网络编程技术

作者：TX

QQ: 10060502

2021年 11月 15日

目录

1. 课程介绍

- 目标要求
- 学习方法
- 参考资料
- 考核标准

总体目标

使学生理解网络编程相关理论和实践基础知识，掌握C语言编程环境下编写网络通信程序的能力，从而加深学生对计算机网络相关知识的理解，提高学生的实践动手能力。

总体要求

1. 了解网络相关基本术语概念
2. 理解网络编程相关模型架构
3. 熟悉不同系统平台网络编程差异
4. 掌握网络编程开发环境搭建使用方法
5. 掌握网络编程通用接口使用方法
6. 熟悉高级输入输出使用方法
7. 理解多进程多线程使用方法

1. 课程介绍

- 目标要求
- 学习方法
- 参考资料
- 考核标准

学习方法

1. 练习英文检索
2. 阅读官方文档
3. 养成盲打习惯
4. 编写大量程序
   - 读需求
   - 画流程
   - 写程序
   - 做比较

(抄袭可耻！！！)

1. 课程介绍

- 目标要求
- 学习方法
- 参考资料
- 考核标准

参考资料——书籍

1. 《UNIX Network Programming》（《UNIX网络编程》卷1、卷2）
2. 《TCP/IP Illustrated》（《TCP/IP协议详解》卷1、卷2、卷3）
3. 《Advanced Programming in the UNIX Environment》（《UNIX环境高级编程》）
4. Richard Stevens网站:

   http://www.kohala.com/start/

参考资料——手册

1. RFC1180-TCP/IP Tutorial（TCP/IP教程）：

   https://tools.ietf.org/html/rfc1180

2. RFC2616-Hypertext Transfer Protocol–HTTP/1.1（超文本传输协议）：

   https://www.rfc-editor.org/rfc/pdfrfc/rfc2616.txt.pdf

3. RFC826-An Ethernet Address Resolution Protocol（以太网地址解析协议）：

   https://www.rfc-editor.org/rfc/pdfrfc/rfc826.txt.pdf

4. RFC792-Internet Control Message Protocol（互联网控制消息协议）：

   https://www.rfc-editor.org/rfc/pdfrfc/rfc792.txt.pdf

5. FreeBSD套接字开发：

   https://docs.freebsd.org/en/books/developers-handbook/sockets/

6. 微软套接字开发：

   https://docs.microsoft.com/en-us/windows/win32/winsock/

参考资料——系统

1. FreeBSD官方网站：

   https://www.freebsd.org/

2. FreeBSD中文手册：

   https://docs.freebsd.org/zh-cn/books/handbook/

3. UNIX开放组织：

   https://unix.org/

4. UNIX社区：

   https://unix.com/

5. Debian GNU/Linux官方网站：

   https://www.debian.org/

6. Debian GNU/Linux中文手册

   https://www.debian.org/doc/user-manuals.zh-cn.html

参考资料——工具

1. MingGW-w64 C语言开发环境：
   http://mingw-w64.org/
2. VIM编辑工具：
   https://www.vim.org/
3. 微软Visual Studio Code编辑工具：
   https://code.visualstudio.com/
4. 微软Visual Studio开发环境：
   https://visualstudio.microsoft.com/
5. Libpcap接口：
   https://www.tcpdump.org/
6. WinPcap接口：
   https://www.winpcap.org/
7. npcap接口：
   https://nmap.org/npcap/
8. Libnet接口：
   http://libnet.sourceforge.net/

1. 课程介绍

- 目标要求
- 学习方法
- 参考资料
- 考核标准

考核方式

- 最终成绩：平时成绩（30%）+期末成绩（70%）
- 平时成绩：
  - 考勤成绩（30%）：理论实验随堂考勤
  - 作业成绩（20%）
  - 测试成绩（20%）：期中期末随堂测试
  - 实验成绩（30%）：实验报告平均成绩
- 期末成绩：
  - 考试：闭卷考试
  - 考查：课程论文、大型作业、开卷考试（期末确定）

评分标准

- 考勤成绩:
  - 迟到1次扣5分
  - 缺勤1次扣10分
- 实验成绩:
  - 基准90分
  - 工整清晰无误，加10分
  - 文字潦草混乱，扣10分
  - 图表模糊混乱，扣10分
  - 缺少报告项目，每项扣10分
  - 不能按时提交，每次扣10分

注意事项

发生以下任意情况按照不及格处理

- 缺勤超过3次
- 没有按时完成作业超过3次
- 没有参加测试
- 没有按时提交报告超过3次
- 学期结束没有提交报告
- 任意一项成绩低于60分

2. 网络编程基础

- 计算机系统
- 计算机网络
- 分布式系统
- 系统模型
- 应用接口
- 开发环境
- 语言速览

计算机系统

计算机系统由硬件系统和软件系统两大部分组成:

- 硬件系统:
  - 主机
    - 中央处理器（Central Processing Unit）:
      运算器（Arithmetic Logical Unit）
      控制器（Control Unit）
      寄存器（Register）
    - 存储器（Memory Unit）
  - 外部设备
    - 输入设备（Input Device）
    - 输出设备（Output Device）
- 软件系统:
  - 系统软件
    - 操作系统（Operating System）
    - 程序语言
  - 应用软件
    - 文档处理
    - 图形图像处理
    - 音频视频处理
    - 虚拟现实
    - 人工智能
    - ……

2. 网络编程基础

- 计算机系统
- 计算机网络
- 分布式系统
- 系统模型
- 应用接口
- 开发环境
- 语言速览

基本概念

- 网络（Network）：一组连接的通信设备。
- 互联网络（internet）：可以互相通信的两个或两个以上网络（network）。
- 国际互联网络（Internet）：成百上千的互连网络（Internet）。
- 计算机网络：将**独立主机**、**通信设备**、**通信线路**等**硬件**连接起来，通过**软件**实现**信息传递**和**资源共享**的系统。

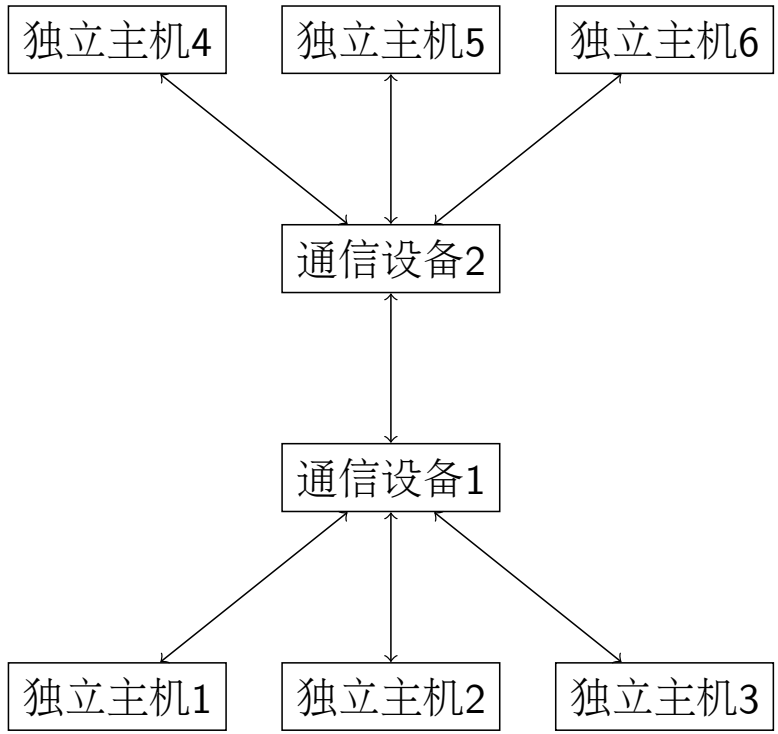


图: 计算机网络

历史发展——国外

- 1969年，ARPA（Advanced Research Projects Agency）建立了4个节点的ARPANET
- 1970年，ARPANET使用了NCP（Network Control Protocol）协议
- 1981年，NSF（National Science Foundation）创建了CSNET，供大学使用
- 1983年，TCP/IP成为ARPANET官方协议
- 1983年，ARPA从ARPANET中分离出MILNET，专供军事使用
- 1990年，NSF创建了由5个超级计算机组成的NSFNET，替换了ARPANET
- 1990年，IBM、Merit、MCI成立ANS（Advanced Network and Services），组建ANSNET
- 1995年，NSFNET转为研究性网络
- 1995年，出现ISP（Internet Service Provider）公司，提供Internet服务

历史发展——国内

- 1986.08 中国科学院高能物理研究所在北京通过卫星登录到日内瓦发出一封电子邮件
- 1990.11 中国的顶级域名.CN完成注册，服务器暂时设在德国卡尔斯鲁厄大学
- 1994.07 清华大学等六所高校"中国教育和科研计算机网（CERNET）"试验网开通
- 1994.09 邮电部电信总局开始建设中国公用计算机互联网(ChinaNET),1996.01开通
- 1996.02 国务院发布了《中华人民共和国计算机信息网络国际联网管理暂行规定》
- 1997.10 CNNIC发布第一次中国互联网络发展状况调查统计报告:
  http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/200905/P020120709345374625930.
- 1997.12 公安部发布了《计算机信息网络国际联网安全保护管理办法》
- 1998.06 CERNET正式参加下一代IP协议(IPv6)试验网6BONE
- 2000.05 中国移动互联网(CMNET)投入运行
- 2000.07 中国联通公用计算机互联网(UNINET)正式开通

分层模型

1. 两个实体（Entity）通过协议（Protocol）通讯
2. 协议通过分层实现安全通信

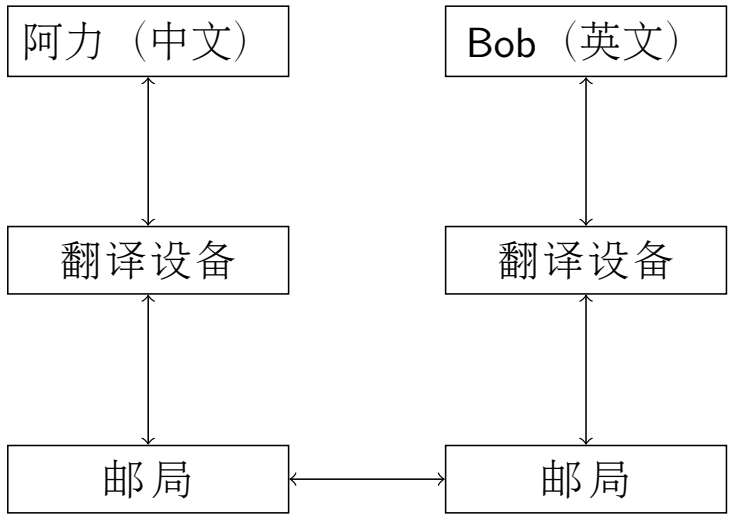


图: 分层模型

OSI模型

OSI模型，又称开放系统互联模型，分为：

1. 应用层（Application Layer）
2. 表示层（Presentation Layer）
3. 会话层（Session Layer）
4. 传输层（Transport Layer）
5. 网络层（Network Layer）
6. 链路层（Data Link Layer），又称数据链路层
7. 物理层（Physical Layer）

OSI模型——物理层

物理层通过介质传输二进制位，提供机械或电子实现，主要关注：

- 接口介质特征（Physical Characteristics of Interfaces and Media）
- 二进制位表示（Representation of Bits）
- 数据传输速率（Data Rate）
- 二进制位同步（Synchronization of Bits）
- 连接线路配置（Line Configuration）
- 物理拓扑结构（Physical Topology）
- 数据传输模式（Transmission mode）

OSI模型——链路层

链路层以分帧方式管理二进制位，提供hop-to-hop服务，主要关注：

- 分帧管理（Framing）
- 物理寻址（Physical Addressing）
- 流动控制（Flow Control）
- 错误控制（Error Control）
- 访问控制（Access Control）

OSI模型——网络层

网络层将分包从来源地址移动到目的地址，提供网络服务，主要关注：

- 逻辑寻址（Logical Addressing）
- 路线安排（Routing），又称路由访问

OSI模型——传输层

传输层提供进程到进程消息传送和错误恢复功能，主要关注：

- 服务寻址（Service-Point Addressing）
- 分段装配（Sementation and Reassembly）
- 连接控制（Connection Control）
- 流动控制（Flow Control）
- 错误控制（Error Control）

OSI模型——会话层

会话层创建、管理、销毁会话，主要关注:

- 对话控制（Dialog Addressing）
- 同步控制（Synchronization）

OSI模型——表示层

表示层翻译、加密、压缩数据，主要关注：

- 翻译（Translation）
- 加密（Encryption）
- 压缩（Compression）

OSI模型——应用层

应用层用于访问网络资源，主要关注：

- 虚拟终端（Network Virtual Terminal）
- 文件服务（File Transfer, Access, and Management）
- 邮件服务（Email Services）
- 目录服务（Directory Services）

TCP/IP模型

四层模型

- 应用层（Application Layer）
- 传输层（Transport Layer）
- 网络层（Internet Layer）
- 接口层（Network Interface Layer），又称网络接口层

五层模型

- 应用层（Application Layer）
- 传输层（Transport Layer）
- 网络层（Network Layer）
- 链路层（Data Link Layer）
- 物理层（Physical Layer）

TCP/IP模型——与OSI之间的关系

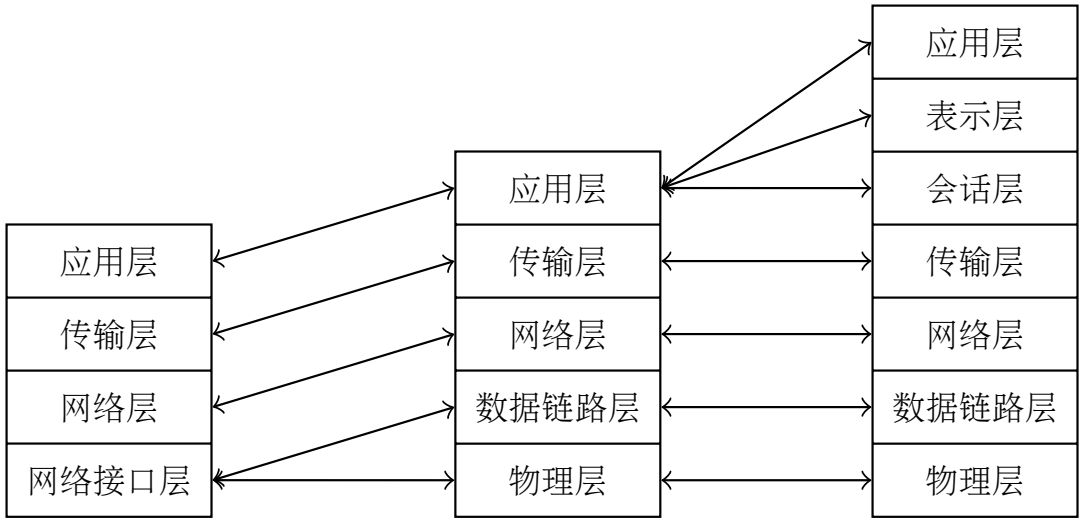


图: TCP/IP四层、五层和OSI七层之间的关系

TCP/IP模型——传输单元

- 应用层：消息（Message）
- 传输层：
  - TCP数据段（TCP Segment）
  - UDP数据报（UDP Datagram）
- 网络层：数据包（Packet）
- 链路层：数据帧（Frame）
- 物理层：数据位（Bit）

TCP/IP模型——常见协议

- 应用层：
  - 基于TCP：HTTP、FTP、SMTP、POP3、IMAP、IRC
  - 基于UDP：DHCP、DNS、NTP
- 传输层：TCP、UDP
- 网络层：IP、ICMP
- 接口层：802.3（Ethernet）、802.11（Wifi等）、PPP

TCP/IP模型——寻址方式

- 应用层：应用地址（Application-Specific Addresses），又称应用特定地址
- 传输层：端口地址（Port Address）
- 网络层：逻辑地址（Logical Address）
- 链路层：物理地址（Physical Address），又称链路地址
- 物理层

TCP/IP模型——寻址方式——物理地址

- 以太网（Ethernet）MAC地址
  - 大小：6字节（48位）
  - 表示：冒号分割6个16进制表示的字节
  - 示例：FF:FF:FF:00:11:22
  - 规则：前3个字节为组织唯一标识（OUI），由IEEE分配给组织
- 苹果公司的LocalTalk网络使用1字节的地址

TCP/IP模型——寻址方式——逻辑地址

IP地址：

- IPv4
- IPv6

TCP/IP模型——寻址方式——逻辑地址——IPv4

- 表示形式
  - 数值形式： 4字节整型数据

    如：0x7f000001
  - 字符形式： 点分十进制

    如：127.0.0.1
- 地址分类
  - A类：网络地址（0开头，8位）+主机地址（24位）
  - B类：网络地址（10开头，16位）+主机地址（16位）
  - C类：网络地址（110开头，24位）+主机地址（8位）
  - D类：组播地址（1110开头）
  - E类：保留地址（11110开头）
- 特殊地址
  - 私有地址
    - 10.0.0.0-10.255.255.255
    - 172.16.0.0-172.31.255.255
    - 192.168.0.0-192.168.255.255
  - 回环地址：127.0.0.0-127.255.255.255
  - 任意主机、所有主机、默认路由：0.0.0.0
  - 广播地址
    - 255.255.255.255
    - 主机地址全1
  - 组播地址

TCP/IP模型——寻址方式——逻辑地址——IPv6

- 表示形式
  - 数值形式： 16字节整型数据
  - 字符形式：

    如：0000:0000:0000:0000:0000:ffff:7f00:0001、::ffff:127.0.0.1
- 特殊地址

TCP/IP模型——寻址方式——端口地址

端口（Port），使用2字节（16位）表示，取值范围为0-65535

常见端口：

- HTTP：80（TCP）
- FTP：21（TCP）
- RTMP：1935（TCP）
- DNS：53（68）
- DHCP：21（68）

TCP/IP模型——寻址方式——应用地址

- 网站地址：

  如： http://www.bbc.edu.cn

- 邮箱地址：

  如： 10060502@qq.com

2. 网络编程基础

- 计算机系统
- 计算机网络
- 分布式系统
- 系统模型
- 应用接口
- 开发环境
- 语言速览

基本概念

分布式系统：硬件或软件分布在网络中的计算机上、软件之间通过传递消息进行通信和动作协调的系统。

总体特征

- 并发
- 缺乏全局时钟
- 故障独立

应用实例

- 搜索引擎
- 大型多人在线游戏
- 金融交易

发展趋势

- 泛在联网：多种类型计算机网络
- 无处不在：移动计算到无处不在
- 多种媒体：离散媒体的存储传输
- 公共设施：网络存储和软件服务

面临挑战

- 异构性：如虚拟机、中间件
- 开放性：如RFC、W3C
- 安全性：如机密性、完整性、可用性
- 伸缩性：如控制物理资源开销、防止软件资源用尽、控制性能损失、避免性能瓶颈
- 容错性：如检测故障、掩盖故障、故障容错、故障恢复、冗余组件
- 并发性：如进程同步、事务同步
- 透明性：如访问透明性、位置透明性、并发透明性、复制透明性、故障透明性、移动透明性、性能透明性、伸缩透明性
- 质量性：如可靠性、安全性、高性能

2. 网络编程基础

- 计算机系统
- 计算机网络
- 分布式系统
- 系统模型
- 应用接口
- 开发环境
- 语言速览

物理模型

三代:

- 早期的分布式系统
- 互联网分布式系统
- 当代的分布式系统

物理模型——早期的分布式系统

早期的分布式系统由通过局域网互联的10-100个结点组成，与互联网连接并有限支持少量的服务

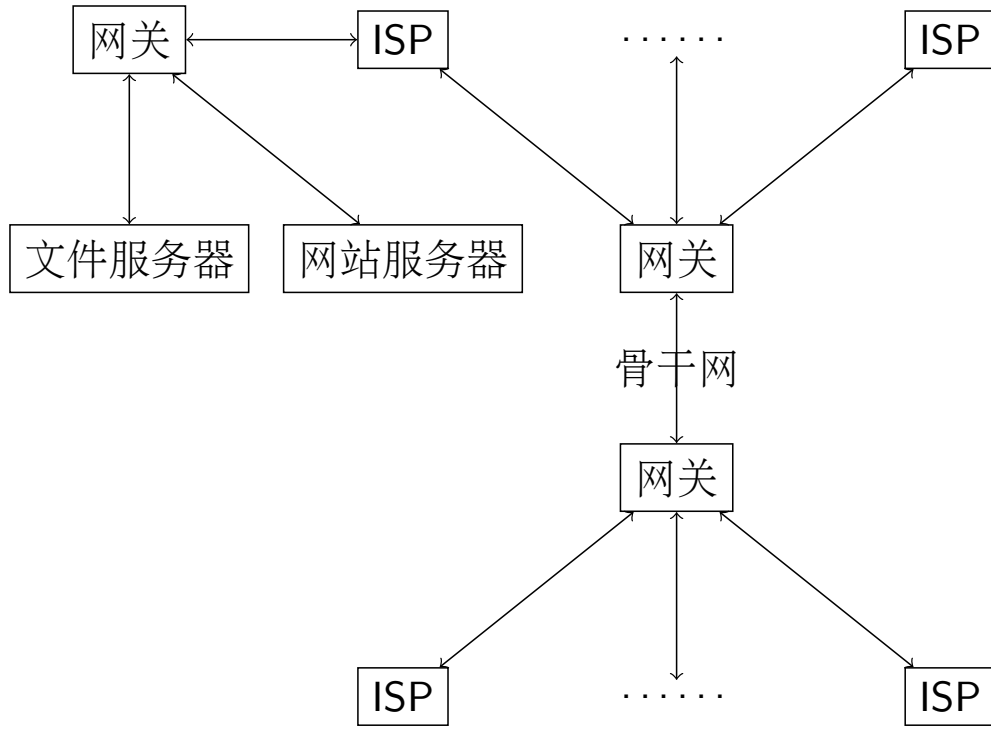


图: 早期的分布式系统

物理模型——互联网分布式系统

互联网分布式系统通过互联网将节点相互连接，为全球化组织提供分布式系统服务

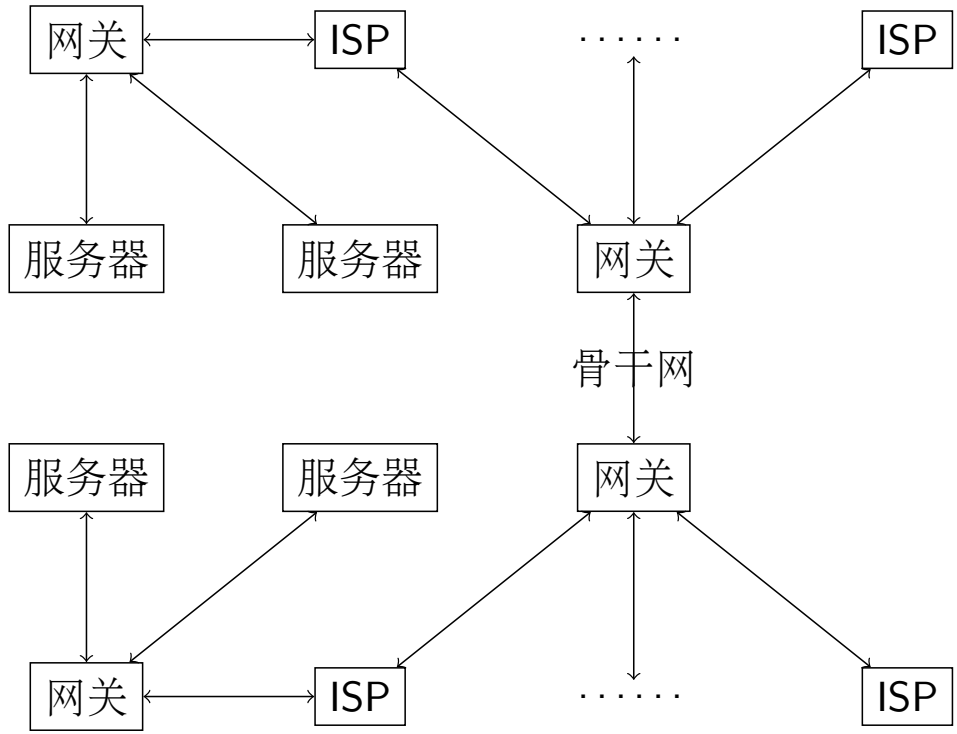


图: 互联网分布式系统

物理模型——当代的分布式系统

当代的分布式系统:

- 移动计算
- 无处不在计算
- 云计算

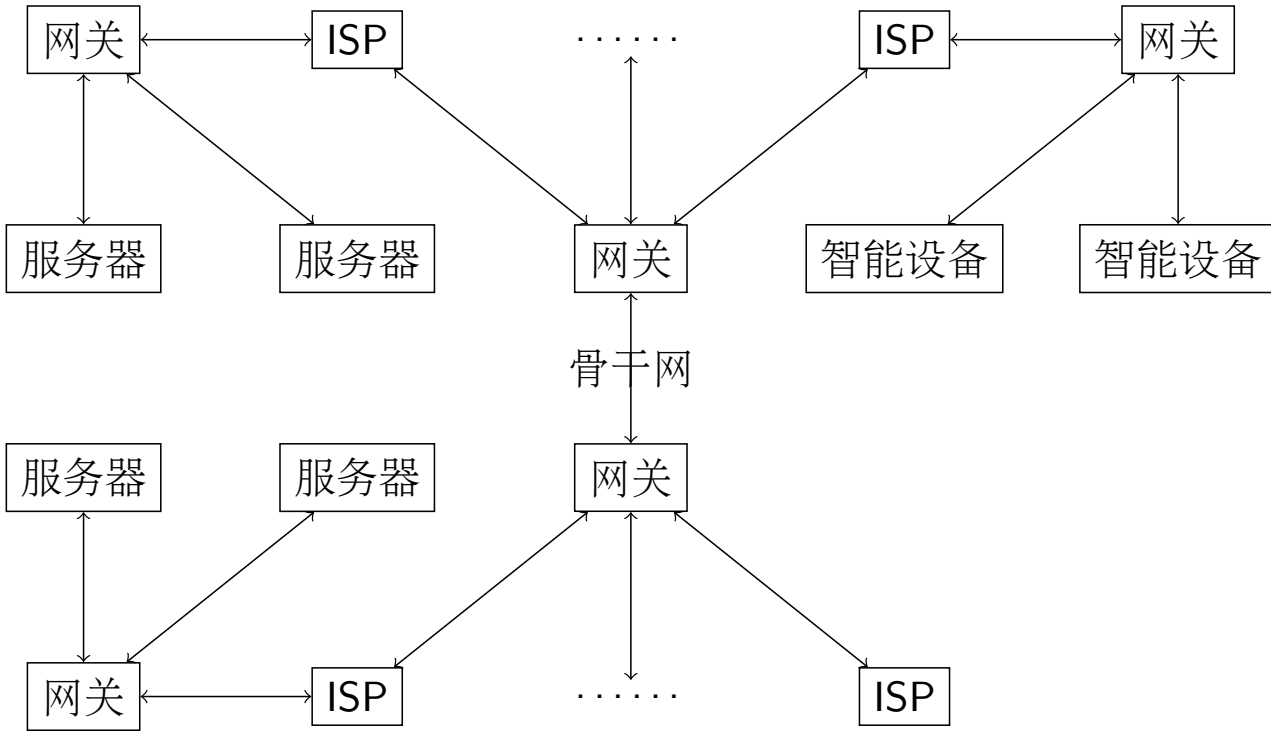


图: 当代的分布式系统

结构模型——结构元素

通信实体（What）：

- 面向系统角度
  - 进程
  - 线程
  - 结点
- 面向问题角度
  - 对象
  - 组件
  - 服务

通信范型（How）：

- 进程通信:
  - 套接字（Socket）
- 远程调用
  - 请求应答协议
  - 远程过程调用（RPC）
  - 远程方法调用（RMI）
- 间接调用
  - 组通信
  - 发布订阅
  - 消息队列
  - 元组空间
  - 分布式共享内存

角色责任（Which）：

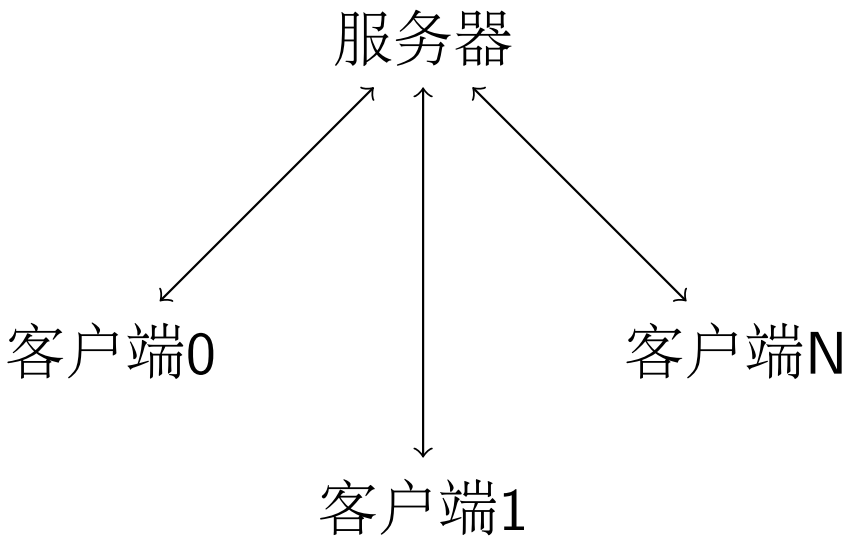
- 客户服务器风格
  - 客户角色
  - 服务器角色
- 对等风格:
  - 对等方角色

放置位置（Where）：

- 多服务器
- 缓存代理
- 移动代码
- 移动代理

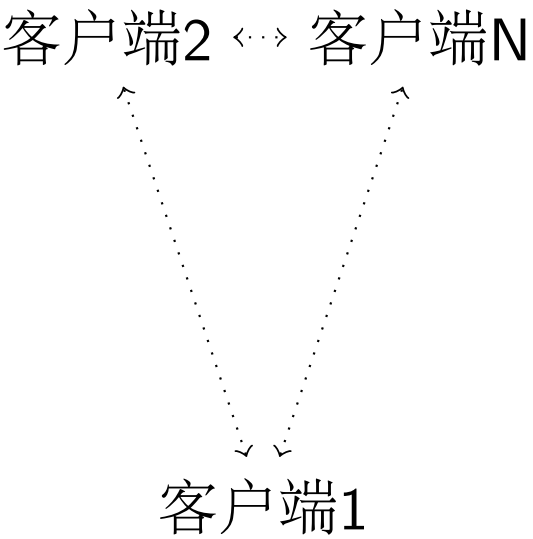
客户服务器结构风格

典型的客户/服务器（C/S，Client/Server）结构风格



对等结构风格

典型的对等（P2P，Peer to Peer，Point to Point）结构风格

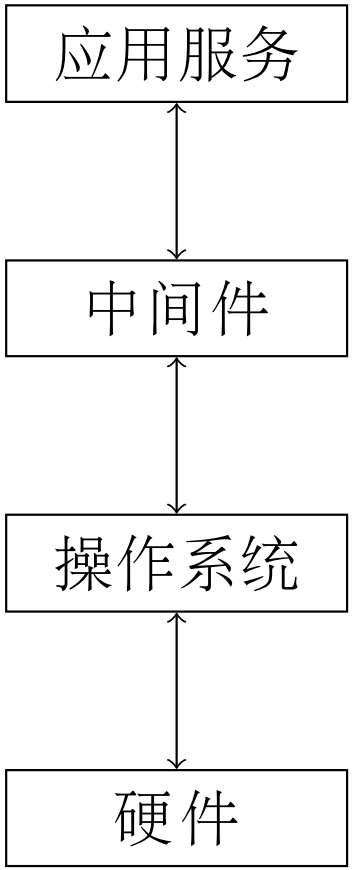


结构模型——结构模式

- 分层体系结构
- 层次体系结构
- 瘦客户
- 代理
- 反射

结构模型——结构模式——分层体系结构

分层体系结构：确定抽象层次

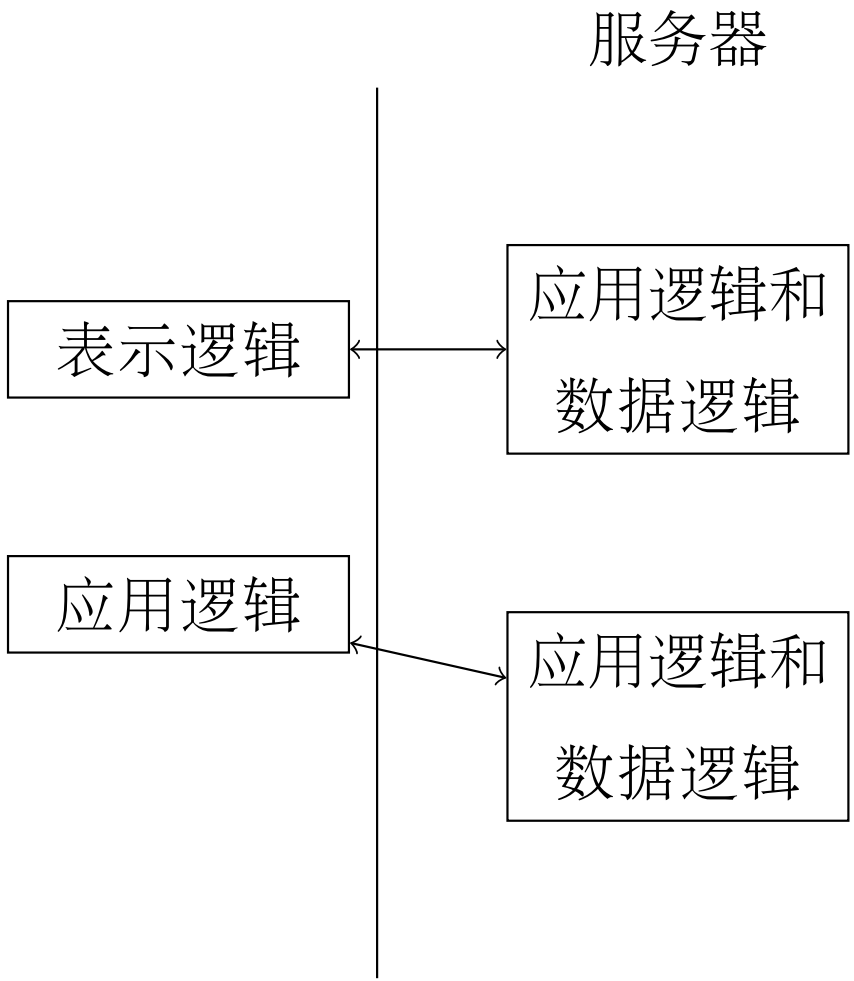


结构模型——结构模式——层次体系结构

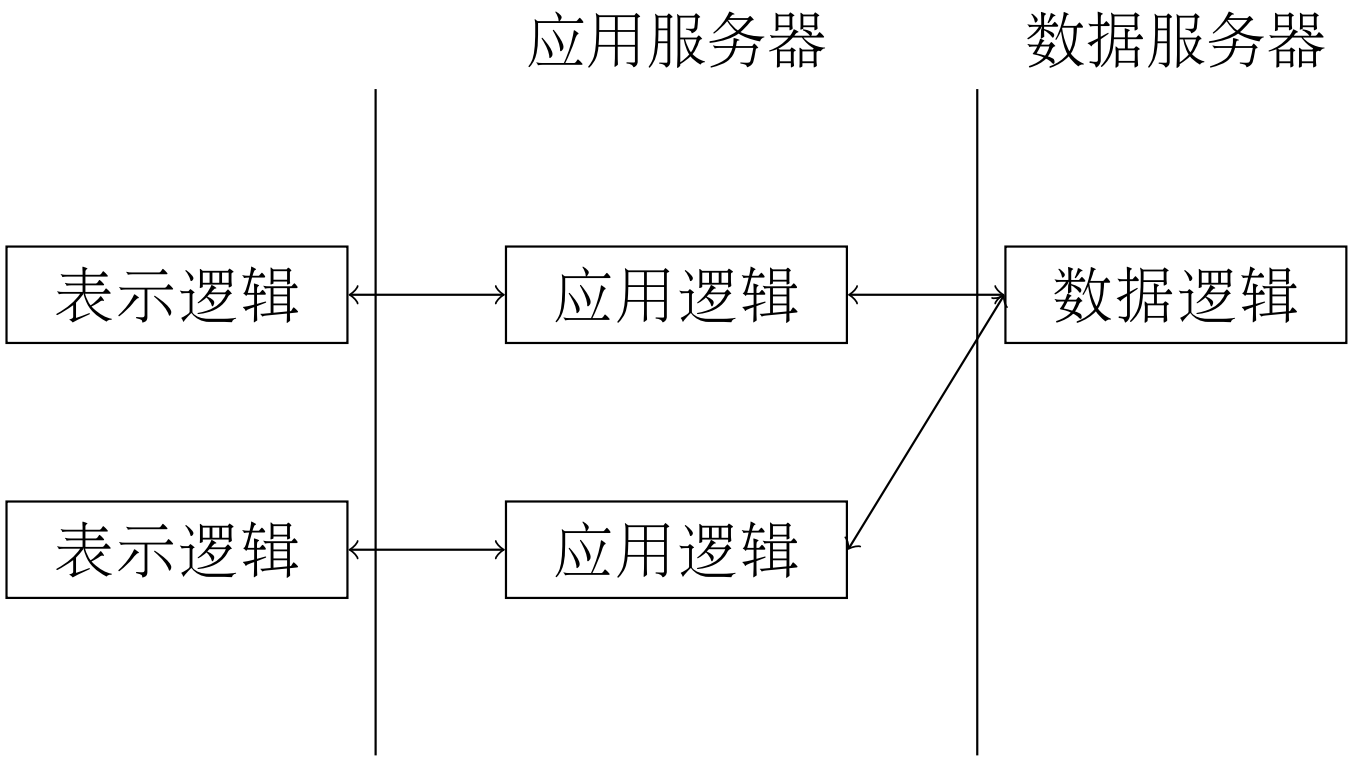
层次体系结构：确定功能位置

- 表示逻辑：用户视图
- 应用逻辑：业务逻辑
- 数据逻辑：持久存储

结构模型——结构模式——层次体系结构——两层



结构模型——结构模式——层次体系结构——三层



2. 网络编程基础

- 计算机系统
- 计算机网络
- 分布式系统
- 系统模型
- 应用接口
- 开发环境
- 语言速览

应用层

- Windows操作系统
  - WinInet
  - WinHttp
  - Http Server API
  - .Net Core
- 跨平台
  - libmicrohttpd
  - libhttpd
  - libsoup

传输层

- Windows操作系统
  - Windows套接字（WinSock）
- 跨平台
  - 套接字（Socket）
  - libevent
  - libev

网络层

- Windows操作系统
  - Windows套接字
- 跨平台
  - 套接字

链路层

- Windows操作系统
  - Windows套接字
  - WinPcap
- 跨平台
  - 套接字
  - libpcap
  - libnet

2. 网络编程基础

- 计算机系统
- 计算机网络
- 分布式系统
- 系统模型
- 应用接口
- 开发环境
- 语言速览

基本概念

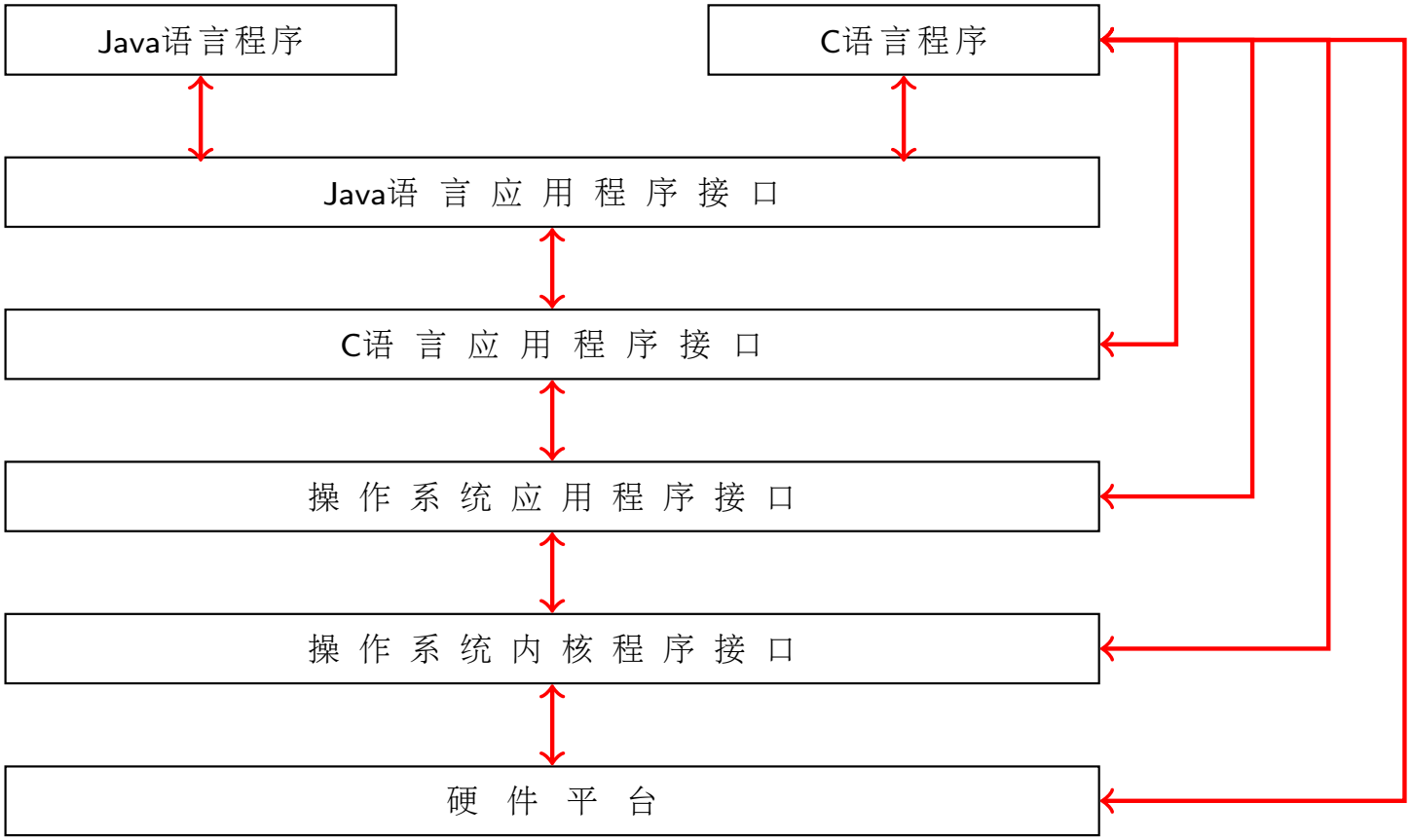


图: 开发环境

基本概念

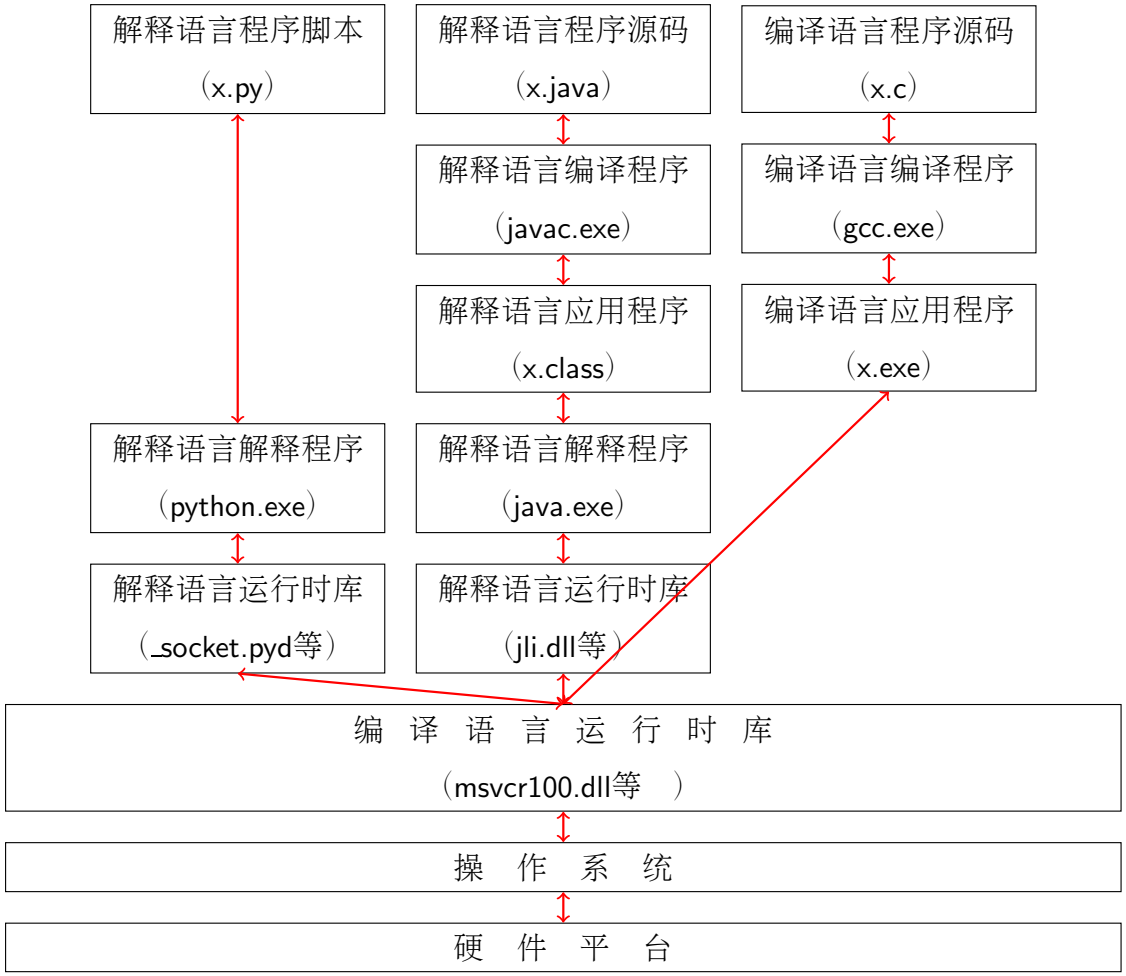


图: 开发环境

硬件平台

按CPU平台架构主要分为：

复杂指令集计算机（Complex Instruction Set Computer, CISC）：

- Intel i386/i486/i586/i686，x86_64
- amd64

精简指令集计算机（Reduced Instruction Set Computer, RISC）：

- ARM、AArch64
- MIPS、MIPS64、MIPSEL、MIPS64EL
- RISC-V

操作系统

开源系统:

- Debian 8/9/10

  官网：https://debian.org

- FreeBSD 11/12

  官网：http://freebsd.org

商业系统:

- Microsoft Windows

  桌面版本：XP/7/8.1/10

  服务版本： Microsoft Windows Server 2003/2008/2012/2016/2019

  推荐：Windows Server 2003或Window XP

- Apple macOS

  官网：https://www.apple.com/macos

  列表：https://support.apple.com/en-us/HT2012

虚拟软件

开源软件：

- Oracle VirtualBox

  官网：https://virtualbox.org

  推荐：v4.0.24（支持XP）

- qemu

  官网：https://qemu.org/

  其他：https://qemu.weilnetz.de/

  推荐：v20160903（支持XP）

商业软件：

- Microsoft Hyper-V
- VMWare

编译工具

编译型:

- C
- Go

解释型:

- Java
- Python

开发环境——编译工具——C语言

- Windows

  使用mingw-w64、mingw32-make、gdb

  官网：http://www.mingw-w64.org

  推荐：i686-win32-dwarf，https://sourceforge.net/projects/mingw-w64/file

- Debian GNU/Linux

  使用gcc、make、gdb

  apt-get install build-essential

- FreeBSD

  使用clang、make、gdb

开发环境——编译工具——Java语言

- Windows
  - 开源：

    官网：http://openjdk.java.net/

    推荐：Java SE 7(Windows i586 Binary), https://jdk.java.net/java-se-ri/7
  - 商业：

    官网：https://www.oracle.com/java/technologies/javase-downloads.html
- Debian GNU/Linux

  默认已安装OpenJDK
- FreeBSD

  默认已安装OpenJDK

源码编辑

- VI
- Emacs
- Microsoft Visual Studio Code
- ATOM
- Sublime
- Windows记事本

测试实例

- 需求

  输出Hello World

- 源码

  src/hello.c

```
1 #include <stdio.h>
2 int main(void)
3 {
4         printf("Hello World\n");
5         return 0;
6 }
```

- 编译

```
1 hello:
2         i686-w64-mingw32-gcc -Wall -g -o bin/hello.i686
              .exe hello.c
3         x86_64-w64-mingw32-gcc -Wall -g -o bin/hello.
              amd64.exe hello.c
```

- 运行

  src/bin/hello.i686.exe

2. 网络编程基础

- 计算机系统
- 计算机网络
- 分布式系统
- 系统模型
- 应用接口
- 开发环境
- 语言速览

语言速览

- 进制转换
- C
- Java

语言速览——进制转换

- 十进制
- 二进制
- 八进制
- 十六进制
- 进制转换
  - 十进制转为其他进制
  - 其他进制转为十进制
  - 1位八进制转为3位二进制
  - 1位十六进制转为4位二进制
- 练习
  - 49转为十六进制
  - 0x20转为十进制
  - 0x65转为二进制

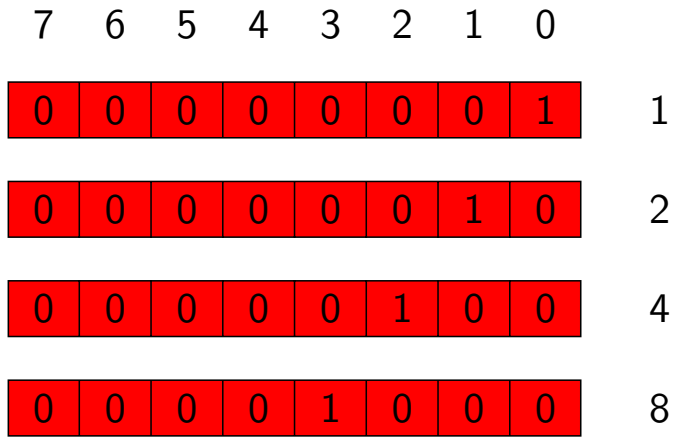


图: 进制转换

语言速览——C语言

数据类型：

- 数值类型
- 字符类型
- 数组类型
- 自定义类型
- 指针类型

数据运算：

- 括号运算
- 增减运算
- 符号运算
- 指针运算
- 算术运算
- 关系运算
- 按位运算
- 逻辑运算
- 条件运算
- 赋值运算
- 逗号运算

数据控制：

- 顺序
- 选择
- 循环

常用函数：

- 输入输出
- 文件读写

源码：`src/syntax_c.c`

语言速览——C语言——数据类型

基本语法：数据类型 变量名称[=数据初值];

- 数值类型（基本数据类型、元数据类型）
- 字符类型
- 数组类型
- 自定义类型
- 指针类型

语言速览——C语言——数值数据类型

- 1字节整数类型：char

```
24 char b1 = -1;
```

- 2字节整数类型：short int

```
27 short int b2 = 0x0102;
```

- 2/4字节整数类型：int

```
30 int b4 = 0x01020304;
```

- 4/8字节整数类型：long

```
33 long int l4 = 0x04000001;
```

- 8字节整数类型：long long int

```
36 long long int b8 = 0x0100000002;
```

- 4字节实数类型：float

```
43 float f = 3.1;
```

- 8字节实数类型：double

```
46 double lf = 3e8;
```

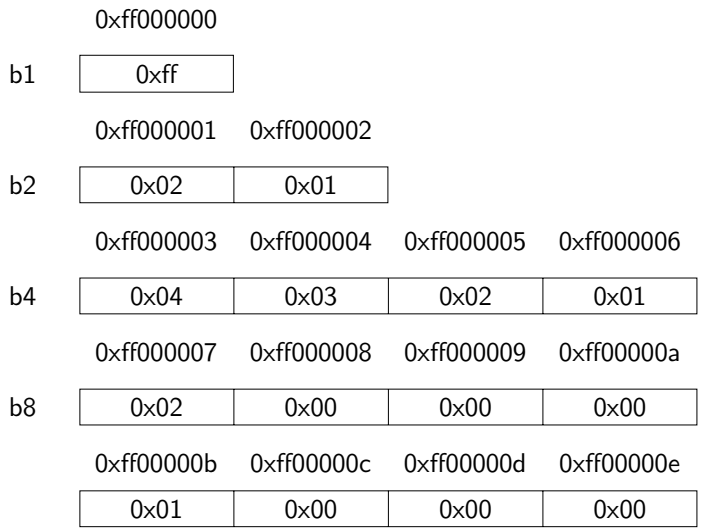


图: 数值类型

语言速览——C语言——字符数据类型

- 单字节字符类型：char

```
49  char ch = '1';
```

- 宽字节字符类型：wchar_t

```
52  wchar_t wch1 =
        0x4e00;
53  wchar_t wch2 =
        0x9fa5;
```

- 单字节字符串类型：char *

```
116 char *s = "123"
        ;
```

注：Linux系统下wchar_t为4字节

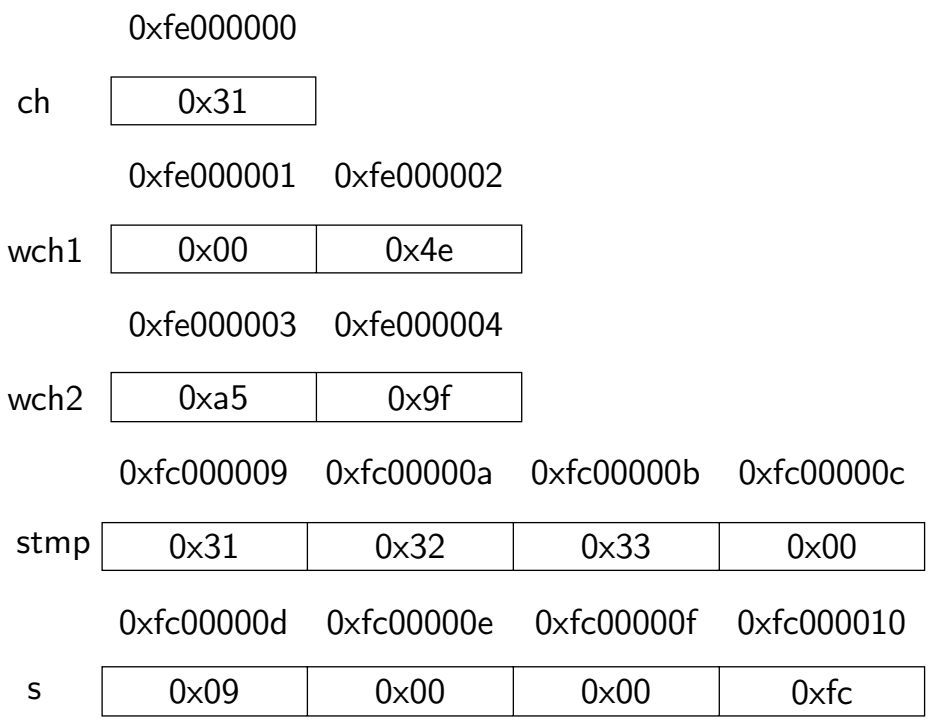


图: 字符类型

语言速览——C语言——数组类型

- 一维数组： 类型 变量[元素个数]

```
70  int arr1[4];
```

```
73  char arr2[4] = {0x31, 0x32};
```

- 二维数组： 类型 变量[行][列]

```
76  char arr3
        [2][4]={{1,2,3,4},{5,6,7,8}};
```

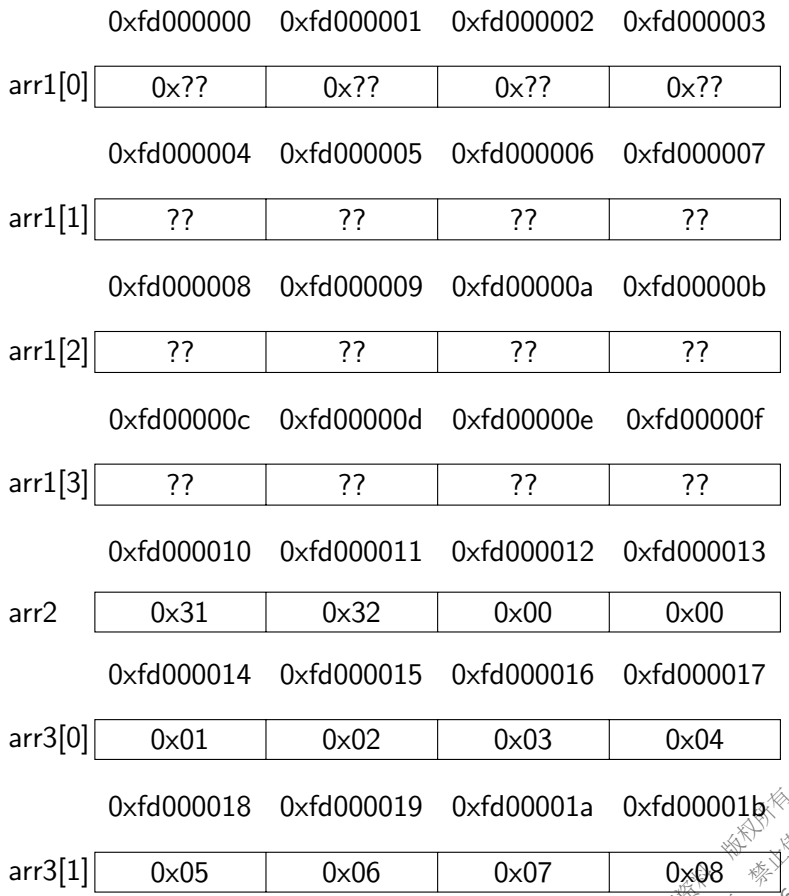


图: 数组类型

语言速览——C语言——自定义类型

- 结构类型：

```
82 struct acnt {
83  unsigned int id;
84  char name[8];
85  char pass[8];
86 };
87 struct acnt acnt1 = { 0, "Zhang", "123" };
88 acnt1.id = 1;
```

```
91 struct auth {
92  unsigned long long token;
93  unsigned char is_expired;
94 };
```

- 联合类型：

```
99 union ip {
100  unsigned long b32;
101  char b8[4];
102 };
```

- 枚举类型：enum
- 类型定义：typedef

语言速览——C语言——指针类型

- 整型指针：

```
109  int *pb4 = &b4;
```

- 数组指针：

```
112  int *parr1 = arr1;
113  *parr1 = 1; parr1[1] = 2; *(
         parr1+2) = 3;
```

- 字符指针、字符串指针：

```
116  char *s = "123";
```

- 结构指针：

```
119  struct acnt *pacnt = &acnt1;
120  pacnt->id = 2; (*pacnt).name
         [0] = 0;
```

- 函数指针：

```
122  void (*phello)(void);
123  phello = hello; phello();
```

	0xfc000001	0xfc000002	0xfc000003	0xfc000004
pb4	0x03	0x00	0x00	0xff
	0xfc000005	0xfc000006	0xfc000007	0xfc000008
parr1	0x00	0x00	0x00	0xfd
	0xfc000009	0xfc00000a	0xfc00000b	0xfc00000c
stmp	0x31	0x32	0x33	0x00
	0xfc00000d	0xfc00000e	0xfc00000f	0xfc000010
s	0x09	0x00	0x00	0xfc

图: 指针类型

语言速览——C语言——数据运算

- 括号运算：()、
- 增减运算：++、−
- 符号运算：+、-
- 指针运算：[]、.、->、*、&
- 算术运算：*、/、%、+、-
- 关系运算：==、!=、<、<=、>、>=
- 按位运算：~、&、|、^、<<、>>
- 逻辑运算：!、&&、||
- 条件运算：?:
- 赋值运算：=、*=、/=、%=、+=、-=、&=、|=、^=、<<=、>>=
- 逗号运算：,

语言速览——C语言——数据控制

- 顺序
- 选择
- 循环

语言速览——C语言——常用函数

- 输入输出
  - printf
  - scanf
  - puts
  - gets
  - fputs
  - fgets
  - sprintf
  - sscanf
- 文件读写
  - fopen
  - fclose
  - fwrite
  - fread
  - fseek
  - ftell
  - popen
  - pclose
- 字符串值
  - strlen
  - strcpy
  - strcmp
  - strchr
  - strcat
- 内存字节
  - memset
  - memcpy
  - memcmp
  - memchr

3. 套接字基础

- 基本概念
- 工作原理
- 套接字库
- 错误处理
- 文件描述
- 地址结构
- 字节顺序
- 网络地址
- 网络信息

基本概念

套接字（Socket），又称伯克利套接字（Berkeley Socket）

- 伯克利套接字起源于1983年发行的4.2BSD操作系统
- 伯克利套接字终结于1994年发行4.4BSD-Lite和1995年发行的4.4BSD-Lite2
- FreeBSD、NetBSD、OpenBSD

3. 套接字基础

- 基本概念
- 工作原理
- 套接字库
- 错误处理
- 文件描述
- 地址结构
- 字节顺序
- 网络地址
- 网络信息

工作原理

- 三元组：传输协议，网络地址，端口地址
- 五元组：传输协议，来源网络地址，来源端口，目标网络地址，目标端口
- （套接字，套接字地址（传输协议，网络地址，端口地址））

3. 套接字基础

- 基本概念
- 工作原理
- 套接字库
- 错误处理
- 文件描述
- 地址结构
- 字节顺序
- 网络地址
- 网络信息

套接字库——启动库（Mingw）

- 函数声明：

```
1009  WINSOCK_API_LINKAGE int WSAAPI WSAStartup(WORD
          wVersionRequested ,LPWSADATA lpWSAData);
```

- 参数说明：
  - Windows套接字信息结构：

```
13 typedef struct WSAData {
14         WORD                wVersion;
15         WORD                wHighVersion;
16 #ifdef _WIN64
17         unsigned short      iMaxSockets;
18         unsigned short      iMaxUdpDg;
19         char                *lpVendorInfo;
20         char                szDescription[WSADESCRIPTION_LEN+1];
21         char                szSystemStatus[WSASYS_STATUS_LEN+1];
22 #else
23         char                szDescription[WSADESCRIPTION_LEN+1];
24         char                szSystemStatus[WSASYS_STATUS_LEN+1];
25         unsigned short      iMaxSockets;
26         unsigned short      iMaxUdpDg;
27         char                *lpVendorInfo;
28 #endif
29 } WSADATA, *LPWSADATA;
```

套接字库——清除库（Mingw）

- 函数声明：

```
1010  WINSOCK_API_LINKAGE int WSAAPI WSACleanup(void);
```

套接字库——程序示例——启动清除

```c
#if defined(WIN32)
        #include <winsock2.h>
#endif

#include <stdio.h>

int main(void)
{
        int r = 0;
#if defined(WIN32)
        WSADATA wsa_data;
        r = WSAStartup(0x0202, &wsa_data);
        if(0!=r) {
                printf("WSAStartup err: %d\n", r);
        }
#endif

#if defined(WIN32)
        r = WSACleanup();
        if(0!=r) {
                printf("WSACleanup err: %d\n", r);
        }
#endif

        return r;
}
```

套接字库——函数封装——启动清除

```
43 int tx_socklib_startup()
44 {
45 #if defined(WIN32)
46         WSADATA wsa_data;
47         return WSAStartup(0x0202, &wsa_data);
48 #else
49         return 0;
50 #endif
51 }
52
53 int tx_socklib_cleanup()
54 {
55 #if defined(WIN32)
56         return WSACleanup();
57 #else
58         return 0;
59 #endif
60 }
```

套接字库——应用接口

伯克利套接字

- 打开：socket
- 关闭：close
- 绑定地址：bind
- 开始侦听：listen
- 接受连接：accept
- 开始连接：connect
- 接收数据：recv
- 发送数据：send
- 获取选项：getsockopt
- 设置选项：setsockopt

Windows套接字

- 打开：WSASocket
- 关闭：closesocket
- 绑定地址：bind
- 开始侦听：listen
- 接受连接：WSAAccept
- 开始连接：WSAConnect
- 接收数据：WSARecv
- 发送数据：WSASend
- 获取选项：getsockopt
- 设置选项：setsockopt
- 启动动态链接库：WSAStartup
- 清除动态链接库：WSACl

3. 套接字基础

- 基本概念
- 工作原理
- 套接字库
- 错误处理
- 文件描述
- 地址结构
- 字节顺序
- 网络地址
- 网络信息

错误处理——获取错误编号（Mingw）

- 函数声明：

```
1012  WINSOCK_API_LINKAGE int WSAAPI WSAGetLastError(void);
```

错误处理——获取错误编号——函数封装

```
115 #if defined(WIN32)
116         #define tx_sock_get_errnum() WSAGetLastError()
117         #define tx_sock_get_herrnum() WSAGetLastError()
118 #else
119         #define tx_sock_get_errnum() errno
120         #define tx_sock_get_herrnum() h_errno
121 #endif
```

错误处理——设置错误编号（Mingw）

- 函数声明：

```
1011 WINSOCK_API_LINKAGE void WSAAPI WSASetLastError(int iError);
```

错误处理——设置错误编号——函数封装

```
123 #if defined(WIN32)
124         #define tx_sock_set_err_num(_num) WSASetLastError(
                _num)
125 #else
126         #define tx_sock_set_err_num(_num) (errno = (_num))
127 #endif
```

错误处理——获取错误信息（Mingw）

- 函数声明：

```
1345 #define FormatMessage _MINGW_NAME_AW(FormatMessage)
```

```
1342 WINBASEAPI DWORD WINAPI FormatMessageA (DWORD dwFlags, LPCVOID
       lpSource, DWORD dwMessageId, DWORD dwLanguageId, LPSTR
     lpBuffer, DWORD nSize, va_list *Arguments);
```

```
1343 WINBASEAPI DWORD WINAPI FormatMessageW (DWORD dwFlags, LPCVOID
       lpSource, DWORD dwMessageId, DWORD dwLanguageId, LPWSTR
     lpBuffer, DWORD nSize, va_list *Arguments);
```

错误处理——获取错误信息——函数封装

```
129 #if defined(WIN32)
130         char *tx_sock_get_errstr(int num)
131         {
132                 char *pbuf = NULL;
133                 FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM|
                        FORMAT_MESSAGE_ALLOCATE_BUFFER, NULL,
                        num, 0, (LPSTR)&pbuf, 256, NULL);
134                 return pbuf;
135         }
136         #define tx_sock_get_herrstr(_num)
                tx_sock_get_errstr(_num)
137 #else
138         #define tx_sock_get_errstr(_num) strerror(_num)
139         #define tx_sock_get_herrstr(_num) hstrerror(_num)
140 #endif
```

错误处理——释放错误信息——函数封装

```
142 #if defined(WIN32)
143         void tx_sock_free_errstr(char *_str)
144         {
145                 LocalFree(_str);
146         }
147 #else
148         #define tx_sock_free_errstr(_str)
149 #endif
```

错误处理——程序示例

```
1  #include "tx.h"
2
3  int main(void)
4  {
5          int r = 0;
6          r = tx_socklib_startup();
7          if(0!=r) {
8                  char *p = tx_sock_get_errstr(r);
9                  printf("tx_sock_open err: %d, %s\n", r, p)
                       ;
10         }
11
12         r = tx_socklib_cleanup();
13         if(0!=r) {
14                 int r = tx_sock_get_errnum();
15                 char *p = tx_sock_get_errstr(r);
16                 printf("tx_sock_open err: %d, %s\n", r, p)
                       ;
17                 tx_sock_free_errstr(p);
18         }
19
20         return r;
21 }
```

3. 套接字基础

- 基本概念
- 工作原理
- 套接字库
- 错误处理
- 文件描述
- 地址结构
- 字节顺序
- 网络地址
- 网络信息

文件描述——类型定义（Mingw）

```
10 #if 1
11 typedef UINT_PTR        SOCKET;
12 #else
13 typedef INT_PTR         SOCKET;
14 #endif
15
16 #define INVALID_SOCKET  (SOCKET)(~0)
17 #define SOCKET_ERROR    (-1)
```

文件描述——打开

- 函数声明：

```
110 /* Create a new socket of type TYPE in domain DOMAIN, using
111    protocol PROTOCOL.  If PROTOCOL is zero, one is chosen
       automatically.
112    Returns a file descriptor for the new socket, or -1 for
       errors.  */
113 extern int socket (int __domain, int __type, int __protocol)
      __THROW;
```

- 返回：整型

文件描述——打开（Mingw）

- 函数声明：

```
1001  WINSOCK_API_LINKAGE SOCKET WSAAPI socket(int af,int type,int
          protocol);
```

```
 961  #define WSASocket _MINGW_NAME_AW(WSASocket)
```

```
1052  WINSOCK_API_LINKAGE SOCKET WSAAPI WSASocketA(int af,int type,
          int protocol,LPWSAPROTOCOL_INFOA lpProtocolInfo,GROUP g,
          DWORD dwFlags);
```

```
1053  WINSOCK_API_LINKAGE SOCKET WSAAPI WSASocketW(int af,int type,
          int protocol,LPWSAPROTOCOL_INFOW lpProtocolInfo,GROUP g,
          DWORD dwFlags);
```

- 返回：文件描述

文件描述——关闭

- 函数声明:

```
356 extern int close (int __fd);
```

文件描述——关闭（Mingw）

- 函数声明：

```
975  WINSOCK_API_LINKAGE int WSAAPI closesocket(SOCKET s);
```

文件描述——函数封装

```
62 #if defined(WIN32)
63         #define tx_sock_open(domain, type, protocol) WSASocket((
               domain), (type), (protocol), NULL, 0,
               WSA_FLAG_OVERLAPPED)
64 #else
65         #define tx_sock_open socket
66 #endif
67
68 int tx_sock_close(int _sock)
69 {
70 #if defined(WIN32)
71         return closesocket(_sock);
72 #else
73         return close(_sock);
74 #endif
75 }
```

文件描述——程序示例——打开

src/sock_open.c

```
1  #include "tx.h"
2
3  int main(void)
4  {
5          tx_socklib_startup();
6
7          printf("sizeof(tx_sock_t): %lu\n", (unsigned long)
               sizeof(tx_sock_t));
8
9          int n = 1;
10         tx_sock_t sock;
11         do {
12                 sock = tx_sock_open(AF_INET, SOCK_STREAM,
                       IPPROTO_TCP);
13                 printf("%d. sock: %ld\n", n, (long)sock);
14                 n++;
15         } while(sock<0);
16
17         tx_socklib_cleanup();
18
19         return 0;
20 }
```

文件描述——程序示例——关闭

src/sock_close.c

```c
#include "tx.h"

int main(void)
{
        tx_socklib_startup();

        tx_sock_t sock;
        sock = tx_sock_open(AF_INET, SOCK_STREAM, 0);
        printf("sock: %ld\n", (long)sock);

        tx_sock_close(sock);

        tx_socklib_cleanup();

        return 0;
}
```

3. 套接字基础

- 基本概念
- 工作原理
- 套接字库
- 错误处理
- 文件描述
- 地址结构
- 字节顺序
- 网络地址
- 网络信息

地址结构

- IPv4套接字地址结构： struct sockaddr_in
- IPv6套接字地址结构： struct sockaddr_in6
- 通用套接字地址结构： struct sockaddr
- 通用套接字地址存储结构： struct sockaddr_storage

地址结构——IPv4套接字地址结构

```
238 /* Structure describing an Internet socket address.  */
239 struct sockaddr_in
240   {
241     __SOCKADDR_COMMON (sin_);
242     in_port_t sin_port;                 /* Port number.
        */
243     struct in_addr sin_addr;            /* Internet
        address.  */
244
245     /* Pad to size of 'struct sockaddr'.  */
246     unsigned char sin_zero[sizeof (struct sockaddr) -
247                            __SOCKADDR_COMMON_SIZE -
248                            sizeof (in_port_t) -
249                            sizeof (struct in_addr)];
250   };
```

地址结构——IPv4套接字地址结构（Mingw）

```
75 struct sockaddr_in {
76         short   sin_family;
77         u_short sin_port;
78         struct in_addr  sin_addr;
79         char    sin_zero[8];
80 };
```

地址结构——IPv6套接字地址结构

```
253 /* Ditto, for IPv6.  */
254 struct sockaddr_in6
255   {
256     __SOCKADDR_COMMON (sin6_);
257     in_port_t sin6_port;        /* Transport layer port #
          */
258     uint32_t sin6_flowinfo;     /* IPv6 flow information
          */
259     struct in6_addr sin6_addr;  /* IPv6 address */
260     uint32_t sin6_scope_id;     /* IPv6 scope-id */
261   };
```

地址结构——IPv6套接字地址结构（Mingw）

```
41 struct sockaddr_in6 {
42   short sin6_family;
43   u_short sin6_port;
44   u_long sin6_flowinfo;
45   struct in6_addr sin6_addr;
46   __C89_NAMELESS union {
47     u_long sin6_scope_id;
48     SCOPE_ID sin6_scope_struct;
49   };
50 };
51
52 typedef struct sockaddr_in6 SOCKADDR_IN6;
53 typedef struct sockaddr_in6 *PSOCKADDR_IN6;
54 typedef struct sockaddr_in6 *LPSOCKADDR_IN6;
```

地址结构——通用套接字地址结构

```
169 /* Structure describing a generic socket address.  */
170 struct sockaddr
171   {
172     __SOCKADDR_COMMON (sa_);     /* Common data: address
            family and length.  */
173     char sa_data[14];            /* Address data.  */
174   };
```

地址结构——通用套接字地址结构（Mingw）

```
70 struct sockaddr {
71         u_short sa_family;
72         char    sa_data[14];
73 };
```

地址结构——通用套接字地址存储结构

```
177 /* Structure large enough to hold any socket address (with
        the historical
178    exception of AF_UNIX). */
179 #define __ss_aligntype  unsigned long int
180 #define _SS_PADSIZE \
181   (_SS_SIZE - __SOCKADDR_COMMON_SIZE - sizeof (
        __ss_aligntype))
182
183 struct sockaddr_storage
184   {
185     __SOCKADDR_COMMON (ss_);     /* Address family, etc.
          */
186     char __ss_padding[_SS_PADSIZE];
187     __ss_aligntype __ss_align;  /* Force desired alignment
          . */
188   };
```

地址结构——通用套接字地址存储结构（Mingw）

```
254 #define _SS_MAXSIZE 128
255 #define _SS_ALIGNSIZE (8)
256
257 #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof (short))
258 #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof (short) +
      _SS_PAD1SIZE + _SS_ALIGNSIZE))
259
260   struct sockaddr_storage {
261     short ss_family;
262     char __ss_pad1[_SS_PAD1SIZE];
263
264     __MINGW_EXTENSION __int64 __ss_align;
265     char __ss_pad2[_SS_PAD2SIZE];
266   };
```

3. 套接字基础

- 基本概念
- 工作原理
- 套接字库
- 错误处理
- 文件描述
- 地址结构
- 字节顺序
- 网络地址
- 网络信息

字节顺序

- 大端（Big Endian）
- 小端（Little Endian）

字节顺序

- 主机字节顺序：大端或小端
  - 大端：mips、alpha、m68k
  - 小端：x86
- 网络字节顺序：大端

字节顺序——格式转换——宏

```
369 /* Functions to convert between host and network byte
       order.
370
371    Please note that these functions normally take '
          unsigned long int' or
372    'unsigned short int' values as arguments and also
          return them.  But
373    this was a short-sighted decision since on different
          systems the types
374    may have different representations but the values are
          always the same.  */
375
376 extern uint32_t ntohl (uint32_t __netlong) _THROW
       __attribute__ ((__const__));
377 extern uint16_t ntohs (uint16_t __netshort)
378      _THROW __attribute__ ((__const__));
379 extern uint32_t htonl (uint32_t __hostlong)
380      _THROW __attribute__ ((__const__));
381 extern uint16_t htons (uint16_t __hostshort)
382      _THROW __attribute__ ((__const__));
```

字节顺序——格式转换

```
393 # if __BYTE_ORDER == __BIG_ENDIAN
394 /* The host byte order is the same as network byte order,
395    so these functions are all just identity.  */
396 # define ntohl(x)         (x)
397 # define ntohs(x)         (x)
398 # define htonl(x)         (x)
399 # define htons(x)         (x)
400 # else
401 #  if __BYTE_ORDER == __LITTLE_ENDIAN
402 #   define ntohl(x)     __bswap_32 (x)
403 #   define ntohs(x)     __bswap_16 (x)
404 #   define htonl(x)     __bswap_32 (x)
405 #   define htons(x)     __bswap_16 (x)
406 #  endif
407 # endif
```

字节顺序——格式转换——Mingw

```
982  WINSOCK_API_LINKAGE u_long WSAAPI htonl(
         u_long hostlong);
```

```
983  WINSOCK_API_LINKAGE u_short WSAAPI htons(
         u_short hostshort);
```

```
989  WINSOCK_API_LINKAGE u_long WSAAPI ntohl(
         u_long netlong);
```

```
990  WINSOCK_API_LINKAGE u_short WSAAPI ntohs(
         u_short netshort);
```

字节顺序——函数封装

src/tx.h

```
220 void tx_mem_dump(void *_buf, size_t _buf_size)
221 {
222         for(int i=0;i<(_buf_size); i++) {
223                 printf("%#x, ", ((unsigned char*)(_buf))[i
                        ]);
224         }
225         printf("\n");
226 }
```

字节顺序——程序示例

src/byte_order.c

```c
#include "tx.h"

void byte_order()
{
        union {
                unsigned short n;
                unsigned char c[2];
        } y;
        y.n = 0x1234;
        printf("%#x\n", y.n);
        tx_mem_dump(&y, sizeof(y));

        if (y.c[0] == 0x34) {
                printf("little_endian\n");
        } else {
                printf("big_endian\n");
        }
}

int main(void)
{
        byte_order();
        return 0;
}
```

3. 套接字基础

- 基本概念
- 工作原理
- 套接字库
- 错误处理
- 文件描述
- 地址结构
- 字节顺序
- 网络地址
- 网络信息

网络地址

- IPv4网络地址：struct in_addr
- IPv6网络字地址：struct in6_addr

网络地址——IPv4网络地址

```
29 /* Internet address.  */
30 typedef uint32_t in_addr_t;
31 struct in_addr
32   {
33     in_addr_t s_addr;
34   };
```

网络地址——IPv4网络地址——Mingw

```
17 typedef struct in_addr {
18   union {
19     struct { u_char  s_b1, s_b2, s_b3, s_b4; } S_un_b;
20     struct { u_short s_w1, s_w2; } S_un_w;
21     u_long S_addr;
22   } S_un;
23 } IN_ADDR, *PIN_ADDR, *LPIN_ADDR;
24
25 #define s_addr  S_un.S_addr
26 #define s_host  S_un.S_un_b.s_b2
27 #define s_net   S_un.S_un_b.s_b1
28 #define s_imp   S_un.S_un_w.s_w2
29 #define s_impno S_un.S_un_b.s_b4
30 #define s_lh    S_un.S_un_b.s_b3
```

网络地址——IPv6网络地址

```
210 /* IPv6 address */
211 struct in6_addr
212   {
213     union
214       {
215         uint8_t __u6_addr8[16];
216 #ifdef __USE_MISC
217         uint16_t __u6_addr16[8];
218         uint32_t __u6_addr32[4];
219 #endif
220       } __in6_u;
221 #define s6_addr                 __in6_u.__u6_addr8
222 #ifdef __USE_MISC
223 # define s6_addr16              __in6_u.__u6_addr16
224 # define s6_addr32              __in6_u.__u6_addr32
225 #endif
226   };
```

网络地址——IPv6网络地址——Mingw

```
17 typedef struct in6_addr {
18   union {
19     u_char Byte[16];
20     u_short Word[8];
21 #ifdef __INSIDE_CYGWIN__
22     uint32_t __s6_addr32[4];
23 #endif
24   } u;
25 } IN6_ADDR, *PIN6_ADDR, *LPIN6_ADDR;
26
27 #define in_addr6          in6_addr
28
29 #define _S6_un            u
30 #define _S6_u8            Byte
31 #define s6_addr           _S6_un._S6_u8
32
33 #define s6_bytes          u.Byte
34 #define s6_words          u.Word
35
36 #ifdef __INSIDE_CYGWIN__
37 #define s6_addr16         u.Word
38 #define s6_addr32         u.__s6_addr32
39 #endif
```

网络地址——IPv4网络地址——格式转换

```
32 /* Convert Internet host address from numbers-and-dots
       notation in CP
33    into binary data in network byte order.  */
34 extern in_addr_t inet_addr (const char *__cp) __THROW;
```

```
51 /* Convert Internet number in IN to ASCII representation.
       The return value
52    is a pointer to an internal array containing the string
          .  */
53 extern char *inet_ntoa (struct in_addr __in) __THROW;
```

网络地址——IPv4网络地址——格式转换——程序示例

字符转数值： src/inet_addr.c

```
1  #include "tx.h"
2
3  int main(int argc, char *argv[])
4  {
5          char *ip = "127.0.0.1"; //255.255.255.255?
6
7          if(argc>1) {
8                  ip = argv[1];
9          }
10
11         int r = tx_socklib_startup();
12         if(0!=r) {
13                 printf("tx_socklib_startup err: %d, %s\n",
                       r, tx_sock_get_errstr(r));
14                 goto quit;
15         }
16
17         struct in_addr addr;
18         addr.s_addr = inet_addr(ip);
19         tx_mem_dump(&addr, sizeof(addr));
20 quit:
21         tx_socklib_cleanup();
22         return r;
23 }
```

网络地址——IPv4网络地址——格式转换——程序示例

数值转字符： src/inet_ntoa.c

```
1  #include "tx.h"
2
3  int main(int argc, char *argv[])
4  {
5          int r = tx_socklib_startup();
6          if(0!=r) {
7                  printf("tx_socklib_startup err: %d, %s\n",
                       r, tx_sock_get_errstr(r));
8                  goto quit;
9          }
10
11         struct in_addr addr;
12         addr.s_addr = ntohl(INADDR_LOOPBACK);
13         tx_mem_dump(&addr, sizeof(addr));
14
15         char *p = inet_ntoa(addr);
16         printf("%s\n", p);
17 quit:
18         tx_socklib_cleanup();
19         return r;
20 }
```

网络地址——格式转换

```
55 /* Convert from presentation format of an Internet number
       in buffer
56    starting at CP to the binary network format and store
         result for
57    interface type AF in buffer starting at BUF.  */
58 extern int inet_pton (int __af, const char *__restrict
       __cp,
59                       void *__restrict __buf) __THROW;
```

```
61 /* Convert a Internet address in binary network format for
        interface
62    type AF in buffer starting at CP to presentation form
         and place
63    result in buffer of length LEN astarting at BUF.  */
64 extern const char *inet_ntop (int __af, const void *
       __restrict __cp,
65                                char *__restrict __buf,
                                     socklen_t __len)
66      __THROW;
```

3. 套接字基础

- 基本概念
- 工作原理
- 套接字库
- 错误处理
- 文件描述
- 地址结构
- 字节顺序
- 网络地址
- 网络信息

网络信息

- 主机
- 网络
- 服务
- 协议

网络信息——主机名称

- 获取主机名称：gethostname
- 设置主机名称：sethostname

网络信息——主机域名

- 获取域名名称：getdomainname
- 设置域名名称：setdomainname

网络信息——IPv4——主机信息数据类型

```
99  /* Description of data base entry for a single host.  */
100 struct hostent
101 {
102   char *h_name;                 /* Official name of host.
        */
103   char **h_aliases;             /* Alias list.  */
104   int h_addrtype;               /* Host address type.  */
105   int h_length;                 /* Length of address.  */
106   char **h_addr_list;           /* List of addresses from
        name server.  */
107 #ifdef __USE_MISC
108 # define         h_addr  h_addr_list[0] /* Address, for
      backward compatibility.*/
109 #endif
110 };
```

网络信息——IPv4——根据名称获取主机信息

```
140 /* Return entry from host data base for host with NAME.
141
142     This function is a possible cancellation point and
        therefore not
143     marked with __THROW.  */
144 extern struct hostent *gethostbyname (const char *__name);
```

网络信息——IPv4——根据名称获取主机信息

src/gethostbyname.c

```
19      struct hostent *phostent = gethostbyname(name);
20      if(NULL==phostent) {
21              err_num = tx_sock_get_herrnum();
22              printf("gethostbyname err: %d, %s\n", err_num,
                    tx_sock_get_herrstr(err_num));
23              goto quit;
24      }
25      printf("phostent->h_name: %s\n", phostent->h_name);
26      for(int i=0; phostent->h_aliases[i]!=NULL; i++) {
27              printf("phostent->h_aliases[%d]: %s\n", i, phostent->h_aliases[i
                    ]);
28      }
29      switch(phostent->h_addrtype) {
30              case AF_INET:
31                      printf("phostent->h_addrtype: %d(AF_INET)\n", phostent->
                            h_addrtype);
32                      break;
33              default:
34                      printf("phostent->h_addrtype: %d\n", phostent->
                            h_addrtype);
35      }
36      printf("phostent->h_length: %d\n", phostent->h_length);
37      for(int i=0; phostent->h_addr_list[i]!=NULL; i++) {
38              printf("phostent->h_addr[%d]: %s\n", i, inet_ntoa(*((struct
                    in_addr*)phostent->h_addr_list[i])));
39      }
```

网络信息——IPv4——根据地址获取主机信息

```
132 /* Return entry from host data base which address match
        ADDR with
133     length LEN and type TYPE.
134
135     This function is a possible cancellation point and
            therefore not
136     marked with __THROW.  */
137 extern struct hostent *gethostbyaddr (const void *__addr,
        __socklen_t __len,
138                                       int __type);
```

网络信息——IPv4——根据地址获取主机信息

src/gethostbyaddr.c

```
19          struct in_addr addr;
20          addr.s_addr = inet_addr(ip);
21          struct hostent *phostent = gethostbyaddr((const
               char *)&addr, sizeof(addr), AF_INET);
22          if(NULL==phostent) {
23                  err_num = tx_sock_get_herrnum();
24                  printf("gethostbyaddr err: %d, %s\n",
                       err_num, tx_sock_get_herrstr(err_num));
25                  goto quit;
26          }
27          printf("phostent->h_name: %s\n", phostent->h_name)
               ;
28          for(int i=0; phostent->h_aliases[i]!=NULL; i++) {
29                  printf("phostent->h_aliases[%d]: %s\n", i,
                        phostent->h_aliases[i]);
30          }
31          switch(phostent->h_addrtype) {
32                  case AF_INET:
33                          printf("phostent->h_addrtype: %d(
                               AF_INET)\n", phostent->
                               h_addrtype);
34                          break;
35                  default:
36                          printf("phostent->h_addrtype: %d\n
                               ", phostent->h_addrtype);
37          }
38          printf("phostent->h_length: %d\n", phostent->
               h_length);
39          for(int i=0; phostent->h_addr_list[i]!=NULL; i++)
               {
40                  printf("phostent->h_addr[%d]: %s\n", i,
                       inet_ntoa(*((struct in_addr*)phostent->
                       h_addr_list[i])));
41          }
```

网络信息——数据类型

```
657 /* Translate name of a service location and/or a service
       name to set of
658    socket addresses.
659
660    This function is a possible cancellation point and
          therefore not
661    marked with __THROW.  */
662 extern int getaddrinfo (const char *__restrict __name,
663                         const char *__restrict __service,
664                         const struct addrinfo *__restrict
                               __req,
665                         struct addrinfo **__restrict __pai
                               );
```

网络信息——根据名称获取主机信息

```
657 /* Translate name of a service location and/or a service
        name to set of
658    socket addresses.
659
660    This function is a possible cancellation point and
          therefore not
661    marked with __THROW.  */
662 extern int getaddrinfo (const char *__restrict __name,
663                          const char *__restrict __service,
664                          const struct addrinfo *__restrict
                               __req,
665                          struct addrinfo **__restrict __pai
                               );
666
667 /* Free 'addrinfo' structure AI including associated
        storage.  */
668 extern void freeaddrinfo (struct addrinfo *__ai) __THROW;
669
670 /* Convert error return from getaddrinfo() to a string.
        */
671 extern const char *gai_strerror (int __ecode) __THROW;
```

网络信息——根据名称获取主机信息

```
673 /* Translate a socket address to a location and service
       name.
674
675    This function is a possible cancellation point and
          therefore not
676    marked with __THROW.  */
677 extern int getnameinfo (const struct sockaddr *__restrict
       __sa,
678                          socklen_t __salen, char *
                                __restrict __host,
679                          socklen_t __hostlen, char *
                                __restrict __serv,
680                          socklen_t __servlen, int __flags);
```

4. 数据流套接字

- 基本概念
- 一般流程
- 绑定地址
- 侦听连接
- 接受连接
- 发起连接
- 收发数据
- 设置选项
- 注意事项

4. 数据流套接字

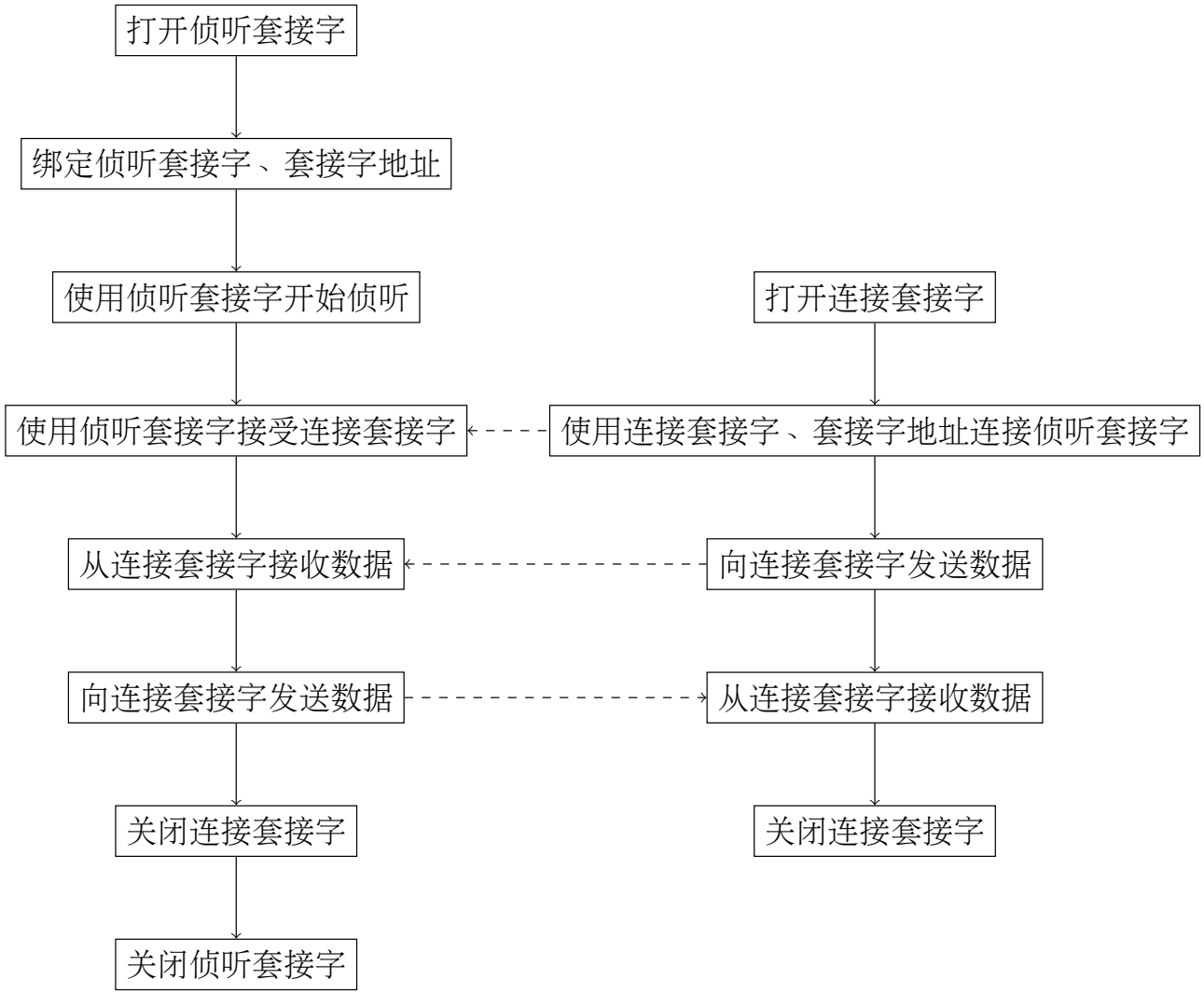
- 基本概念
- 一般流程
- 绑定地址
- 侦听连接
- 接受连接
- 发起连接
- 收发数据
- 设置选项
- 注意事项

逻辑视图

服务器：套接字（用于侦听），（通信协议，网络地址，端口地址）

客户端：套接字（用于连接），（通信协议，网络地址，端口地址）

流程视图



4. 数据流套接字

- 基本概念
- 一般流程
- 绑定地址
- 侦听连接
- 接受连接
- 发起连接
- 收发数据
- 设置选项
- 注意事项

绑定地址

- 函数声明:

```
122 /* Give the socket FD the local address
       ADDR (which is LEN bytes long). */
123 extern int bind (int __fd,
       __CONST_SOCKADDR_ARG __addr, socklen_t
       __len)
124      __THROW;
```

4. 数据流套接字

- 基本概念
- 一般流程
- 绑定地址
- 侦听连接
- 接受连接
- 发起连接
- 收发数据
- 设置选项
- 注意事项

侦听连接

- 函数声明:

```
230 /* Prepare to accept connections on socket FD.
231    N connection requests will be queued before further
       requests are refused.
232    Returns 0 on success, -1 for errors.  */
233 extern int listen (int __fd, int __n) __THROW;
```

4. 数据流套接字

- 基本概念
- 一般流程
- 绑定地址
- 侦听连接
- 接受连接
- 发起连接
- 收发数据
- 设置选项
- 注意事项

接受连接

- 函数声明:

```
235 /* Await a connection on socket FD.
236    When a connection arrives, open a new socket to
       communicate with it,
237    set *ADDR (which is *ADDR_LEN bytes long) to the
       address of the connecting
238    peer and *ADDR_LEN to the address's actual length,
       and return the
239    new socket's descriptor, or -1 for errors.
240
241    This function is a cancellation point and therefore
       not marked with
242    __THROW.  */
243 extern int accept (int __fd, __SOCKADDR_ARG __addr,
244                    socklen_t *__restrict __addr_len);
```

4. 数据流套接字

- 基本概念
- 一般流程
- 绑定地址
- 侦听连接
- 接受连接
- 发起连接
- 收发数据
- 设置选项
- 注意事项

发起连接

- 函数声明:

```
130 /* Open a connection on socket FD to peer at ADDR (
        which LEN bytes long).
131    For connectionless socket types, just set the
           default address to send to
132    and the only address from which to accept
           transmissions.
133    Return 0 on success, -1 for errors.
134
135    This function is a cancellation point and therefore
           not marked with
136    __THROW.  */
137 extern int connect (int __fd, __CONST_SOCKADDR_ARG
        __addr, socklen_t __len);
```

4. 数据流套接字

- 基本概念
- 一般流程
- 绑定地址
- 侦听连接
- 接受连接
- 发起连接
- 收发数据
- 设置选项
- 注意事项

接收数据

- 函数声明:

```
151 /* Read N bytes into BUF from socket FD.
152    Returns the number read or -1 for errors.
153
154    This function is a cancellation point and therefore
          not marked with
155    __THROW.  */
156 extern ssize_t recv (int __fd, void *__buf, size_t __n,
       int __flags);
```

发送数据

- 函数声明:

```
145 /* Send N bytes of BUF to socket FD.  Returns the
       number sent or -1.
146
147    This function is a cancellation point and therefore
         not marked with
148    __THROW.  */
149 extern ssize_t send (int __fd, const void *__buf,
       size_t __n, int __flags);
```

4. 数据流套接字

- 基本概念
- 一般流程
- 绑定地址
- 侦听连接
- 接受连接
- 发起连接
- 收发数据
- 设置选项
- 注意事项

设置选项

- 函数声明:

```
216 /* Put the current value for socket FD's
       option OPTNAME at protocol level LEVEL
217    into OPTVAL (which is *OPTLEN bytes
          long), and set *OPTLEN to the value'
          s
218    actual length.  Returns 0 on success,
          -1 for errors.  */
219 extern int getsockopt (int __fd, int
       __level, int __optname,
```

4. 数据流套接字

- 基本概念
- 一般流程
- 绑定地址
- 侦听连接
- 接受连接
- 发起连接
- 收发数据
- 设置选项
- 注意事项

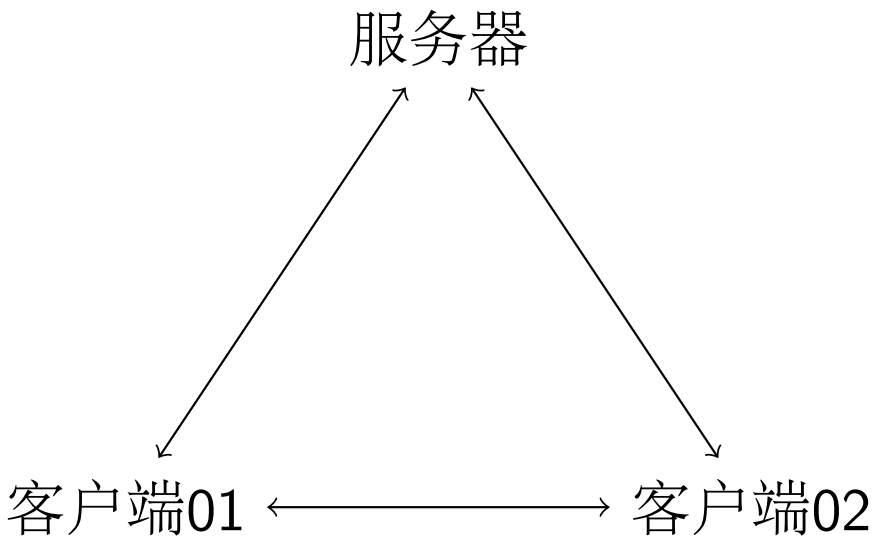
5. 数据报套接字

- 基本概念
- 一般流程
- 收发数据
- 广播机制
- 多播机制
- 注意事项

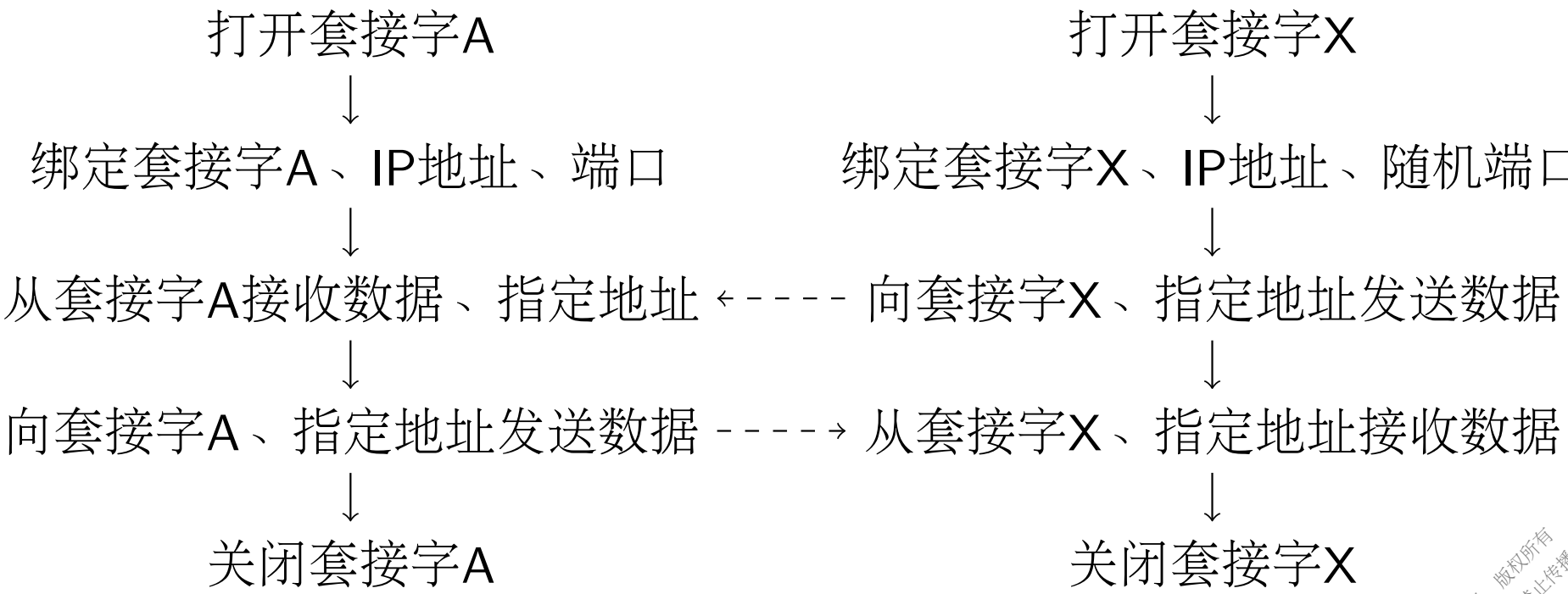
5. 数据报套接字

- 基本概念
- 一般流程
- 收发数据
- 广播机制
- 多播机制
- 注意事项

逻辑视图



流程视图



5. 数据报套接字

- 基本概念
- 一般流程
- 收发数据
- 广播机制
- 多播机制
- 注意事项

接收数据

- 函数声明:

```
167 /* Read N bytes into BUF through socket FD
        .
168     If ADDR is not NULL, fill in *ADDR_LEN
            bytes of it with tha address of
169     the sender, and store the actual size
            of the address in *ADDR_LEN.
170     Returns the number of bytes read or -1
            for errors.
171
```

发送数据

- 函数声明:

```
158 /* Send N bytes of BUF on socket FD to
       peer at address ADDR (which is
159    ADDR_LEN bytes long). Returns the
          number sent, or -1 for errors.
160
161    This function is a cancellation point
          and therefore not marked with
162    _THROW. */
163 extern ssize_t sendto (int __fd, const
```

编程实例——基本UDP响应服务器

- 需求

  与多个UDP响应客户端通信

- 源码
- 编译
- 运行

  使用netstat -ano查看

  使用telnet测试

编程实例——基本UDP响应客户端

- 需求

  与UDP响应服务器或其他UDP响应客户端通信

- 源码
- 编译
- 运行

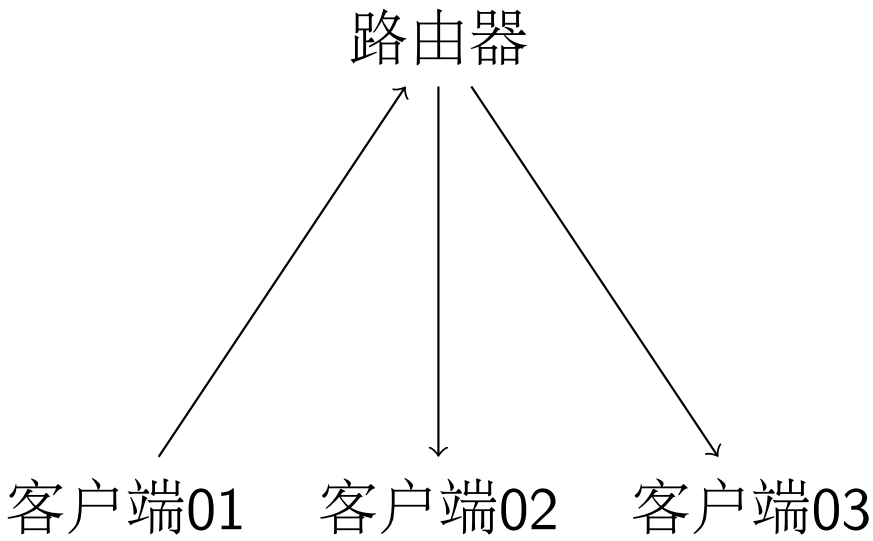
  使用netstat -ano查看

  使用telnet测试

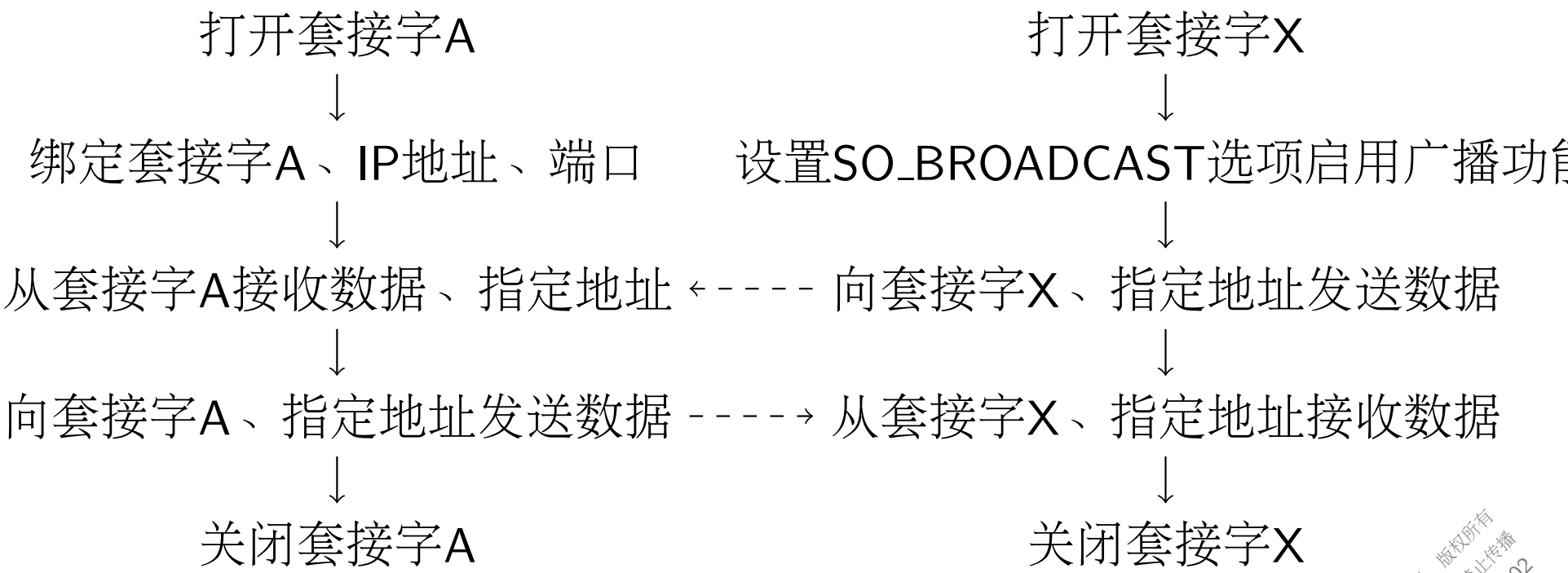
5. 数据报套接字

- 基本概念
- 一般流程
- 收发数据
- 广播机制
- 多播机制
- 注意事项

逻辑视图



流程视图



编程实例——广播服务器 I

- 需求
- 源码
- 编译
- 运行

  使用netstat -ano查看

  使用telnet测试

编程实例——广播客户端 I

- 需求
- 源码
- 编译
- 运行

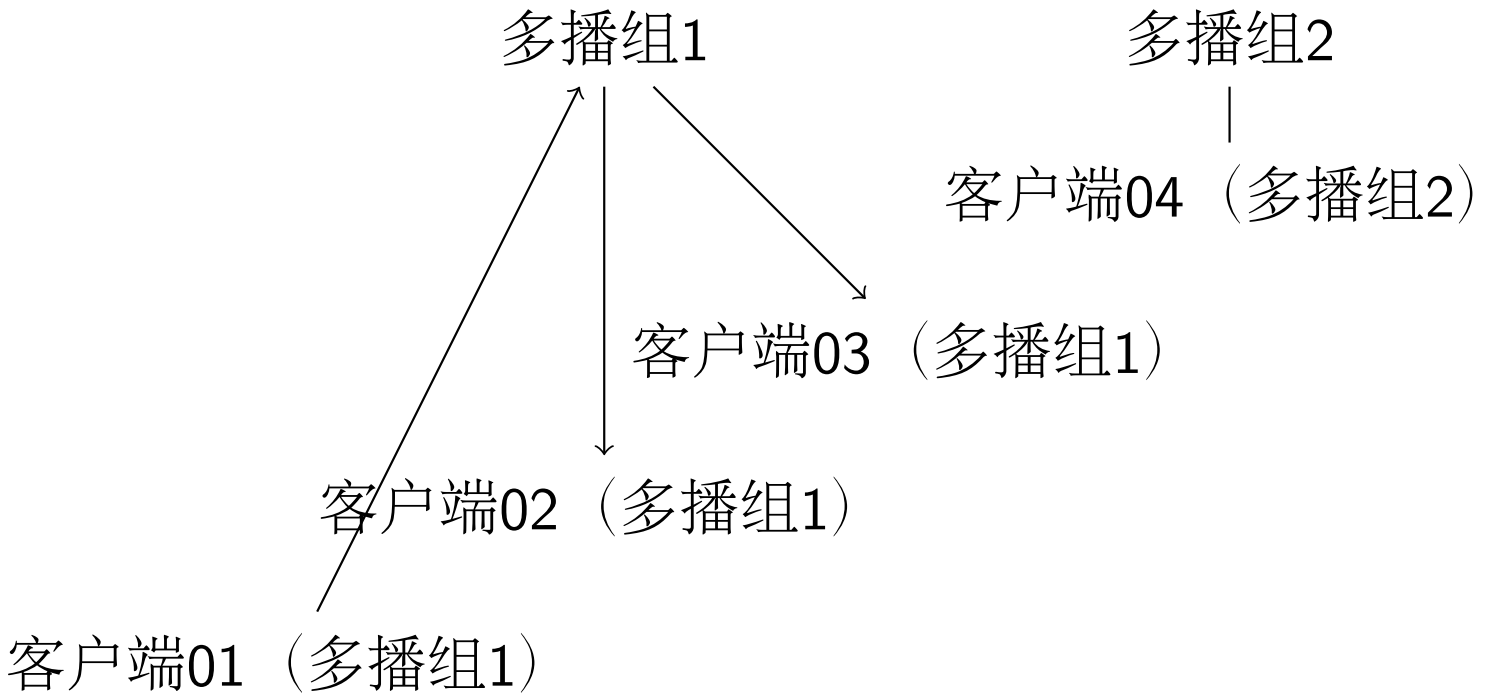
  使用netstat -ano查看

  使用telnet测试

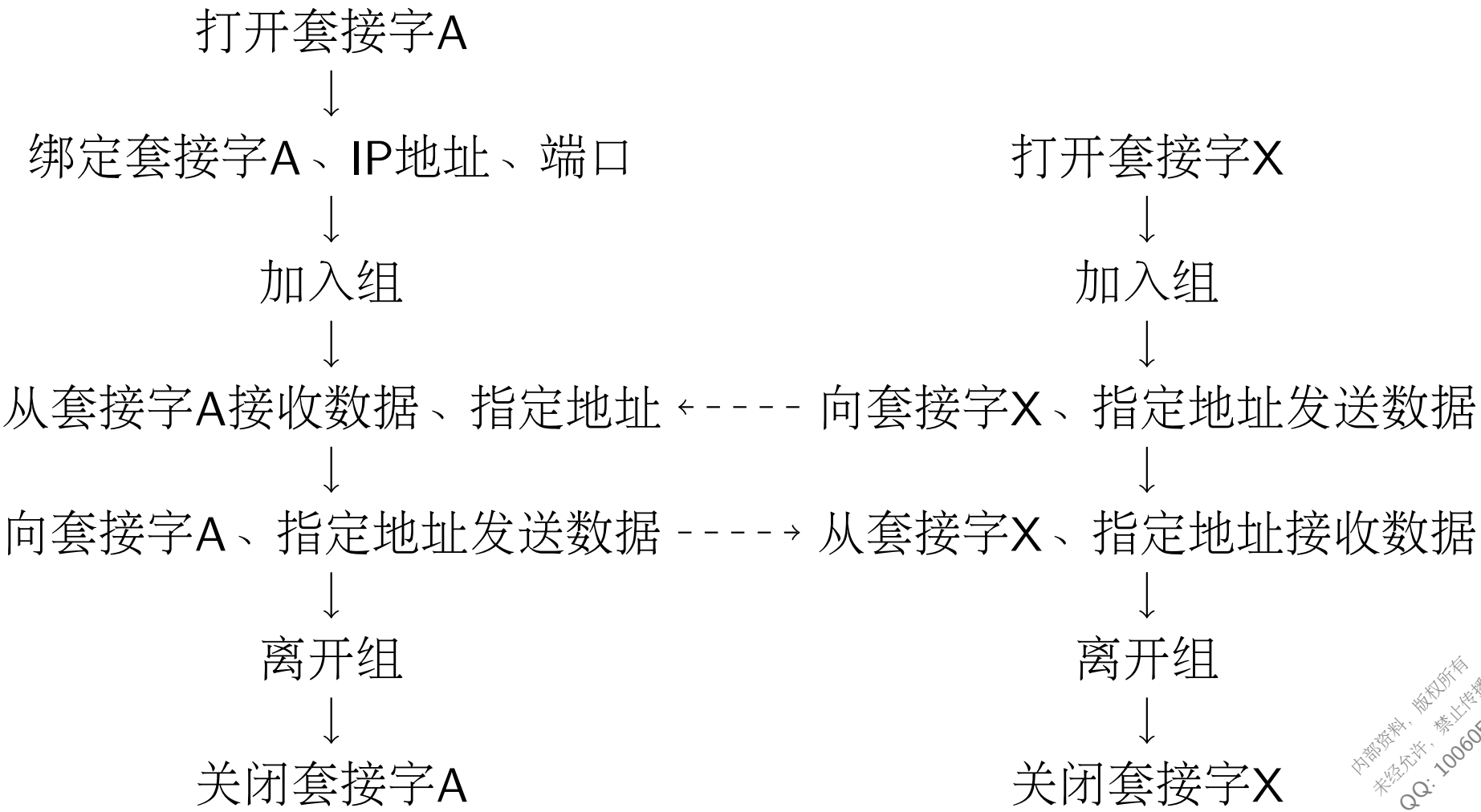
5. 数据报套接字

- 基本概念
- 一般流程
- 收发数据
- 广播机制
- 多播机制
- 注意事项

逻辑视图



流程视图



编程实例——多播服务器 I

- 需求
- 源码
- 编译
- 运行

  使用netstat -ano查看

  使用telnet测试

编程实例——多播客户端 I

- 需求
- 源码
- 编译
- 运行

  使用netstat -ano查看

  使用telnet测试

5. 数据报套接字

- 基本概念
- 一般流程
- 收发数据
- 广播机制
- 多播机制
- 注意事项

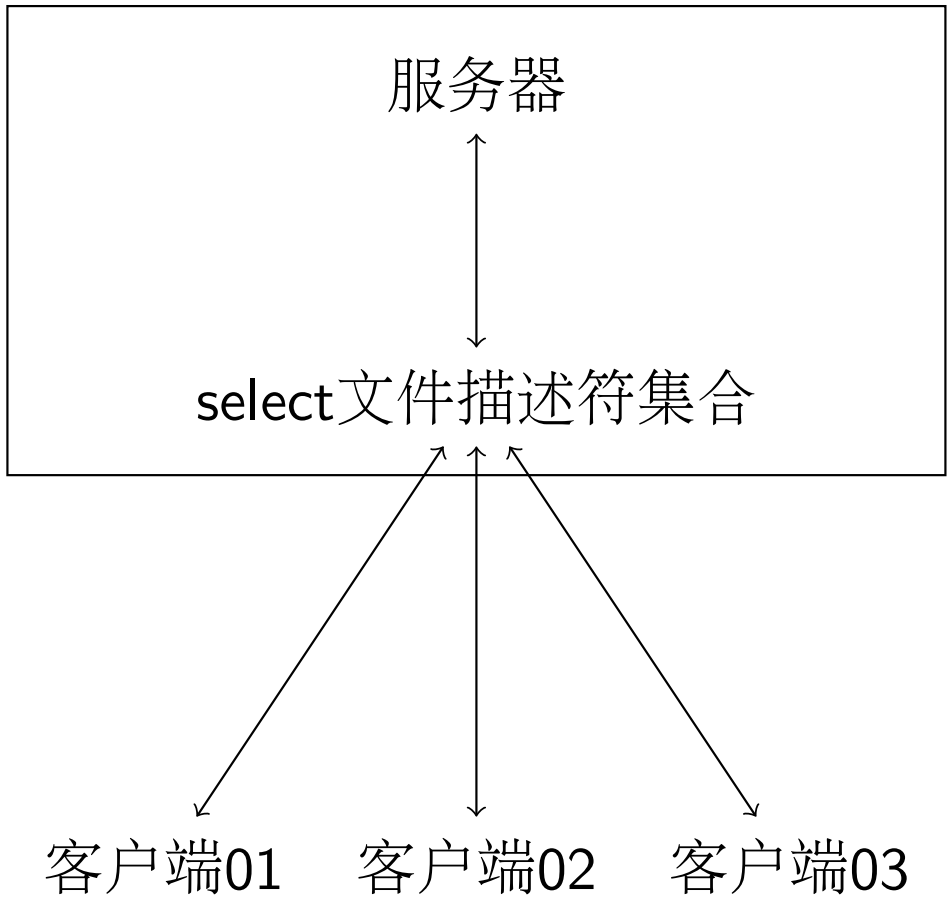
6. 输入输出模型

- 阻塞模型
- 迭代模型
- 并发模型
- 异步模型

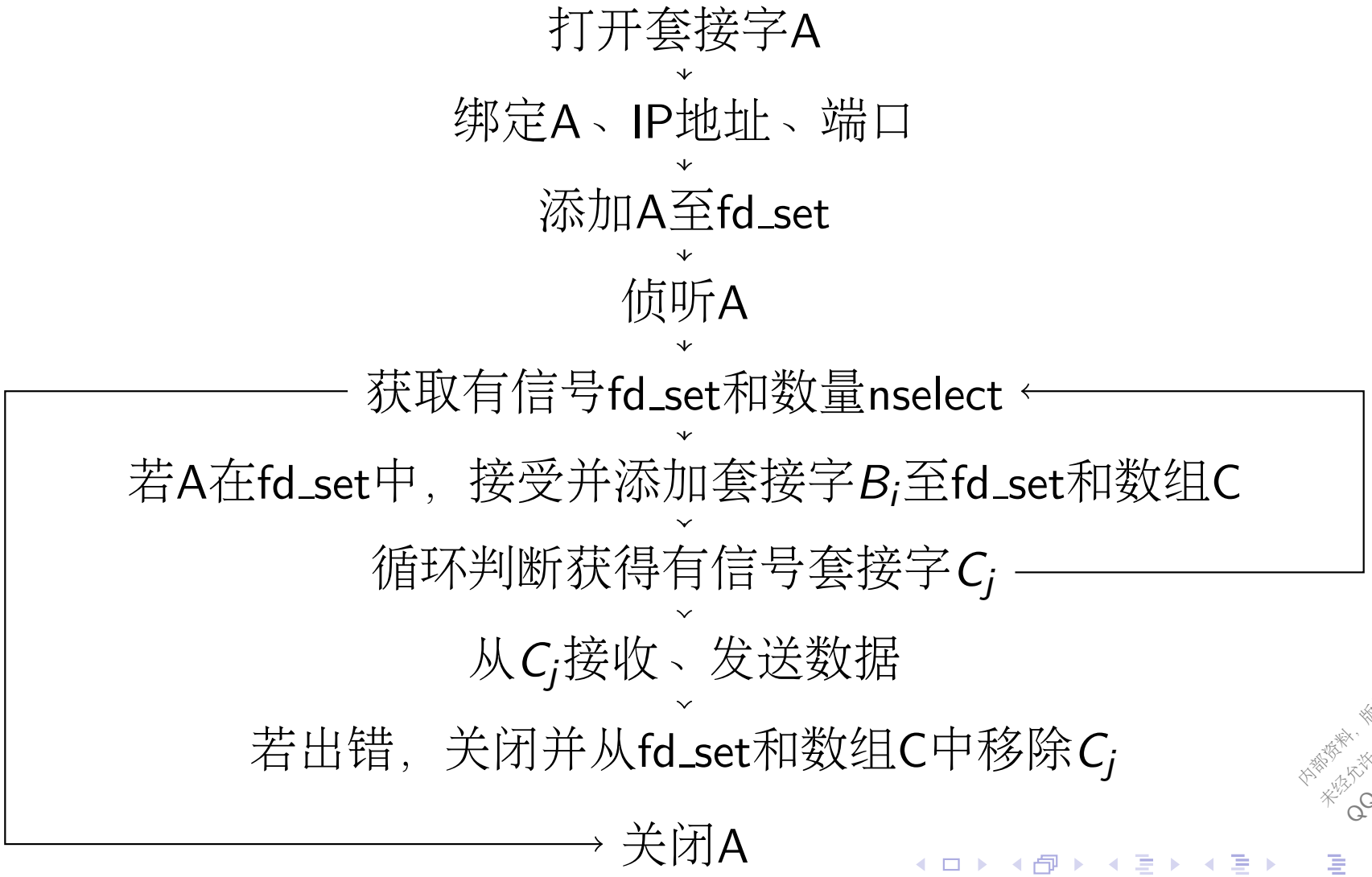
6. 输入输出模型

- 阻塞模型
- 迭代模型
- 并发模型
- 异步模型

逻辑视图



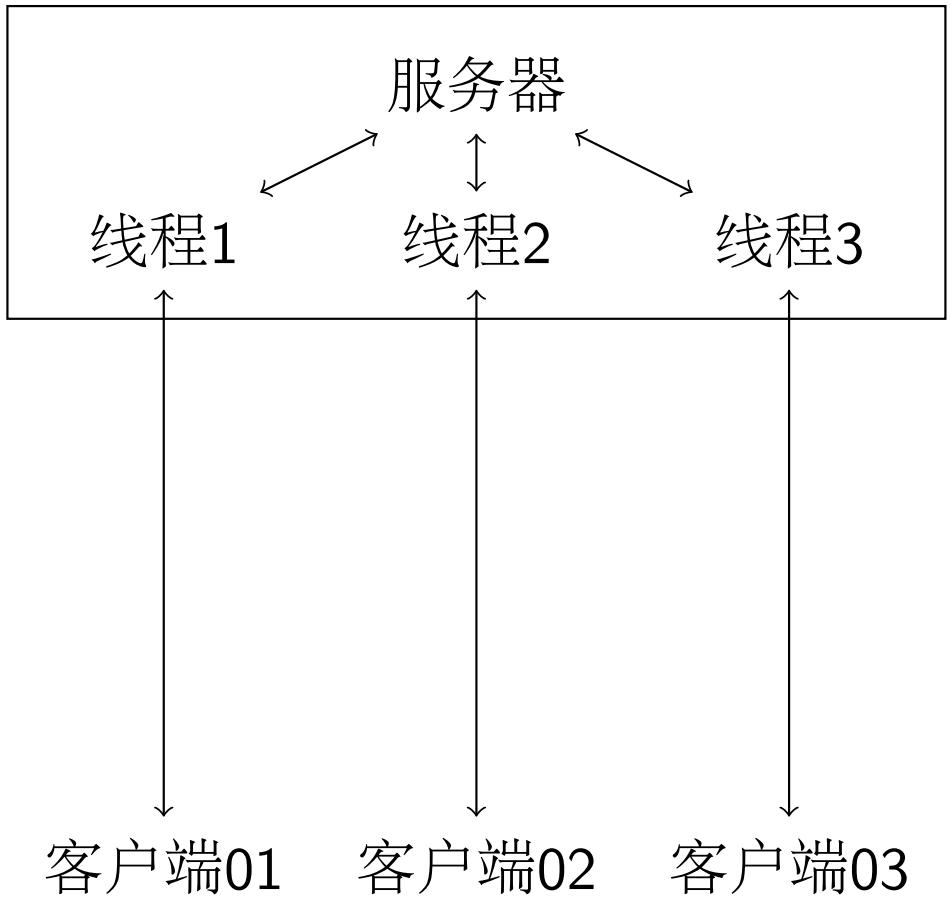
流程视图



6. 输入输出模型

- 阻塞模型
- 迭代模型
- 并发模型
- 异步模型

多线程模型



6. 输入输出模型

- 阻塞模型
- 迭代模型
- 并发模型
- 异步模型

异步模型

7. 原始套接字

- 基本介绍
- 创建关闭
- 收发数据
- 注意事项

7. 原始套接字

- 基本介绍
- 创建关闭
- 收发数据
- 注意事项

7. 原始套接字

- 基本介绍
- 创建关闭
- 收发数据
- 注意事项

7. 原始套接字

- 基本介绍
- 创建关闭
- 收发数据
- 注意事项

8. 链路套接字

- 基本介绍
- 创建关闭
- 收发数据
- 注意事项
- 其他接口

8. 链路套接字

- 基本介绍
- 创建关闭
- 收发数据
- 注意事项
- 其他接口

8. 链路套接字

- 基本介绍
- 创建关闭
- 收发数据
- 注意事项
- 其他接口

8. 链路套接字

- 基本介绍
- 创建关闭
- 收发数据
- 注意事项
- 其他接口

8. 链路套接字

- 基本介绍
- 创建关闭
- 收发数据
- 注意事项
- 其他接口

编程实例——多播客户端 I

- 需求
- 源码
- 编译
- 运行

  使用netstat -ano查看

  使用telnet测试