网络编程套接字

IP地址

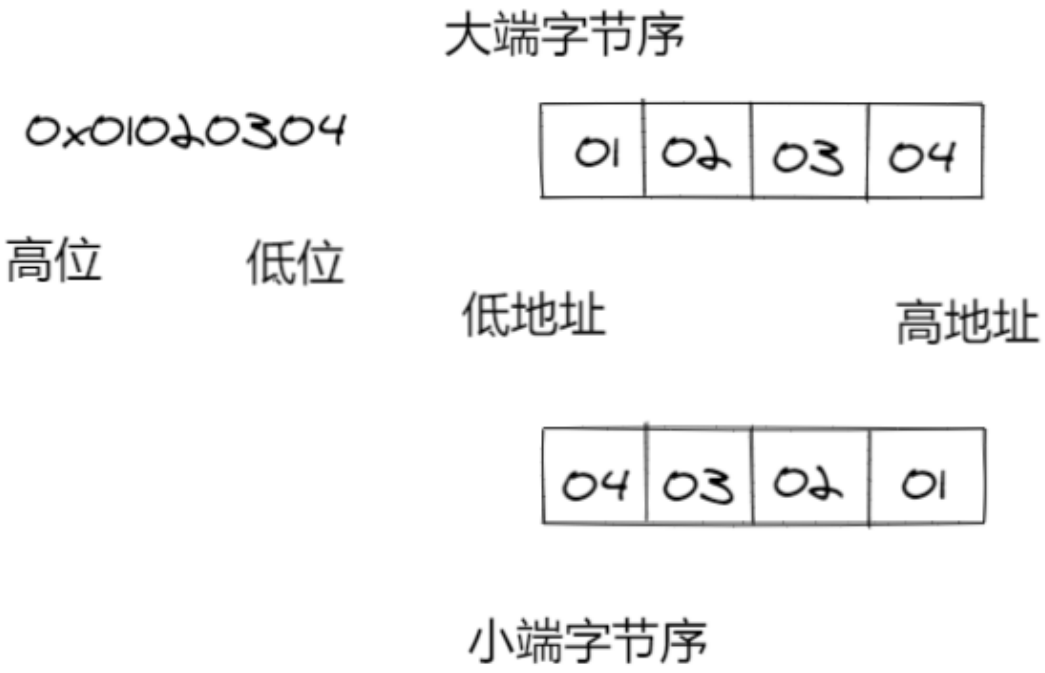
- 作用：在网络当中唯一标识一台主机
- 本质：无符号32位整型，4个字节
  - 192.168.113.192——>点分十进制表示法
  - 点分法每一个字节最大能够表示的数字是255
- 源IP地址和目的IP地址
  - 网络通信当中，每一条数据都是需要具备5个信息，我们成为5元组(源IP+源端口+目的IP+目的端口+protocol)
  - 源IP：src_ip——>标识数据从哪里来
  - 目的IP：dest_ip——>表示数据往哪里去
  - 源端口：src_port
  - 目的端口：dest_port
- ipv6
  - ipv4和ipv4指的是不同版本的IP协议
  - 并且ipv6并不向下兼容ipv4
  - ipv6——>16字节整数

port

- 作用：端口是在一台主机当中标识一个进程
- 本质：无符号16位整数，范围0~2^16(65535)；
  - 其中0~1023是知名端口
  - mysql——侦听端口3306
  - oracle——侦听端口1521
- 使用
  - 网络当中的程序，通信的时候，都是需要使用端口进行通信的
  - 客户端：主动发起请求的一方，称之为客户端
  - 服务端：被动的固定在同一个位置上接收请求的一方，称之为服务端

网络字节序

- 字节序：CPU对内存当中的数据进行存取的顺序
  - 大端字节序：低地址存高位
  - 小段字节序：低地址存低位



- 网络字节序都是大端字节序
  - 小段机器和大端机器进行网络通信的时候，如果不转换字节序，则会造成小端机器很小的数字，经过网络传输，被大端机器读出来变成一个大的数字
- 主机字节序：当前机器的字节序
- 字节序转换
  - 大端字节序转换成网络字节序
  - 小端字节序转化成网络字节序
- uint32_t htonl(uint32_t hostlog)
  - 将32位的主机字节序转换为网络字节序
- uint32_t ntohl(uint32_t netlog)
  - 将32位的网络字节序转换为主机字节序
- uint16_t htons(uint16_t hostshort)
  - 将16位的主机字节序转换为网络字节序
- uint16_t ntohs(uint16_t netshort)
  - 将16位的网络字节序转换为主机字节序

TCP与UDP协议的区别

- TCP
  - 面向连接：TCP的通信双方在发送数据之前，需要先建立连接，才可以发送数据
  - 可靠传输：保证TCP数据保可靠的到达对端
  - 面向字节流：对于TCP数据可以随意的存取
- UDP
  - 无连接：UDP通信双方在发送数据之前，不需要建立连接，只需要知道对方的IP和port就可以直接发送数据
  - 不可靠：如果UDP数据在网络当中传输的时候，丢失掉了，则不会保证UDP数据一定到达对端
  - 面向数据报：UDP在发送数据的时候，是整条发送整条接收的

UDP通信流程和编程接口

流程

client(客户端)

- 创建一个套接字
- 绑定地址信息
  - 一般情况下，不让客户端程序绑定地址信息，让操作系统默认分配一个空闲的端口；一个端口只能被一个进程所绑定
- 发送数据
- 接收数据
- 关闭套接字

server(服务端)

- 创建一个套接字：将创建出来的套接字的进程和网卡建立联系，相当于在内核中创建struct socket{.....};
- 绑定地址信息　port+ip
  - 将端口和进程联系起来，port表明服务端进程哪一个端口上侦听数据；ip表明服务端进程在哪一个网卡上接收数据；
- 接收数据
- 发送数据
- 关闭套接字

接口

1.创建套接字

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- domain:地址域，设置网络层使用什么协议
  - 网络层：ipv4——> AF_INET、ipv6——> AF_INET6
- type:套接字类型，传输层使用什么协议：TCP/UDP
  - 流式套接字：默认协议是TCP，不支持UDP——> SOCK_STREAM
  - 数据报套接字：默认协议是UDP，不支持TCP——> SOCK_DGRAM
- protocol:指定套接字所使用的协议
  - 0：采用套接字默认的类型
  - IPPROTO_TCP——> 6
  - IPPROTO_UDP——>17
- 返回值：返回套接字的操作句柄，其值就是一个文件描述符，我们一般称之为套接字描述符

2.绑定地址信息

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr* my_addr, socklen_t addrlen);
```

- sockfd：套接字操作句柄
- my_addr：地址信息 ip+port

```
struct sockaddr{
    sa_family_t sa_family;  //填充地址域，告诉bind函数，网络层使用什么协议
    char sa_data[14];
}
```

```
//ipv4结构体
struct sockaddr_in
{
    _SOCKADDR_COMMON (sin_);    //2个字节的地址域信息
    in_port_t sin_port; //2个字节的无符号整数，端口
    struct in_addr sin_addr;    //4个字节无符号整数，ip地址
    unsighed char sin_zero[sizeof(struct sockaddr)-
        _SOCKADDR_COMMON_SIZE -
        sizeof(int_port_t) -
        sizeof(struct in_addr)];    //8个字节的保留空间
}
```

- bind接口在设计的时候为了通用各种协议，定义了一个结构体struct sockaddr;而在进行具体协议地址信息绑定的时候，填充不同的结构体，之后将结构体对象的地址，强转传参给bind函数。
- addrlen：地址信息的长度，防止有的协议的地址信息长度大于16字节，所以传递地址信息长度，告诉bind函数如何去解析地址信息。
- ```
  #include <sys/socket.h>
  #include <netinet/in.h>
  #include <arpa/inet.h>
  unsigned long int inet_addr(const char *str)
  ```
  - 将点分十进制的IP转换成unit32
  - 将主机字节序转换为网络字节序

3.发送数据

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t sendto(int sockfd, const void* buf, size_t len, int flags, const
strucr sockaddr* dest_addr, socklen_t addrlen)
```

- sockfd：套接字描述符
- buf：待发送的数据
- len：发送的数据长度
- flags：0——>阻塞发送
  - UDP也是有发送缓冲区的，只不过UDP的特点是整条发送，所以在发送缓冲区当中打上UDP协议报头之后就提交给网络层；
  - 阻塞发送的含义：当发送缓冲区中有数据的时候，再次调用sendto接口，会阻塞等待，直到上一条数据发送完毕，sendto才将数据写入到发送缓冲区。
- dest_addr：目标主机地址信息
- addrlen：地址信息长度

4.接收接口
```

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void* buf, size_t len, int flags, struct
sockaddr* src_addr, socklen_t* addrlen)
```

sockfd：套接字描述符

buf：从接收缓冲区当中拿到的数据放到哪一个buffer当中

len：buffer的最大长度，意味着最大可以接收多少数据，预留 “\0” 的位置

flags：标志位 0——>阻塞接受

src_addr：源主机的地址信息(标识这条数据从哪一个主机上面的哪一个进程来)

addrlen：地址信息长度，同时是一个输入输出型参数

5.关闭套接字的接口

```
#include <unistd.h>
int close(int sockfd);
```