Linux 服务器开发

从入门到精通

课程介绍

课程链接：https://ke.qq.com/course/348911?tuin=8e7d0ee7

讲师介绍

动脑学院 Martin 老师

个人简介： 网易企业邮局架构师，老板邮局的首席架构师，曾任职于多家上市公司。

十二年行业（C/C++）开发经验，一直专注于 Linux 服务器端高并发应用程序和分布式大数据存储系统的开发和架构。

了解和研究过多款 C/C++优秀开源软件（Nginx,TFS 等）的框架和代码实现，擅长在其基础上进行二次开发。

QQ 交流: 2390560487

课程目标

零基础培养 Linux 服务器高级开发工程师。

本课程旨在为想系统学习 Linux 服务器开发的朋友提供专业系统的教学服务；课程编排采取与众不同的方式，

注重讲师多年开发经验分享，注重技术在项目开发中的实用性；以企业实战项目案例驱动整个服务器开发课程

的学习，快速提升你的项目经验和开发技术水平！认真学习完本课程并全面掌握课程各实战项目后，可以达到

Linux 服务器高级开发工程师的水准！

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！

2. 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 在未来服务器领域仍是大势所趋，大有可为!

适用于

1）如果你熟悉 C 语言，想从事服务器开发工作，博取高薪；

2）如果你是具有 C 语言基础的在校学生，鄙视学校脱离实际开发的教学模式，想快速积累项目实战经验；

3）如果你厌倦了简单的应用开发、界面开发和客户端开发，疲于学习新的平台和各种框架，想转型做服务器开发的在职工程师；

4）如果你有志于从事 IT 行业，但什么也不知道，那你可以从我们的《C/C++从入门到精通开发》开始。

不适用于

已具备丰富的 Linux 服务器开发经验的 C 程序员。

学习内容

《Linux 服务器开发从入门到精通大纲》

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

课程服务

- 一对一讲师指导
- 一对一习题批改
- 企业级项目全程指导
- 简历设计
- 简历包装
- 面试指导
- 试用期全程技术指导

第一阶段-零基础极速入门

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	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	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）

项目 1 搭建 Linux 开发环境

开发平台的选择

1. 如果已经了解 Linux 操作系统的基本使用，建议使用 Linux 平台
2. 如果不了解 Linux 操作系统，就直接使用 Windows 平台，以后再学习 Linux 操作系统。
3. 零基础的初学者，建议使用 Windows 平台。

Windows 平台开发环境的搭建

安装包可加群从群文件夹中获取! 群号: 646266989 加群口令： 动脑学院-服务器开发课程

1. 安装 VMware 虚拟机

从群文件夹=>Linux 开发环境安装 中获取网盘链接安装

2. 安装 Linux 操作系统

Linux 操作系统，可选择：

1) CentOS（建议：Centos 7.0 以上）

补充：国内大部分企业的服务器是使用 CentOS 或（RedHat）

CentOS 是 Redhat 的社区版，用法相同。

2) Ubuntu 系统

从群文件夹=>Linux 开发环境安装 中获取网盘链接安装

安装完后执行命令安装 openssh 服务器： sudo apt-get install openssh-server

3. 访问我们的 Linux 系统

1）通过 Vmware 界面操作

2）通过远程终端操作
* Putty
* MobaXterm

4. 在让我们的 Linux 连上网

3.1 虚拟机网卡设置

建议把虚拟机的网卡设置为桥接模式。

检查：
ping www.baidu.com
或直接在浏览器中打开百度网站(www.baidu.com)

3.2 Linux 系统网络设置

5. 在让编程效率飞起来

4.1 Ubuntu 安装 Samba 服务器

确认安装: dpkg -l | grep samba
安装： sudo apt-get install samba samba-common
卸载： sudo apt-get autoremove samba

4.2 Samba 服务器配置

sudo vi /etc/samba/smb.conf

在文件最后添加

[Share]
comment=This is samba dir
path=/home/martin/share

writable=yes
browseable=yes

4.3 启动和关闭

启动 Samba 服务器: sudo service smbd start
关闭 Samba 服务器： sudo service smbd stop

4.4 直接在 Windows 下编码

编辑器的选择

编辑器的作用：
编写程序（源代码）。

编辑器的选择：
初学者最好使用最简单的文本编辑器，不要使用集成开发环境 IDE
Linux 平台：vi, vim, 或 gedit
Windows 平台：记事本，Sublime Text, UltraEdit, notePad, notePad++, source insight

开发方式

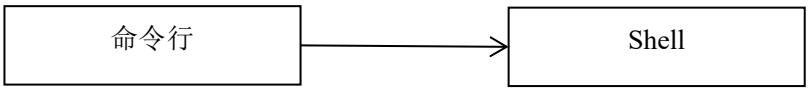
先用编辑器编写源代码
然后进入 Linux 系统，使用 gcc 编译器，对源代码进行编译运行。

建议初学者，使用方式 2。

项目 2 极速入门必备命令

Shell 简介

Shell 是运维和系统管理员操作 Linux 系统的首选，简单说，它是一个命令解释器.



输入命令 解释执行

命令行相关： 行首“$” 或 “#” — 命令行提示符
行中 ## — 视为注释开始
注意： 命令行是区分大小写的！

使用命令行补全（Tab） 和通配符可以提高输入效率

通配符共有 3 个： “*”， “?”， “[]”
* — 用于匹配文件名中任意长度的字符串；
? — 只匹配一个字符；
[] — 用于匹配所有出现在方括号内的字符。可以使用短线“-”来指定字符集范围。如：
ls text[1-3] 或 *ls test[a-z]*

Linux 下环境下有几种不同 Shell ,常用的有 BASH 、TCSH Shell 和 Z-Shell 等.BASH 是默认安装和使用的 Shell。

项目精讲

1.1. 入门必备命令

1.1.1. 寻求帮助 - man 命令

作用： Linux 为所有命令和系统调用编写了帮助手册。使用 man 命令可以方便地获取某个命令的帮助信息。
用法： man [手册编号] 命令名

man 命令在显示手册页时实际调用 less 完成显示，J K 可以上下翻动，空格用于向下翻页。Q 键退出。
手册总共分为 9 节，各部分内容如下：

目 录	内 容
/usr/share/man/man1	普通命令和应用程序
/usr/share/man/man2	系统调用
/usr/share/man/man3	库调用，主要是 libc()函数的使用文档
/usr/share/man/man4	设备驱动和网络协议
/usr/share/man/man5	文件的详细格式信息
/usr/share/man/man6	游戏
/usr/share/man/man7	文档使用说明
/usr/share/man/man8	系统管理命令
/usr/share/man/man9	内核源代码或模块的技术指标

1.1.2. 用户间切换 SU

作用： 切换到其他用户
用法： su [用户名]

```
$ su    martin        #切换到 martin 这个用户

$ su                  #切换到 特权用户 root
```

注意： Ubuntu 默认情况下，系统没有合法的 root 权限，不能直接使用 su 命令提升到 root 权限，而必须要用 sudo 来获得 root 权限

1.1.3. 特权命令 sudo

作用： 提升当前执行命令的权限，以 root 身份执行它.
用法： sudo 命令行

```
$ sudo  su                  #切换到 root 用户

$ sudo   rm   root.txt      #切换到 特权用户 root
```

1.2. 文件操作常用命令

1.2.1. 显示当前目录 pwd 和改变目录 cd

pwd 命令
作用： 显示当前目录,即工作目录
用法： pwd

cd 命令
作用： 改变目录位置
用法： cd **...[OPTION]...[FILE]...**

cd 目录路径 – 进入指定的目录中去
cd .. – 返回父目录
cd / – 进入根目录

cd 或 cd ~ — 进入用户主目录
cd ./* - 进入当前目录下*表示的子目录

1.2.2. ls 命令

用法： **ls ...[OPTION] ...[FILE]...**

常用参数：

1. 不带任何参数 列出当前目录下的所有文件和子目录
2. -F 分类显示，方便阅读
3. -a 显示隐含文件
4. -l 查看文件的各种属性

1.2.3. 列出目录内容： dir 和 vdir

用法： dir **...[OPTION] ...[FILE]...**
vidr **...[OPTION] ...[FILE]...**

dir 和 ls 差不多，就比 ls 功能少
vdir 相当于 ls -l 命令

1.2.4. 建立目录 mkdir

用法： mkdir **...[OPTION] ...[FILE]...**

mkdir 一次可以建立一个或 几个目录

常用参数：

不带任何参数 创建相应目录，如果目录的父级目录路径不存在，则创建失败;
-p 创建相应目录，如果目录的父级目录路径不存在，则一起创建;

1.2.5. 移动、复制和删除

移动命令 mv

用法： mv **. . .[OPTION] 源文件 目标文件**

常用参数：

不带任何参数	将源文件移动到目标文件，注意：如果目标文件存在则**替换**;
-i	将源文件移动到目标文件，如果目标文件存在则提示是否**替换**;
-b	将源文件移动到目标文件，如果目标文件存在则不进行覆盖，而是在目标文件后加~

复制命令 cp

用法： cp **. . .[OPTION] 源文件 目标文件**

常用参数：

不带任何参数	将源文件复制到目标文件，注意：如果目标文件存在则**替换**;
-i	将源文件复制到目标文件，如果目标文件存在则提示是否**替换**;
-b	将源文件复制到目标文件，如果目标文件存在则不进行覆盖，而是在目标文件后加~
-r	将子目录及其中的文件一起复制到另一个子目录下

删除命令 rm

删除命令可以一次永久性删除一个或几个文件(包含目录)

用法： rm **. . .[OPTION]. . . [FILE]. . .**

常用参数：

不带任何参数	删除文件或相应目录，不给予任何提示;
-i	删除文件或相应目录，删除时进行提醒;
-f	强制性删除文件或相应目录;
-r	将子目录及其中的文件一并删除。**（慎用！特别时在 root 权限下）**

1.2.6. 文件链接 ln

ln 建立文件链接

用法： ln **. . .[OPTION] 源文件 目标文件**

常用参数：

不带任何参数	创建硬链接，ls -i 查看可以看到两个文件的 inode 值一致;
-s	创建软链接,即别名,如果源文件删除，则软链接(别名)也无法访问。

1.2.7. 改变文件所有权 chown 和 chgrp

chown 命令用于改变文件的所有权。

用法： chown **. . .[OPTION] [OWNER][:[GROUP]] FILE . . .**

常用参数：

不带任何参数	改变单个或多个文件的属主和属组;
-r	改变一个目录及其下所有文件（和子目录）的所有权设置。

chgrd 用于单独设置文件的属组。

用法： chown **. . .[GROUP] FILE . . .**

```
$ chgrp          nogroup          text.txt
```

1.2.8. 改变文件权限 chmod

chmod 用于改变一个文件的权限。它以“用户组 +/- 权限”的表达方式来增加/删除相应的权限。具体来说，用户组包括了文件属主(u)、文件属组(g)、其他人(o) 和所有人(a)，而权限则包括读取(r、w、x)

用法： chmod **. . .[OPTION] . . .[FILE]. . .**

例：

```
$chmod u+x test.txt
```

```
$chmod u-x test.txt
```

```
$chmod ug=wr、o=r  text.txt
```

1.2.9. 阅读文件的头部和尾部: head 和 tail

用法： head/tail **. . .[OPTION]. . . FILE**

常用参数：

不带任何参数 显示文件的头部/尾部 10 行;

-n 按指定的行数显示文件的头部/尾部 ;

1.2.10. 查看文本文件: cat 和 more

cat 命令用来一次性查看全部文本文件的内容，后跟文件名作为参数.也可以带上 -n 显示每行的行号。

more 命令用来分页查看文本文件。空格翻页；回车向下滚动一行;Q 键退出。

1.2.11. 查看文本文件: cat 、more 和 less

cat 命令用来一次性查看全部文本文件的内容，后跟文件名作为参数.也可以带上 -n 显示每行的行号。如果文件长达几十上百页，不建议使用 cat.

more 命令用来分页查看文本文件。空格翻页；回车向下滚动一行;Q 键退出。

less 更人性化的文本阅读工具。

1.2.12. 编辑器: vim

vim 是 UNIX 和 Linux 上标配的编辑器，功能十分强大。

用法 ：

vim file

或 vim

1.编辑保存文件

分插入和命令两种模式。

插入模式

操作：输入字符，并可以按光标键移动输入字符位置.

命令	操作
a	在光标后插入
i	在光标所在位置插入
o	在光标所在位置的下一行插入
Esc	进入命令模式

:	进入行命令模式

模式切换

命令模式

执行除输入字符之外的所有操作，包括保存、搜索、移动光标等。

vim 启动默认是在命令模式，如果在编辑模式，需要按 Esc 切换回命令模式.

可以使用光标键控制上下左右移动，或者 h、j、k、l 移动。

保存和退出

保存文件、退出等需要切换到行命令模式，在插入模式下输入 Esc 再 输入 :w :q 或 :wq 等。

前提：须切换到命令模式。注意组合命令执行的顺序：w ->q ->!

命令	操作
:w	保存文件
:w filename	另存为 filename
:q	退出 Vim
:q!	强行退出，放弃保存

搜索字符串

前提：须切换到命令模式

/string 用于向下搜索一个字符串

?string 用于向上搜索一个字符串

如果需要启动或关闭大小写敏感,执行：

```
:set ignorecase
```

或

```
:set noignorecase
```

替换字符串

前提：须切换到命令模式。

语法 ： :[range]s/pattern/string/[c、e、g、i]

这条命令将 pattern 所代表的字符串替换成 string。开头的 range 用于指定替换作用的范围，如“1,10” 表示从第 1 行到第 10 行，“1,$”表示从第 1 行到最后一行，也就是全文。全文也可以用 “%” 表示。

最后的方括号是可选选项，含义如下:

标 志	含 义
c	每次替换前询问
e	不显示错误信息
g	替换一行中的所有匹配项(这个选项通常需要使用)
i	不区分大小写

如 Windows 环境下的源码经常会有 “^M”的字符，要清除可以使用下面的命令：

```
:%s/^M$//g
```

删除、复制、粘贴

前提：须切换到命令模式。

命令	操作
x	删除光标所在位置的字符
dd	删除光标所在的行， 2 dd 表示删除 2 行
D	删除光标所在位置到行尾之间所有的字符
d	普遍意义上的删除命令，和移动命令配合使用。例如 dw 表示删除光标所在位置到下一单词词头之间所有的字符
yy	复制光标所在的行
y	普遍意义上的复制命令，和移动命令配合使用。例如 yw 表示父子光标所在位置到下一个单词词头之间所有的字符
p	在光标所在位置粘贴最近复制/删除的内容

撤销和重做

前提：须切换到命令模式。

命令	操作
u	撤销一次操作
Ctrl+R	重做被撤销的操作

前提：须切换到命令模式。

语法 ： :[range]s/pattern/string/[c、e、g、i]

程序员特有配置

```
:syntax on          ##  语法高亮
:set autoindent     ##  自动缩进
:set shiftwidth=4   ##  设置 Tab 键对应的空格数
```

1.3. 查找和定位常用命令

1.3.1. 我的东西呢 - find 命令

作用： 在指定范围内迅速查找到文件。
用法： find [OPTION] [path . . .] [expression]

例：

```
$find /usr   -name test.txt
```

常用参数：

-type 查找时指定文件的类型，可使用参数如下表;
-atime n 查找最后一次使用在 n 天前的文件,n 使用负数表示;
-mtime n 查找最后一次修改在 n 天前的文件;

参数	含义	参数	含义
b	块设备文件	f	普通文件
c	字符设备文件	p	命名管道
d	目录文件	l	符号链接

1.3.2. 更快速的定位文件 - locate 命令

作用： 火箭般的速度定位文件。
用法： locate [expression]

locate 并不进入子目录进行搜索，它通过检索数据库来确定文件的位置。可以使用 updatedb 来更新检索数据库。

1.3.3. 查找文件内容 - grep 命令

作用： 在文件中寻找某些信息。
用法： grep [OPTIONS] PATTERN [FILE...] ## pattern 使用基础正则表达式

$grep open ./test.c

项目 3 开启 Linux 编程之旅

C 和 C++编译器

C 和 C++ 编译器: gcc

GNU C Compiler 的缩写，经过十来年发展，意义变成了 GNU Compiler Collection，可同时支持 C、C++、Objective C 和 Java 等.

1.编译第一个 C/C++程序

只编译执行一个 C 程序

```
$ gcc hello.c
$ ./a.out
$Hello world!
```

默认的 a.out 并不友好，gcc 提供 -o 选项指定执行文件的文件名：

```
$gcc -o hello   hello.c          ##编译源代码，并把可执行文件命名为 hello
$Hello world!
```

编译 C++程序，我们可以直接用 GCC 编译其中的 g ++命令，用法同 gcc；当然 g++ 和 gcc 都可以用来编译 c 和 c++程序。gcc 编译 c++程序需要带上 -lstdc++ 指定使用 c++库。

2.编译常用选项

选　项	功　能
-c	只激活预处理、编译和汇编,生成.o 目标代码文件
-S	只激活预处理和编译，生成扩展名为.s 的汇编代码文件
-E	只激活预处理，并将结果输出至标准输出
-g	为调试程序(如 gdb)生成相关信息
-O	等同-O1,常用的编译优化选项
-Wall	打开一些很有用的警告选项，建议编译时加此选项。

注意：-c 选项在编写大型程序是必须的，多个文件的源代码首先需要编译成目标代码，再链接成执行文件。如果由多个源文件，工程做法建议采用 makefile 。

项目 4 网络服务器开发

项目需求

实现回声服务器的客户端/服务器程序，客户端通过网络连接到服务器，并发送任意一串英文信息，服务器端接收信息后，

将每个字符转换为大写并回送给客户端显示。

项目实现

echo_server.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#include <ctype.h>
#include <arpa/inet.h>



#define SERVER_PORT 666

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];

        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));
        /*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;
}
```

echo_client.c

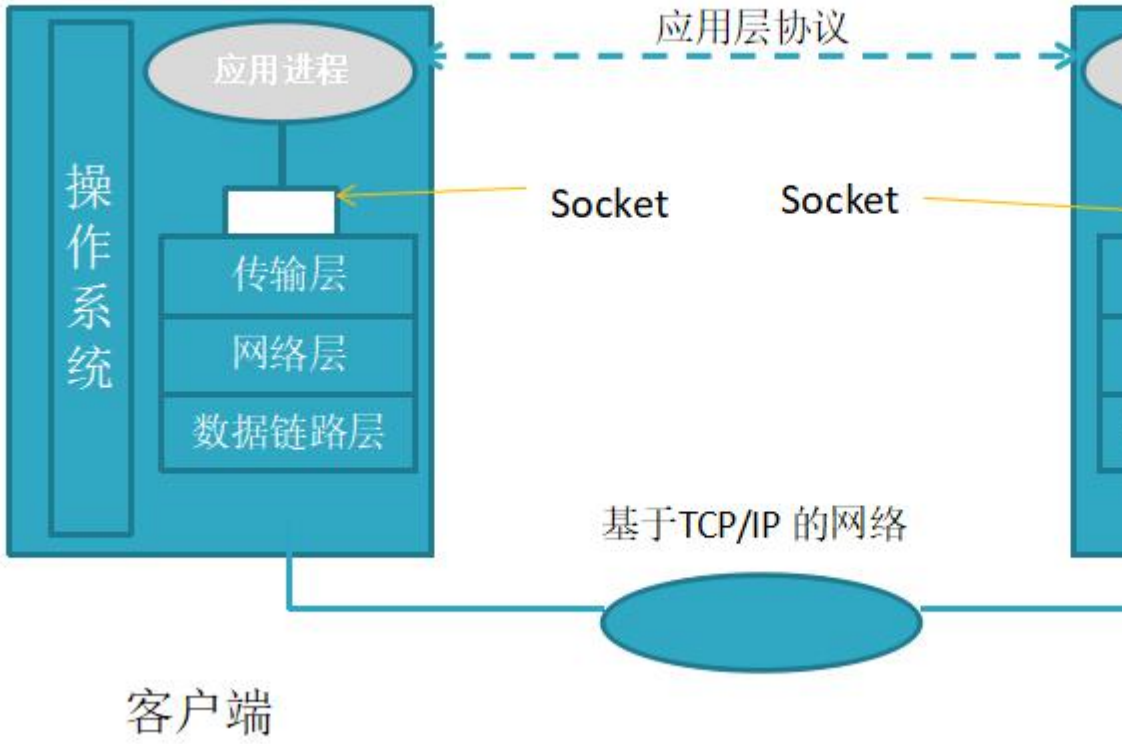
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVER_PORT 666
#define SERVER_IP    "127.0.0.1"

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

        int sockfd;
        char *message;
        struct sockaddr_in servaddr;
        int n;
        char buf[64];

        if(argc != 2){
                fputs("Usage: ./echo_client message \n", stderr);
                exit(1);
        }

        message = argv[1];

        printf("message: %s\n", message);

        sockfd = socket(AF_INET, SOCK_STREAM, 0);

        memset(&servaddr, '\0', sizeof(struct sockaddr_in));

        servaddr.sin_family = AF_INET;
        inet_pton(AF_INET, SERVER_IP, &servaddr.sin_addr);
        servaddr.sin_port = htons(SERVER_PORT);

        connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

        write(sockfd, message, strlen(message));

        n = read(sockfd, buf, sizeof(buf)-1);

        if(n>0){
                buf[n]='\0';
                printf("receive: %s\n", buf);
```

```
    }else {
        perror("error!!!");
    }

    printf("finished.\n");
    close(sockfd);

    return 0;
}
```

项目精讲

1. 网络通信与 Socket

Socket 通信模型



2. Socket 编程详解

2.1 套接字概念

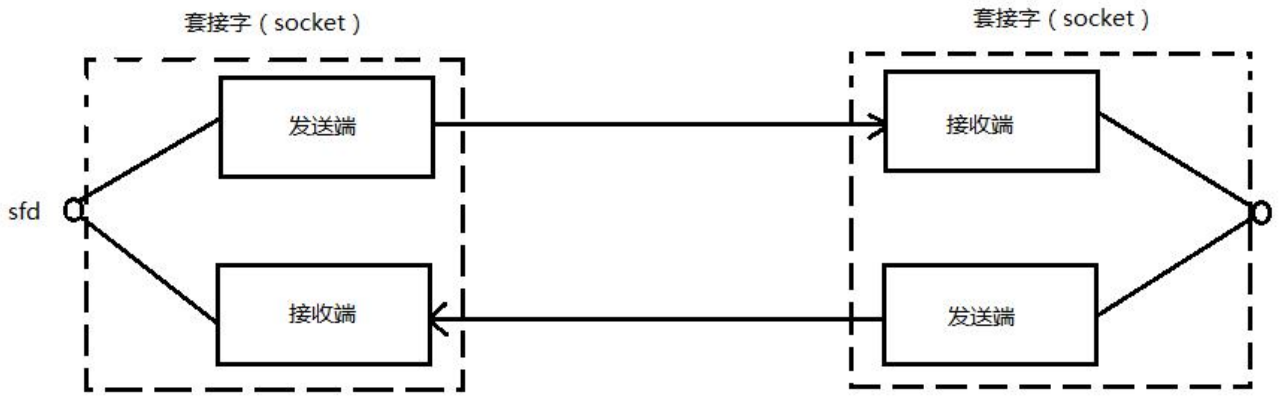
套接字概念

Socket 中文意思是“插座”，在 Linux 环境下，用于表示进程 x 间网络通信的特殊文件类型。本质为内核借助缓冲区形成的伪文件。

既然是文件，那么理所当然的，我们可以使用文件描述符引用套接字。Linux 系统将其封装成文件的目的是为了统一接口，使得读写套接字和读写文件的操作一致。区别是文件主要应用于本地持久化数据的读写，而套接字多应用于网络进程间数据的传递。

在 TCP/IP 协议中，“IP 地址+TCP 或 UDP 端口号”唯一标识网络通讯中的一个进程。“IP 地址+端口号”就对应一个 socket。欲建立连接的两个进程各自有一个 socket 来标识，那么这两个 socket 组成的 socket pair 就唯一标识一个连接。因此可以用 Socket 来描述网络连接的一对一关系。

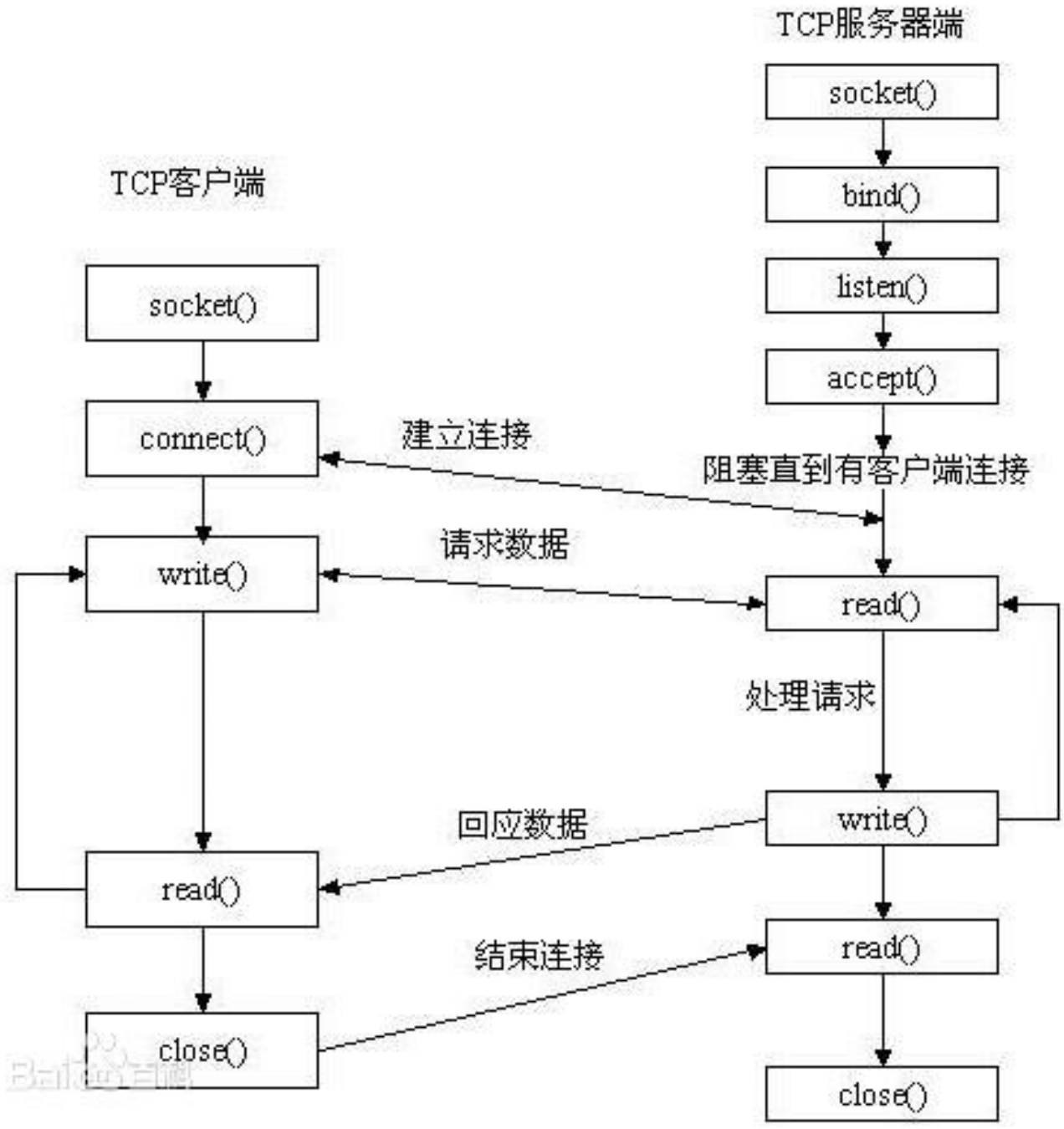
套接字通信原理如下图所示：



套接字通讯原理示意

在网络通信中，套接字一定是成对出现的。一端的发送缓冲区对应对端的接收缓冲区。我们使用同一个文件描述符索发送缓冲区和接收缓冲区。

Socket 通信创建流程图

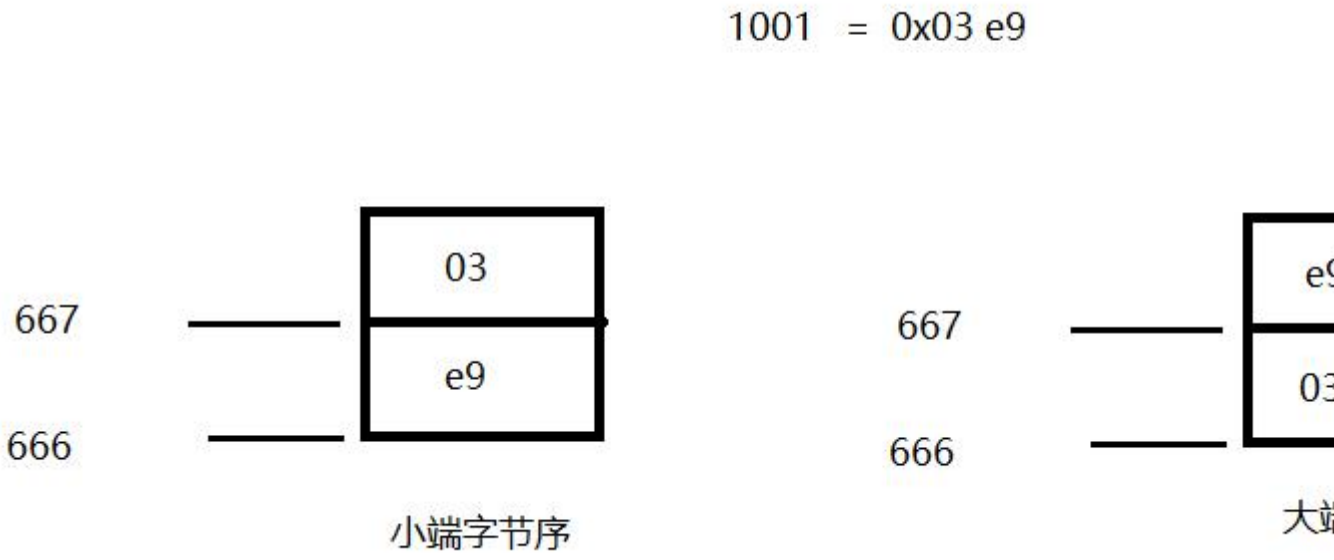


2.2 Socket 编程基础

2.2.1 网络字节序

在计算机世界里，有两种字节序：

大端字节序 - 低地址高字节,高地址低字节
小段字节序 - 低地址低字节,高地址高字节



内存中的多字节数据相对于内存地址有大端和小端之分，磁盘文件中的多字节数据相对于文件中的偏移地址也有大端小端之分。网络数据流同样有大端小端之分，那么如何定义网络数据流的地址呢？发送主机通常将发送缓冲区中的数据按内存地址从低到高的顺序发出，接收主机把从网络上接到的字节依次保存在接收缓冲区中，也是按内存地址从低到高的顺序保存，因此，网络数据流的地址应这样规定：先发出的数据是低地址，后发出的数据是高地址。

TCP/IP 协议规定，网络数据流应采用大端字节序，即低地址高字节。

例如端口号是 1001（0x3e9），由两个字节保存，采用大端字节序，则低地址是 0x03，高地址是 0xe9，也就是先发 0x03，再发 0xe9，这 16 位在发送主机的缓冲区中也应该是低地址存 0x03，高地址存 0xe9。但是，如果发送主机是小端字节序的，这 16 位被解释成 0xe903，而不是 1001。因此，发送主机把 1001 填到发送缓冲区之前需要做字节序的转换。同样地，接收主机如果是小端字节序的，接到 16 位的源端口号也要做字节序的转换。如

果主机是大端字节序的，发送和接收都不需要做转换。同理，32 位的 IP 地址也要考虑网络字节序和主机字节序的问题。

为使网络程序具有可移植性，使同样的 C 代码在大端和小端计算机上编译后都能正常运行，可以调用以下库函数做网络字节序和主机字节序的转换。

```
#include <arpa/inet.h>


uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);
```

h 表示 host，n 表示 network，l 表示 32 位长整数，s 表示 16 位短整数。

如果主机是小端字节序，这些函数将参数做相应的大小端转换然后返回，如果主机是大端字节序，这些函数不做转换，将参数原封不动地返回。

2.2.2 sockaddr 数据结构

很多网络编程函数诞生早于 IPv4 协议，那时候都使用的是 sockaddr 结构体，为了向前兼容，现在 sockaddr 退化成了（void *）的作用，传递一个地址给函数，至于这个函数是 sockaddr_in 还是其他的，由地址族确定，然后函数内部再强制类型转化为所需的地址类型。

sockaddr 数据结构

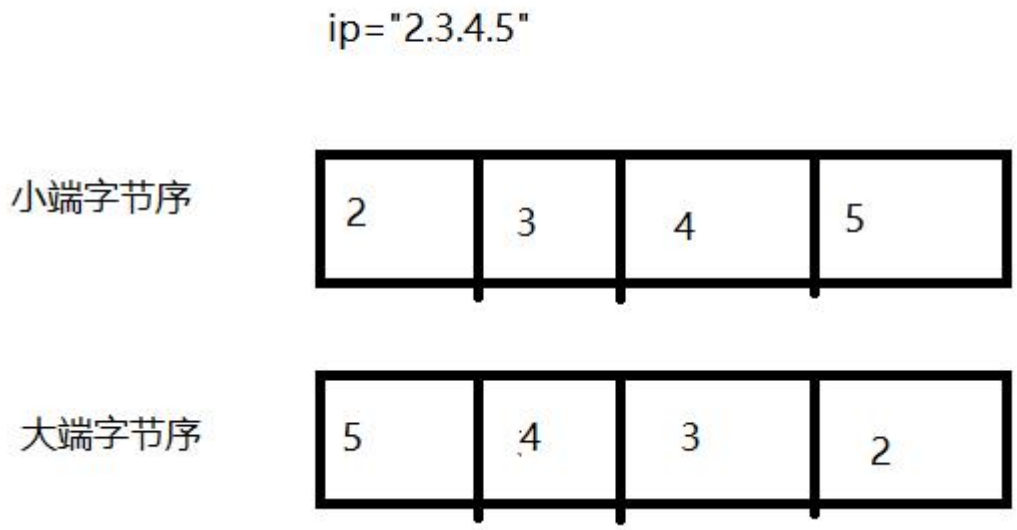
```
struct sockaddr {
    sa_family_t sa_family;          /* address family, AF_xxx */
    char sa_data[14];               /* 14 bytes of protocol address */
};

 struct sockaddr_in {
      sa_family_t     sin_family; /* address family: AF_INET */
      in_port_t       sin_port;   /* port in network byte order */
      struct in_addr sin_addr;    /* internet address */
 };

 /* Internet address. */
struct in_addr {
    uint32_t            s_addr;         /* address in network byte order */
};
```

IPv4 的地址格式定义在 netinet/in.h 中，IPv4 地址用 sockaddr_in 结构体表示，包括 16 位端口号和 32 位 IP 地址，但是 sock API 的实现早于 ANSI C 标准化，那时还没有 void *类型，因此这些像 bind 、accept 函数的参数都用 struct sockaddr *类型表示，在传递参数之前要强制类型转换一下，例如：

struct sockaddr_in servaddr;

bind(listen_fd, (struct sockaddr *)&servaddr, sizeof(servaddr)); /* initialize servaddr */

2.2.3 IP 地址转换函数



#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);

const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);

af 取值可选为 AF_INET 和 AF_INET6 ，即和 ipv4 和 ipv6 对应

支持 IPv4 和 IPv6

其中 inet_pton 和 inet_ntop 不仅可以转换 IPv4 的 in_addr，还可以转换 IPv6 的 in6_addr。

因此函数接口是 void *addrptr。

范例：4.2.2.3

```
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>



int main(void){

	char ip[]="2.3.4.5";
	char server_ip[64];

	struct sockaddr_in server_addr;


	inet_pton(AF_INET, ip, &server_addr.sin_addr.s_addr);

	printf("s_addr : %x\n", server_addr.sin_addr.s_addr);

	printf("s_addr from net to host: %x\n", ntohl(server_addr.sin_addr.s_addr));

	inet_ntop(AF_INET, &server_addr.sin_addr.s_addr, server_ip, 64);

	printf("server ip : %s\n", server_ip);

	printf("INADDR_ANY: %d\n", INADDR_ANY);
	server_addr.sin_addr.s_addr = INADDR_ANY;
	inet_ntop(AF_INET, &server_addr.sin_addr.s_addr, server_ip, 64);
	printf("INADDR_ANY ip : %s\n", server_ip);
	return 0;
}
```

2.3 Socket 编程函数

2.3.1 socket 函数

```
#include <sys/types.h> /* See NOTES */
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
domain:
    AF_INET 这是大多数用来产生 socket 的协议，使用 TCP 或 UDP 来传输，用 IPv4 的地址
    AF_INET6 与上面类似，不过是来用 IPv6 的地址
    AF_UNIX 本地协议，使用在 Unix 和 Linux 系统上，一般都是当客户端和服务器在同一台及其上的时候使用
type:
    SOCK_STREAM 这个协议是按照顺序的、可靠的、数据完整的基于字节流的连接。这是一个使用最多的 socket 类型，这个 socket 是使用 TCP 来进行传输。
    SOCK_DGRAM 这个协议是无连接的、固定长度的传输调用。该协议是不可靠的，使用 UDP 来进行它的连接。
    SOCK_SEQPACKET 该协议是双线路的、可靠的连接，发送固定长度的数据包进行传输。必须把这个包完整的接受才能进行读取。
    SOCK_RAW socket 类型提供单一的网络访问，这个 socket 类型使用 ICMP 公共协议。（ping、traceroute 使用该协议）
    SOCK_RDM 这个类型是很少使用的，在大部分的操作系统上没有实现，它是提供给数据链路层使用，不保证数据包的顺序
protocol:
    传 0 表示使用默认协议。
返回值:
    成功：返回指向新创建的 socket 的文件描述符，失败：返回-1，设置 errno
```

socket()打开一个网络通讯端口，如果成功的话，就像 open()一样返回一个文件描述符，应用程序可以像读写文件一样用 read/write 在网络上收发数据，如果 socket()调用出错则返回-1。对于 IPv4，domain 参数指定为 AF_INET。对于 TCP 协议，type 参数指定为 SOCK_STREAM，表示面向流的传输协议。如果是 UDP 协议，则 type 参数指定为 SOCK_DGRAM，表示面向数据报的传输协议。protocol 参数的介绍从略，指定为 0 即可。

2.3.2 bind 函数

```
#include <sys/types.h> /* See NOTES */
```

```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
sockfd:
    socket 文件描述符
addr:
    构造出 IP 地址加端口号
addrlen:
    sizeof(addr)长度
返回值:
    成功返回 0，失败返回-1，设置 errno
```

服务器程序所监听的网络地址和端口号通常是固定不变的，客户端程序得知服务器程序的地址和端口号后就可以向服务器发起连接，因此服务器需要调用 bind 绑定一个固定的网络地址和端口号。

bind()的作用是将参数 sockfd 和 addr 绑定在一起，使 sockfd 这个用于网络通讯的文件描述符监听 addr 所描述的地址和端口号。前面讲过，struct sockaddr *是一个通用指针类型,addr 参数实际上可以接受多种协议的 sockaddr 结构体，而它们的长度各不相同，所以需要第三个参数 addrlen 指定结构体的长度。如：

```
struct sockaddr_in servaddr;
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(6666);
```

首先将整个结构体清零,然后设置地址类型为 AF_INET,网络地址为 INADDR_ANY，这个宏表示本地的任意 IP 地址，因为服务器可能有多个网卡，每个网卡也可能绑定多个 IP 地址，这样设置可以在所有的 IP 地址上监听，直到与某个客户端建立了连接时才确定下来到底用哪个 IP 地址，端口号为 6666。

2.3.3 listen 函数

```
#include <sys/types.h> /* See NOTES */
#include <sys/socket.h>
int listen(int sockfd, int backlog);
sockfd:
    socket 文件描述符
backlog:
    在 Linux 系统中，它是指排队等待建立 3 次握手队列长度
```

查看系统默认 backlog

```
cat /proc/sys/net/ipv4/tcp_max_syn_backlog
```

改变 系统限制的 backlog 大小

```
vim /etc/sysctl.conf

最后添加
net.core.somaxconn = 1024
net.ipv4.tcp_max_syn_backlog = 1024

保存，然后执行
sysctl -p
```

典型的服务器程序可以同时服务于多个客户端，当有客户端发起连接时，服务器调用的 accept()返回并接受这个连接，如果有大量的客户端发起连接而服务器来不及处理，尚未 accept 的客户端就处于连接等待状态，listen()声明 sockfd 处于监听状态，并且最多允许有 backlog 个客户端处于连接待状态，如果接收到更多的连接请求就忽略。listen()成功返回 0，失败返回-1。

2.3.4 accept 函数

```
#include <sys/types.h>        /* See NOTES */
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

sockdf:

socket 文件描述符

addr:

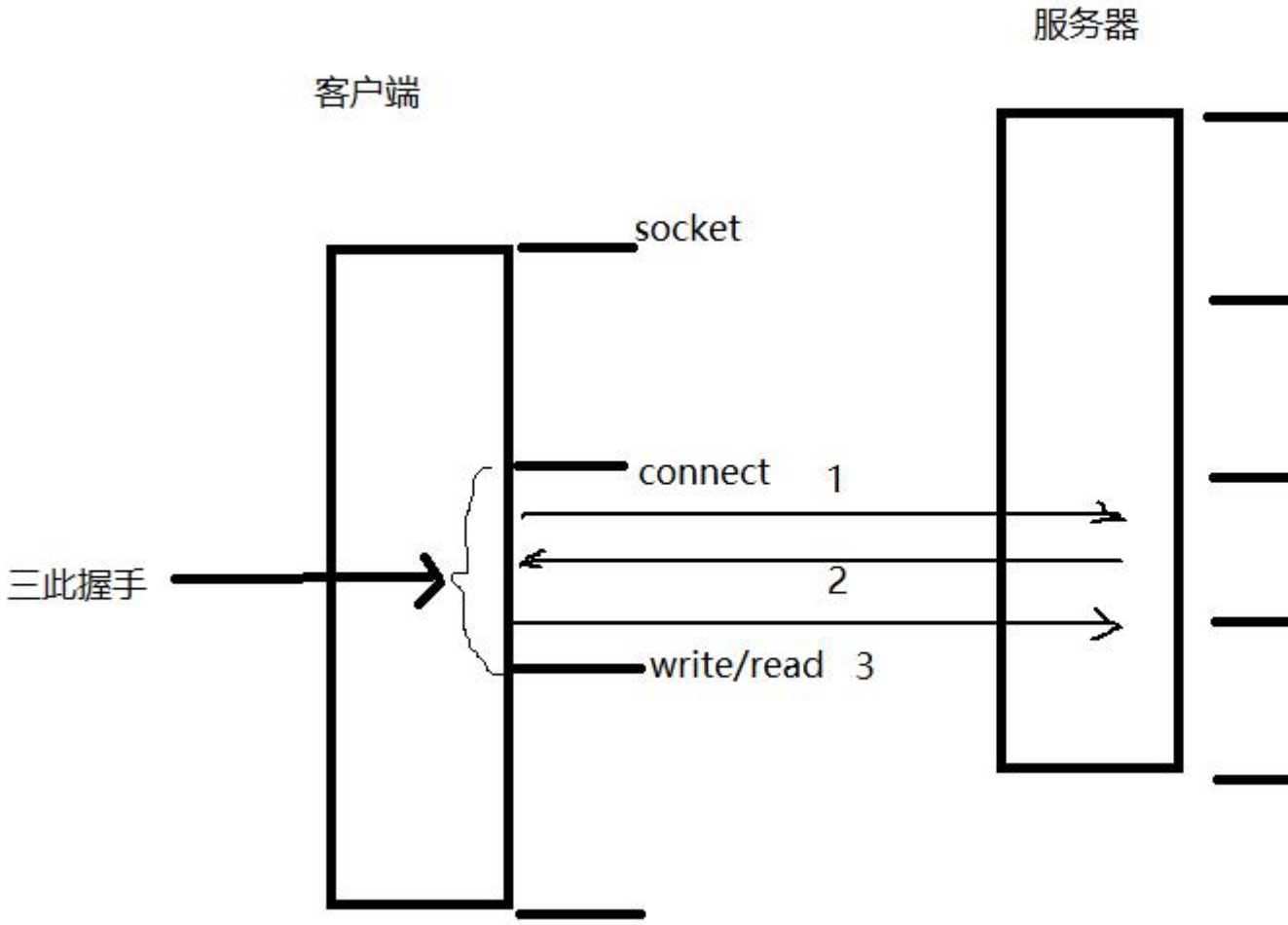
传出参数，返回链接客户端地址信息，含 IP 地址和端口号

addrlen:

传入传出参数（值-结果）,传入 sizeof(addr)大小，函数返回时返回真正接收到地址结构体的大小

返回值：

成功返回一个新的 socket 文件描述符,用于和客户端通信,失败返回-1,设置 errno



三次握手过程

三次握手完成后，服务器调用 accept()接受连接，如果服务器调用 accept()时还没有客户端的连接请求，就阻塞等待直到有客户端连接上来。addr 是一个传出参数，accept()返回时传出客户端的地址和端口号。addrlen 参数是一个传入传出参数（value-result argument），传入的是调用者提供的缓冲区 addr 的长度以避免缓冲区溢出问题，传出的是客户端地址结构体的实际长度（有可能没有占满调用者提供的缓冲区）。如果给 addr 参数传 NULL，表示不关心客户端的地址。

我们的服务器程序结构是这样的：

```
while (1) {
    cliaddr_len = sizeof(cliaddr);
    connfd = accept(listenfd, (struct sockaddr *)&cliaddr, &cliaddr_len);
    n = read(connfd, buf, MAXLINE);
```

```
    ......
    close(connfd);
}
```

整个是一个 while 死循环,每次循环处理一个客户端连接。由于 cliaddr_len 是传入传出参数，每次调用 accept()之前应该重新赋初值。accept()的参数 listenfd 是先前的监听文件描述符，而 accept()的返回值是另外一个文件描述符 connfd，之后与客户端之间就通过这个 connfd 通讯，最后关闭 connfd 断开连接，而不关闭 listenfd，再次回到循环开头 listenfd 仍然用作 accept 的参数。accept()成功返回一个文件描述符，出错返回-1。

2.3.5 connect 函数

```
#include <sys/types.h>                  /* See NOTES */
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

sockdf:

socket 文件描述符

addr:

传入参数，指定服务器端地址信息，含 IP 地址和端口号

addrlen:

传入参数,传入 sizeof(addr)大小

返回值：

成功返回 0，失败返回-1，设置 errno

客户端需要调用 connect()连接服务器，connect 和 bind 的参数形式一致，区别在于bind的参数是自己的地址,而connect的参数是对方的地址。connect()成功返回 0，出错返回-1。

2.3.6 出错处理函数

我们知道，系统函数调用不能保证每次都成功，必须进行出错处理，这样一方面可以保证程序逻辑正常，另一方面可以迅速得到故障信息。

出错处理函数

```
#include <errno.h>
#include <string.h>
char *strerror(int errnum);      /* See NOTES */
```

errnum:

传入参数,错误编号的值，一般取 errno 的值

返回值：

错误原因

```
#include <stdio.h>
#include <errno.h>
void perror(const char *s);     /* See NOTES */
```

s:

传入参数,自定义的描述

返回值：

无

向标准出错 stderr 输出出错原因

范例：

```
#include <stdio.h>

#include <stdlib.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>



#define IP "1.1.1.1"



#define SERVER_PORT 666




perror_exit(const char * des){

	//fprintf(stderr, "%s error, reason: %s\n", des, strerror(errno));

	perror(des);

	exit(1);

}
```

```
int main(void){


    int sock;//代表信箱

    int i, ret;

    struct sockaddr_in server_addr;




    //1.美女创建信箱

    sock = socket(AF_INET, SOCK_STREAM, 0);


    if(sock == -1){

        perror_exit("create socket");

    }


    //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

    inet_pton(AF_INET, IP, &server_addr.sin_addr.s_addr);

    //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
ret = bind(sock, (struct sockaddr *)&server_addr,  sizeof(server_addr));
if(ret == -1){
    perror_exit("bind");
}

//把信箱挂置到传达室，这样，就可以接收信件了
ret = listen(sock, 128);
if(ret == -1){
    perror_exit("listen");
}

//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];


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));
/*读取客户端发送的数据*/
len = read(client_sock, buf, sizeof(buf)-1);
buf[len] = '\0';
printf("recive[%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("write finished. len: %d\n", len);
            close(client_sock);

        }
        return 0;
}
```

2.4 项目练习

将回声服务器的服务器端和客户端进行改造，实现客户端和服务器端完成一段简单的日常对话，以下 C 代表客户端，S 代表服务器端:

C: Do you like C++?
S: Yes, I do.
C Why do you like C++?
S: Because I can use it to write programs.
S: And you ?
注： 话音未落，服务器发送完 “And you ?”后，调用 close 函数故意把与客户端的连接关闭了。。。
接下来，客户端必须重连到服务器发送以下信息，然后双方再友好的关闭连接。
C: Me too.

项目 5 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，欢迎来到动脑学院 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>
```

2. 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。

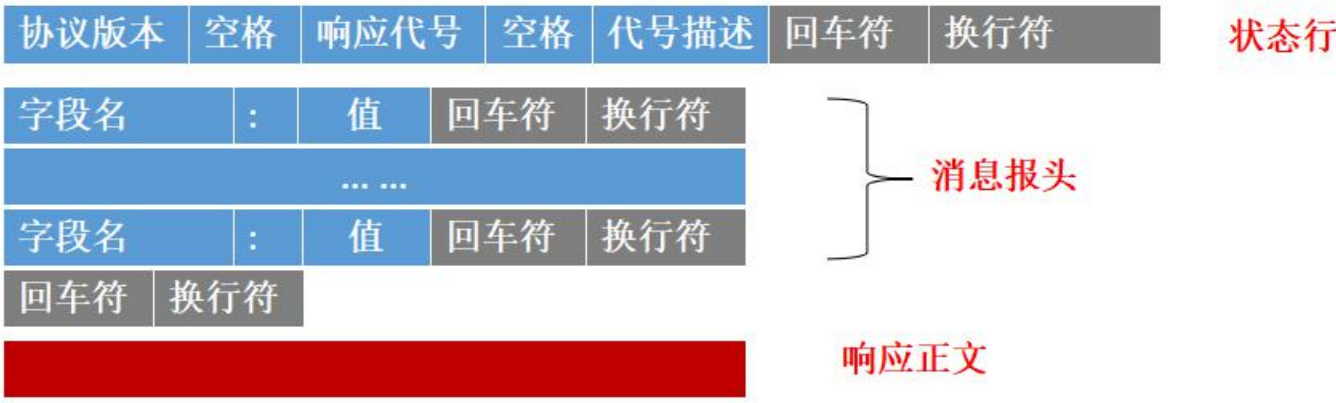
请求格式：

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：

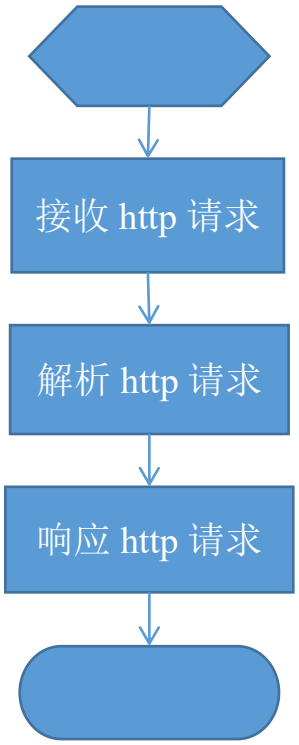
```
GET /demo.html HTTP/1.1
Host: 47.100.162.191
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.26 Safari/537.36 Core/1.63.6788.400 QQBrowser/10.3.2767.400
Upgrade-Insecure-Requests: 1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
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，欢迎来到动脑学院 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。。。。



接收 http 请求

1. 实现按行读取请求头部

```
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;
            }

            buf[count] = ch;
            count++;

        }else if(len ==-1){ //读取出错
            perror("read failed.");
            break;
        }else {// 客户端 socket 关闭,read 返回 0
            fprintf(stderr, "client close.\n");
            break;
        }
    }
    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
        len = get_line(client_sock, buf, sizeof(buf));

        if(len>0){
            int i=0, j=0;
            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=0;
            while(!isspace(buf[j]) && ( i< sizeof(url)-1)){
                url[i]=buf[j];
                i++;
                j++;
            }
            url[i]='\0';

            printf("url: %s\n", url);


            }else {
                printf("other requst = %s\n", method);
            }

        }else {//出错的处理


        }
```

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，欢迎来到动脑学院 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\n
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 Method 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>
```

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>
```