在UDP/TCP套接字编程，因为业务需要知道本地客户端、对端服务端的地址信息。先将有关地址获取的函数进行说明，再根据使用场景选择对应函数。

**1、不同函数介绍**

先介绍有关获取地址信息的函数 recvfrom、accept、getsockname、getpeername等，其他函数这里暂不做说明。

**1.1 recvfrom**

#include <sys/socket.h>

ssize\_t recvfrom(int sock, void \*buf, size\_t len, int flags, struct sockaddr \*from, socklen\_t \*fromlen);

关注后两个参数from和fromlen，用于获取对端的地址信息。指针from为NULL时，表示不关心对端地址信息，同时指针fromlen也赋值为NULL。

当需要获取对端信息时，需要传递对象保存地址结构的指针from，以及当前结构地址的大小fromlen。recvfrom函数正常执行时，会将对端的地址信息写入from指向对象，并且会重写fromlen值为实际对端地址结构大小。但是，当传入的fromlen值小于对端地址结构大小，会造获取信息截断。例如传入的是sockaddr\_in，而实际的对端地址结构是sockaddr\_in6。

可以选择足够大的空间对象保存对端地址结构，例如传入sockaddr\_in6对象，以满足对端是ipv4或ipv6地址，再根据fromlen来解析不同地址结构的信息。

**1.2 accept**

#include <sys/socket.h>

int accept(int sockfd, struct sockaddr \*clientaddr, socklen\_t\* addrlen); // 成功返回非负的已连接套接字，出错返回-1

使用参数完全同recvfrom函数，获取地址结构也相同。注意返回值，成功时会返回一个非负的已连接套接字描述符（已经绑定对端地址信息）。我们可**以利用这个返回值，调用getpeername获取对端客户端的地址信息**。

**1.3 getsockname、getpeername**

#include <sys/socket.h>

int getpeername (int sockfd, struct sockaddr \*localaddr, socklen\_t \* addrlen);

int getsockname (int sockfd, struct sockaddr \*peeraddr, socklen\_t \* addrlen); // 成功返回0，出错返回-1

两个函数放在一起说明，后两个参数含义、使用方法同上述recvfrom。

当使用getsockname时，sockfd 应该是一个已经绑定到本地地址的套接字，这个本地地址可以是手动bind或者由内核分配的。

当使用getpeername时，sockfd 应该是一个已经绑定对端地址的套接字，例如服务端经accept后的返回值套接字，客户端经过connect之后的本地套接字。

**2、使用场景**

**2.1、获取本地地址信息**

用于udp和tcp客户端，不同客户端继续可以划分。

**2.1.1 UDP客户端获取本地地址**

客户端bind或者connect之后使用 getsockname。为操作方便，先实现一个通用函数以打印输出本地地址信息，

1. 输出本地地址函数

void print\_getsockname(int socket\_fd)

{

sockaddr\_storage storage;

socklen\_t sock\_len = sizeof(storage); // 必须给初值

int ret = getsockname(socket\_fd, (sockaddr\*)&storage, &sock\_len);

if(ret < 0){

printf("getsockname error: %s\n", strerror(errno));

return;

}

if (storage.ss\_family == AF\_INET){

sockaddr\_in\* addr = (sockaddr\_in\* )&storage;

printf("local addr: %s:%d\n", inet\_ntoa(addr->sin\_addr), ntohs(addr->sin\_port));

}

else if(storage.ss\_family == AF\_INET6){

sockaddr\_in6\* addr = (sockaddr\_in6\* )&storage;

char ip[INET6\_ADDRSTRLEN];

inet\_ntop(AF\_INET6, &addr->sin6\_addr, ip, sizeof(addr));

printf("local addr: %s:%d\n", ip, ntohs(addr->sin6\_port));

}

}

（2）常规使用

对于未bind的套接字，必须发送数据后，内核会分配端口。

/// 创建socket

int socket\_fd = socket(AF\_INET,SOCK\_DGRAM, 0); // udp

/// 连接

sockaddr\_in servaddr;

servaddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET,"127.0.0.1", &servaddr.sin\_addr);

servaddr.sin\_port = htons(8080);

// 对于未bind的套接字，必须发送数据后，内核会分配端口

Int ret = sendto(socket\_fd, "",0,0, (sockaddr\*)&servaddr, sizeof(servaddr));

if(ret < 0){

printf("bind error: %s\n", strerror(errno));

}

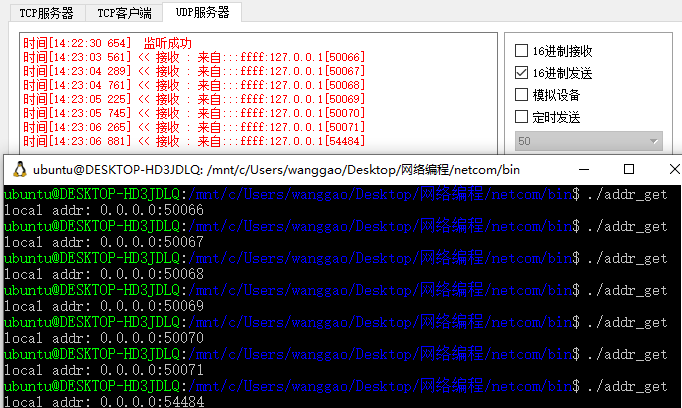
/// 获取本地信息

print\_getsockname(socket\_fd);

/// 4、关闭连接

::close(socket\_fd);

print\_getsockname(socket\_fd);函数输出的ip地址默认是”0.0.0.0”，但是服务端是仍然能解析的。如下：



（3）使用connect函数

将上面常规方式中发送数据的代码替换成connect，效果一样。

/// 创建socket

int socket\_fd = socket(AF\_INET,SOCK\_DGRAM, 0); // udp

/// 连接

sockaddr\_in servaddr;

servaddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET,"127.0.0.1", &servaddr.sin\_addr);

servaddr.sin\_port = htons(8080);

int ret = ::connect(socket\_fd, (sockaddr\*)&servaddr, sizeof(servaddr));

if(ret < 0){

printf("bind error: %s\n", strerror(errno));

return 0;

}

/// 获取本地信息

print\_getsockname(socket\_fd);

/// 4、关闭连接

::close(socket\_fd);

（4）使用bind

首先，替换常规方法中发送数据部分，仅调用bind后执行print\_getsockname函数。结果相对会存在一些问题。

例如，ip指定为127.0.0.1, port指定9000，bind成功后，getsockname函数返回正常；

ip指定127.0.0.1, port指定0，bind成功后，getsockname函数返回正常，端口为内核分配；

ip指定INADDR\_ANY, port指定0，bind成功后，getsockname函数返回ip为”0.0.0.0”， 端口为内核分配。

接着，bind之后，先发送任意数据到服务端，再调用getsockname，结果不变。

换句话说，**使用bind后，端口要要么是指定的，要么是内核分配的，最终的端口getsockname都能正确获取；当地址为通配INADDR\_ANY时，最终选择的地址getsockname没有办法知道的**。

（5）不常见的使用方法。见2.2.1节。

**2.1.2 TCP客户端获取本地地址**

对于TCP客户端，使用bind后再调用getsockname的结果，也是跟UDP的情况一样。

**TCP客户端获取本地地址，常规使用connect函数，之后调用getsockname**。代码如下

/// 创建socket

int socket\_fd = socket(AF\_INET,SOCK\_STREAM, 0); // tcp

/// 连接

sockaddr\_in servaddr;

servaddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET,"127.0.0.1", &servaddr.sin\_addr);

servaddr.sin\_port = htons(8080);

int ret = ::connect(socket\_fd, (sockaddr\*)&servaddr, sizeof(servaddr));

if(ret < 0){

printf("bind error: %s\n", strerror(errno));

return 0;

}

/// 获取本地信息

print\_getsockname(socket\_fd);

/// 4、关闭连接

::close(socket\_fd);

结果如下



当先使用bind指定端口，且指定地址为INADDR\_ANY， 再经过connect之后，能够使用getsockname获取最终tcp选择的ip地址。

如bind部分代码

/// 创建socket

int socket\_fd = socket(AF\_INET,SOCK\_STREAM, 0); // tcp

/// 连接

sockaddr\_in servaddr;

servaddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET,"127.0.0.1", &servaddr.sin\_addr);

servaddr.sin\_port = htons(8080);

/// bind 或 connect

int ret;

sockaddr\_in localaddr;

localaddr.sin\_family = AF\_INET;

localaddr.sin\_addr.s\_addr = INADDR\_ANY;

localaddr.sin\_port = htons(9000);

ret = ::bind(socket\_fd, (sockaddr\*)&localaddr, sizeof(localaddr));

if(ret < 0){

printf("bind error: %s\n", strerror(errno));

}

ret = ::connect(socket\_fd, (sockaddr\*)&servaddr, sizeof(servaddr));

if(ret < 0){

printf("bind error: %s\n", strerror(errno));

return 0;

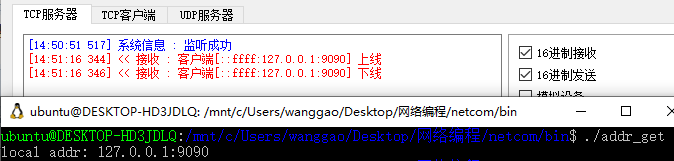
}

/// 获取本地信息

print\_getsockname(socket\_fd);

/// 4、关闭连接

::close(socket\_fd);



**2.2、获取对端地址信息**

获取对端的地址，UDP主要通过recvfrom函数， TCP主要通过getpeername函数。

**2.2.1 UDP中获取对端地址**

**UPP服务端和客户端都可能使用recvfrom函数接收来自对端的数据，同时也能获取对端的地址信息**。以服务端为例，如下

/// 创建socket

int socket\_fd = socket(AF\_INET,SOCK\_DGRAM, 0); // udp

/// bind

int ret;

sockaddr\_in localaddr;

localaddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, "127.0.0.1", &localaddr.sin\_addr);

localaddr.sin\_port = htons(8080);

ret = ::bind(socket\_fd, (sockaddr\*)&localaddr, sizeof(localaddr));

if(ret < 0){

printf("bind error: %s\n", strerror(errno));

}

/// 接收

char buf[1024];

int len;

while(true)

{

sockaddr\_storage storage;

socklen\_t sock\_len = sizeof(storage); // 必须给初值

len = ::recvfrom(socket\_fd, buf, sizeof(buf), 0, (struct sockaddr \*)&storage, &sock\_len);

if (len < 0){

printf("recv failed. err %s\n", strerror(errno));

return;

}

buf[len] = '\0';

/// 输出对端信息

if (storage.ss\_family == AF\_INET){

sockaddr\_in\* addr = (sockaddr\_in\* )&storage;

char ip[INET\_ADDRSTRLEN];

inet\_ntop(AF\_INET, &addr->sin\_addr, ip, sock\_len);

printf("recv client [%s:%d] %2d: %s", ip, ntohs(addr->sin\_port), len, buf);

}

else if(storage.ss\_family == AF\_INET6){

sockaddr\_in6\* addr = (sockaddr\_in6\* )&storage;

char ip[INET6\_ADDRSTRLEN];

inet\_ntop(AF\_INET6, &addr->sin6\_addr, ip, sock\_len);

printf("recv client [%s:%d] %2d: %s", ip, ntohs(addr->sin6\_port), len, buf);

}

}

// 关闭连接

::close(socket\_fd);

函数recvfrom返回成功后，参数storage、sock\_len存储了对端的地址信息，同前面获取本地地址一样解析，根据长度或协议类型进行解析即可。

在UDP客户端使用中，有一个不常用的方法获取本地信息，即通过connect后，使用getsockname获取本地信息。

int socket\_fd = socket(AF\_INET,SOCK\_DGRAM, 0); // udp

/// 连接

sockaddr\_in servaddr;

servaddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET,"127.0.0.1", &servaddr.sin\_addr);

servaddr.sin\_port = htons(8080);

::connect(socket\_fd, (sockaddr\*)&servaddr, sizeof(servaddr));

/// 获取地址信息

print\_getpeername(socket\_fd); // 函数见下一节 TCP中获取对端地址

print\_getsockname(socket\_fd);

// 当前socket\_fd是一个已连接的UDP套接字, 理论上需要使用write或send函数

//::sendto(socket\_fd,"123",3, 0, (sockaddr\*)&servaddr, sizeof(servaddr));

::write(socket\_fd,"123",3);

// 关闭连接

::close(socket\_fd);



**2.2.2 TCP中获取对端地址**

TCP服务端直接使用accept，能够接收客户端的连接，并且能够获取客户端的地址信息；其次，accept返回值是当前客户端已连接的套接字，可以使用getpeername获取客户端地址信息。

TCP客户端也可以在connect之后，调用getpeername获取服务端信息。

1. 使用accept直接获取对端信息

/// 创建socket

int socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0); // tcp

/// bind

int ret;

sockaddr\_in localaddr;

localaddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, "127.0.0.1", &localaddr.sin\_addr);

localaddr.sin\_port = htons(8080);

ret = ::bind(socket\_fd, (sockaddr \*)&localaddr, sizeof(localaddr));

// 监听

::listen(socket\_fd, 5);

sockaddr\_storage storage;

socklen\_t sock\_len = sizeof(storage); // 必须给初值

::accept(socket\_fd, (sockaddr \*)&storage, &sock\_len);

/// 输出对端信息

if (storage.ss\_family == AF\_INET){

sockaddr\_in \*addr = (sockaddr\_in \*)&storage;

char ip[INET\_ADDRSTRLEN];

inet\_ntop(AF\_INET, &addr->sin\_addr, ip, sock\_len);

printf("client [%s:%d] \n", ip, ntohs(addr->sin\_port));

}

else if (storage.ss\_family == AF\_INET6){

sockaddr\_in6 \*addr = (sockaddr\_in6 \*)&storage;

char ip[INET6\_ADDRSTRLEN];

inet\_ntop(AF\_INET6, &addr->sin6\_addr, ip, sock\_len);

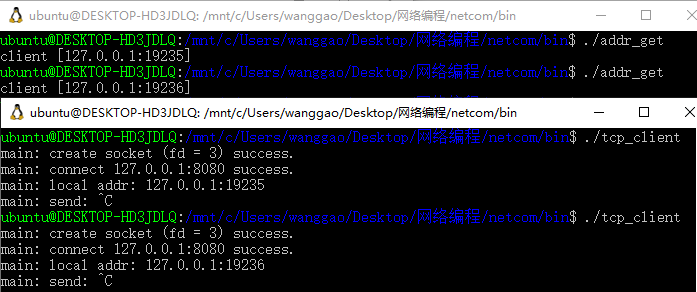
printf("client [%s:%d] \n", ip, ntohs(addr->sin6\_port));

}

// 关闭连接

::close(socket\_fd);

经过accept之后，需要根据协议类型解析地址信息。测试代码反复运行，结果如下：



1. 使用accept返回值调用getpeername

注意，传递给getpeername的是函数accept的返回值sock\_id，这个是已连接的客户端套接字，不是服务端的套接字sock\_fd。

其中包含了print\_getpeername()的代码，见下面注释。演示效果同使用accept直接获取对端地址信息方法。

/// 创建socket

int socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0); // tcp

/// bind

int ret;

sockaddr\_in localaddr;

localaddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, "127.0.0.1", &localaddr.sin\_addr);

localaddr.sin\_port = htons(8080);

ret = ::bind(socket\_fd, (sockaddr \*)&localaddr, sizeof(localaddr));

// 监听

::listen(socket\_fd, 5);

// 等待连接

int sock\_id = ::accept(socket\_fd, NULL, NULL);

/// 输出对端信息，实际是 print\_getsockname()函数;

sockaddr\_storage storage;

socklen\_t sock\_len = sizeof(storage); // 必须给初值

ret = ::getpeername(sock\_id, (sockaddr \*)&storage, &sock\_len); // 注意是sock\_id

if (ret < 0){

printf("getpeername error: %s\n", strerror(errno));

}

else{

if (storage.ss\_family == AF\_INET){

sockaddr\_in \*addr = (sockaddr\_in \*)&storage;

char ip[INET\_ADDRSTRLEN];

inet\_ntop(AF\_INET, &addr->sin\_addr, ip, sock\_len);

printf("client [%s:%d] \n", ip, ntohs(addr->sin\_port));

}

else if (storage.ss\_family == AF\_INET6){

sockaddr\_in6 \*addr = (sockaddr\_in6 \*)&storage;

char ip[INET6\_ADDRSTRLEN];

inet\_ntop(AF\_INET6, &addr->sin6\_addr, ip, sock\_len);

printf("client [%s:%d] \n", ip, ntohs(addr->sin6\_port));

}

}

// 关闭连接

::close(sock\_id);

::close(socket\_fd);

**3、总结**

**3.1 获取对端地址信息**

recvfrom 多用于udp服务端和客户端

accept 用于tcp服务端

getpeername tcp服务端要在accept之后，tcp/udp客户端要在connect之后

**3.2 获取本地地址信息**

getsockname 可以直接在bind后获取准确的port，在connect、accept之后可以获取准确的port和ip。

**3.3 解析地址信息**

接收地址新的对象空间足够大，根据函数返回的地址信息长度或者协议类型进行解析。 以Ipv4和ipv6地址为例，选择sockaddr\_in6时根据长度解析，选择sockaddr\_storage时根据协议解析。