**网络协议： TCP/IP 和UDP/IP**

TCP五层模型：

应用层、传输层、网络层、链路层、物理传输层

OSI七层模型

应用层、表现层、会话层、传输层、网络层、链路层、物理传输层

## TCP/IP

TCP/IP（Transmission Control Protocol/Internet Protocol）是一种可靠的网络数据传输控制协议。定义了主机如何连入因特网以及数据如何在他们之间传输的标准。

TCP/IP协议参考模型把所有TCP/IP系列协议归类到四个抽象层中；

每一个抽象层建立在低一层提供的服务上，并且为高一层提供服务

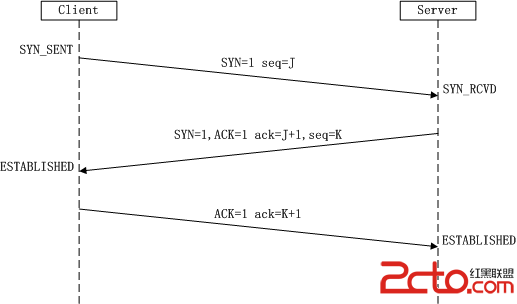
|  |  |
| --- | --- |
| http://images.cnitblog.com/blog/349217/201312/05230830-04807bb739954461a8bfc7513707f253.jpg | ICMP:控制报文协议  IGMP:internet组管理协议  ARP:地址解析协议  RARP:反向地址转化协议 |

OSI模型（开放式系统互联通信参考模型），它是由国际标准化组织提出的，试图使各种计算机在世界范围内互联为网络的标准框架

OSI模型多了表达层、会话层

### 3次握手协议

所谓三次握手（Three-Way Handshake）即建立TCP连接，就是指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立



（1）第一次握手：Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。

（2）第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。

（3）第三次握手：Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。

*SYN攻击：*

在三次握手过程中，Server发送SYN-ACK之后，收到Client的ACK之前的TCP连接称为半连接（half-open connect），此时Server处于SYN\_RCVD状态，当收到ACK后，Server转入ESTABLISHED状态。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，Server回复确认包，并等待Client的确认，由于源地址是不存在的，因此，Server需要不断重发直至超时，这些伪造的SYN包将长时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络堵塞甚至系统瘫痪。SYN攻击时一种典型的DDOS攻击，检测SYN攻击的方式非常简单，即当Server上有大量半连接状态且源IP地址是随机的，则可以断定遭到SYN攻击了，使用如下命令可以让之现行：

#netstat -nap | grep SYN\_RECV

### 4次挥手协议

三次握手耳熟能详，四次挥手估计就听得比较少了，所谓四次挥手（Four-Way Wavehand）即终止TCP连接，就是指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开

|  |  |
| --- | --- |
| http://www.2cto.com/uploadfile/2013/1022/20131022025350523.png | 单工：数据传输只支持数据在一个方向上传输  半双工：数据传输允许数据在两个方向上传输，但是在某一时刻，只允许在一个方向上传输，实际上有点像切换方向的单工通信  全双工：数据通信允许数据同时在两个方向上传输，因此全双工是两个单工通信方式的结合，它要求发送设备和接收设备都有独立的接收和发送能力 |

由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭，上图描述的即是如此。

（1）第一次挥手：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态。

（2）第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE\_WAIT状态。

（3）第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态。

（4）第四次挥手：Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。

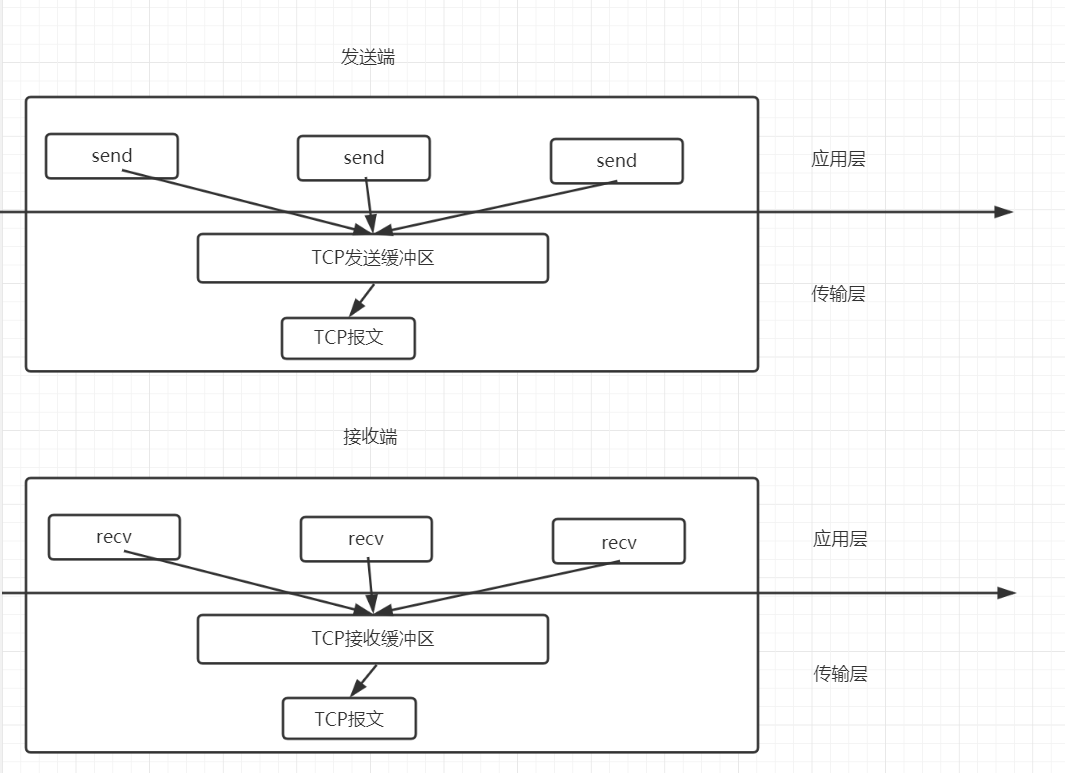
**TCP通信原理**

首先，对于TCP通信来说，每个TCP Socket的内核中都有一个发送缓冲区和一个接收缓冲区，TCP的全双工的工作模式及TCP的滑动窗口就是依赖于这两个独立的Buffer和该Buffer的填充状态。

接收缓冲区把数据缓存到内核，若应用进程一直没有调用Socket的read方法进行读取，那么该数据会一直被缓存在接收缓冲区内。不管进程是否读取Socket，对端发来的数据都会经过内核接收并缓存到Socket的内核接收缓冲区。

read索要做的工作，就是把内核接收缓冲区中的数据复制到应用层用户的Buffer里。

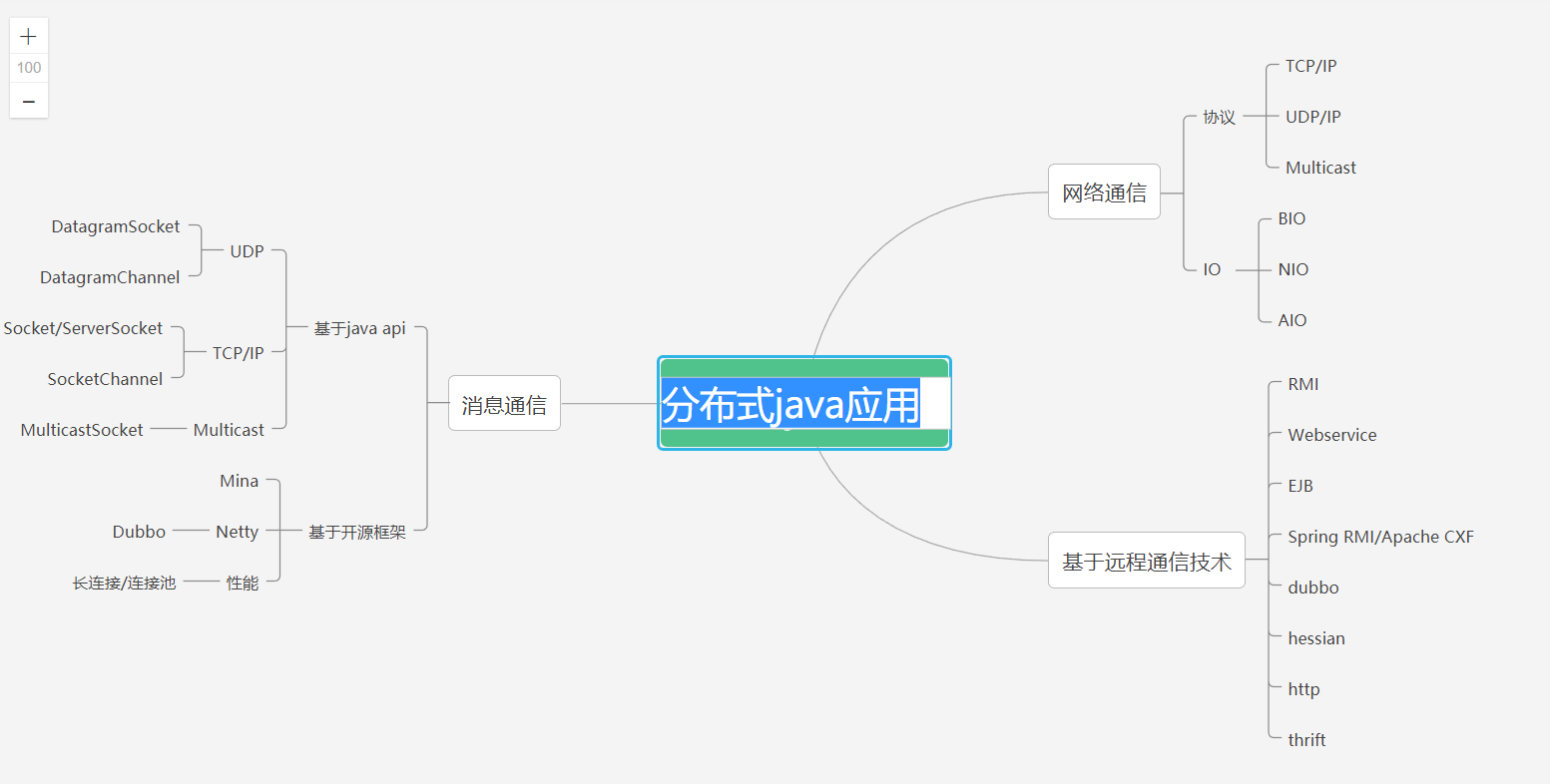
进程调用Socket的send发送数据的时候，一般情况下是讲数据从应用层用户的Buffer里复制到Socket的内核发送缓冲区，然后send就会在上层返回。换句话说，send返回时，数据不一定会被发送到对端。



BIO同步阻塞

NIO同步非阻塞

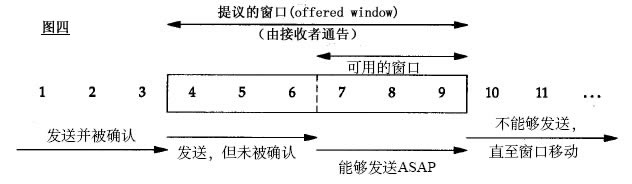
AIO异步非阻塞



## 什么是滑动窗口协议

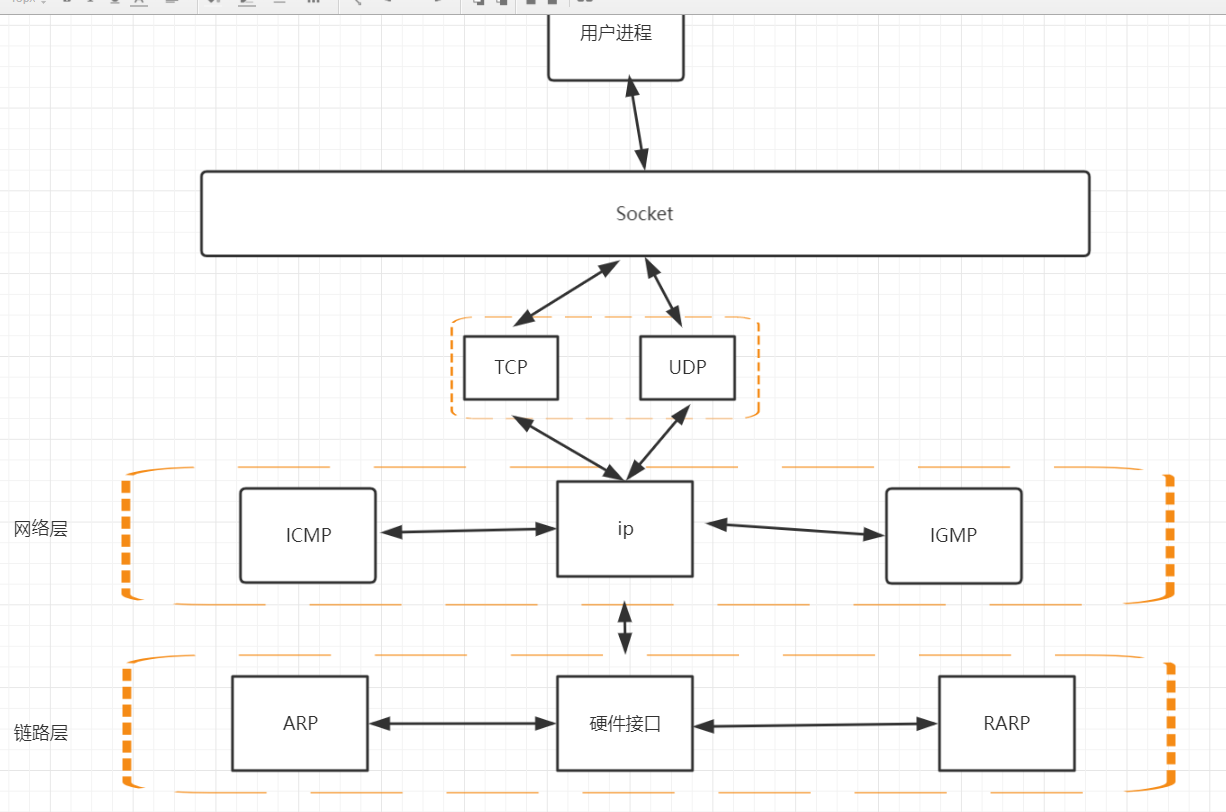
发送方和接收方都会维护一个数据帧的序列，这个序列被称作窗口。发送方的窗口大小由接收方确认，目的是控制发送速度，以免接收方的缓存不够大导致溢出，同时控制流量也可以避免网络拥塞。

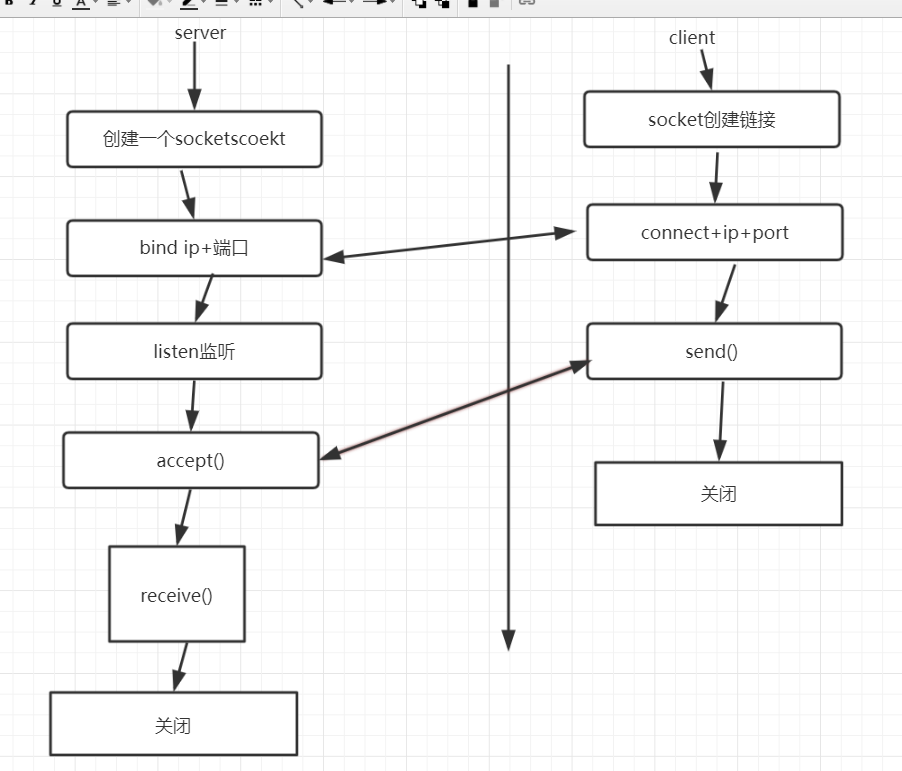
下面图中的4,5,6号数据帧已经被发送出去，但是未收到关联的ACK，7,8,9帧则是等待发送。可以看出发送端的窗口大小为6，这是由接受端告知的（事实上必须考虑拥塞窗口cwnd，这里暂且考虑cwnd>rwnd）。此时如果发送端收到4号ACK，则窗口的左边缘向右收缩，窗口的右边缘则向右扩展，此时窗口就向前“滑动了”，即数据帧10也可以被发送



明白了Socket读写数据的底层原理，我们就很容易理解“阻塞模式”：对于读取Socket数据的过程而言，如果接收缓冲区为空，则调用Socket的read方法的线程会阻塞，知道有数据进入接收缓冲区；而对于写数据到Socket中的线程来说，如果待发送的数据长度大于发送缓冲区空余长度，则会阻塞在write方法上，等待发送缓冲区的报文被发送到网络上，然后继续发送下一段数据，循环上述过程直到数据都被写入到发送缓冲区为止

从前面分析的过程来看，传统的Socket阻塞模式直接导致每个Socket都必须绑定一个线程来操作数据，参与通信的任意一方如果处理数据的速度较慢，会直接拖累到另一方，导致另一方的线程不得不浪费大量的时间在I/O等待上，所以这就是Socket阻塞模式的“缺陷”。但是这种模式在少量的TCP连接通信的情况下，双方都可以快速的传输数据，这个时候的性能是最高的。





**Multicast（组播）**

**单播**

是指封包在计算机网络的传输中，目的地址为单一目标的一种传输方式。它是现今网络应用最为广泛，通常所使用的网络协议或服务大多采用单播传输，例如一切基于TCP的协议。

**组播**

也叫多播， 多点广播或群播。 指把信息同时传递给一组目的地址。它使用策略是最高效的，因为消息在每条网络链路上只需传递一次，而且只有在链路分叉的时候，消息才会被复制。

多播组通过 D 类 IP 地址和标准 UDP 端口号指定。D 类 IP 地址在 224.0.0.0 和 239.255.255.255 的范围内（包括两者）。地址 224.0.0.0 被保留，不应使用。

**广播**

是指封包在计算机网络中传输时，目的地址为网络中所有设备的一种传输方式。实际上，这里所说的“所有设备”也是限定在一个范围之中，称为“广播域”。

**任播(anycast):**

是一种网络寻址和路由的策略，使得资料可以根据路由拓朴来决定送到“最近”或“最好”的目的地。

## 来自维基百科

**单播：**

每次只有两个实体相互通信，发送端和接收端都是唯一确定的。

在IPv4网络中，0.0.0.0到223.255.255.255属于单播地址。

你对小月月喊“小月月”，那么只有小月月回过头来答应你。

**组播**

“组播”这个词通常用来指代IP组播。IP组播是一种通过使用一个组播地址将数据在同一时间以高效的方式发往处于TCP/IP网络上的多个接收者的协议。此外，它还常用来与RTP等音视频协议相结合。

互联网架构师戴夫·克拉克是这样描述IP组播的：“你把数据包从一头放进去，网络就会试图将它们传递到想要得到它们的人那里。”

组播报文的目的地址使用D类IP地址， D类地址不能出现在IP报文的源IP地址字段。

你在大街上大喊一声“美女”， 会有一群女性回头看你。

组播地址（参考 iana）

组播组可以是永久的也可以是临时的。组播组地址中，有一部分由官方分配的，称为永久组播组。永久组播组保持不变的是它的ip地址，组中的成员构成可以发生变化。永久组播组中成员的数量都可以是任意的，甚至可以为零。那些没有保留下来供永久组播组使用的ip组播地址，可以被临时组播组利用。

224.0.0.0～224.0.0.255为预留的组播地址（永久组地址），地址224.0.0.0保留不做分配，其它地址供路由协议使用；

224.0.1.0～224.0.1.255是公用组播地址，Internetwork Control Block；

224.0.2.0～238.255.255.255为用户可用的组播地址（临时组地址），全网范围内有效；

239.0.0.0～239.255.255.255为本地管理组播地址，仅在特定的本地范围内有效。

永久的组播地址：

224.0.0.0 基准地址（保留）

224.0.0.1 所有主机的地址 （包括所有路由器地址）

224.0.0.2 所有组播路由器的地址

224.0.0.3 不分配

224.0.0.4 dvmrp路由器

224.0.0.5 所有ospf路由器

224.0.0.6 ospf DR/BDR

224.0.0.7 st路由器

224.0.0.8 st主机

224.0.0.9 rip-2路由器

224.0.0.10 Eigrp路由器

224.0.0.11 活动代理

224.0.0.12 dhcp 服务器/中继代理

224.0.0.13 所有pim路由器

224.0.0.14 rsvp封装

224.0.0.15 所有cbt路由器

224.0.0.16 指定sbm

224.0.0.17 所有sbms

224.0.0.18 vrrp

以太网传输单播ip报文的时候，目的mac地址使用的是接收者的mac地址。但是在传输组播报文时，传输目的不再是一个具体的接收者，而是一个成员不确定的组，所以使用的是组播mac地址。组播mac地址是和组播ip地址对应的。iana（internet assigned number authority）规定，组播mac地址的高24bit为0x01005e，mac 地址的低23bit为组播ip地址的低23bit。

由于ip组播地址的后28位中只有23位被映射到mac地址，这样就会有32个ip组播地址映射到同一mac地址上。

**广播**

并非所有的计算机网络都支持广播，例如X.25网络和帧中继都不支持广播，而且也没有在“整个互联网范围中”的广播。IPv6亦不支持广播，广播相应的功能由组播代替。

通常，广播都是限制在局域网中的，比如以太网或令牌环网络。因为广播在局域网中造成的影响远比在广域网中小得多。

以太网和IPv4网都用全1的地址表示广播，分别是ff:ff:ff:ff:ff:ff和255.255.255.255。

令牌环网络使用IEEE 802.2控制域中的一个特殊值来表示广播。

你在公司大喊一声“放假了”， 全部同事都会响应，大叫爽死了。

**任播**

任播是与单播、广播和组播不同的方式。

在单播中，在网络位址和网络节点之间存在一一对应的关系。

在广播和组播中，在网络位址和网络节点之间存在一对多的关系：每一个目的位址对应一群接收可以复制资讯的节点。

在任播中，在网络位址和网络节点之间存在一对多的关系：每一个位址对应一群接收节点，但在任何给定时间，只有其中之一可以接收到传送端来的资讯。

在互联网中，通常使用边界网关协议来实现任播。

作为老板，你在公司大喊一声“开发组的过来一个人”， 总会有一个人灰溜溜去响应， 挨批还是发钱啊？

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.InetAddress;

import java.net.MulticastSocket;

import java.net.UnknownHostException;

import java.util.concurrent.TimeUnit;

public class MuiltcastDemo {

private static int PORT = 5555;

private static String MULTICAST\_ADDRESS = "225.0.0.1";

static class MulticastClient {

private MulticastSocket socket;

private String id;

private InetAddress group;

private int port;

public MulticastClient(final String id, InetAddress group, int port) throws IOException {

this.id = id;

this.group = group;

this.port = port;

socket = new MulticastSocket(port);

socket.joinGroup(group);

final byte[] buf = new byte[256];

new Thread(new Runnable() {

@Override

public void run() {

while (true) {

try {

DatagramPacket packet = new DatagramPacket(buf, buf.length);

socket.receive(packet);

String msg = new String(packet.getData());

System.out.println("客户端" + id + "接受到的数据：" + msg);

} catch (IOException e) {

e.printStackTrace();

}

}

}

}).start();

}

public void send(String msg) {

byte[] bytes = msg.getBytes();

try {

socket.leaveGroup(group);//发送的人不接收组播消息

msg = "客户端" + id + "发送数据：" + msg;

System.out.println(msg);

socket.send(new DatagramPacket(bytes, bytes.length, group, port));

socket.joinGroup(group);

} catch (IOException e) {

e.printStackTrace();

}

}

}

public static void main(String[] args) throws UnknownHostException, IOException, InterruptedException {

MulticastClient mcA = new MulticastClient("A", InetAddress.getByName(MULTICAST\_ADDRESS), PORT);

MulticastClient mcB = new MulticastClient("B", InetAddress.getByName(MULTICAST\_ADDRESS), PORT);

MulticastClient mcC = new MulticastClient("C", InetAddress.getByName(MULTICAST\_ADDRESS), PORT);

mcA.send("Hello!");

TimeUnit.SECONDS.sleep(2);

mcB.send("Hello!");

TimeUnit.SECONDS.sleep(2);

mcC.send("Hello!");

TimeUnit.SECONDS.sleep(2);

}

}