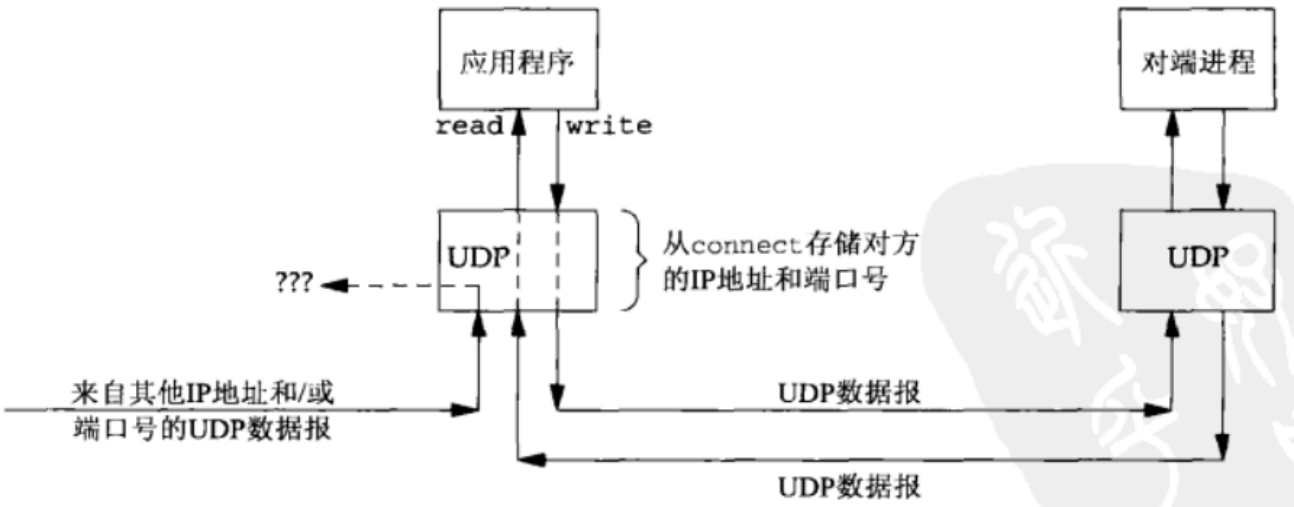
Connected UDP

但是UDP提供了这样的一个connect（）方法，它有两种使用方式，**当使用了这个方法之后，那么就限定了这个socket的使用范围： 只允许从这个指定的SocketAddress 上获得数据包和向这个指定的SocketAddress 发送数据包，** 当你一旦通过这个socket向别的地址发送了数据包，或者接收到了不是这个地址发送来的数据包，那么程序就会抛出IllegalArgumentException 异常（即**ICMP错误**）， 特殊的是如果指定的这个SocketAddress 是个多播地址或者广播地址，那么只允许向这个地址发送数据包，不允许从这个地址接收数据包。

```
// SocketAddress 其实就是IP地址+端口号 ，  而InetAddress 只有IP地址
connect(InetAddress address, int port)
connect(SocketAddress addr)
```

UDP通过这样的方式，限定了单向的通信，但是注意的是，这里的限定只是仅限于一方，而并没有限制另外一方，另外一方依旧是可以向多个IP地址发送数据包的，因此这和TCP/IP 是极大的不同。**TCP/IP的connect()是要有一个三次握手的过程的，而UDP的connect显然没有，它只是将IP地址和端口号进行了存储，对要进行通信的对象做了一个限制。** 而且 **TCP/IP的connect()只可以进行一次，但是UDP的connect()可以调用多次，** 每次重新调用connect(SocketAddress)，就是将原来限制通信的对象修改为新的这个地址的对象， 或者调用disConnect() 方法，就会解除对通信对象的限制，这样这个socket就又可以多向的通信了。

即存储UPD地址和端口号，对通信对象加限制，可以重复connect方法，绑定地址和端口号或者断开连接。



当我们进行UDP通信的对象只有一个时，建议使用connect()方法，使用了这个方法之后有一个极大的好处： **当我们使用了connect(SocketAddress addr) 方法时，那么在socket对象里面就会将发送方的地址设置为此地址，那么发送的数据包对象就不用显式的标明 IP地址和 Port ，这样在调用send（packet）方法时，就不会对数据包再进行 IP地址和Port的安全检查，** 要发送的数据包少时优势体现不出来，但是当数据包多时，可以节省大量的时间。

即connected UDP可以省去数据包IP地址和Port的安全检查，节省大量时间。

send方法的源码：

```
    public void send(DatagramPacket p) throws IOException  {
        InetAddress packetAddress = null;
        synchronized (p) {
            if (isClosed())
                throw new SocketException("Socket is closed");
            checkAddress (p.getAddress(), "send");
            // ST_NOT_CONNECTED 表示着当前没有要限制通信的对象
            if (connectState == ST_NOT_CONNECTED) {
                // check the address is ok wiht the security manager on every send.
                SecurityManager security = System.getSecurityManager();

                // The reason you want to synchronize on datagram packet
                // is because you dont want an applet to change the address
                // while you are trying to send the packet for example
                // after the security check but before the send.
                // 进行IP 和 port 的安全检查
                if (security != null) {
                    if (p.getAddress().isMulticastAddress()) {
                        security.checkMulticast(p.getAddress());
                    } else {
                        security.checkConnect(p.getAddress().getHostAddress(),
                                              p.getPort());
                    }
                }
            } else {
                // we're connected
                packetAddress = p.getAddress();
                if (packetAddress == null) {
                    p.setAddress(connectedAddress);
                    p.setPort(connectedPort);
                } else if ((!packetAddress.equals(connectedAddress)) ||
                           p.getPort() != connectedPort) {
                    throw new IllegalArgumentException("connected address " +
                                                       "and packet address" +
                                                       " differ");
                }
            }
            // Check whether the socket is bound
            if (!isBound())
                bind(new InetSocketAddress(0));
            // call the  method to send
            getImpl().send(p);
        }
    }
```

Connected UDP的调用方法：

1. 再不能给输出操作指定目的IP地址和端口号。

   sendto改用write或send，写到已连接UDP套接字上的任何内容都自动发送到由connect指定的协议地址（如IP地址和端口号）。

2. 可不使用recvfrom以获悉数据报的发送者，改用read、recv或recvmsg。

3. 由已连接UDP套接字引发的异步错误会返回给它们所在的进程，而未连接UDP套接字不接受任何异步错误。

```
void dg_cli(FILE *fp, int sockfd, SA *pservaddr, socklen_t servlen)
{
    int n;
    char sendline[MAXLINE], recvline[MAXLINE + 1];
    if (connect(sockfd, (SA*)pservaddr, servlen) < 0){
        cout<<"conn error!"<<endl;
        exit(0);
    }
    else
        cout<<"conn succ!"<<endl;

    while (fgets(sendline, MAXLINE, fp) != NULL){
        if (write(sockfd, sendline, strlen(sendline)) < 0){
            cout<<"write error"<<endl;
            exit(0);
        }

        if ( (n = read(sockfd, recvline, MAXLINE)) < 0){
            cout<<"read error"<<endl;
            exit(0);
        }

        recvline[n] = 0;
        fputs(recvline, stdout);

        bzero(sendline, MAXLINE);
        bzero(recvline, MAXLINE);
    }
}
```