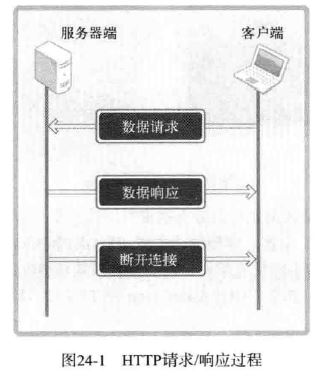
1. HTTP：Hypertext Transfer Protocol超文本传输协议。超文本是可以根据客户端请求而跳转的结构化信息。

HTTP协议是以超文本传输为目的而设计的应用层协议，是基于TCP/IP实现的协议，实现该协议就相当于实现了Web服务器端。

2. Http协议又称为无状态的stateless协议

从下图中可以看出服务器端响应客户端请求后立即断开，不会维持连接状态。即使同一个客户端再次发送请求，服务器端也无法辨认，而是以同等方式对待。

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。



3. 请求消息的结构

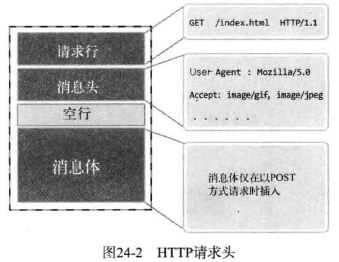
客户端向服务器端发送的请求：请求行、消息头、消息体。

请求行包含请求方式，GET或POST，请求行只能一行发送。图中含义请求GET index.html文件，希望以1.1版本的HTTP协议进行通信。

消息头：发送请求的浏览器信息，用户认证信息等HTTP附加信息。

消息体：需要以POST方式发送请求。

以空行分开。

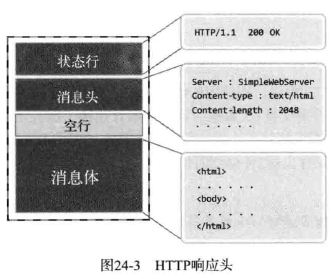


4. 相应消息的结构

服务器端向客户端传递的响应消息：状态行、消息头、消息体

状态行：客户端请求的处理结果，图中的含义，Http 1.1版本相应，你的请求已正确处理200 OK。

消息头：数据类型、长度等。图中，服务器名simplewebserver，数据类型text/html，数据长度不超过2048.



1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <string.h>
5. #include <arpa/inet.h>
6. #include <sys/socket.h>
7. #include <pthread.h>
9. #define BUF\_SIZE    1024
10. #define SMALL\_BUF   100
12. **void** \* request\_handler(**void** \* arg);
13. **void** send\_data(**FILE** \* fp, **char** \* ct, **char** \* file\_name);
14. **char** \* content\_type(**char** \* file);
15. **void** send\_error(**FILE** \* fp);
16. **void** error\_handling(**char** \* message);
18. **int** main(**int** argc, **char** \* argv[]){
19. **int** serv\_sock,clnt\_sock;
20. **struct** sockaddr\_in serv\_adr,clnt\_adr;
21. **int** clnt\_adr\_size;
22. **char** buf[BUF\_SIZE];
23. pthread\_t t\_id;
25. **if**(argc != 2){
26. printf("Usage : %s <port> \n",argv[0]);
27. exit(1);
28. }
30. serv\_sock = socket(PF\_INET,SOCK\_STREAM,0);
32. memset(&serv\_adr, 0, **sizeof**(serv\_adr));
34. serv\_adr.sin\_family = AF\_INET;
35. serv\_adr.sin\_addr.s\_addr = htonl(INADDR\_ANY);
36. serv\_adr.sin\_port = htons(atoi(argv[1]));
38. **if**(bind(serv\_sock,(**struct** sockaddr \*)&serv\_adr,**sizeof**(serv\_adr)) == -1){
39. error\_handling("bind error");
40. }
42. **if**(listen(serv\_sock,20) == -1){
43. error\_handling("listen error");
44. }
46. **while**(1){
47. clnt\_adr\_size = **sizeof**(clnt\_adr);
48. clnt\_sock = accept(serv\_sock,(**struct** sockaddr \*)&clnt\_adr,&clnt\_adr\_size);
50. printf("connection request : %s %d\n",inet\_ntoa(clnt\_adr.sin\_addr),ntohs(clnt\_adr.sin\_port));
52. pthread\_create(&t\_id,NULL,request\_handler,&clnt\_sock);
54. pthread\_detach(t\_id);
55. }
57. close(serv\_sock);
59. **return** 0;
60. }

63. **void** \* request\_handler(**void** \* arg){
64. **int** clnt\_sock = \*((**int** \*)arg);
65. **char** req\_line[SMALL\_BUF];
66. **FILE** \* clnt\_read;
67. **FILE** \* clnt\_write;
68. **char** method[10];
69. **char** ct[15];
70. **char** file\_name[30];
72. clnt\_read = fdopen(clnt\_sock,"r");
73. clnt\_write = fdopen(dup(clnt\_sock),"w");
74. fgets(req\_line,SMALL\_BUF,clnt\_read);
76. **if**(strstr(req\_line,"HTTP/") == NULL){
77. send\_error(clnt\_write);
78. fclose(clnt\_read);
79. fclose(clnt\_write);
80. **return** NULL;
81. }
83. strcpy(method,strtok(req\_line," /"));
84. strcpy(file\_name,strtok(NULL," /"));
85. strcpy(ct,content\_type(file\_name));
86. **if**(strcmp(method,"GET") != 0){
87. send\_error(clnt\_write);
88. fclose(clnt\_read);
89. fclose(clnt\_write);
90. **return** NULL;
91. }
93. fclose(clnt\_read);
95. send\_data(clnt\_write,ct,file\_name);
96. }
98. **void** send\_data(**FILE** \*fp,**char** \*ct,**char** \*file\_name){
99. **char** protocol[] = "HTTP/1.0 200 OK\r\n";
100. **char** server[] = "Server:Linux Web Server \r\n";
101. **char** cnt\_len[] = "Content-length:2048\r\n";
102. **char** cnt\_type[SMALL\_BUF];
103. **char** buf[BUF\_SIZE];
104. **FILE** \* send\_file;
106. sprintf(cnt\_type,"Content-type:%s\r\n\r\n",ct);
107. send\_file = fopen(file\_name,"r");
108. **if**(send\_file == NULL){
109. send\_error(fp);
110. **return**;
111. }
113. //send head
114. fputs(protocol,fp);
115. fputs(server,fp);
116. fputs(cnt\_len,fp);
117. fputs(cnt\_type,fp);
119. **while**(fgets(buf,BUF\_SIZE,send\_file) != NULL){
120. fputs(buf,fp);
121. fflush(fp);
122. }
124. fflush(fp);
125. fclose(fp);
126. }
128. **char** \* content\_type(**char** \*file){
129. **char** extension[SMALL\_BUF];
130. **char** file\_name[SMALL\_BUF];
132. strcpy(file\_name,file);
133. strtok(file\_name,".");
134. strcpy(extension,strtok(NULL,"."));
136. **if**(!strcmp(extension,"html") || !strcmp(extension,"htm")){
137. **return** "text/html";
138. }**else**{
139. **return** "text/plain";
140. }
141. }
143. **void** send\_error(**FILE** \* fp){
144. **char** protocol[] = "HTTP/1.0 400 Bad Request\r\n";
145. **char** server[] = "Server:Linux Web Server \r\n";
146. **char** cnt\_len[] = "Content-length:2048\r\n";
147. **char** cnt\_type[] = "Content-type:text/html\r\n\r\n";
148. **char** content[] = "<html><head><title>NETWORK</title></head>"
149. "<body><font size=+5><br>br error filename request!"
150. "</font><body><html>";
152. fputs(protocol,fp);
153. fputs(server,fp);
154. fputs(cnt\_len,fp);
155. fputs(cnt\_type,fp);
157. fflush(fp);
158. }
160. **void** error\_handling(**char** \* message){
161. fputs(message,stderr);
162. fputs("\n",stderr);
163. exit(1);
164. }