

计算机网络编程

第2章 Socket编程基础知识

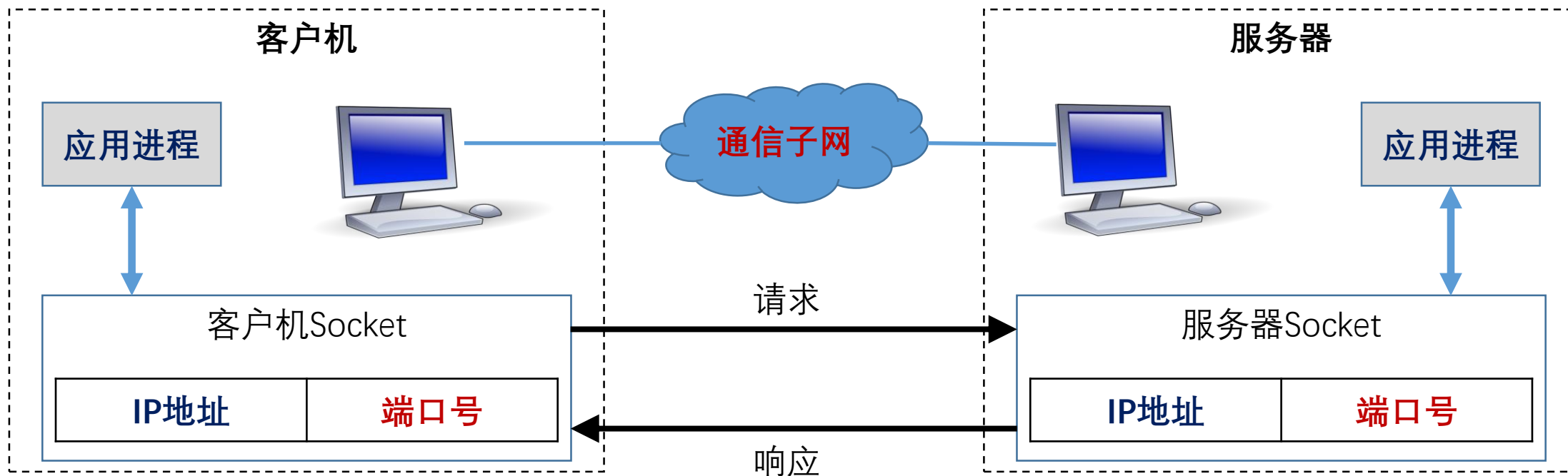
信息工程学院 方徽星
fanghuixing@hotmail.com

本章主要内容

- Socket编程的基本概念
- Winsock网络编程接口

2.1 Socket编程的基本概念

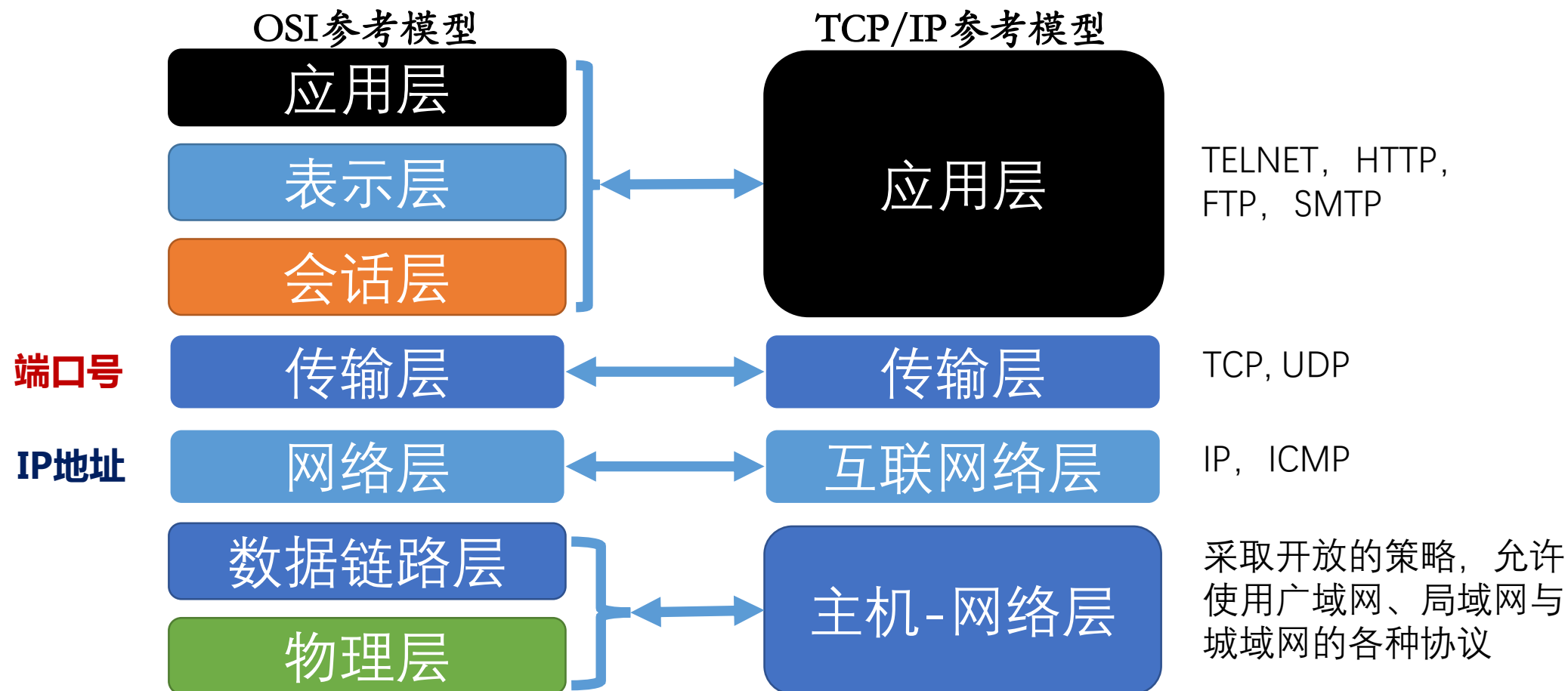
- **套接字 (Socket) : 网络层的IP地址 + 传输层的端口号**



客户机/服务器工作模式

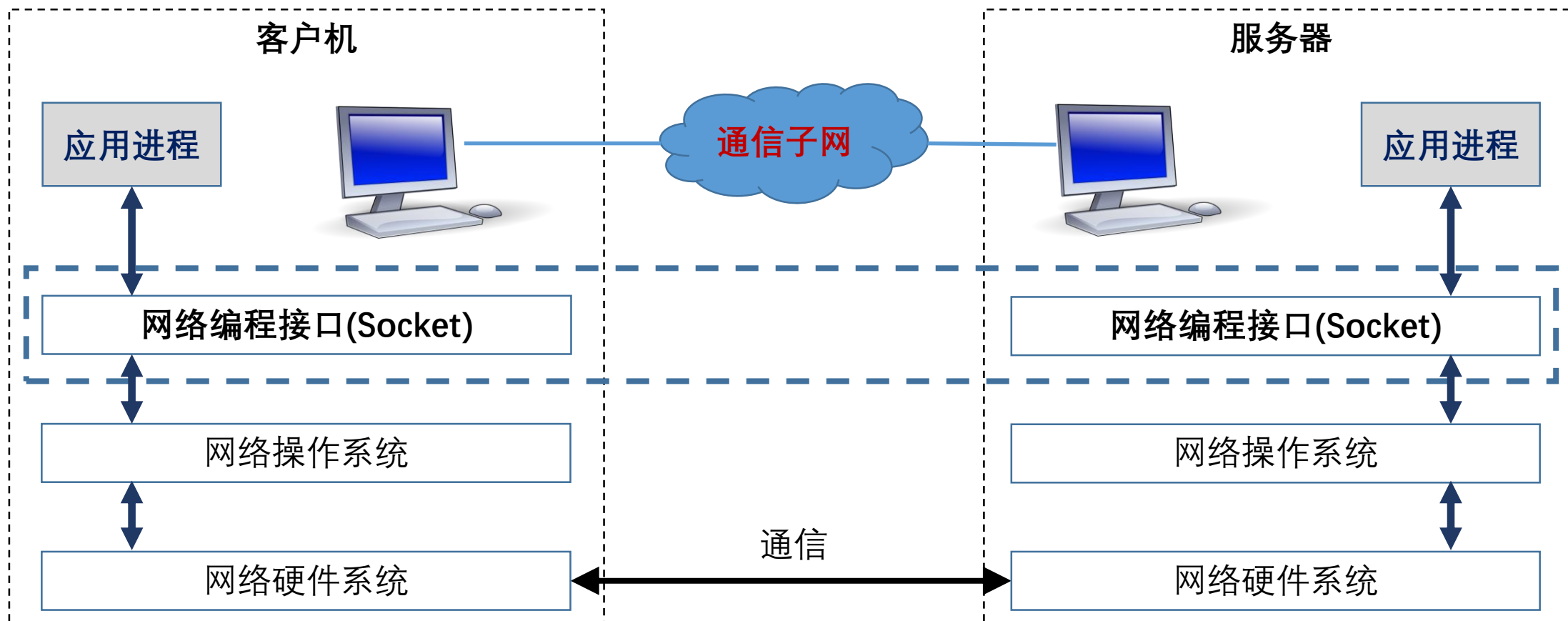
2.1 Socket编程的基本概念

- **套接字 (Socket) : 网络层的IP地址 + 传输层的端口号**



2.1 Socket编程的基本概念

- 网络环境中的分布式进程通信通常采用Socket编程方式



2.1 Socket编程的基本概念

- **Socket(网络编程接口) :**
 - 定义了很多用于网络通信的函数与数据结构
 - 程序员可以利用它们来开发TCP/IP网络中的应用程序
 - 支持不同的通信协议和套接字类型

2.1 Socket编程的基本概念

- **套接字类型**

1. **流式套接字**提供双向的、有序的、无重复的、无记录边界的数据流服务，主要用于TCP协议

- ✓双向的：允许数据在两个方向上传输

- ✓有序的：数据包按照发送顺序送达

- ✓无重复的：一个特定数据包只能获取一次

- ✓无记录边界的：数据以字节流的方式传送，需要应用层自己判断包的边界

2.1 Socket编程的基本概念

流式套接字的工作过程



2.1 Socket编程的基本概念

- **套接字类型**

2. **数据报套接字**提供双向的、无序的、可重复的、有记录边界的数据流服务，主要用于UDP协议

- ✓双向的：允许数据在两个方向上传输

- ✓无序的：不保证各数据包的发送顺序

- ✓可重复的：可能出现数据的重发

- ✓有记录边界的：会控制数据的记录边界，记录小于接收端的内部大小限制

2.1 Socket编程的基本概念

数据报套接字的工作过程



2.1 Socket编程的基本概念

- **套接字类型**

- 3. **原始套接字**允许对较低层的协议，如IP、ICMP等的直接访问

- ✓ 能够对网络数据包进行某种程度的控制操作
 - ✓ 通常用于开发简单的网络监视程序以及
 - ✓ 实现网络探测、网络攻击等工具

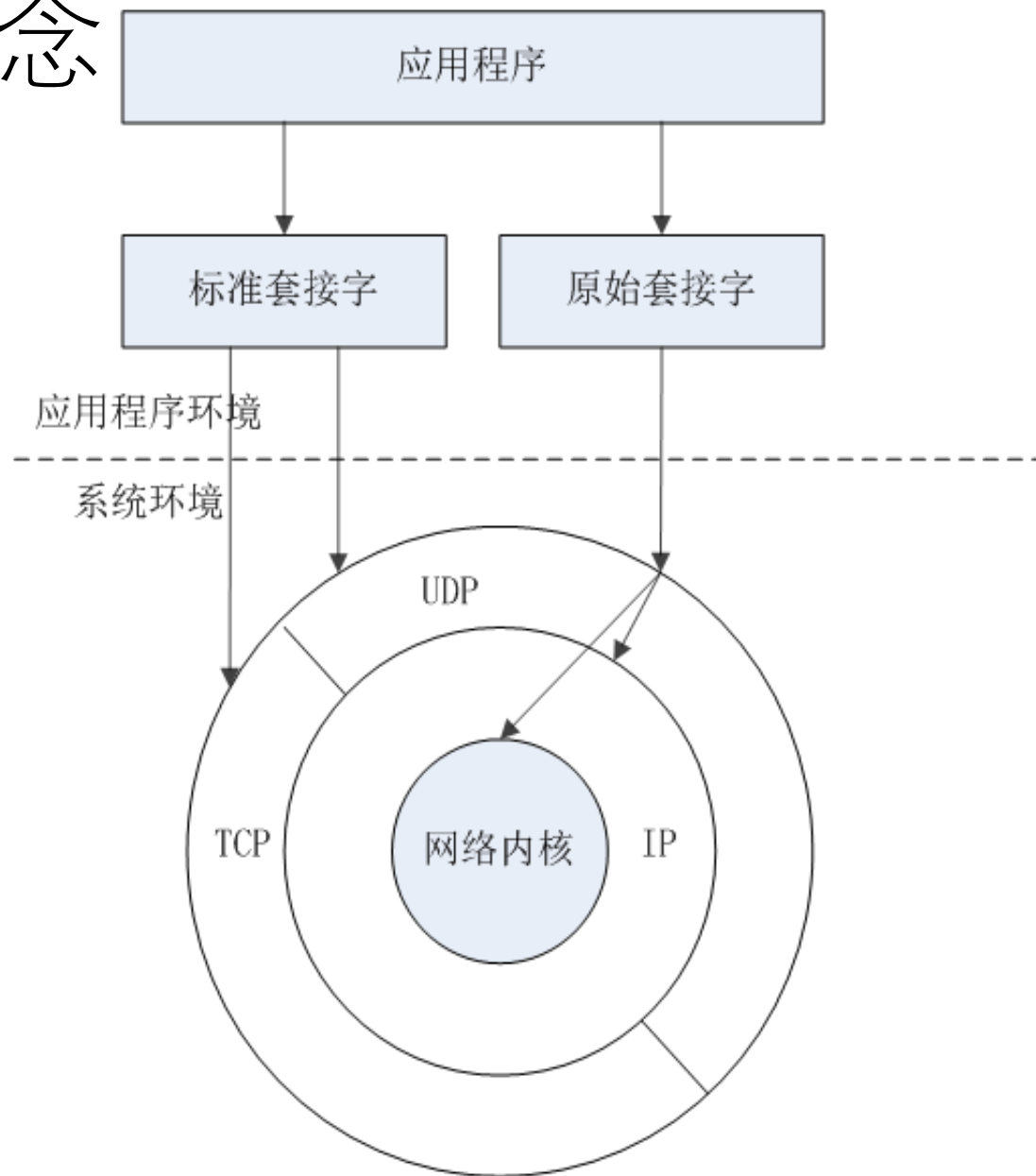
2.1 Socket编程的基本概念

- 普通的流式套接字和数据报套接字

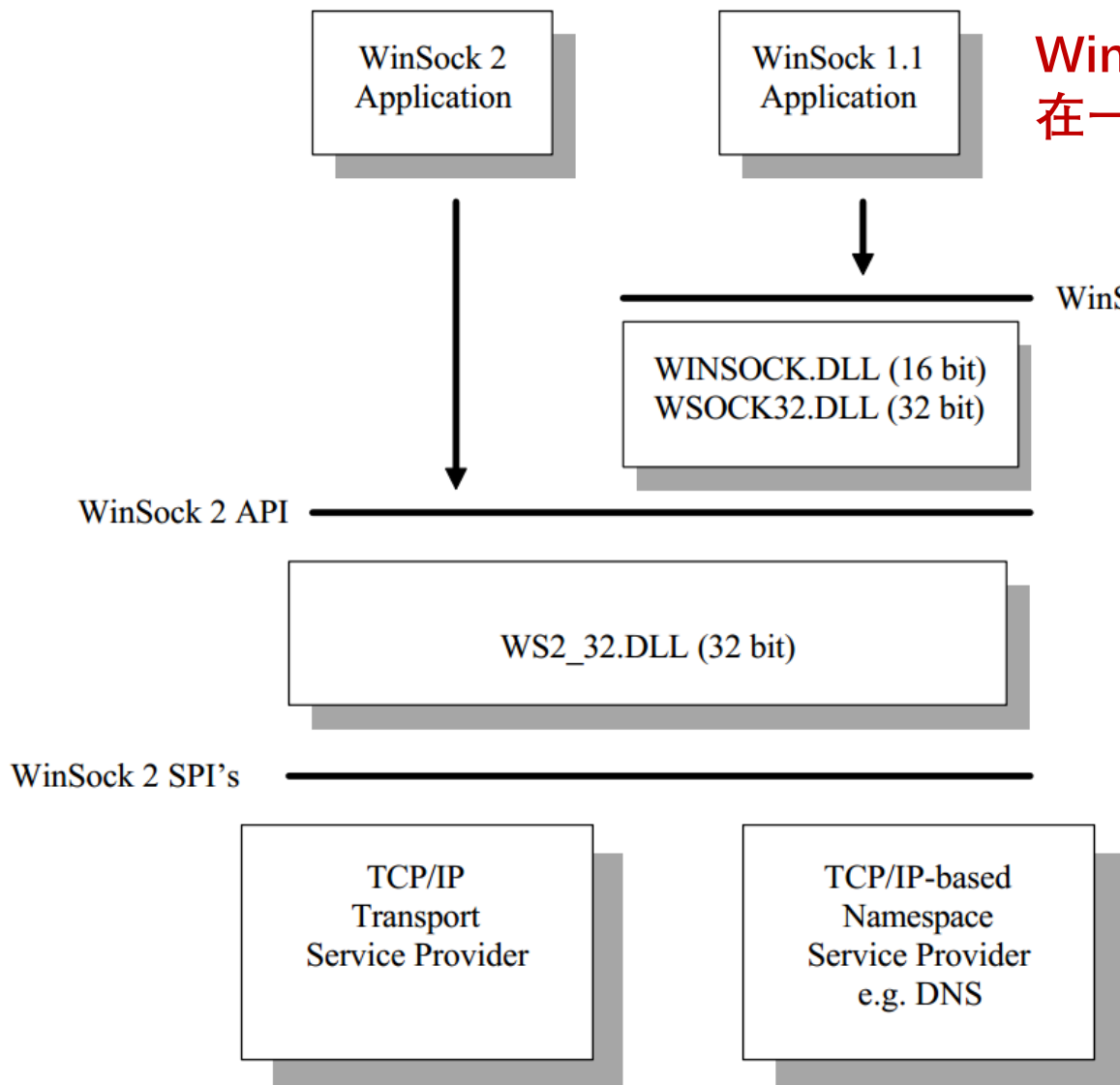
- ✓ 只能控制数据包的数据部分
- ✓ 传输层首部和网络层首部由协议栈根据创建套接字时指定的参数填充

- 原始套接字

- ✓ 可以控制传输层和网络层首部
- ✓ 给程序员提供了很大的灵活性
- ✓ 给网络安全带来了一定的安全隐患



2.2 Winsock网络编程接口

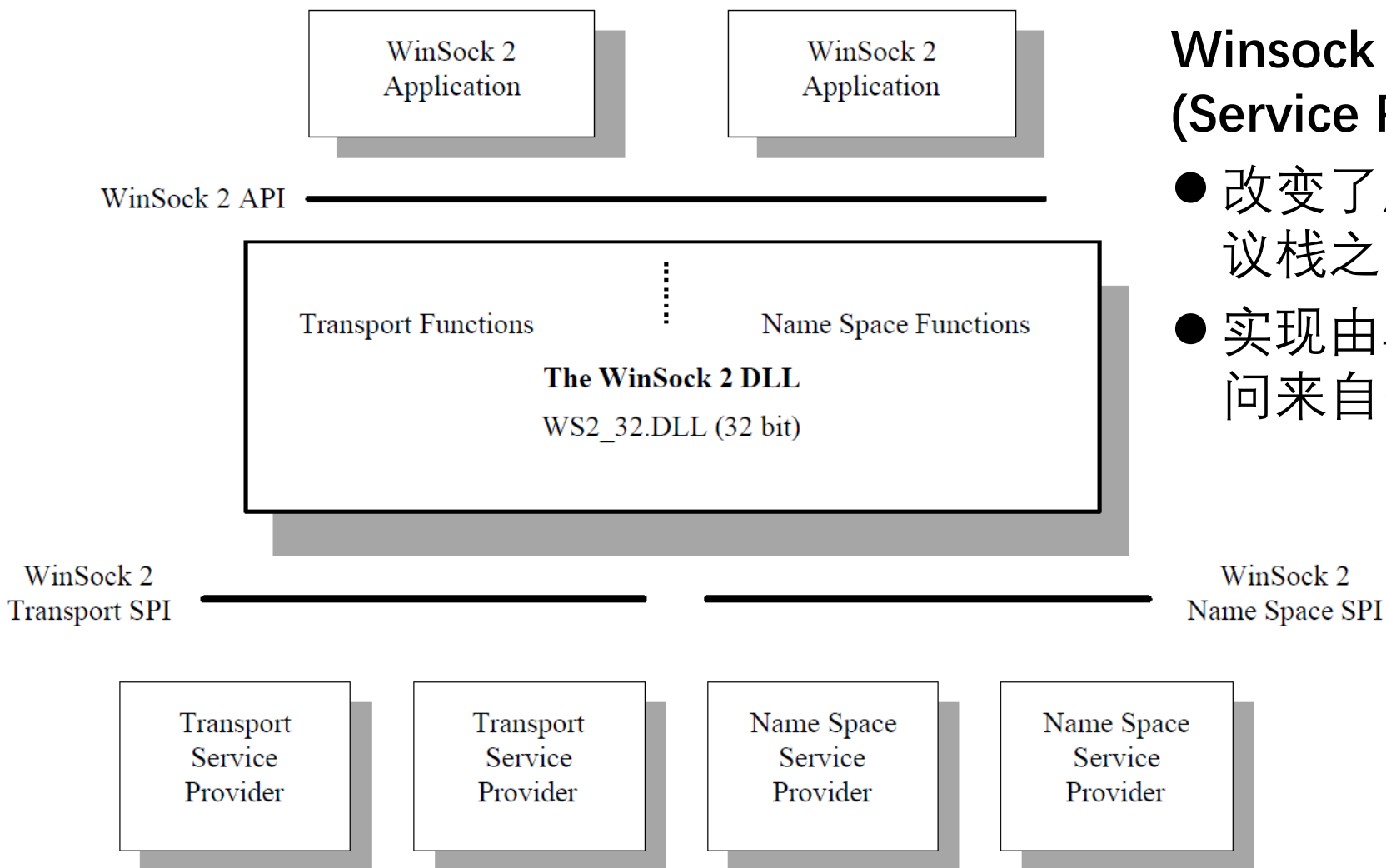


Winsock 1.1与TCP/IP协议族绑定在一起，仅支持TCP/IP协议族

Winsock是Windows下网络编程的规范

- Winsock向上，面向用户应用程序提供一个标准的API接口；
- Winsock(2)向下在Winsock组件和Winsock服务提供者（比如TCP/IP协议栈）之间提供标准的SPI接口

2.2 Winsock网络编程接口



Winsock 2引入服务提供者接口 (Service Provider Interface, SPI) :

- 改变了原Winsock 1.1和底层协议栈之间的私有接口模式
- 实现由单个Winsock DLL同时访问来自多个厂商的多个协议栈

2.2 Winsock网络编程接口

- Winsock提供两种I/O方式的函数：
 - **同步(也称阻塞)方式**：
 - 函数被调用后，在完成任务之前不会返回；
 - 在函数返回之前，不能进行其他操作，调用者进程处于挂起状态
 - **异步(也称非阻塞)方式**：函数被调用后，立即返回

2.2 Winsock网络编程接口

- Winsock异步I/O模型

1. 选择模型
2. 异步选择模型
3. 事件选择模型
4. 重叠I/O模型
5. 完成端口模型

2.2 Winsock网络编程接口

1. 选择模型（也称Select模型或I/O复用模型）：可以使Windows Sockets应用程序同时对多个套接字进行管理

- 调用Select()函数可以获取一组指定套接字的状态
- 可以保证及时捕捉到最先满足条件的I/O事件
- 实现对多个套接字上的不同网络事件进行及时处理

2.2 Winsock网络编程接口

2. 异步选择模型：为了适应Windows的消息驱动环境而设置的

- 以Windows系统中最常用的消息机制来反馈网络I/O事件达到收发数据的异步通知效果
- 通过调用WSAAsyncSelect函数自动将套接字设置（转变）为非阻塞模式，并向Windows注册一个或多个网络事件，并提供一个通知时使用的窗口句柄（可以理解为一个编号）
- 当注册的事件发生时，对应的窗口将收到一个**基于消息的通知**

2.2 Winsock网络编程接口

2. 异步选择模型：为了适应Windows的消息驱动环境而设置的

- 需要消息队列，通常消息队列依附于窗口实现
- 有些应用程序可能并不需要窗口，为了支持消息机制，就必须创建一个窗口来接收消息，对于特殊的应用场合并不适合
- 在一个窗口中处理大量的消息可能成为性能瓶颈

2.2 Winsock网络编程接口

3. 事件选择模型

- 以WSAEventSelect()函数为核心
- 与异步选择模型的最主要的区别是网络事件发生时系统通知应用程序的方式不同
- 允许在多个套接字上接收**以事件为基础**的网络事件的通知

2.2 Winsock网络编程接口

3. 事件选择模型

- 应用程序在创建套接字后，调用WSAEventSelect()函数将事件对象与网络事件集合相关联
- 当网络事件发生时，应用程序**以事件的形式**接收网络事件通知
- **优点**：不依赖于消息，可以在没有窗口的环境下比较简单地实现对网络通信的异步操作
- **缺点**：等待的事件对象的总数是有限制的

2.2 Winsock网络编程接口

4. 重叠I/O模型

- 基本设计原理是允许应用程序使用重叠数据结构，一次投递一个或多个异步 I/O请求
- 如果应用程序投递了一个10KB大小的缓冲区来接收数据，且数据已经到达套接字，则该数据将直接被拷贝到应用程序所投递的缓冲区
- 与前几种相比，重叠I/O的优势在于减少了一次从I/O缓冲区到应用程序缓冲区的拷贝

2.2 Winsock网络编程接口

5. 完成端口模型

- 提供了线程池管理，预先创建和维护线程，避免反复创建线程的开销
- 可以根据CPU数量，灵活决定线程个数，减少线程调度的次数，从而提高程序的并行处理能力
- 在处理多个并发异步I/O请求时，使用完成端口模型比在I/O请求时创建线程更快更有效

2.2 Winsock网络编程接口

5. 完成端口模型

- 完成端口可以看成系统维护的一个队列，操作系统把重叠I/O操作完成的事件通知放到该队列中
- 当某项I/O操作完成时，系统会向完成端口发送一个I/O完成数据包
- 应用程序收到数据包后，完成端口队列中的一个线程被唤醒为客户服务
- **服务完成后**，该线程会继续在完成端口上等待后续I/O请求事件的通知

2.2 Winsock网络编程接口

5. 完成端口模型

- 在Windows平台上比较成熟，伸缩性好
- 适用于管理上千个套接字
- 完成端口实际上是Windows I/O的一种结构，可以接收多种对象的句柄，除了对套接字对象进行管理之外，还可以应用于文件对象

2.2 Winsock网络编程接口

●初始化Winsock : WSAStartup()函数

- 在Winsock的DLL内部维持着一个计数器，首次调用WSAStartup()函数时装载DLL，以后调用时计数器+1

```
int WSAStartup(
```

```
WORD wVersionRequested,
```

```
LPWSADATA lpWSADATA
```

```
)
```

指定Socket版本

高位字节：副版本

低位字节：主版本

指向WSADATA数据结构
用于存放Socket版本信息

应用程序调用
WSAStartup()函数

操作系统

搜索Socket库

将应用程序
与Socket库绑定

应用程序调用
Socket库其他函数

2.2 Winsock网络编程接口

16为二进制

	<u>高位字节</u>	<u>低位字节</u>
	<u>01001010</u>	<u>00001111</u>
	high order byte	low order byte

Winsock 2.2 : 00000010 00000010


	<u>副版本</u>	<u>主版本</u>
--	------------	------------

2.2 Winsock网络编程接口

- 初始化Winsock : WSAStartup()函数

```
int main(int argc, char* argv[])
{
    WSADATA wsaData;
    ...
    if(WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
        ErrorHandling("WSAStartup() error!");
    ...
    return 0;
}
```

主版本号 副版本号



2.2 Winsock网络编程接口

● 卸载Winsock : WSACleanup()函数

- 功能与WSAStartup()相反，每调用一次将计数器减1
- 当计数器减到0时，将DLL从内存中卸载
- 调用WSACleanup()与调用WSAStartup()的次数应该相同

```
int WSACleanup()
```

版本	头文件	静态链接库文件	动态链接库文件
Winsock 2	winsock2.h	ws2_32.lib	ws2_32.dll

应用程序调用
WSACleanup()函数



解除应用程序
与Socket库的绑定



释放Socket库占用的
系统资源

2.2 Winsock网络编程接口

- 创建套接字：socket()函数

type	解释
SOCK_STREAM	流式套接字
SOCK_DGRAM	数据报套接字
SOCK_RAW	原始套接字

Socket类型



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

地址类型

address family

af	解释
AF_INET	IPv4 地址族
AF_INET6	IPv6 地址族



协议类型

protocol	解释
IPPROTO_TCP	TCP协议
IPPROTO_UDP	UDP协议
IPPROTO_ICMP	ICMP



2.2 Winsock网络编程接口

- **创建套接字** : `socket()`函数

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



套接字描述符
整数类型的数值

每个进程空间中都有一个套接字描述符表

套接字描述符  套接字数据结构

2.2 Winsock网络编程接口

- 关闭套接字：**closesocket()**函数

```
int closesocket( SOCKET s )
```



指定要关闭的
套接字描述符

2.2 Winsock网络编程接口

●绑定套接字：bind()函数

- 应用程序创建套接字后，套接字结构中会有一个默认IP地址和端口号
- 服务器和客户端程序都需要绑定本地地址到套接字上

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

指定要绑定的套接字描述符

指定sockaddr结构的套接字地址

指定套接字地址结构的长度(字节数)

2.2 Winsock网络编程接口

● 监听端口连接请求：listen()函数

- 专门为流式套接字设计，用于有连接的TCP服务
- 服务器端程序调用listen()函数后，流式套接字处于监听状态
- 分配监听队列

```
int listen( SOCKET s,  
           int backlog  
)
```

指定要监听的
套接字描述符

指定套接字要维护的
客户连接队列大小

2.2 Winsock网络编程接口

●连接请求：connect()函数

- 请求与服务器建立连接，流式套接字，TCP
- 客户端调用connect()函数向服务器端Socket发出建立建立连接请求

```
int connect(SOCKET s,  
            const sockaddr * name,  
            int namelen  
            )
```

客户端的
套接字描述符

服务器端的
套接字地址结构

套接字地址结构
的长度(字节数)

2.2 Winsock网络编程接口

●响应连接请求：accept()函数

- 响应连接建立请求，流式套接字，TCP
- 服务器端调用accept()函数，从处于监听状态的流式套接字的客户连接请求队列中取出排在最前面的一个客户请求，并创建一个新的套接字来与客户端套接字建立连接

```
SOCKET accept(SOCKET s,  
              sockaddr * addr,  
              int addrlen  
)
```

要监听的套接字描述符

新创建的套接字地址结构

套接字地址结构的长度(字节数)

2.2 Winsock网络编程接口

●发送数据：send()和sendto()函数

➤send() 函数，流式套接字，TCP

➤sendto() 函数，数据报套接字，UDP

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

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

s: 发送端套接字描述符

buf: 发送端等待发送数据的缓冲区

len: 待发送数据的字节数

flags: 0表示默认;

MSG_DONTROUTE表示不经过本地路由器;

MSG_OOB表示发送带外数据

to: 接收端套接字地址结构

tolen: 接收端套接字地址结构字节数

2.2 Winsock网络编程接口

●接收数据：recv()和recvfrom()函数

➤recv() 函数，流式套接字，TCP

➤recvfrom() 函数，数据报套接字，UDP

```
int recv( SOCKET s, const char * buf, int len, int flags);
```

```
int recvfrom( SOCKET s, const char * buf, int len, int flags,  
             sockaddr * from, int fromlen);
```

s: 接收端套接字描述符

buf: 接收端等待接收数据的缓冲区

len: 缓冲区字节数大小

from: 发送端套接字地址结构

fromlen: 发送端套接字地址结构字节数

flags: MSG_PEEK数据将被复制到缓冲区中，但并不从输入队列中删除;

MSG_WAITALL等待所有数据、连接关闭或则发送错误时返回;

MSG_OOB表示接收带外数据;
默认0

2.2 Winsock网络编程接口

●读取Socket属性：getsockopt()函数

```
int getsockopt(SOCKET s,  
              int level,          套接字级别，如IP协议是IPPROTO_IP  
              int optname,       属性名称，如SO_TYPE，表示Socket类型(SOCK_STREAM等)  
              char * optval,     存放属性值的缓冲区指针  
              int * optlen       缓冲区长度  
              )
```

2.2 Winsock网络编程接口

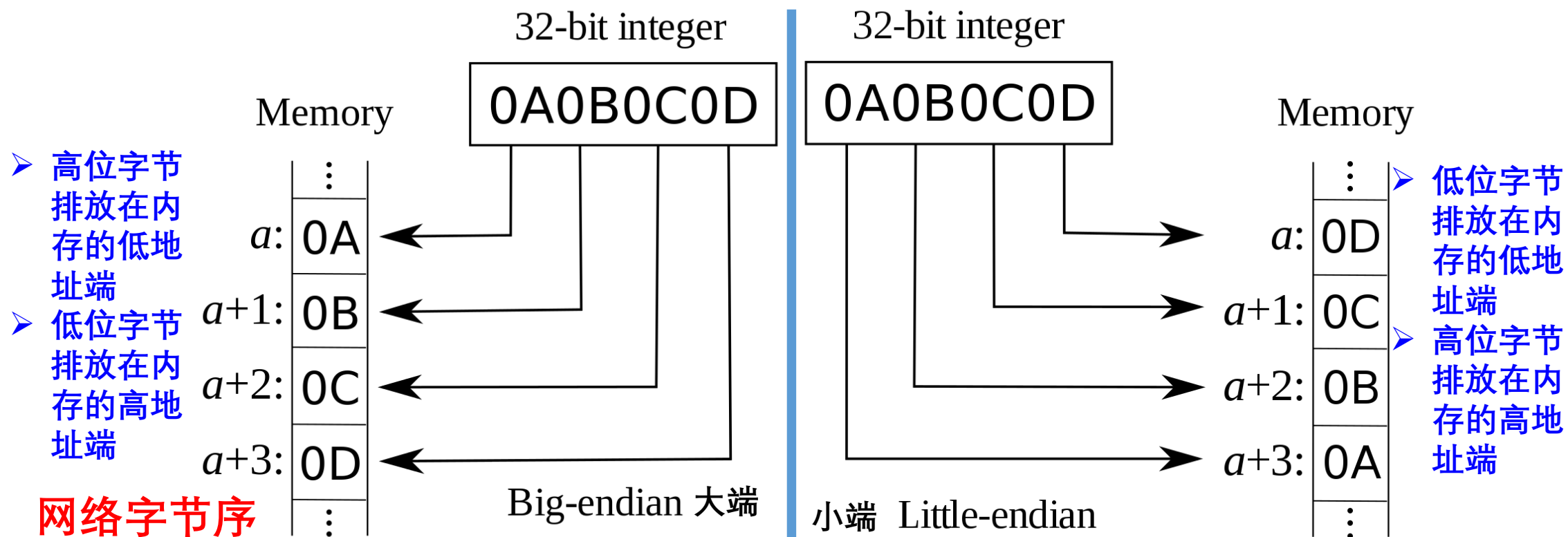
●设置Socket属性：setsockopt()函数

```
int getsockopt(SOCKET s,  
              int level,          套接字级别，如IP协议是IPPROTO_IP  
              int optname,       属性名称，如SO_TYPE，表示Socket类型(SOCK_STREAM等)  
              const char * optval, 存放属性值的缓冲区指针  
              int * optlen       缓冲区长度  
              )  
              const char *ptr; 定义一个指向字符常量的指针，*ptr是常量，ptr不是常量
```


2.2 Winsock网络编程接口

●字节序转换函数

字节序：存放多字节数据的字节（byte）顺序



2.2 Winsock网络编程接口

●字节序转换函数

1. unsigned long **htonl**(unsigned long hostlong) : 将无符号长整型数从**主机字节序**转换为**网络字节序**
2. unsigned long **ntohl**(unsigned long netlong) : 将无符号长整型数从**网络字节序**转换为**主机字节**
3. unsigned short **htons**(unsigned short hostlong) : 将无符号短整型数从**主机字节序**转换为**网络字节序**
4. unsigned short **ntohs**(unsigned short netlong) : 将无符号短整型数从**网络字节序**转换为**主机字节**

2.2 Winsock网络编程接口

●辅助性函数

1. `unsigned long inet_addr(const char *cp)` : 将点分十进制 IP地址转换为无符号长整型 (网络字节序)
2. `char * inet_ntoa(in_addr in)` : 将`in_addr`结构IP地址转换为点分十进制IP地址, 参数`in`为网络字节序
3. `int gethostname(char *name, int namelen)` : 获取主机名, `name`是存放主机名的缓冲区, `namelen`是缓冲区大小
4. `hostent * gethostbyname(const char *name)` : 根据主机名获取主机信息

2.2 Winsock网络编程接口

●GetLastError()函数

- 如果Socket函数调用后返回SOCKET_ERROR, 则说明出现错误
- 调用GetLastError()可以获得该错误对应的类型码
- *Winerror.h*

返回值	描述
WSAEACCES 10013	执行权限不支持的套接字操作
WSAEFAULT 10014	非法的指针地址
WSAEINVAL 10022	无效参数
WSAEMFILE 10024	打开的套接字太多

2.2 Winsock网络编程接口

●套接字地址结构

通用的Socket
地址结构

```
struct sockaddr {  
    ushort sa_family; ← 地址类型  
    char sa_data[14];  
};
```

TCP/IP协议族
地址结构

```
struct sockaddr_in {  
    short sin_family;  
    u_short sin_port; ← 端口号 (使用网络字节顺序)  
    struct in_addr sin_addr; ← IP地址, in_addr结构  
    char sin_zero[8]; ← 让sockaddr与sockaddr_in保  
}; 持大小相同而保留的空字节
```

2.2 Winsock网络编程接口

●hostent结构：表示主机信息

```
typedef struct hostent {  
    char *h_name;           //主机名  
    char **h_aliases;      //别名  
    short h_addrtype;      //主机地址类型  
    short h_length;        //地址长度  
    char **h_addr_list;    //地址列表(地址是网络字节序的)  
} HOSTENT, *PHOSTENT, *LPHOSTENT;
```

2.2 Winsock网络编程接口

- **protoent结构：表示协议信息**

```
typedef struct protoent {  
    char *p_name;           //协议名  
    char **p_aliases;      //协议别名列表  
    short p_proto;         //协议使用的端口号(主机字节序)  
} PROTOENT, *PPROTOENT, *LPPROTOENT;
```

2.2 Winsock网络编程接口

●常用协议类型定义

- `#define IPPROTO_IP 0`
- `#define IPPROTO_ICMP 1`
- `#define IPPROTO_IGMP 2`
- `#define IPPROTO_TCP 6`
- `#define IPPROTO_UDP 17`
- `#define IPPROTO_RAW 255`

本章小结

- Socket编程的基本概念
- Winsock网络编程接口
- https://docs.microsoft.com/en-us/windows/desktop/api/_winsock/