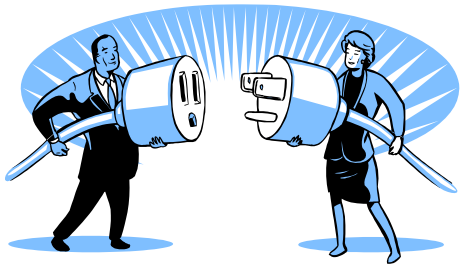
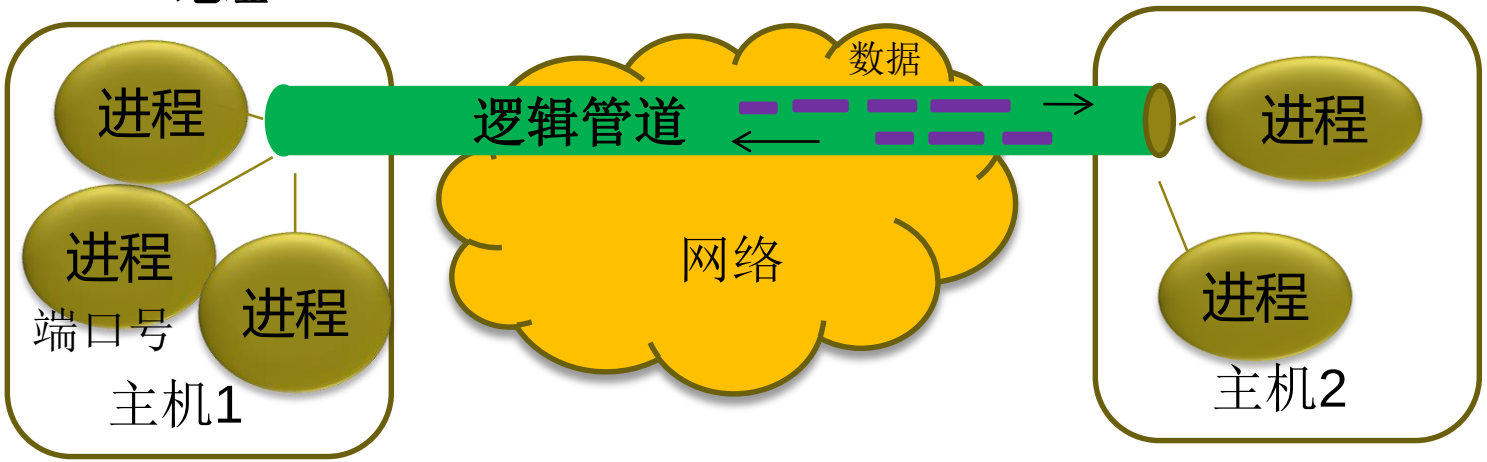
套接字程序设计
(Socket Programming)

中山大学 计算机科学系　张永民
2019年3月6日

什么是套接字(Socket)?

Socket（插座）





(1) 通过**IP地址**或域名找到主机

www.sysu.edu.cn->202.116.64.8

IP地址加端口号类似电话号码加分机号

(2) 通过**端口号**找到主机上的进程

UDP协议(不可靠)

TCP协议(面向连接、可靠)

端口号

端口号 | 16比特

分类
- 知名端口号(0~1023)
  - 知名应用层协议
    - 80-HTTP协议
    - 21-ftp 协议
    - 23-telnet协议
    - 25-SMTP协议
    - 110-POP3协议
    - .....
- 注册端口号(1024~49151)
  - 公司业务
- 动态端口号(49152~65535)
  - 临时连接

用户可以随意使用。

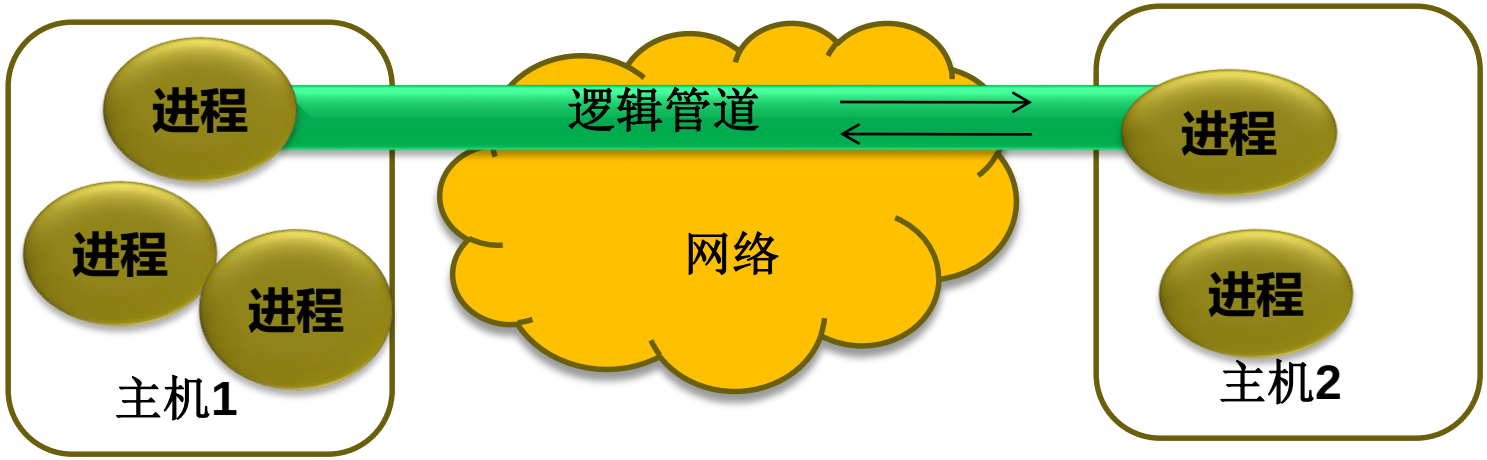
控制台命令：

ipconfig 查看本机所有的IP地址

netstat 查看所有被占用的端口号。

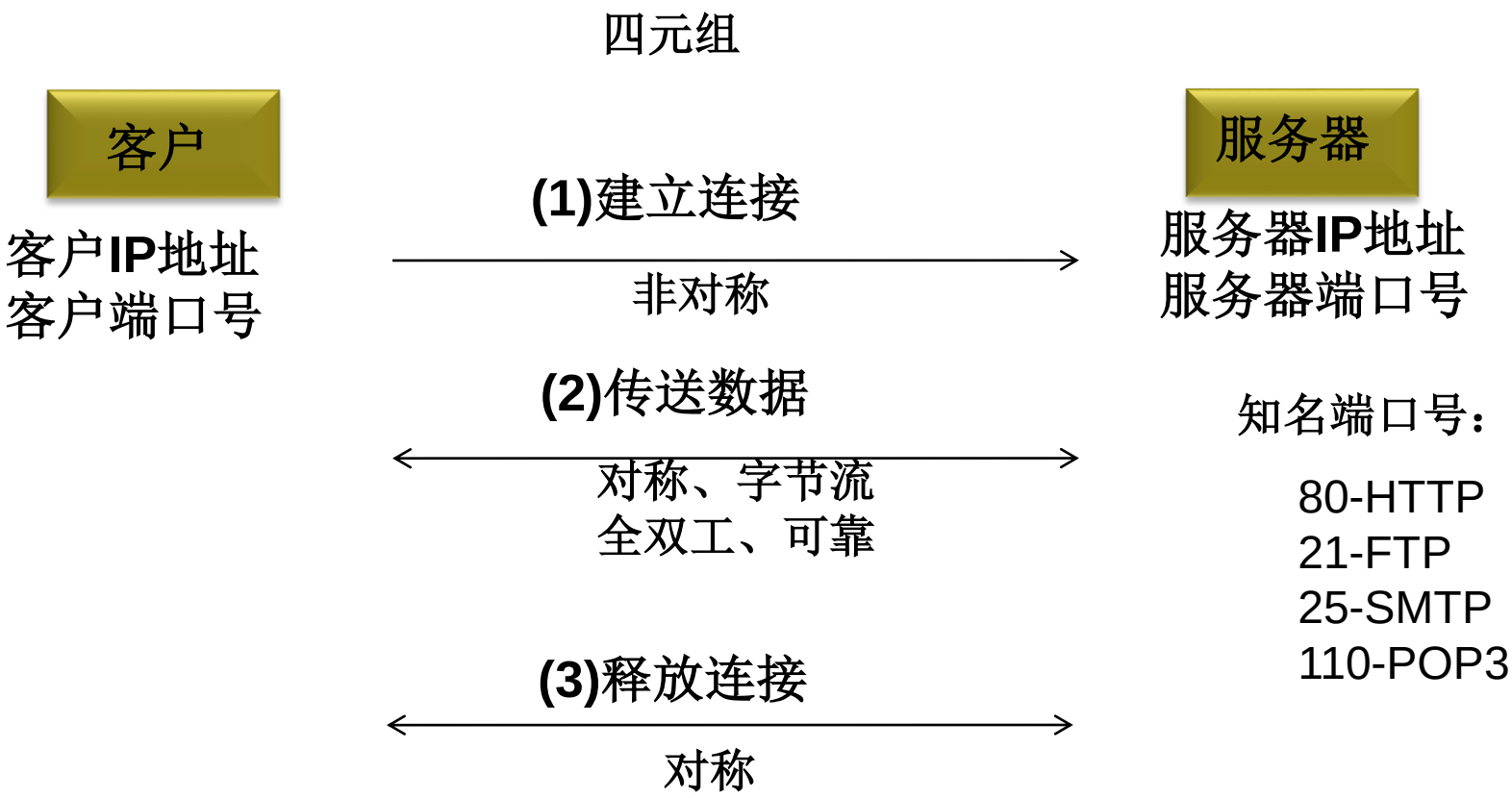
TCP协议

- TCP协议是一种传输层协议。它可以在两个进程之间建立一条逻辑管道，用来为它们提供全双工的可靠的数据传送服务。
- TCP协议是通过重传机制在不可靠的网络中实现可靠传输的，即实现无比特错、不丢失、无错序和无重复的数据传输。



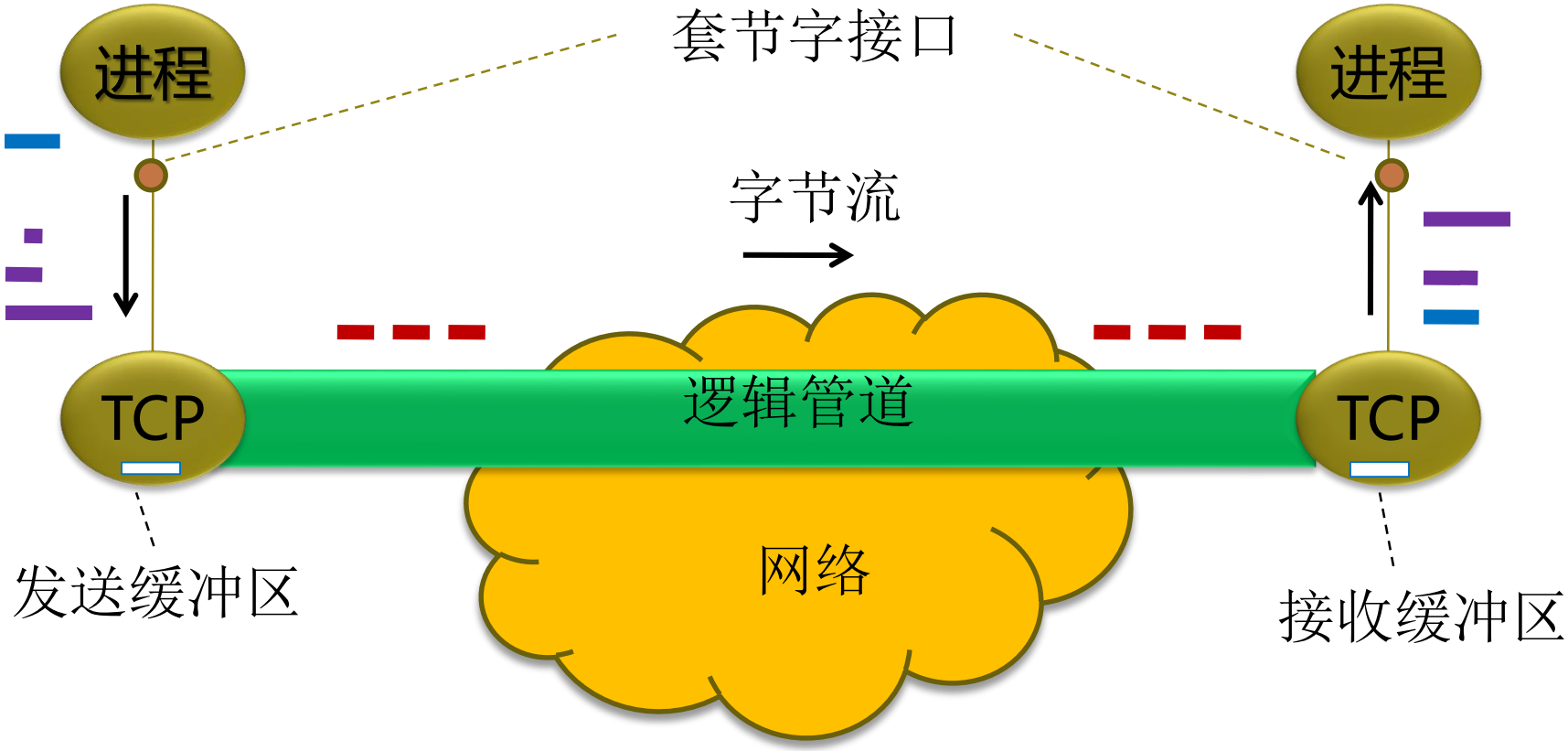
一台主机上每个端口号关联一个进程，其他进程只可以监听。

- TCP协议在数据传送之前需要先建立连接。发起建立连接请求的一方称为客户(Client)，被动等待的一方称为服务器(Server)。



- 建立TCP连接时，客户的端口号可以自己默认由系统选择一个未用的端口号。

■ **套接字函数**是TCP协议的编程接口。



每个TCP套接字都有自己的接收缓冲区和发送缓冲区。

接电话—TCP编程

13600001111

(1) 拨号　　接听

13688889999

connect　　accept

知道服务器的
IP地址和端口号

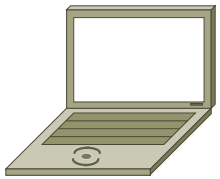
(2) 拨通

OK　　OK

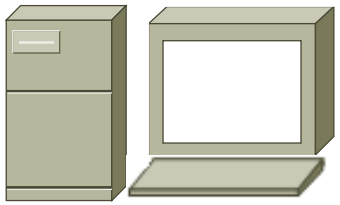
(3) 通话

Send　　Receive

Receive　　Send



客户
（Client）

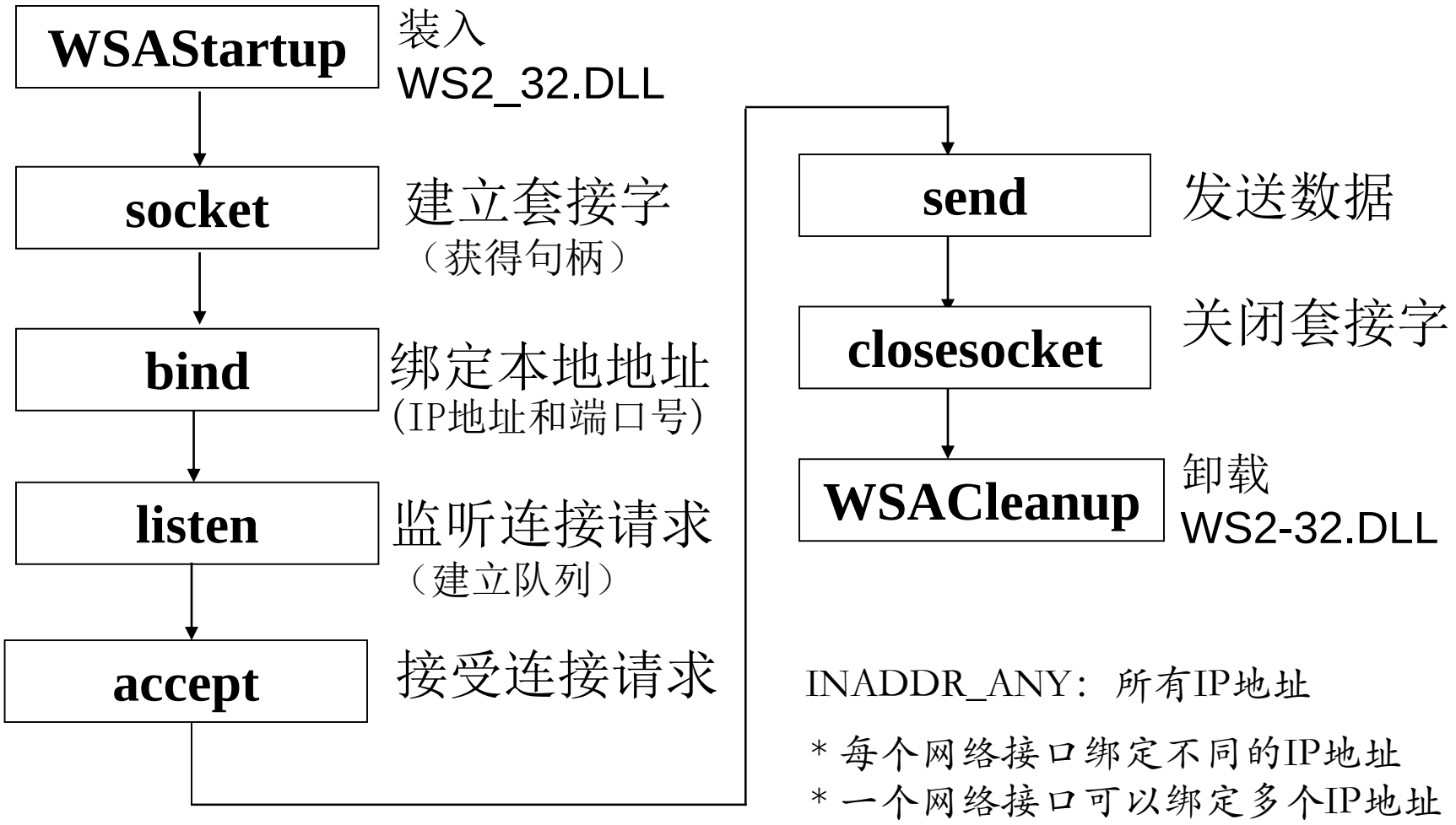


服务器
（Server）

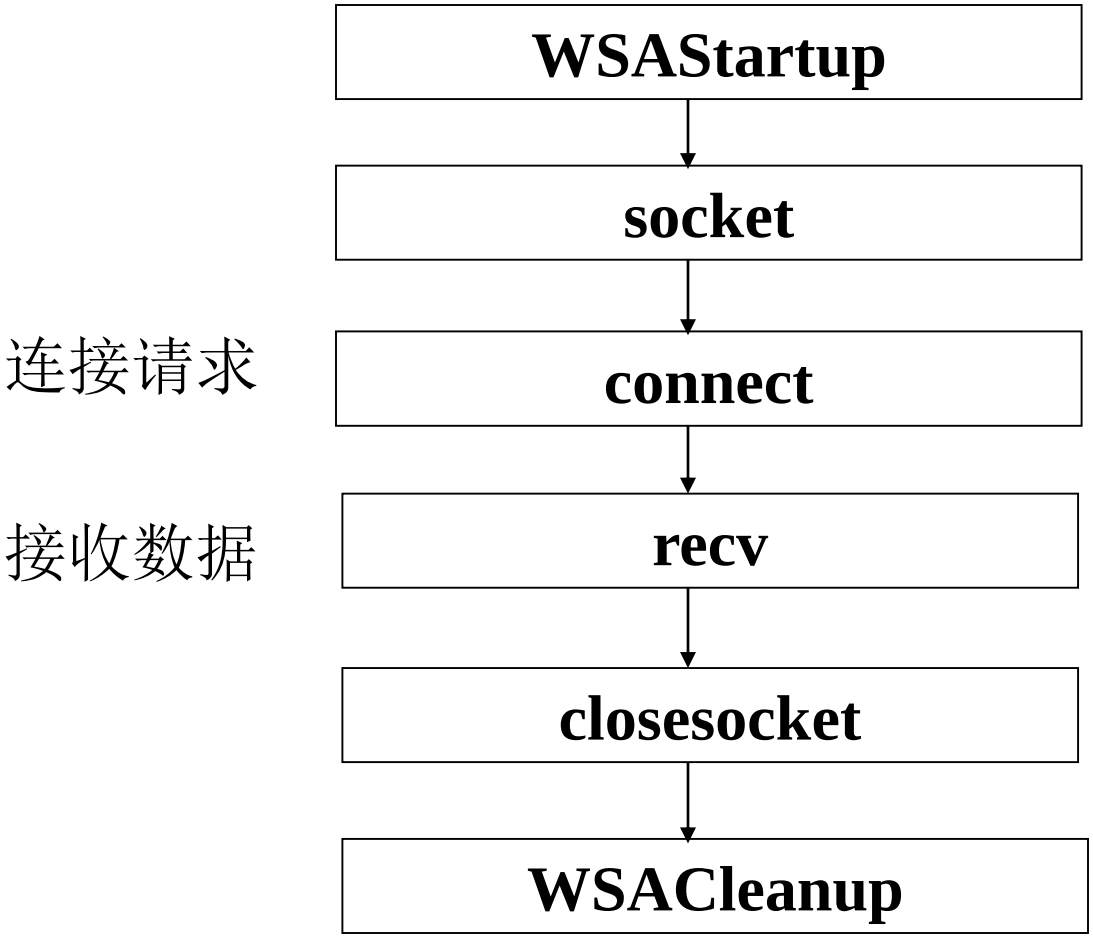
(4) 结束

CloseSocket

TCP服务器程序流程图

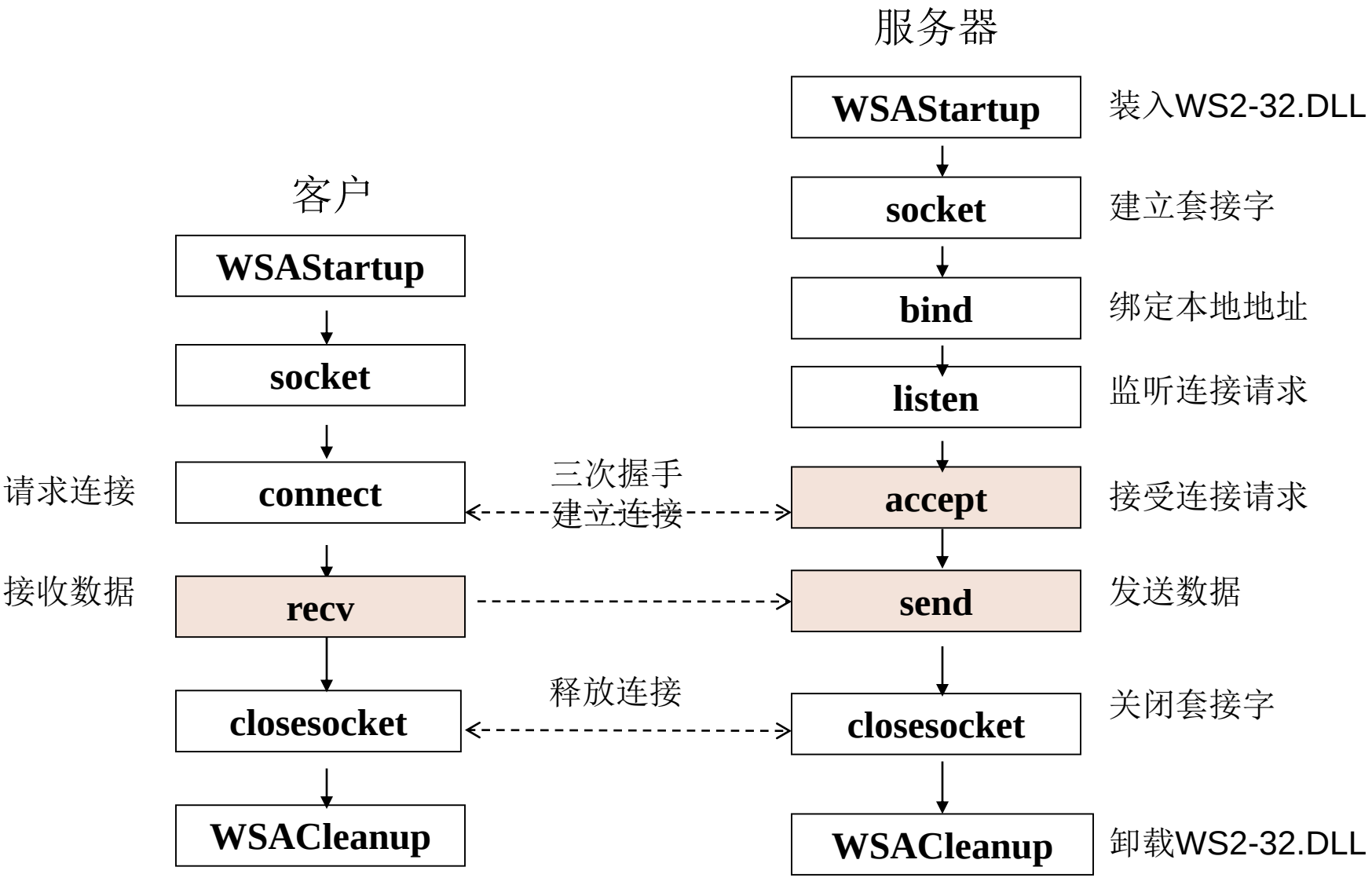


TCP客户程序流程图



默认客户端采用发送连接请求的接口的IP地址，端口采用系统未用的端口号。
可以在connect之前用bind函数绑定自己的IP地址和端口号。

TCP数据传送流程图


服务器
WSAStartup
装入WS2-32.DLL
socket
建立套接字
bind
绑定本地地址
listen
监听连接请求
accept
接受连接请求
send
发送数据
closesocket
关闭套接字
WSACleanup
卸载WS2-32.DLL
客户
WSAStartup
socket
connect
请求连接
三次握手
建立连接
recv
接收数据
closesocket
释放连接
WSACleanup

TCP服务器端程序

```
#include <stdlib.h>
#include <stdio.h>
#include <winsock2.h>
#include <time.h>
#include "conio.h"

#define WSVERS          MAKEWORD(2, 0)
#pragma comment(lib,"ws2_32.lib")     // 使用winsock 2.2 library
void  main(int argc, char *argv[])
   /* argc: 命令行参数个数，例如：C:\> TCPServer 8080
                    argc=2   argv[0]="TCPServer",   argv[1]="8080" */
```

目前Windows Socket API的版本：

Winsock 1.0、1.1（winsock.h，wsock32.lib，wsock32.dll）

Winsock 2.0（winsock2.h，ws2_32.lib，ws2_32.dll）

```
{
   WSADATA     wsadata;          /* 使用的动态链接库版本*/
   structsockaddr_in fsin;       /* 客户端的地址                    */
   SOCKET  msock, ssock;         /* master & slave sockets          */
   char  *service = "50500";     /* the local port to bind          */
   struct  sockaddr_in sin;      /* an Internet endpoint address    */
   int     alen,cc;              /* from-address length             */
   char  *pts;                   /* pointer to time string          */
   time_t   now;                 /* current time                    */
   // 加载winsock library，WSVERS为请求版本，
   //     wsadata返回系统实际支持的最高版本
   WSAStartup(WSVERS, &wsadata);
   // 创建套接字。 参数：因特网协议簇(family)，字节流，TCP协议号。
      返回：要监听套接字的描述符或INVALID_SOCKET
   msock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

使用TCP协议

```
// 从&sin开始的长度为sizeof(sin)的内存清0
memset(&sin, 0, sizeof(sin));    // sin为一个地址结构

// 因特网地址簇(INET-Internet)
sin.sin_family = AF_INET;

// 监听所有(接口的)IP地址(32位)
sin.sin_addr.s_addr = INADDR_ANY;     // 0.0.0.0

// 监听的端口号(16位) 。atoi--把ascii转化为int，htons—主机序到网络序
sin.sin_port = htons((u_short)atoi(service));

//  通过sin把要监听的IP地址和端口号绑定到套接字上
bind(msock, (struct sockaddr *)&sin, sizeof(sin));

// 建立长度为5的连接请求队列，并开始监听是否有连接请求到来，
//       来了则放入队列
listen(msock, 5);
```

```
    while(!_kbhit()) {                          // 检测是否有按键  (什么时候执行？)
        alen = sizeof(struct sockaddr);
        // accept：如果有新的连接请求，返回连接套接字，否则，被阻塞，
        //                fsin包含客户端IP地址和端口号
        ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
        (void) time(&now);      // 取得系统时间
        pts = ctime(&now);      // 把时间转换为字符串
        int len = strlen(pts);

        // 把缓冲区pts的数据发送出去，len为要发送的字节数，

        // 返回值：(>0) 实际发送的字节数(≤len), (=0) 对方正常关闭，
        //    （=SOCKET_ERROR) 出错，用函数WSAGetLastError取错误码。
        cc = send(ssock, pts,len, 0);
        shutdown(sock,SD_SEND);    // 关闭连接(不再发送数据）
        (void) closesocket(ssock);    // 关闭连接套接字
    } // while
   (void) closesocket(msock);        //关闭监听套接字
    WSACleanup();                    //卸载winsock 2.2 library
    printf("按回车键继续...\r\n");  getchar(); getchar();     // 等待任意按键
}
```

* 要确定accept使用了哪个服务器IP地址和端口号要通过msock用API查询得到。

TCP客户端程序

```
/* TCPClient.cpp */

#include <stdlib.h>
#include <stdio.h>
#include <winsock2.h>
#include <string.h>

#define BUFLEN          2000
#define  WSVERS         MAKEWORD(2, 0)
#pragma comment(lib,"ws2_32.lib")
```

```
void main(int argc, char *argv[]) {
    char *host = "127.0.0.1"; /* server ip address. 127.0.0.1指本机   */
    char *service = "50500";  /* sever port                           */
    struct sockaddr_in sin;     /* an Internet endpoint address       */
    char buf[BUFLEN+1];       /* buffer for one line of text          */
    SOCKET      sock;         /* socket descriptor                    */
    int    cc;                 /* recv character count               */
    int     res;               /* result of connect                  */

    WSADATA wsadata;
    WSAStartup(WSVERS, &wsadata);
    sock = socket(PF_INET, SOCK_STREAM,IPPROTO_TCP);
```

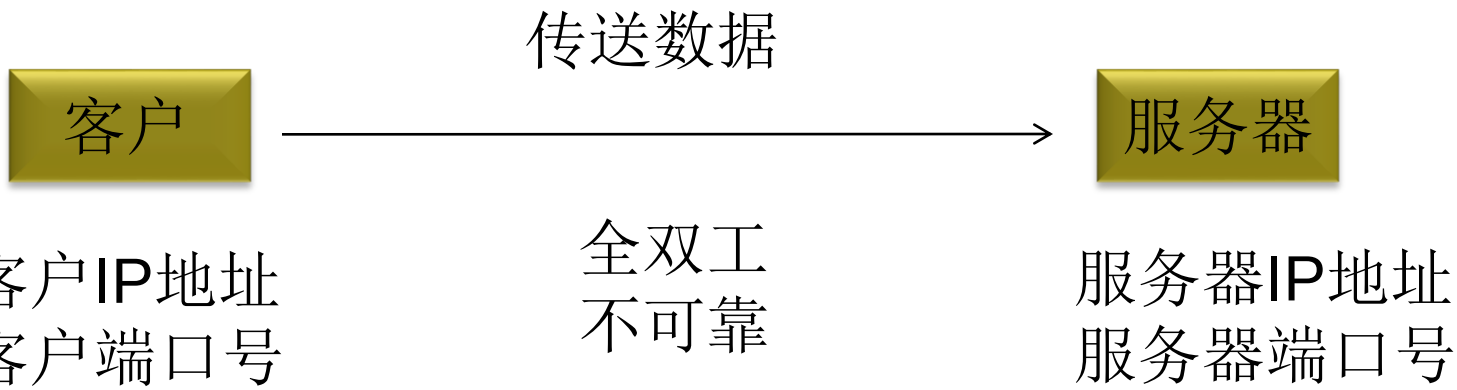
```
memset(&sin, 0, sizeof(sin));                    // sin的内存清0
sin.sin_family = AF_INET;                        // 因特网地址簇
sin.sin_addr.s_addr = inet_addr(host);           // 服务器IP地址(32位)
sin.sin_port = htons((u_short)atoi(service)); // 服务器端口号(16位)

// 连接到服务器.无错时，返回0，
//    否则，返回SOCKET_ERROR，可以调用
//              函数WSAGetLastError取得错误代码。
res = connect(sock, (struct sockaddr *)&sin, sizeof(sin));
```

函数bind加在哪里？

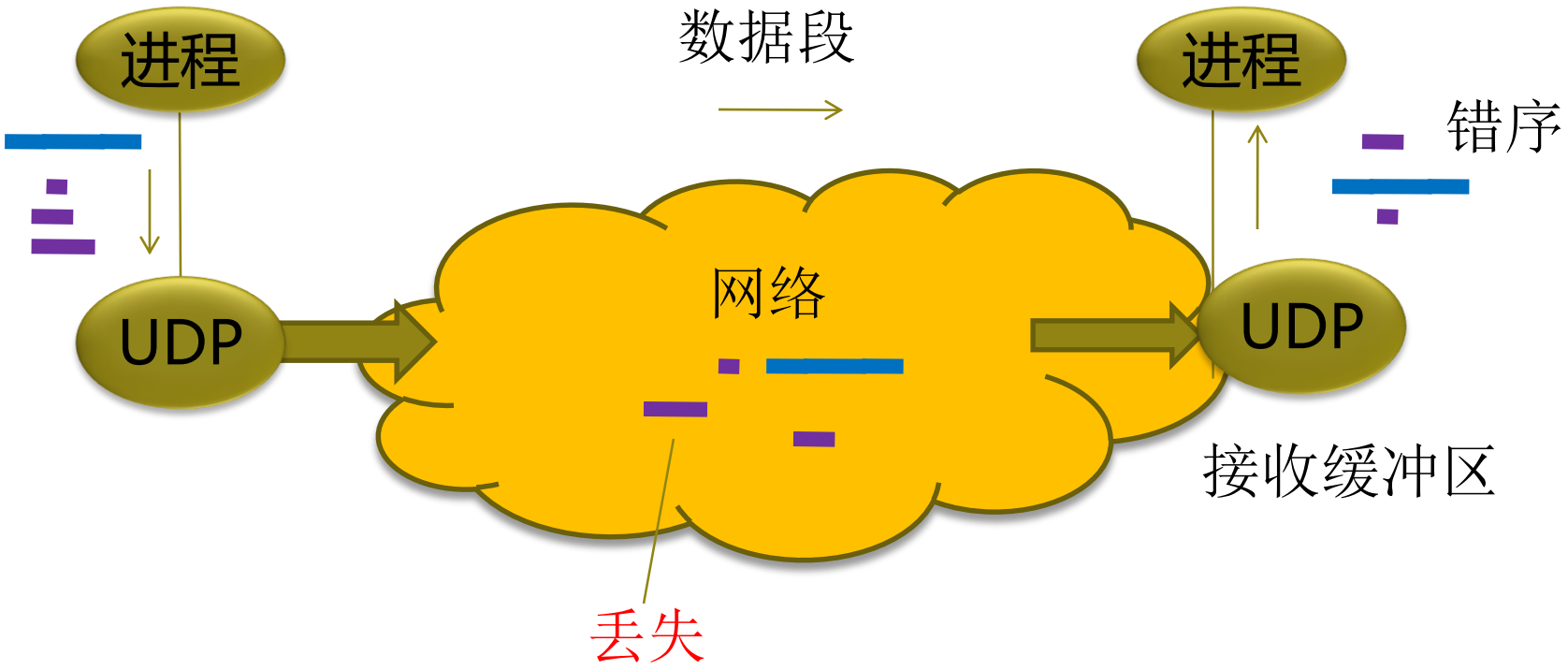
```
    // BUFLEN为缓冲区buf的长度。
    // 返回值：接收的字符数(>0)、对方已关闭(=0) 或连接出错(<0)
    cc = recv(sock, buf, BUFLEN, 0);
    if(cc == SOCKET_ERROR || cc == 0){
        printf("Error: %d.\n",GetLastError());
    }
    else  if(cc > 0) {
        buf[cc] = '\0';                  // ensure null-termination
        printf("%s",buf);                  // 显示所接收的字符串
     }
    closesocket(sock);                  // 关闭套接字
    WSACleanup();                       // 卸载winsock library
    printf("按回车键继续...");
    getchar();                          // 等待任意按键
    getchar();
}
```

UDP协议

- **UDP协议是另一个传输层协议，它提供不可靠无连接的服务。**

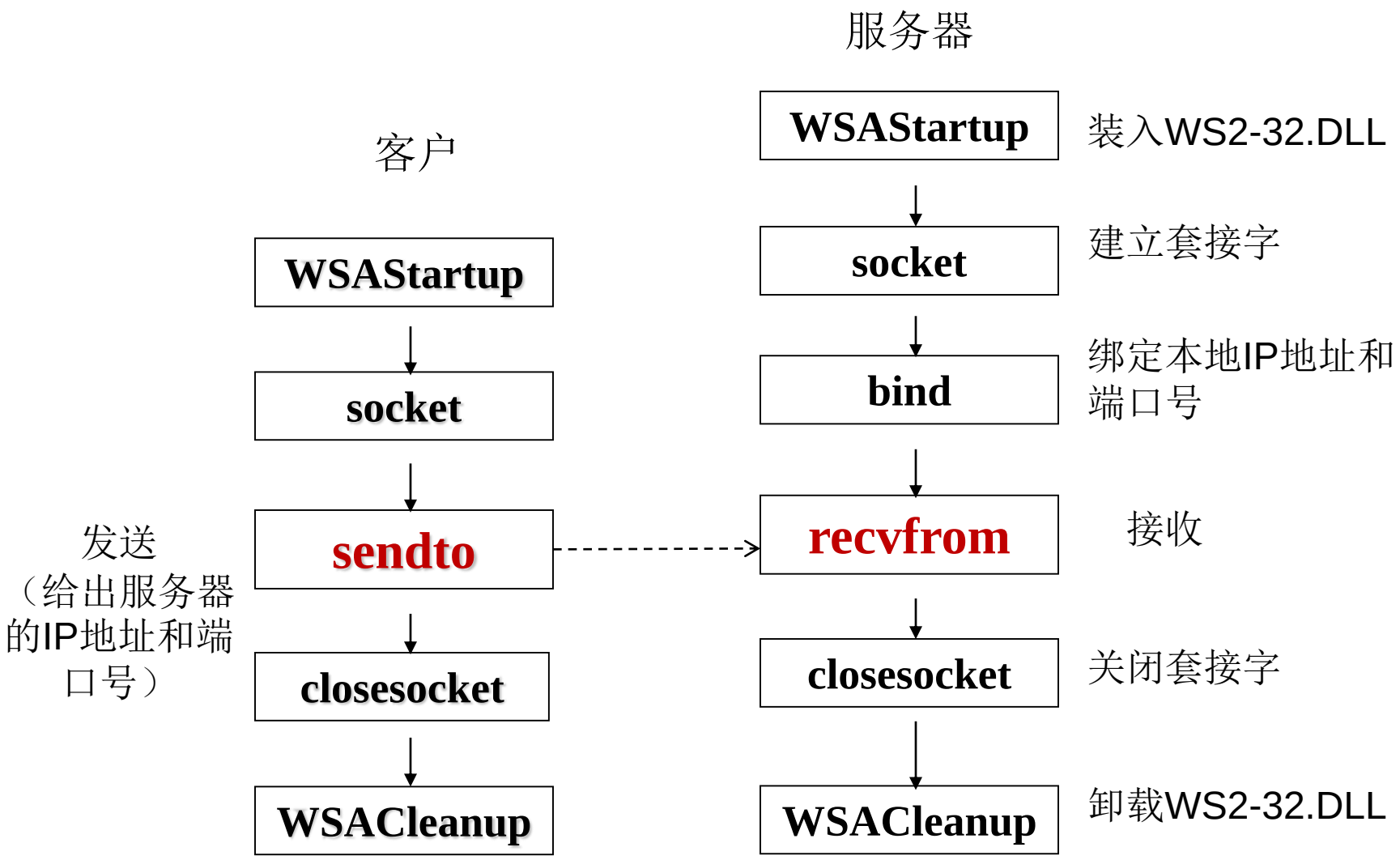


UDP协议会为每个UDP套接字设置接收缓冲区和发送缓冲区。



- UDP协议采用数据报方式传送。每次接收都接收一个数据段。接收时如果缓存区太小，数据段将被截断。

UDP数据传送流程图


客户
WSAStartup
socket
sendto
closesocket
WSACleanup
发送
（给出服务器的IP地址和端口号）
服务器
WSAStartup
装入WS2-32.DLL
socket
建立套接字
bind
绑定本地IP地址和端口号
recvfrom
接收
closesocket
关闭套接字
WSACleanup
卸载WS2-32.DLL

UDP服务器端程序

```
/* UDPServer.cpp */

#include <stdlib.h>
#include <stdio.h>
#include <winsock2.h>
#include <string.h>
#include "conio.h"

#define BUFLEN          2000
#define WSVERS                  MAKEWORD(2, 2)
#pragma comment(lib,"ws2_32.lib")
```

```
void main(int argc, char *argv[])
{     char   *host = "127.0.0.1";      /* server IP Address to connect          */
      char   *service = "50500";       /* server port to connect                */
      struct sockaddr_in sin;          /* an Internet endpoint address          */
      struct sockaddr_in from;         /* sender address                        */
      int    fromsize = sizeof(from);
      char   buf[BUFLEN+1;            /* buffer for one line of text           */
      SOCKET     sock;                /* socket descriptor                     */
      int   cc;                       /* recv character count                  */

      WSADATA wsadata;
      // 加载winsock library，WSVERS为请求版本
      // wsadata返回系统实际支持的最高版本
      WSAStartup(WSVERS, &wsadata);
```

```
// 创建UDP套接字
// 参数：因特网协议簇(family)，数据报，UDP协议号
// 返回：要监听套接字的描述符或INVALID_SOCKET
sock = socket(PF_INET, SOCK_DGRAM,IPPROTO_UDP);
// 内存清0
memset(&sin, 0, sizeof(sin));
// 因特网地址簇
sin.sin_family = AF_INET;
// 监听所有的接口(0.0.0.0)。
sin.sin_addr.s_addr = INADDR_ANY;
// 绑定指定端口号，atoi--把ascii转化为int
//          htons – 把数据(short类型)从主机序转化为网络序
sin.sin_port = htons((u_short)atoi(service));
// 绑定本地端口号和IP地址
bind(sock, (struct sockaddr *)&sin, sizeof(sin));
```

```
    while(!_kbhit()){     // 检测是否有按键
        // 接收客户数据。cc为接收的字符数，from包含客户IP地址和端口号。
        cc = recvfrom(sock, buf, BUFLEN, 0,
                                    (SOCKADDR *)&from, &fromsize);
        if (cc == SOCKET_ERROR)    {
            printf("recvfrom() failed; %d\n", WSAGetLastError());
            break;
        } else {
            buf[cc] = '\0';
            printf("%s\n", buf);    // 显示客户端发来的数据
        }
    }  //while
    closesocket(sock);
    WSACleanup();

} //main
```

UDP客户端程序

```
/* UDPClient.cpp */

#include <stdlib.h>
#include <stdio.h>
#include <winsock2.h>
#include <string.h>

#define BUFLEN                  2000                // 缓冲区大小
#define WSVERS                  MAKEWORD(2, 2) // 指明版本.2
#pragma comment(lib,"ws2_32.lib")           // 加载winsock 2.2 Llibrary
```

```
void    main(int argc, char *argv[])
{
        char  *host = "127.0.0.1";          /* server IP to connect          */
        char  *service = "50500";           /* server port to connect        */
        struct sockaddr_in  toAddr;         /* an Internet endpoint address */
        char buf[BUFLEN+1];                 /* buffer for one line of text   */
        SOCKET    sock;                     /* socket descriptor             */
        int  cc;                            /* recv character count          */
```

```
WSADATA wsadata;
WSAStartup(WSVERS, &wsadata);            // 启动Socket DLL
//创建UDP套接字
sock = socket(PF_INET, SOCK_DGRAM,IPPROTO_UDP);

memset(&toAddr, 0, sizeof(toAddr));              // 内存清0
toAddr.sin_family = AF_INET;                     // 因特网协议簇
toAddr.sin_addr.s_addr = inet_addr(host);        // 服务器IP地址
toAddr.sin_port = htons((u_short)atoi(service)); // 服务器端口号

memset(buf,'e', 1000);                           //填充1000字节的字符'e'
buf[1000] = '\0';
```

```
    //发送缓冲区数据给服务器
    cc = sendto(sock, buf, sizeof(buf),
              0,(SOCKADDR *)&toAddr, sizeof(toAddr));
    if (cc == SOCKET_ERROR)
    {
        printf("发送失败，错误号：%d\n", WSAGetLastError());
     }
    else{
        printf("发送成功，按任意键继续...");
    }
    closesocket(sock);
    WSACleanup();                          /* 卸载某版本的DLL */

    getchar();
}  // main
```

地址结构

- sockaddr结构

```
// sockaddr表示一般地址，与sockaddr_in字节数相同，可以相互转换。
struct  sockaddr {
    ushort sa_family;              // 地址簇
    char sa_data[14];              // 地址
};
```

- sockaddr_in结构

```
struct   sockaddr_in {             // 用来存放因特网的IP地址和端口号
        short sin_family;
        u_short sin_port;          // 端口号
        struct in_addr sin_addr;   // IP地址
        char sin_zero[8];
};
```

- IP地址表示

```
//IP地址是32位数，采用点分十进制数表示，例如，66.2.3.4。
//IP地址的3种表示方法：四个字节，两个字，一个32位整数
typedef struct in_addr {
        union {
                struct {
                        u_char s_b1,s_b2,s_b3,s_b4;   //四个字节
                } s_un_b;
                struct {
                        u_short s_w1,s_w2;                //两个字
                } s_un_w;
                u_long s_addr;                            //一个32位整数
        } S_un;
} IN_ADDR, *PIN_ADDR, FAR *LPIN_ADDR;
```

■ 地址转换函数

(1) atoi把字符串表示的数字(如，"80")转换成整数。

int atoi(const char *nptr);

(2) inet_addr把点分十进制的IP地址(例，"192.168.1.1")转化为32位IP地址。

unsigned long inet_addr(const char* cp)

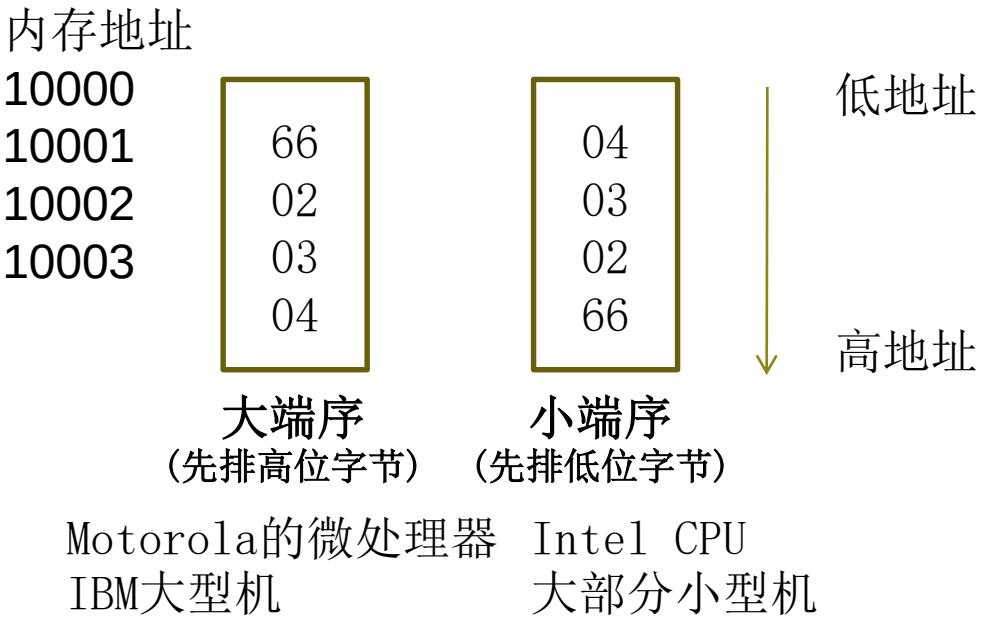
(3) inet_ntoa把32位IP地址转化为点分十进制的IP地址。

char* FAR inet_ntoa(struct in_addr in)

网络字节序函数

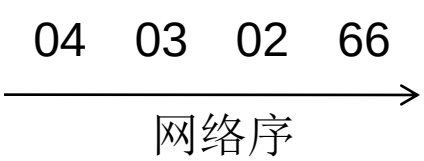
- 多字节数在主机内存的存放方式称为**主机字节序**(host byte order)，分为**大端序**(big-endian)和**小端序**(little-endian)两种。

0x66020304存放在内存的两种次序：



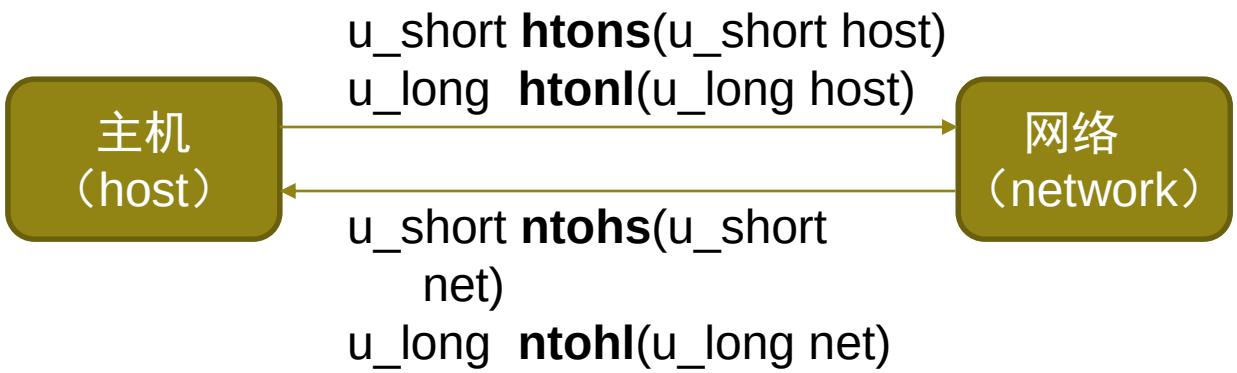
- 因特网的**网络字节序**(network byte order)采用大端序，即先发送高位字节。

发送0x66020304



- 当把多字节数从主机(host)发往网络(network)或从网络(network) 发往主机(host)时，就需要用函数进行转换，以免出现顺序错误.
- 函数**ntohs**(u_short)把一个16位数从网络字节序转换到主机字节序。
  函数**htons**(u_short)把一个16位数从主机字节序转换到网络字节序。
- 函数格式：

  u_short **htons**(u_short host)
  u_short **ntohs**(u_short net)
  u_long **htonl**(u_long host)　// 32位数，主机到网络
  u_long **ntohl**(u_long net)　// 32位数，网络到主机



函数gethostbyname

- gethostbyname函数把域名映射成IP地址，例如： 把www.sysu.edu.cn映射为202.116.64.9。

```
struct hostent FAR * gethostbyname(
        const char FAR*name;          /* 主机名或域名,如, www.sohu.com  */
)
```

- **hostent结构**

```
struct  hostent {
    char    FAR * h_name;             /* 主机官方名或域名, 如, www.sohu.com */
    char    FAR * FAR * h_aliases;    /* alias list, null-terminated */
    short   h_addrtype;               /* host address type */
    short   h_length;                 /* length of address */
    char    FAR * FAR * h_addr_list;  /* list of addresses */
   #define h_addr  h_addr_list[0]     /* 获得IP地址(网络字节序) */
};
```

函数getaddrinfo

- gethostbyname只能用于IPv4，函数getaddrinfo()已取代gethostbyname。
- getaddrinfo函数能够处理名字到地址以及服务到端口这两种转换，返回的是一个sockaddr结构的链表而不是一个地址清单。

```
int getaddrinfo(const char *hostname, const char *service,
    const struct addrinfo *hints, struct addrinfo **result );
```

参数：

hostname：一个主机名或者地址串(IPv4的点分十进制串或者IPv6的16进制串)
service：服务名可以是十进制的端口号，也可以是已定义的服务名称，如ftp、http等，hints：可以是一个空指针，也可以是一个指向某个addrinfo结构体的指针，调用者在这个结构中填入关于期望返回的信息类型的暗示。

result：本函数通过result指针参数返回一个指向addrinfo结构体链表的指针。
返回值：0——成功，非0——出错

函数getpeername和getsockname

- 用函数getpeername()可以从已连接套接字获得对方地址。

int getpeername(SOCKET s, struct sockaddr* name, int* namelen);

参数：

s	标识一个已捆绑或已连接套接口的描述字。
name	接收套接口的地址（名字）。
namelen	名字缓冲区长度。

- 用函数getsockname()可以从监听或连接套接字获得己方的地址

int getsockname(SOCKET s, struct sockaddr FAR* name,
int FAR* namelen);

参数：

s	标识一个已捆绑或已连接套接口的描述字。
name	接收套接口的地址（名字）。
namelen	名字缓冲区长度。

BSD套接字的结构*

- 采用Windows Socket API函数进行网络编程。这些函数是从UNIX BSD(Berkeley Software Distributed)的套接字版本4.3扩展来的。
- 目前Windows Socket API的版本有 Winsock 1.0、1.1和2.0，其头文件、库文件和动态链接库文件分别为:

  版本1.0和1.1: winsock.h
  wsock32.lib
  wsock32.dll

  版本2.0: winsock2.h
  ws2_32.lib
  ws2_32.dll

- BSD套接字的结构：

```
// include/linux/net.h
struct socket {
    socket_state        state;      //套接字状态
    unsigned long       flags;
    struct proto_ops    *ops;       //操作函数集
    struct fasync_struct    *fasync_list;
    struct file  *file;             //每个BSD套接字都有一个inode结点，
                                    //          通过文件对象与其关联起来
    struct sock         *sk;        //socket内部结构,与具体的协议簇(比如PF_INET)相关
    wait_queue_head_t    wait;
    short           type;           //套接字类型
    unsigned char       passcred;
};

//套接字类型：SOCK_STREAM，SOCK_DGRAM，SOCK_RAW，SOCK_RDM
              SOCK_SEQPACKET， SOCK_PACKET
```

```
//操作函数集
struct proto_ops {
    int     family;
    struct module    *owner;
    int  (*release)   (struct socket *sock);
    int  (*bind) (struct socket *sock, struct sockaddr *myaddr, int sockaddr_len);
    int  (*connect)( … );
    int  (*accept) (…);
                ……
    int   (*listen) (struct socket *sock, int len);
    int   (*sendmsg)  (…);
    int   (*recvmsg)   (…);
};


//套接字状态
typedef enum {
    SS_FREE = 0,                /* not allocated                */
    SS_UNCONNECTED,             /* unconnected to any socket    */
    SS_CONNECTING,              /* in process of connecting     */
    SS_CONNECTED,               /* connected to socket          */
    SS_DISCONNECTING            /* in process of disconnecting  */
} socket_state;
```

==完==