- 易配置性。

- 用户可以在不停止服务器的情况下修改配置。
- 易管理性，管理员可以通过辅助工具对服务器进行有效的管理。
- 丰富的日志。

1.2.3 WWW 文档

WWW 服务器上的文档是以层状或树型结构进行组织的。与传统的层次化文件系统相比，文档树也有一个根，以及一些包含若干子目录和文件的目录。WWW 服务器上的每个文档都有一个唯一的名字，该名字取自根“/”开始的路径，所以一个名为 tree.htm 的文件可能有一个类似/some/where/in/the/tree.html 的完整名字。

但是需要特别注意的是：在 WWW 服务器上包含 Web 文档的文件实际上肯定有许多不同的组织方式。Web 文档树只是反映了一个具有复杂结构的文档集的简单视图。这个简单视图作用重大：它帮助人们在复杂的服务器系统中查找需要的东西。

WWW 服务器上的每个文档都是整个文档树的一部分。它们有各自的名字，从而使浏览器可以对它们包含的信息提出请求。HTML 文档中的超链接，使用文档名来指向对应的信息。链接可以是完整的文档名字，如：
http://www.server.org/there/silly.html

或者是当前文档的名字，如：

./silly.gif

第二种方式表明 silly.gif 与包含该链接的文档处于同一路径中，也就是说，如果存在/there/silly.html，silly.gif 的全名应该是 *http://www.server.org/there/silly.gif*。

相对名称通常用于内嵌的图像。一般来说，一个文档所包含的各个部分都在 WWW 服务器的同一地方。

需要强调的是，Web 文档树并不能真实反映 Web 文档的组织结构。那么，Web 文档为什么不组织成为简单的树型结构呢?这是由很多原因造成的。比如 Web 文档是由若干个小组分别制作的，它们每个都有自己的工作区和存储区；再如，Web 文档太多，使得这些文档只保存在一个文件系统下甚至一台计算机上都显得过于庞大。Web 服务器的流行和高速发展，使得它有必要将文档分布到若干台服务器上，以提高整个 Web 系统的最大承受能力。

图 1-6、图 1-7、图 1-8 给出了 WWW 服务器上文档组织的三种最重要的形式。所有的文档可以位于同一台机器上，构成一棵树，也可以分散到几台机器上，还可以在不同的机器上对同一文档进行镜像。需要注意的是，尽管内部组织方式不同，但是对于用户而言，则没有任何的差异。

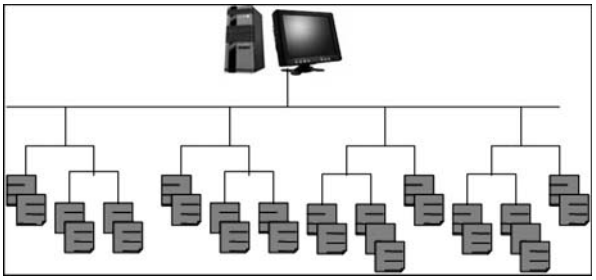

图 1-6 Web 服务器文档组织形式：所有文档位于同一机器上

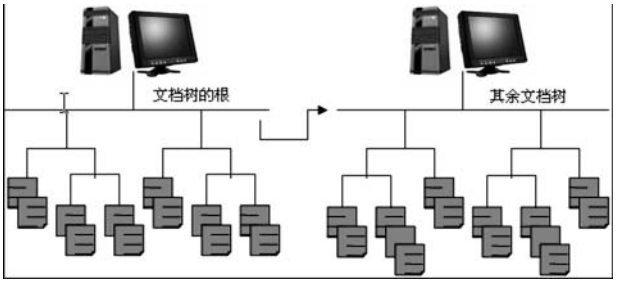

图 1-7 Web 服务器文档组织形式：WWW 的文档分布于不同的机器上，相互之间没有形成镜像

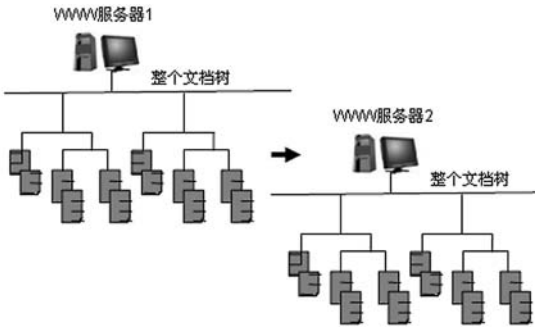

图 1-8 Web 服务器文档组织形式：WWW 文档在不同的机器上形成镜像

1.2.4 工作方式

WWW 服务器拥有一系列组织成文档树的信息，并遵循 HTTP 协议为客户服务。那么它是如何工作的呢？我们分步骤展示 WWW 服务器的工作原理。

等待客户请求

WWW 服务器与其余的任何一个服务器都类似，它必须等待来自客户端的请求。服务器侦听一个指定的端口直到有请求进来。在处理请求之前，服务器处于休眠状态，此时可以用

图 1-9 描述。

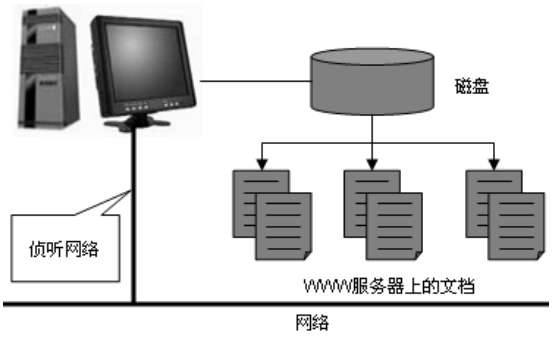

图 1-9 Web 服务器等待客户请求

客户请求到来

客户端通过浏览器对文档提出请求。用户可能是直接在浏览器中输入了 URI 地址，或者在浏览的时候点击了一个超链接。无论何种方式，请求最终都会发送给服务器，比如我们以 *http://www.apache.org/index.html* 作为示例。

浏览器负责查找服务器主机，并与之建立一条双向网络连接。实现这一点要依赖于 Internet 上的诸多协议和域名服务器系统。一旦连接建立起来，浏览器便按照 HTTP 协议发出请求。在本示例中，用户访问的是 *http://www.apache.org* 主机上的 index.html，因此它向服务器端发出如下的 ASCII 字符串：

```
GET/index.html HTTP/1.0
```

这些字符串经过网上传输，被服务器接受并保存到内存中，图 1-10 演示了客户端与 WWW 服务器连接请求的发送。

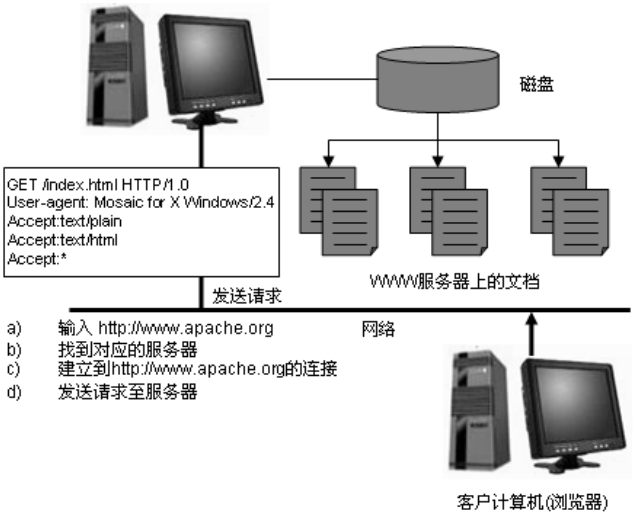

图 1-10 客户端和服务器的连接请求发送

接受客户请求

WWW 服务器对请求按照 HTTP 协议进行解码来确定进一步的动作，涉及的内容包括三方面：方法（GET）、客户端所要请求的文档（/index.html），以及浏览器使用的协议（HTTP/1.0）。其中方法告诉服务器应该完成的动作，GET 方法的含义很显然是：服务器应定位、读取文件并将它返回给客户端。

WWW 服务器现在已经知道它应该找到/index.html，并使用 HTTP/1.0 协议将内容返回给客户。信息是经过与请求到来时相同的连接发出的，所以服务器不需要重新建立与客户端的连接。

读取其他信息

WWW 服务器根据需要去读取请求的其他部分。在 HTTP/1.0 下，客户还应该给服务器提供关于它的一些信息。元信息可用来描述浏览器及其能力，以使服务器根据它确定如何返回应答。在本例中，还有如下请求信息：

```
User-agent:Mosaic for X Windows/2.4
Accept: text/plain
Accept: text/html
Accept: image/*
```

这些内容表明浏览器是 NCSA Mosaic，它可以显示文本及任意图像。

完成请求的其余动作

如果没有任何错误发生，WWW 服务器将执行请求所要求的动作。要获取（GET）一个文档，WWW 服务器在其文档树中搜索请求的文件 index.html，这是由服务器上作为操作系统一部分的文件完成的。如果文件能找到并可正常读取，服务器则将其返回给客户端，如图 1-11 所示。

如果成功，文件将被发送出去。首先，WWW 服务器将发送一个响应码及一些描述信息。既然文件已经找到，响应码为 200，表示一切正常（OK），文档随后发出。由于发送的是 HTML 文档 ，所以 Content-type 为 text/html。文档长度为 1 066 个字节，同时 Content-length 为 1 066。服务器软件的标识及文档的时间属性也被包含在对应的头域中，如下所示：

```
HTTP/1.0 200 Document follows
Server:Apache/1.3.6
Date:Thu,20 Jul.1998 22:00:00 GMT
Content-type:text/html
Content-length:1066
Last-modified:Thu,20 Jul,1998 22:00:00 GMT
```

在给出这些描述信息之后，WWW 服务器便从磁盘上读取该 index.html 文件，并将其输出到网络中。

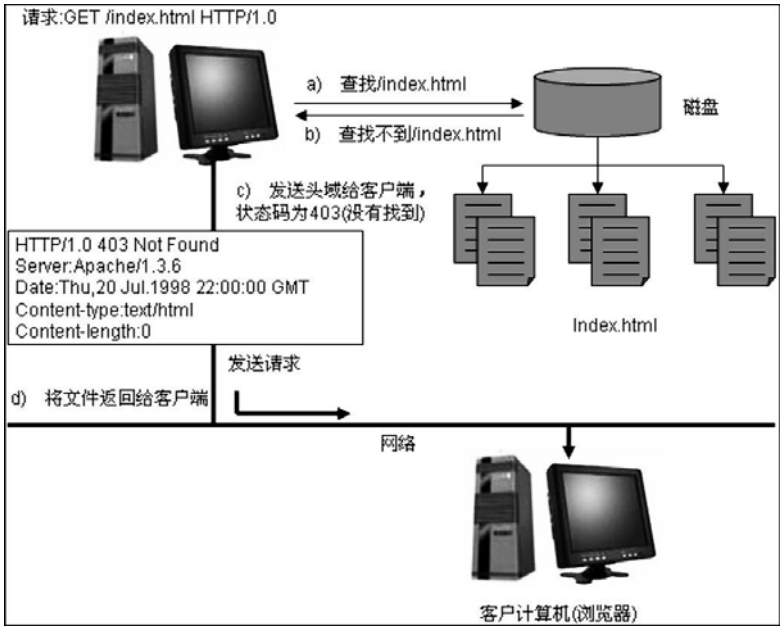

图 1-11 客户端和服务器的连接请求发送

如果处理失败，WWW 服务器将返回错误提示信息。通常最主要的错误是请求的文件在 WWW 服务器中找不到，或者找到无法读取，那么此时返回不同于 200 的其余错误码，比如 403，意味着请求的文件不存在，这样响应信息就变化为如下所示：

```
HTTP/1.0 403 Not Found
Server:Apache/1.3.6
```

```
Date:Thu,20 Jul.1998 22:00:00 GMT
Content-type:text/html
Content-length:0
```

关闭文件和网络连接，结束会话

在文件已经发出或错误指示已经发出后，WWW 服务器将结束整个会话，它关闭打开的被请求的文件，关闭网络端口，从而结束网络延迟。有关的其他工作则是由客户端来完成的，包括接收数据，并将这些数据以可读的方式布局在浏览器中。这些都与服务器无关。

1.3 Apache 功能

作为一个标准的 HTTP 服务器，Apache 同时支持 HTTP0.9、HTTP/1.0 及 HTTP/1.1 协议，实现了协议中规定的绝大多数内容。同时还实现了协议中并没有规定的内容，比如虚拟主机。

本节我们概要地介绍 Apache 中提供的各种基本功能。

1.3.1 虚拟主机

虚拟主机（Virtual Host）是指在一个机器上运行多个 Web 站点的机制（比如：*www.company1.com* 和 *www.company2.com*）。虚拟主机的实现包括以下三种方式。

（1）Web 服务器中配备多个 IP 地址，并且每一个逻辑 Web 服务器使用一个 IP 地址。这种虚拟主机的实现技术被称为“基于 IP”，这是最简单的虚拟主机的实现机制，但是这种机制存在一些问题，比如扩展性的问题。一台机器所能存在的物理 IP 地址总是有限的，因此对于一个专门的 ISP 而言，如果要提供大量的虚拟主机，则会存在相当大的困难。另外一个存在的问题就是 IP 地址的有限性，目前 Web 站点的数目远远超过 IP 地址的数目，因此，以 IP 地址区分虚拟主机，则会使 Web 站点的发展受到限制。

（2）Web 服务器只有一个 IP 地址，不同的 Web 服务器使用不同的端口进行侦听。因此这种服务器的请求 URI 中必须明确地给出端口，而不能使用默认的 Web 端口 80，比如 *http://127.0.0.1:8900*。这种虚拟主机的实现技术可称为“基于端口”。这种策略存在的问题是用户必须显式给出请求的端口，这对大部分用户来说显然是不太方便的。如果忘记输入端口号或输入一个错误的端口号，则会使用错误的虚拟主机。

（3）Web 服务器只有一个 IP 地址，同时多个域名被映射到该 IP 地址上。所有的 Web 服务器侦听同一个端口。服务器通过 HTTP 请求头中的 HOST 域对请求进行区分。对于 HTTP 1.1 协议而言，该域是必须具备的，而低于 HTTP 1.1 的协议则未必如此。因此从这个意义上说，只有 HTTP 1.1 协议才可以支持这种基于“HOST 域”的协议。

Apache 中支持上面三个方式的虚拟主机，而且通过 mod_vhost_alias 模块，可以使得类似的虚拟主机配置起来非常容易，减轻了管理员的负担。

1.3.2 内容协商

Apache 中对于同一个文档可能会保存多个不同的版本，比如英语版、法语版、日语版及中文版，等等。不同的用户可能需要不同的版本，比如中国的用户可能只关心中文版，如果没有中文版，也可以考虑英文版；而日本的用户可能更希望安装日文版。语言的不同是一方面，另外还可能文档的格式也不一样。最简单的例子就是 Apache 安装后显示的标准页面。

那么，我们怎么选择最适合给定用户的版本呢？有两种办法可以进行这种客户端和服务器端资源的协商：服务器端驱动的内容协商和客户端驱动的内容协商。这两种协商是相互正交的，因此它们可以单独或混合使用。混合使用中有一种被称为“透明的协商方法”，可在缓存使用由初始服务器提供的代理驱动协商信息，为后续的请求提供服务器驱动协商发生时，使用该方法。

服务器端驱动的内容协商

对于服务器端驱动的内容协商，毫无疑问，需要发送到客户端的文档版本由服务器端决定。通过使用“Accept”请求域字段，客户端会提供一系列的它能够接受处理的格式列表，根据这个请求域，服务器会选择最适合客户端的内容。服务器的选择可能基于语言、内容编码、请求消息中特殊报文字段的内容及隶属于请求的其他信息，比如网络 IP 地址等。

服务器端驱动的协商在下面的情况比较有利：从可用表示中选择的算法很难用于描述用户代理，或者服务器向和第一个应答一起把它的“最佳猜测”发送给客户端。如果最佳猜测对于用户来说已经足够好，就可以避免一个后续请求交互时间的延迟。为了改善服务器的猜测，客户端可以包含 Accept、Accept-Language、Accept-Encoding。

服务器端驱动的协商具有以下几个缺点。

（1）服务器不可能准确判断出对于一个给定的用户什么是“最佳”的协商，因为这需要全面了解客户端的能力和对应答的使用。比如用户是希望在显示器上看到，还是希望将其打印到纸上阅读。

（2）客户端在一个请求中描述自己能力的效率很低，另一方面，如果描述能力很强，又会对用户的隐私造成泄漏。

（3）服务器端驱动的协商使得初始服务器的实现及对请求生成应答的算法变得复杂。

（4）服务器端驱动的协商可能会限制一个公共缓存的能力，使其对多个用户的请求使用同样的应答。

Apache 支持 HTTP/1.1 规范中定义的“服务器驱动”的内容协商，可以完全支持 Accept、Accept-Language、Accept-Charset、Accept-Encoding 请求头，这些是 RFC2295 和 RFC2296 中定义的实验协商协议，但是不支持这些 RFC 中定义的“功能协商”。

客户端驱动的内容协商

客户端驱动的协商由浏览器端完成，它不属于 HTTP 服务器的一部分，我们不做过多的讨论。

1.3.3 持续连接

在 HTTP/1.0 版本中，对于每一个单一的 HTTP 请求，客户端和服务器端之间都必须建立一次 TCP 连接。在 HTTP 刚开发出来的时候，HTML 文档通常只包含 HTML 文件，在这种情况下，对一个请求建立一次连接是很合适的。但是随着 Web 技术的发展，一个 Web 页面中会包含非常多的多媒体数据，比如图片、音频、视频、FLASH 等。据统计，对于一个稍微大型一点的 Web 站点的主索引页面，如果要将它的整个文档下载到客户端，至少需要 150 次不同的文件请求。因此，如果打开每一个文件请求，然后关闭，那么也就是说我们至少需要打开和关闭 TCP 连接 150 次，这显然不是一个明智的选择。一方面，打开和关闭的

不断重复会造成文档下载时间过长；另一方面，也会给服务器造成非常大的负载压力。

基于对上面问题的解决，持续连接的概念被引入，尽管这个概念并不是标准的 HTTP/1.1 协议中所含有的。所谓持续连接，就是某个连接在打开后不立即关闭，而是继续使用，后续的数据传输都基于该连接。因此，对于某一个 Web 页面来说，不管其中包含了视频还是图片等，都是基于该连接被传送的。为了使用持续连接，客户端和服务器端之间会使用“Connection:keep-alive”请求域；默认情况下连接就是持续连接，除非进行了特殊的指定。如果客户端和服务器端的某一方不愿意使用持续连接，它只需要设置“Connection:close”请求域，另一方一旦接收到“Connection:close”，就会在当前请求处理完毕后关闭当前请求。Apache 中提供了配置指令，允许限制同一连接上的处理请求的数目，以及处理超时的时间，一旦超过该处理时间，所有的连接都将被关闭。

1.3.4 缓存

HTTP 通常位于分布式信息系统中，在这些系统中，可以通过采用缓存应答的方式改善系统的性能。HTTP/1.1 协议中包含了大量的元素，尽可能地使缓存产生更好的效果。通过缓存，可以加快对客户端的响应速度。

HTTP/1.1 协议中缓存的设计目标就是在很多情况下降低发送请求的必要性，以及在很多其他情况下降低发送完整应答的必要性。前者就使你少了很多操作所需要的网络回合的数量——它使用过期机制来实现这一目的。后者减少了网络带宽需求——通过验证机制达到这一目的。

HTTP/1.1 协议中提供了一些非常有用的缓存字段。

- Expires

Expires 字段声明了一个网页或 URL 地址不再被浏览器缓存的时间，一旦超过了这个时间，浏览器都应该联系原始服务器。RFC 告诉我们：“由于推断的失效时间也许会降低语义透明度，应该被谨慎使用，同时我们鼓励原始服务器尽可能提供确切的失效时间”。

对于一般的纯静态页面，如 HTML、gif、jpg、css、js，默认安装的 Apache 服务器，不会在响应头添加这个字段。Firefox 浏览器接收到相应命令后，如果发现没有 Expires 字段，浏览器根据文件的类型和“Last-Modified”字段来推断出一个合适的失效时间，并存储在客户端。推测出的时间一般是接收到响应后的三天左右。

- Cache-Control

Cache-Control 字段中可以声明多种元素，例如 no-cache、must-revalidate、max-age=0 等。这些元素用来指明页面被缓存的最大时限，如何被缓存的，如何被转换到另一个不同的媒介，以及如何被存放在持久媒介中的。但是，任何一个 Cache-Control 指令都不能保证隐私性或数据的安全性。“private”和“no-store”指令可以为隐私性和安全性方面提供一些帮助，但

是它们并不能用于替代身份验证和加密。

Apache 的 mod_cern_meta 模块允许文件级 http 响应头的控制，同时它也可以配置 Cache-Control 头（或任何其他头）。响应头文件是放在原始目录的子目录中，根据原始文件名所命名的一个文件。具体用法请参阅 Apache 的官方网站。其中 Cache-Control:max-age 表示失效日期。如果没有启动 mod_cern_meta 模块，Apache 服务器会把 Expires 字段中的日期换算成以秒为单位的一个 delta 值，赋值给 max-age。如果启动 mod_cern_meta 模块，并且配置了 max-age 值，Apache 会用它覆盖 Expires 字段。同时，max-age 隐含了 Cache-Control: public。这样，浏览器接收到的 Cache-Control:max-age 和 Expires 值就是一致的。

如果失效日期 Cache-Control:max-ag 等于零或为负值，浏览器会在对应的缓存中把 Expires 设置为 1970-01-01 08:00:00。

- Last-Modified

Last-Modified 和 ETag 是条件请求（Conditional Request）相关的两个字段。如果一个缓存收到了针对一个页面的请求，它发送一个验证请求询问服务器页面是否已经更改，在 HTTP 头里面带上“ETag”和“If Modify Since”头。服务器根据这些信息判断是否更新了信息，如果没有更新，就返回 HTTP 304（NotModify）；如果更新了，就返回 HTTP 200 和更新的页面内容，并且携带新的“ETag”和“LastModified”。

使用这个机制，能够避免重复发送文件给浏览器，不过仍然会产生一个 HTTP 请求。

一般纯静态页面本身都会有 Last-Modified 信息，Apache 服务器会读取页面文件中的 Last-Modified 信息，并添加到 HTTP 响应头部。

对于动态页面，如果在页面内部没有通过函数强制加上 Last-Modified，例如 header（"Last-Modified: " . gmdate ("D, d M Y H:i:s") . " GMT"），Apache 服务器会把当前时间作为 Last-Modified 返回给浏览器。

无论是纯静态页面还是动态页面，Firefox 浏览器巧妙地按照接收到服务器响应的时间设置缓存页面的 Last-Modified，而不是按照 http 响应头中的 Last-Modified 字段。

- ETag

既然有了 Last-Modified，为什么还要用 ETag 字段呢？因为如果在一秒钟之内对一个文件进行两次更改，Last-Modified 就会不正确。因此，HTTP/1.1 利用 Entity Tag 头提供了更加严格的验证。

Apache 服务器在默认情况下，会对所有的静态、动态文件的响应头添加 ETag 字段。在 Apache 的 httpd.conf 文件中可以通过 FileETag 指令配置该选项。

FileETag 指令配置了当文档基于一个文件时用以创建 ETag（entity tag）响应头的文件的属性。在 Apache 1.3.22 及以前，ETag 的值是对文件的索引节（INode）、大小（Size）和最后修改时间（MTime）进行 Hash 后得到的。如果一个目录的配置包含了“FileETag INode

MTime Size”，而其中一个子目录包含了“FileETag –INode”，那么这个子目录的设置（并会被其下任何没有进行覆盖的子目录继承）将等价于“FileETag MTime Size”。

在多台负载平衡的服务器环境下，同一个文件会有不同的 ETag 或文件修改日期，浏览器每次都会重新下载。设置“FileETag None”可以使响应头不再包含 ETag 字段。

1.3.5 访问控制和安全

对于 Web 站点上存在的一些私有文件，我们必须确保这些文件在绝对的受控范围之内，比如 passwd。通过认证、授权和访问控制等一系列的安全措施，可以确保受控资料的安全性。

认证（Authentication）、授权（Authorization）及账户确认（Accounting）三者合起来称为 AAA 模块。

访问控制

AAA 安全措施的第一个措施就是访问控制。访问控制意味着服务器会基于用户不能控制的请求特性进行限制访问，比如客户端计算机的 IP 地址。Apache 中只有一个模块实现了访问控制。如果用户在访问控制的范围之外，那么它将无法访问服务器。

认证

认证意味着用户要具有它所声称的身份，与大多数认证一样，HTTP 认证也是通过提供用户名称和共享密码或口令实现的，然后服务器再根据所发送的口令与它所知道的口令进行对比。如果两者匹配，就可以认为你具有它所生成的身份。如果两者不匹配，Web 服务器就会向浏览器发送一条消息，要求重新提供正确的用户名称和口令。服务器会在用户数据库中查找用户 ID 和密码组合是否存在。如果找到，则意味着该用户是认证通过的。这些数据库可能是一个简单的文本文件、一个类似于 MySQL 或 Oracle 的关系数据库，或者是操作系统本身所提供的授权机制，也有可能是类似于 LDAP2、NIS3 或 NTLM4 的用户管理服务。

认证包括两种：基本认证和摘要认证。

授权

尽管认证（authentication）和授权（authorization）两者的关系非常接近，但是二者之间还是存在着很大的区别。授权通常发生在认证之后。一旦用户认证通过，则只能说明它具有合法的身份，但并不意味着它就具有对特定资源访问的权限。对于服务器中的一些特殊的资源，管理员通常会严格限制访问。Apache 通过解析全局及本地的配置文件.htaccess 来决定用户的

授权身份。如果用户正在请求自己没有访问权限的页面，就可能只通过认证阶段而通不过授权阶段。从用户角度来看，不能获得授权类似于不能获得认证：浏览器都会要求他们再次输入用户名和密码。

对资源的访问可能限制在特定的域，浏览器所在机器的网络、地址或特定的用户及用户组。Apache 通过解析全局的及局部配置文件来决定用户对特定资源的访问权限。如果管理员允许，Web 内容提供者还可以通过本地配置文件.htaccess 限制对他们的文档的访问。

HTTP 服务器中认证和授权的流程如图 1-12 所示。

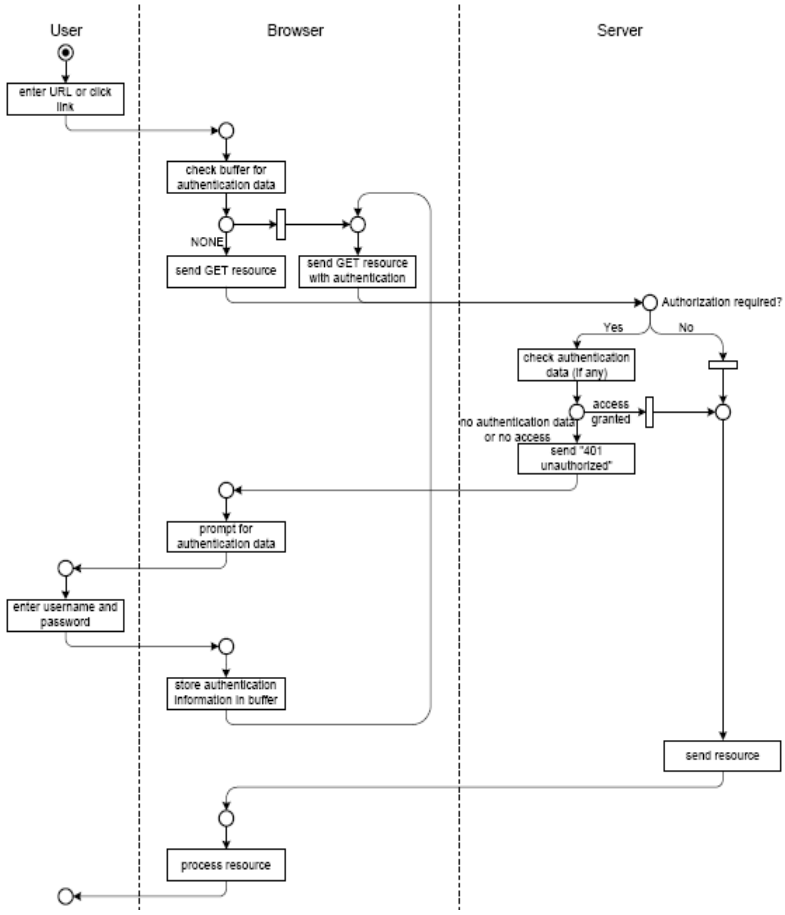

图 1-12 HTTP 服务器中认证授权流程

1.3.6 动态内容生成

一个 Web 服务器最简单的功能就是将存放于服务器上的静态的 HTML 文件发送给客户端。如果 Web 提供者须修改返回的 HTML，那么它必须手工更改这些文件。如果页面的数

目很少，那么这种工作量容易被忽略，但是对于一个大的 Web 站点而言，如果有成千上万的页面，那么显然不合适。随着 Internet 的发展，以及多媒体页面的增长，生成动态网页的需求也随之增长。基于 Web 的应用从最简单的个人地址本、留言本到复杂的在线银行、在线交易等，这些应用都必须要求 Web 服务器提供动态内容生成的功能。

Apache 中提供了服务器端脚本的功能用以动态地生成返回给客户端的 HTML 页面。允许服务器生成动态内容的一个简单方法就是使用服务器端脚本。允许服务器产生动态内容的第一个技术就是使用 CGI。为了使用 CGI 脚本，Web 服务器须执行一个外部应用程序，该程序可以解释脚本代码并且返回执行输出后的 HTML 至 Web 服务器，而 Web 服务器接收到 HTML 会继续将其转发给客户端。

另外，服务器还可以使用额外的模块来支持脚本语言，比如 mod_perl。当请求被处理的时候，脚本语言将在服务器的上下文环境中解释它。脚本模块为脚本语言提供了执行环境，另一方面，脚本模块中还必须提供 API 函数以便允许脚本访问脚本之外的数据，只有这样它才能接受客户端的数据。

在 Apache 中，每一个脚本语言模块通常都会注册一个单独的 MIME 类型，同时定义一个文件扩展，然后当该扩展的文件被请求的时候，对应的内容处理器将被触发调用。一般来说，服务器端的脚本分为两类：嵌入在 HTML 中的脚本和完全独立生成 HTML 的脚本。

支持嵌入脚本的 HTML 文件

对于这种情况，脚本是被嵌入在 Web 服务器上的 HTML 文档中的，这种情况大家最熟悉的就是 asp、php 等文件。不管哪一种脚本，它们通常都会被包含在一个特定的标签中，这些标签将被脚本引擎模块所识别。脚本被执行后，这些脚本将被输出为 HTML 文本，并将原有的脚本替换。脚本生成的数据可能基于外部数据源，比如数据库，也有可能是从客户端接收的数据。

服务器端包含功能（**S**erver-**S**ide **I**ncludes，SSI）是 Apache 本身所支持的嵌入脚本。SSI 中允许一些最基本的指令，比如对变量赋值、访问系统变量、处理基本的运算甚至执行一些系统命令。这些命令的数据将成为 Web 页面的一部分。

SSI 最通常的作用就是将其余的文件包含到当前 HTML 文档中。因此通常 SSI 会用来对 CGI 页面进行装饰，比如增加页面头和页面尾注，这样可以减少编写 CGI 页面的工作量。

通过在注释标签中使用特殊的指令，SSI 指令可以被直接包含在 HTML 页面中。浏览器总是会忽略这些注释，而服务器会对这些注释进行处理，然后对其进行内容包含，通常类似于：

```
<!--#set var="name" value="Rich" -->
```

完全由程序生成的 HTML 文档

对于一些复杂的 Web 应用程序而言，直接在 HTML 中嵌入脚本并不是一个最终的完美的解决方案。这些应用程序会完全生成所需要的 HTML 文档。

如果脚本中不包含静态 HTML 代码，那么无疑会减轻服务器的压力，因为服务器不需要解析整个脚本文档，并将其中的脚本指令提取出来。另外，纯粹的脚本语言执行要比 HTML 中嵌入脚本速度快得多。一些脚本语言会进行编译，所以速度也会快于那些解释性的脚本语言。

纯脚本生成 HTML 的例子最主要的就是 CGI 程序和 Java Servlets 两种。CGI 程序通常是 C、C++或 Perl 等程序，Java Servlets 则是使用 Java 程序编写的。这两种都是编译型脚本语言。