网络程序设计

数据报套接字编程

数据报套接字编程的适用场合

优点：灵活性　　**缺点：不可靠性**

- **推荐：**
  - **在可靠的本地环境中运行；**
  - **广播或多播应用程序；**
- **不推荐：**
  - **在不可靠的广域网环境中运行；**
  - **海量数据传输。**

数据报套接字的通信过程

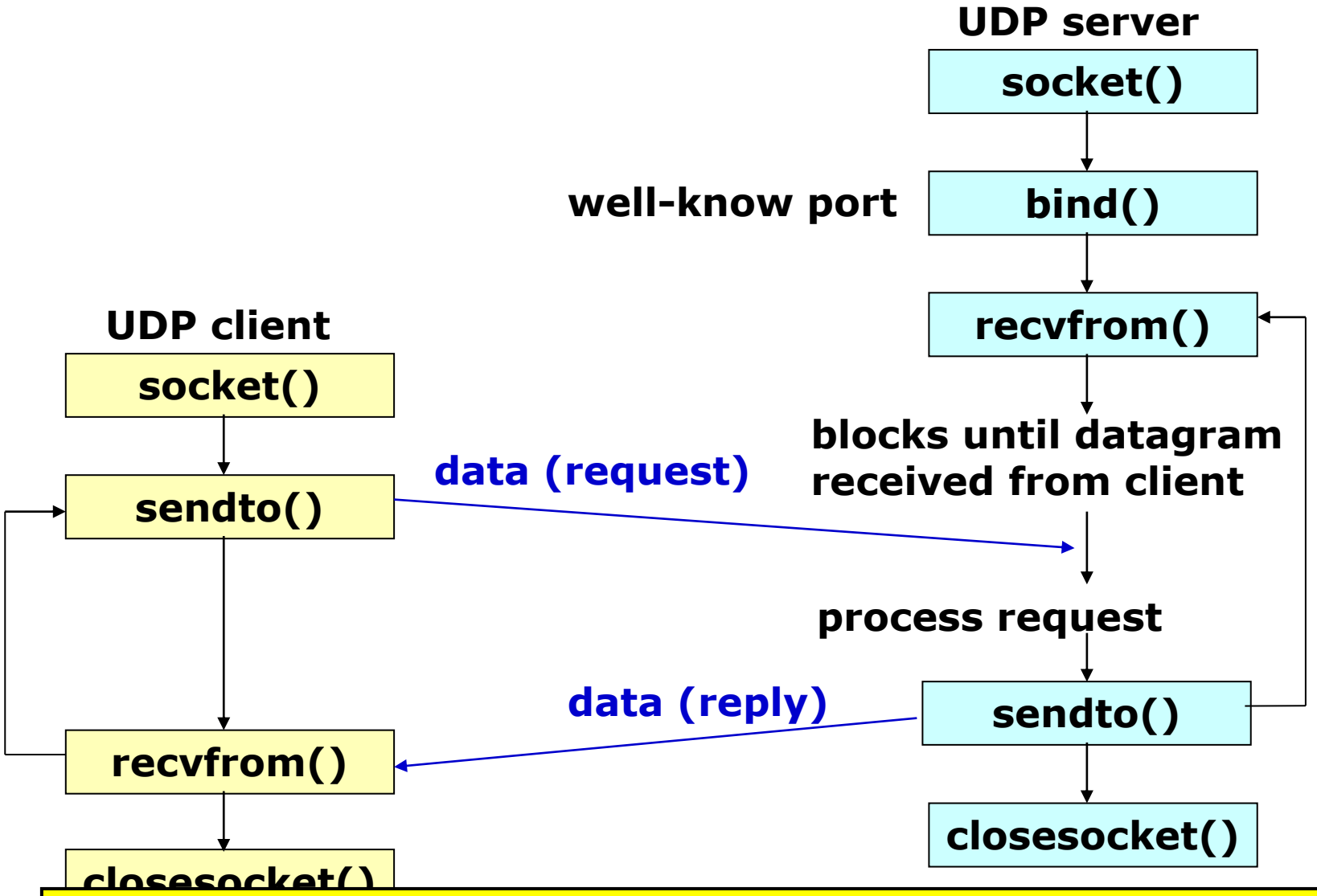
①创建套接字，指定使用UDP（不可靠的传输服务）进行通信；

②指定本地和远端IP地址和通信端口；

③进行数据传输；

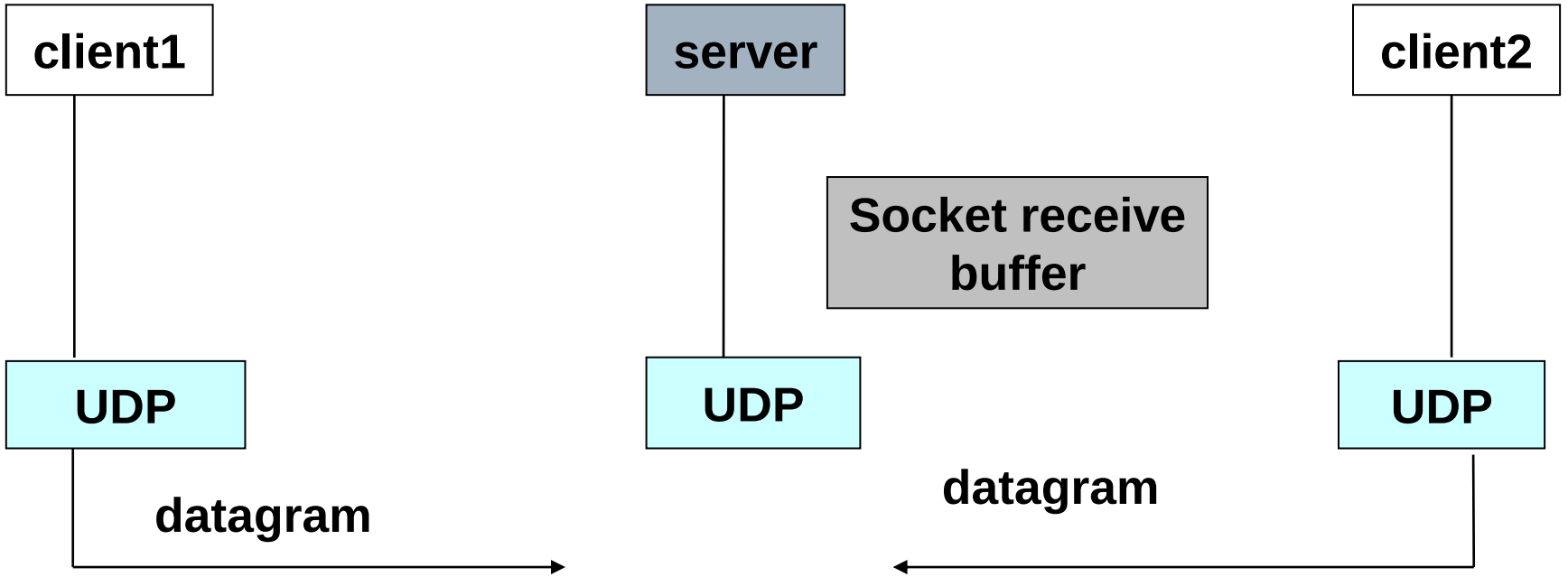
④关闭套接字；

数据报套接字的编程模型

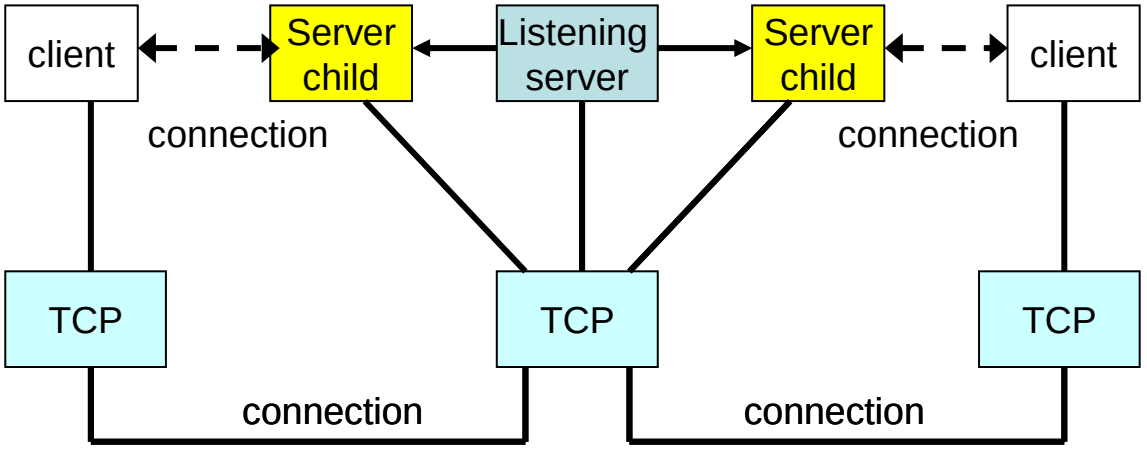


问题：server如何处理与多个client交互？

服务器工作原理



TCP服务器的工作原理

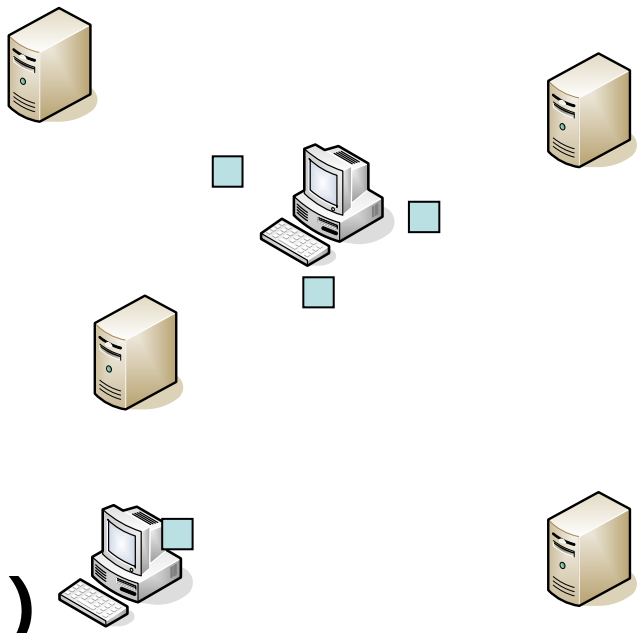


服务器的两种模式

- 循环服务器
- 并发服务器

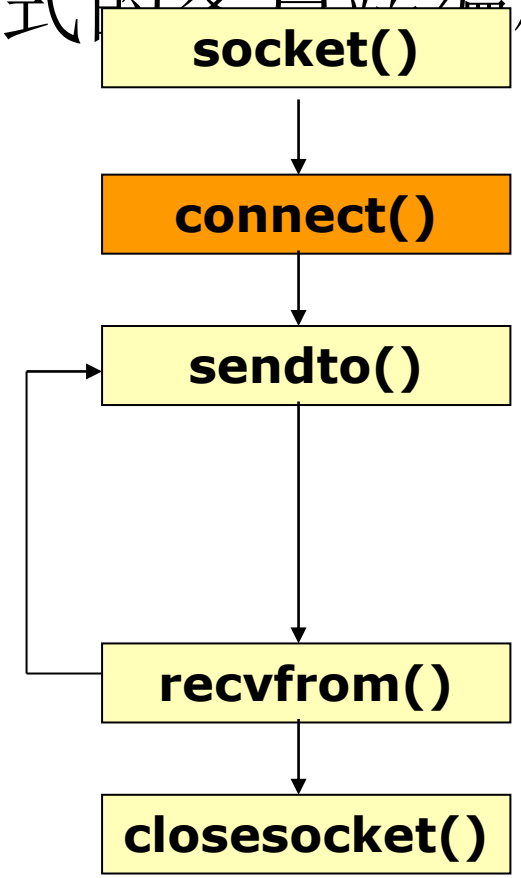
数据报套接字使用模式

- 非连接模式
  - sendto（指明目标）
  - recvfrom（记录来源）
- 连接模式
  - connect（指明远程端点地址）
  - sendto / send
  - recvfrom / recv



数据报套接字使用模式

- 使用连接模式的客户端编程



“连接”在套接字中的含义

- 对TCP而言，调用connect()将导致双方进入三次握手，建立连接。
- connect()函数只能在流式套接字上调用一次。
- 对UDP而言，调用connect()函数完全是本地操作，不会产生任何网络数据。
- 这时connect完成的功能是：在调用方为套接字关联远程主机的地址和端口号。
- 一个数据报套接字可以多次调用connect()函数。目的可能是：
  - 1)指定新的IP地址和端口号
  - 2)断开套接字(再次调用connect()函数时，把地址簇设为AF_UNSPEC)

3 基本函数

①创建套接字——socket

SOCK_DGRAM

SOCKET socket(int *af*, int *type*, int *protocol*)

②指定本地地址——bind

本地IP地址 | 本地端口号

int bind(SOCKET *s*, const struct sockaddr* *name*, int *namelen*)

③发送数据——sendto

目的IP + 目的端口号

INADDR_BROADCAST

int sendto(SOCKET *s*, const char FAR* *buf*, int *len*, int *flags*, const struct sockaddr FAR* *to*, int *tolen*)

int send(SOCKET s, const char FAR* buf, int len, int flags)

④接收数据——recvfrom

int recvfrom(SOCKET *s*, char FAR* *buf*, int *len*, int *flags*, struct sockaddr FAR* *from*, int FAR* *fromlen*)

来源IP＋来源端口号

int recv(SOCKET s, char FAR* bu

是否可通过from参数控制只接收特定来源的报文？ 否

问题1：如何向操作系统注册通信对方的地址？

- 怎样指明目标？

  - 连接模式

    connect（ ……,struct sockaddr FAR* name,…… ）

  - 非连接模式

    sendto(……, const struct sockaddr FAR* to, ……)

- 怎样获知来源？

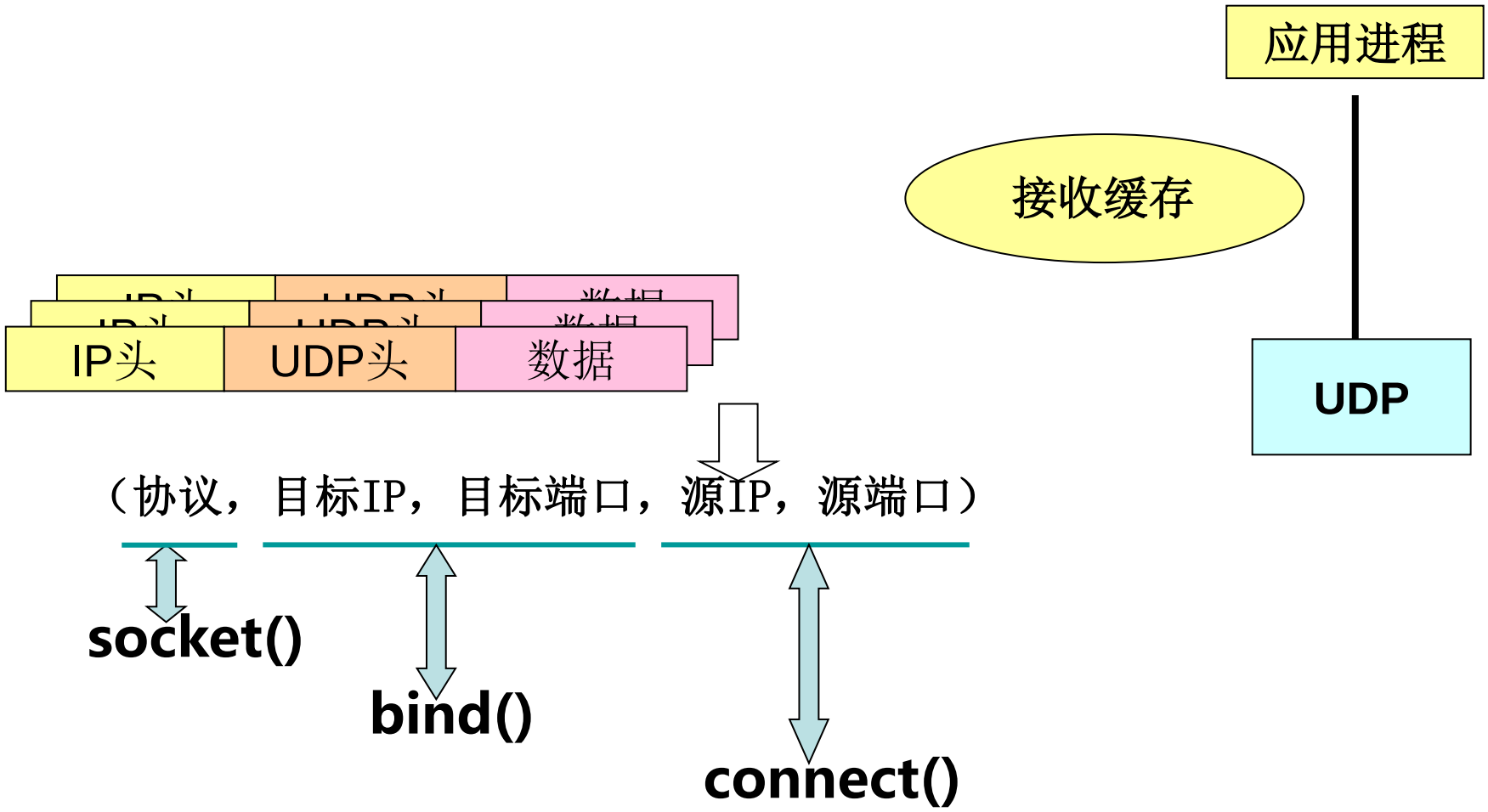
  - 连接模式

    connect（ ……,struct sockaddr FAR* name,…… ）

  - 非连接模式

    recvfrom(……,struct sockaddr FAR* from,……)

问题2：如何控制可接收的数据报类型？



问题3：如何选择合适的发送函数？

- 发送（send vs. sendto）
  - **Send**
    - 用于SOCK_STREAM:最常用
    - 用于SOCK_DGRAM:套接字地址通过连接函数connect获得
  - **Sendto**
    - 用于SOCK_DGRAM:最常用
    - 用于SOCK_STREAM: to和tolen被忽略，此时，sendto＝send

问题4：如何选择合适的接收函数？

- 接收（recv vs. recvfrom）
  - Recv：只接收已确定了连接来源的数据
    - 用于SOCK_STREAM:最常用
    - 用于SOCK_DGRAM:套接字地址通过连接函数connect获得
  - Recvfrom
    - 用于SOCK_DGRAM:最常用
    - 用于SOCK_STREAM:from和fromlen被省略，此时，recvfrom=recv

使用UDP传输的服务器回射程序

服务端程序

```c
#include <WinSock2.h>
#include <ws2tcpip.h>
#include <stdio.h>

#pragma comment(lib,"Ws2_32.lib")

//自定义的默认端口号,缓冲区长度
#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLEN 512

int main()
{
    WSADATA wsaData; //用于Winsock初始化的数据结构
    int iResult; //执行结果返回值. 为0表示成功
    struct addrinfo* result = NULL, hints; //地址信息结构
    SOCKET serverSocket = INVALID_SOCKET; //服务器套接字
    char recvbuf[DEFAULT_BUFLEN];  //接收数据缓冲区
    int recvbuflen = DEFAULT_BUFLEN; //接收数据缓冲区的长度
    int iSendResult;  //调用send时，成功发送数据的字节数
    sockaddr_in clientaddr;
    int clientlen = sizeof(sockaddr_in);

    // 初始化套接字
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup执行失败，错误码：%d\n", iResult);
        return 1;
    }
```

```
//解析服务器地址和端口号
ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = IPPROTO_UDP;
hints.ai_flags = AI_PASSIVE; //当地址无法解析时就置为0.0.0.0，便于bind
iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
if (iResult != 0) {
    printf("getaddrinfo执行错误，错误码：%d\n", iResult);
    WSACleanup();
    return 1;
}
//为无连接的服务器创建套接字
serverSocket = socket(result->ai_family, (int)result->ai_socktype, result->ai_protocol);
if (serverSocket == INVALID_SOCKET) {
    printf("socket执行错误，错误码:%d\n", WSAGetLastError());
    freeaddrinfo(result);
    WSACleanup();
    return 1;
}
//为无连接服务器套接字绑定地址和端口号
iResult = bind(serverSocket, result->ai_addr, (int)result->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    printf("bind失败，错误码:%d\n", WSAGetLastError());
    freeaddrinfo(result);
    closesocket(serverSocket);
    WSACleanup();
    return 1;
}
freeaddrinfo(result);
printf("UDP服务器起动\n");
```

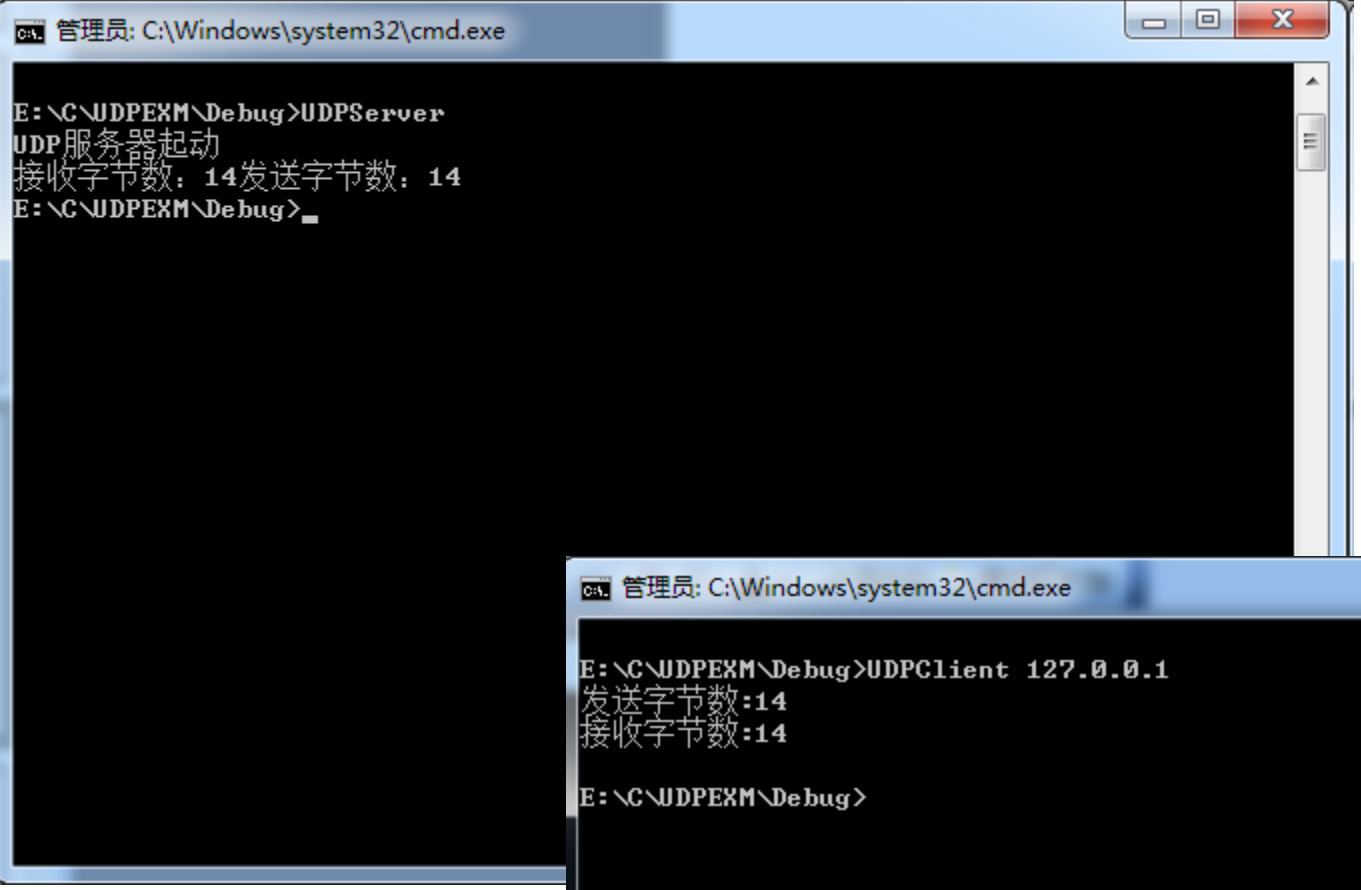
```
    //接收数据
    ZeroMemory(&clientaddr,sizeof(clientaddr));
    iResult = recvfrom(serverSocket, recvbuf, recvbuflen, 0,(sockaddr*)&clientaddr,&clientlen);
    if (iResult > 0) {
        //成功收到数据
        printf("接收字节数：%d", iResult);
        //将缓冲区的内容回送给客户端
        iSendResult = sendto(serverSocket, recvbuf, iResult, 0, (sockaddr*)&clientaddr, clientlen);
        if (iSendResult == SOCKET_ERROR) {
            printf("send失败，错误码:%d\n", WSAGetLastError());
            closesocket(serverSocket);
            WSACleanup();
            return 1;
        }
        printf("发送字节数：%d", iSendResult);
    }
    else if (iResult == 0) {
        //连接关闭
        printf("连接关闭...");
    }
    else {
        //接收发生错误
        printf("recv失败，错误码:%d\n", WSAGetLastError());
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }
    //关闭套接字，释放资源
    closesocket(serverSocket);
    WSACleanup();
    return 0;
}
```

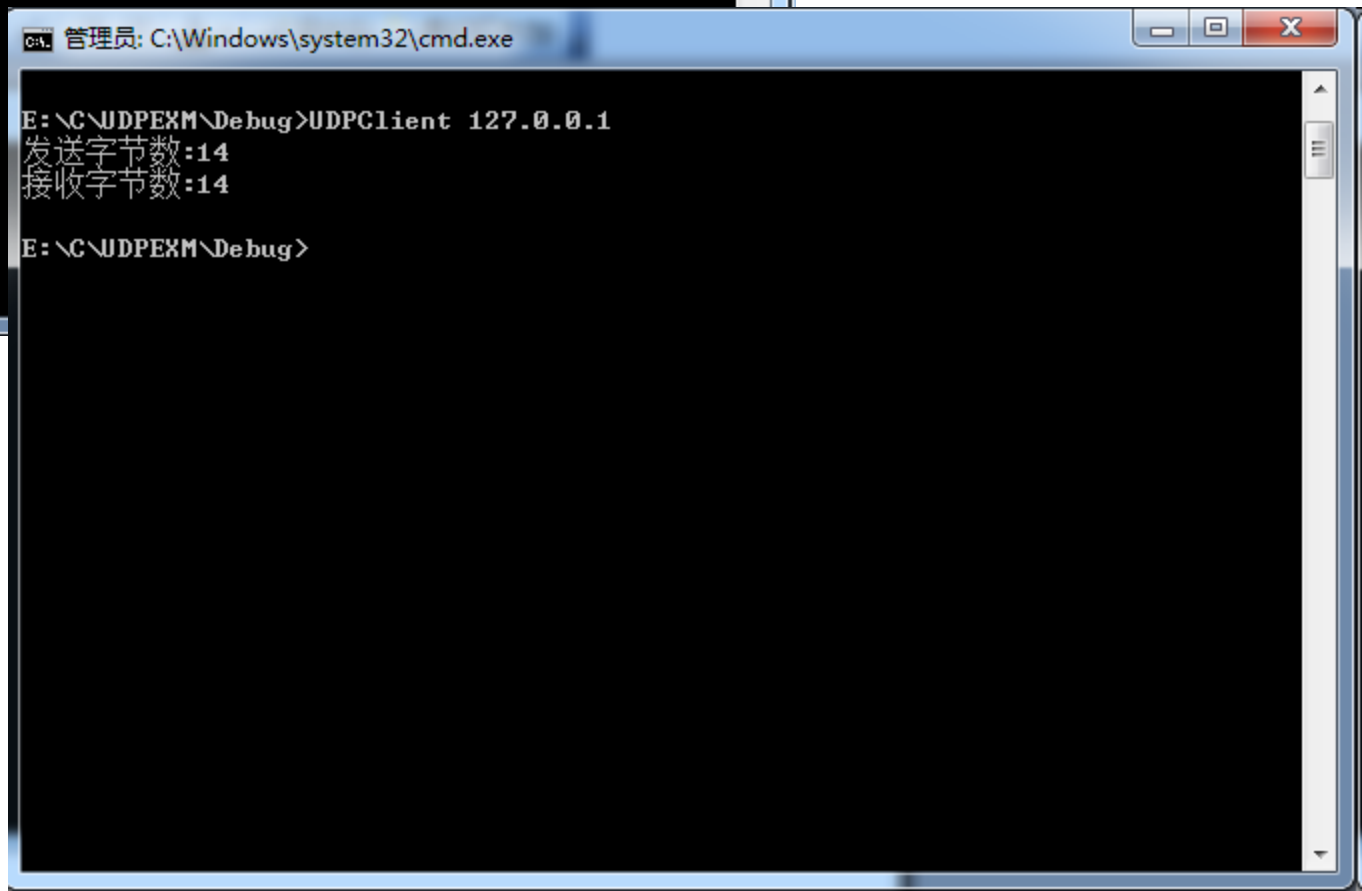
客户端程序

```c
#include <WinSock2.h>
#include <ws2tcpip.h>
#include <stdio.h>


#pragma comment(lib,"Ws2_32.lib")

//自定义的默认端口号,缓冲区长度
#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLEN 512

int main(int argc,char **argv)
{
    WSADATA wsaData;
    int iResult;
    SOCKET connectLessSocket = INVALID_SOCKET;
    struct addrinfo* result = NULL, hints;
    const char* sendbuf = "This is a test"; //要发送的测试数据
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    //验证参数合法性
    if (argc != 2) {
        printf("命令用法： %s 服务器域名或ip\n", argv[0]);
        return 1;
    }
    //初始化套接字
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup执行失败，错误码：%d\n", iResult);
        return 1;
    }
```

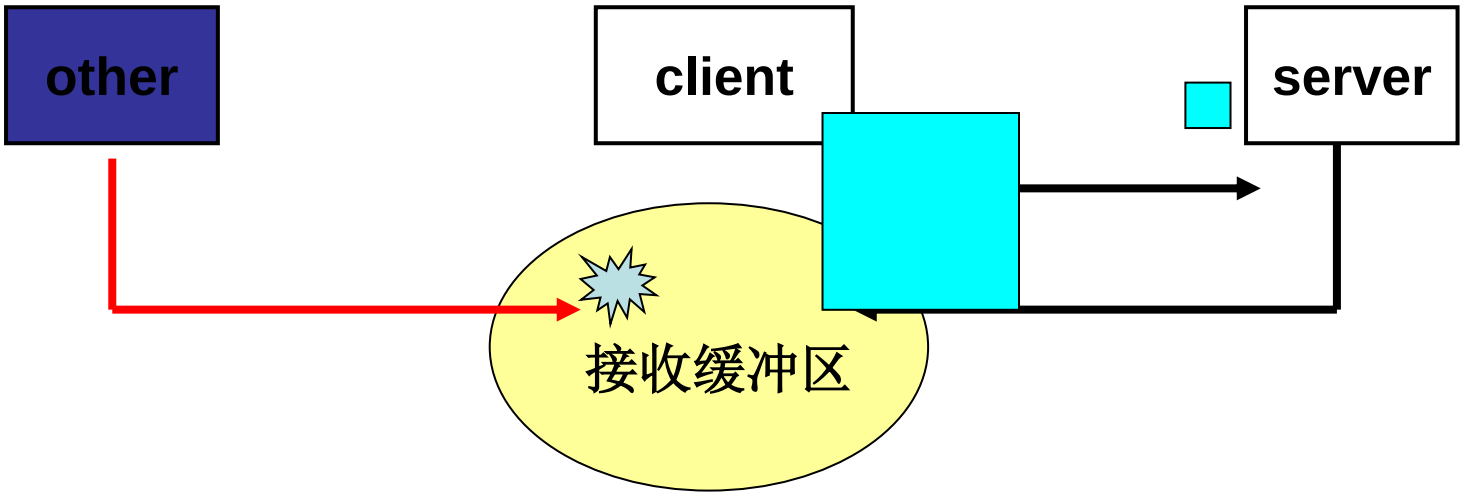
```
//解析服务器地址和端口号
ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = IPPROTO_UDP;
iResult = getaddrinfo(argv[1], DEFAULT_PORT, &hints, &result);
if (iResult != 0) {
    printf("getaddrinfo执行错误，错误码：%d\n", iResult);
    WSACleanup();
    return 1;
}
//创建数据报套接字
connectLessSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
if (connectLessSocket == INVALID_SOCKET) {
    printf("socket执行失败,错误码:%d\n", WSAGetLastError());
    WSACleanup();
    return 1;
}
```

```
    //发送缓冲区中的测试数据报
    iResult = sendto(connectLessSocket, sendbuf, (int)strlen(sendbuf), 0,result->ai_addr,result->ai_addrlen);
    if (iResult == SOCKET_ERROR) {
        printf("发送失败,错误码:%d\n", WSAGetLastError());
        closesocket(connectLessSocket);
        WSACleanup();
        return 1;
    }
    freeaddrinfo(result);
    printf("发送字节数:%ld\n", iResult);
    //接收数据
    iResult = recvfrom(connectLessSocket, recvbuf, recvbuflen, 0,NULL,NULL);
    if (iResult > 0)
        printf("接收字节数:%d\n", iResult);
    else if (iResult == 0)
        printf("连接关闭\n");
    else
        printf("接收数据失败，错误码: %d\n", WSAGetLastError());
    //关闭套接字
    closesocket(connectLessSocket);
    //释放资源
    WSACleanup();
    return 0;
}
```


管理员: C:\Windows\system32\cmd.exe
E:\C\UDPEXM\Debug>UDPServer
UDP服务器起动
接收字节数：14发送字节数：14
E:\C\UDPEXM\Debug>


管理员: C:\Windows\system32\cmd.exe
E:\C\UDPEXM\Debug>UDPClient 127.0.0.1
发送字节数:14
接收字节数:14
E:\C\UDPEXM\Debug>

5.UDP的不可靠性问题



问题1：如果出现了噪音数据，客户端应该怎样分辨？

问题2：如果报文丢失，如何处理？

问题3：如果服务器没有启动， 客户端如何知道？

问题4：如果客户端发送的数据量太大， 服务器如何处理？

排除噪声数据

- 噪声的产生主要有以下两种情形
  - 无关的数据源发送的无关数据（IP地址和端口号不对）
  - 相关的数据源发送的无关数据（比如长度不对、类型不对或内容有误）
- 针对第一种，我们可以通过增加对数据源的判断来确切地接收已知来源的数据。

问题1:如果出现了噪音数据，客户端应该怎样分辨？

解决：增加对数据源的判断

```
//接收服务器发回的响应
recvfrom(sockfd, recvline, MAXLINE,0,NULL,NULL);
```

```
……
//发送用户输入的数据
sendto(sockfd,sendline,strlen(sendline),0, pservaddr,servlen);
//接收服务器发回的响应
recvfrom(sockfd, recvline, MAXLINE,0,preplyaddr,&len);
if(memcmp(pservaddr,preplyaddr,len)==0)
{
        输出结果;
}
……
```

改写后的客户端程序

```c
#include <WinSock2.h>
#include <ws2tcpip.h>
#include <stdio.h>


#pragma comment(lib,"Ws2_32.lib")

//自定义的默认端口号,缓冲区长度
#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLEN 512

int main(int argc, char** argv)
{
    WSADATA wsaData;
    int iResult;
    SOCKET connectLessSocket = INVALID_SOCKET;
    struct addrinfo* result = NULL, hints;
    const char* sendbuf = "This is a test"; //要发送的测试数据
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;
    struct sockaddr_in reply_addr; //存储应答报文的源地址
    int addrlen = sizeof(sockaddr_in);

    //验证参数合法性
    if (argc != 2) {
        printf("命令用法： %s 服务器域名或ip\n", argv[0]);
        return 1;
    }
    //初始化套接字
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup执行失败，错误码：%d\n", iResult);
        return 1;
    }
```

```
    //解析服务器地址和端口号
    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_protocol = IPPROTO_UDP;
    iResult = getaddrinfo(argv[1], DEFAULT_PORT, &hints, &result);
    if (iResult != 0) {
        printf("getaddrinfo执行错误，错误码: %d\n", iResult);
        WSACleanup();
        return 1;
    }
    //创建数据报套接字
    connectLessSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
    if (connectLessSocket == INVALID_SOCKET) {
        printf("socket执行失败,错误码:%d\n", WSAGetLastError());
        WSACleanup();
        return 1;
    }
    //发送缓冲区中的测试数据报
    iResult = sendto(connectLessSocket, sendbuf, (int)strlen(sendbuf), 0, result->ai_addr, result->ai_addrlen);
    if (iResult == SOCKET_ERROR) {
        printf("发送失败,错误码:%d\n", WSAGetLastError());
        closesocket(connectLessSocket);
        WSACleanup();
        return 1;
    }
    printf("发送字节数:%ld\n", iResult);
    //接收数据
    iResult = recvfrom(connectLessSocket, recvbuf, recvbuflen, 0, (sockaddr *)&reply_addr, &addrlen);
    if (iResult > 0 && memcmp(result->ai_addr,&reply_addr,addrlen)==0)//注意，转入的是地址或指针
        printf("接收字节数:%d\n", iResult);
    else if (iResult == 0)
        printf("连接关闭\n");
    else
        printf("接收数据失败，错误码: %d\n", WSAGetLastError());
    //释放地址
    freeaddrinfo(result);
    //关闭套接字
    closesocket(connectLessSocket);
    //释放资源
    WSACleanup();
    return 0;
}
```

排除噪声数据

- 如果客户在一段时间内只有一个唯一的服务器通信，那么对无关来源的判断还可以通过数据报套接字的连接模式来实现。
- 在套接字发送和接收数据前，先调用connect()函数来注册通信的对方地址，那么该函数能够保证此后接收到的数据只能是连接声明的数据源发回的，其他数据源发送的数据报都不会 被协议栈提交给改套接字。

客户端使用connect示例

```c
#include <WinSock2.h>
#include <ws2tcpip.h>
#include <stdio.h>


#pragma comment(lib,"Ws2_32.lib")

//自定义的默认端口号,缓冲区长度
#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLEN 512

int main(int argc, char** argv)
{
    WSADATA wsaData;
    int iResult;
    SOCKET connectLessSocket = INVALID_SOCKET;
    struct addrinfo* result = NULL, hints;
    const char* sendbuf = "This is a test"; //要发送的测试数据
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    //验证参数合法性
    if (argc != 2) {
        printf("命令用法： %s 服务器域名或ip\n", argv[0]);
        return 1;
    }
    //初始化套接字
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup执行失败，错误码：%d\n", iResult);
        return 1;
    }
```

```
//解析服务器地址和端口号
ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = IPPROTO_UDP;
iResult = getaddrinfo(argv[1], DEFAULT_PORT, &hints, &result);
if (iResult != 0) {
    printf("getaddrinfo执行错误，错误码：%d\n", iResult);
    WSACleanup();
    return 1;
}
//创建数据报套接字
connectLessSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
if (connectLessSocket == INVALID_SOCKET) {
    printf("socket执行失败,错误码:%d\n", WSAGetLastError());
    WSACleanup();
    return 1;
}
//调用connect()函数使套接字进入连接模式
iResult = connect(connectLessSocket, result->ai_addr, (int)result->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    closesocket(connectLessSocket);
    WSACleanup();
    return 1;
}
freeaddrinfo(result);
```

```
    //发送缓冲区中的测试数据报
    iResult = sendto(connectLessSocket, sendbuf, (int)strlen(sendbuf), 0,NULL,0);
    if (iResult == SOCKET_ERROR) {
        printf("发送失败,错误码:%d\n", WSAGetLastError());
        closesocket(connectLessSocket);
        WSACleanup();
        return 1;
    }
    printf("发送字节数:%ld\n", iResult);
    //接收数据
    iResult = recvfrom(connectLessSocket, recvbuf, recvbuflen, 0, NULL, NULL);
    if (iResult > 0)
        printf("接收字节数:%d\n", iResult);
    else if (iResult == 0)
        printf("连接关闭\n");
    else
        printf("接收数据失败，错误码：%d\n", WSAGetLastError());
    //关闭套接字
    closesocket(connectLessSocket);
    //释放资源
    WSACleanup();
    return 0;
}
```

当然，这里直接用send和recv也可以

排除噪声数据

- 针对第二种情况，即相关的数据源发送无关的数据。我们还需要对数据内容特征进行判断。
  - 1)增加对数据长度的判断，过小的报文可能由于传输环节上的接收截断导致，需要丢弃
  - 2)增加对数据类型的定义和判断（TLV）
  - 3)增加对数据发送源的身份鉴别来保护资源的安全性。

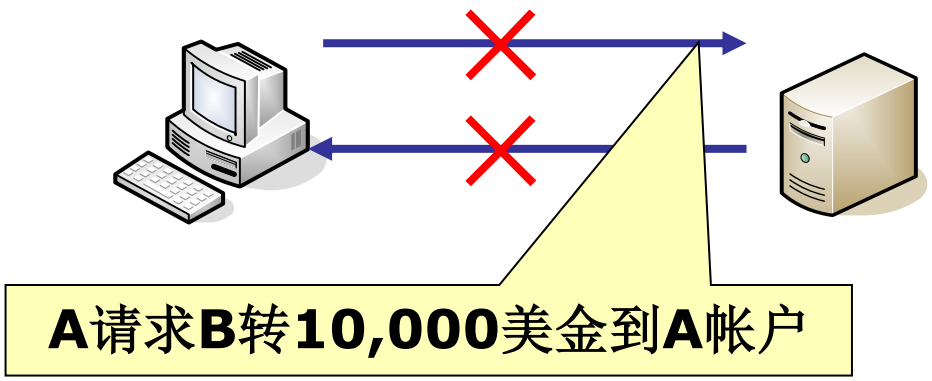
问题2：如果报文丢失，如何处理？

解决：增加对recvfrom的超时判断

- **int setsockopt(SOCKET** *s*, **int** *level*, **int** *optname*, **const char*** *optval*, **int** *optlen* **);**

```
//创建套接字
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
//服务器地址赋值：……
//设置接收超时
nTimeOver=1000;//超时时限为1000ms
setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO,
(char*)&nTimeOver,sizeof(nTimeOver));
while(fgets(sendline,MAXLINE,fp)!=NULL)
{
        发送请求;
        接收应答;
}
……
```

- 报文丢失情况
  - 请求报文丢失
  - 响应报文丢失



进一步解决：带确认的数据报服务

- 序列号：用于客户验证一个应答是否匹配相应的请求
- 超时和重传：用于处理丢失的数据报

问题3：如果服务器没有启动，客户端如何知道？

- **原理：当UDP不开放服务时返回端口不可达的ICMP报文。**

解决：用ICMP端口不可达错误来判断服务器没有启动。

Windows Sockets增加了ICMP错误的反馈，如果服务器未启动，则客户的recvfrom()调用会返回错误，通过WSAGetLastError()得到错误码来判断错误原因。

- **问题：当套接字同时访问多个服务器时，如何判断哪一个服务器没有启动？**

- **解决：**
  - **方法1：使用连接模式**
  - **方法2：使用原始套接字接收ICMP报文**

问题4：如果客户端发送的数据量太大，服务器如何处理？

- 现象：服务器计数收到的数据报个数小于客户端实际发送的数据报个数。
- 解释：UDP缺乏流量控制
- 解决：
  - 预先协商最大的请求和应答数据量
  - 使用SO_RCVBUF设置接收缓存

- Windows系统中的接收缓冲区的大小默认是8KB，最大是8MB

UDP服务器的并发性

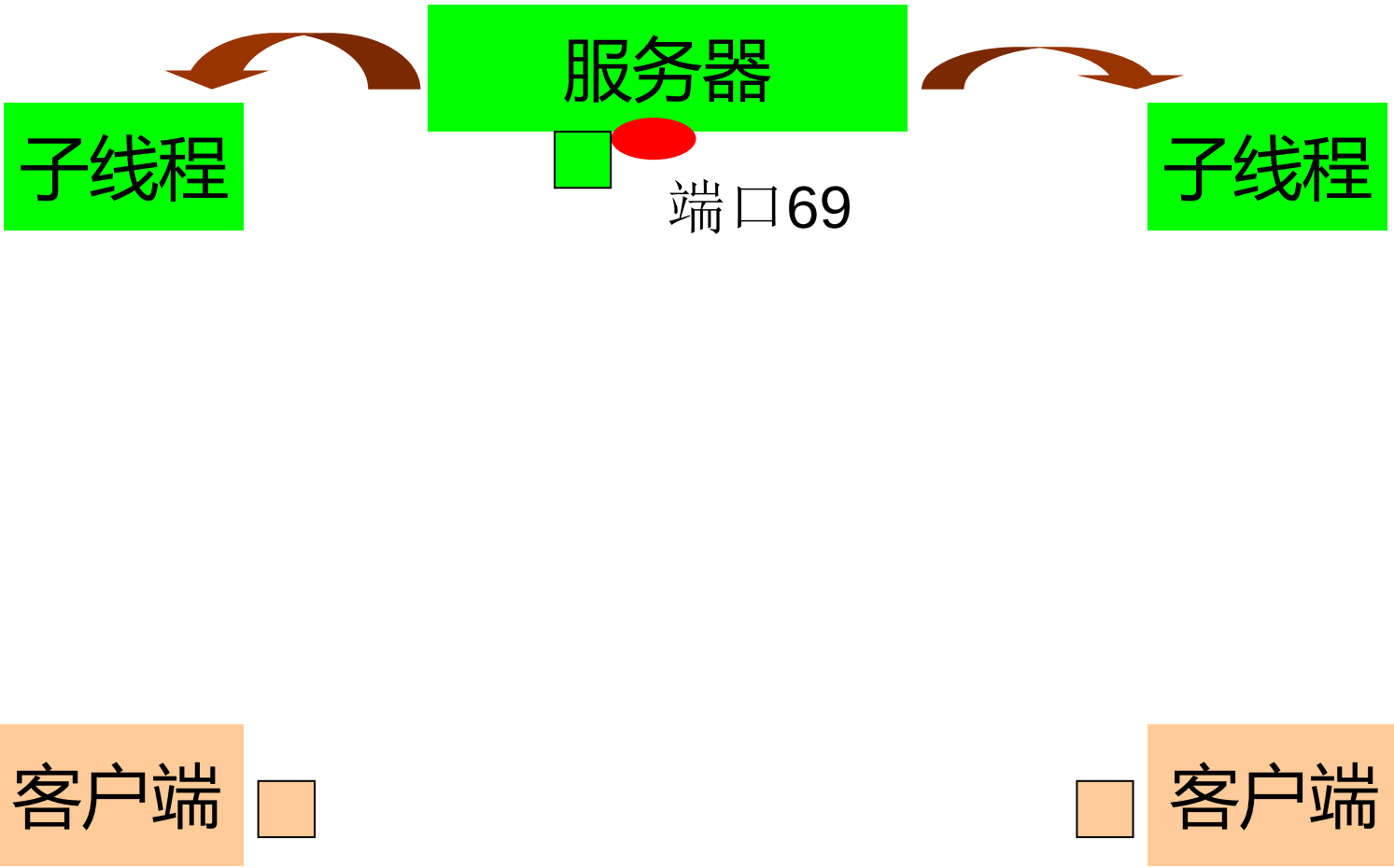
- 循环UDP服务器
  - 服务器等待一个客户请求，读入这个请求，处理这个请求，送回其应答，接着等待下一个请求。

问题：何时需要并发UDP服务器？

时机：当客户的请求处理耗时过长时

- 并发UDP服务器
  - 单次交互的客户请求
  - 多次交互的客户请求

单次交互客户请求的并发处理



多次交互客户请求的并发处理

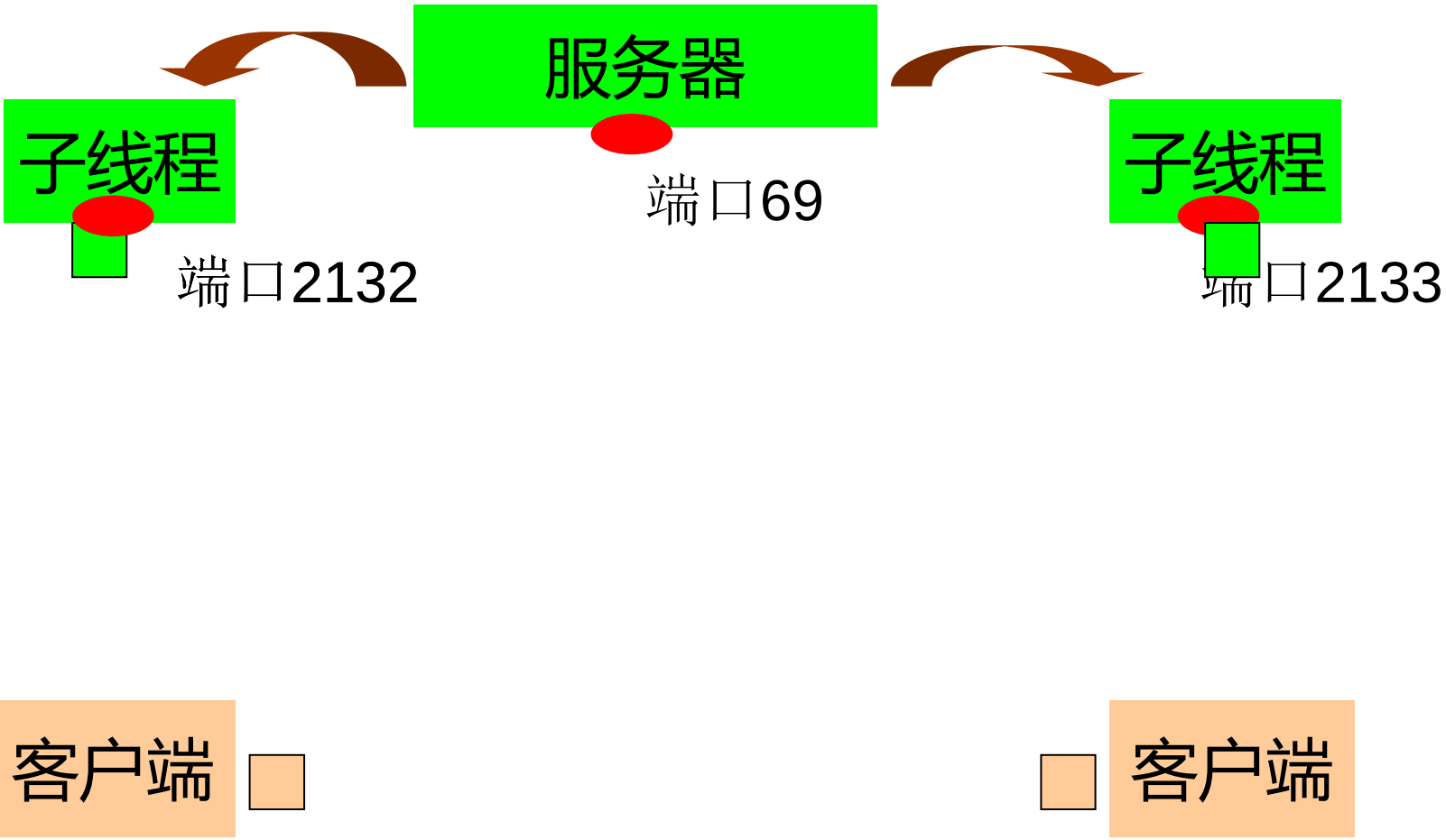
服务器如何区分来自客户的第一个请求和后续请求？

服务器在新的套接口上绑定临时端口

客户端如何获知服务器的后续响应？

将服务器第一个应答中的源端口号作为后续目的端口

多次交互客户请求的并发处理



7.2 广播程序设计

- 根据参与一次通信的对象的多少，可将通信分为两大类：一类是点对点通信，也称其为单播通信，TCP协议仅支持这种单播通信方式；一类是多点通信，也称为群通信或组通信，UDP协议既支持单播通信方式，又支持多点通信方式。
- 多点通信方式又分为广播和多播两种。
- 所谓广播是指一种同时与单一网络中的所有主机进行交互数据的通信方式，传输者通过一次数据传输就可以使网络上的所有主机接收到这个数据信息。
- 广播通信的主要用于减少网络数据流量和资源查找。

广播地址

- 用于表示网络中所有主机的地址称为广播地址。
- 广播地址有直接地址和有限地址之分。
  - 直接广播地址用于向一个指定的网络(已知网络号)发送数据包的情况，比如，向网络号为192.168.2.0/24的网络中的所有主机发送一个广播数据包，其所使用的目的地址就应该是直接广播地址192.168.2.255；
  - 若向本地网络发送广播数据包，则需要使用有限广播地址255.255.255.255。

广播程序的设计步骤

- 要在程序中实现广播数据的发送，需要使用数据报套接字。
- 但数据报套接字在默认情况下是不能广播数据的，要让数据报套接字能够发送广播数据，必须先使用setsockopt()函数对相应的套接字选项进行设置。
- 除了需要对新创建的套接字设置广播选项外，编写广播程序的其它步骤与普通的数据报套接字编程的步骤基本相同。

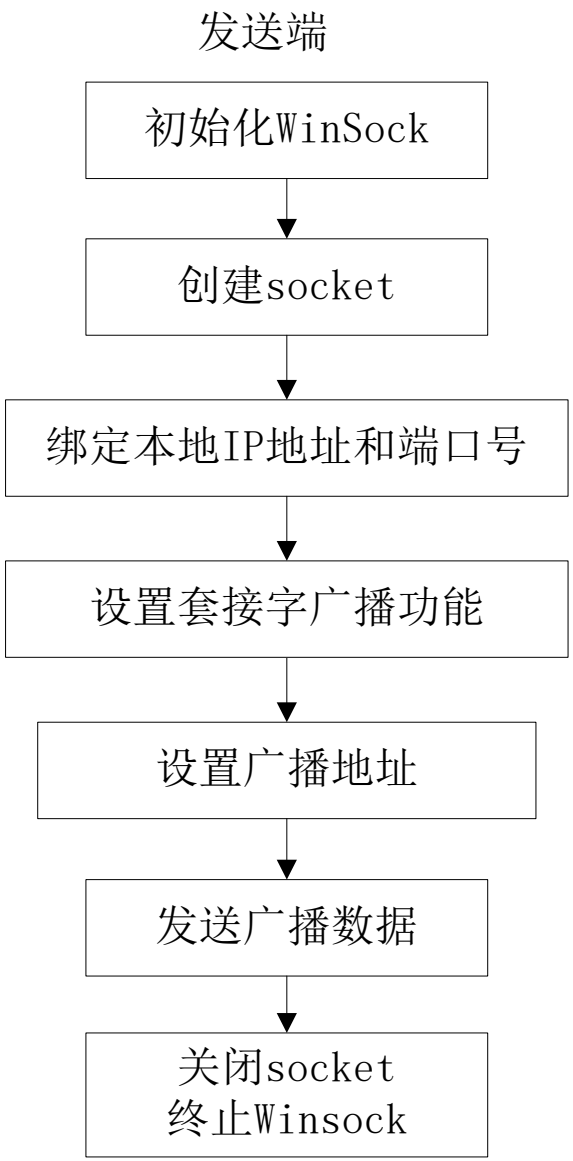
设置套接字的广播选项

- 套接字选项（Option）也称为套接字属性，诸如套接字接收缓冲区的大小、加急数据是否允许在普通数据流中接受、是否允许发送广播数据、是否要加入一个多播组等这样一些套接字的行为特性，均是由套接字选项的值决定的。
- 一般情况下，套接字选项的默认值能够满足大多数应用的需求，不必做任何修改，但有些时候为了使套接字能够满足某些需求，比如希望套接字能发送广播数据，必须对套接字的选项值作出更改。
- 更改或查看套接字的选项值分别使用函数getsockopt()和setsockopt()。

- 例如，要想使一个数据报套接字能够发送广播数据，必须要对其SOL_SOCKET选项级别下的SO_BROADCAST选项进行设置。该选项为布尔型选项，只对数据报套接字和原始套接字有效。
- 将一个数据报套接字设置为允许发送广播信息可使用如下所示的代码，其中sockDG是一个以创建好的数据包套接字。

```
BOOL yes=TRUE;
int ret=setsockopt(sockDG, SOL_SOCKET,
    SO_BROADCAST,  (char*)&yes, sizeof(BOOL));
```

广播程序的流程


发送端
初始化WinSock
创建socket
绑定本地IP地址和端口号
设置套接字广播功能
设置广播地址
发送广播数据
关闭socket
终止Winsock

流程说明

- 如果过该套接字除发送广播数据外还用于接收数据，则需要调用bind()函数绑定本地地址及端口号，如果不需要则该步骤可忽略。
- 调用setsockopt()函数设置数据报套接字的广播属性，使该套接字能发送广播数据；
- 广播地址是指广播信息发送的目的地址
- 如果目的地址为某一指定网络，则地址中的IP地址为一个直接广播地址，如果是本网络，IP地址则可以使有限广播地址255.255.255.255，编写程序时，有限广播地址可以用宏INADDR_BROADCAST表示。
- 除IP地址需要设为广播地址外，还必须指定接收者所使用的UDP端口号，该端口号必须与接收端用于接收广播数据的数据包套接字所绑定的端口号一直，否则接收端将接收不到广播数据。

- 下面的代码就是一个设置广播地址的例子。

```
SOCKADDR_IN   addrSrv;
addrSrv.sin_family = AF_INET;
addrSrv.sin_port = htons(56789);
addrSrv.sin_addr.S_un.S_addr
   =INADDR_BROADCAST;
```

- 接收端接收广播数据与接收普通数据的流程完全一样，只是需要注意，用于接收广播数据的套接字绑定的IP地址必须是INADDR_ANY或INADDR_BROADCAST，而不能是分配给本机的某个单播IP地址。

广播发送者代码

```c
#include <WinSock2.h>
#include <ws2tcpip.h>
#include <stdio.h>


#pragma comment(lib,"Ws2_32.lib")

//自定义的默认端口号,缓冲区长度
#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLEN 512

int main()
{
    WSADATA wsaData;
    int iResult;
    SOCKET connectLessSocket = INVALID_SOCKET;
    struct addrinfo* result = NULL, hints;
    const char* sendbuf = "This is a test"; //要发送的测试数据
    BOOL broadcastable = TRUE; //广播选项参数值

    //初始化套接字
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup执行失败，错误码：%d\n", iResult);
        return 1;
    }

    //解析服务器地址和端口号
    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_protocol = IPPROTO_UDP;
    iResult = getaddrinfo("255.255.255.255", DEFAULT_PORT, &hints, &result); //广播地址
    if (iResult != 0) {
        printf("getaddrinfo执行错误，错误码：%d\n", iResult);
        WSACleanup();
        return 1;
    }
```

```
    //创建数据报套接字
    connectLessSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (connectLessSocket == INVALID_SOCKET) {
        printf("socket执行失败,错误码:%d\n", WSAGetLastError());
        WSACleanup();
        return 1;
    }
    //设置套接字广播选项
    int ret = setsockopt(connectLessSocket, SOL_SOCKET, SO_BROADCAST, (char*)&broadcastable, sizeof(BOOL));

    //发送缓冲区中的测试数据报
    iResult = sendto(connectLessSocket, sendbuf, (int)strlen(sendbuf), 0, result->ai_addr, result->ai_addrlen);
    if (iResult == SOCKET_ERROR) {
        printf("发送失败,错误码:%d\n", WSAGetLastError());
        closesocket(connectLessSocket);
        WSACleanup();
        return 1;
    }
    freeaddrinfo(result);
    printf("发送: %s\n", sendbuf);
    printf("发送字节数:%ld\n", iResult);

    //关闭套接字
    closesocket(connectLessSocket);
    //释放资源
    WSACleanup();
    return 0;
}
```

广播接收者代码

```
#include <WinSock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include <iostream>

#pragma comment(lib,"Ws2_32.lib")

//自定义的默认端口号,缓冲区长度
#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLEN 512

int main()
{
    WSADATA wsaData; //用于Winsock初始化的数据结构
    int iResult; //执行结果返回值. 为0表示成功
    struct addrinfo* result = NULL, hints; //地址信息结构
    SOCKET receiveSocket = INVALID_SOCKET; //接收者套接字
    char recvbuf[DEFAULT_BUFLEN];  //接收数据缓冲区
    int recvbuflen = DEFAULT_BUFLEN; //接收数据缓冲区的长度
    sockaddr_in clientaddr;
    int clientlen = sizeof(sockaddr_in);
    BOOL broadcastable = TRUE; //广播选项参数值

    // 初始化套接字
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup执行失败，错误码：%d\n", iResult);
        return 1;
    }
```

```
//解析服务器地址和端口号
ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = IPPROTO_UDP;
hints.ai_flags = AI_PASSIVE; //当地址无法解析时就置为0.0.0.0，便于bind
iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result); //NULL，获得的地址等价于INADDR_ANY
if (iResult != 0) {
    printf("getaddrinfo执行错误，错误码：%d\n", iResult);
    WSACleanup();
    return 1;
}
//为无连接的服务器创建套接字
receiveSocket = socket(result->ai_family, (int)result->ai_socktype, result->ai_protocol);
if (receiveSocket == INVALID_SOCKET) {
    printf("socket执行错误，错误码:%d\n", WSAGetLastError());
    freeaddrinfo(result);
    WSACleanup();
    return 1;
}
//为无连接服务器套接字绑定地址和端口号
iResult = bind(receiveSocket, result->ai_addr, (int)result->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    printf("bind失败，错误码:%d\n", WSAGetLastError());
    freeaddrinfo(result);
    closesocket(receiveSocket);
    WSACleanup();
    return 1;
}
//设置套接字广播选项
int ret = setsockopt(receiveSocket, SOL_SOCKET, SO_BROADCAST, (char*)&broadcastable, sizeof(BOOL));
freeaddrinfo(result);
```

```
    //接收数据
    ZeroMemory(&clientaddr, sizeof(clientaddr));
    iResult = recvfrom(receiveSocket, recvbuf, recvbuflen, 0, (sockaddr*)&clientaddr, &clientlen);
    if (iResult > 0) {
        //成功收到数据
        std::string str(recvbuf, 0, iResult);
        std::cout << "接收到:" <<str <<std::endl;
        printf("接收字节数: %d", iResult);
    }
    else if (iResult == 0) {
        //连接关闭
        printf("连接关闭...");
    }
    else {
        //接收发生错误
        printf("recv失败，错误码:%d\n", WSAGetLastError());
        closesocket(receiveSocket);
        WSACleanup();
        return 1;
    }
    //关闭套接字，释放资源
    closesocket(receiveSocket);
    WSACleanup();
    return 0;
}
```

3. 多播程序设计

- 多播又称为组播，是一种同时向一组选定的目的地传输数据的通信方式，这种通信方式允许一台或多台主机（多播源）同时一次性地发送单一数据包到多台主机。
- 多播是节省网络带宽的有效方法之一，目前被广泛应用在网络音频/视频广播、AOD/VOD、网络视频会议、多媒体远程教育、信息分发（如股票行情等）和虚拟现实游戏等方面。
- 多播与广播不同，广播是向某个网络中的所有主机发送数据，而多播只向事先规划好的多播组(由多台主机组成)发送数据，而且多播组的成员可以是互联的不同网络上的主机。多播不局限于局域网，也可用于广域网。

多播地址

- IP采用D类地址来支持多播，一个D类IP地址用于标识一个IP多播组。
- 一个IP多播组的成员可以向全组发送数据信息，并且可以收到所有本组成员发送的数据信息。每个成员都可以随时离开这个多播组，也可以再次加入进来。
- D类IP地址范围在224.0.0.0到239.255.255.255之间，分为永久地址和临时地址两类。永久地址是为特殊用途而保留的，例如，224.0.0.1代表子网内的所有主机，224.0.0.2代表子网内的所有路由器。
- 临时组地址才是某个进程参与多播时使用的地址，进程要想使用某个多播地址参与多播，则必须在使用前先加入该多播组，一个进程可以要求其主机加入特定的组，它也能要求其主机脱离该组。

- WinSock提供了对多播通信的支持，因此你可以自己编程实现多播数据的发送。与广播一样，多播程序也需要使用数据报套接字。使用数据报套接字编写多播程序，主要解决多播组的加入与退出、多播数据的发送与接收等问题。
- 无论是发送多播数据还是接收多播数据，所使用的数据报套接字都必须要先加入到多播组，通信完成还需要退出多播组。
- 加入和离开一个多播组，需要使用setsockopt()函数对套接字的相关选项进行设置。

- 要使一个数据报套接字加入一个多播组，需要设置其IPPROTO_IP选项级别下的IP_ADD_MEMBERSHIP选项。要离开一个多播组，则需要设置其IPPROTO_IP选项级别下的IP_DROP_MEMBERSHIP选项。这两个选项都是只能设置不能读取，并且只对数据报套接字有效。
- 设置这两个选项的值需要用到一个ip_mreq结构变量，该结构定义如下：

```
struct ip_mreq{
    struct in_addr  imr_multiaddr; //多播组IP地址
    struct in_addr  imr_interface; //本机IP地址
};
```

  - 成员imr_multiaddr用于存放要加入或退出的多播组的地址，imr_interface存放用于发送或接收多播数据的一个本机接口的IP地址。

- 例如，将一个数据报套接字sockDG加入一个多播组可使用如下代码：

```
struct ip_mreq mq// 定义
mq.imr_multiaddr.S_un.Saddr=inet_addr("230.10.10.1");
                                                    //指定多播组
mq.imr_interface.S_un.Saddr=inet_addr("192.168.1.100");
                                                    //指定本机IP地址
setsockopt(sockDG,IPPROTO_IP,IP_ADD_MEMBERSHIP,
        (char *)&mq,sizeof(mq));
```

- 注意，要接收多播数据，套接字sockDG绑定（bind）的本机IP地址如果不是INADDR_ANY，则必须与结构变量mq中指定的本机IP地址相同。

- 要将套接字sockDG退出上面加入的多播组则需要使用如下代码：

```
setsockopt(sockDG,IPPROTO_IP,
   IP_DROP_MEMBERSHIP,(char
   *)&mq,sizeof(mq));
```

- 上面加入和退出多播组的方法通常在WinSock1中使用，如果你使用的平台支持WinSock2，则你还可以直接使用WSAJoinLeaf()函数加入一个多播组。

- WSAJoinLeaf()函数原型：

```
SOCKET WSAJoinLeaf(
    SOCKET s,
    const struct sockaddr FAR *name,
    int namelen,
    LPWSABUF lpCallerData,
    LPWSABUF lpCalleeData,
    LPQOS lpSQOS,
    LPQOS lpGQOS,
    DWORD dwFlags
);
```

- 参数：
  - s：代表一个套接字句柄，是由WSASocket()函数创建的，该套接字创建时必须使用恰当的多播标志进行创建；否则的话WSAJoinLeaf()函数就会失败，并返回错误WSAEINVAL。
  - name：SOCKADDR（套接字地址）结构，指定要加入的多播组。
  - Namelen：name参数的长度，以字节为单位。
  - lpCallerData：呼叫者数据，指向要传送给远端的用户数据。
  - lpCalleeData：被叫者数据，指向一个用来接收来自通信对方的数据缓冲区。注意在当前的Windows平台上，lpCallerData和lpCalleeData这两个参数并未真正实现，所以均应设为NULL。

  - LpSQOS和lpGQOS：这两个参数是有关Qos（服务质量）的设置，通常也设为NULL，有关Qos内容请参阅MSDN或有关书籍。
  - dwFlags：指出该主机是发送数据、接收数据或收发兼并。该参数可选值分别是：JL_SENDER_ONLY、JL_RECEIVER_ONLY或者JL_BOTH。
- 返回值：若无错误发生，WSAJoinLeaf()返回创建的多点套接口的描述符，其值就是参数s的值。否则的话，将返回INVALID_SOCKET错误，应用程序可通过WSAGetLastError()来获取相应的错误代码。

- WSAJoinLeaf()函数的第一个参数s指定的套接字就是要加入多播组的套接字，加入成功，函数将返回该套接字标识符。作为参数，该套接字必须WSASocket()函数创建并且创建时必须使用恰当的多播标志。

- WSASocket()是Winsock2提供的用于创建套接字的扩展套接字函数，其原型如下：

```
SOCKET WSAAPI WSASocket(
    int af,
    int type,
    int protocol,
    LPPROTOCOL_INFO lpProtocolInfo,
    Group g,
    int iFlags
);
```

- 前三个参数与socket()函数完全相同，后三个则是socket()函数没有的。

- lpProtocolInfo：一个指向PROTOCOL_INFO结构的指针，该结构定义所创建套接口的特性。如果本参数非零，则前三个参数（af, type, protocol）被忽略。
- g：套接口组的描述字，该参数始终为0，因为目前所有Winsock版本均不支持套接口组。
- iFlags：套接口属性描述。可用的值包括：WSA_FLAG_OVERLAPPED、WSA_FLAG_MULTIPOINT_C_LEAF 和 WSA_FLAG_MULTIPOINT_D_LEAF。
  - 其中WSA_FLAG_OVERLAPPED指明套接字具备重叠I/O特性，一般来说，在使用WSASocket()函数创建套接字时，建议设置该标识，使用socket函数创建的套接字默认具备重叠I/O特性。另外两个值用于指明套接字的多播特性。
  - C_LEAF是多点会话的控制平面的叶子节点
  - D_LEAF是多点会话的数据平面的叶子节点，详细

- 使用WSASocket()函数创建一个数据报套接字并指定其多播属性可使用如下代码：

```
SOCKET
    sock=WSASocket(AF_INET,SOCK_DGRAM,0,NULL,0,
    WSA_FLAG_MULTIPOINT_C_LEAF|WSA_FLAG_MUL
    TIPOINT_D_LEAF|WSA_FLAG_OVERLAPPED);
    if(sock==INVALID_SOCKET)
    {
      printf("socket failed with:%d\n",WSAGetLastError());
      WSACleanup();
      return -1;
    }
```

- 使用WSAJoinLeaf()函数将上面创建的套接字加入一个多播组的程序代码如下：

```
//定义并初始化多播地址
struct sockaddr_in  remote;
remote.sin_family = AF_INET;
remote.sin_port = htons(65432);//指定要绑定的端口
remote.sin_addr.s_addr = inet_addr("230.10.10.1");//指定多播组地址
   //加入多播组
sockM=WSAJoinLeaf(sock,(SOCKADDR*)&remote,sizeof(remote),NULL,
                                        NULL,NULL,NULL,JL_BOTH);
if(sockM == INVALID_SOCKET)
{
      printf("WSAJoinLeaf() failed:%d\n",WSAGetLastError());
      closesocket(sock);
      WSACleanup();
      return -1;
}
```

多播程序的流程

- 编写接收多播数据的程序必须要有以下几个步骤：

    1. 创建一个SOCK_DGRAM类型的Socket；
    2. 将此Socket绑定到本地的一个端口上；
    3. 将套接字加入多播组；
    4. 接收数据并处理；
    5. 关闭套接字。

- 发送多播数据的程序则需要有以下几个步骤：
  1. 创建一个SOCK_DGRAM类型的Socket；
  2. 将套接字加入多播组；
  3. 发送数据；
  4. 关闭套接字。
- 注意：接收程序必须要将套接字绑定到本地地址，如果套接字仅仅用来发送数据，可以不必绑定本地地址和端口号。一个绑定了本地地址的多播套接字既可以发送多播数据，也可以从绑定的接口上接收多播数据。

多播发送者代码

```
#include <WinSock2.h>
#include <ws2tcpip.h>
#include <stdio.h>


#pragma comment(lib,"Ws2_32.lib")

//自定义的默认端口号,缓冲区长度
#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLEN 512
#define GROUP_IP "230.10.10.1"

int main()
{
    WSADATA wsaData;
    int iResult;
    SOCKET connectLessSocket = INVALID_SOCKET;
    SOCKET socketM = INVALID_SOCKET; //用来多播通信的套接字
    struct addrinfo* result = NULL, * pMAddrInfo = NULL, hints; //地址信息结构
    const char* sendbuf = "This is a test"; //要发送的测试数据
    BOOL broadcastable = TRUE; //广播选项参数值

    //初始化套接字
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup执行失败，错误码：%d\n", iResult);
        return 1;
    }
```

```
//解析多播地址和端口号
ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = 0;
iResult = getaddrinfo(GROUP_IP, NULL, &hints, &result); //多播地址.随机端口
if (iResult != 0) {
    printf("getaddrinfo执行错误，错误码：%d\n", iResult);
    WSACleanup();
    return 1;
}
iResult = getaddrinfo(GROUP_IP, DEFAULT_PORT, &hints, &pMAddrInfo); //多播地址,有目的端口的
if (iResult != 0) {
    printf("getaddrinfo执行错误，错误码：%d\n", iResult);
    WSACleanup();
    return 1;
}

//创建数据报套接字,必须用WSASocket来创建
connectLessSocket = WSASocket(AF_INET, SOCK_DGRAM, 0,NULL,0, WSA_FLAG_MULTIPOINT_C_LEAF | WSA_FLAG_MULTIPOINT_D_LEAF);
if (connectLessSocket == INVALID_SOCKET) {
    printf("socket执行失败,错误码:%d\n", WSAGetLastError());
    WSACleanup();
    return 1;
}
//加入多播组
socketM = WSAJoinLeaf(connectLessSocket, (SOCKADDR*)result->ai_addr, result->ai_addrlen, NULL,  NULL, NULL, NULL, JL_BOTH);
if (socketM == INVALID_SOCKET)
{
    printf("WSAJoinLeaf() failed:%d\n", WSAGetLastError());
    closesocket(connectLessSocket);
    WSACleanup();
    return -1;
}
```

```
    //发送缓冲区中的测试数据报
    iResult = sendto(socketM, sendbuf, (int)strlen(sendbuf), 0, pMAddrInfo->ai_addr, pMAddrInfo->ai_addrlen);
    if (iResult == SOCKET_ERROR) {
        printf("发送失败,错误码:%d\n", WSAGetLastError());
        closesocket(connectLessSocket);
        closesocket(socketM);
        WSACleanup();
        return 1;
    }
    freeaddrinfo(pMAddrInfo);
    printf("发送: %s\n", sendbuf);
    printf("发送字节数:%ld\n", iResult);

    //关闭套接字
    closesocket(connectLessSocket);
    closesocket(socketM);
    //释放资源
    WSACleanup();
    return 0;
}
```

接收多播报文

```
#include <WinSock2.h>
#include <ws2tcpip.h>
#include <stdio.h>
#include <iostream>

#pragma comment(lib,"Ws2_32.lib")

//自定义的默认端口号,缓冲区长度
#define DEFAULT_PORT "27015"
#define DEFAULT_BUFLEN 512
#define GROUP_IP "230.10.10.1"

int main()
{
    WSADATA wsaData; //用于Winsock初始化的数据结构
    int iResult; //执行结果返回值. 为0表示成功
    struct addrinfo* result = NULL,*pMAddrInfo=NULL, hints; //地址信息结构
    SOCKET receiveSocket = INVALID_SOCKET; //接收者套接字
    SOCKET socketM = INVALID_SOCKET; //多播套接字
    char recvbuf[DEFAULT_BUFLEN];  //接收数据缓冲区
    int recvbuflen = DEFAULT_BUFLEN; //接收数据缓冲区的长度
    sockaddr_in clientaddr;
    int clientlen = sizeof(sockaddr_in);
    BOOL broadcastable = TRUE; //广播选项参数值

    // 初始化套接字
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup执行失败，错误码：%d\n", iResult);
        return 1;
    }
```

```
//解析服务器地址和端口号
ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_DGRAM;
hints.ai_protocol = IPPROTO_UDP;
hints.ai_flags = AI_PASSIVE; //当地址无法解析时就置为0.0.0.0，便于bind
iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result); //NULL，获得的地址等价于INADDR_ANY
if (iResult != 0) {
    printf("getaddrinfo执行错误，错误码：%d\n", iResult);
    WSACleanup();
    return 1;
}
iResult = getaddrinfo(GROUP_IP, DEFAULT_PORT, &hints, &pMAddrInfo); //多播地址
if (iResult != 0) {
    printf("getaddrinfo执行错误，错误码：%d\n", iResult);
    WSACleanup();
    return 1;
}
//为无连接的服务器创建套接字
receiveSocket = WSASocket(AF_INET, SOCK_DGRAM, 0, NULL, 0, WSA_FLAG_MULTIPOINT_C_LEAF | WSA_FLAG_MULTIPOINT_D_LEAF);
if (receiveSocket == INVALID_SOCKET) {
    printf("socket执行错误，错误码:%d\n", WSAGetLastError());
    freeaddrinfo(result);
    WSACleanup();
    return 1;
}
//为无连接服务器套接字绑定地址和端口号
iResult = bind(receiveSocket, result->ai_addr, (int)result->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    printf("bind失败，错误码:%d\n", WSAGetLastError());
    freeaddrinfo(result);
    closesocket(receiveSocket);
    WSACleanup();
    return 1;
}
//加入多播组
socketM = WSAJoinLeaf(receiveSocket, (SOCKADDR*)pMAddrInfo->ai_addr, pMAddrInfo->ai_addrlen, NULL, NULL, NULL, NULL, JL_BOTH);
if (socketM == INVALID_SOCKET)
{
    printf("WSAJoinLeaf() failed:%d\n", WSAGetLastError());
    closesocket(receiveSocket);
    WSACleanup();
    return -1;
}
freeaddrinfo(result);
freeaddrinfo(pMAddrInfo);
```

```
    //接收多播数据
    ZeroMemory(&clientaddr, sizeof(clientaddr));
    iResult = recvfrom(socketM, recvbuf, recvbuflen, 0, (sockaddr*)&clientaddr, &clientlen);
    if (iResult > 0) {
        //成功收到数据
        std::string str(recvbuf, 0, iResult);
        std::cout << "接收到:" << str << std::endl;
        printf("接收字节数: %d", iResult);
    }
    else if (iResult == 0) {
        //连接关闭
        printf("连接关闭...");
    }
    else {
        //接收发生错误
        printf("recv失败，错误码:%d\n", WSAGetLastError());
        closesocket(socketM);
        closesocket(receiveSocket);
        WSACleanup();
        return 1;
    }
    //关闭套接字，释放资源
    closesocket(socketM);
    closesocket(receiveSocket);
    WSACleanup();
    return 0;
}
```