# C/C++从入门到精通-高级程序员之路

**【服务器篇】**

# 第一部分 Linux介绍

课程链接：https://ke.qq.com/course/388295

## 为什么要选择Linux

无论你信与不信，Linux 已经成为这个世界上增长最迅速的操作系统!

在服务器领域，IBM、HP、Novell、Oracle 等厂商对Linux 系统提供了全方位的支持。2004年，IBM宣布其全线

服务器均支持Linux,思科公司在网络防火墙和路由器中也使用了定制的Linux，阿里云也开发了一套基于Linux

的智能操作系统“YunOS”,可用于智能手机、平板电脑和网络电视。

2010年排名前500名的超级计算机中，92.4%（462台）都采用了Linux 操作系统。从2001年以来，基于Linux

的服务器操作系统逐步发展壮大。国内几个主要的Linux厂商和科研机构，国防科技大学、中标软件、中科红旗

等先后推出了Linux服务器操作系统产品，并且已经在政府、企业等领域得到了广泛的应用。

1. **市场价值**

薪资待遇从月薪几千到年薪百万，你想要的薪资待遇都有！

岗位需求从初级开发到技术总监、CTO都需要具备Linux 开发技能！

1. **市场应用**

小到小孩用的玩具、小米的智能音箱手机、电脑、Pad 、智能手表、智能眼镜，以及电视等都是使用的Linux 系统。

大到火箭、航母（航天军工）等装备，淘宝、京东、网易、百度等BAT公司使用的大数据、云服务器集群，其内部使用的都是Linux 操作系统。

**Linux**因其稳定、开源、免费、安全、高效的特点，发展迅猛，在服务器市场占有率超过80%。

随着云计算的发展，[Linux](http://www.oldboyedu.com/Public/lnh/kec/index1.html)在未来服务器领域仍是大势所趋，大有可为!

## 学完Linux服务器开发能做什么

## （有网络的世界就离不开我）

棋牌、网络游戏服务器开发

微信小程序服务器开发

视频直播服务器开发

购物网站服务器开发

物联网服务器开发

区块链服务器开发

其它应用服务器开发

# 第二部分 Linux服务器开发学习方法

1. 多动手实践，理论结合实际。对于每个命令都要亲自操作实践，对于命令的每个参数也要亲自实践，只有这样才能理解其含义。
2. 一定要习惯命令行方式工作。Linux下90%的操作都是通过命令行完成的，因此命令是必须要熟练掌握的。
3. 学会使用Linux的联机帮助。Linux常用命令有上百个，如要识记每个命令，每个人都办不到。但是通过查询帮助文档，  
    执行man XX，即可列出XX命令的所有参数和用法。熟练、灵活运用联机帮助，在Linux下工作会有事半功倍的效果。
4. 学会利用网络资源。可以在网络上的Linux技术社区、网站、论坛上，发现很多Linux爱好者发表的个人学习经验，

这些免费的技术经验和资料是学习Linux的瑰宝。

|  |  |
| --- | --- |
| **国内外网站** | **说明** |
| freecode.com | 最齐全的Linux/UNIX软件库 |
| www.justlinux.com | 信息齐全的Linux学习网站 |
| www.kernel.org | Linux 内核的官方网站 |
| www.linux.com | 提供全方位的Linux 信息 |
| [www.linuxhq.com](http://www.linuxhq.com) | 提供内核信息和不定的汇总 |
| www.chinaunix.net | 国内最大的Linux/UNIX技术社区网站 |
| www.linuxeden.com | Linux 伊甸园,最大的中文开源咨询门户网站 |
| www.linuxfans.org | 中国Linux 公社 |
| [www.linuxsir.org](http://www.linuxsir.org) | 提供Linux各种资源。包括资讯、软件和手册 |

常用的国内外Linux 资源

## 大道至简

1. 不要刻意死板记忆函数、概念和系统命令。
2. 以项目为导向，在解决项目问题中学习。
3. 不断试错，在错误中学习。

## 初学者遇到问题的解决办法

1. 自己先思考10分钟。
2. 如果还不能解决，马上问老师。
3. 把问题的解决方案记录下来。(建议用博客）

## 老鸟遇到问题的解决办法

1. 自己研究30分钟以上。
2. 如果还不能解决，百度、谷歌查询类似问题。
3. 重复以上2个步骤。
4. 把问题的解决方案记录下来。(建议用博客）

**第三部分EPOLL详解 + 项目实战**

前方大招，见附件《**Martin 带你们玩Epoll**》

项目源码 epoll\_web\_server.c

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<string.h>  #include<errno.h>  #include<sys/types.h>  #include<sys/epoll.h>  #include<sys/socket.h>  #include<arpa/inet.h>  #include<netinet/in.h>  #include<assert.h>  #include<fcntl.h>  #include<unistd.h>  // int fd;  typedef struct \_ConnectStat ConnectStat;  typedef void(\*response\_handler) (ConnectStat \* stat);  struct \_ConnectStat {  int fd;  char name[64];  char age[64];  struct epoll\_event \_ev;  int status;//0 -未登录 1 - 已登陆  response\_handler handler;//不同页面的处理函数  };  //http协议相关代码  ConnectStat \* stat\_init(int fd);  void connect\_handle(int new\_fd);  void do\_http\_respone(ConnectStat \* stat);  void do\_http\_request(ConnectStat \* stat);  void welcome\_response\_handler(ConnectStat \* stat);  void commit\_respone\_handler(ConnectStat \* stat);  const char \*main\_header = "HTTP/1.0 200 OK\r\nServer: Martin Server\r\nContent-Type: text/html\r\nConnection: Close\r\n";  static int epfd = 0;  void usage(const char\* argv)  {  printf("%s:[ip][port]\n", argv);  }  void set\_nonblock(int fd)  {  int fl = fcntl(fd, F\_GETFL);  fcntl(fd, F\_SETFL, fl | O\_NONBLOCK);  }  int startup(char\* \_ip, int \_port) //创建一个套接字，绑定，检测服务器  {  //sock  //1.创建套接字  int sock = socket(AF\_INET, SOCK\_STREAM, 0);  if (sock < 0)  {  perror("sock");  exit(2);  }  int opt = 1;  setsockopt(sock, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt));  //2.填充本地 sockaddr\_in 结构体（设置本地的IP地址和端口）  struct sockaddr\_in local;  local.sin\_port = htons(\_port);  local.sin\_family = AF\_INET;  local.sin\_addr.s\_addr = inet\_addr(\_ip);  //3.bind（）绑定  if (bind(sock, (struct sockaddr\*)&local, sizeof(local)) < 0)  {  perror("bind");  exit(3);  }  //4.listen（）监听 检测服务器  if (listen(sock, 5) < 0)  {  perror("listen");  exit(4);  }  //sleep(1000);  return sock; //这样的套接字返回  }  int main(int argc, char \*argv[])  {  if (argc != 3) //检测参数个数是否正确  {  usage(argv[0]);  exit(1);  }  int listen\_sock = startup(argv[1], atoi(argv[2])); //创建一个绑定了本地 ip 和端口号的套接字描述符  //1.创建epoll epfd为epoll\_id  epfd = epoll\_create(256); //可处理的最大句柄数256个  if (epfd < 0)  {  perror("epoll\_create");  exit(5);  }  struct epoll\_event \_ev; //epoll结构填充  ConnectStat \* stat = stat\_init(listen\_sock);  \_ev.events = EPOLLIN; //初始关心事件为读  \_ev.data.ptr = stat;  //\_ev.data.fd = listen\_sock; //  //2.托管  epoll\_ctl(epfd, EPOLL\_CTL\_ADD, listen\_sock, &\_ev); //将listen sock添加到epfd中，关心读事件  struct epoll\_event revs[64];  int timeout = -1;  int num = 0;  int done = 0;  while (!done)  {  //epoll\_wait()相当于在检测事件  switch ((num = epoll\_wait(epfd, revs, 64, timeout))) //返回需要处理的事件数目 64表示 事件有多大  {  case 0: //返回0 ，表示监听超时  printf("timeout\n");  break;  case -1: //出错  perror("epoll\_wait");  break;  default: //大于零 即就是返回了需要处理事件的数目  {  struct sockaddr\_in peer;  socklen\_t len = sizeof(peer);  int i;  for (i = 0; i < num; i++)  {  ConnectStat \* stat = (ConnectStat \*)revs[i].data.ptr;  int rsock = stat->fd; //准确获取哪个事件的描述符  if (rsock == listen\_sock && (revs[i].events) && EPOLLIN) //如果是初始的 就接受，建立链接  {  int new\_fd = accept(listen\_sock, (struct sockaddr\*)&peer, &len);  if (new\_fd > 0)  {  printf("get a new client:%s:%d\n", inet\_ntoa(peer.sin\_addr), ntohs(peer.sin\_port));  //sleep(1000);  connect\_handle(new\_fd);  }  }  else // 接下来对num - 1 个事件处理  {  if (revs[i].events & EPOLLIN)  {  do\_http\_request((ConnectStat \*)revs[i].data.ptr);  }  else if (revs[i].events & EPOLLOUT)  {  do\_http\_respone((ConnectStat \*)revs[i].data.ptr);  }  else  {  }  }  }  }  break;  }//end switch  }//end while  return 0;  }  ConnectStat \* stat\_init(int fd) {  ConnectStat \* temp = NULL;  temp = (ConnectStat \*)malloc(sizeof(ConnectStat));  if (!temp) {  fprintf(stderr, "malloc failed. reason: %m\n");  return NULL;  }  memset(temp, '\0', sizeof(ConnectStat));  temp->fd = fd;  temp->status = 0;  //temp->handler = welcome\_response\_handler;  }  //初始化连接，然后等待浏览器发送请求  void connect\_handle(int new\_fd) {  ConnectStat \*stat = stat\_init(new\_fd);  set\_nonblock(new\_fd);  stat->\_ev.events = EPOLLIN;  stat->\_ev.data.ptr = stat;  epoll\_ctl(epfd, EPOLL\_CTL\_ADD, new\_fd, &stat->\_ev); //二次托管  }  void do\_http\_respone(ConnectStat \* stat) {  stat->handler(stat);  }  void do\_http\_request(ConnectStat \* stat) {  //读取和解析http 请求  char buf[4096];  char \* pos = NULL;  //while header \r\n\r\ndata  ssize\_t \_s = read(stat->fd, buf, sizeof(buf) - 1);  if (\_s > 0)  {  buf[\_s] = '\0';  printf("receive from client:%s\n", buf);  pos = buf;  //Demo 仅仅演示效果，不做详细的协议解析  if (!strncasecmp(pos, "GET", 3)) {  stat->handler = welcome\_response\_handler;  }  else if (!strncasecmp(pos, "Post", 4)) {  //获取 uri  printf("---Post----\n");  pos += strlen("Post");  while (\*pos == ' ' || \*pos == '/') ++pos;  if (!strncasecmp(pos, "commit", 6)) {//获取名字和年龄  int len = 0;  printf("post commit --------\n");  pos = strstr(buf, "\r\n\r\n");  char \*end = NULL;  if (end = strstr(pos, "name=")) {  pos = end + strlen("name=");  end = pos;  while (('a' <= \*end && \*end <= 'z') || ('A' <= \*end && \*end <= 'Z') || ('0' <= \*end && \*end <= '9')) end++;  len = end - pos;  if (len > 0) {  memcpy(stat->name, pos, end - pos);  stat->name[len] = '\0';  }  }  if (end = strstr(pos, "age=")) {  pos = end + strlen("age=");  end = pos;  while ('0' <= \*end && \*end <= '9') end++;  len = end - pos;  if (len > 0) {  memcpy(stat->age, pos, end - pos);  stat->age[len] = '\0';  }  }  stat->handler = commit\_respone\_handler;  }  else {  stat->handler = welcome\_response\_handler;  }  }  else {  stat->handler = welcome\_response\_handler;  }  //生成处理结果 html ,write  stat->\_ev.events = EPOLLOUT;  //stat->\_ev.data.ptr = stat;  epoll\_ctl(epfd, EPOLL\_CTL\_MOD, stat->fd, &stat->\_ev); //二次托管  }  else if (\_s == 0) //client:close  {  printf("client: %d close\n", stat->fd);  epoll\_ctl(epfd, EPOLL\_CTL\_DEL, stat->fd, NULL);  close(stat->fd);  free(stat);  }  else  {  perror("read");  }  }  void welcome\_response\_handler(ConnectStat \* stat) {  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 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 sendbuffer[4096];  char content\_len[64];  strcpy(sendbuffer, main\_header);  snprintf(content\_len, 64, "Content-Length: %d\r\n\r\n", (int)strlen(welcome\_content));  strcat(sendbuffer, content\_len);  strcat(sendbuffer, welcome\_content);  printf("send reply to client \n%s", sendbuffer);  write(stat->fd, sendbuffer, strlen(sendbuffer));  stat->\_ev.events = EPOLLIN;  //stat->\_ev.data.ptr = stat;  epoll\_ctl(epfd, EPOLL\_CTL\_MOD, stat->fd, &stat->\_ev);  }  void commit\_respone\_handler(ConnectStat \* stat) {  const char \* commit\_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>欢迎学霸同学&nbsp;%s &nbsp;,你的芳龄是&nbsp;%s！</h2><br/><br/>\n\  </div>\n\  </body>\n\  </html>\n";  char sendbuffer[4096];  char content[4096];  char content\_len[64];  int len = 0;  len = snprintf(content, 4096, commit\_content, stat->name, stat->age);  strcpy(sendbuffer, main\_header);  snprintf(content\_len, 64, "Content-Length: %d\r\n\r\n", len);  strcat(sendbuffer, content\_len);  strcat(sendbuffer, content);  printf("send reply to client \n%s", sendbuffer);  write(stat->fd, sendbuffer, strlen(sendbuffer));  stat->\_ev.events = EPOLLIN;  //stat->\_ev.data.ptr = stat;  epoll\_ctl(epfd, EPOLL\_CTL\_MOD, stat->fd, &stat->\_ev);  } |

./server 127.0.0.1 666