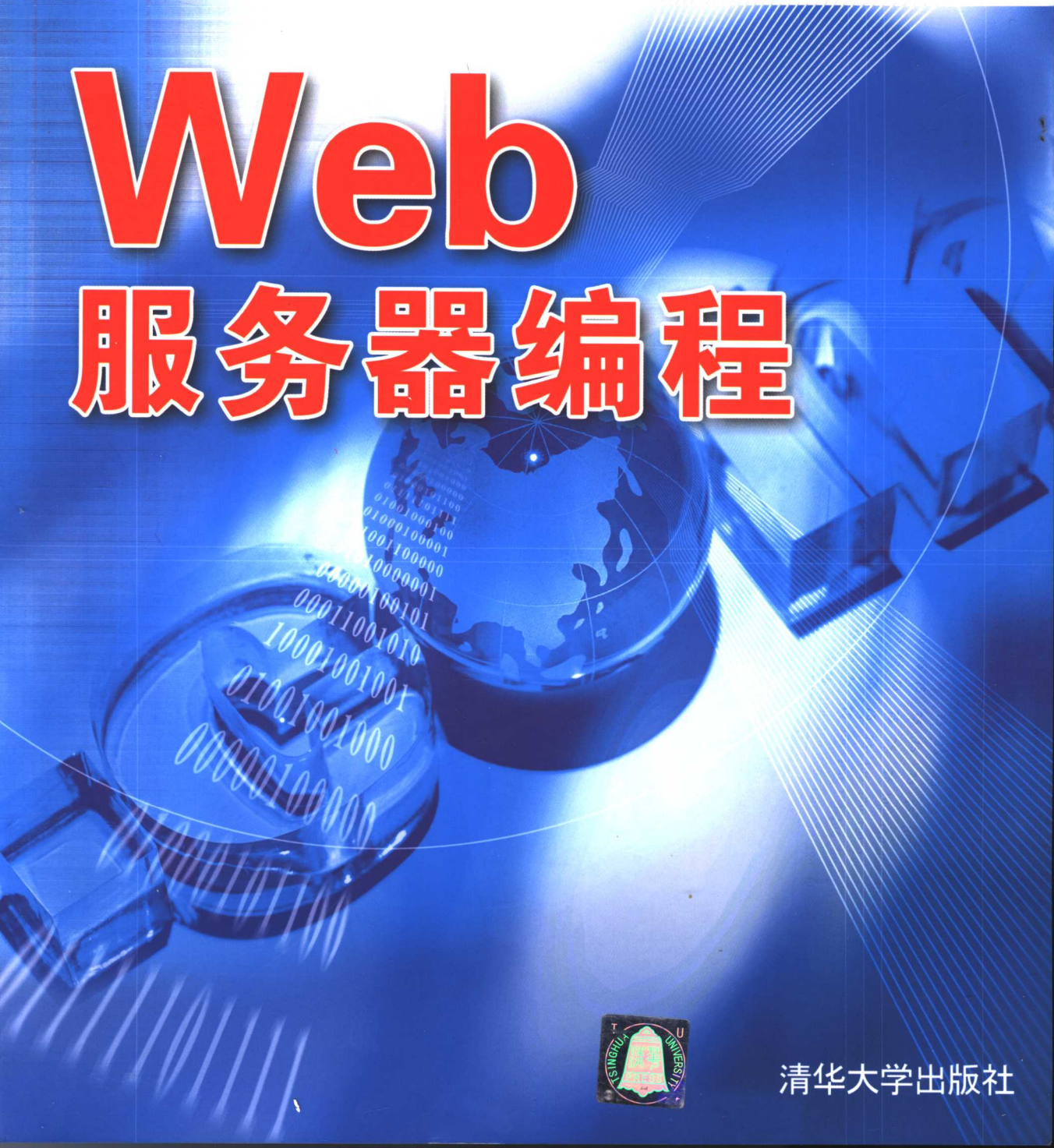
WILEY

Neil Gray 著 汪青青 卢祖英 等 译

Web 服务器编程



清华大学出版社

Web 服务器编程

Neil Gray 著

汪青青 卢祖英 等译

清华大学出版社

北京

Neil Gray
Web Server Programming
EISBN: 0470850973

图书在版编目 (CIP) 数据

Web 服务器编程 / 格雷（Gray, n.）著；汪青青，卢祖英等译. —北京：清华大学出版社，2004.9
书名原为：Web Server Programming
ISBN 7-302-09260-5

Ⅰ. W… Ⅱ.①格… ②汪… ③卢… Ⅲ. 网络服务器-应用软件-程序设计 Ⅳ. TP393.09

中国版本图书馆 CIP 数据核字（2004）第 084534 号

出 版 者：清华大学出版社 **地 址**：北京清华大学学研大厦
http://www.tup.com.cn **邮 编**：100084
社总机：010-62770175 **客户服务**：010-62776969
责任编辑：常晓波
封面设计：立日新
印 装 者：清华大学印刷厂
发 行 者：新华书店总店北京发行所
开 本：185×230 **印张**：35.75 **字数**：799 千字
版 次：2004 年 9 月第 1 版 2004 年 9 月第 1 次印刷
书 号：ISBN 7-302-09260-5/TP・6500
印 数：1~4000
定 价：56.00 元

前　言

本书的目的在于让你理解网站技术，如果将要从事 Web 应用程序的开发工作，那么这些技术对你来说会很有帮助。本书假定读者已具有以下背景知识：

- 基本熟悉 Internet，会使用浏览器，理解 Web 服务器在返回静态和动态网页时的作用。
- 熟练掌握 HTML。
- 了解客户端脚本语言（如 JavaScript），不过无须拥有有关技术的编程经验。
- 拥有 Java 和 C++的编程经验，大约相当于每门语言已进修了一个学期的课程（C++在本书中使用得不多。只在讨论更常见的使用 Perl 脚本语言实现 CGI 之前，在展示 CGI 系统结构的例子中使用了 C++）。
- 基本了解网络通信端口和套接字方面的知识（如可使用 java.net.Socket 和 java.net.ServerSocket 类完成一道习题）。
- 拥有使用关系数据库和 SQL 查询语句的经验。
- 至少尝试过使用 Java 的 JDBC 包从保存在关系数据库中的简单数据表中检索数据，或修改其中的数据。

本书共分 11 章，分别介绍如下：

第 1 章：简介　该章回顾 Web 服务器的基本知识，同时复习 Web 服务器是如何处理静态和动态页面请求的你应该很熟悉这些知识了。本章对 HTML、通用网关接口（CGI）和 HTTP 协议都会做简要描述。还会举例介绍一个 CGI 程序，它是用 C++编写的，将展示如何处理客户端请求（附录 A 有 HTML 和客户端 JavaScript 的简要介绍）。

第 2 章：HTTP　该章更加详细地介绍 HTTP 协议。需要基本理解诸如授权和内容协商等知识。

第 3 章：Apache　该章介绍网站管理员在服务器运行方面的职责。以 Apache 服务器作为例子进行介绍。你将负责设置并运行这样的服务器。需要理解如何管理文件的可访问性，以及对 CGI 程序的授权。

第 4 章：IP 和 DNS　该章综述域名服务（DNS）。DNS 系统把人们易于理解的域名映射为底层网络协议所需的 IP 地址。DNS 系统是成功的“开放源代码的”软件开发实例，并且被作为端对端（peer-to-peer）处理的范例。机器名称和机器地址这样的数据对任何中枢机构来说，管理起来都很庞大，而且变化也太快了。映射系统依靠整个名称空间的管理程序子部分来管理它们自有的名称服务器以及协作支持整个命名系统。管理公司的 DNS 系统未必是你的首要任务，但你有可能随时都要负责该系统，对 DNS 有个基本理解对于作为网站管理员的你来说就是很基本的要求了。

第5章：Perl 该章涵盖了Perl编程的基础知识，介绍了数据库接口模块（DBI），还列举了几个用Perl作为CGI脚本语言的例子。尽管Perl语言依然应用广泛，但多数Web站点都开始退出使用Perl-CGI脚本技术了。然而，Perl语言最初的应用——“提取和报告（extracfion and reporting）”语言——仍旧占有重要的位置。虽然站点有可能使用其他的服务器技术，但是你会发现在很多分析和报告任务中，你仍旧依赖于Perl。

第6章：PHP4 PHP系统容易部署，而且PHP脚本也容易编写。当帮助公司发掘交互式网站的潜力时，就应当使用该技术。对于大多数小型Web系统的实现来说，PHP是最合适的语言（除非你有约定，必须使用Microsoft专有的ASP脚本技术）。该章的例子展示了Web服务大量典型问题的解决方案，包括隐藏数据域的表单序列的使用、文件上传机制、图形输出页面以及cookie的使用（附录B有对Microsoft的ASP脚本技术的简要介绍）。

第7章：Java Servlet 现在转向讨论Java技术，首先看看servlet。实际上，和基于类似PHP技术的系统比较起来，Java系统部署很复杂而且编写起来也更困难。然而，Java解决方案具有固有的良好结构，而且更加规范化。Web应用程序的尺度有一个阈值；一旦跨越了这个阈值，最好采用Java解决方案。

第8章：JSP 实际上，这些技术并没有哪一项与servlet有着截然不同的差别。它们只是在某些问题上的进一步细化和分离。在正确实现Java Server Pages（JSP）系统过程中，你会发现被编写用于处理接受客户端请求的基本逻辑的各个servlet、负责处理诸如和数据库通信以及实现事务逻辑这样的任务的各种bean对象，以及通过在bean中传递的数据来生成最终页面的各个JSP。

第9章：XML 该章将简要介绍可扩展标记语言（eXtensible Markup Language，XML）以及相关技术（比如XSLT），然后列举几个应用的例子。这些例子基本上都是基于Java的Web应用程序，其中包括组合使用JSP和XSLT，这个例子可以为基于HTML的Web浏览器生成输出，或者为使用无线标记语言（Wireless Markup Language，WML）的WAP电话生成输出。

第10章：企业级Java 该章是企业级Java的一个概要描述，并介绍了企业级JavaBean。该章的目的仅仅是让你理解EJB技术是如何扩展前面章节中介绍的其他Java服务的，而且该章并不讲授如何编写EJB（仅仅讲述如何编写EJB就需要至少450页的篇幅）。了解Java RMI或其他分布式对象技术将有助于进一步理解EJB（附录C对Microsoft的.NET技术做了介绍。一方面，.NET技术为大型Web应用程序提供了一个和servlets/JSP/EJB平行的可选技术。另一方面，.NET为浏览器客户端和Web服务器组件之间的关联提供了全新的和更加一致的模型）。

第11章：未来的技术 该章介绍了一个最近的Web支持服务成功的例子（Akamai的Edge服务器）、其他领域（诸如端对端计算）的一些推测以及“个人网上展示（Personal Internet Presence）”的可能性，还解释了有关SOAP、UDDI、WSDL方面的技术。

本书的素材都是基于“开放源代码”软件的，这些软件都可以通过互联网免费下载获

得。可以在自己基于 Windows 98/2000/XP 的 PC 上完成所有的工作——需要从 http://www.apache.org/上下载 Windows 版本的 Apache 和 Tomcat 服务器，从 http://www.activeperl.com/上下载 Perl，从 Apache 站点下载 PHP4，从 http://java.sun.com/上下载 Java 开发工具包和一些附加 javax 组件，从 Apache 站点下载 XML 组件，并且从 Sun 公司站点下载 J2EE 组件。使用 Microsoft 的 Access 数据库系统足够运行这些例子了（通过 ODBC 驱动程序），但是也可以下载像 MySQL 这样的数据库。这些站点也提供了适合于 Linux 和某些 Unix 平台的软件版本。

本书列举的大多数编程例子实际上都使用了 Unix（Solaris）版本的软件，大多数数据库的例子都是基于 Oracle 数据库。将这些例子迁移到个人 Windows 系统中实际上不需要对代码做多大的改动。之所以使用 Unix 作为例子，原因之一就是考虑到实际操作系统中很多文件许可问题和其他安全控制。

所有与 Web 相关的软件都提供了大量文档，阅读本书时应该结合相关系统的文档资料。当你把 Apache 系统解压缩后，会找到大量 HTML 文档以及一些指向基于 Web 的指南的链接，这些指南会说明如何配置服务器。Perl 的文档可能是通过 perldoc 阅读器打开的文本文件，也可能是 HTML 文件。无论什么格式，都有许多关于 Perl 各个方面的内容，并且都包含了实例。PHP 发布了上千页的手册（PDF 文件格式），里面包含一部语言参考手册以及大量介绍 PHP 可用的各种函数库的章节，并且所有章节都包含了范例。Java 的 servlet 类以及相关的组件都做成了标准的 Javadoc 类型的文档。Apache Tomcat 服务器用于运行 servlet，它提供了一套完全配置好的简单 servlet 和 JSP 的例子。Sun 的 EJB 工具包中含有一本 450 页厚的指南。因为这些源码资料唾手可得，所以许多标准的参考资料在本书中都没有介绍。例如，本书就没有赘述有关 Perl 和 PHP 的操作符优先级表的内容。

有些章节附带练习。这些练习有 3 种形式。实践练习涉及到如何实现实际的基于 Web 的客户端-服务器系统。实践练习有着冗长的规范说明书，设计、实现与测试估计得花 6～18 小时。简答题类似于考试题，它针对章节内所介绍概念的定义和解释提出问题。同时还有少量的探索练习。探索练习仅对进一步学习使用万维网（World Wide Web）上可利用资源提出建议，这种练习适合于以书面报告的形式完成。

Neil Gray

目　录

第1章 简 介

本章将简要介绍基于 Web 服务的基础知识。在 1994—1995 年间，Web 由一个纯粹的发布媒介转变为一个互动的电子商务媒介。HTML 在支持数据输入表单方面的扩展，HTTP 协议在支持安全性及其他特性上的扩展，以及 Web 服务器用来进行应用程序通信的通用网关接口（CGI，Common Gateway Interface）标准的引入，使得基于 Web 的客户端所提交的数据可以由 Web 服务器上的服务器应用程序进行处理。

这些扩展使 B2C 型 Web 商务变得可行。假定所有消费者都使用标准的 Web 浏览器，浏览器与服务器间的连接具有相当的安全性，Web 服务器具有标准应用程序间通信机制以查询并更新数据库，接受订单，报告交货计划等等。

标准的 Web 浏览器同样可以取代众多在商务中用于内部应用的客户端程序。自 20 世纪 80 年代末起，私有内部客户端-服务器应用程序广泛地运行在企业内部网络中。客户端对数据输入与显示以及部分或所有商务数据进行处理，而其他部分商务处理和数据库连接则由服务器执行。然而，这种系统通常存在很多问题：客户端程序包括“商务规则”，而这些规则经常要进行一些改变。若规则改变时，所有的客户端应用程序都要进行升级，而这种升级通常很麻烦。另外，开发小组通常要开发多种版本的客户端软件以适应不同的平台和操作系统。如果一个系统转换为一个基于 Web 的架构，那么这两个问题都可以解决。基于 Web 的系统是多层次的，浏览器客户端仅用来处理数据输入和显示，中间件组件处理商务规则，后端数据库处理其他内容。在中间件层次处理事务规则后，转换会更加容易实施，同时与数据库连接的安全性会提高。在客户端使用标准的 Web 浏览器，则无须再开发客户端应用程序。

随着基于 Web 的服务的标准化及其广泛应用，开发人员可以依赖于相对稳定的客户端组件，并聚焦于服务器端。其后的服务器技术开发就可以更容易地进行编程，更有效更完善地实现服务器端环境。现在出现了一系列服务器端技术，其中一些将在本书后续内容中介绍。在某种程度上，这些技术是具有竞争力的，但是每种技术都有其独特的应用领域，在这一领域中，采用它是最理想的。

本章第 1 节复习了客户端-服务器编程基础，客户端-服务器编程是在使用 TCP/IP 与 Berkeley 套接字应用程序编程接口的基础上进行的。这些内容在 Stevens 所著 *Advanced Programming in the Unix Environment* 一书和 *TCP/IP Illustrated* 一书（由 Addison-Wesley 出版）中 1~3 卷中有更详细的介绍。Sun 的 Java 指南站点简单演示了 Java 版本的网络程序，地址是 http://java.sun.com/docs/books/tutorial/networking/index.html。本章接下来的两节，介绍了

Web 基础知识。首先介绍静态 Web 资源，然后介绍诸如服务器端包含（SSI）以及最早支持动态生成网页的 CGI 技术。接下来的一节简要地介绍用于数据输入表单的 HTML 以及基本的 CGI 规则。另外，一些网站中也有可参考的补充信息，如 http://www.wdvl.com/。接下来用一个小的 C++应用程序来说明 CGI 编程（C++并不常用于实现 CGI 编程，在这里使用 C++是由于许多读者有 C 和 C++编程经验并且可以阅读并理解相关代码）。最后一节简单地介绍了客户端脚本语言 JavaScript。如果基于 Web 的数据输入表单设计良好的话则可以结合 JavaScript 检查用户输入内容，其作用只是校验所有域中是否都输入了数据，并校验数据有没有明显错误，从而消除会被服务器拒绝的表单数据。为简单起见，本书后面给出的例子中不包含 JavaScript 检查方面的代码，但应该设想，实际中无论哪种场合，网页中都应包含这样的检查代码。JavaScript 在 http://www.wdvl.com/有其技术文档，其他 Internet 网站上也有这些内容（附录 A 简要介绍了 HTML 网页布局编码和客户端 JavaScript）。

1.1 Internet 中的服务器

自 20 世纪 60 年代末的 ARPANET 起，客户端-服务器系统一直运行于网络中。在 20 世纪 80 年代初，所采用的机制或多或少都得到了标准化。随后多数客户端-服务器系统开始运行在不同的 Unix 操作系统中。网络通信在 Internet 的传输控制协议/ Internet 协议（TCP/IP）上进行标准化。最受欢迎的应用程序编程接口是 Berkeley“套接字（Socket）”库。当时占主流的服务器包括文件传输（FTP）服务器、支持远程登录的 telnet 服务器、邮件服务器、新闻服务器以及诸如 echo 那样的实用程序。在大多数情况中，服务器总是以后台守护进程运行在机器上。典型的服务器守护程序处在一种阻塞状态，以等待客户端初始连接并请求服务。

网际协议（IP）定义了一种在 Internet 中两台机器间得到数据格式数据包的机制。很多更详尽的协议均以 IP 为基础。若不想在机器之间发送数据包，但希望在运行于这些机器的特定进程之间交换数据，这种处理方式在分层协议方案中是下一层的任务，所使用的协议为 UDP（用户数据报协议）和 TCP 等。这两个协议均定义了以“端口（Port）”形式在进程中通信的端点。端口属于操作系统资源。简单来说，可认为它们包含了一个整数标识符，并且还关联了一些输入输出数据缓冲。一个进程可以请求一个端口。服务器程序使用一些“常用端口号”；它们通常请求同样的端口号（例如 FTP 后台程序在进行文件传输时总是请求端口号 21）。客户端只需在一段时期内使用一个由操作系统分配的任意一个端口。UDP 和 TCP 数据包的头信息中包括客户端和服务器的 IP 地址以及客户端和服务器进程的端口号。客户端组成第一个数据包，加入已确定的主机 IP 地址以及那些所需服务的常用端口号。它还会加入自身的临时端口号以及程序在机器上运行时机器的 IP 地址。这些数据使得服务器可以发送目标为客户端的响应数据包。

UDP 协议相对来讲是轻型的。它允许客户端组成一个数据报，并将其发送至服务器。通常，数据报会安全地抵达，随后服务器进行响应。程序员需要决定如何应对没有响应的情况，并完成相应的代码，以重新发送请求或放弃尝试连接通信。TCP 协议则更加复杂一些。它在客户端与服务器间建立一个可靠的双向数据流。TCP 库中的代码可处理通信中碰到的所有问题，比如丢包、超时、重复包以及流控制要求。大多数客户端-服务器系统都使用位于 IP 层之上 TCP 协议。

Berkeley 套接字 API 则允许程序将套接字绑定到端口上。一个正常的套接字就像一个连接到输入/输出文件的连接，该文件支持 Unix 读写操作以及 Unix 其他更特殊的控制选项。这种数据套接字通常封装在支持更高级 I/O 操作的类中。服务器会使用“服务器套接字”，这与普通的数据套接字不同。操作系统处理到达这些套接字的初始输入，将这些数据视为新客户端尝试建立通信的请求。客户端连接后，操作系统产生新的数据流套接字，并将其返回到服务器进程中。随后服务器可以通过此套接字从客户端读数据并向客户端发送响应。服务器进程使用的单个端口可以和多个套接字关联，这些就是“服务器套接字”，而各种不同的数据流套接字是为并发客户端创建的。操作系统将它们保持分开状态，每个套接字使用不同的 I/O 缓冲。

客户端程序的理想化结构如下所示：

```
Read user input with hostname for server and its well known port number
Convert hostname to IP address
Open a data stream socket connecting to the server (IP, port)
forever
  Read next command as input by user
  Compose request to server
  Write request to data stream socket
  If command was quit then break
  Read response from server
  Display response to client
Close data stream socket
Exit
```

也许有些简单，但是这些伪代码的结构说明了 telnet 客户端，ftp 客户端以及 http 客户端的基本结构。

服务器架构多种多样，最简单的是串口服务器：

```
Main()
  Create a socket ("server socket")
  Bind server socket to "well known port"
  "Listen" on server socket (activating it so that it can be used for
    client connections)
  forever do
    newDataSocket = accept(server socket, ...)
```

```
    handleClient(newDataSocket)
    close newDataSocket

handleClient(datastream)
  forever do
    command = read from datastream
    If(read-error or command==quit) then break
    Process command and generate response
    Write response to datastream
```

accept 系统调用可以阻止服务器进程，直到一个客户端与之连接。当操作系统完成产生与客户端新的 TCP 连接后，即可允许服务器进程继续执行并且返回一个新的数据流套接字作为 accept 调用的结果。服务器随后开始处理客户端命令，由此数据流套接字读取客户端输入并向此套接字写入响应。当客户端命令执行完毕时，数据流接口套接字关闭，同时服务器进程又向操作系统发出一个阻塞 accept 调用。

串口服务器并不常用。其最大的优点是很容易实现，但是串口服务器一次只能处理一个客户端，它们在处理能力上的局限性远超过其优点。操作系统在处理当前客户端时，让其他等待连接的客户端排队等候，但尽管具有这个队列特性，串口服务器架构仍旧很不实用。通常，用户需要同时支持多个并发客户端。

现在有很多种可能的架构使并发服务器可以处理多个客户端。一种方法是使用一个单线程程序来欺骗多个 I/O 连接。这种服务器比较有效，它们向操作系统发出较少的请求，并处理相当数量的客户端。遗憾的是，它们很难以一种正确和稳健的方式实现。20 世纪 90 年代初，线程开始变得很常见了，而现在线程库都已经标准化了。多线程服务器变得越来越常见。在这种架构下，一个线程处理主 accept 循环。这个线程通常在 accept 系统调用中阻塞。当 accept 调用返回新的数据流套接字时，控制线程创建一个新的工作线程以处理这个套接字上的通信。每个单独的工作线程都可以在读取用户命令时使用标准的阻塞读取操作，发送响应时使用标准的（潜在阻塞）写入操作。单独的线程无须中断整个服务器进程的运行就可完成阻塞。当一个客户端断开连接后，刚才用来处理客户端的线程可能会终结，或者置于缓冲池中，以后用于处理另外的客户端。

最初的 Unix 服务器均基于“分叉服务器（forking server）”架构，这种架构仍然是最常用的一种。此架构包括一个用 accept 系统调用来处理客户端连接的“总台（receptionist）”进程，以及处理每个并发客户端的独立服务器进程。线程服务器简单地在服务器进程中创建一个新的线程以处理一个新的客户端，“分叉服务器”产生一个新的进程（产生一个新的子进程的系统调用在 Unix 中称为 fork，故取名“分叉服务器”）。以下的伪代码粗略地描述了这种服务器。

```
Main()
  Create a socket ("server socket")
  Bind server socket to "well known port"
```

```
    "Listen" on server socket (activating it so that it can be used for
      client connections)
    forever do
      newDataSocket = accept(server socket, ...)
      fork a new process
      if(this process is the child process) {
        close child's copy of the accept server socket
        handleClient(newDataSocket)
        close newDataSocket
        exit
      }
      else {
        close parent's copy of newDataSocket
      }

handleClient(datastream)
  forever do
    command = read from datastream
    If(read-error or command==quit) then break
    Process command and generate response
    Write response to datastream
```

这种基本架构有很多变例。第 3 章中所述的 Apache 服务器工作时采用“预分叉”子进程的集合。这种安排可以加速对客户端的处理(“分叉”操作相对较慢，大约需要几毫秒)。同样也可以使用 inetd 后台进程来监控不同服务器的常用端口；当一个客户端连接一个特定的端口时，inetd 将加载一个新的进程，运行与此端口关联的服务器程序。系统管理员必须选择采用 inetd 后台进程，还是采用标准的“总台”后台进程。这个选择取决于诸如性能要求和资源需求那样的因素。

每个服务器程序都有其自身的命令指令系统或“应用程序协议”。这也许相当有限（例如 telnet 只有一些简单的控件，以及一个令客户端与服务器上 Unix 登录 shell 相联系的机制)。在其他情况下，如“文件传输协议（FTP)”，其中包含很多命令。ftp 具有登录、改变目录、列出文件及获取或提交文件等命令。每个 FTP 命令由一个关键字和一些参数来定义。每个命令都定义了很多可能的响应，响应的标题行有一个整数代号，表明命令是否成功，之后为数据行。

1.2　静态超文本服务

Tim Berners-Lee 最初的 Web 目标十分有限。目的就是为了更容易访问由 CERN 生成的大量文档——CERN 是研究次原子结构的物理学家运行点加速器的地处日内瓦的欧洲研究所。像任何其他的大型组织一样，CERN 也生成大量的报告——管理报告、关于加速器