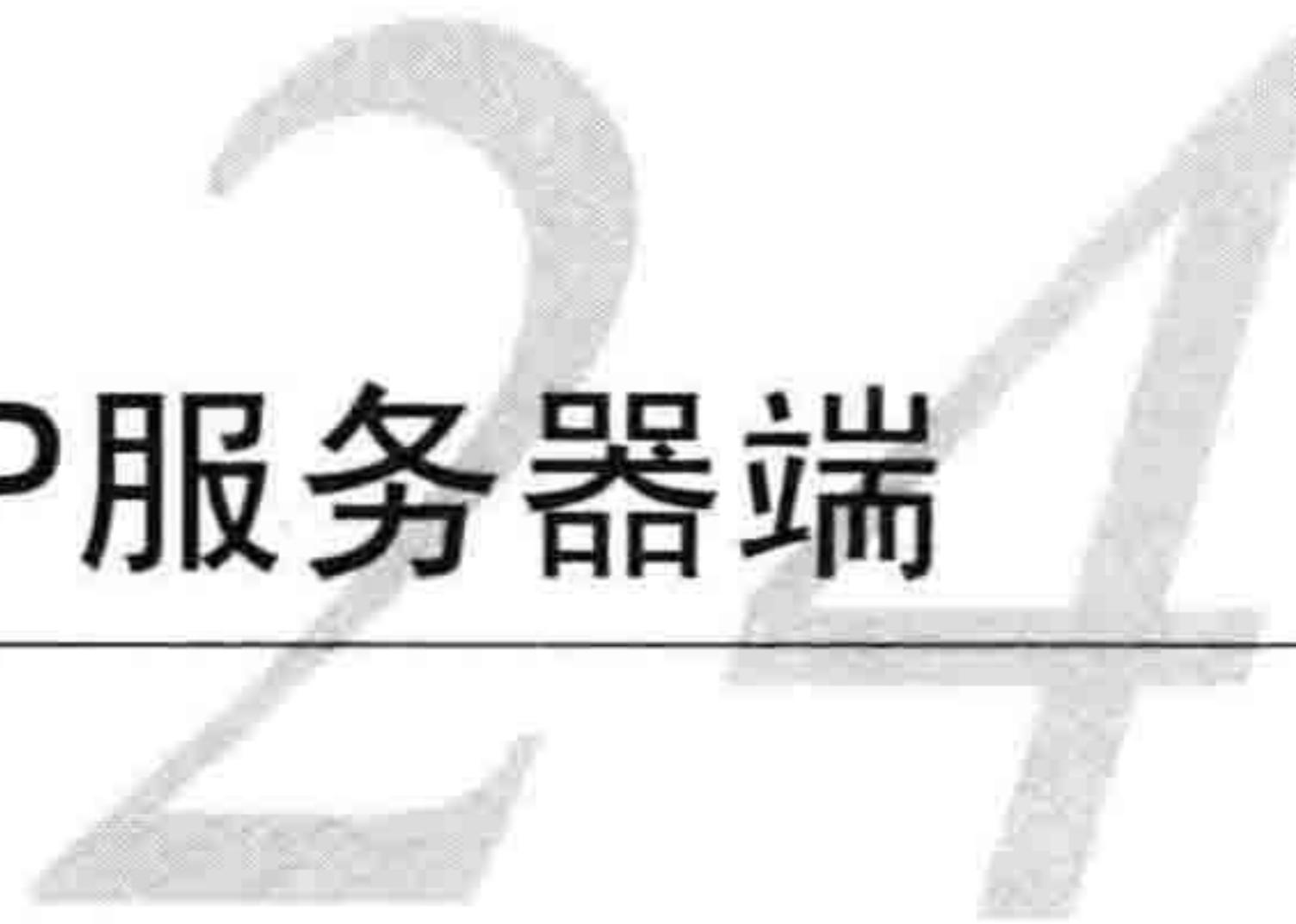


Part 04

结束网络编程



各位已学习了大量编程知识，即便如此，大家脑海中浮现的大部分还是回声服务器端/客户端，最多也就是聊天程序。但掌握了这些基础程序的编写方法，就相当于具备了开发应用层网络程序的基本能力。我们已经能够编写服务器端和客户端进行数据交换，接下来应该学习应用程序的编写方法。

24.1 HTTP 概要

本章以编写真实应用程序为目标，在所学理论知识的基础上，编写HTTP（Hypertext Transfer Protocol，超文本传输协议）服务器端，即Web服务器端。

+ 理解 Web 服务器端

互联网的普及使Web服务器端为大众熟知。下面是我对Web服务器端的定义：

“基于HTTP协议，将网页对应文件传输给客户端的服务器端。”

HTTP是Hypertext Transfer Protocol的缩写，Hypertext（超文本）是可以根据客户端请求而跳转的结构化信息。例如，各位通过浏览器访问图灵社区的主页时，首页文件将传输到浏览器并展现给大家，此时各位可以点击鼠标跳转到任意页面。这种可跳转的文本（Text）称为超文本。

HTTP协议又是什么呢？HTTP是以超文本传输为目的而设计的应用层协议，这种协议同样属于基于TCP/IP实现的协议，因此，我们也可以直接实现HTTP。从结果上看，实现该协议相当于实现Web服务器端。另外，浏览器也属于基于套接字的客户端，因为连接到任意Web服务器端时，浏览器内部也会创建套接字。只不过浏览器多了一项功能，它将服务器端传输的HTML格式的超文本解析为可读性较强的视图。总之，Web服务器端是以HTTP协议为基础传输超文本的服务器端。

提 示**HTTP 是专用名词**

HTTP 中的 P 是 Protocol (协议) 的缩写，所以 “HTTP 协议” 这种表示其实重复表达了 “协议” 一词。但 HTTP 属于专用名词，因此通常直接表述为 “HTTP 协议” 。

+ HTTP

下面详细讨论HTTP协议。虽然它相对简单，但要完全驾驭也并非易事。接下来只介绍编写Web服务器端时的必要内容。

无状态的Stateless协议

为了在网络环境下同时向大量客户端提供服务，HTTP协议的请求及响应方式设计如图24-1所示。

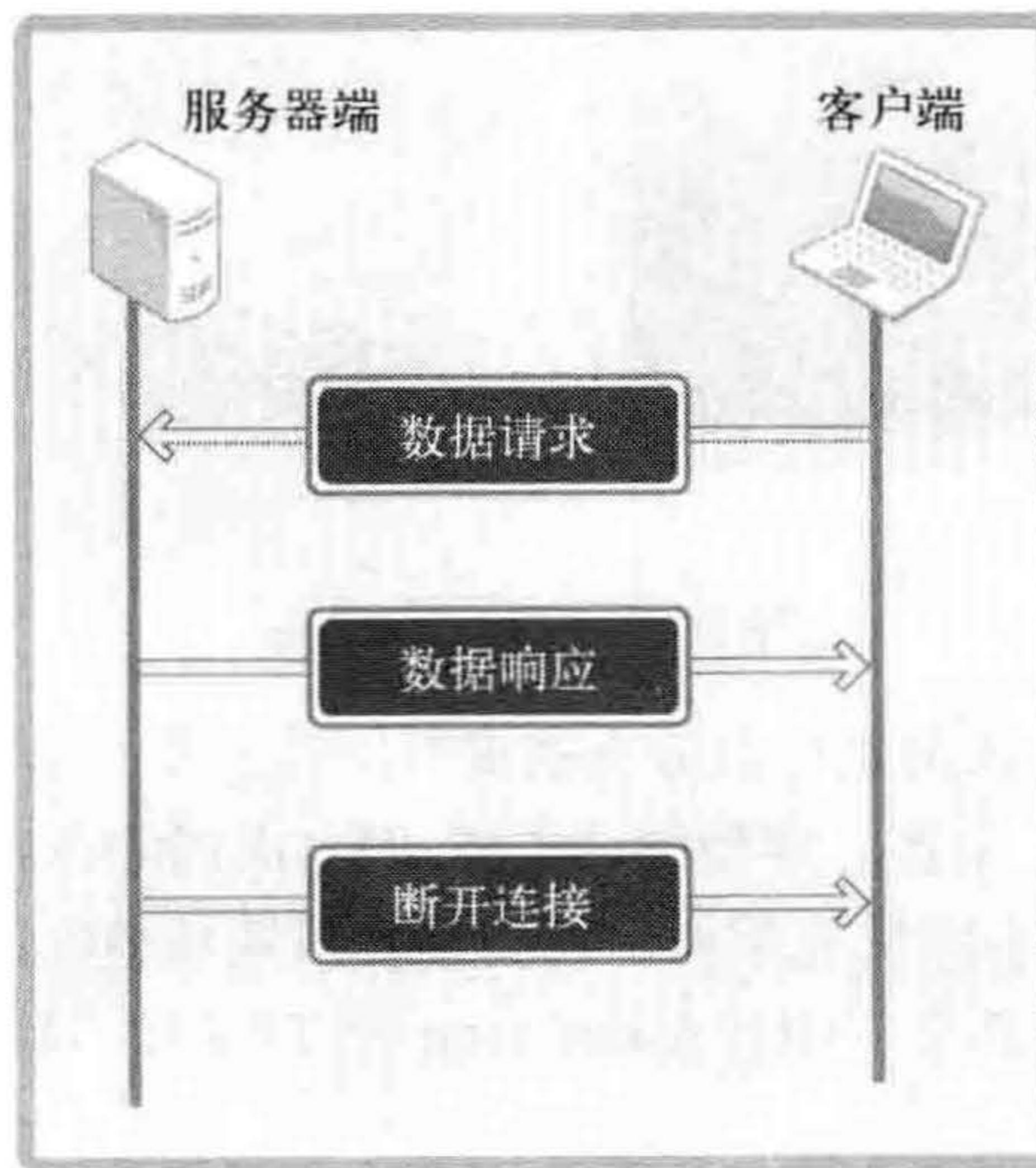


图24-1 HTTP请求/响应过程

从图24-1中可以看到，服务器端响应客户端请求后立即断开连接。换言之，服务器端不会维持客户端状态。即使同一客户端再次发送请求，服务器端也无法辨认出是原先那个，而会以相同方式处理新请求。因此，HTTP又称“无状态的Stateless协议”。

提 示

Cookie & Session

为了弥补HTTP无法保持连接的缺点，Web编程中通常会使用Cookie和Session技术。相信各位都接触过购物网站的购物车功能，即使关闭浏览器也不会丢失购物车内的信息（甚至不用登录）。这种保持状态的功能都是通过Cookie和Session技术实现的。

请求消息（Request Message）的结构

下面介绍客户端向服务器端发送的请求消息的结构。Web服务器端需要解析并响应客户端请求，客户端和服务器端之间的数据请求方式标准如图24-2所示。

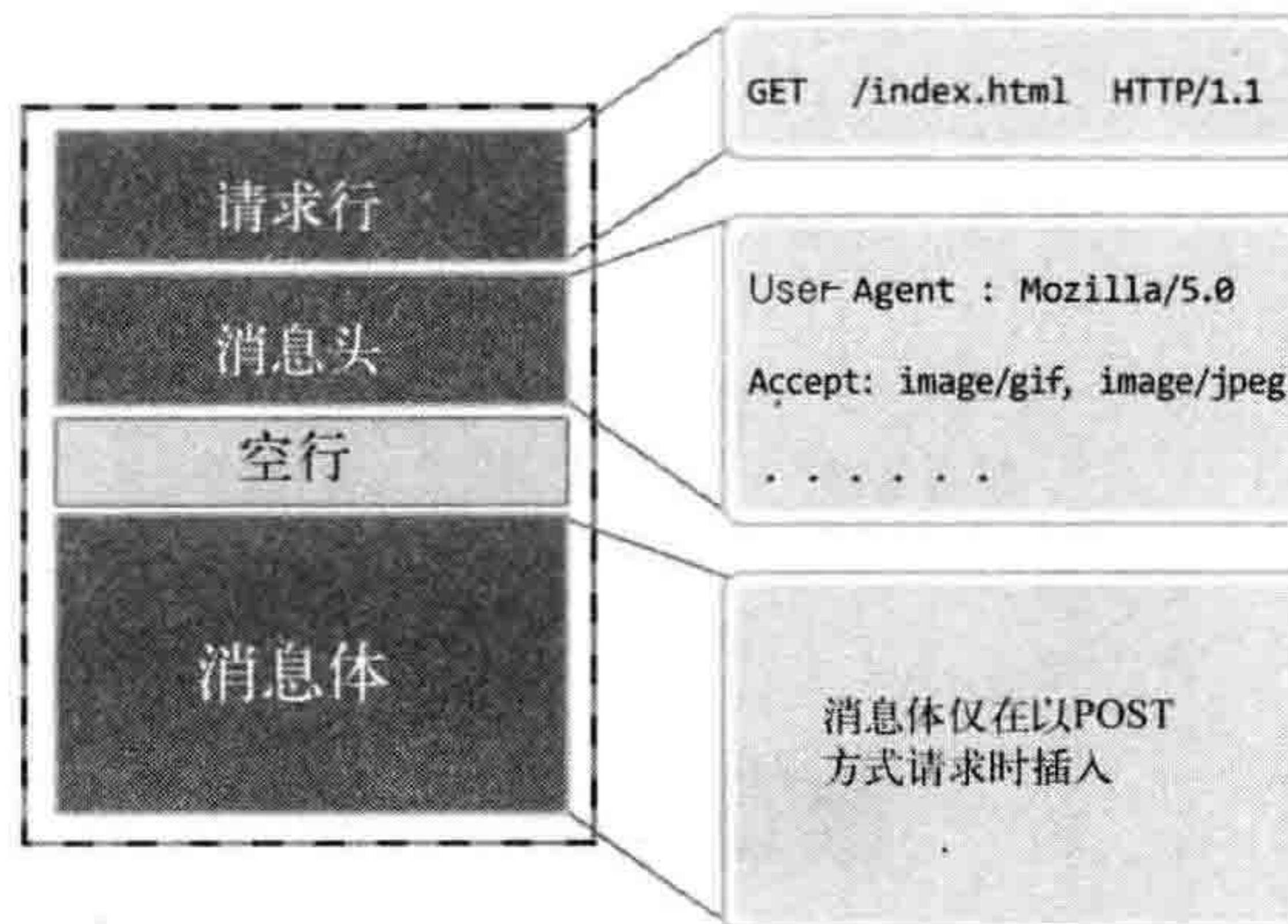


图24-2 HTTP请求头

从图24-2中可以看到，请求消息可以分为请求行、消息头、消息体等3个部分。其中，请求行含有请求方式（请求目的）信息。典型的请求方式有GET和POST，GET主要用于请求数据，POST主要用于传输数据。为了降低复杂度，我们实现只能响应GET请求的Web服务器端。下面解释图24-2中的请求行信息。其中“GET /index.html HTTP/1.1”具有如下含义：

“请求（GET）index.html文件，希望以1.1版本的HTTP协议进行通信。”

请求行只能通过1行（line）发送，因此，服务器端很容易从HTTP请求中提取第一行，并分析请求行中的信息。

请求行下面的消息头中包含发送请求的（将要接收响应信息的）浏览器信息、用户认证信息等关于HTTP消息的附加信息。最后的消息体中装有客户端向服务器端传输的数据，为了装入数据，需要以POST方式发送请求。但我们的目标是实现GET方式的服务器端，所以可以忽略这部分内容。另外，消息体和消息头之间以空行分开，因此不会发生边界问题。

响应消息 (Response Message) 的结构

下面介绍Web服务器端向客户端传递的响应信息的结构。从图24-3中可以看到，该响应消息由状态行、头信息、消息体等3个部分构成。状态行中含有关于请求的状态信息，这是其与请求消息相比最为显著的区别。

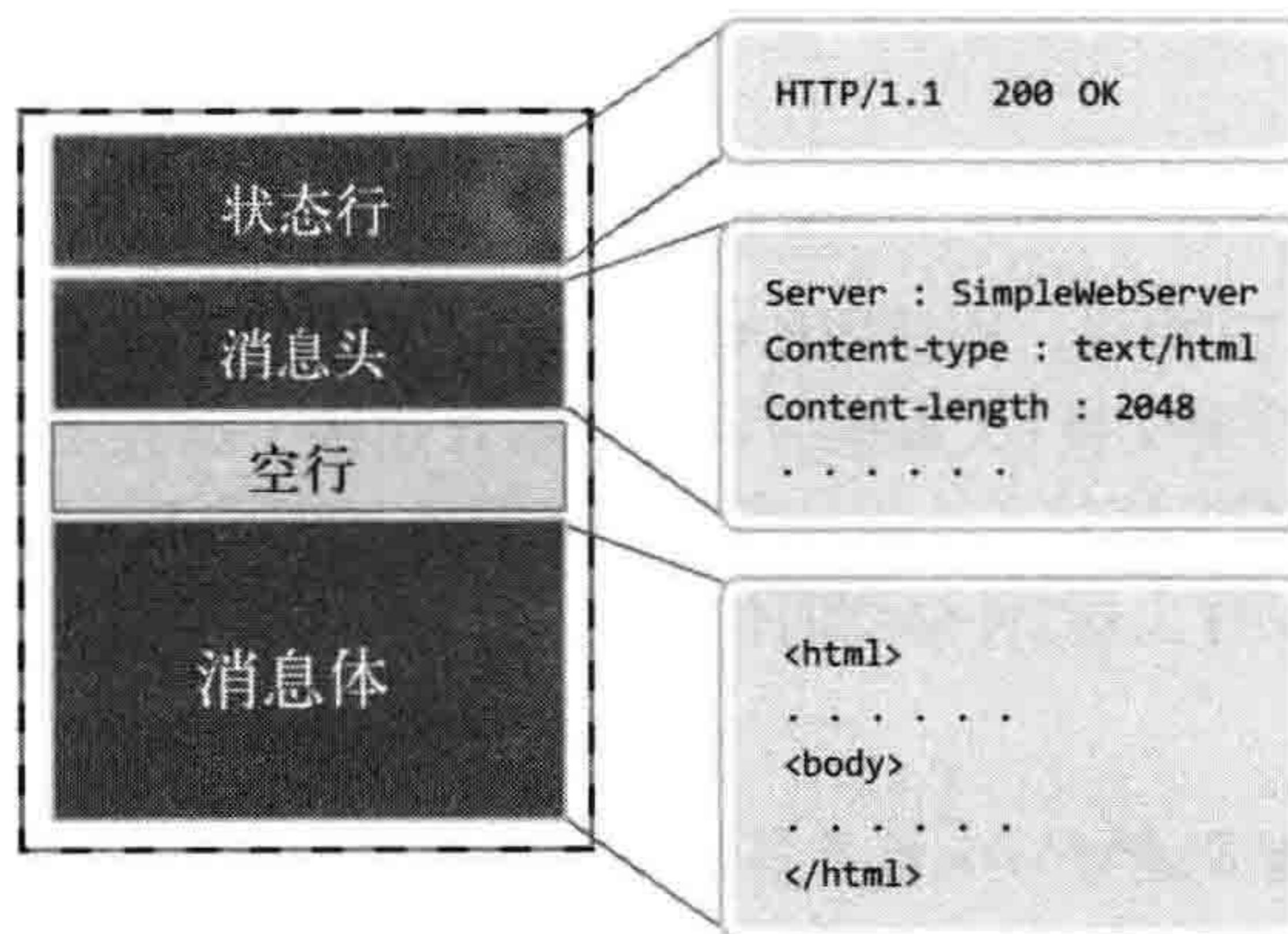


图24-3 HTTP响应头

从图24-3中可以看到，第一个字符串状态行中含有关于客户端请求的处理结果。例如，客户端请求index.html文件时，表示index.html文件是否存在、服务器端是否发生问题而无法响应等不同情况的信息将写入状态行。图24-3中的“HTTP/1.1 200 OK”具有如下含义：

“我想用HTTP1.1版本进行响应，你的请求已正确处理（200 OK）。”

表示“客户端请求的执行结果”的数字称为状态码，典型的有以下几种。

- 200 OK：成功处理了请求！
- 404 Not Found：请求的文件不存在！
- 400 Bad Request：请求方式错误，请检查！

消息头中含有传输的数据类型和长度等信息。图24-3中的消息头含有如下信息：

“服务器端名为SimpleWebServer，传输的数据类型为text/html（html格式的文本数据）。数据长度不超过2048字节。”

最后插入1个空行后，通过消息体发送客户端请求的文件数据。以上就是实现Web服务器端过程中必要的HTTP协议。要编写完整的Web服务器端还需要更多HTTP协议相关知识，而对于我们的目标而言，这些内容已经足够了。

24.2 实现简单的Web服务器端

现在开始在HTTP协议的基础上编写Web服务器端。先给出Windows平台下的示例，再给出Linux下的示例。前面介绍了HTTP协议的相关背景知识，有了这些基础就不难分析源代码。因此，除了一些简单的注释外，不再另行说明代码。

+ 实现基于Windows的多线程Web服务器端

Web服务器端采用HTTP协议，即使用IOCP或epoll模型也不会大幅提升性能（当然并不是完全没有）。客户端和服务器端交换1次数据后将立即断开连接，没有足够时间发挥IOCP或epoll的优势。在服务器端和客户端保持较长连接的前提下频繁发送大小不一的消息时（最典型的就是网游服务器端），才能真正发挥出这2种模型的优势。

提 示

能否通过Web服务器端比较IOCP和epoll的性能

利用IOCP和epoll实现Web服务器端”本身没有问题，现实中有时也会通过IOCP或epoll实现类似的服务器端，但无法通过这种服务器端完全体会到IOCP和epoll的优点。我从事过软件性能评估相关工作，当时，客户公司的工程师向我展示过Web服务器端中IOCP和epoll的性能分析结果，我也耐心解释了为何分析结果没有太大意义。

因此，我通过多线程模型实现Web服务器端。也就是说，客户端每次请求时，都创建1个新线程响应客户端请求。

◆ webserv_win.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
#include <process.h>

#define BUF_SIZE 2048
#define BUF_SMALL 100

unsigned WINAPI RequestHandler(void* arg);
char* ContentType(char* file);
void SendData(SOCKET sock, char* ct, char* fileName);
void SendErrorMsg(SOCKET sock);
void ErrorHandling(char *message);

int main(int argc, char *argv[])
```

```

{
    WSADATA wsaData;
    SOCKET hServSock, hClntSock;
    SOCKADDR_IN servAddr, clntAddr;

    HANDLE hThread;
    DWORD dwThreadID;
    int clntAddrSize;

    if(argc!=2) {
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }

    if(WSAStartup(MAKEWORD(2, 2), &wsaData)!=0)
        ErrorHandling("WSAStartup() error!");

    hServSock=socket(PF_INET, SOCK_STREAM, 0);
    memset(&servAddr, 0, sizeof(servAddr));
    servAddr.sin_family=AF_INET;
    servAddr.sin_addr.s_addr=htonl(INADDR_ANY);
    servAddr.sin_port=htons(atoi(argv[1]));

    if(bind(hServSock, (SOCKADDR*) &servAddr, sizeof(servAddr))==SOCKET_ERROR)
        ErrorHandling("bind() error");
    if(listen(hServSock, 5)==SOCKET_ERROR)
        ErrorHandling("listen() error");

    /*请求及响应*/
    while(1)
    {
        clntAddrSize(sizeof(clntAddr));
        hClntSock=accept(hServSock, (SOCKADDR*)&clntAddr, &clntAddrSize);
        printf("Connection Request : %s:%d\n",
               inet_ntoa(clntAddr.sin_addr), ntohs(clntAddr.sin_port));
        hThread=(HANDLE)_beginthreadex(
            NULL, 0, RequestHandler, (void*)hClntSock, 0, (unsigned *)&dwThreadID);
    }
    closesocket(hServSock);
    WSACleanup();
    return 0;
}

unsigned WINAPI RequestHandler(void *arg)
{
    SOCKET hClntSock=(SOCKET)arg;
    char buf[BUF_SIZE];
    char method[BUF_SMALL];
    char ct[BUF_SMALL];
    char fileName[BUF_SMALL];

    recv(hClntSock, buf, BUF_SIZE, 0);
    if(strstr(buf, "HTTP/")) // 查看是否为HTTP提出的请求
    {

```

```

SendErrorMsg(hClntSock);
closesocket(hClntSock);
return 1;
}

strcpy(method, strtok(buf, " /"));
if(strcmp(method, "GET")) // 查看是否为GET方式的请求
    SendErrorMsg(hClntSock);

strcpy(fileName, strtok(NULL, " /")); // 查看请求文件名
strcpy(ct, ContentType(fileName)); // 查看Content-type
SendData(hClntSock, ct, fileName); // 响应
return 0;
}

void SendData(SOCKET sock, char* ct, char* fileName)
{
    char protocol[]="HTTP/1.0 200 OK\r\n";
    char servName[]="Server:simple web server\r\n";
    char cntLen[]="Content-length:2048\r\n";
    char cntType[BUF_SMALL];
    char buf[BUF_SIZE];
    FILE* sendFile;

    sprintf(cntType, "Content-type:%s\r\n\r\n", ct);
    if((sendFile=fopen(fileName, "r"))==NULL)
    {
        SendErrorMsg(sock);
        return;
    }

    /*传输头信息*/
    send(sock, protocol, strlen(protocol), 0);
    send(sock, servName, strlen(servName), 0);
    send(sock, cntLen, strlen(cntLen), 0);
    send(sock, cntType, strlen(cntType), 0);

    /*传输请求数据*/
    while(fgets(buf, BUF_SIZE, sendFile)!=NULL)
        send(sock, buf, strlen(buf), 0);

    closesocket(sock); // 由HTTP协议响应后断开
}

void SendErrorMsg(SOCKET sock) // 发生错误时传递消息
{
    char protocol[]="HTTP/1.0 400 Bad Request\r\n";
    char servName[]="Server:simple web server\r\n";
    char cntLen[]="Content-length:2048\r\n";
    char cntType[]="Content-type:text/html\r\n\r\n";
    char content[]="<html><head><title>NETWORK</title></head>
                    <body><font size=+5><br>发生错误！查看请求文件名和请求方式！
                    </font></body></html>";
}

```

```

    send(sock, protocol, strlen(protocol), 0);
    send(sock, servName, strlen(servName), 0);
    send(sock, cntLen, strlen(cntLen), 0);
    send(sock, cntType, strlen(cntType), 0);
    send(sock, content, strlen(content), 0);
    closesocket(sock);
}

char* ContentType(char* file) // 区分Content-type
{
    char extension[BUF_SMALL];
    char fileName[BUF_SMALL];
    strcpy(fileName, file);
    strtok(fileName, ".");
    strcpy(extension, strtok(NULL, "."));
    if(!strcmp(extension, "html")||!strcmp(extension, "htm"))
        return "text/html";
    else
        return "text/plain";
}

void ErrorHandling(char* message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}

```

图24-4给出上述示例的运行结果。首先启动该服务器端，再启动Web浏览器进行连接。



图24-4 运行结果1

24

从图24-4的运行结果可以看出，地址栏中输入了如下地址：

`http://localhost:9190/index.html`

该请求相当于连接到IP为127.0.0.1、端口为9190的套接字，并请求获取index.html文件。可以用回送地址127.0.0.1代替“localhost”。通过图24-5观察客户端，可以看出未找到请求的文件。



图24-5 运行结果2

可以在页面上半部分找到状态码400，因为地址栏中输入的index2.html文件不存在。

+ 实现基于Linux的多线程Web服务器端

Linux下的Web服务器端与上述示例不同，将使用标准I/O函数。在此列出的目的主要是为了让各位多复习各种知识点，没有任何特殊含义。

❖ webserv_linux.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <pthread.h>

#define BUF_SIZE 1024
#define SMALL_BUF 100

void* request_handler(void* arg);
void send_data(FILE* fp, char* ct, char* file_name);
char* content_type(char* file);
void send_error(FILE* fp);
void error_handling(char* message);

int main(int argc, char *argv[])
{
```

```

int serv_sock, clnt_sock;
struct sockaddr_in serv_addr, clnt_addr;
int clnt_addr_size;
char buf[BUF_SIZE];
pthread_t t_id;
if(argc!=2) {
    printf("Usage : %s <port>\n", argv[0]);
    exit(1);
}

serv_sock=socket(PF_INET, SOCK_STREAM, 0);
memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
serv_addr.sin_port = htons(atoi(argv[1]));
if(bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
    error_handling("bind() error");
if(listen(serv_sock, 20)==-1)
    error_handling("listen() error");

while(1)
{
    clnt_addr_size(sizeof(clnt_addr));
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr, &clnt_addr_size);
    printf("Connection Request : %s:%d\n",
        inet_ntoa(clnt_addr.sin_addr), ntohs(clnt_addr.sin_port));
    pthread_create(&t_id, NULL, request_handler, &clnt_sock);
    pthread_detach(t_id);
}
close(serv_sock);
return 0;
}

void* request_handler(void *arg)
{
    int clnt_sock=*((int*)arg);
    char req_line[SMALL_BUF];
    FILE* clnt_read;
    FILE* clnt_write;

    char method[10];
    char ct[15];
    char file_name[30];

    clnt_read=fopen(clnt_sock, "r");
    clnt_write=fopen(dup(clnt_sock), "w");
    fgets(req_line, SMALL_BUF, clnt_read);
    if(strstr(req_line, "HTTP/")==NULL)
    {
        send_error(clnt_write);
        fclose(clnt_read);
        fclose(clnt_write);
        return;
    }
}

```

```
strcpy(method, strtok(req_line, " /"));
strcpy(file_name, strtok(NULL, " /"));
strcpy(ct, content_type(file_name));
if(strcmp(method, "GET")!=0)
{
    send_error(clnt_write);
    fclose(clnt_read);
    fclose(clnt_write);
    return;
}

fclose(clnt_read);
send_data(clnt_write, ct, file_name);
}

void send_data(FILE* fp, char* ct, char* file_name)
{
    char protocol[]="HTTP/1.0 200 OK\r\n";
    char server[]="Server:Linux Web Server \r\n";
    char cnt_len[]="Content-length:2048\r\n";
    char cnt_type[SMALL_BUF];
    char buf[BUF_SIZE];
    FILE* send_file;

    sprintf(cnt_type, "Content-type:%s\r\n\r\n", ct);
    send_file=fopen(file_name, "r");
    if(send_file==NULL)
    {
        send_error(fp);
        return;
    }

    /*传输头信息*/
    fputs(protocol, fp);
    fputs(server, fp);
    fputs(cnt_len, fp);
    fputs(cnt_type, fp);

    /*传输请求数据*/
    while(fgets(buf, BUF_SIZE, send_file)!=NULL)
    {
        fputs(buf, fp);
        fflush(fp);
    }
    fflush(fp);
    fclose(fp);
}

char* content_type(char* file)
{
    char extension[SMALL_BUF];
    char file_name[SMALL_BUF];
    strcpy(file_name, file);
```

```

strtok(file_name, ".");
strcpy(extension, strtok(NULL, "."));

if(!strcmp(extension, "html")||!strcmp(extension, "htm"))
    return "text/html";
else
    return "text/plain";
}

void send_error(FILE* fp)
{
    char protocol[]="HTTP/1.0 400 Bad Request\r\n";
    char server[]="Server:Linux Web Server \r\n";
    char cnt_len[]="Content-length:2048\r\n";
    char cnt_type[]="Content-type:text/html\r\n\r\n";
    char content[]="<html><head><title>NETWORK</title></head>
                    <body><font size=+5><br>发生错误！查看请求文件名和请求方式！
                    </font></body></html>";

    fputs(protocol, fp);
    fputs(server, fp);
    fputs(cnt_len, fp);
    fputs(cnt_type, fp);
    fflush(fp);
}

void error_handling(char* message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}

```

关于TCP/IP套接字编程的讲解到此结束。第25章将简单介绍网络编程进阶部分需要学习的内容。

24.3 习题

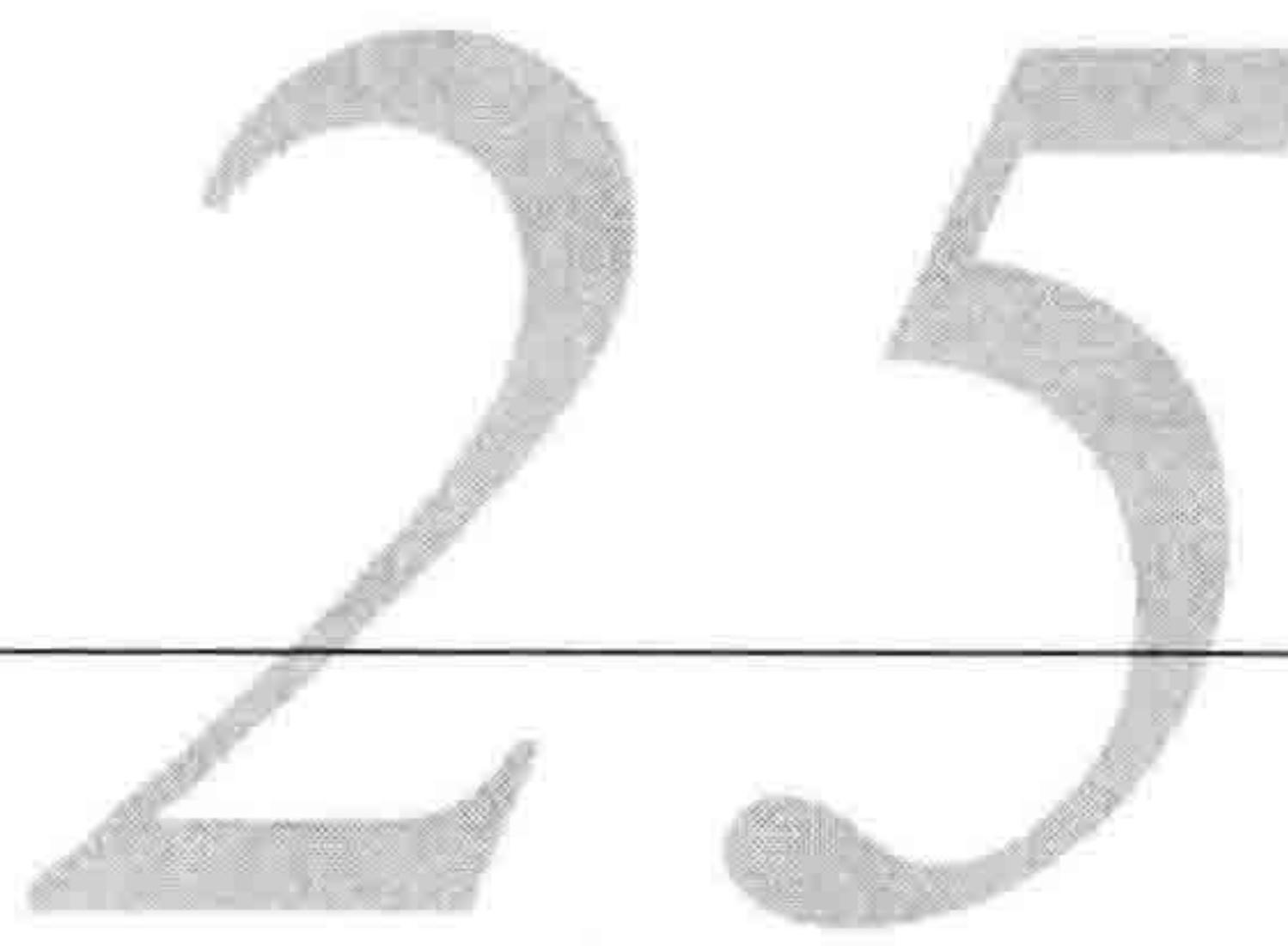
(1) 下列关于Web服务器端和Web浏览器的说法错误的是?

- a. Web浏览器并不是通过自身创建的套接字连接服务器端的客户端。
- b. Web服务器通过TCP套接字提供服务，因为它将保持较长的客户端连接并交换数据。
- c. 超文本与普通文本的最大区别是其具有可跳转的特性。
- d. Web服务器端可视为向浏览器提供请求文件的文件传输服务器端。
- e. 除Web浏览器外，其他客户端都无法访问Web服务器端。

(2) 下列关于HTTP协议的描述错误的是?

- a. HTTP协议是无状态的Stateless协议，不仅可以通过TCP实现，还可通过UDP实现。

- b. HTTP协议是无状态的Stateless协议，因为其在1次请求和响应过程完成后立即断开连接。因此，如果同一服务器端和客户端需要3次请求及响应，则意味着要经过3次套接字创建过程。
 - c. 服务器端向客户端传递的状态码中含有请求处理结果信息。
 - d. HTTP协议是基于因特网的协议，因此，为了同时向大量客户端提供服务，HTTP被设计为Stateless协议。
- (3) IOCP和epoll是可以保证高性能的典型服务器端模型，但如果在基于HTTP协议的Web服务器端使用这些模型，则无法保证一定能得到高性能。请说明原因。



相信各位已经对套接字编程和系统编程有了一定认识。入门的过程感觉有些漫长，可一旦有了基础，接下来的学习并非难事。进入本章意味着我们已经走过了漫长的入门过程。

本章将介绍进阶学习的内容。我希望多介绍一些好的学习方法和经验教训，这也是编写本章的宗旨。

25.1 网络编程学习的其他内容

+ 成为网络程序员的前提是成为程序员

如果对某一领域感兴趣，自然会更关注该领域，并会产生从事该领域相关工作的想法。也有读者向我询问如何成为一名网络程序员，但按照这部分读者的标准，我也很难成为合格的网络程序员。因为我的兴趣并非完全集中于网络领域，而且现阶段也并不从事网络编程相关工作。但工作中也会遇到网络编程的部分，这种情况下也会编写网络程序。

网络编程曾被认为是一种专业领域（现阶段也有人这么认为），但大部分最新软件是基于网络环境开发的，因此，网络编程成为程序员的一种必备技能。

如果对网络编程感兴趣，就应该在兴趣驱使学习程序员必备的所有知识。因为在现代服务器端，收发数据的网络编程部分所占篇幅远远比不上其他部分。优秀的服务器端并不是由优秀的网络程序员实现的，而是由优秀的程序员编写的。

+ 编写这些程序时的参考书

编程经验的不足使我们更加依赖书籍。我接触陌生领域的项目时，也会收集国内外最新出版的图书。但我们无法通过书籍解决项目中遇到的问题，因为软件本身属于创造性工作的产物。编

写同样功能的软件时，每位程序员的实现方法各不相同，此处没有正确答案。本身没有正解的东西很难从书中找到需要的信息，因此，在具备了坚实的理论知识的基础上，应该加强编程训练。

假设各位要开发在线俄罗斯方块对战游戏，大家能在身边找到有经验的程序员。这时不应该提如下这种问题：

“我想实现在线俄罗斯方块对战游戏，该如何编写呢？应该读哪些书？”

我能理解各位的焦虑，但提出这种问题无法得到满意的答复（被提问的人也不知所措）。应该缩小问题范围，而且问题要具体：

“我想实现在线俄罗斯方块对战游戏。用什么方法传输对方的砖块信息比较合理呢？我想出了一种方法，不知是否合理。”

提问的范围越小、方式越具体越好。一个大问题可以分成若干个子问题，通过这样的提问可以思考和理解更多内容。

+ 后续学习内容

大部分学生对系统编程（System Programming）有着极深的恐惧感，特别是在没能理解其本质的情况下，这种感觉尤为深刻。但本书的绝大部分内容都与系统编程有关，也就是说，各位已经掌握了相当多的系统编程知识。

网络编程与操作系统有着密切的关系，IOCP和epoll就是典型的例子。因此，我们无法完全避免依赖于操作系统的代码。而编写依赖于操作系统的代码时，有必要深入学习系统编程。因此，如果各位想了解下一步应该学习哪些内容，我建议大家进一步学习Windows和Linux系统编程。学习过程中会加深对操作系统的理解，同时，系统编程知识可以应用到多种领域，可谓一举多得。操作系统是计算机专业的5大核心科目之一，学习系统编程必定有百利而无一害。

25.2 网络编程相关书籍介绍

最后推荐一些书目，各位可以将此视为我的主观建议。大家身边这种人越多越好，也希望各位多倾听其他前辈的意见。

+ 系统编程相关书籍

如前所述，最好多学习系统编程相关知识。下面介绍2本相关书籍。

《UNIX环境高级编程（第2版）》

这本书的作者是大名鼎鼎的W. Richard Stevens，由Addison Wesley出版。这是一本广为人知

的好书，如果您希望学习UNIX系列操作系统的系统编程，我向您强烈推荐。这本书的学习方法大致可分为2种：第一种方法是短时间内浏览一遍，通过这种快速阅读可以了解主要内容，并掌握各种系统级别（操作系统级别）的函数，浏览后可以参考需要的部分；第二种方法是从开始就把本书当作参考书，如果已经有了其他操作系统下的编程经验，可以通过这种方式提高学习效率。对于缺乏系统编程经验的读者，我还是推荐第一种方法。

《Windows核心编程》

本书的作者是Jeffrey Richter，由Microsoft Press出版。可以说本书与前一本书分别引领了Windows和UNIX领域的编程技术趋势。

+ 协议相关书籍

相信各位在学习网络编程时也有所体会，要想编好网络程序，需要深入理解TCP/IP协议。下面介绍与之相关的2本书。

《TCP/IP详解》（卷1~卷3）

本书的作者同样是W. Richard Stevens，也由Addison Wesley出版。毋庸赘言，《TCP/IP详解》系列极为有名，网络方向的研究生案头一般都会摆着这3本书（特别是卷1和卷2）。但就各位现在的水平而言，卷1已经足够了。卷1讲解的是TCP/IP协议，卷2和卷3主要介绍TCP/IP的代码实现和扩展的（Transactional）TCP协议。下面为初学者推荐另一本书。

《TCP/IP协议族》

本书的作者是Behrouz A. Forouzan，由McGraw Hill出版。其优点是语句简洁，并加入大量插图以帮助理解。这本书本身就浅显易懂，改版中又加入了最新的热点问题，不仅是学习TCP/IP协议的好书，而且涵盖了多种应用层协议。据我所知，不少大学都将此书用作教材。

所有学习到此结束，各位辛苦了！

感谢各位选择本书！

作者 尹圣雨敬上

索引

B

本地广播 237, 239
标准I/O函数 246, 252, 258
标准错误 9
标准输出 9, 222, 250
标准输入 9, 202, 224
并发服务器 155, 173, 192

C

CRITICAL_SECTION对象 329
传输（Transport）层 101

D

大端序（Big Endian） 42
单线程模型 318
地址分配错误（Binding Error） 146
地址再分配 149
端口号 3, 6, 9
断开连接 95, 100, 102
堆（Heap） 285
多播（Multicast） 230
多进程服务器端 155, 157, 194
多线程服务器端 284, 306, 339
多线程模型 317, 326, 394

E

二进制信号量 304, 334, 336

F

非阻塞I/O 370
非阻塞模式 278, 281, 357
分布式数据库系统, 130
分离I/O流 255
服务器端套接字 6, 8, 21
复制文件描述符 177, 216, 260

G

管道（PIPE） 183
广播（Broadcast） 236

H

函数调用相关规定 321
后台处理（Background Processing） 161
互斥量对象 331, 332
滑动窗口（Sliding Window） 92
缓冲 11, 13, 23
缓冲（buffer） 28
回声服务器端 72, 76, 80
回声客户端 72, 74, 76

I

I/O复用（Multiplexing） 194
I/O复用服务器端 196, 203, 207
I/O缓冲 91, 100, 140

J

基于事件对象的同步 336
 基于信号量对象的同步 334
 监听 (listening) 套接字 6
 僵尸 (Zombie) 进程 159
 进程 (Process) 156
 进程ID 157, 161, 164
 进程间双向通信 185
 进程间通信 183, 188, 193

K

开放式系统 (Open System) 61
 客户端套接字 6, 8, 69

L

连接请求等待队列 65, 68, 69
 链路层 62, 81
 临界区 (Critical Section) 292
 流 (Stream) 119
 流形成的状态 119
 路由 (Routing) 212, 231
 路由器 (Router) 37

M

manual-reset模式事件对象 348
 默认DNS服务器 128, 139

N

non-signaled状态 322, 331, 338
 内核模式同步 328, 331, 343

O

OSI 7 Layer (层) 61

Q

轻量级进程 285
 请求消息 (Request Message) 392
 全双工 (Full-duplex) 方式 93

S

事件对象 336, 347, 353
 事件同步对象 336
 受理连接请求 5, 6, 66
 数据边界 (Boundary) 28, 35
 数据区 285, 286

T

TCP/IP协议栈 59, 62, 81
 TCP/UDP层 63
 Time-wait状态 145, 150, 154
 套接字句柄 18, 23, 32
 套接字类型 26, 32, 142
 条件触发 (Level Trigger) 273
 条件触发的事件特性 274
 同步I/O 344
 同步消息 93
 同时访问 183, 261, 292

W

完成端口对象 379, 387
 网络编程 2, 4, 15
 网络地址 (Internet Address) 36
 网络地址 (网络ID) 37
 网络接口卡 38, 62
 网络流量 151, 230, 243
 网络字节序 (Network Byte Order) 43
 文件句柄 9, 10, 20
 文件描述符 (File Descriptor) 9

X

系统函数 20, 130, 247

线程 72, 156, 193
线程ID 287, 289, 307
响应消息 (Response Message) 393
小端序 (Little Endian) 42
协议 (Protocol) 26
协议族 (Protocol Family) 27
信号处理 (Signal Handling) 165
信号量对象 304, 334
虚拟网络 236, 243

Y
异步 (Asynchronous) 344
异步I/O 345, 370

异步通知I/O模型 344, 350, 352
应用层 64, 85, 390
应用层协议 64, 85, 87
用户模式 (User mode) 327
用户模式同步 328, 343
域名 128, 138

Z

栈 (Stack) 285
知名端口 (Well-known PORT) 39
重叠I/O 228, 357, 370
字节序 41, 58

欢迎加入

图灵社区 ituring.com.cn

——最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

优惠提示：现在购买电子书，读者将获赠书款20%的社区银子，可用于兑换纸质样书。

——最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版的梦想。利用“合集”功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要你有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

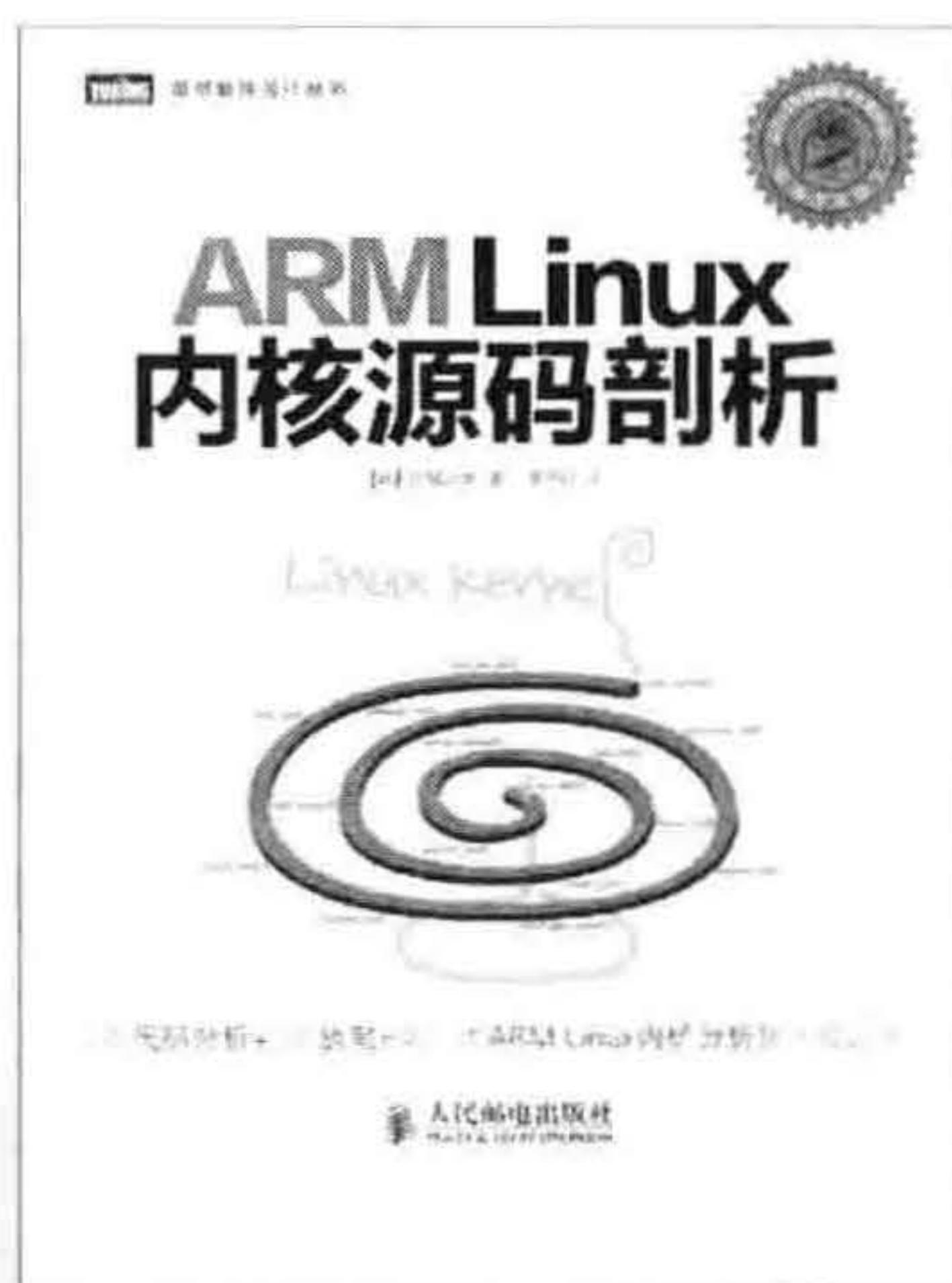
——最直接的读者交流平台

在图灵社区，你可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、乐译、评选等多种活动，赢取积分和银子，积累个人声望。



图灵最新重点图书



在本书是多位作者在 3 年 Linux 内核分析经验和庞大资料基础上写成的，收录了其他同类书未曾讲解的内容并进行逐行分析，一扫当前市场中其他理论书带给读者的郁闷。书中详细的代码分析与大量插图能够使读者对 Linux 内核及 ARM 获得正确认识，自然而然习得如何有效分析定期发布的 Linux 内核。

本书适合想从 Linux 内核启动开始透彻分析全部启动过程的读者，因 Linux 代码量庞大而束手无策的人、想要了解 Linux 实际运行过程的人、渴求 OS 实操理论的人，本书必将成为他们不可或缺的参考书。

ARM Linux 内核源码剖析

书号: 978-7-115-35910-0

作者: [韩] 尹锡训等著 崔范松译

定价: 99.00 元



逆向工程核心原理
书号: 978-7-115-35018-3
作者: [韩] 李承远著 武传海译
定价: 109.00 元



Python 计算机视觉编程
书号: 978-7-115-35232-3
作者: Jan Erik Solem
定价: 69.00 元



深入理解 Oracle 12c 数据库管理 (第 2 版)
书号: 978-7-115-35540-9
作者: Darr Kuhn
定价: 119.00 元



两周自制脚本语言
书号: 978-7-115-35564-5
作者: [日] 千叶滋著 陈筱烟译
定价: 59.00 元



机器学习系统设计
书号: 978-7-115-35682-6
作者: Willi Richert, Luis Pedro Coelho
定价: 49.00 元



Android 编程实战
书号: 978-7-115-35733-5
作者: Erik Hellman
定价: 69.00 元

TCP/IP网络编程

本书涵盖操作系统、系统编程、TCP/IP协议等多种内容，结构清晰、讲解细致、通俗易懂。此书面向利用套接字进行网络编程的初学者，具备C语言基础知识的读者会获得更多帮助。书中收录了丰富的示例，详细展现了Linux和Windows平台下套接字编程的共性与个性。特别是从代码角度说明了不同模型服务器端的区别，还包括了条件触发与边缘触发等知识，对开发实践也有很大帮助。

为初学者准备
的网络编程



本书结构

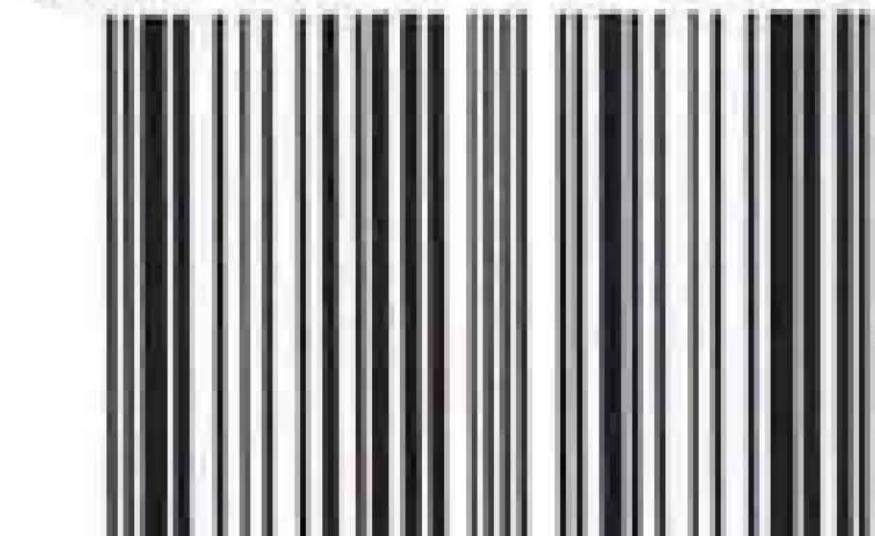
第一部分主要介绍网络编程基础知识。此部分主要论述Windows和Linux平台网络编程必备基础知识，未过多涉及不同操作系统特性。

第二部分和第三部分与操作系统有关。第二部分主要是Linux相关内容，而第三部分主要是Windows相关内容。从事Windows编程的朋友浏览第二部分内容后，同样可以提高技艺。

第四部分对全书内容进行总结，包含了作者在自身经验基础上总结的学习建议，还介绍了网络编程经典书籍。



ISBN 978-7-115-35885-1



9 787115 358851

ISBN 978-7-115-35885-1

定价：79.00元

图灵社区：iTuring.cn

热线：(010)51095186转600

分类建议 计算机/网络通信

人民邮电出版社网址：www.ptpress.com.cn