网络编程

- 概念

网络编程可以让程序与网络上的其他设备中的程序进行数据交互

- 网络通信基本模式

常见的通信模式有如下2种形式：Client-Server(CS) 、 Browser/Server(BS)
c/s: 需要程序员开发实现客户端和服务端，并且需要客户下载安装客户端
b/s: 需要程序员开发实现服务端，并且需要客户下载安装浏览器

网络通信三要素

- 概念

IP地址：设备在网络中的地址，是唯一的标识。

端口：应用程序在设备中唯一的标识。

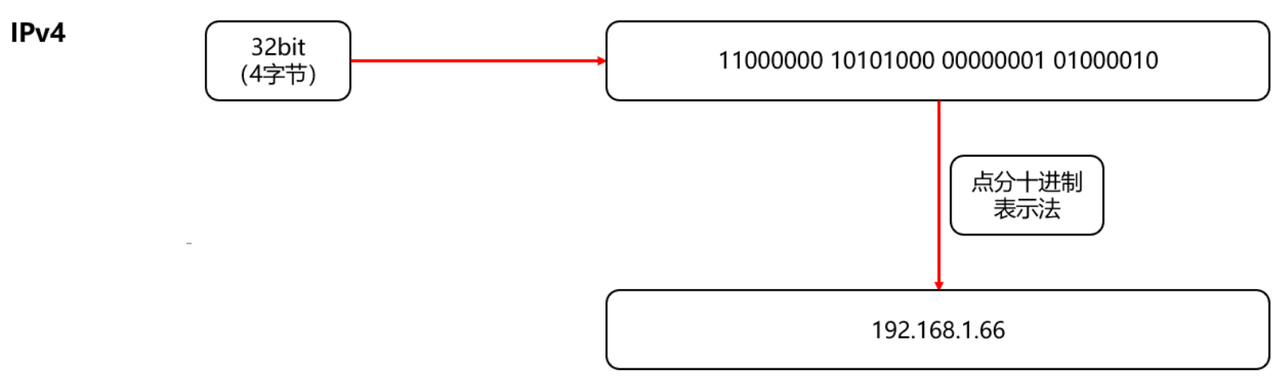
协议:数据在网络中传输的规则，常见的协议有UDP协议和TCP协议

- IP地址

IP（Internet Protocol）：全称"互联网协议地址"，是分配给上网设备的唯一标志。
常见的IP分类为：IPv4和IPv6

- IPV4

目前正在使用的网络协议地址，4字节，32位；采用点分十进制表示法



- IPV6

16个字节，128位，冒分十六进制，分为八个整数用： 隔开

- **IP**地址基本寻路

- **IP**地址形式：

公网地址、和私有地址(局域网使用)。
192.168. 开头的就是常见的局域网地址，范围即为192.168.0.0--192.168.255.255，专门为组织机构内部使用

- **IP**常用命令：

ipconfig：查看本机IP地址
ping IP地址：检查网络是否连通

- **特殊**IP地址

本机IP: 127.0.0.1或者localhost：称为回送地址也可称本地回环地址，只会寻找当前所在本机

- 小结

说说网络通信至少需要几个要素
　IP、端口、协议。
IP地址是做什么的，具体有几种
　定位网络上的设备的，有IPv4 , IPv6.
如何查看本机IP地址，如何看是否与对方互通
　ipcofig
　ping 192.168.10.23
本机IP是谁?
　127.0.0.1或者是localhost

IP地址操作类-InetAddress

InetAddress API如下

名称	说明
public static InetAddress getLocalHost()	返回本主机的地址对象
public static InetAddress getByName(String host)	得到指定主机的IP地址对象，参数是域名或者IP地址
public String getHostName()	获取此IP地址的主机名
public String getHostAddress()	返回IP地址字符串
public boolean isReachable(int timeout)	在指定毫秒内连通该IP地址对应的主机，连通返回true

- 使用

```
    public static void main(String[] args) throws Exception {
        final InetAddress localHost = InetAddress.getLocalHost();
        System.out.println("该IP的主机名：" + localHost.getHostName());
        System.out.println("该IP值：" + localHost.getHostAddress());
        System.out.println("InetAddress实例：" + localHost);
        System.out.println("----------------------------");
        final InetAddress baidu = InetAddress.getByName("www.baidu.com");
        System.out.println("InetAddress实例(百度)：" + baidu);
        System.out.println("该IP的主机名：" + baidu.getHostName());
        System.out.println("该IP值：" + baidu.getHostAddress());
        System.out.println("----------------------------");
        final InetAddress baidu2 = InetAddress.getByName("14.215.177.39");
        System.out.println("InetAddress实例(百度)：" + baidu2);
        System.out.println("该IP的主机名：" + baidu2.getHostName());
        System.out.println("----------------------------");
        System.out.println(baidu.isReachable(5000));
    }
```

```
该IP的主机名：LAPTOP-KH07R9VQ
该IP值：192.168.175.1
InetAddress实例：LAPTOP-KH07R9VQ/192.168.175.1
----------------------------
InetAddress实例(百度)：www.baidu.com/14.215.177.39
该IP的主机名：www.baidu.com
该IP值：14.215.177.39
----------------------------
InetAddress实例(百度)：/14.215.177.39
该IP的主机名：14.215.177.39
----------------------------
true
```

- 小结

IP地址的代表类是谁?
　InetAddress类
如何获取本机IP对象
　public static InetAddress getLocalHost()
如何判断与该IP地址对象是否互通?
　public boolean isReachable(int timeout)

要素二：端口号

- 概念

标识正在计算机设备上运行的进程（程序），被规定为一个 16 位的二进制，范围是 0~65535

- 端口类型

周知端口：0~1023，被预先定义的知名应用占用（如：HTTP占用 80，FTP占用21）

注册端口：1024~49151，分配给用户进程或某些应用程序。（如：Tomcat占 用8080，MySQL占用3306）

动态端口：49152到65535，之所以称为动态端口，是因为它 一般不固定分配某种进程，而是动态分配。

- 注意

我们自己开发的程序选择注册端口，且一个设备中不能出现两个程序的端口号一样，否则出错
但同一个应用，在不同设备上的端口应该是要一致的

要素三：协议

- **通信协议**

连接和通信数据的规则被称为网络通信协议

- **网络通信协议有两套参考模型**

OSI参考模型：世界互联协议标准，全球通信规范，由于此模型过于理想化，未能在因特网上进行广泛推广。
TCP/IP参考模型(或TCP/IP协议)：事实上的国际标准；实际应用中

OSI参考模型	TCP/IP参考模型	各层对应	面向操作
应用层 表示层 会话层	应用层	HTTP、FTP、DNS、SMTP...	应用程序需要关注的：浏览器， 邮箱。程序员一般在这一层开发
传输层	传输层	TCP、UDP...	选择使用的TCP，UDP协议
网络层	网络层	IP、ICMP...	封装源和目标IP，进行路径选择
数据链路层 物理层	数据链路层+物理	物理寻址、比特流...	物理设备中传输

- **传输层的**2个常见协议

TCP(Transmission Control Protocol)：传输控制协议
UDP(User Datagram Protocol)：用户数据报协议

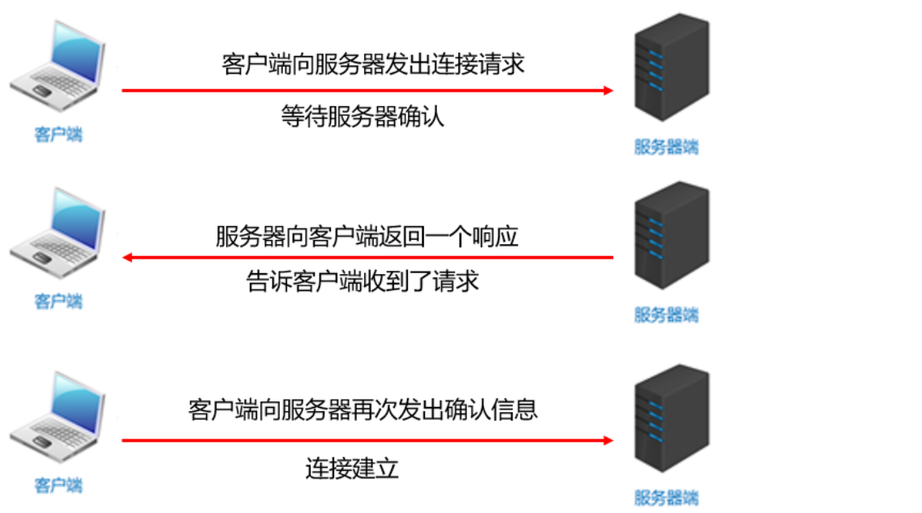
- **TCP**协议特点

使用TCP协议，必须双方先建立连接，它是一种面向连接的可靠通信协议。
传输前，采用"三次握手"方式建立连接，所以是可靠的 。
在连接中可进行大数据量的传输 。
连接、发送数据都需要确认，且传输完毕后，还需释放已建立的连接，通信效率较低

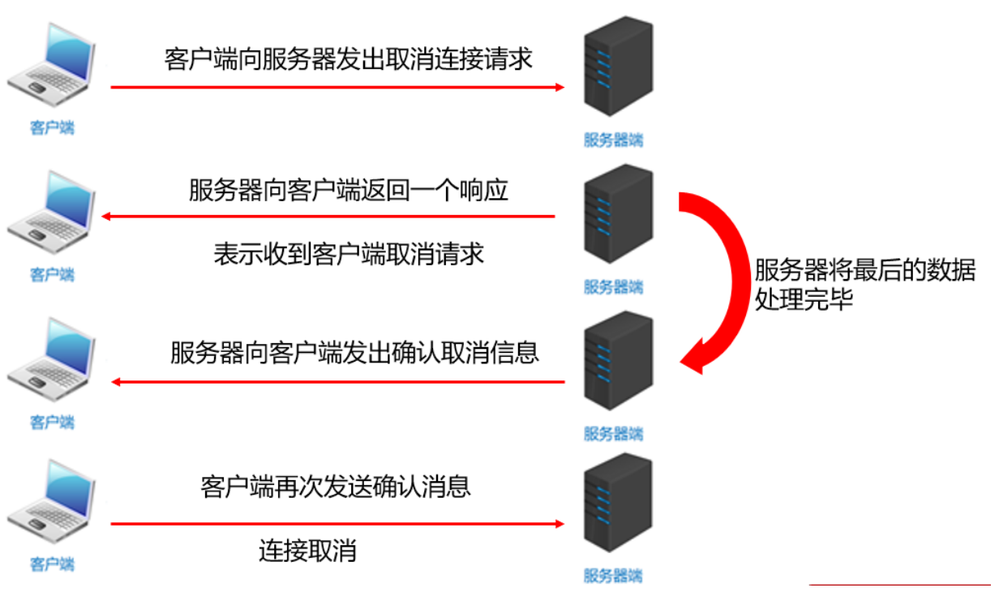
- **TCP**协议通信场景

对信息安全要求较高的场景，例如：文件下载、金融等数据通信。

TCP三次握手确立连接



TCP四次挥手断开连接



- UDP协议

UDP是一种无连接、不可靠传输的协议。
将数据源IP、目的地IP和端口封装成数据包，不需要建立连接
每个数据包的大小限制在64KB内
发送不管对方是否准备好，接收方收到也不确认，故是不可靠的
可以广播发送，发送数据结束时无需释放资源，开销小，速度快

- **UDP**协议通信场景

语音通话，视频会话等

- 小结

通信协议是什么?
　计算机网络中，连接和通信数据的规则被称为网络通信协议。

TCP通信协议的特点是什么样的?
　它是一种面向连接的可靠通信协议。
　传输前，采用“三次握手”方式建立连接，点对点的通信，所以可靠。
　在连接中可进行大数据量的传输。
　通信效率较低。

UDP协议的特点是什么
　用户数据报协议(User Datagram Protocol)
　UDP是面向无连接，不可靠传输的通信协议。
　速度快，有大小限制一次最多发送64K，数据不安全，易丢失数据。

UDP通信

实现一收一发

- UDP协议特点

UDP是一种无连接、不可靠传输的协议。
将数据源IP、目的地IP和端口以及数据封装成数据包，大小限制在64KB内，直接发送出去即可。

- UDP的数据包对象

DatagramPacket：数据包对象

构造器	说明
public DatagramPacket(byte[] buf, int length, InetAddress address, int port)	创建发送端数据包对象 buf：要发送的内容，字节数组 length：要发送内容的字节长度 address：接收端的IP地址对象 port：接收端的端口号
public DatagramPacket(byte[] buf, int length)	创建接收端的数据包对象 buf：用来存储接收的内容 length：能够接收内容的长度

- 数据包的常用方法

DatagramPacket常用方法

方法	说明
public int getLength()	获得实际接收到的字节个数

DatagramSocket：发送端和接收端对象

构造器	说明
public DatagramSocket()	创建发送端的Socket对象，系统会随机分配一个端口号。
public DatagramSocket(int port)	创建接收端的Socket对象并指定端口号

- 收发端常用方法

DatagramSocket类成员方法

方法	说明
public void send(DatagramPacket dp)	发送数据包
public void receive(DatagramPacket p)	接收数据包

- 使用UDP通信实现步骤

客户端实现步骤
　1创建DatagramSocket对象（发送端对象）
　2创建DatagramPacket对象封装需要发送的数据（数据包对象）
　3使用DatagramSocket对象的send方法传入DatagramPacket对象
　4释放资源

接收端实现步骤
　1创建DatagramSocket对象并指定端口（接收端对象）
　2创建DatagramPacket对象接收数据（数据包对象）
　3使用3DatagramSocket对象的receive方法传入DatagramPacket对象
　4释放资源

```java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

/**
 * ClassName: ClientDemo1
 * Description:客户端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ClientDemo1 {
    public static void main(String[] args) throws Exception {
        System.out.println("=============客户端启动=============");
        final DatagramSocket socket = new DatagramSocket();//发送端使用默认端口
        final byte[] data = "韭菜无敌，冲冲冲！！！！".getBytes();
        /*
         public DatagramPacket(byte buf[], int length,
                               InetAddress address, int port)
             参数一:要发送的数据，需要转化为字节数组
             参数二:字节数组的长度，可以指定发送数据的长度
             参数三:向那个ip的主机发送数据
             参数四:向哪个主机的进程(端口号)发送数据
        指定向主机ip为localhost，端口为1111的进程发送，字节长度为data.length的数组
        * */
        final DatagramPacket packet = new DatagramPacket(data, data.length, InetAddress.getLocalHost(),
1111);
        socket.send(packet);
         socket.close();
    }
}
```

```
=============客户端启动=============
```

```java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

/**
 * ClassName: ServerDemo1
 * Description:服务端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ServerDemo1 {
    public static void main(String[] args) throws Exception {
        System.out.println("=============服务端启动=============");
        final DatagramSocket socket = new DatagramSocket(1111);//服务端使用指定端口
        final byte[] data = new byte[1024 * 64];//udp数据包默认64kb
        final DatagramPacket packet = new DatagramPacket(data, data.length);//新建一个数据包用来接数据
        socket.receive(packet);

        //韭菜无敌，冲冲冲！！！！  ///、/、/、、/、/、/一堆utf-8没有对应的字符
//        System.out.println(new String(data, 0, data.length));//数组的长度是数组有多长就取多长（出事）

        final int length = packet.getLength();//通过数据包获取数据长度
        System.out.println(new String(data, 0, length));//一收一发 实现

        System.out.println(packet.getAddress());//获取对方的ip
        System.out.println(packet.getPort());//获取对方端口
        System.out.println(packet.getSocketAddress());//获取对方端口和ip
         socket.close();
    }
}
```

```
=============服务端启动=============
韭菜无敌，冲冲冲！！！！
/192.168.175.1
57397
/192.168.175.1:57397
```

- 小结

UDP发送端和接收端的对象是哪个?
  public DatagramSocket()：创建发送端的Socket对象
  public DatagramSocket(int port)：创建接收端的Socket对象

数据包对象是哪个?
  DatagramPacket

如何发送、接收数据包
  使用DatagramSocket的如下方法：
  public void send(DatagramPacket dp)：发送数据包
  public void receive(DatagramPacket dp)：接收数据包

UDP通信：多发多收

- 代码实现

```java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Objects;
import java.util.Scanner;

/**
 * ClassName: ClientDemo1
 * Description:客户端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
```

```
 */
public class ClientDemo2 {
    public static void main(String[] args) throws Exception {
        System.out.println("=============客户端启动=============");

        //当指定端口号，想要将这个这个程序开两个进程时，第二个会报错 java.net.BindException: Address already in use: Cannot bind
//        final DatagramSocket socket = new DatagramSocket(5555);

        final DatagramSocket socket = new DatagramSocket();//使用无参构造，使用注册端口四万多到65536
        final Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.print("请说：");
            final String line = sc.nextLine();
            if (Objects.equals("exit", line)) {
                System.out.println("告辞");
                socket.close();
                break;
            }
            final byte[] data = line.getBytes();//字节数组，toCharArray();是字符数组

            //指定想要连接的主机和端口号
            final DatagramPacket packet = new DatagramPacket(data, data.length, InetAddress.getLocalHost(), 1111);

            socket.send(packet);
        }
    }
}
```

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;

/**
 * ClassName: ServerDemo1
 * Description:服务端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ServerDemo2 {
    public static void main(String[] args) throws Exception {
        System.out.println("=============服务端启动=============");
        final DatagramSocket socket = new DatagramSocket(1111);//服务端使用指定端口
        final byte[] data = new byte[1024 * 64];//udp数据包默认64kb
        final DatagramPacket packet = new DatagramPacket(data, data.length);//新建一个数据包用来接数据【可以重复利用的工具包】
        while (true) {
            socket.receive(packet);
            final int length = packet.getLength();//接完数据后才能知到包中数据的长度
            System.out.println("收到来自[" + packet.getSocketAddress() + "]的消息：" + new String(data, 0, length));//取多少，输出多少
        }

    }
}
```

```
 =============服务端启动=============
收到来自[/192.168.175.1:50997]的消息：说你妹夫
收到来自[/192.168.175.1:50996]的消息：狗看了都摇头
```

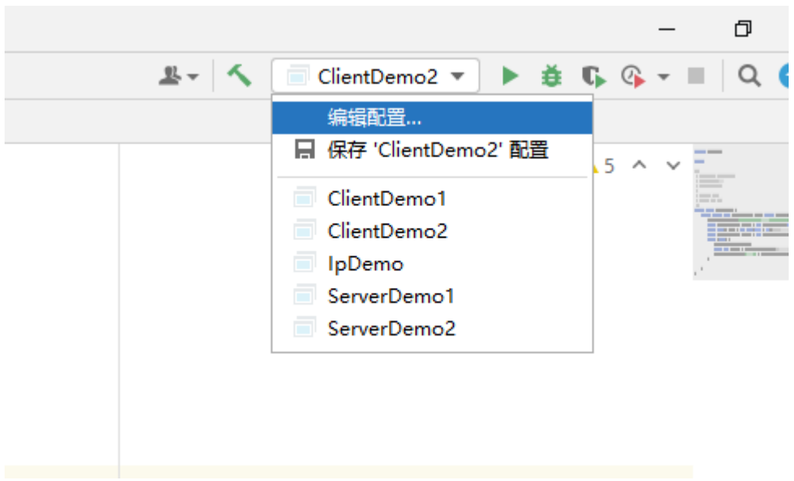
```
=============客户端启动=============
请说：狗看了都摇头
请说：
```

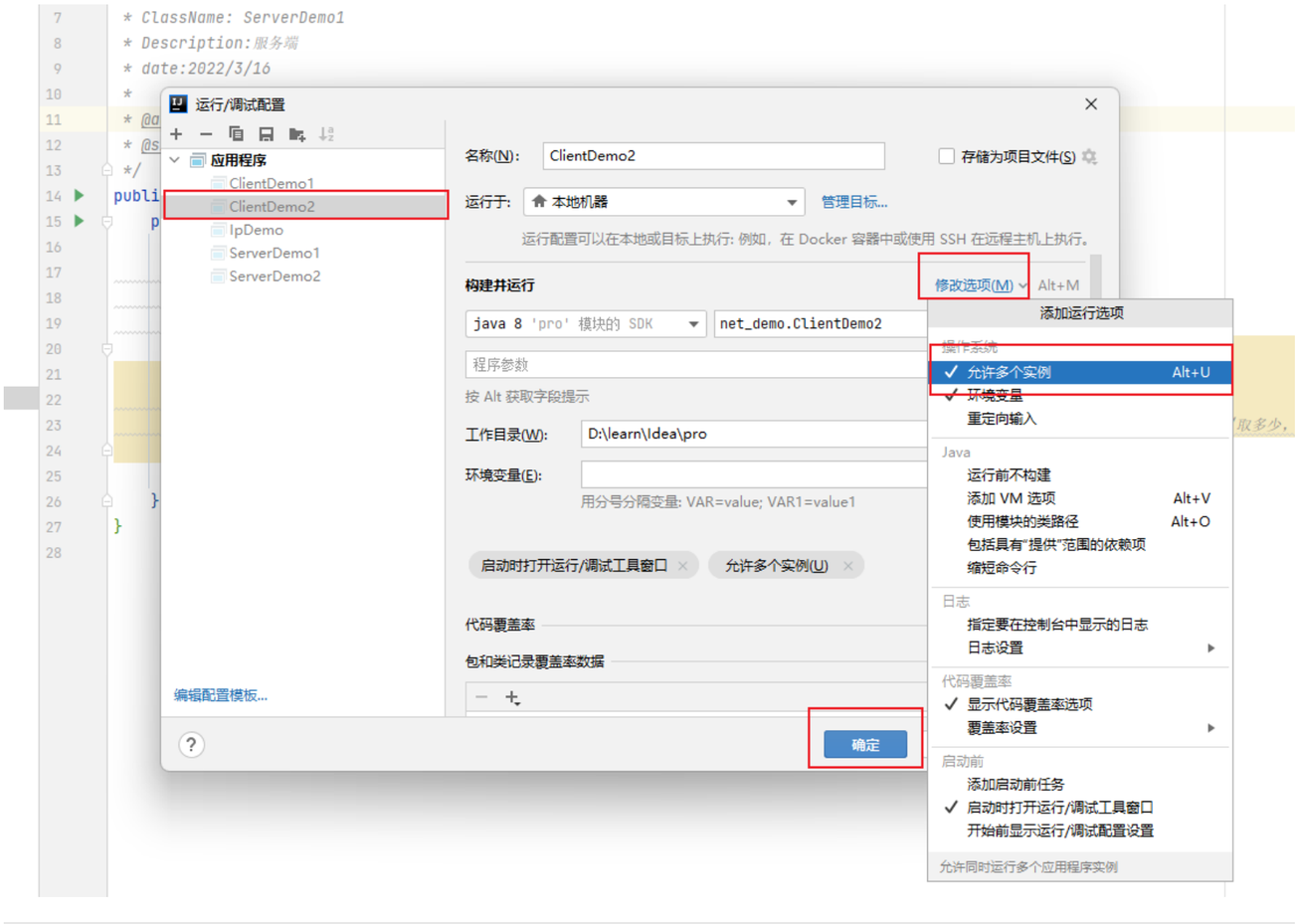
```
=============客户端启动=============
请说：说你妹夫
请说：
```

- UDP的接收端为什么可以接收很多发送端的消息

接收端只负责接收数据包，无所谓是哪个发送端的数据包

程序多开



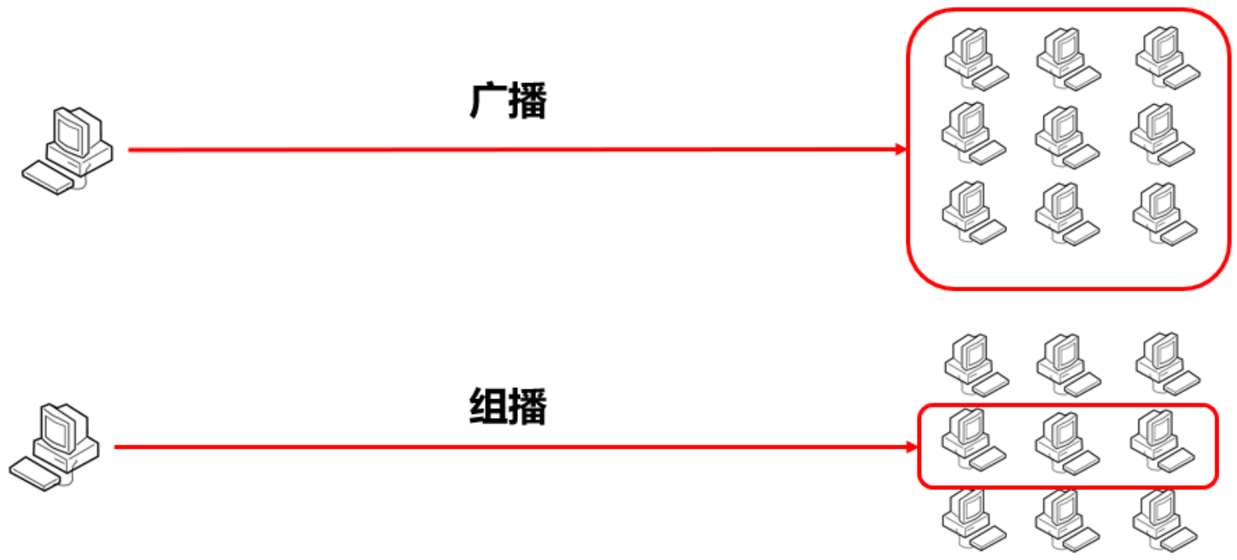


UDP通信广播、组播

- UDP的三种通信方式

单播：单台主机与单台主机之间的通信。
组播：当前主机与选定的一组主机的通信。
广播：当前主机与所在网络中的所有主机通信

- 形式



- **UDP如何实现广播**

发送端发送的数据包的目的地写的是广播地址、且指定端口。（255.255.255.255 ， 9999）
本机所在网段的其他主机的程序只要注册对应端口就可以收到消息了。（9999）

- 代码实现

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Objects;
import java.util.Scanner;

/**
 * ClassName: ClientDemo1
 * Description:客户端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ClientDemo3 {
    public static void main(String[] args) throws Exception {
        System.out.println("=============客户端启动=============");

        final DatagramSocket socket = new DatagramSocket();//使用无参构造，使用注册端口四万多到65536
        final Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.print("请说：");
            final String line = sc.nextLine();
            if (Objects.equals("exit", line)) {
                System.out.println("告辞");
                socket.close();
                break;
            }
            final byte[] data = line.getBytes();//字节数组，toCharArray();是字符数组
```

```
            //指定端口号，使用代表广播的地址
            final DatagramPacket packet = new DatagramPacket(data, data.length,
InetAddress.getByName("255.255.255.255"), 9999);
            socket.send(packet);
        }
    }
}
```

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;

/**
 * ClassName: ServerDemo1
 * Description:服务端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ServerDemo3 {
    public static void main(String[] args) throws Exception {
        System.out.println("=============服务端启动=============");
        final DatagramSocket socket = new DatagramSocket(9999);//服务端使用指定端口
        final byte[] data = new byte[1024 * 64];//udp数据包默认64kb
        final DatagramPacket packet = new DatagramPacket(data, data.length);//新建一个数据包用来接数据
        while (true) {
            socket.receive(packet);
            final int length = packet.getLength();//接完数据后才能知到包中数据的长度
            System.out.println("收到来自[" + packet.getSocketAddress() + "]的消息：" + new String(data, 0,
length));//取多少，输出多少
        }

    }
}
```

```
=============服务端启动=============
收到来自[/192.168.152.1:59460]的消息：122
收到来自[/192.168.152.1:59460]的消息：傻狗
```

```
=============客户端启动=============
请说：122
请说：傻狗
请说：
```

- **UDP如何实现组播**

发送端的数据包的目的地是组播IP（例如：224.0.1.1，端口：9999)
使用组播地址：224.0.0.0 ~ 239.255.255.255
接收端必须绑定该组播IP(224.0.1.1)，端口还要注册发送端的目的端口9999 ，这样即可接收该组播消息。
只有被服务端添加进组的组播地址，客户端一起作为目的主机地址才能被服务器接收到；
DatagramSocket的子类MulticastSocket可以在接收端绑定组播IP

- 组播代码实现

```
import javax.xml.soap.SOAPMessage;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.MulticastSocket;

/**
 * ClassName: ServerDemo1
 * Description:服务端 组播
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ServerDemo4 {
    public static void main(String[] args) throws Exception {
        System.out.println("=============服务端启动=============");
        final MulticastSocket socket = new MulticastSocket(9999);//服务端使用指定端口
        socket.joinGroup(InetAddress.getByName("224.0.1.1"));
        socket.joinGroup(InetAddress.getByName("224.0.1.2"));
//        socket.joinGroup(InetAddress.getByName("224.0.0.3"));
        final byte[] data = new byte[1024 * 64];//udp数据包默认64kb
        final DatagramPacket packet = new DatagramPacket(data, data.length);//新建一个数据包用来接数据
        while (true) {
            socket.receive(packet);
            final int length = packet.getLength();//接完数据后才能知到包中数据的长度
            System.out.println("收到来自[" + packet.getSocketAddress() + "]的消息：" + new String(data, 0,
length));//取多少，输出多少
        }

    }
}
```

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Objects;
import java.util.Scanner;

/**
 * ClassName: ClientDemo1
 * Description:组播
 * 开三个进程去连客户端，224.0.1.1 224.0.1.2 224.0.1.3【客户端没有添加组】
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ClientDemo4 {
    public static void main(String[] args) throws Exception {
        System.out.println("=============客户端启动=============");
```

```
        final DatagramSocket socket = new DatagramSocket();//使用无参构造，使用注册端口四万多到65536
        final Scanner sc = new Scanner(System.in);
        while (true) {
            System.out.print("请说：");
            final String line = sc.nextLine();
            if (Objects.equals("exit", line)) {
                System.out.println("告辞");
                socket.close();
                break;
            }
            final byte[] data = line.getBytes();//字节数组，toCharArray();是字符数组
            //
            final DatagramPacket packet = new DatagramPacket(data, data.length,
InetAddress.getByName("224.0.1.2"), 9999);
            socket.send(packet);
        }
    }
}
```

```
=============服务端启动=============
收到来自[/192.168.152.1:49990]的消息：我你么个
收到来自[/192.168.152.1:57577]的消息：deal late no doubt
```

```
=============客户端启动=============
请说：132134654  //这个的地址是224.0.1.3，服务器没有添加组，所以客户端发了没用
请说：
```

```
=============客户端启动=============
请说：我你么个
请说：
```

```
=============客户端启动=============
请说：deal late no doubt
请说：
```

- 小结

如何实现广播，具体怎么操作?
　发送端目的IP使用广播IP： 255.255.255.255 9999。
　所在网段的其他主机对应了端口（9999）即可接收消息。
如何实现组播，具体怎么操作?
　发送端目的IP使用组播IP，且指定端口。
　所在网段的其他主机注册了该组播IP和对应端口即可接收消息

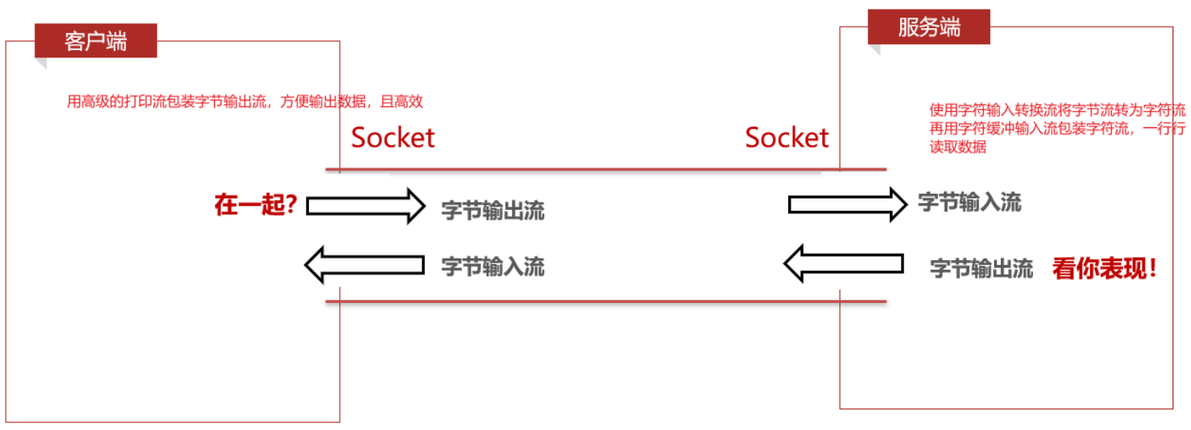
TCP通信快速入门

编写客户端代码

- **TCP协议回顾**

TCP是一种面向连接，安全、可靠的传输数据的协议
传输前，采用"三次握手"方式，点对点通信，是可靠的
在连接中可进行大数据量的传输(面向连接)

TCP通信模式演示：



- 注意

在java中只要是使用java.net.Socket类实现通信，底层即是使用了TCP协议

Socket

构造器	说明
public Socket(String host , int port)	创建发送端的Socket对象与服务端连接，参数为服务端程序的ip和端口。

Socket类成员方法

方法	说明
OutputStream getOutputStream()	获得字节输出流对象
InputStream getInputStream()	获得字节输入流对象

ServerSocket(服务端)

构造器	说明
`public ServerSocket(int port)`	注册服务端端口

ServerSocket类成员方法

方法	说明
`public Socket accept()`	等待接收客户端的Socket通信连接 连接成功返回Socket对象与客户端建立端到端通信

- 一收一发代码实现

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.*;

/**
 * ClassName: ServerDemo1
 * Description:TCP连接服务端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ServerDemo5 {
    public static void main(String[] args) throws Exception {
        System.out.println("------------SERVER RUNNING----------------");
        final ServerSocket serverSocket = new ServerSocket(2222);

        //等待客户端向这里发送建立连接请求
        //得到一个socket对象，建立点对点连接
        final Socket socket = serverSocket.accept();
        final InputStream is = socket.getInputStream();
        final BufferedReader br = new BufferedReader(new InputStreamReader(is));
        String msg;

        // java.net.SocketException: Connection reset
 /*       while ((msg = br.readLine()) != null) {//这里是读一行，但是客户端发送完数据后没有换行；所以处于等待状态，客户端跑完了，这里就会外抛异常
            System.out.println(msg);
        }*/

        if ((msg = br.readLine()) != null) {
            System.out.println(socket.getRemoteSocketAddress() + " 发送了: " + msg);
        }
    }
}
```

```
import java.io.OutputStream;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.util.Objects;
import java.util.Scanner;

/**
 * ClassName: ClientDemo1
 * Description: TCP连接客户端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ClientDemo5 {
    public static void main(String[] args) throws Exception {
        try {
            System.out.println("=============客户端启动=============");

            /*
               public Socket(String host, int port)
               服务端主机和端口
            * */
            final Socket socket = new Socket("127.0.0.1", 2222);
            final OutputStream os = socket.getOutputStream();
            final PrintStream ps = new PrintStream(os);

            //需要换行，因为读取是使用BufferedReader读一行；没有读完，服务端会等着，但此时客户端已经跑完了；
            // 又由于TCP是面向连接的，其中一端没了，另一端会直接外抛连接重置异常
//            ps.print("我是一个调理农务系的学生");
            ps.println("我是一个调理农务系的学生");//这里换行了，服务端可以接收到这行数据；但因为使用了while循环，这里又快快跑完了，服务端又会外抛异常
            ps.flush();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
------------SERVER RUNNING----------------
/127.0.0.1:50942 发送了：我是一个调理农务系的学生
```

```
=============客户端启动=============
```

- 实现步骤

服务端实现接收消息
需求：服务端实现步骤
创建ServerSocket对象，注册服务端端口。
调用ServerSocket对象的accept()方法，等待客户端的连接，并得到Socket管道对象。
通过Socket对象调用getInputStream()方法得到字节输入流、完成数据的接收。
释放资源：关闭socket管道

客户端发送消息
需求：客户端实现步骤
创建客户端的Socket对象，请求与服务端的连接。
使用socket对象调用getOutputStream()方法得到字节输出流。
使用字节输出流完成数据的发送。
释放资源：关闭socket管道。

- 小结

TCP通信的客户端的代表类是谁?
Socket类
public Socket(String host , int port)

TCP通信如何使用Socket管道发送、接收数据。
OutputStream getOutputStream()：获得字节输出流对象（发）
InputStream getInputStream()：获得字节输入流对象（收）

TCP通信服务端用的代表类?
ServerSocket类,注册端口。
调用accept()方法阻塞等待接收客户端连接。得到Socket对象。

TCP通信的基本原理?
客户端怎么发，服务端就应该怎么收。
客户端如果没有消息，服务端会进入阻塞等待。
Socket一方关闭或者出现异常、对方Socket也会失效或者出错

TCP通信多发多收消息

- 一个客户端与一个服务端的多发多收

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.*;

/**
 * ClassName: ServerDemo1
 * Description:TCP连接服务端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ServerDemo5 {
    public static void main(String[] args) throws Exception {
        System.out.println("------------SERVER RUNNING----------------");
        final ServerSocket serverSocket = new ServerSocket(2222);

        //等待客户端向这里发送建立连接请求
        //得到一个socket对象，建立点对点连接
        final Socket socket = serverSocket.accept();
        final InputStream is = socket.getInputStream();
        final BufferedReader br = new BufferedReader(new InputStreamReader(is));
        String msg;
        while ((msg = br.readLine()) != null) {
            System.out.println(socket.getRemoteSocketAddress() + " 发送消息:" + msg);
        }

    /*    if ((msg = br.readLine()) != null) {
            System.out.println(socket.getRemoteSocketAddress() + " 发送了：" + msg);
        }*/
    }
}
```

```
import java.io.OutputStream;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.util.Objects;
import java.util.Scanner;

/**
 * ClassName: ClientDemo1
 * Description: TCP连接客户端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ClientDemo5 {
    public static void main(String[] args) throws Exception {
        try {
```

```
            System.out.println("=============客户端启动=============");

            /*
               public Socket(String host, int port)
               服务端主机和端口
            * */
            final Socket socket = new Socket("127.0.0.1", 2222);
            final OutputStream os = socket.getOutputStream();
            final PrintStream ps = new PrintStream(os);

            while (true) {
                final Scanner sc = new Scanner(System.in);
                System.out.print("请说:");
                ps.println(sc.nextLine());
                ps.flush();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
=============客户端启动=============
请说:莫听传林打叶声
请说:和方孝祥且徐行
请说:天王盖地虎
请说:
```

```
------------SERVER RUNNING----------------
/127.0.0.1:50317 发送消息:莫听传林打叶声
/127.0.0.1:50317 发送消息:和方孝祥且徐行
/127.0.0.1:50317 发送消息:天王盖地虎
```

- 本案例实现了多发多收，那么是否可以同时接收多个客户端的消息

不可以的。
因为服务端现在只有一个线程，只能与一个客户端进行通信

- 小结

本次多发多收是如何实现的
　客户端使用循环反复地发送消息。
　服务端使用循环反复地接收消息。
现在服务端为什么不可以同时接收多个客户端的消息。
　目前服务端是单线程的，每次只能处理一个客户端的消息

TCP通信同时接受多个客户端消息

- 问题引入

之前我们的通信是否可以同时与多个客户端通信，为什么?
　不可以的
　单线程每次只能处理一个客户端的Socket通信
2、如何才可以让服务端可以处理多个客户端的通信需求?
　引入多线程

- 代码实现

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * ClassName: ServerDemo1
 * Description:TCP连接服务端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ServerDemo6 {
    public static void main(String[] args) throws Exception {
        System.out.println("------------SERVER RUNNING----------------");
        //注册端口
        final ServerSocket serverSocket = new ServerSocket(2222);
        //定义一个死循环负责不断接收客户端的socket连接（由main线程负责）
        while (true) {
            final Socket socket = serverSocket.accept();
            //客户端下线提醒
            System.out.println(socket.getRemoteSocketAddress() + " 他来了，他来了！！");
            new ServerReaderThread(socket).start();
        }
    }
}
```

```
class ServerReaderThread extends Thread {

    private Socket socket;

    public ServerReaderThread(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            final InputStream is = socket.getInputStream();
            final BufferedReader br = new BufferedReader(new InputStreamReader(is));
            String msg;
```

```
            while ((msg = br.readLine()) != null) {
                System.out.println(socket.getRemoteSocketAddress() + " 发送消息:" + msg);
            }
        } catch (IOException e) {
            //客户端下线提醒
            System.out.println(socket.getRemoteSocketAddress() + " 它走了！！");
        }
    }
}
```

```
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.util.Scanner;

/**
 * ClassName: ClientDemo1
 * Description: TCP连接客户端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ClientDemo6 {
    public static void main(String[] args) throws Exception {
        try {
            System.out.println("=============客户端启动=============");

            final Socket socket = new Socket("127.0.0.1", 2222);
            final OutputStream os = socket.getOutputStream();
            final PrintStream ps = new PrintStream(os);

            while (true) {
                final Scanner sc = new Scanner(System.in);
                System.out.print("请说:");
                ps.println(sc.nextLine());
                ps.flush();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
------------SERVER RUNNING----------------
/127.0.0.1:64658 他来了，他来了！！
/127.0.0.1:64658 发送消息:1
/127.0.0.1:64658 发送消息:告辞!
/127.0.0.1:64658 它走了！！
```

```
=============客户端启动=============
请说:1
请说:告辞!
请说:
```

- 小结

本次是如何实现服务端接收多个客户端的消息的。
主线程定义了循环负责接收客户端Socket管道连接
每接收到一个Socket通信管道后分配一个独立的线程负责处理它。

TCP通信使用线程池优化

- 目前的架构存在什么问题

客户端与服务端的线程模型是： N-N的关系。
客户端并发越多，系统瘫痪的越快

```
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.util.Scanner;

/**
 * ClassName: ClientDemo1
 * Description: TCP连接客户端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ClientDemo7 {
    public static void main(String[] args) throws Exception {
        try {
            System.out.println("=============客户端启动=============");

            final Socket socket = new Socket("127.0.0.1", 2222);
            final OutputStream os = socket.getOutputStream();
            final PrintStream ps = new PrintStream(os);

            while (true) {
                final Scanner sc = new Scanner(System.in);
                System.out.print("请说:");
                ps.println(sc.nextLine());
                ps.flush();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
}
```

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.*;

/**
 * ClassName: ServerDemo1
 * Description:TCP连接服务端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ServerDemo7 {

    private static ExecutorService pool = new ThreadPoolExecutor(2, 3, 6, TimeUnit.SECONDS
            , new ArrayBlockingQueue<>(2), Executors.defaultThreadFactory(), new
ThreadPoolExecutor.AbortPolicy());

    public static void main(String[] args) throws Exception {
        System.out.println("------------SERVER RUNNING----------------");
        final ServerSocket serverSocket = new ServerSocket(2222);
        while (true) {
            final Socket socket = serverSocket.accept();
            System.out.println(socket.getRemoteSocketAddress() + " 他来了");
            pool.execute(new DealServerThread(socket));
        }

    }
}
```

```
class DealServerThread implements Runnable {
    private Socket socket;

    public DealServerThread(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            final InputStream is = socket.getInputStream();
            final BufferedReader br = new BufferedReader(new InputStreamReader(is));
            String msg;
            while ((msg = br.readLine()) != null) {
                System.out.println(socket.getRemoteSocketAddress() + " 发送消息:" + msg);
            }
        } catch (Exception e) {
            System.out.println(socket.getRemoteSocketAddress() + " 他下线了！！");
        }
    }
}
```

```
------------SERVER RUNNING----------------
/127.0.0.1:65169 他来了
/127.0.0.1:65169 发送消息:客户端1，核心线程1
/127.0.0.1:65177 他来了
/127.0.0.1:65177 发送消息:客户端2，核心线程2
/127.0.0.1:65187 他来了    //进入了等待队列
/127.0.0.1:65195 他来了    //进入了等待队列
/127.0.0.1:65202 他来了
/127.0.0.1:65202 发送消息:客户端5 //分配一个临时线程
/127.0.0.1:65213 他来了
Exception in thread "main" java.util.concurrent.RejectedExecutionException: Task
net_demo.DealServerThread@29453f44 rejected from java.util.concurrent.ThreadPoolExecutor@5cad8086[Running,
pool size = 3, active threads = 3, queued tasks = 2, completed tasks = 0]
    at java.util.concurrent.ThreadPoolExecutor$AbortPolicy.rejectedExecution(ThreadPoolExecutor.java:2063)
    at java.util.concurrent.ThreadPoolExecutor.reject(ThreadPoolExecutor.java:830)
    at java.util.concurrent.ThreadPoolExecutor.execute(ThreadPoolExecutor.java:1379)
    at net_demo.ServerDemo7.main(ServerDemo7.java:29)//所有线程在忙，队列已满，还未达到最大线程，分配一个临时线程
/127.0.0.1:65169 发送消息:客户端下线，归还线程
/127.0.0.1:65169 他下线了！！
/127.0.0.1:65187 发送消息:客户端3
/127.0.0.1:65177 发送消息:客户端2下线，归还线程
/127.0.0.1:65177 他下线了！！
/127.0.0.1:65195 发送消息:客户端4
进程已结束，退出代码为 -1
```

```
=============客户端启动=============
请说:客户端1，核心线程1
请说:客户端下线，归还线程
请说:
进程已结束，退出代码为 -1
```

```
=============客户端启动=============
请说:客户端2，核心线程2
请说:客户端2下线，归还线程
请说:
进程已结束，退出代码为 -1
```

```
=============客户端启动=============
请说:客户端3
请说:
进程已结束，退出代码为 -1
```

```
==========客户端启动==========
请说:客户端4
请说:
进程已结束，退出代码为 -1
```

```
==========客户端启动==========
请说:客户端5
请说:
进程已结束，退出代码为 -1
```

```
==========客户端启动==========
请说:客户端6
请说:
进程已结束，退出代码为 -1
```

- 小结

本次使用线程池的优势在哪里?
服务端可以复用线程处理多个客户端，可以避免系统瘫痪。
适合客户端通信时长较短的场景。

TCP通信实战案例即时通信

- 即时通信是什么含义，要实现怎么样的设计

即时通信，是指一个客户端的消息发出去，其他客户端可以接收到。
之前我们的消息都是发给服务端的。
即时通信需要进行端口转发的设计思想。

- TCP广播实现

```
import java.io.*;
import java.net.Socket;
import java.util.Scanner;

/**
 * ClassName: ClientDemo1
 * Description: TCP连接客户端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ClientDemo8 {
    public static void main(String[] args) throws Exception {
        try {
            System.out.println("==========客户端启动==========");
            final Socket socket = new Socket("127.0.0.1", 2222);
            //开启一个线程用来接收消息
            new DealClientThread(socket).start();

            //通过主线程来发送消息
            final OutputStream os = socket.getOutputStream();
            final PrintStream ps = new PrintStream(os);
            final Scanner sc = new Scanner(System.in);
            while (true) {
                System.out.print("请说:");
                ps.println(sc.nextLine());
                System.out.println();
                ps.flush();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
class DealClientThread extends Thread {
    private Socket socket;

    public DealClientThread(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            final InputStream is = socket.getInputStream();
            final InputStreamReader isr = new InputStreamReader(is);
            final BufferedReader br = new BufferedReader(isr);
            String msg;
            while ((msg = br.readLine()) != null) {
                System.out.println("收到广播消息：" + msg);
            }
        } catch (IOException e) {
            System.out.println("服务端把你移除群聊！！");
        }
    }
}
```

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
```

```java
/**
 * ClassName: ServerDemo1
 * Description:TCP连接服务端
 * date:2022/3/16
 *
 * @author fgcy
 * @since JDK 1.8
 */
public class ServerDemo8 {
    public static List<Socket> allSockets = new ArrayList<>();

    public static void main(String[] args) throws Exception {
        System.out.println("------------SERVER RUNNING----------------");
        final ServerSocket serverSocket = new ServerSocket(2222);
        while (true) {
            final Socket socket = serverSocket.accept();
            System.out.println(socket.getRemoteSocketAddress() + " 上线了！！");
            allSockets.add(socket);//上线
            new MyServerDeal(socket).start();
        }
    }
}
```

```java
class MyServerDeal extends Thread {
    private Socket socket;

    public MyServerDeal(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            final InputStream is = socket.getInputStream();
            final BufferedReader br = new BufferedReader(new InputStreamReader(is));
            String msg;
            while ((msg = br.readLine()) != null) {
                System.out.println(socket.getRemoteSocketAddress() + " 发送消息:" + msg);
                //去除当前自己的管道，不给自己发消息
                ServerDemo8.allSockets.remove(socket);
                sendToAllSockets(msg);
                //将自己的管道再次添加进去
                ServerDemo8.allSockets.add(socket);
            }
        } catch (IOException e) {
            ServerDemo8.allSockets.remove(socket);
            System.out.println(socket.getRemoteSocketAddress() + "下线了！！！");
        }
    }

    private void sendToAllSockets(String msg) {
        ServerDemo8.allSockets.forEach(socket1 -> {
            try {
                final OutputStream os = socket1.getOutputStream();
                final PrintStream pw = new PrintStream(os);//要用字节输出打印流
                pw.println(msg);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }
}
```

```
------------SERVER RUNNING----------------
/127.0.0.1:50201 上线了！！
/127.0.0.1:50207 上线了！！
/127.0.0.1:50207 发送消息:1
/127.0.0.1:50201 发送消息:2
```

```
=============客户端启动=============
请说:收到广播消息：1
2

请说:
进程已结束，退出代码为 -1
```

```
=============客户端启动=============
请说:1
请说:收到广播消息：2

进程已结束，退出代码为 -1
```

- 小结

即时通信是什么含义，要实现怎么样的设计?
即时通信，是指一个客户端的消息发出去，其他客户端可以接收到
即时通信需要进行端口转发的设计思想。
服务端需要把在线的Socket管道存储起来
一旦收到一个消息要推送给其他管道

TCP通信实战案例模拟BS系统

- 问题引出

之前的客户端都是什么样的
　其实就是CS架构，客户端实需要我们自己开发实现的。
BS结构是什么样的，需要开发客户端吗?
　浏览器访问服务端，不需要开发客户端

- **HTTP响应数据的协议格式：就是给浏览器显示的网页信息**

- **实现BS开发**（多线程）

```
package net_bs;

import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;

public class BSDemo {
    public static void main(String[] args) {
        try {
            final ServerSocket serverSocket = new ServerSocket(5421);
            while (true) {
                final Socket socket = serverSocket.accept();
                new DealBSThread(socket).start();
            }

        } catch (IOException e) {
        }
    }
}
```

```
class DealBSThread extends Thread {
    private Socket socket;

    public DealBSThread(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            final PrintStream ps = new PrintStream(socket.getOutputStream());
            ps.println("HTTP/1.1 200 OK");//请求头 协议类型和版本相应信息
            ps.println("Content-Type:text/html;charset=UTF-8");//请求行 相应格式类型字符集
            ps.println();//请求空行
            ps.println("<h1 style='color:red'>调理农务系的</h1>");
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
<h1 style='color:red'>调理农务系的</h1>//http请求格式被浏览器解析，剩下的是文本数据
```

- BS代码实现（线程池）

```
import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.*;

public class BSDemo2 {
    private static ExecutorService pool = new ThreadPoolExecutor(3, 5, 6, TimeUnit.SECONDS,
            new ArrayBlockingQueue<>(2), Executors.defaultThreadFactory(), new
ThreadPoolExecutor.AbortPolicy());

    public static void main(String[] args) {
        try {
            final ServerSocket serverSocket = new ServerSocket(5555);
            while (true) {
                final Socket socket = serverSocket.accept();
                pool.execute(new MyServerTask(socket));
            }

        } catch (IOException e) {
        }
    }
}
```

```
class MyServerTask implements Runnable {
    private Socket socket;

    public MyServerTask(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            final PrintStream ps = new PrintStream(socket.getOutputStream());
            ps.println("HTTP/1.1 200 OK");
            ps.println("Content-Type:text/html;charset=UTF-8");
            ps.println();
            ps.println("<div style='background-color:grey;color:white;margin-top:200px'>个个个等待</div>");
```

```
            System.out.println(socket.getRemoteSocketAddress() + "访问");
            ps.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
/0:0:0:0:0:0:0:1:58354访问//浏览器发出一次请求，显示了四次访问
/0:0:0:0:0:0:0:1:58355访问
/0:0:0:0:0:0:0:1:58356访问
/0:0:0:0:0:0:0:1:58358访问
```

- 小结

TCP通信如何实现BS请求网页信息回来呢?
客户端使用浏览器发起请求（不需要开发客户端）
服务端必须按照浏览器的协议规则响应数据。
浏览器使用什么协议规则呢?
HTTP协议（简单了解下）