第二章 关键技术分析与研究

2.1 Web 服务器介绍

2.1.1 Web 服务基础

时至今日，网络服务已经触及了世界的绝大部分角落，结伴而生的网络应用在当下也是随处可见。当你在使用电脑、手机或者其他终端设备时，总免不了聊聊微信、看看咨询、刷刷网剧亦或是玩游戏，在这一过程中你已身在网络世界中。而那些让你通过终端设备与外部世界联系起来的都是网络应用，是这些网络应用让你实现了足不出户，知尽天下之事、缓解思念之情、满足好奇之心。更加有趣的是，几乎所有的网络应用都是基于相同的客户端-服务器基础编程模型，它们有着相似的整体架构并且依赖着相同的基本编程接口。

那么什么是客户端-服务器编程模型呢？就像它的名字，这个模型由一个或多个客户端进程和一个服务器进程组成。客户端向服务器请求服务，而服务器则负责管理各种资源并根据客户端的请求通过对应的操作为客户端提供服务。这样一个简单的请求-响应事务由四步组成如图 2.1 所示，首先当客户端需要某项服务时，它向服务器发送一个请求；然后，服务器接收该请求，并采取对应方式操作资源；服务器操作完成后向客户端发送对应的响应并转而等待下一个请求；最后客户端接收响应并通过适当的方式处理该响应。

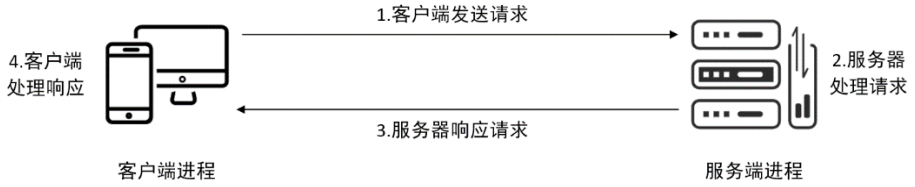

图 2.1 一个请求响应事务

客户端进程和服务器进程都是运行在主机上，而一台主机可以同时运行多个不同的客户端进程或者服务器进程。但是一般情况下客户端和服务器都是运行在不同的主机上的，并通过复杂的计算机网络实现通信。对于一个服务器进程而言，当它要和远程的客户端建立连接时，首先需要通过系统调用运用到套接字接口与客户端建立连接，然后根据特定传输协议处理请求并使用系统级 I/O 的系统调用操作资源，最后根据特定的传输协议将响应内容发送出去。这一过程对于被广泛使用的 Web 服务器而言如图 2.2 所示。

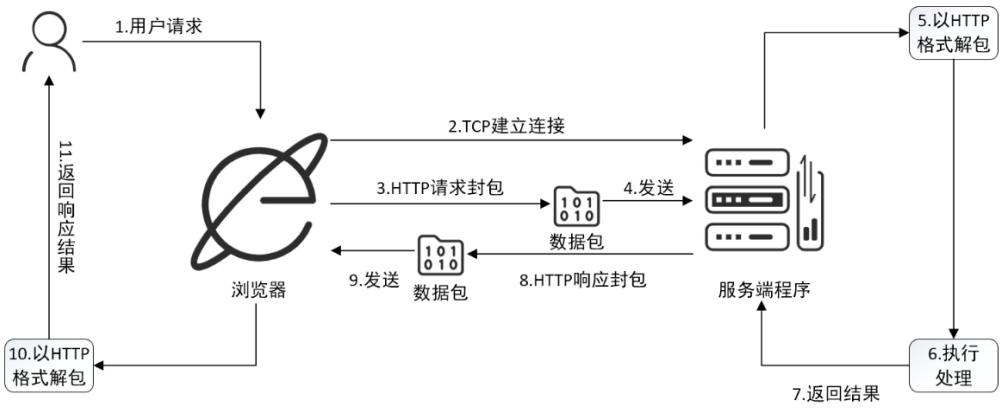

图 2.2 Web 服务器工作流程图

当用户在一个浏览器或其他 Web 客户端提出用户请求时，浏览器首先根据计算机网络中的 TCP/IP 协议使用套接字接口系统调用请求与 Web 服务器建立连接。当 Web 服务器通过同样的套接字接口函数组响应了连接之后，客户端进程将请求内容使用一个基于文本的应用级协议 HTTP（Hypertext Transfer Protocol，超文本传输协议）将数据封装并发送给 Web 服务器。Web 服务器接收到客户端请求后，首先使用 HTTP 协议对请求内容进行解释，然后根据请求内容使用系统调用对相应的资源进程操作来处理客户端的请求。Web 服务器完成处理之后将处理结果通过 HTTP 协议将响应内容发送给客户端，而客户端收到响应后同样需要解析响应内容然后将其呈现给用户。这就是一个完整的 Web 请求响应事务。

在一个 Web 请求响应事务中，Web 客户端和服务器的处理对象是 Web 内容，而 Web 内容通常是与 MIME（Multipurpose Internet Mail Extensions，多用途的网际邮件扩充协议）类型相关的字节序列。就像它的名字一样，MIME 诞生之初是为了在发送电子邮件时能够附加多媒体数据，让邮件应用能够根据文件类型处理多媒体数据。在最早的 HTTP 协议中，客户端和服务器间传送的数据是没有附加数据的类型信息的，所有从 Web 服务器传送的数据在 Web 客户端中都将这些数据解析为一种叫做 HTML（Hypertext Markup Language，超文本标记语言）的文档作为载体展示 Web 内容。HTML 是伴随着万维网诞生的，它作为一种排版网页中资源显示位置的标记结构语言，一个 HTML 文档中包含着大量的结构标记，而就是这些标记告诉了 Web 客户端应该如何显示响应内容中的各种文本图像等对象。另外，HTML 最大的特点在于它的超链接功能，一个页面中的超链接可以指向因特网中任何主机存储的内容，这一特点极大地方便的资源的共享。HTML 具有良好的简易性和可扩展性，同时它是与平台无关的能够很好的独立于各种类型的操作系统，这些优秀的特点让它在万维网中大行其道。

后来，随着多媒体内容的飞快发展，显然简单的文本内容难以满足网民。然而，Web 客户端在接收到 Web 内容之后，需要在插件系统中查找能够读取该 Web 内容的插件，如果无法识别 Web 内容中的各种文件类型，Web 客户端使用错误的插件进行文件读取时可能会导致系统崩溃。为了支持多媒体数据，HTTP 协议通过使用附加 MIME 数据类型在传输的文档前的方法来表示数据类型来解决这一问题。MIME 数据类型由类型和子类型两个部分组成，部分 MIME 类型如表 2.1 所示。

表 2.1 MIME 数据类型

大类型	MIME 类型	描述
text	text/html，text/plain	普通文本
application	application/pdf	二进制数据
image	image/gif，image/png，image/jpeg	二进制图像
audio	audio/wav	音频文件
video	video/mpeg	视频文件

Web 服务器不仅可以向客户端提供通过 MIME 数据类型标记的 HTML 文件、图像和二进制数据等被称为静态内容的磁盘文件，还可以提供动态内容服务。动态内容服务是指 Web 服务器不是直接返回通过 URL（Universal Resource Locator，通用资源定位符）标识的磁盘文件的内容，而是指根据可执行文件的 URL 并在该定位文件的 URL 后使用“？”字符添加用“&”字符隔开的程序参数来运行可执行文件，并将它产生的输出返回给 Web 客户端的过程。为了让 Web 服务器能沟通提供动态内容服务，需要为其增加一种能够运行外部程序的策略，而通用网关接口 CGI（Common Gateway Interface）就实现了这种策略。CGI 是一种标准，它规定了 Web 服务器与外部程序进行参数传递实现动态内容服务的一系列规范，关于基于 CGI 的动态内容服务将在本文的 2.4.1 节中再做详细介绍。随着技术的进步，目前除了 CGI 实现动态内容服务，微软在 1996 年推出的 ASP（Active Server Page， 动态服务器页

面）作为一种 HTML 页面的脚本语言以更加简易的方式实现动态网页。随后，Sun 公司也推出了名为 JSP（Java Server Page，Java 服务器页面）的动态网页技术标准，采用 Java 语言作为脚本描述语言，具有良好的可移植性。

在现实情况中，一个 Web 服务器需要面对多个客户端的请求，这是就需要使用并发的思路来提高服务器的性能。构造并发的 Web 服务器有三种基本的构造方法。首先，使用进程实现并发，每有一个客户端请求就创建一个子进程形成一个逻辑控制流由操作系统内核进行调度和维护。然后是基于线程的并发，主线程等待客户端请求，当 Web 服务器收到请求时，创建一个对等线程来处理该请求。最后是基于 I/O 多路复用的并发，根据客户端请求内容的读状态来模拟对应逻辑流的状态机，通过状态切换同时处理多个客户端请求。基于 I/O 多路复用的事件驱动 Web 服务器是现代诸多高性能服务器采用的并发方案，相比基于进程和线程的并发具有良好的性能优势。

2.1.2 现有的几种 Web 服务器及其特点

随着信息技术的进一步发展，越来越多的企业级 Web 应用系统部署在 Web 服务器上，这也极大的促进了 Web 相关技术的发展，同时出现了一大批性能优异的 Web 服务器。当下最为流行的几种 Web 服务器当属 Apache、Tomcat、Lighttpd 和 Nginx，下面将逐一介绍这几种 Web 服务器的特点。

（1）Apache

Apache【1】作为最流行的 Web 服务器是由非盈利组织 Apache Group 在 1995 年发布的最早版本。而 Apache 能够取得巨大的成功在很大程度上还是要归功于它采用了和 Linux 相似的源码开放策略。作为一个开源项目，所有组织和个人都有权利根据其运行平台和应用环境进行源码修改，当修改内容通过了 Apache Group 相关机构的审核和测试之后，Apache 服务器将集成修改部分变得更加完善。长年累月的积累，开源发展已有二十五年的 Apache 功能不断完善，一直是各种网络应用最为青睐的 Web 服务器，也始终保持着巨大的市场占有率。

Apache 具有良好的跨平台和可移植性，它具有一套基本的通用编程接口，例如输入输出和内存分配等基本功能都是与平台无关的，这也使得它几乎能够在所有的平台上运行，而且在部署 Apache 的过程中也不需要很复杂的开发和配置工作。另一方面，在最开始的开发过程中，Apache 的开发团队尽量使用了简单的体系结构和处理算法，这一做法极大的方便了后续加入的开源开发人员对它的理解，也降低了服务器维护工作者的工作复杂性，同时还提高了服务器的稳定性。Apache 采用的是模块化的体系结构即将请求处理元操作和具体操作分离，内核中实现处理请求的元操作例如 I/O、进程管理、内存分配和模块管理等基本操作，而模块则主要负责处理用户的实际请求。采用这样的体系结构使得 Apache 能够被不断的扩展，这也极大的丰富了对各类系统应用的支持。

但是，Apache 的 Web 请求处理机制为它带来一些缺陷，每当有一个 HTTP 请求时，Apache 都会创建一个子进程去处理该请求，从内核的角度讲每次创建一个进程的开销是很大的，需要给进程分配独立的内存空间。后来 Apache 也改进了请求处理机制，使用预线程的方式来减小开销，具体来说就是在服务器初始化的时候就创建多个进程，主进程接收 Web 客户端的请求，然后把请求处理的任务分配给其他线程进行具体的处理工作。这样的机制，极大的降低了开销，避免在多个客户端接入是产生巨大的内存消耗的 CPU 使用，但是相较高性能的其他 Web 服务器仍然存在性能上不足的问题。

（2）Tomcat

和 Apache 一样 Tomcat 服务器【2】也是一个免费开源的 Web 服务器，并且是由 Apache 和 Java 语言的提出者 Sun 公司以及一些其他组织和个人共同开发的。实际上 Tomcat 是作为

Apache 的一个在 Servlet 和 JSP 规范的扩展而存在的，但是在实际运行过程中它又是独立运行的。Tomcat 是使用 Java Servlet API 和 JSP 两项技术标准来实现的，所以它主要被使用在使用 Java 开发的 Web 应用中作为一个 Servlet 和 JSP 容器。当然，有 Java 语言创造者 Sun 公司参与开发的 Tomcat 在各个版本中都能够很好的支持最新的 Servlet 和 JSP 规范，所以在 Apache-Jakarta 规范的实行上，作为中小型系统或者学习和调试用途的 Web 服务器 Tomcat 比很多商业 Web 服务器要做得好得多。

Tomcat 除了是一个 Servlet 和 JSP 的容器，它也具备传统 Web 服务器的许多功能，例如它也可以处理 HTML 页面的请求。但是与 Apache 等其他一些 Web 服务器相比，Tomcat 处理 HTML 页面的能力还是差了很多，而且在并发访问用户量巨大的情况下它的高并发处理能力的弱势也会体现出来。在实际使用中很多为了克服 Tomcat 这些缺点，开发者可以采用一些配置手段将 Apache 或者一些其他 Web 服务器和 Tomcat 集成在一起使用，Apache 这类的 Web 服务器负责 HTML 页面和高并发等传统功能的处理，而 Tomcat 则运行 Servlet 和 JSP 页面。但是，总的来说 Tomcat 还是作为 Servlet 和 JSP 容器存在的，更加适合在并发量不是很大的中小型 Web 应用系统中使用，也非常适合和开发者在开发过程中调试 JSP 程序。

（3）Lighttpd

Lighttpd【3】也是一个开源的 Web 服务器，由德国的开发团队领衔开发，是开源的轻量级 Web 服务器中较为出色的。相较于经过时间磨练得非常成熟的 Apache，Lighttpd 的特点在于它响应 Web 请求的过程中并非采用 Apache 那种基于进程的并发处理策略，而是采用基于异步 I/O 的事件驱动技术。基于事件驱动处理并发请求的 Web 服务器在运行过程中一般只运行在单一进程的上下文中，这种策略使得各个请求处理逻辑流都能够访问该主进程的所以地址空间使得在逻辑流之间共享数据变得很容易。因此，事件驱动服务器 Lighttpd 在 CPU 占用率和内存开销等资源使用上消耗非常低，同时也极大的提高了该 Web 服务器的性能。另外，使用 Lighttpd 的 Web 应用一般都是使用 PHP 编写的，而 Lighttpd 的 FastCGI 模块在运行 PHP 内容时能够用很少的进程数量来响应巨大数量的 PHP 请求，这种方式也极大的提高了其性能。除了 FastCGI 模块外，Lighttpd 也具备很多其他丰富的模块，同时也支持 Auth、CGI、Alias、URL 重写和输出压缩等诸多重要的功能。

但是，Lighttpd 也具有一定的局限性，例如在为 Web 应用提供动态内容服务时，负责处理动态内容请求的 CGI 进程的效率和磁盘输入输出以及数据传输的带宽都将在很多程度上影响 Lighttpd 的性能。另一方面，相较于模块化体系结构的 Apache，Lighttpd 在功能的丰富程度上还是逊色很多的。总的来说，Lighttpd 是一个快速安全并且性能优秀的 Web 服务器，它良好的兼容性和灵活性使得它能够很好的适应很多高性能 Web 应用的需求，但是在功能扩展问题上丰富程度不够，在具体的特殊请求内容还需要特殊情况的考虑。

（4）Nginx

Nginx【4】是出自俄罗斯工程师 Igor Sysoev 之手，它是轻量级 Web 服务器最为典型的例子，最初被俄罗斯访问量巨大的搜索引擎 Rambler 所使用。所谓轻量级 Web 服务器是相对于 Apache 这种软件包大、软件耦合度高的 Web 服务器而言的，像 Nginx 这类 Web 服务器部署安装过程相较简单而且对硬件资源要求也比较低。虽然 Nginx 更小更简单，但是在一些特性和服务性能上具有明显的优势。

和 Apache 一样 Nginx 也使用了模块化体系结构，这种体系结构让它具备许多不一样的特性。HTTP 模块可以让 Nginx 作为普通的 Web 服务器使用，具备着 Web 服务器的一些基本功能，例如处理 HTML 等静态文件、提供动态内容服务以及简单的容错性和抗并发能力。邮件模块则可以让 Nginx 作为一个电子邮件代理服务器，基于 IMAP 或者 POP3 等相关协议实现邮件发送接收功能。更为特殊的是 Nginx 可以作为反向代理服务器提供反向代理服务，所谓反向代理是指代理服务器不是为局域网内的主机代理对互联网其他网络的访问，而是代

理局域网之外的主机访问本网络，也就是反向代理是将局域网内的服务器提供给其他网络的用户访问。在反向代理的基础上 Nginx 还可以通过特殊的调度算法实现客户端到服务器的负载均衡。在作为反向代理服务器时，Nginx 还具备本地缓存功能来增强其对高并发请求的处理能力。Nginx 本地缓存功能的实现是把 Web 客户端请求的 URL 和其他内容动作检索的 Key，使用 MD5 码进行编码并通过哈希结构存储到磁盘上，所以一般情况下 Nginx 可以支持任何类型的 URL。

基于事件驱动的 Web 服务器 Nginx 在服务性能上具有明显的优势。事件驱动是指在 Web 服务器完成与客户端的连接建立之后，客户端的读写事件进行检测。事件驱动技术是基于非阻塞 I/O 技术实现的，在迭代循环检测输入输出事件过程中，当磁盘 I/O 变得可读写时，对应的事件被唤醒继续处理请求。在 Linux 中 Nginx 是使用 Epoll 来实现事件驱动的，Epoll 是 Linux 内核中的一种可扩展的 I/O 事件处理机制，用来代替系统级 I/O 的 select 和 poll 进行 I/O 事件处理，而且 Epoll 的处理效率更高。采用事件驱动策略使得 Nginx 在高并发的环境中表现出优异的性能，同时使用非阻塞 I/O 机制可以有效的缓解请求处理进程在磁盘 I/O 上的阻塞延迟，从而提高了其响应效率。总的来说，Nginx 是一个高性能、高稳定性和丰富功能的轻量级 Web 服务器。

（5）上述四种 Web 服务器的对比

通过对几种 Web 服务器的详细介绍可以看出它们都有各自的长处和短处，分别适用在不同的使用环境中。Apache 作为 Web 服务器具有良好的安全性具有诸如数据加密和用户认证等功能，同时具有丰富的扩展功能。但是在响应速度和抗高并发能力上 Apache 稍逊一筹，另外其特殊的请求处理机制也导致很多额外的资源消耗。Tomcat 相比其他 Web 服务器应用面稍微要狭窄一些，主要针对 Servlet 和 JSP 容器使用。Lighttpd 和 Ngnix 都具有结构体系简单可移植性强的特点，具有非常高的用户请求响应速度，在高并发使用环境中也具有良好的稳定性，同时在对主机的计算资源和内存资源的消耗上具有明显优势；而且，Ngnix 强大的反向代理能力和本地缓存功能让它的特性得到充分的发挥。但是它们也同时在处理静态动态内容请求上比较弱势，没有 Apache 完备。表 2.2 对 Apache、Lighttpd 和 Ngnix 三种使用面比较广泛的服务器做出了对比。

表 2.2 三种 Web 服务器对比

指标	Apache	Lighttpd	Ngnix
响应性能	一般	好	非常好
稳定性	好	不好	好
安全性	非常好	一般	一般
抗高并发性	不好	好	非常好
静态内容处理能力	非常好	好	好
可移植性	好	非常好	非常好
灵活性	不好	好	非常好

2.2 网络通信

在 2.1.1 节的 Web 服务基础中已经提到 Web 服务器和客户端是通过套接字接口建立网络连接，并使用 HTTP 进行数据传输来实现远程的网络通信的。在本节将更加详细的介绍 Web 服务器和客户端是怎么通过套接字接口和 HTTP 协议实现网络通信的。

2.2.1 套接字接口

从计算机网络的底层来看，网络中的各个主机是通过各种线缆、集线器、网桥、路由器等硬件设备连接在一起的。然而，发展了几十年的互联网已经形成了十分复杂而且还在不断

变化的体系结构，为了协调结构组成各异的网络，一些基本的硬件和软件组织是必不可少的。现如今身处网络中的绝大部分主机都运行着实现 TCP/IP 协议的软件，TCP/IP 是一个协议簇，这个协议簇中的诸多协议规范了互联网中主机间的通信机制。其中，IP（Internet Protocol，互联网络协议）协议是计算机网络的网络层协议，作为底层协议的 IP 协议主要提供数据报的命名方法和传输机制，并将网络中的所有主机映射为一个 32 位无符号整数称为主机的 IP 地址，也称为相较于主机 MAC（Media Access Control Address，媒体存取控制地址）硬件地址的逻辑地址。而 TCP（Transmission Control Protocol，传输控制协议）和 UDP（User Data Protocol，用户数据协议）协议则是计算机网络中较为高层的传输层协议，UDP 协议是对 IP 协议的扩展是面向无连接的不可靠传输协议，在传输过程中不对数据报进行过的处理，不提供确认、排序、重发和流量控制等功能；而 TCP 协议则是面向连接的可靠传输协议，使用经典的“三次握手”建立连接、“四次握手”断开连接，在传输过程中是点对点的单向传播，并对报文段进行差错检测和按序接收，同时提供流量控制和拥塞控制。

在本文 2.1.1 节 Web 服务基础中提到客户端和服务器是运行在主机上的进程，而网络中的一台主机可以运行多个客户端进程或者多个服务器进程。显然，网络中的主机只有一个 IP 地址来标识自己显然不能够将彼此的一对进程点对点地连接起来，这时就要用到套接字。一个套接字是网络连接中的一个端点，而这个套接字的地址是由主机的 IP 和一个 16 位无符号整数的端口号组成的表示为 Socket=（IP 地址：端口号）。在客户端/服务器编程模型中，当一个客户端向服务器发起连接请求时，操作系统内核首先为该客户端进程分配一个临时端口号构成套接字地址来标识这个连接的一个端点。而在服务器进程这边，在它初始化时就已经被分配的特定的端口号并形成了一个监听套接字。这样一个连接也就可以由这两个套接字地址唯一标识了，并将这个连接用一个四元组来表示（IP 地址 1：端口号 1，IP 地址 2：端口号 2）构成了唯一的一个套接字对。

在 Web 服务器和客户端中通常是使用系统调用实现的套接字接口和系统级 I/O 来实现网络通信的，而系统级 I/O 是用在 Web 服务器和客户端建立连接之后对资源进行操作的过程这将在本文的 2.3.1 节中详细介绍。套接字接口是 TCP/IP 网络的 API 在操作系统内核中被实现为一组系统调用，服务器和客户端混合使用这一组系统调用来实现连接的建立，一个简易的基于套接字接口的网络应用示例如图 2.3 所示。

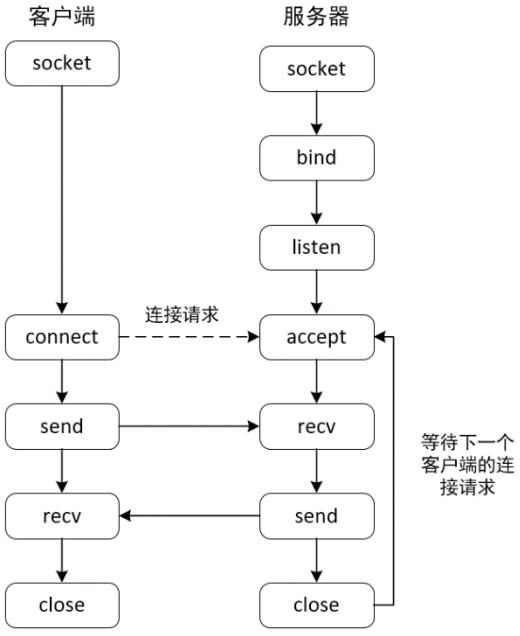

图 2.3 基于套接字接口的网络应用示例

这一基础的网络应用包括连接建立、数据传送和连接释放三个主要部分。连接建立部分，在服务器这边，首先调用 bind 函数将端口号和 IP 地址填入已创建的 socket 描述符中，然后调用 listen 函数把套接字设置为被动行式，形成监听套接字来接受客户端的服务请求，若收到客户端连接建立请求就调用 accept 函数接受请求；对于客户端在服务器创建套接字后就调用 connect 函数向服务器发送连接请求。数据传送部分，在完成 TCP 连接的基础上，客户端使用 send 函数发送请求内容，服务器使用 recv 函数接受该请求并使用 send 函数发送处理请求后形成的响应内容，而客户端则使用 recv 函数接收响应内容并处理。连接释放部分，客户端和服务器一旦结束使用套接字，就调用 close 释放连接和撤销套接字，对于服务器则是释放当前连接然后阻塞在 accept 函数的调用中来等待下一个客户端的连接请求。

下面将进一步介绍上述示例中的套接字接口函数，在此处将不详细讲述数据传送部分的 send 和 recv 函数，因为在此处它们是对系统级 I/O 中的系统调用的抽象，本文将在 2.3.1 节中详细介绍它们。

socket 函数，客户端和服务器用来创建套接字描述符的

函数。它有三个参数，domain 参数表示通信协议簇的类型，AF_INET 表示 ipv4 协议簇而 AF_INET6 则表示的是 ipv6 协议簇；type 则表示传输协议的类型，SOCK_STREAM 表示可靠传输 TCP 协议，SOCK_DGRAM 表示不可靠传输 UDP 协议；protocol 参数默认为 0。

bind 函数，服务器用来绑定其套接字地址和套接字描述符的函数。它的三个参数 sockfd 表示其套接字描述符，*addr 表示其套接字地址而 addrlen 则是该套接字地址结构的长度。

listen 函数，服务器用来将其主动套接字转化一个被动的监听套接字来接收客户端的连接请求。它有两个参数，分别是表示其套接字描述符的 sockfd 和表示正在排队等待建立连接的客户端请求数量。

accept 函数，服务器用来等待客户端连接请求的函数。它也有三个参数，listenfd 表示服务器的监听套接字描述符，而*addr 表示请求连接的客户端的套接字地址，同样 addrlen 表示套接字地址结构的长度。

connect 函数，客户端用来向服务器发送连接请求的函数。它也有三个参数，clientfd 表示客户端的套接字描述符，*addr 表示其套接字地址，addrlen 表示套接字地址结构的长度。和 accept 函数一样在为成功建立连接时都会处于阻塞状态等待连接成功。

close 函数，客户端用来关闭连接，服务器用来释放当前连接并转而等待下一个连接请求。它只有一个参数即为表示套接字描述符的 sockfd。

2.2.2 HTTP 协议

在 2.1.1 节 Web 服务基础中的图 2.2 可以看到整个 Web 服务器的工作流程中涉及最多的内容是 HTTP 协议的使用，事实上 Web 服务器的的核心就是 HTTP 协议。HTTP 协议定义 Web 客户端如何从 Web 服务器请求 Web 页面，以及服务器如何把 Web 页面传送给客户端的信息处理过程，而网络通信功能则是在上一节 2.2.1 中介绍的底层协议 TCP/IP 完成的。所以 HTTP 协议也是基于 TCP/IP 协议的应用层协议。

随着信息技术的发展，HTTP 协议也在不断更新迭代，共经历了 HTTP0.9、HTTP1.0 和 HTTP1.1 三个版本。在最早的 HTTP0.9 版本，HTTP 协议只能在服务器和客户端之间直接进行交互，而且只能处理文本类型的信息。1982 年万维网之父 Berners-Lee 提出了 HTTP1.0【5】，这个版本也是被最为广泛使用的 HTTP 协议版本【6】。在 HTTP0.9 的基础上，HTTP1.0 版本运行客户端和服务器之间由代理进行间接的交互，同时引入了 MIME 数据类型使得 Web 服务器不仅可以处理简单的 HTML 纯文本文件，还能够提供图像、音频、视频等多媒体内容服务，这极大丰富了网络服务。HTTP1.0 版本也是本文重点使用的 HTTP 协议版本，它虽然没有后来的 HTTP1.1 在稳定性、安全性和性能上的诸多优势，但它是一个足够经典的

HTTP 协议版本，对互联网的发展有着巨大的影响。HTTP1.1【7】版本是最新 HTTP 协议版本，在 HTTP1.0 的基础上增加了安全和缓冲等一些高级机制，其中有一个比较特殊的机制是它支持客户端和服务器在同一条持久连接上执行多个事务。

HTTP 协议的主要内容是其请求/响应模型，并基于 TCP/IP 协议建立的客户端服务器连接使用文本行传送内容。而一个请求响应过程又称为一个 HTTP 事务，它由 HTTP 请求和 HTTP 响应两部分构成。HTTP 请求是指客户端在向服务器发送请求提供服务的文本行，HTTP 响应则相反是服务器根据客户端向服务器发送的请求内容经过资源操作之后形成的响应结果的文本行。

（1）HTTP 请求

一个 HTTP 请求是由一个请求行、零个或多个请求报头和一个表示请求报头终止的空行组成的。而一个请求行则是由请求方法、URL 和 HTTP 版本三个部分组成的。HTTP 协议支持许多种不同的请求方法，例如 GET、HEAD、POST、PUT、DELETE 等，几种常用的请求方法用途如表 2.3 所示。请求行中的 URL 即为请求资源对象在服务器管理资源中的位置；HTTP 版本即为该 HTTP 请求遵循的 HTTP 协议版本。

表 2.3 HTTP 协议支持的几种常用请求方法

请求方法	主要用途
GET	根据请求的 URL 向服务器请求对应资源对象
HEAD	根据用户请求向服务器请求文件大小、修改时间等元信息
POST	客户端主动向服务器传送数据，例如提交 HTML 表单等
PUT	通过客户端创建或替换服务器管理的资源
DELETE	通过客户端删除服务器管理的资源

HTTP 请求的请求报头由请求报头字段和字段信息两部分组成，请求报头可以为服务器提供一些额外的信息，包括 Web 客户端的名称或类型、Web 客户端可理解的语言类型等一些非请求内容信息，一些常见的请求报头字段如表 2.4 所示。

表 2.4 HTTP 请求中几种常见请求报头字段

请求报头字段	字段含义
Host	指示了原始服务器的域名或 IP 地址
User-Agent	Web 客户端的名称或类型
Accept-Language	Web 客户端可以理解的语言类型
Accept-Encoding	Web 客户端可以理解的编码类型

（2）HTTP 响应

和 HTTP 请求相似，HTTP 响应也有响应行、响应报头和表示响应报头终止的空行三部分，除了这几个部分之外 HTTP 响应还多出了响应客户端请求的响应主体。响应行由 HTTP 版本、状态码和状态消息三个部分。和请求行一样，响应行中的 HTTP 版本也表示着该 HTTP 响应所遵循的 HTTP 版本；状态码是一个三位正整数用来表示服务器是否正确地响应客户端请求，状态信息则是对状态码的具体描述，一些常见的状态码和状态信息如表 2.5 所示

表 2.5 HTTP 响应中几种常见的状态码和状态信息

状态码	状态信息	具体含义
200	响应成功	HTTP 请求处理无误
400	错误请求	服务器无法解析请求内容
403	禁止访问	服务器对客户端所请求的内容无访问权限
404	未发现	服务器未找到客户端所请求的内容
501	未实现	服务器不支持该 HTTP 请求的请求方法
505	HTTP 版本不支持	服务器不支持该 HTTP 请求所遵循的 HTTP 版本

和请求报头相似，响应报头也是由响应报头字段和字段信息两部分组成的，为客户端提供了一些关于 HTTP 响应的附加信息，包括响应主体中内容的 MIME 数据类型和响应主体的字节大小等信息，一些常见的响应报头字段如表 2.6 所示。HTTP 响应的最后一部分响应主体则相对简单，包含着客户端请求的内容。

表 2.6 HTTP 响应中几种常见响应报头字段

响应报头字段	字段含义
Server	服务器的名称和版本
Date	HTTP 响应发送的时间
Content-Type	HTTP 响应主体中内容的 MIME 类型
Content-Length	HTTP 响应主体的字节大小

2.3 资源操作

在 2.2 节中已经详细介绍了 Web 服务器和客户端进行网络通信实现的基础，那么在客户端和 Web 服务器实现通信之后，服务器就要做它应该做的事情了，那就是为客户端提供 Web 服务。在前面已经介绍过 Web 服务器是通过对其管理的资源进行操作，并将操作结果发送给客户端来实现为其提供服务的。而系统级 I/O 和内存映射是 Web 服务器内核层面实现资源操作的基础，无论是在静态内容服务还是动态内容服务这两个技术点都是必要的。在 Web 服务器响应 HTTP 请求过程中需要使用系统级 I/O 来读取 HTTP 请求内容和写 HTTP 响应的部分内容，当服务器提供静态内容服务时需要使用内存映射的手段将客户端请求的资源对象拷贝到响应主体并发送出去；当服务器提供动态内容服务时需要涉及内存映射的内容来创建运行 CGI 程序的子进程，当然为了返回 CGI 程序的执行结果还需要用到系统级 I/O 的 I/O 重定向直接将结果返送给客户端。总之在 Web 服务器的资源操作过程中需要广泛的使用到系统级 I/O 和内存映射，下面将详细介绍它们的具体内容。

2.3.1 系统级 I/O

在计算机系统结构中，I/O 即输入/输出是主存和外设之间的数据交互过程，输入输出是对于计算机主存而言的，数据从外设复制到主存是输入，反之则为输出。本文的 Web 服务器是基于 Linux 系统来编写的，在 Linux 内核中系统级 I/O 是其引出的一个底层应用接口，称为 Unix I/O。在 Linux 操作系统中一切外部设备都被看作是文件，这样的策略统一了计算机输入输出的执行方式，系统的输入输出被简化成了针对文件的操作。而网络对于主机来说也是一个外设，所以 Web 服务器进行资源操作的过程也主要是通过对网络映射成的文件和主机存储系统中的文件的操作。

（1）Unix I/O

Unix I/O 主要包括创建或打开文件、读写文件和关闭文件三个主要的文件操作。

创建或打开文件，Web 服务器在资源操作过程中需要访问某个文件时，操作系统内核会为其返回一个文件描述符，在后续的操作过程中这个描述符将唯一标识这个文件。在内核中是通过 open 函数实现创建或打开文件，它有三个参数，filename 参数是指被请求访问文件的名字，在调用 open 函数时将根据这个参数生成一个对应的文件描述符并返回；flags 参数则指明了访问文件的方式，它可以是一个或者多个掩码来提供指示信息，一些常用的文件访问方式掩码如表 2.7 所示；mode 参数表示的是在 open 函数被调用创建新文件时的文件访问权限，和 flags 参数一样，mode 参数也可以由一个或者多个掩码组成来设定文件拥有者和其他用户对该文件的访问权限。

表 2.7 open 函数文件访问方式常见掩码

掩码	掩码含义
O_RDONLY	只读地访问文件
O_WRONLY	只写地访问文件
O_RDWR	可读可写地访问文件
O_CREAT	当访问文件不存在时，创建一个空文件
O_TRUNC	当访问文件存在时，截断该文件

表 2.8 open 函数文件访问权限常见掩码

掩码	掩码含义
S_IRUSR	文件拥有者可读
S_IWUSR	文件拥有者可写
S_IXUSR	文件拥有者可执行
S_IRGRP	文件拥有者所在用户组成员可读
S_IWGRP	文件拥有者所在用户组成员可写
S_IXGRP	文件拥有者所在用户组成员可执行
S_IROTH	文件对其他用户可读
S_IWOTH	文件对其他用户可写
S_IXOTH	文件对其他用户可执行

读写文件，Web 服务器在资源操作过程中对文件进行读写时，调用操作系统内核中实现的 read 和 write 函数，它们分别实现从文件复制指定字节数的内容到内存和从内存复制指定字节数的内容到文件。当文件被访问进行读写操作中出现文件或内存内容不足指定读写字节数时会触发 end-of-file（EOF）条件，这种情况下 read 和 write 函数会返回不足的字节数量作为 EOF 信号告知调用的进程。read 和 write 函数具有三个相同的参数分别是表示文件描述符的 fd、目标位置 buf 和读写内容字节大小 n。

关闭文件，当 Web 服务器完成了对指定文件的访问之后，它就调用 close 函数通知操作系统内核关闭这个文件，并由内核释放各种创建或打开文件中文件描述符等消耗的资源，当进程运行结束是内核会关闭所有它访问过的文件。

读取文件元信息，HTTP 请求方法中的 HEAD 就是向服务器请求文件元信息的，在操作系统内核则是通过 stat 函数来实现文件元信息的获取的。在内核中有一个记录文件元信息的数据结构 stat，它记录着文件权限、文件修改时间等元信息。stat 函数的两个参数分别是文件名和 stat 数据结构的成员名，并返回对应成员名记录的信息。

I/O 重定向，在 Web 服务器提供动态内容服务时，它代理客户端运行 CGI 程序就需要将输出结构重定向到客户端进程。在内核中 dup2 函数实现了这一功能，它的两个参数分别是描述符表项中表示旧文件描述符的 oldfd 和描述符表项中表示新文件描述符的 newfd，在进程调用 dup2 函数时使用 newfd 的内容覆盖 oldfd 使得该文件描述符的 I/O 对象改变。

（2）Robust I/O

在上文我们提到 Linux 内核实现的文件读写 read 和 write 函数是不处理文件中的不足值的，而在 Web 服务器这种网络应用的中会出现大量的文件存在不足值的情况，为了解决这一问题本文引入了 RIO（Robust I/O，健壮的输入输出），这个 I/O 包提供了自动处理文件上下文中出现不足值的问题，很好地优化了 Web 服务器对文件的读写。RIO 包提供了无缓冲输入输出函数和带缓冲输入函数两组不同的输入输出函数。

无缓冲输入输出函数包括 rio_readn 和 rio_writen 这一对函数，它们实现直接在内存和文件之间传送数据，参数和 Uinx I/O 中的 read 和 write 函数类似，但是在遇到 EOF 情况时，

rio_readn 函数只能返回一个不足值而 rio_writen 函数则不返回不足值，这使得在同一个文件中可以交错的调用这对函数。

带缓冲的输入函数，为了避免在特殊限定情况下读取文件是，需要反复调用 read 函数检测读取字节内容，RIO 包提供了一组带缓冲的输入函数，已缓冲区作为中间体读取内容，当缓冲区变空时才继续读取内容填满缓冲区。RIO 包中 rio_readinitb 函数实现一个空缓冲区的创建并且将这个空缓冲区和一个已打开的文件描述符建立联系。rio_readnb 和 rio_readlineb 两个函数则分别实现按字节读取和按行读取，这种方式提高了 HTTP 响应内容这种二进制数据的读取效率。

2.3.2 内存映射

Web 服务器在为客户端提供静态内容服务时，很多情况下被请求的资源对象很多，采用一般的系统级 I/O 需要反复调用导致读写效率低下。而采用内存映射的方式将一个磁盘上的文件直接和一个虚拟存储区域关联起来，这样文件的目标内容就被整块的放到了内存中，避免了反复的系统调用极大的提高了访问效率。

在 Linux 中使用 mmap 和 mumap 这一对函数实现用户级的内存映射 mmap 函数是用来创建新的虚拟内存区域，并使用对应的资源对象初始化这个虚拟内存区域。mmap 函数的参数比较多，它在创建一个虚拟内存区域的过程中，这个虚拟内存区从参数 start 指定的位置开始，将参数 fd 文件描述符所指定的文件从参数 offset 偏移量开始，对象长度为 length 字节大小的内容映射到这个新区域。参数 flags 表示的是被映射对象的类型，它可以是 MAP_ANON 标记的请求二进制零的匿名对象，也可以是 MAP_PRIVATE 标记的私有的且在写时复制的对象，也可以是 MAP_SHARED 标记的一个共享对象。和文件的访问权限类似，虚拟内存区域也有表示访问权限的 port 参数，一些常用的访问权限掩码如表 2.9 所示。mumap 函数则相对简单，它完成的功能是删除从虚拟地址 start 开始长度为 length 字节的虚拟内存区域。

表 2.9 mmap 函数虚拟内存区域访问权限常见掩码

掩码	掩码含义
PORT_EXEC	该虚拟内存区域内容可以有 CPU 执行指令组成
PORT_READ	该虚拟内存区域内容可读
PORT_WRITE	该虚拟内存区域内容可写
PORT_NONE	该虚拟内存区域内容不可被访问

内存映射不仅提高了 Web 服务器在提供静态内容服务时的效率，在它提供动态内容服务时内存映射也起着重要作用。在动态内容服务的过程中，Web 服务器需要使用 fork 函数创建一个子进程去执行 CGI 程序。在创建子进程的过程中，操作系统内核需要为它创建各种标识性的数据结构并为其分配一个 PID，并且子进程将带有一个自己独立的虚拟地址空间。而这个独立的虚拟地址空间即是子进程从父进程内存映射过来的私有的写时复制的对象。

在新创建的子进程中，该进程需要调用 execve 函数来加载并执行 CGI 程序，在这个加载并执行的过程中，首先需要删除已存在用户部分中的已存在区域结构；然后就需要用到内存映射将 CGI 程序的代码段、数据段和堆栈段等区域结构映射成一个新的私有的、写时复制的区域；接着需要将 CGI 程序涉及的共享对象链接在动态链接的基础上映射到用户的虚拟地址空间中的共享区域中；最后 execve 函数需要做的是设置该 CGI 程序上下文中的程序计数器将其指向代码段入口。

总的说，Web 服务器提供动态内容服务时，内存映射在 CGI 程序加载并执行的过程中扮演者十分关键的角色。

2.4 Web 服务器扩展

实现了网络通信和资源管理，一个 Web 服务器已经具备了它最为基本的功能。但是，像在上文介绍过的当下流行 Web 服务器都有一个共同的特点，那就是它们都多多少少具有丰富的扩展功能，这也是随着用户需求不断提升技术演变的结果。在这一节中本文将介绍对 Web 服务器来说十分重要的两个扩展，那就是动态内容服务和高并发处理能力。为了更加深入的了解 Web 服务器提供动态内容服务的机制，本文设计的 Web 服务器将采用最为经典的 CGI 通用网关接口来实现该功能。同时，使用 I/O 多路并发技术来实现 Web 服务器的并发处理能力，因为当下几乎所有的高性能 Web 服务器都是采用这种基于 I/O 描述符访问技术的事件驱动方式实现高性能的并发处理能力。

2.4.1 动态内容服务

早期的 Web 服务器只能提供纯文本内容，在引入 MIME 数据类型后也能够支持多媒体内容。后来，Web 服务中引入了一个非常重要的扩展—动态内容服务，这种 Web 内容交互式的、动态的内容发布方式极大的提高了 Web 服务的质量。所谓动态内容服务就是 Web 服务器能够根据用户在客户端输入的请求产生对应的结果响应给客户端，例如一个提供计算器服务的网页能够根据用户输入返回对应结果、一个有账号系统的网页在登陆不同账号时会进入内容不同的 Web 页面等 Web 服务都是动态内容服务的体现。

然而，要实现动态内容服务的话，你会发现很多问题。首先，运行在不同主机上的客户端和服务器，要怎么才能让 Web 服务器接收到用户在客户端输入的内容呢？当服务器收到了用户输入的内容又要怎么去处理这些内容呢？最后，当服务器完成了对用户输入内容的处理之后又怎样把处理结构返回给客户端呢？第一个问题是很容易解决的，各种 HTTP 请求方法帮助我们解决了这一问题，在 GET 请求方法中，HTTP 请求中的 URL 使用“？”字符添加并用“&”字符隔开用户输入的参数；在 POST 请求方法中参数的传递则是通过请求主体实现。

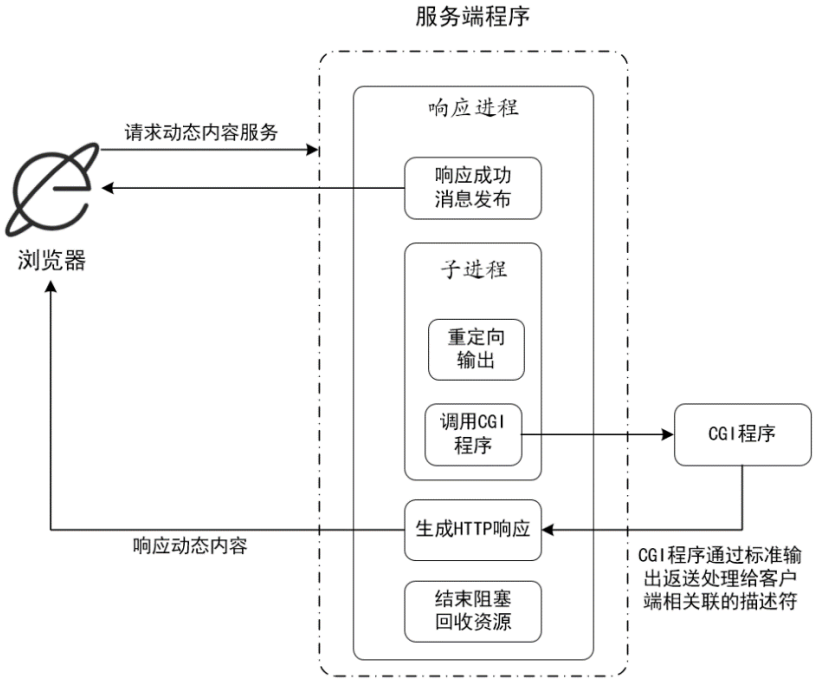

图 2.4 CGI 程序实现动态内容服务流程图

第二个问题就需要花费一下脑筋，当下较为流行的解决方案就是使用 ASP 或者 JSP 编写 HTML 文件脚本实现动态内容服务，这种方式的确极大的加速了动态内容页面的开发【8】。

这种将动态内容的生成与结果显示分离、强调可移植性、注重跨平台的可重用组件使用和强调易用性的策略，在跨平台能力、快速开发和动态内容响应上都有着显著的优势。但是为了更加深入的了解 Web 服务器提供动态内容服务的机制，本文选择了通用网关接口 CGI 来解决这一问题，从更加底层的角度去理解这一过程。

CGI 使用一种实际标准，它的功能就是为 Web 服务器和 CGI 程序提供参数传递的标准接口。在这个过程中，当 Web 服务器收到用户在客户端输入的参数之后，它将调用 fork 函数创建一个子进程，而在这个子进程中 execve 函数将被调用来执行指定的根据 CGI 标准写的 CGI 程序来完成请求处理工作【cite15】。CGI 程序是遵守 CGI 标准的程序，它们通常是由 HTML 触发的然后由 Web 服务器来运行的。当 Web 服务器收到一个与 CGI 程序有关的动态内容请求时，它将创建子进程来调用指定的 CGI 程序来解析该动态内容请求并执行相关程序完成相关分析和处理工作，并最终生成 HTML 文件返送给客户端并显示在浏览器上。这样一个过程如图 2.4 所示。

Web 服务器和执行 CGI 程序的子进程之间是通过 CGI 定义的大量环境变量、标准 I/O 和命令行的方式来传递参数和其他信息的【cite18】。例如 GET 请求方法中的参数被 Web 服务器存储在 QUERY_STRING 环境变量中，CGI 程序可以通过 getenv 函数读取数据内容；POST 请求方法通过标准输入提交的数据表单，在 Web 服务器中被作为请求体读取并用环境变量 CONTENT_TYPE 和 CONTENT_LENGTH 分别记录该表单的 MIME 数据类型和字节大小，当 CGI 程序需要使用该数据时可以直接根据环境变量读取请求体中传递的数据。一些常用的 CGI 环境变量如表 2.10 所示。

表 2.10 CGI 定义的部分常用环境变量

环境变量	环境变量表示的含义
QUERY_STRING	客户端传递的参数
SERVER_PORT	Web 服务器的端口号
REMOTE_HOST	客户端的域名
REMOTE_ADDR	客户端的 IP 地址
REQUEST_METHOD	客户端使用的请求方法
CONTENT_TYPE	POST 请求方法中请求体的 MIME 数据类型
CONTENT_LENGTH	POST 请求方法中的请求体字节大小

上述的动态内容请求处理过程中提到，CGI 程序完成请求分析和处理工作之后，生成了处理结果并以此构建了 HTML 文件。最后就来到了第三个问题，Web 服务器是怎么把这个这个处理结果传送给客户端的。和 CGI 程序读取标准输入获取 POST 请求方法的请求体相似，在它完成处理之后，将动态内容作为响应体通过标准输出返送给客户端的，因为整个处理过程是在调用 CGI 程序的子进程中完成的，Web 服务器的主进程无法获知 HTTP 响应内容的具体相关信息，所以这一任务由调用的子进程完成，它需要完成响应报头的 CONTENT_TYPE 和 CONTENT_LENGTH 等信息的生成。另外，在子进程调用 execve 函数加载支持 CGI 程序前，需要调用 dup2 函数完成标准输出重定向到和链接的客户端相关联文件描述符，完成动态内容的直接返送。

2.4.2 基于 I/O 多路复用的并发处理

Web 服务器的 I/O 模型常见的有同步阻塞 I/O（Blocking I/O）、同步非阻塞 I/O（Non-blocking I/O）、I/O 多路复用（I/O Multiplexing）和异步 I/O（Asynchronous I/O）四种模型。同步与异步是指用户线程与操作系统内核的交互方式，同步机制是指在用户线程发起 I/O 请求之后需要等待内核 I/O 操作完成，异步机制则是指用户进程发起 I/O 请求之后继续往后执行，当内核 I/O 空闲时告知用户线程可以完成 I/O 操作。阻塞与非阻塞方式则是指用户线程调用操作

系统内核 I/O 操作的方式，阻塞方式是指调用内核 I/O 操作时如果没有可读写的数据，用户线程会陷入阻塞状态直至可读写数据出现；非阻塞方式是指即使没有可读写数据，在 I/O 操作被调用时会立即返回给用户一个状态值，不会陷入阻塞状态继续占用系统资源。

清楚了同步与异步和阻塞与非阻塞的概念，理解那四种 I/O 模型的工作机制就比较简单了。同步阻塞 I/O 是最为初级的 I/O 模型，当用户线程需要进行 I/O 操作时，该线程将被完全阻塞在请求内核 I/O 操作的过程中，这严重影响了 CPU 的利用率。同步非阻塞 I/O 在同步阻塞 I/O 的基础上，在请求内核 I/O 操作后会立即返回到程序执行并等待内核 I/O 操作使用的应答，虽然避免了被完全阻塞，但是在内核 I/O 操作请求过程中需要不断的轮询和重复请求 I/O 这将严重消耗计算资源。I/O 多路复用也称为异步阻塞 I/O，本文的 I/O 多路复用实现是基于内核实现的多路分离函数 select，采用 select 函数可以避免同步非阻塞 I/O 中不断轮询等待 I/O 操作的问题。使用 select 函数也可以实现 I/O 多路复用可以在面对多个 socket 的情况下，使用 select 函数不断读取处于可读状态的 socket 实现在一个线程内同时处理多个 I/O 请求的目的。另外，虽然 I/O 多路复用模型采用阻塞的方式调用 I/O 操作使得多个线程被阻塞更长的时间，但是使用 select 函数可以让用户线程只进行 socket 或者 I/O 请求的注册工作，完成注册后即可返回继续执行程序使得 CUP 利用率变高。异步 I/O 也称异步非阻塞 I/O，相比 I/O 多路复用模型中用户线程自行进行 I/O 操作，在异步 I/O 模型中，当内核告知用户线程文件句柄的状态信息时，要读取的数据已经被内核读取并存放在了用户线程的指定缓冲区中，接收到状态信息通知后，用户线程可以直接使用数据。但是异步 I/O 模型对操作系统内核支持要求较高，并不常用在 Web 服务器的 I/O 模型中，所以现代的高性能 Web 服务器都使用基于 I/O 多路复用的 I/O 模型并以事件驱动实现高性能。

I/O 多路复用的 I/O 模型可以用作事件驱动编程的基础。事件驱动编程是指事务执行的逻辑流由某些事件推进，而一个事务执行的逻辑流可以模型化为编译原理中的状态机由一组状态、输入事件和状态转移组成。在采用 I/O 多路复用的并发 Web 服务器中，每当接入一个客户端时，服务器便为其创建一个状态机，并将这个状态机和已连接的描述符建立联系。在最开始时，客户端处于“等待已连接描述符准备好可读”状态，当有输入事件“已连接描述符可读”发生时，输入事件和输入状态触发了状态转移将状态机的状态转换为“从已连接描述符读取文本行”，这时 Web 服务器为该状态机执行转移操作为其从已连接描述符读或写文本行。这就是基于 I/O 多路复用的并发 Web 服务器实现并发的基本原理，将逻辑流模型化为状态机，不断读取多个状态机的状态来完成多个 I/O 请求的处理。

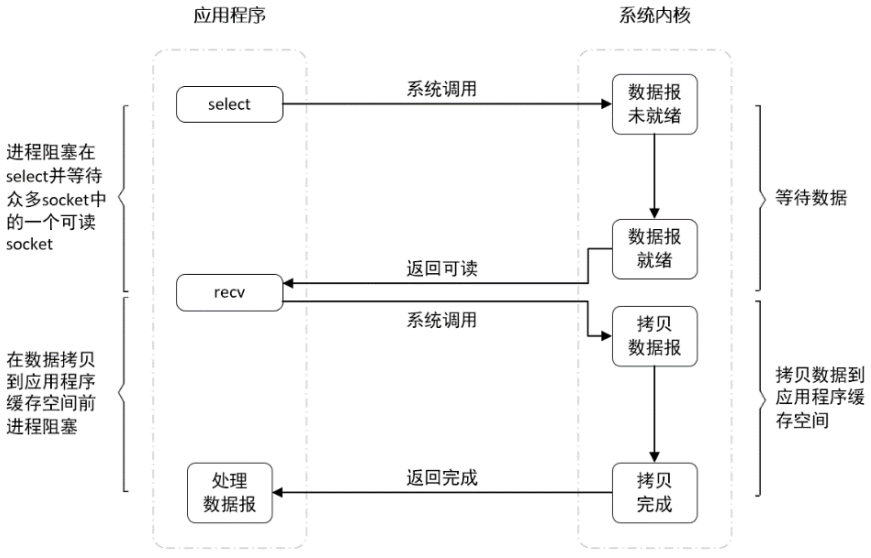

图 2.5 基于 select 模型的 I/O 多路复用技术实现原理图

使用 select 函数实现 I/O 多路复用技术的原理图如图 2.5 所示。本文在 I/O 多路复用技术使用中对输入事件检测是使用 select 函数来实现的，其基本原理是 select 函数将客户端请求处理线程挂起，直到一个或多个 I/O 事件发生之后，才返回到客户端请求处理线程完成处理工作。在上述介绍的 I/O 多路复用并发 Web 服务器中，select 需要检测标准输入的描述符和 Web 服务器的套接字描述符等一组读集合的描述符集合。在这一过程中，当 Web 服务器启动并初始化套接字描述符设置一个监听描述符后，便开始调用 select 函数，调用伊始 select 函数使用 FD_ZERO 宏创建一个空的读集合的描述符集合；然后使用 FD_SET 宏将标准输入描述符和 Web 服务器的监听套接字描述符加入到读描述符集合中。在此之后完成了最开始的初始化工作，接下来在 Web 服务器的无限迭代循环中就需要不断的检测描述符的可读状态，并使用 FD_ISSET 宏来确定已连接描述符的可读状态。如果是监听套接字描述符准备好可读，就使用 accept 函数与服务器建立连接，并读取套接字描述符中的内容，将每一个文本行回送直到客户端的连接关闭。

2.5 本章小结

本章对本文的 I/O 多路复用的并发 Web 服务器设计中可能涉及的关键技术进行了详细的分析与研究。在第一节的 Web 服务器介绍中，先从上帝视角整体地介绍了构建一个 Web 服务器需要的基本知识以及一些 Web 服务相关的基础内容；另外，也详细介绍了当下较为流行的几种 Web 服务器，并总结出了具备高性能和高抗并发能力的事件驱动 Web 服务器的开发趋势。

以第一节介绍的 Web 服务器基础知识为基础，本文在后续几节中详细介绍分析了 Web 服务器开发过程中需要使用的网络通信、资源操作和服务器扩展实现的一些关键技术。首先，本文从网络通信开始，在相对底层的角度讲述了客户端-服务器的 Web 服务器模型需要通过套接字接口建立连接和传输数据，并以 HTTP 协议为基础提供 Web 服务传输内容。然后，本文以 Unix I/O 和内存映射为例介绍了 Web 服务器在为客户端提供服务时所进行资源操作在底层的实现基础。最后，本文介绍了 Web 服务器非常重要的两个扩展，一个是动态内容服务，另一个则是抗并发能力。动态内容服务实现介绍了最为经典的 CGI 标准更深入的理解 Web 服务器实现动态内容服务的原理，而抗并发能力则是使用基于 select 模型的 I/O 多路复用技术，这一技术是事件驱动编程的实现方式之一，可以实现 Web 服务器的高性能。

[1] Apache Software Foundation. Apache[EB/OL]. https://www.apache.org, 2019
[2] Apache Software Foundation. Apache Tomcat[EB/OL]. https://www.tomcat.apache.org, 2020
[3] Jan Kneschke. Lighttpd[EB/OL]. https://www.lighttpd.net, 2020
[4] Igor Sysoev. Nginx[EB/OL]. https://www.nginx.org, 2020
[5] Fielding R T, Bernerslee T, Frystyk H. Hypertext Transfer Protocol -- HTTP/1.0[M]. 1996.
[6] 钱宏武. HTTP 协议之前世今生——兼谈网络应用结构设计[J]. 程序员, 2008(5):78-80.
[7] Fielding R , Gettys J , Mogul J , et al. Hypertext Transfer Protocol -- HTTP/1.1[J]. Computer Science & Communications Dictionary, 1999, 7(4):595-599.
[8] 蔡剑, 景楠. Java Web 应用开发:J2EE 和 Tomcat.第 2 版[M]. 清华大学出版社, 2005.