网络编程

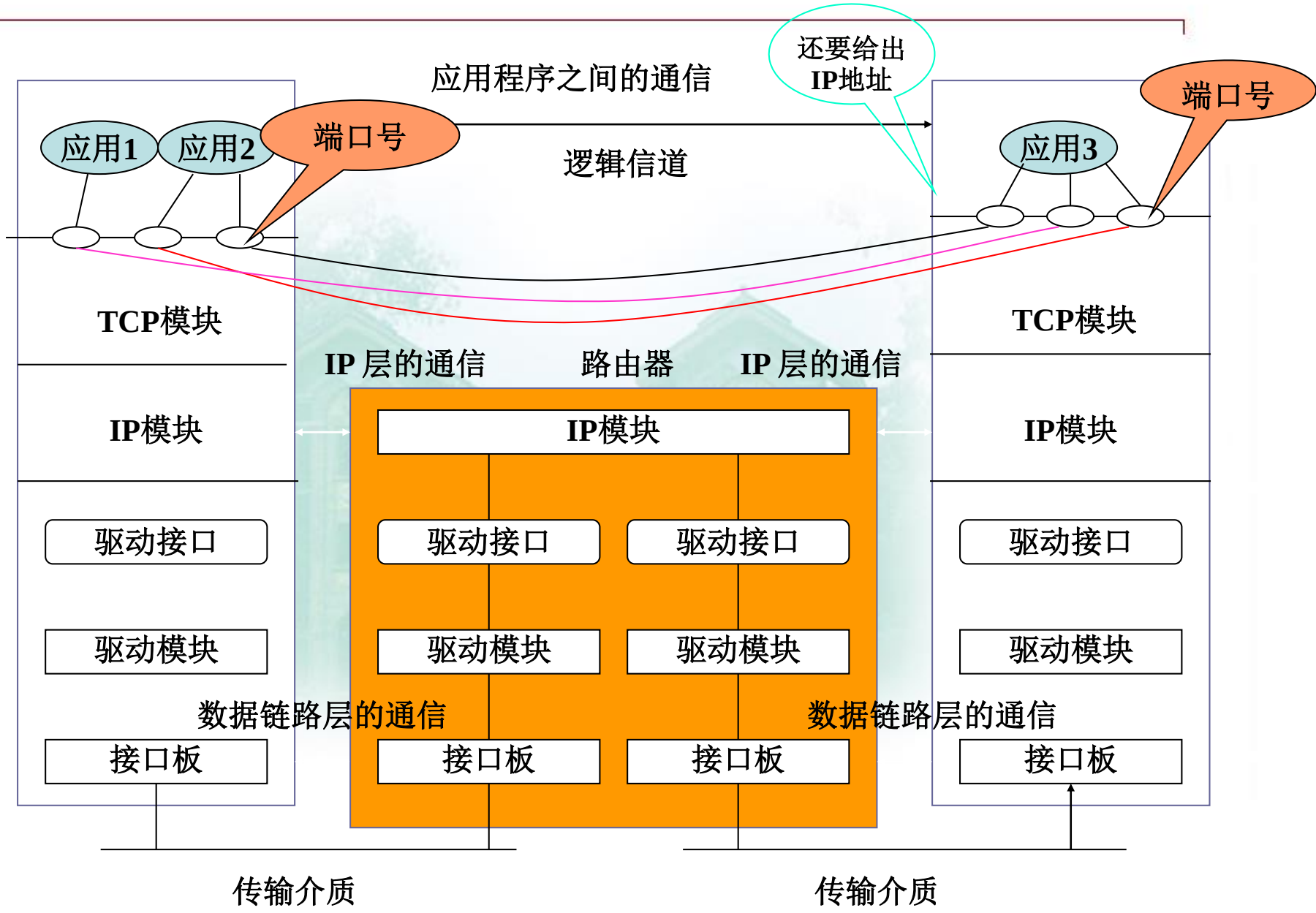
学习内容

- 套接字 (Socket)
- Socket编程原理
- 客户和服务器工作模式
- Socket的主要函数
- winsock通信模式
- 面向连接的服务器工作流程
- 面向连接的客户端工作流程
- TCP通信例
- UDP通信例

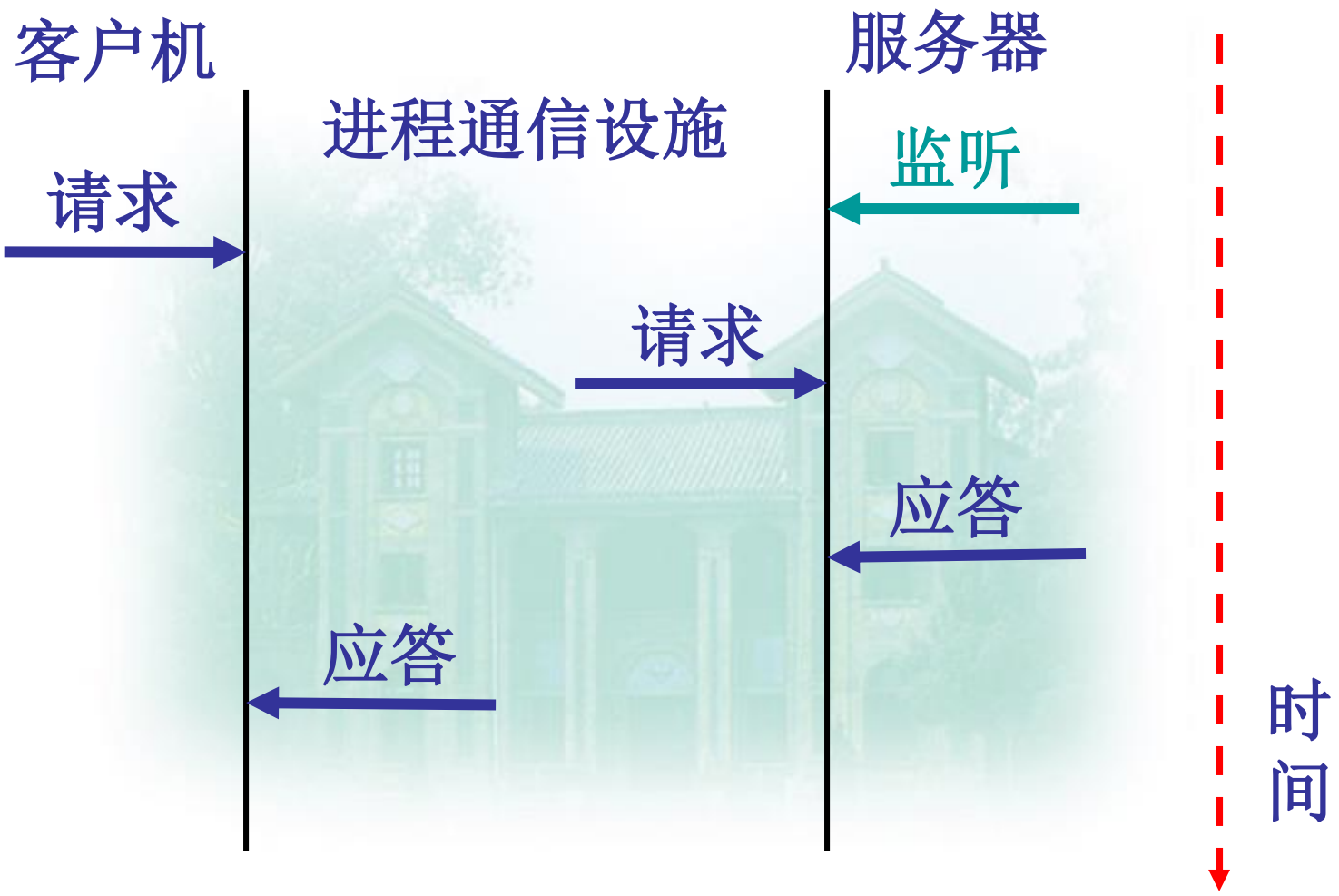
套接字 (Socket)

- 是一个网络应用编程接口
- 是一种通信协议
- 套接字是由传输层提供的应用程序(进程)和网络之间的接入点。应用程序(进程)可以通过套接字访问网络
- 套接字可以用于多种协议，包括面向连接的TCP协议和无连接的UDP协议
- 套接字利用主机的网络层地址和端口号为两个进程建立逻辑连接。 IP地址指定主机, 端口号指定应用程序(进程)
- 客户机可以通过端口号来访问服务器提供的服务
- 微软的socket称为Windows socket或Winsock

套接字使用图示


还要给出
IP地址
应用程序之间的通信
端口号
端口号
应用1
应用2
应用3
逻辑信道
TCP模块
TCP模块
IP 层的通信
路由器
IP 层的通信
IP模块
IP模块
IP模块
驱动接口
驱动接口
驱动接口
驱动接口
驱动模块
驱动模块
驱动模块
驱动模块
数据链路层的通信
数据链路层的通信
接口板
接口板
接口板
接口板
传输介质
传输介质

Socket编程原理：采用C/S结构



客户和服务器通过请求响应方式可以进行双向数据传输；
当结束数据传输时，需要关闭该连接；
这种工作模式是有连接的客户服/务器模式(Client/Server)

客户和服务器工作模式分类

- 有状态和无状态
  - 服务器是否记录客户的当前状态
- 有连接和无连接
  - 客户和服务器之间是否先建立连接再传输数据
- 循环和并发
  - 服务器对多客户请求的服务是采用循环方法还是并发程序方法

利用套接字建立逻辑信道

- 通信的一方(被动方，称为服务器)监听某个端口
- 通信的另一方(主动方，称为客户端) 试图对服务器端发送请求建立连接 。该连接请求包含:

  (服务器IP地址, 服务器端口号, 客户IP地址,客户端口号)
- 由于客户端口号由客户端的系统(TCP进程)自动选取一个当前未用的端口, 这个四元组便可以在因特网中唯一标识一个逻辑连接
- 如果服务器收到客户端来的连接请求后, 便发出响应建立该连接，这样就建立了一条逻辑信道

Winsock

如同文件描述符是对文件的抽象一般，套接字是对网络通信的抽象，是由操作系统创建并维护的一种数据结构。

对文件操作

```
CFile f = new CFile();
f.read();
f.write();
f.close();
f.open();
…..
```

对套接字操作

```
Socket s = creat();
s.read();
s.write();
s.close();
s.bind();
…..
```

套接字的数据结构

Family: PF_INET
Service: SOCK_STREAM
Local IP:
Remote IP:
Local Port:
Remote Port:

Family: PF_INET → TCP/IP协议族

Service: SOCK_STREAM → 服务类型
- TCP:SOCK_STREAM
- UDP:SOCK_DGRAM

套接字两种不同类型
- 流套接字：面向连接的TCP
- 数据报套接字：无连接的UDP

Socket使用的地址结构

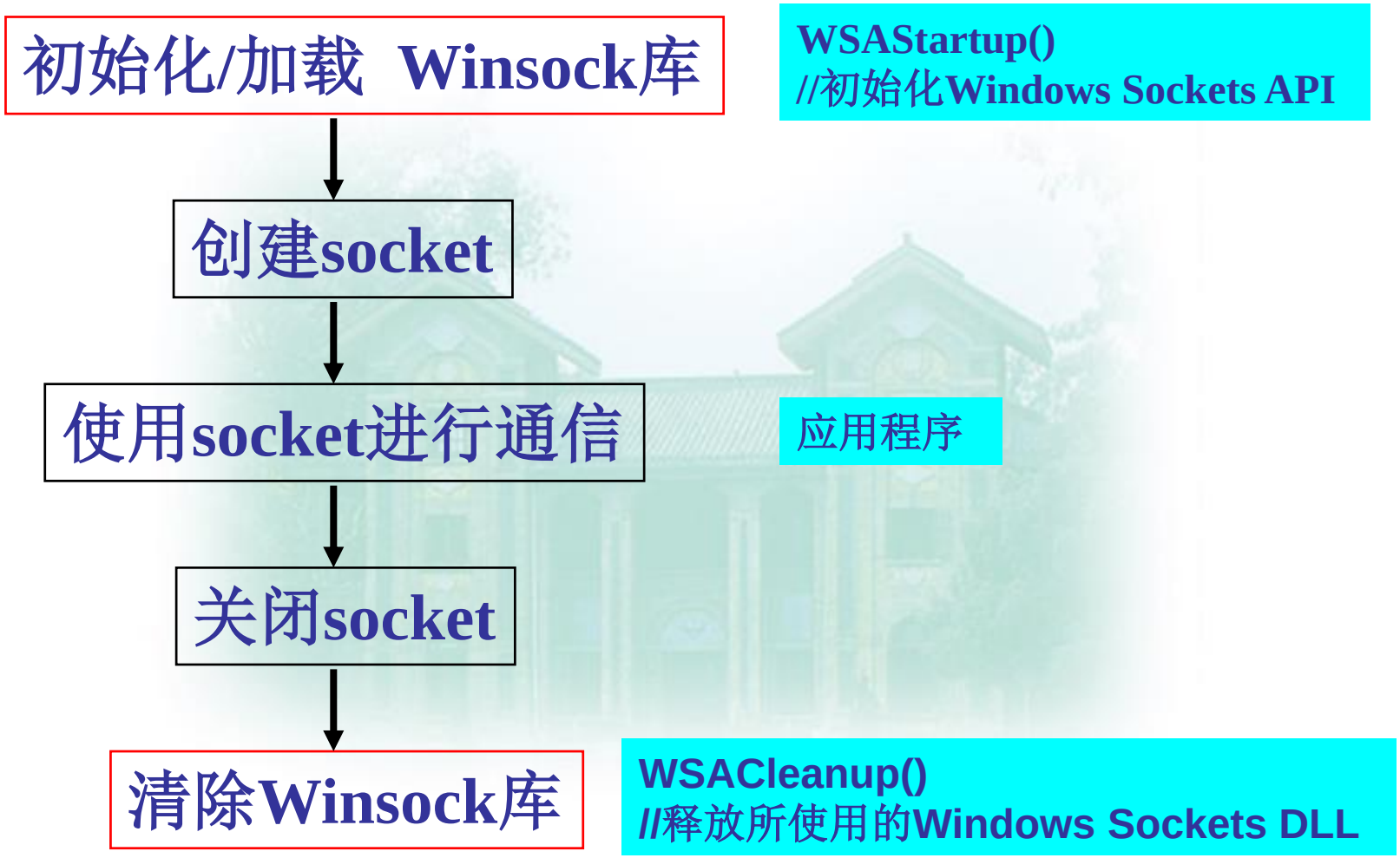
```
Struct  sockaddr_in
{
    short               sin_family;   //AF_INET地址族
    unsigned short      sin_port;     //使用的端口，2字节
    struct  in_addr     sinaddr;      //IP地址，4字节
    char                sinzero[8];   //预留，8字节
}

Struct sockaddr
{
    u_short     sa_family;    //地址族
    char        sa_data[14];  //地址
}
```

Socket的主要函数

- 创建用于网络通信的描述符(即套接字): socket()
- 将本地IP地址和协议端口号绑定到套接字: bind()
- 将套接字置入监听模式并准备接受连接请求:listen()
- 接受客户端连接的准备: accept()
- 连接远程对等实体（服务器）: connect()
- 接收下一个传入的数据报文: read()
- 发送外发的一个数据报文: write()
- 终止通信并释放套接字占用的资源: close()

winsock通信模式



winsock通信模式-初始化与结束

- 每个Winsock 应用都必须加载Winsock DLL的相应版本（有1.1、2.2等版本）。
- 加载Winsock 库是通过调用WSAStartup()函数实现的。
- 如果调用Winsock 之前，没有加载Winsock 库，这个函数就会返回一个SOCKET_ERROR，错误信息是WSANOTINITIALISED。
- 程序结束退出前，应该调用函数WSAcleanup(),释放对Winsock的使用。

winsock通信模式-错误检查和控制

- 如果调用一个Winsock 函数发生了错误，就可用WSAGetLastError函数就会返回所发生的特定错误的完整代码。
- WSAGetLastError 函数返回的这些错误都已预定义常量值，根据Winsock 版本的不同，这些值的声明不在Winsock.h中，就会在Winsock2.h中。
- WSAGetLastError函数定义如下：
  - int WSAGetLastError(void);

面向连接的服务器工作流程

（1）创建套接字

SOCKET socket(int domain, int type, int protocol)

参数: 协议簇, 套接字类型,协议簇中的协议号

返回: 新创建的套接字句柄(以下称此套接字为监听套接字)

domain: 协议簇

PF_INET表示因特网

PF_UNIX表示Unix管道功能

type: 套接字类型

SOCK_STREAM 表示基于连接的字节流方式(如TCP)

SOCK_DGRAM 表示无连接的数据报方式(如UDP)

Protocol: 协议簇中的协议号

可以说明为UNSPEC(unspectified).

domain和type已经可以指定一个传输层协议, 如,

TCP(domain=PF_INET, type=SOCK_STREAM)

UDP (domain=PF_INET, type=SOCK_ DGRAM)

面向连接的服务器工作流程

（2）将本地IP地址和端口号绑定到所创建的套接字上

int bind(SOCKET socket, struct sockaddr * address, int addr_len)

参数: 套接字, 本地地址, 地址长度

返回: 0 (无错时), 或错误码

*** 本地地址包括服务器端口号**

*** 有一些端口号已成为标准端口号, 如：80一般作为Web服务器的端口号**

*** 端口号也可以自己定义, 一般使用2000以上的端口号**

建立半连接，需要明确address中关于主机的部分

面向连接的服务器工作流程

（3）套接字的监听（服务器端）

要求系统（TCP进程）监听该套接字设置的端口，并为该端口建立客户请求连接等待队列。从客户来的连接请求将首先进入该等待队列，等待本进程的处理。

int listen(SOCKET socket, int backlog)

参数: 监听套接字(已绑定但未联接的套接字), 指定正在等待联接的最大队列长度

返回: 0 (无错时), 或错误码

- 例如：listen(s,1)表示连接请求队列长度为1,即只允许有一个请求,若有多个请求 (因为服务器一般可以提供多个连接),则出现错误,给出错误代码WSAECONNREFUSED

面向连接的服务器工作流程

（4）套接字等待连接,接受从客户端来的请求。若该端口的请求连接等待队列非空，则从请求连接等待队列中获得一个连接请求，若队列为空，则阻塞自己。

SOCKET accept(SOCKET socket, struct sockaddr * address, int *addr_len)

参数:处于监听模式的套接字,接收成功后返回客户端的网络地址(若接受一个连接请求，该地址中将包括客户的IP地址和端口号),网络地址长度

返回: 一个新的套接字(以下成为连接字), 或INVALID_SOCKET

accept()阻塞(缺省)等待请求队列中的请求

client也是一个sockaddr_in结构,连接建立时填入请求连接的套接口的半相关信息

面向连接的服务器工作流程

（5）发送和接收数据

- int send(SOCKET socket, char *message, int msg_len,int flags)
  参数: 连接套接字,缓冲区起始地址, 要发送字节数
  socket:服务器端监听已经连接的套接字。
  Message:指向待发送数据缓冲区的指针。
  msg_len:发送数据缓冲区的长度。
  Flags:数据发送标记,可为0、MSG_DONTROUTE或MSG_OOB
- 返回: 实际发送的字节数(无错时), 或SOCKET_ERROR
- 注意：send()并不保证发送所有请求的数据。它实际发送的字节数由返回值指示。也许需要循环调用send()来得到需要的结果。

简例:

```
#define BUFSIZE 4096
char buf[BUFSIZE];
int  left = BUFSIZE;
char* p = buf;
// 给buf填充4096字节的待发数据
// 假设s是已经连接的流式Socket
```

```
while (left > 0)
{
    ret = send(s, p, left, 0);
    if (ret == SOCKET_ERROR)
    {       // 错误处理
    }
    left -= ret;
    p += ret;
}
```

面向连接的服务器工作流程

- **int recv(SOCKET socket, char *message, int msg_len,int flags)**

  参数:连接套接字,缓冲区起始地址,缓冲区长度

  socket:准备接收数据的套接字;

  message:准备接收数据的缓冲区(用来存储所接收数据的字符串**);**

  msg_len:准备接收数据缓冲区的大小(字符串的长度减1，留下一个字节用于存放结束符**);**

  flags:数据接收标记,可为0、MSG_PEEK或MSG_OOB;

  - 0:最常用的参数值,它将信息移到指定的字符串，并从缓冲区清除。
  - **MSG_PEEK:**只查看数据而不将数据从缓冲区清除。
  - **MSG_OOB:**用于DECnet协议。

- **返回: 实际接收的字节数(无错时), 或SOCKET_ERROR**

面向连接的服务器工作流程

（6）关闭连接套接字,释放所占有的资源。

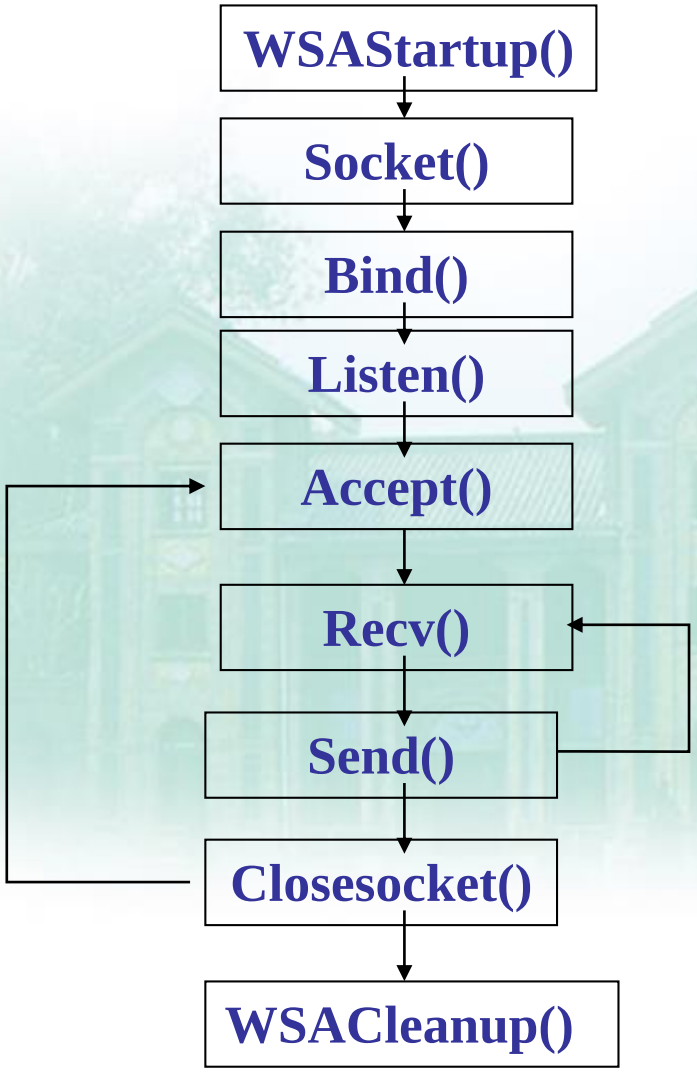
int closesocket (SOCKET socket);

- 参数:

socket：欲关闭的套接字。

（7）转（4）或结束。

Windows 服务器端流程(循环方式)



//WSAStartup用于加载WS2_32.DLL

面向连接的客户端工作流程

（1）创建套接字

SOCKET socket(int domain, int type, int protocol)

（2）发出连接请求

int connect(SOCKET socket, struct sockaddr * address, int addr_len)

参数: 套接字, 地址, 地址长度

返回: 0(无错), 或错误码

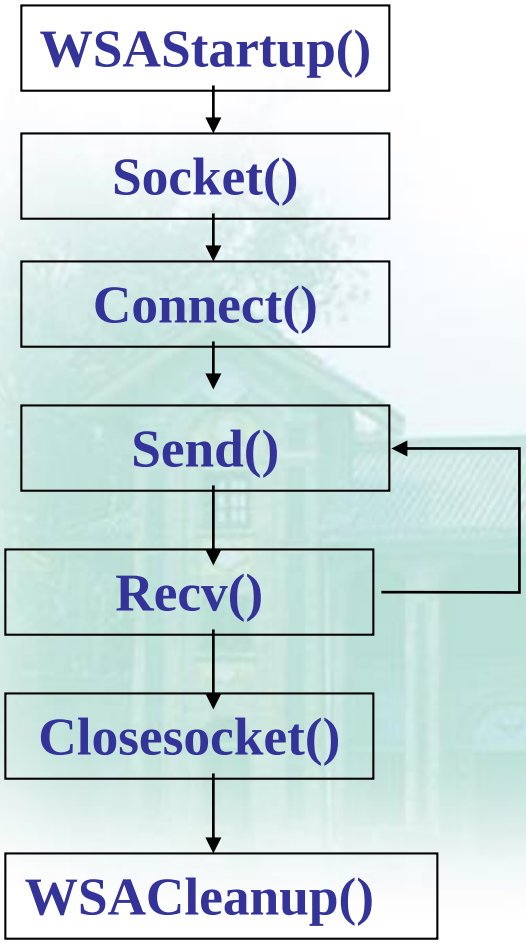
* 调用前, 参数‘地址’需要给出服务器的IP地址和端口号

* 系统自动获得客户端IP地址, 并产生一个客户端当前未使用的端口号.

（3）发送和接收数据

（4）关闭此连接的套接字

Windows 客户端端流程



* WSAStartup用于加载WS2_32.DLL

TCP通信例

TCP协议编程，包括两部分：

- TCP服务器端编程，例如server.c
- TCP客户端编程，例如client.c

TCP通信例

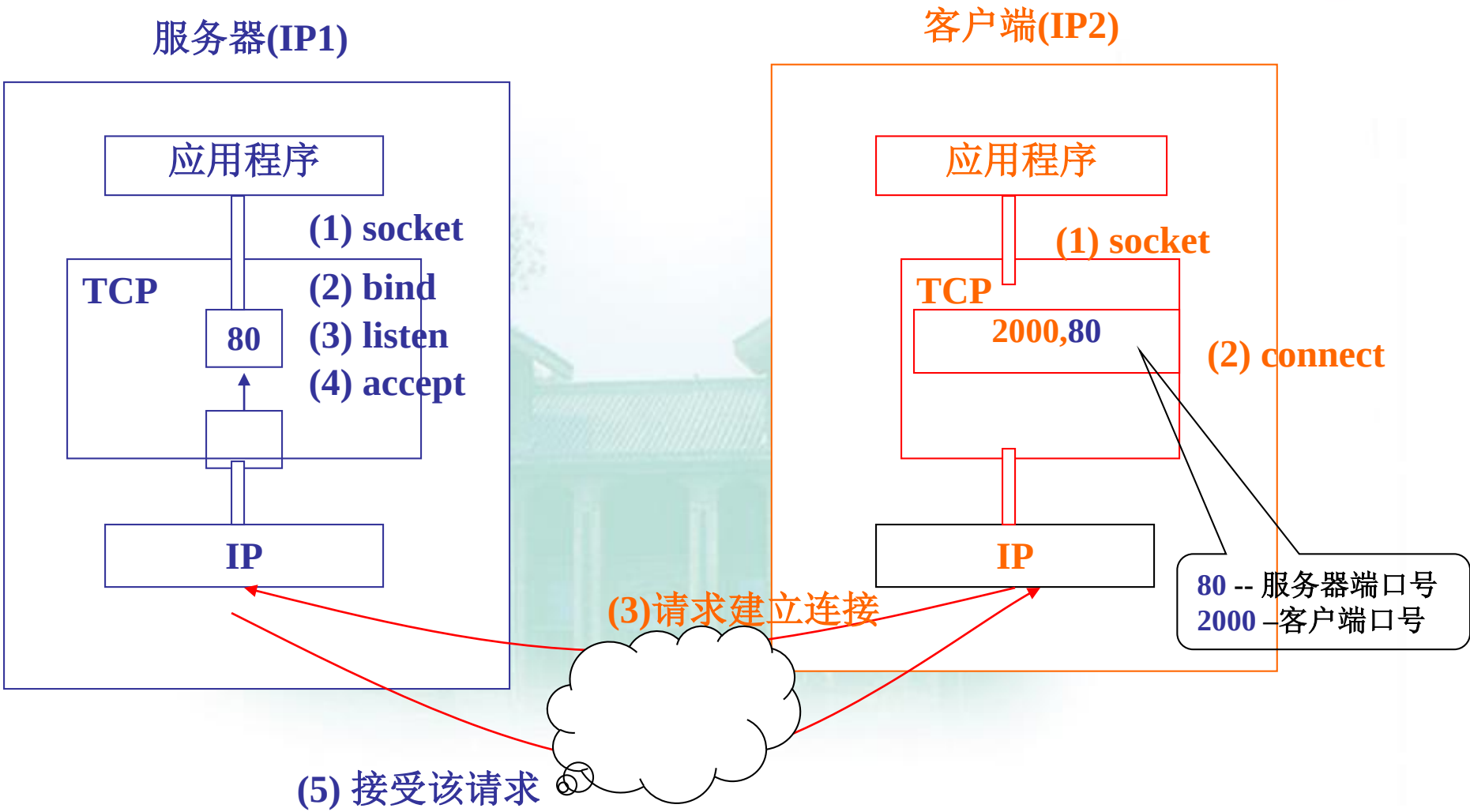
服务器端（先启动，并根据请示提供相应服务）：

- ■1 打开一通信通道并告知本地主机，它愿意在某一个公认地址上接收客户请求。
- ■2 等待客户请求到达该端口。
- ■3 接收到重复服务请求，处理该请求并发送应答信号。
- ■4 返回第2步，等待另一客户请求。
- ■5 关闭服务器。

客户端：

- ■1 打开一通信通道，并连接到服务器所在主机的特定端口。
- ■2 向服务器发送服务请求报文，等待并接收应答；继续提出请求。
- ■3 请求结束后关闭通信通道并终止。

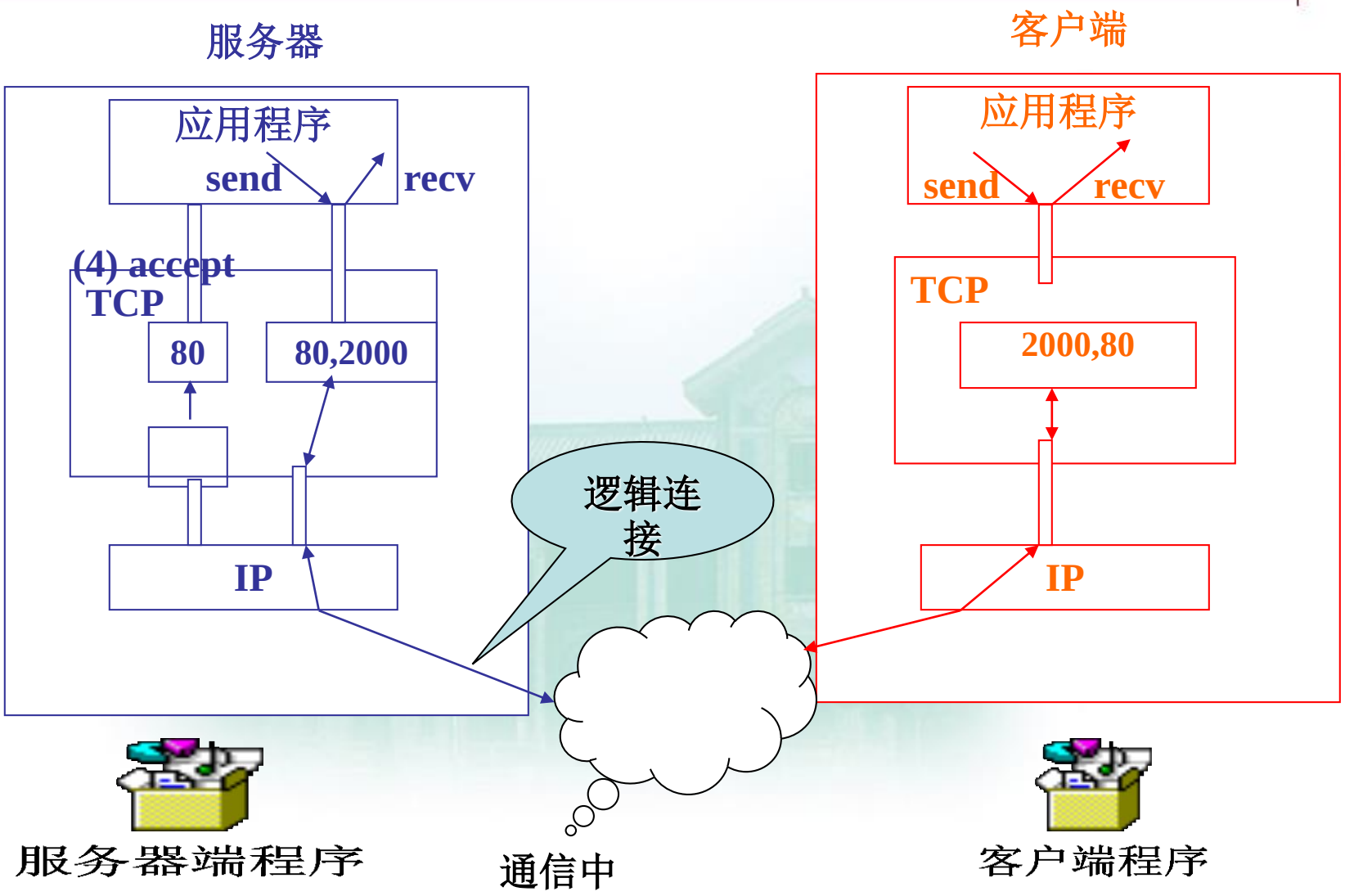
TCP客户服务器建立连接的过程



服务器绑定并监听端口；
客户端请求连接，服务器接受该请求

IP1-- 服务器IP地址
IP2-- 客户端IP地址

TCP客户服务器建立连接的过程



建立逻辑连接（源IP地址，源端口号，目标IP地址，目标端口号）

UDP通信

- UDP是典型的无连接协议，通信双方接收端可视之为服务器，发送端可认为客户端。
- 接收端
  - 用socket或WSASocket建立套接字；
  - 通过bind函数把这个套接字和准备接收数据的接口绑定在一起；
  - 和面向会话不同的是，不必调用listen和accept。相反，只需等待接收数据；
  - 由于它是无连接的，因此始发于网络上任何一台机器的数据报都可被接收端的套接字接收。
- 发送端
  - 要在一个无连接的套接字上发送数据，最简单的一种，便是建立一个套接字，然后调用sendto或WSASendTo。
- 释放套接字资源
  - 因为无连接协议没有连接，所以也不会有正式的关闭。在接收端或发送端结束收发数据时，只需在套接字句柄上调用closesocket函数，便释放了为套接字分配的所有相关资源。