# C/C++从入门到精通-高级程序员之路

**【服务器进阶篇】**

# 写在前面的话

* 有C语言基础就可以跟着我来学习并掌握这个项目中的全部知识点
* 我们的项目是在Linux 环境下开发的，如果没有Linux 基础，可以先学习本B站视频 - Linux 环境快速搭建+极速入门:

<https://www.bilibili.com/video/BV1cL41137H1/>

* 学习http项目同时需要B友们具有网络编程基础, 如果没有请学习如下视频

网络通信与Socket 编程： <https://www.bilibili.com/video/BV14Y411s7yB/>

* 源码+笔记+技术交流，请加QQ群： 701751885



# 第六部分 实现高并发http 服务器

## 项目需求

实现一个http 服务器项目，服务器启动后监听80端口的tcp 连接，当用户通过任意一款浏览器（IE、火狐和腾讯浏览器等）访问我们的http服务器，http服务器会查找用户访问的html页面是否存在，如果存在则通过http 协议响应客户端的请求，把页面返回给浏览器，浏览器显示html页面；如果页面不存在，则按照http 协议的规定，通知浏览器此页面不存在（404 NOT FOUND）

## 需求分析

1. **何为Html 页面**

**html**，全称Hypertext Markup Language，也就是“超文本链接标示语言”。HTML文本是由 HTML命令组成的描述性文本，HTML 命令可以说明文字、 图形、动画、声音、表格、链接等。 即平常上网所看到的的网页。

demo.html

|  |
| --- |
| <html lang=\"zh-CN\">  <head>  <meta content=\"text/html; charset=utf-8\" http-equiv=\"Content-Type\">  <title>This is a test</title>  </head>  <body>  <div align=center height=\"500px\" >  <br/><br/><br/>  <h2>大家好，欢迎来到奇牛学院VIP 课！</h2><br/><br/>  <form action="commit" method="post">  尊姓大名: <input type="text" name="name" />  <br/>芳龄几何: <input type="password" name="age" />  <br/><br/><br/><input type="submit" value="提交" />  <input type="reset" value="重置" />  </form>  </div>  </body>  </html> |

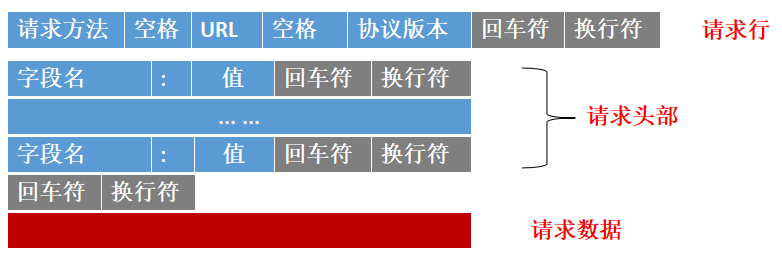
1. **何为http 协议**

HTTP协议是Hyper Text Transfer Protocol(超文本传输协议)的缩写,是用于从万维网(WWW:World Wide Web )服务器传输超文本到本地浏览器的传送协议。

请求格式：

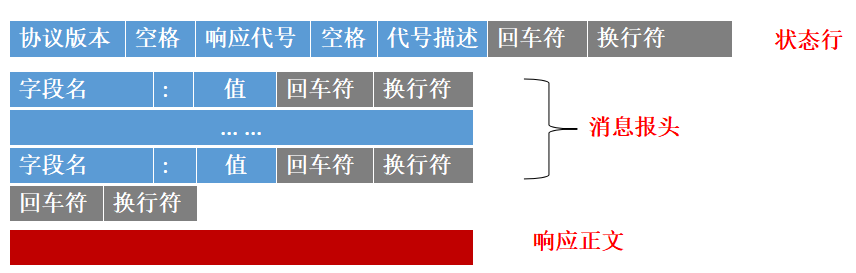
**客户端请求**

客户端发送一个HTTP请求到服务器的请求消息包括以下格式：请求行（request line）、请求头部（header）、空行和请求数据四个部分组成，下图给出了请求报文的一般格式。



**服务端响应**

服务器响应客户端的HTTP响应也由四个部分组成，分别是：状态行、消息报头、空行和响应正文。



|  |  |  |
| --- | --- | --- |
|  | 响应代号 | 代号描述 |
| 服务器上存在请求的内容，并可以响应给客户端 | 200 | OK |
| 客户端的请求有异常，方法有问题 | 501 | Method Not Implemented |
| 服务器收到请求后，因为自生的问题没法响应 | 500 | Internal Server Error |
| 请求的内容不存在 | 404 | NOT FOUND |
| 客户端发送的请求格式有问题等 | 400 | BAD REQUEST |

Demo

|  |
| --- |
| **浏览器请求：**  GET /demo.html HTTP/1.1  Host: 47.100.162.191  Connection: keep-alive  User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.26 Safari/537.36 Core/1.63.6788.400 QQBrowser/10.3.2767.400  Upgrade-Insecure-Requests: 1  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8  Accept-Encoding: gzip, deflate  Accept-Language: zh-CN,zh;q=0.9  Cookie:cna=BT0+EoVi1FACAXH3Nv5I7h6k;isg=BIOD99I03BNYvZDfE2FJUOsMB0ftUBZcBFi4E7VgYOJZdKOWPcvRinAl6kSfVG8y  **服务器响应：**  HTTP/1.0 200 OK  Server: Martin Server  Content-Type: text/html  Connection: Close  Content-Length: 526  <html lang="zh-CN">  <head>  <meta content="text/html; charset=utf-8" http-equiv="Content-Type">  <title>This is a test</title>  </head>  <body>  <div align=center height="500px" >  <br/><br/><br/>  <h2>大家好，欢迎来到奇牛学院VIP 试听课！</h2><br/><br/>  <form action="commit" method="post">  尊姓大名: <input type="text" name="name" />  <br/>芳龄几何: <input type="password" name="age" />  <br/><br/><br/><input type="submit" value="提交" />  <input type="reset" value="重置" />  </form>  </div>  </body>  </html> |

**实现Mini型http 服务器**

小马同学心猿意马，心想，Echo 服务器我已经学会了，不就是从客户端收信息，服务器再响应信息嘛，现在我又知道了http 协议，html文本自己搞不定就直接用老马老师的demo.html撒，现在我只要把浏览器发送的http 请求按照http 协议的格式进行解析，获取到浏览器想要访问的html 是哪个文本，然后按照http 响应的格式把html 文本响应给客户端，不就。。。。。 嘿嘿嘿。。。。QQ图片20181130170126

接收http请求

解析http请求

响应http请求

**接收http请求**

1. **实现按行读取请求头部**

|  |
| --- |
| //返回值： -1 表示读取出错， 等于0表示读到一个空行， 大于0 表示成功读取一行  int get\_line(int sock, char \*buf, int size){  int count = 0;  char ch = '\0';  int len = 0;      while( (count<size - 1) && ch!='\n'){  len = read(sock, &ch, 1);    if(len == 1){  if(ch == '\r'){  continue;  }else if(ch == '\n'){  //buf[count] = '\0';  break;  }    //这里处理一般的字符  buf[count] = ch;  count++;    }else if( len == -1 ){//读取出错  perror("read failed");  count = -1;  break;  }else {// read 返回0,客户端关闭sock 连接.  fprintf(stderr, "client close.\n");  count = -1;  break;  }  }    if(count >= 0) buf[count] = '\0';    return count;  } |

**思考题?**

如果客户端发送的请求是： GET /baidu.html HTTP/1.1’\r’Host: 47.100.162.191’\r’’\n’ Connection: keep-alive’\r’’\n’ ,

我们仍然想get\_line返回正确的三行，又该怎么写代码呢？

(**注意： ‘\r’ 代表 回车符，Ascii码是13 ,’\n’代表的是换行符，Ascii码是14**)

**2. 如果碰到两个连续的回车换行，即，意味着请求头部结束**

**解析请求**

|  |
| --- |
| //1.读取请求行  void do\_http\_request(int client\_sock){  int len = 0;  char buf[256];  char method[64];  char url[256];  char path[256];      /\*读取客户端发送的http 请求\*/      //1.读取请求行  len = get\_line(client\_sock, buf, sizeof(buf));    if(len > 0){//读到了请求行  int i=0, j=0;  while(!isspace(buf[j]) && (i<sizeof(method)-1)){  method[i] = buf[j];  i++;  j++;  }    method[i] = '\0';  if(debug) printf("request method: %s\n", method);    if(strncasecmp(method, "GET", i)==0){ //只处理get请求  if(debug) printf("method = GET\n");    //获取url  while(isspace(buf[j++]));//跳过白空格  i = 0;    while(!isspace(buf[j]) && (i<sizeof(url)-1)){  url[i] = buf[j];  i++;  j++;  }    url[i] = '\0';    if(debug) printf("url: %s\n", url);    //继续读取http 头部  do{  len = get\_line(client\_sock, buf, sizeof(buf));  if(debug) printf("read: %s\n", buf);    }while(len>0);    //\*\*\*定位服务器本地的html文件\*\*\*    //处理url 中的?  {  char \*pos = strchr(url, '?');  if(pos){  \*pos = '\0';  printf("real url: %s\n", url);  }  }    sprintf(path, "./html\_docs/%s", url);  if(debug) printf("path: %s\n", path);    //执行http 响应    }else {  //非get请求, 读取http 头部，并响应客户端 501 Method Not Implemented  fprintf(stderr, "warning! other request [%s]\n", method);  do{  len = get\_line(client\_sock, buf, sizeof(buf));  if(debug) printf("read: %s\n", buf);    }while(len>0);    //unimplemented(client\_sock); //在响应时再实现    }            }else {//请求格式有问题，出错处理  //bad\_request(client\_sock); //在响应时再实现  }            } |

**响应http 请求**

|  |
| --- |
| void do\_http\_request(int client\_sock){  int len=0;  char buf[256];  char method[16];  char url[256];  /\*读取客户端发送的http请求\*/  //1.读取请求行  len = get\_line(client\_sock, buf, sizeof(buf));  if(len>0){  int i, j;  while(!isspace(buf[j]) && ( i< sizeof(method)-1)){  method[i]=buf[j];  i++;  j++;  }  method[i]='\0';  //判断方法是否合法  if(strncasecmp(method, "GET", i)==0){//GET 方法  printf("requst = %s\n", method);  //获取url  while(isspace(buf[++j]));  i=-1;  while(!isspace(buf[j]) && ( i< sizeof(url)-1)){  url[i]=buf[j];  i++;  j++;  }  url[i]='\0';  printf("url: %s\n", url);  //读取http 头部，不做任何处理  do{  len = get\_line(client\_sock, buf, sizeof(buf));  printf("read line: %s\n", buf);  }while(len > 0);  do\_http\_response(client\_sock);  }else {  printf("other requst = %s\n", method);  //读取http 头部，不做任何处理  do{  len = get\_line(client\_sock, buf, sizeof(buf));  printf("read line: %s\n", buf);  }while(len > 0);  }  }else {//出错的处理  }  } |
| void do\_http\_response(int client\_sock){  const char \*main\_header = "HTTP/1.0 200 OK\r\nServer: Martin Server\r\nContent-Type: text/html\r\nConnection: Close\r\n";  const char \* welcome\_content = "\  <html lang=\"zh-CN\">\n\  <head>\n\  <meta content=\"text/html; charset=utf-8\" http-equiv=\"Content-Type\">\n\  <title>This is a test</title>\n\  </head>\n\  <body>\n\  <div align=center height=\"500px\" >\n\  <br/><br/><br/>\n\  <h2>大家好，欢迎来到奇牛学院VIP 试听课！</h2><br/><br/>\n\  <form action=\"commit\" method=\"post\">\n\  尊姓大名: <input type=\"text\" name=\"name\" />\n\  <br/>芳龄几何: <input type=\"password\" name=\"age\" />\n\  <br/><br/><br/><input type=\"submit\" value=\"提交\" />\n\  <input type=\"reset\" value=\"重置\" />\n\  </form>\n\  </div>\n\  </body>\n\  </html>";  char send\_buf[64];  int wc\_len = strlen(welcome\_content);  int len = write(client\_sock, main\_header, strlen(main\_header));  if(debug) fprintf(stdout, "... do\_http\_response...\n");  if(debug) fprintf(stdout, "write[%d]: %s", len, main\_header);  len =snprintf(send\_buf, 64,"Content-Length: %d\r\n\r\n", wc\_len);  len = write(client\_sock, send\_buf, len);  if(debug) fprintf(stdout, "write[%d]: %s", len, send\_buf);  len = write(client\_sock, welcome\_content, wc\_len);  if(debug) fprintf(stdout, "write[%d]: %s", len, welcome\_content);  } |

501 http 响应

|  |
| --- |
| HTTP/1.0 501 Method Not Implemented\r\ const char \*main\_header = "HTTP/1.0 200 OK\r\nServer: Martin Server\r\nContent-Type: text/html\r\nConnection: Close\r\n";  const char \* welcome\_content = "\  <html lang=\"zh-CN\">\n\  <head>\n\  <meta content=\"text/html; charset=utf-8\" http-equiv=\"Content-Type\">\n\  <title>This is a test</title>\n\  </head>\n\  <body>\n\  <div align=center height=\"500px\" >\n\  <br/><br/><br/>\n\  <h2>大家好，欢迎来到奇牛学院VIP 试听课！</h2><br/><br/>\n\  <form action=\"commit\" method=\"post\">\n\  尊姓大名: <input type=\"text\" name=\"name\" />\n\  <br/>芳龄几何: <input type=\"password\" name=\"age\" />\n\  <br/><br/><br/><input type=\"submit\" value=\"提交\" />\n\  <input type=\"reset\" value=\"重置\" />\n\  </form>\n\  </div>\n\  </body>\n\  </html>";  Content-Type: text/html\r\n  \r\n  <HTML>  <HEAD>  <TITLE>Method Not Implemented</TITLE>  </HEAD>  <BODY>  <P>HTTP request method not supported.  </BODY>  </HTML> |

404 http 响应

|  |
| --- |
| HTTP/1.0 404 NOT FOUND\r\n  Content-Type: text/html\r\n  \r\n  <HTML>  <HEAD>  <TITLE>NOT FOUND</TITLE>  </HEAD>  <BODY>  <P>The server could not fulfill your request because the resource specified is unavailable or nonexistent.  </BODY>  </HTML> |

400 http 响应 - BAD REQUEST

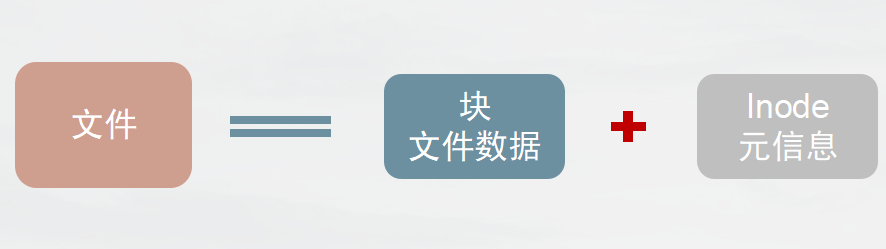
|  |
| --- |
| HTTP/1.0 400 BAD REQUEST\r\n  Content-Type: text/html\r\n  \r\n  <HTML>  <HEAD>  <TITLE>BAD REQUEST</TITLE>  </HEAD>  <BODY>  <P>Your browser sent a bad request！  </BODY>  </HTML> |

500 http 响应 - 内部服务器错误

|  |
| --- |
| HTTP/1.0 500 Internal Sever Error\r\n  Content-Type: text/html\r\n  \r\n  <HTML>  <HEAD>  <TITLE>Method Not Implemented</TITLE>  </HEAD>  <BODY>  <P>Error prohibited CGI execution.  </BODY>  </HTML> |

**读取文件**

**文件概念简介**



inode - "索引节点",储存文件的元信息，比如文件的创建者、文件的创建日期、文件的大小等等。每个inode都有一个号码，操作系统用inode号码来识别不同的文件。ls -i 查看inode 号

##### stat函数

**作用：**返回文件的状态信息

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

int stat(const char \*path, struct stat \*buf);

int fstat(int fd, struct stat \*buf);

int lstat(const char \*path, struct stat \*buf);

path:

文件的路径

buf:

传入的保存文件状态的指针，用于保存文件的状态

返回值：

成功返回0，失败返回-1，设置errno

struct stat {

dev\_t st\_dev; /\* ID of device containing file \*/

ino\_t st\_ino; /\* inode number \*/

mode\_t st\_mode; /\* **S\_ISREG(st\_mode) 是一个普通文件 S\_ISDIR(st\_mode) 是一个目录**\*/

nlink\_t st\_nlink; /\* number of hard links \*/

uid\_t st\_uid; /\* user ID of owner \*/

gid\_t st\_gid; /\* group ID of owner \*/

dev\_t st\_rdev; /\* device ID (if special file) \*/

off\_t st\_size; /\* total size, in bytes \*/

blksize\_t st\_blksize; /\* blocksize for filesystem I/O \*/

blkcnt\_t st\_blocks; /\* number of 512B blocks allocated \*/

time\_t st\_atime; /\* time of last access \*/

time\_t st\_mtime; /\* time of last modification \*/

time\_t st\_ctime; /\* time of last status change \*/

};

**当前阶段完整代码实现**：

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/socket.h>  #include <string.h>  #include <ctype.h>  #include <arpa/inet.h>  #include <errno.h>  #include <sys/stat.h>  #include <unistd.h>  #define SERVER\_PORT 80  static int debug = 1;  int get\_line(int sock, char \*buf, int size);  void do\_http\_request(int client\_sock);  void do\_http\_response(int client\_sock, const char \*path);  int headers(int client\_sock, FILE \*resource);  void cat(int client\_sock, FILE \*resource);  void not\_found(int client\_sock);  void inner\_error(int client\_sock);  int main(void){  int sock;//代表信箱  struct sockaddr\_in server\_addr;  //1.美女创建信箱  sock = socket(AF\_INET, SOCK\_STREAM, 0);  //2.清空标签，写上地址和端口号  bzero(&server\_addr, sizeof(server\_addr));  server\_addr.sin\_family = AF\_INET;//选择协议族IPV4  server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);//监听本地所有IP地址  server\_addr.sin\_port = htons(SERVER\_PORT);//绑定端口号  //实现标签贴到收信得信箱上  bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));  //把信箱挂置到传达室，这样，就可以接收信件了  listen(sock, 128);  //万事俱备，只等来信  printf("等待客户端的连接\n");  int done =1;  while(done){  struct sockaddr\_in client;  int client\_sock, len, i;  char client\_ip[64];  char buf[256];  socklen\_t client\_addr\_len;  client\_addr\_len = sizeof(client);  client\_sock = accept(sock, (struct sockaddr \*)&client, &client\_addr\_len);  //打印客服端IP地址和端口号  printf("client ip: %s\t port : %d\n",  inet\_ntop(AF\_INET, &client.sin\_addr.s\_addr,client\_ip,sizeof(client\_ip)),  ntohs(client.sin\_port));  /\*处理http 请求,读取客户端发送的数据\*/  do\_http\_request(client\_sock);  close(client\_sock);  }  close(sock);  return 0;  }  void do\_http\_request(int client\_sock){  int len = 0;  char buf[256];  char method[64];  char url[256];  char path[256];    struct stat st;      /\*读取客户端发送的http 请求\*/      //1.读取请求行  len = get\_line(client\_sock, buf, sizeof(buf));    if(len > 0){//读到了请求行  int i=0, j=0;  while(!isspace(buf[j]) && (i<sizeof(method)-1)){  method[i] = buf[j];  i++;  j++;  }    method[i] = '\0';  if(debug) printf("request method: %s\n", method);    if(strncasecmp(method, "GET", i)==0){ //只处理get请求  if(debug) printf("method = GET\n");    //获取url  while(isspace(buf[j++]));//跳过白空格  i = 0;    while(!isspace(buf[j]) && (i<sizeof(url)-1)){  url[i] = buf[j];  i++;  j++;  }    url[i] = '\0';    if(debug) printf("url: %s\n", url);    //继续读取http 头部  do{  len = get\_line(client\_sock, buf, sizeof(buf));  if(debug) printf("read: %s\n", buf);    }while(len>0);    //\*\*\*定位服务器本地的html文件\*\*\*    //处理url 中的?  {  char \*pos = strchr(url, '?');  if(pos){  \*pos = '\0';  printf("real url: %s\n", url);  }  }    sprintf(path, "./html\_docs/%s", url);  if(debug) printf("path: %s\n", path);    //执行http 响应  //判断文件是否存在，如果存在就响应200 OK，同时发送相应的html 文件,如果不存在，就响应 404 NOT FOUND.  if(stat(path, &st)==-1){//文件不存在或是出错  fprintf(stderr, "stat %s failed. reason: %s\n", path, strerror(errno));  not\_found(client\_sock);  }else {//文件存在    if(S\_ISDIR(st.st\_mode)){  strcat(path, "/index.html");  }    do\_http\_response(client\_sock, path);    }          }else {//非get请求, 读取http 头部，并响应客户端 501 Method Not Implemented  fprintf(stderr, "warning! other request [%s]\n", method);  do{  len = get\_line(client\_sock, buf, sizeof(buf));  if(debug) printf("read: %s\n", buf);    }while(len>0);    //unimplemented(client\_sock); //在响应时再实现    }            }else {//请求格式有问题，出错处理  //bad\_request(client\_sock); //在响应时再实现  }            }  void do\_http\_response(int client\_sock, const char \*path){  int ret = 0;  FILE \*resource = NULL;    resource = fopen(path, "r");    if(resource == NULL){  not\_found(client\_sock);  return ;  }    //1.发送http 头部  ret = headers(client\_sock, resource);    //2.发送http body .  if(!ret){  cat(client\_sock, resource);  }    fclose(resource);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*返回关于响应文件信息的http 头部  \*输入：  \* client\_sock - 客服端socket 句柄  \* resource - 文件的句柄  \*返回值： 成功返回0 ，失败返回-1  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int headers(int client\_sock, FILE \*resource){  struct stat st;  int fileid = 0;  char tmp[64];  char buf[1024]={0};  strcpy(buf, "HTTP/1.0 200 OK\r\n");  strcat(buf, "Server: Martin Server\r\n");  strcat(buf, "Content-Type: text/html\r\n");  strcat(buf, "Connection: Close\r\n");    fileid = fileno(resource);    if(fstat(fileid, &st)== -1){  inner\_error(client\_sock);  return -1;  }    snprintf(tmp, 64, "Content-Length: %ld\r\n\r\n", st.st\_size);  strcat(buf, tmp);    if(debug) fprintf(stdout, "header: %s\n", buf);    if(send(client\_sock, buf, strlen(buf), 0)<0){  fprintf(stderr, "send failed. data: %s, reason: %s\n", buf, strerror(errno));  return -1;  }    return 0;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*说明：实现将html文件的内容按行  读取并送给客户端  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void cat(int client\_sock, FILE \*resource){  char buf[1024];    fgets(buf, sizeof(buf), resource);    while(!feof(resource)){  int len = write(client\_sock, buf, strlen(buf));    if(len<0){//发送body 的过程中出现问题,怎么办？1.重试？ 2.  fprintf(stderr, "send body error. reason: %s\n", strerror(errno));  break;  }    if(debug) fprintf(stdout, "%s", buf);  fgets(buf, sizeof(buf), resource);    }  }  void do\_http\_response1(int client\_sock){  const char \*main\_header = "HTTP/1.0 200 OK\r\nServer: Martin Server\r\nContent-Type: text/html\r\nConnection: Close\r\n";  const char \* welcome\_content = "\  <html lang=\"zh-CN\">\n\  <head>\n\  <meta content=\"text/html; charset=utf-8\" http-equiv=\"Content-Type\">\n\  <title>This is a test</title>\n\  </head>\n\  <body>\n\  <div align=center height=\"500px\" >\n\  <br/><br/><br/>\n\  <h2>大家好，欢迎来到奇牛学院VIP 试听课！</h2><br/><br/>\n\  <form action=\"commit\" method=\"post\">\n\  尊姓大名: <input type=\"text\" name=\"name\" />\n\  <br/>芳龄几何: <input type=\"password\" name=\"age\" />\n\  <br/><br/><br/><input type=\"submit\" value=\"提交\" />\n\  <input type=\"reset\" value=\"重置\" />\n\  </form>\n\  </div>\n\  </body>\n\  </html>";  //1. 发送main\_header  int len = write(client\_sock, main\_header, strlen(main\_header));    if(debug) fprintf(stdout, "... do\_http\_response...\n");  if(debug) fprintf(stdout, "write[%d]: %s", len, main\_header);    //2. 生成Content-Length  char send\_buf[64];  int wc\_len = strlen(welcome\_content);  len = snprintf(send\_buf, 64, "Content-Length: %d\r\n\r\n", wc\_len);  len = write(client\_sock, send\_buf, len);    if(debug) fprintf(stdout, "write[%d]: %s", len, send\_buf);    len = write(client\_sock, welcome\_content, wc\_len);  if(debug) fprintf(stdout, "write[%d]: %s", len, welcome\_content);  }  //返回值： -1 表示读取出错， 等于0表示读到一个空行， 大于0 表示成功读取一行  int get\_line(int sock, char \*buf, int size){  int count = 0;  char ch = '\0';  int len = 0;      while( (count<size - 1) && ch!='\n'){  len = read(sock, &ch, 1);    if(len == 1){  if(ch == '\r'){  continue;  }else if(ch == '\n'){  //buf[count] = '\0';  break;  }    //这里处理一般的字符  buf[count] = ch;  count++;    }else if( len == -1 ){//读取出错  perror("read failed");  count = -1;  break;  }else {// read 返回0,客户端关闭sock 连接.  fprintf(stderr, "client close.\n");  count = -1;  break;  }  }    if(count >= 0) buf[count] = '\0';    return count;  }  void not\_found(int client\_sock){  const char \* reply = "HTTP/1.0 404 NOT FOUND\r\n\  Content-Type: text/html\r\n\  \r\n\  <HTML lang=\"zh-CN\">\r\n\  <meta content=\"text/html; charset=utf-8\" http-equiv=\"Content-Type\">\r\n\  <HEAD>\r\n\  <TITLE>NOT FOUND</TITLE>\r\n\  </HEAD>\r\n\  <BODY>\r\n\  <P>文件不存在！\r\n\  <P>The server could not fulfill your request because the resource specified is unavailable or nonexistent.\r\n\  </BODY>\r\n\  </HTML>";  int len = write(client\_sock, reply, strlen(reply));  if(debug) fprintf(stdout, reply);    if(len <=0){  fprintf(stderr, "send reply failed. reason: %s\n", strerror(errno));  }      }  void inner\_error(int client\_sock){  const char \* reply = "HTTP/1.0 500 Internal Sever Error\r\n\  Content-Type: text/html\r\n\  \r\n\  <HTML lang=\"zh-CN\">\r\n\  <meta content=\"text/html; charset=utf-8\" http-equiv=\"Content-Type\">\r\n\  <HEAD>\r\n\  <TITLE>Inner Error</TITLE>\r\n\  </HEAD>\r\n\  <BODY>\r\n\  <P>服务器内部出错.\r\n\  </BODY>\r\n\  </HTML>";  int len = write(client\_sock, reply, strlen(reply));  if(debug) fprintf(stdout, reply);    if(len <=0){  fprintf(stderr, "send reply failed. reason: %s\n", strerror(errno));  }  } |

**并发处理**

**并发概述**

通俗的**并发**通常是指同时能并行的处理多个任务。



**并发**

同时拥有两个或者多个线程，如果程序在单核处理器上运行，多个线程将交替的换入或者换出内存，这些线程是同时“存在”的。  
每个线程都处于执行过程中的某个状态，如果运行在多核处理器上，此时，程序中的每个线程都将分配到一个处理器核上，因此可以同时运行。

**高并发**

高并发是互联网分布式系统架构设计中必须考虑的因素之一，它通常是指，通过设计保证系统能够 **同时并行处理** 很多请求

**pthread\_create函数**

创建一个新线程，并行的执行任务。

#include <pthread.h>

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine) (void \*), void \*arg);

返回值：成功：0； 失败：错误号。

参数：

pthread\_t：当前Linux中可理解为：typedef unsigned long int pthread\_t;

参数1：传出参数，保存系统为我们分配好的线程ID

参数2：通常传NULL，表示使用线程默认属性。若想使用具体属性也可以修改该参数。

参数3：函数指针，指向线程主函数(线程体)，该函数运行结束，则线程结束。

参数4：线程主函数执行期间所使用的参数。

在一个线程中调用pthread\_create()创建新的线程后，当前线程从pthread\_create()返回继续往下执行，而新的线程所执行的代码由我们传给pthread\_create的函数指针start\_routine决定。start\_routine函数接收一个参数，是通过pthread\_create的arg参数传递给它的，该参数的类型为void \*，这个指针按什么类型解释由调用者自己定义。start\_routine的返回值类型也是void \*，这个指针的含义同样由调用者自己定义。start\_routine返回时，这个线程就退出了，其它线程可以调用pthread\_join得到start\_routine的返回值，以后再详细介绍pthread\_join。

pthread\_create成功返回后，新创建的线程的id被填写到thread参数所指向的内存单元。

attr参数表示线程属性，本节不深入讨论线程属性，所有代码例子都传NULL给attr参数，表示线程属性取缺省值。

**并发回声服务器改造**

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/socket.h>  #include <string.h>  #include <ctype.h>  #include <arpa/inet.h>  #include <pthread.h>  #include <stdlib.h>  #include <errno.h>  #define SERVER\_PORT 666  void \* thread\_handleRequest(void\* arg);  int main(void){  int sock;//代表信箱  struct sockaddr\_in server\_addr;  //1.美女创建信箱  sock = socket(AF\_INET, SOCK\_STREAM, 0);  //2.清空标签，写上地址和端口号  bzero(&server\_addr, sizeof(server\_addr));  server\_addr.sin\_family = AF\_INET;//选择协议族IPV4  server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);//监听本地所有IP地址  server\_addr.sin\_port = htons(SERVER\_PORT);//绑定端口号  //实现标签贴到收信得信箱上  bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));  //把信箱挂置到传达室，这样，就可以接收信件了  listen(sock, 128);  //万事俱备，只等来信  printf("等待客户端的连接\n");  int done =1;  while(done){  struct sockaddr\_in client;  int client\_sock, len;  char client\_ip[64];  //char buf[256];  //int i=0;  socklen\_t client\_addr\_len;  client\_addr\_len = sizeof(client);  client\_sock = accept(sock, (struct sockaddr \*)&client, &client\_addr\_len);  //打印客服端IP地址和端口号  printf("client ip: %s\t port : %d\n",  inet\_ntop(AF\_INET, &client.sin\_addr.s\_addr,client\_ip,sizeof(client\_ip)),  ntohs(client.sin\_port));    //启动线程，完成和和客户端的交互  {  pthread\_t tid;  int \*ptr\_int = NULL;  int err = 0;    ptr\_int = (int \*)malloc(sizeof(int));  \*ptr\_int = client\_sock;    if(err = pthread\_create(&tid, NULL, thread\_handleRequest, (void\*)ptr\_int)){  fprintf(stderr, "Can't create thread, reason: %s\n", strerror(errno));  if(ptr\_int) free(ptr\_int);  }    }    /\*读取客户端发送的数据\*/  /\*len = read(client\_sock, buf, sizeof(buf)-1);  buf[len] = '\0';  printf("receive[%d]: %s\n", len, buf);  //转换成大写  for(i=0; i<len; i++){  //if(buf[i]>='a' && buf[i]<='z'){  // buf[i] = buf[i] - 32;  //}  buf[i] = toupper(buf[i]);  }  len = write(client\_sock, buf, len);  printf("finished. len: %d\n", len);  close(client\_sock);  \*/  }  close(sock);  return 0;  }  void \* thread\_handleRequest(void\* arg){  int client\_sock = \*(int \*)arg;  char buf[256];  int i=0;    int len = read(client\_sock, buf, sizeof(buf)-1);  buf[len] = '\0';  printf("receive[%d]: %s\n", len, buf);  //转换成大写  for(i=0; i<len; i++){  //if(buf[i]>='a' && buf[i]<='z'){  // buf[i] = buf[i] - 32;  //}  buf[i] = toupper(buf[i]);  }  len = write(client\_sock, buf, len);  printf("finished. len: %d\n", len);  close(client\_sock);    if(arg) free(arg);  } |

**并发Mini http 服务器改造**

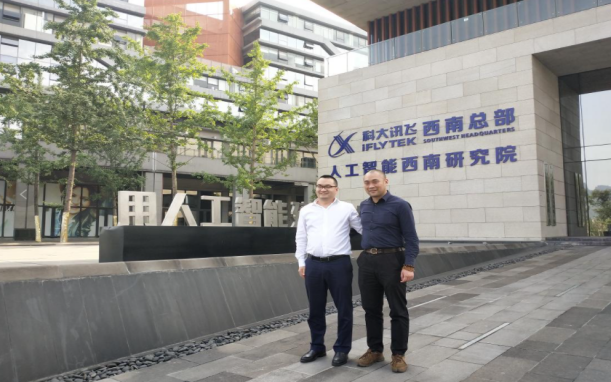
|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/socket.h>  #include <string.h>  #include <ctype.h>  #include <arpa/inet.h>  #include <errno.h>  #include <sys/stat.h>  #include <unistd.h>  #include <pthread.h>  #define SERVER\_PORT 80  static int debug = 1;  int get\_line(int sock, char \*buf, int size);  void\* do\_http\_request(void \*client\_sock);  void do\_http\_response(int client\_sock, const char \*path);  int headers(int client\_sock, FILE \*resource);  void cat(int client\_sock, FILE \*resource);  void not\_found(int client\_sock);//404  void unimplemented(int client\_sock);//500  void bad\_request(int client\_sock); //400  void inner\_error(int client\_sock);  int main(void){  int sock;//代表信箱  struct sockaddr\_in server\_addr;  //1.美女创建信箱  sock = socket(AF\_INET, SOCK\_STREAM, 0);  //2.清空标签，写上地址和端口号  bzero(&server\_addr, sizeof(server\_addr));  server\_addr.sin\_family = AF\_INET;//选择协议族IPV4  server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);//监听本地所有IP地址  server\_addr.sin\_port = htons(SERVER\_PORT);//绑定端口号  //实现标签贴到收信得信箱上  bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));  //把信箱挂置到传达室，这样，就可以接收信件了  listen(sock, 128);  //万事俱备，只等来信  printf("等待客户端的连接\n");  int done =1;  while(done){  struct sockaddr\_in client;  int client\_sock, len, i;  char client\_ip[64];  char buf[256];  pthread\_t id;  int\* pclient\_sock = NULL;  socklen\_t client\_addr\_len;  client\_addr\_len = sizeof(client);  client\_sock = accept(sock, (struct sockaddr \*)&client, &client\_addr\_len);  //打印客服端IP地址和端口号  printf("client ip: %s\t port : %d\n",  inet\_ntop(AF\_INET, &client.sin\_addr.s\_addr,client\_ip,sizeof(client\_ip)),  ntohs(client.sin\_port));      /\*处理http 请求,读取客户端发送的数据\*/  //do\_http\_request(client\_sock);    //启动线程处理http 请求  pclient\_sock = (int \*)malloc(sizeof(int));  \*pclient\_sock = client\_sock;    pthread\_create(&id, NULL, do\_http\_request, (void \*)pclient\_sock);    //close(client\_sock);  }  close(sock);  return 0;  }  void\* do\_http\_request(void\* pclient\_sock){  int len = 0;  char buf[256];  char method[64];  char url[256];  char path[256];  int client\_sock = \*(int \*)pclient\_sock;    struct stat st;      /\*读取客户端发送的http 请求\*/      //1.读取请求行  len = get\_line(client\_sock, buf, sizeof(buf));    if(len > 0){//读到了请求行  int i=0, j=0;  while(!isspace(buf[j]) && (i<sizeof(method)-1)){  method[i] = buf[j];  i++;  j++;  }    method[i] = '\0';  if(debug) printf("request method: %s\n", method);    if(strncasecmp(method, "GET", i)==0){ //只处理get请求  if(debug) printf("method = GET\n");    //获取url  while(isspace(buf[j++]));//跳过白空格  i = 0;    while(!isspace(buf[j]) && (i<sizeof(url)-1)){  url[i] = buf[j];  i++;  j++;  }    url[i] = '\0';    if(debug) printf("url: %s\n", url);    //继续读取http 头部  do{  len = get\_line(client\_sock, buf, sizeof(buf));  if(debug) printf("read: %s\n", buf);    }while(len>0);    //\*\*\*定位服务器本地的html文件\*\*\*    //处理url 中的?  {  char \*pos = strchr(url, '?');  if(pos){  \*pos = '\0';  printf("real url: %s\n", url);  }  }    sprintf(path, "./html\_docs/%s", url);  if(debug) printf("path: %s\n", path);    //执行http 响应  //判断文件是否存在，如果存在就响应200 OK，同时发送相应的html 文件,如果不存在，就响应 404 NOT FOUND.  if(stat(path, &st)==-1){//文件不存在或是出错  fprintf(stderr, "stat %s failed. reason: %s\n", path, strerror(errno));  not\_found(client\_sock);  }else {//文件存在    if(S\_ISDIR(st.st\_mode)){  strcat(path, "/index.html");  }    do\_http\_response(client\_sock, path);  }  }else {//非get请求, 读取http 头部，并响应客户端 501 Method Not Implemented  fprintf(stderr, "warning! other request [%s]\n", method);  do{  len = get\_line(client\_sock, buf, sizeof(buf));  if(debug) printf("read: %s\n", buf);    }while(len>0);    unimplemented(client\_sock); //请求未实现    }  }else {//请求格式有问题，出错处理  bad\_request(client\_sock); //在响应时再实现  }    close(client\_sock);  if(pclient\_sock) free(pclient\_sock);//释放动态分配的内存    return NULL;  }  void do\_http\_response(int client\_sock, const char \*path){  int ret = 0;  FILE \*resource = NULL;    resource = fopen(path, "r");    if(resource == NULL){  not\_found(client\_sock);  return ;  }    //1.发送http 头部  ret = headers(client\_sock, resource);    //2.发送http body .  if(!ret){  cat(client\_sock, resource);  }    fclose(resource);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*返回关于响应文件信息的http 头部  \*输入：  \* client\_sock - 客服端socket 句柄  \* resource - 文件的句柄  \*返回值： 成功返回0 ，失败返回-1  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int headers(int client\_sock, FILE \*resource){  struct stat st;  int fileid = 0;  char tmp[64];  char buf[1024]={0};  strcpy(buf, "HTTP/1.0 200 OK\r\n");  strcat(buf, "Server: Martin Server\r\n");  strcat(buf, "Content-Type: text/html\r\n");  strcat(buf, "Connection: Close\r\n");    fileid = fileno(resource);    if(fstat(fileid, &st)== -1){  inner\_error(client\_sock);  return -1;  }    snprintf(tmp, 64, "Content-Length: %ld\r\n\r\n", st.st\_size);  strcat(buf, tmp);    if(debug) fprintf(stdout, "header: %s\n", buf);    if(send(client\_sock, buf, strlen(buf), 0)<0){  fprintf(stderr, "send failed. data: %s, reason: %s\n", buf, strerror(errno));  return -1;  }    return 0;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*说明：实现将html文件的内容按行  读取并送给客户端  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void cat(int client\_sock, FILE \*resource){  char buf[1024];    fgets(buf, sizeof(buf), resource);    while(!feof(resource)){  int len = write(client\_sock, buf, strlen(buf));    if(len<0){//发送body 的过程中出现问题,怎么办？1.重试？ 2.  fprintf(stderr, "send body error. reason: %s\n", strerror(errno));  break;  }    if(debug) fprintf(stdout, "%s", buf);  fgets(buf, sizeof(buf), resource);    }  }  void do\_http\_response1(int client\_sock){  const char \*main\_header = "HTTP/1.0 200 OK\r\nServer: Martin Server\r\nContent-Type: text/html\r\nConnection: Close\r\n";  const char \* welcome\_content = "\  <html lang=\"zh-CN\">\n\  <head>\n\  <meta content=\"text/html; charset=utf-8\" http-equiv=\"Content-Type\">\n\  <title>This is a test</title>\n\  </head>\n\  <body>\n\  <div align=center height=\"500px\" >\n\  <br/><br/><br/>\n\  <h2>大家好，欢迎来到奇牛学院VIP 试听课！</h2><br/><br/>\n\  <form action=\"commit\" method=\"post\">\n\  尊姓大名: <input type=\"text\" name=\"name\" />\n\  <br/>芳龄几何: <input type=\"password\" name=\"age\" />\n\  <br/><br/><br/><input type=\"submit\" value=\"提交\" />\n\  <input type=\"reset\" value=\"重置\" />\n\  </form>\n\  </div>\n\  </body>\n\  </html>";  //1. 发送main\_header  int len = write(client\_sock, main\_header, strlen(main\_header));    if(debug) fprintf(stdout, "... do\_http\_response...\n");  if(debug) fprintf(stdout, "write[%d]: %s", len, main\_header);    //2. 生成Content-Length  char send\_buf[64];  int wc\_len = strlen(welcome\_content);  len = snprintf(send\_buf, 64, "Content-Length: %d\r\n\r\n", wc\_len);  len = write(client\_sock, send\_buf, len);    if(debug) fprintf(stdout, "write[%d]: %s", len, send\_buf);    len = write(client\_sock, welcome\_content, wc\_len);  if(debug) fprintf(stdout, "write[%d]: %s", len, welcome\_content);  }  //返回值： -1 表示读取出错， 等于0表示读到一个空行， 大于0 表示成功读取一行  int get\_line(int sock, char \*buf, int size){  int count = 0;  char ch = '\0';  int len = 0;      while( (count<size - 1) && ch!='\n'){  len = read(sock, &ch, 1);    if(len == 1){  if(ch == '\r'){  continue;  }else if(ch == '\n'){  //buf[count] = '\0';  break;  }    //这里处理一般的字符  buf[count] = ch;  count++;    }else if( len == -1 ){//读取出错  perror("read failed");  count = -1;  break;  }else {// read 返回0,客户端关闭sock 连接.  fprintf(stderr, "client close.\n");  count = -1;  break;  }  }    if(count >= 0) buf[count] = '\0';    return count;  }  void not\_found(int client\_sock){  const char \* reply = "HTTP/1.0 404 NOT FOUND\r\n\  Content-Type: text/html\r\n\  \r\n\  <HTML lang=\"zh-CN\">\r\n\  <meta content=\"text/html; charset=utf-8\" http-equiv=\"Content-Type\">\r\n\  <HEAD>\r\n\  <TITLE>NOT FOUND</TITLE>\r\n\  </HEAD>\r\n\  <BODY>\r\n\  <P>文件不存在！\r\n\  <P>The server could not fulfill your request because the resource specified is unavailable or nonexistent.\r\n\  </BODY>\r\n\  </HTML>";  int len = write(client\_sock, reply, strlen(reply));  if(debug) fprintf(stdout, reply);    if(len <=0){  fprintf(stderr, "send reply failed. reason: %s\n", strerror(errno));  }      }  void unimplemented(int client\_sock){  const char \* reply = "HTTP/1.0 501 Method Not Implemented\r\n\  Content-Type: text/html\r\n\  \r\n\  <HTML>\r\n\  <HEAD>\r\n\  <TITLE>Method Not Implemented</TITLE>\r\n\  </HEAD>\r\n\  <BODY>\r\n\  <P>HTTP request method not supported.\r\n\  </BODY>\r\n\  </HTML>";  int len = write(client\_sock, reply, strlen(reply));  if(debug) fprintf(stdout, reply);    if(len <=0){  fprintf(stderr, "send reply failed. reason: %s\n", strerror(errno));  }  }  void bad\_request(client\_sock){  const char \* reply = "HTTP/1.0 400 BAD REQUEST\r\n\  Content-Type: text/html\r\n\  \r\n\  <HTML>\r\n\  <HEAD>\r\n\  <TITLE>BAD REQUEST</TITLE>\r\n\  </HEAD>\r\n\  <BODY>\r\n\  <P>Your browser sent a bad request！\r\n\  </BODY>\r\n\  </HTML>";  int len = write(client\_sock, reply, strlen(reply));  if(len<=0){  fprintf(stderr, "send reply failed. reason: %s\n", strerror(errno));  }  }  void inner\_error(int client\_sock){  const char \* reply = "HTTP/1.0 500 Internal Sever Error\r\n\  Content-Type: text/html\r\n\  \r\n\  <HTML lang=\"zh-CN\">\r\n\  <meta content=\"text/html; charset=utf-8\" http-equiv=\"Content-Type\">\r\n\  <HEAD>\r\n\  <TITLE>Inner Error</TITLE>\r\n\  </HEAD>\r\n\  <BODY>\r\n\  <P>服务器内部出错.\r\n\  </BODY>\r\n\  </HTML>";  int len = write(client\_sock, reply, strlen(reply));  if(debug) fprintf(stdout, reply);    if(len <=0){  fprintf(stderr, "send reply failed. reason: %s\n", strerror(errno));  }      } |

# 目标平台

第一梯队（一线大厂，要求逻辑思维、编程思维比较强）

第二梯队（国企、上市公司研发部门，逻辑思维能力一般）

第三梯队（各大软件园的海量中小企业，只要你够努力就够了）

# 学员就业详情

<https://www.yuque.com/qiniuxueyuan/jiuye/ny150b#FGYIL>

