

网络编程套接字

网络编程

预备知识

H₃ 源IP地址和目的IP地址

在IP数据包头部中，有两个IP地址，分别叫做(☆ 源IP地址)和(☆ 目的IP地址) --> 始终不变的

- 与之对应的还有(☆ 源MAC地址)和(☆ 目的MAC地址) --> 是会不断变化的

H₃ 认识端口号

(☆ 端口号(port))是传输层协议的内容

- 端口号是一个2字节16位的数据
- 端口号用来标识一个进程，告诉操作系统，当前的这个数据要交给哪一个进程来处理
- IP地址+端口号能够标识网络上的某一台主机的某一个进程 --> 称之为socket通信
- 一个端口号只能被一个进程占用

socket通信，本质上是进程间通信，跨网络的进程间通讯

任何的网路服务与网络客服端，如果要进行正常的数据通讯，必须要用端口号，来唯一标识自身。

H₃ 端口号与进程ID

一个进程可以绑定多个端口号，一个端口号只能绑定一个进程

一台机器上，可能存在大量的进程，但不是所有的进程都要对外进行网络数据请求

H₃ 源端口号与目的端口号

传输层协议(TCP和UDP)的数据段中有两个端口号，分别叫做源端口号和目的端口号，就是在描述“数据是谁发的，要发给谁？”

H₃ TCP协议

☆ TCP，Transmission Control Protocol传输控制协议

- 传输层协议
- 有连接
- 可靠传输
- 传输速度慢
- 面向字节流

H₃ UDP协议

☆ UDP，User Datagram Protocol用户数据报协议

- 传输层协议
- 无连接
- 不可靠传输
- 传输速度快
- 面向数据报

H₃ 网络字节序

- 发送主机通常将发送缓冲区中的数据按内存地址从低到高的顺序发出
- 接收主机把从网络上接到的字节依次保存在接收缓冲区中，也是按内存地址从低到高的顺序保存
- 因此，网络数据流的地址应这样规定：**先发出的数据是低地址，后发出的数据是高地址**
- TCP/IP协议规定，**网络数据流应采用大端字节序，即低地址高字节**

为了网络程序具有可移植性，可以调用以下的库函数进行网络字节序和主机字节序的转换

```
1  #include <arpa/inet.h>
2
3  uint32_t htonl(uint32_t hostlong);
4  uint16_t htons(uint16_t hostshort);
5  uint32_t ntohl(uint32_t netlong);
6  uint16_t ntohs(uint16_t netshort);
```

- h表示host，n表示network，表示32位长整数，s表示16位短整数
- htonl表示将32位的长整数从主机字节序转换为网络字节序
- 如果主机是小端字节序，这些函数将参数做相应的大小端转换后返回
- 如果主机是大端字节序，这些函数不做转换，将参数原封不动返回

H₃ socket编程接口

socket常见API

```
1  //创建socket文件描述符(TCP/UDP, 客户端 + 服务器)
2  int socket(int domain, int type, int protocol);
3  //绑定端口号(TCP/UDP, 服务器)
4  int bind(int socket, const struct sockaddr* address, sock
```

```

len_t address_len);
5 //开始监听socket(TCP, 服务器)
6 int listen(int socket, int backlog);
7 //接收请求(TCP, 服务器)
8 int accept(int socket, struct sockaddr* address, socklen_t address_len);
9 //创建连接(TCP, 服务器)
10 int connect(int sockfd, const struct sockaddr* addr, socklen_t addrlen);

```

sockaddr结构

在这套结构出来的时候，C语言还不支持void*，因此不用void*来承接sockaddr

```

1 struct sockaddr
2 struct sockaddr_in
3 struct sockaddr_un

```

- struct sockaddr_in
 - 16位地址类型为：AF_INET
 - 16位端口号
 - 32位IP地址
 - 8字节填充
- struct sockaddr_un
 - 16位地址类型为：AF_UNIX
 - 108字节路径名
- struct sockaddr
 - 16位地址类型

- 14位地址数据

详解socketAPI

socket

```
1 int socket(int domain, int type, int protocol);
```

- domain：域，协议家族
 - 一般参数使用：
 - AF_INET --> IPv4 Internet protocols
 - AF_UNIX --> Local Communication
- type：服务类型
 - 一般参数使用：
 - SOCK_STREAM --> TCP流式套接字
 - SOCK_DGRAM --> UDP用户数据报服务
- protocol：协议类别
 - 一般参数使用：
 - 0，因为会自动推导
- RETURN VAR：
 - 成功返回一个新的文件描述符，失败则返回-1

bind

```
1 int bind(int sockfd, const struct sockaddr* addr, socklen_t addrlen);
```

- sockfd
 - 一般参数使用：

- 创建socket时返回的fd

- addr

- 一般参数使用

- struct sockaddr_in

- 端口号通常是16位，IPv4通常是32位比特位的数据

(☆ 点分十进制)： "192.168.233.123"类似这样的， [0~255].[0~255].
[...].[...], uint32_t 整数IP

```
C++>
1 //如果想要将字符串IP转化为整形IP
2 in_addr_t inet_addr(const char* cp);    //最简单的一种方式
3 int inet_aton(const char* cp, struct in_addr* inp)
4 ;
5 //如果想要将整形IP转化为字符串IP
6 char* inet_ntoa(struct in_addr in);
```

```
C++>
1 //操作系统内核使用的方式，来进行字符串IP和整数IP的转换
2 struct A{
3     uint32_t p1 : 8;
4     uint32_t p2 : 8;
5     uint32_t p3 : 8;
6     uint32_t p4 : 8;
7 };
8 union IP{
9     uint32_t ip;
10    struct A p;
11 };
```

- 列表

```
C++>
1 struct sockaddr_in{
2     __SOCKADDR_COMMON (sin_);    //Address family
3     in_port_t sin_addr;    //Port number
4     struct in_addr sin_addr;    //Internet address
5 };
6 typedef uint32_t in_addr_t;
7 struct in_addr{
```

```
8     in_addr_t s_addr;  
9 };
```

- RETERN VAL :
 - 成功返回0， 失败则是返回-1
- The Header

```
1 #include <sys/socket.h>  
2 #include <netinet/in.h>  
3 #include <arpa/inet.h>
```

基于UDP服务器的接口

```
1 //接收数据  
2 ssize_t recvfrom(int sockfd, void* buf, size_t len, int f  
lags, struct sockaddr* src_addr, socklen_t* addrlen);  
3 //发送数据  
4 ssize_t sendto(int sockfd, const void* buf, size_t len, i  
nt flags, coonst struct sockaddr* dest_addr, socklen_t ad  
drlen);
```

- sockfd
 - 特定套接字描述符
- buf
 - 数据存储的缓冲区
- len
 - 数据读写的长度
- flags
 - 读写的方式
 - 一般参数使用：
 - 暂时使用0
- src_addr

- 谁向我这个服务器发送的消息， 对端信息(socket, IP, port)，是一个输入输出型数据
- dest_addr
 - 我这个服务器向谁发送了信息， 源端信息(socket, IP, port)，是一个输入输出型数据
- addrlen
 - 对应的结构体大小
- RETURN VAR:
 - 如果成功，返回实际发送的字节数，失败则返回 -1



netstat

- netstat
 - 查看当前主机的网络状态
 - 一般参数使用：
 - -nltp
 - n： 能显示成数字就显示成数字
 - l： list
 - t： TCP
 - u： UDP
 - p： PID

客户端的问题

- 客户端不需要绑定端口吗？
 - 不需要，但不是不能
 - 服务器为何要bind一个端口：服务器是要为了给别人提供服务的，别人一定要知道你的ip和端口
 - 端口一定要是一个众所周知的端口，且bind之后，不能轻易改变

- 因为客户端访问server,端口只要是唯一的即可，不需要和特定client进程强相关，**client的端口可以动态的设置**
- 只需要调用一个类似sendto的接口，client直接在OS层面自动给client获取一个唯一的端口

- C++> 

```
1 ssize_t sendto(int sockfd, const void* buf, size_t  
len, int flags, const struct sockaddr* dest_addr,  
socklen_t addrlen);
```

- 客户端不需要ip吗?
 - 需要
 - 获取到服务器的IP地址

为什么云服务器绑定公网ip会失败？

- 云服务器的ip是由对应的云厂商提供的，这个ip不能直接被绑定，如果需要bind,需要让外网访问，就需要bind 0也就是INADDR_ANY，就意味着服务器可以接收来自任意client的请求，一般推荐直接使用该方案