网络编程

1 Socket

1.1 IP

1.1.1 什么是IP

IP是根据TCP/IP协议划定,由32位二进制数组成,而且在因特网上是唯一的数值

例如:某台计算机,连上网的IP是:

11010101 01001001 11110000 11001100

为了便于记忆,会将这32位二进制数,每8位一组,每段之间用小数点分割

11010101.01001001.11110000.11001100

再将每八位转化为十进制

213.73.240.204

1.1.2 如何查看自己电脑的ip

1. 按win+R, 输入cmd,打开dos窗口
2. 在dos窗口输入ipconfig

```
C:\Users\Administrator>ipconfig

Windows IP 配置


以太网适配器 以太网:

   连接特定的 DNS 后缀 . . . . . : tarena.net
   本地链接 IPv6 地址. . . . . : fe80::410d:3232:f44c:5c06%16
   IPv4 地址 . . . . . : 192.168.66.177
   子网掩码 . . . . . : 255.255.255.0
   默认网关. . . . . : 192.168.66.1
```

1.2 什么是端口号

- 端口号(port number)就是计算机为了给每个网络程序分配一个独一无二的区别符,有了这些端口号,就可以准确定位到具体的程序。
- 端口号是个整数,范围0-65535,分为周知端口和动态端口
  - 周知端口就是众所周知的端口,是端口号中的明星,本身的存在就是有自身用途,这些端口我们一般不使用,范围是0-1023。
  - 动态端口,剩下的端口号都是动态端口,动态端口的意思就是将这些端口动态的分配给每个需要端口号的程序,当开启一个程序时,就分配给它一个端口

2 Socket概述

在计算机领域中,Socket也被称为套接字编程,它是计算机之间进行通信的一种约定或者说是一种方式。

应用程序可以通过它发送或者接收数据,可以对其发送过来的内容像处理文件一样,打开、关闭或者读写等操作,套接字允许应用程序将I/O插入到互联网上,并与网络中的其他程序进行通信。

2.1 Socket常用方法

2.1.1 服务器端ServerSocket

在服务器端选择一个端口号,然后在指定的端口号上等待客户端发起的连接

构造方法:

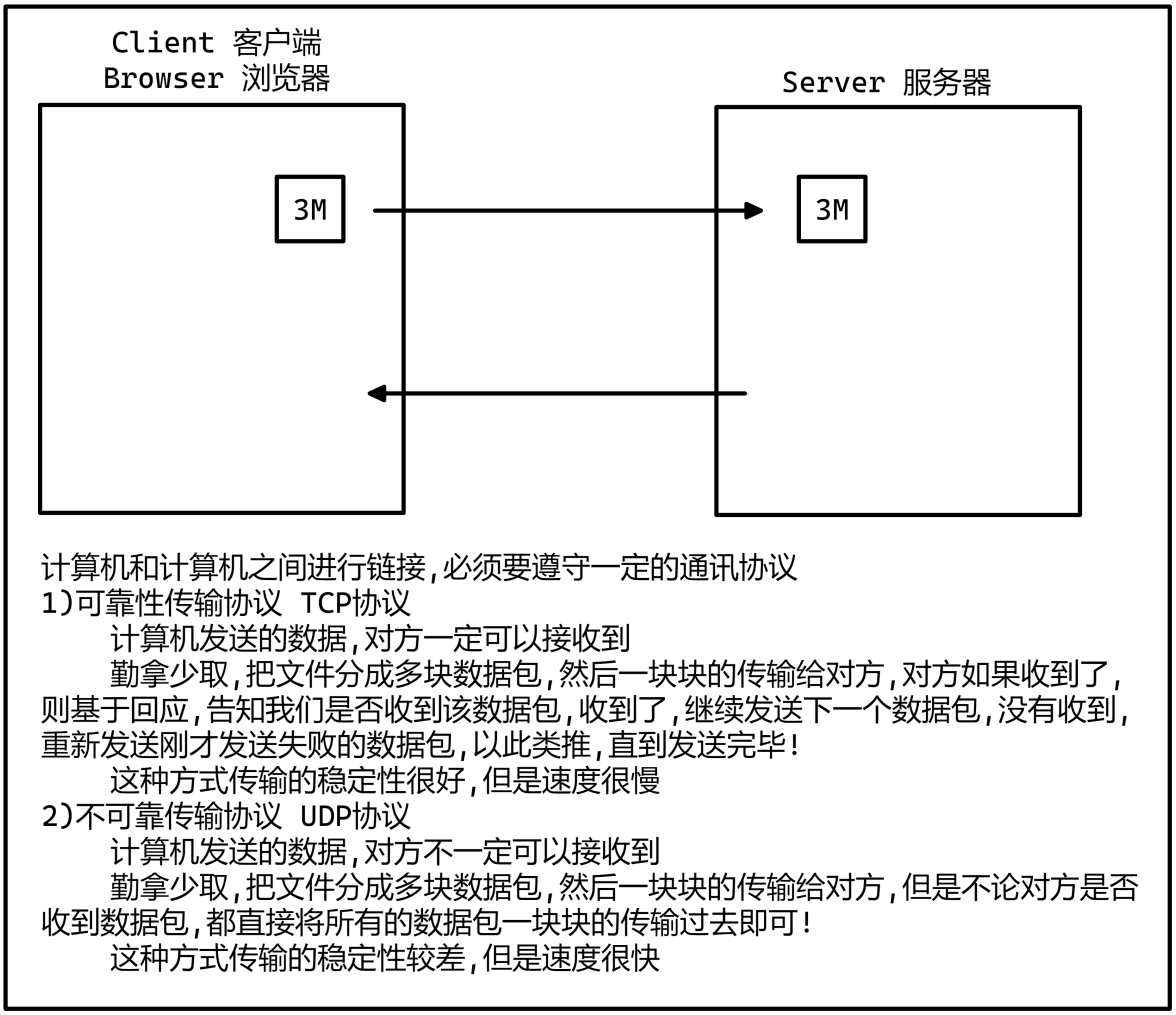
```
ServerSocket(int port) 创建一个绑定特定端口号的服务器套接字
accept() 侦听并接受到发送到此套接字的连接
close() 关闭此套接字
```

2.1.2 客户端Socket

构造方法:

```
Socket(String host,int port) 创建一个套接字,并且连接到host,并且绑定端口号
close() 关闭此套接字
getInputStream() 返回此套接字的输入流
getOutputStram() 返回此套接字的输出流
```

3 聊天室



计算机和计算机之间进行链接,必须要遵守一定的通讯协议
1)可靠性传输协议 TCP协议
　　计算机发送的数据,对方一定可以接收到
　　勤拿少取,把文件分成多块数据包,然后一块块的传输给对方,对方如果收到了,则基于回应,告知我们是否收到该数据包,收到了,继续发送下一个数据包,没有收到,重新发送刚才发送失败的数据包,以此类推,直到发送完毕!
　　这种方式传输的稳定性很好,但是速度很慢
2)不可靠传输协议 UDP协议
　　计算机发送的数据,对方不一定可以接收到
　　勤拿少取,把文件分成多块数据包,然后一块块的传输给对方,但是不论对方是否收到数据包,都直接将所有的数据包一块块的传输过去即可!
　　这种方式传输的稳定性较差,但是速度很快

3.1 初始Client

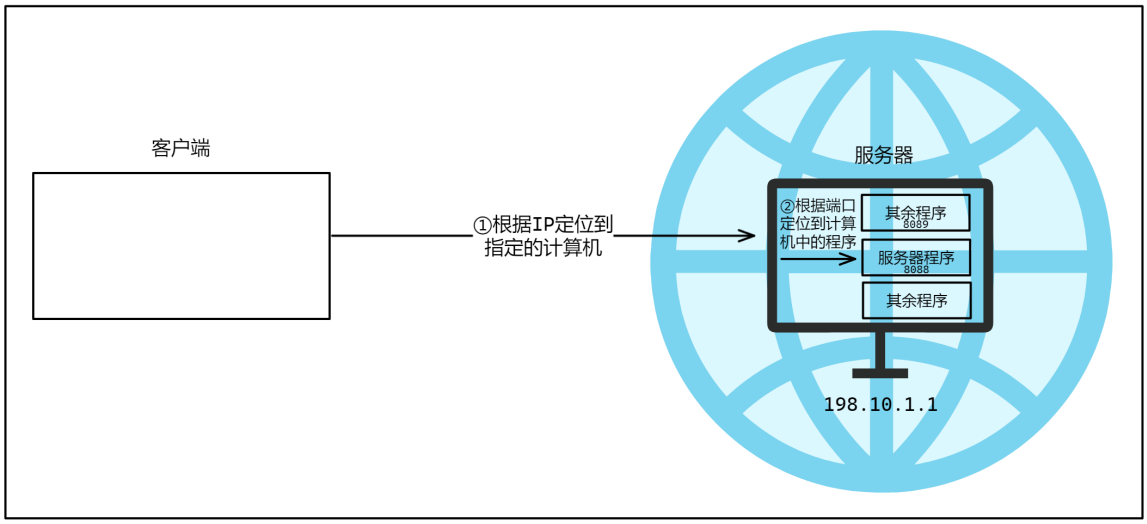
1. 创建socket包,并在包下创建Client类和Server类
2. 在Client类中定义如下内容

- **构造器**的作用主要是用于初始化客户端的内容,主要是用于做前期准备
- **start方法**的作用主要是用于执行客户端的工作逻辑
- **main方法**中只用于书写流程调用内容

```
package cn.tedu.socket;

/**
 * 聊天室的客户端
 */
public class Client {
    /**
     * 初始化客户端
     */
    public Client() {
    }

    /**
     * 客户端开始工作的方法
     */
    public void start() {

    }

    public static void main(String[] args) {
        Client client = new Client();
        client.start();
    }
}
```

3. 在Client中声明Socket实例,用于搭建TCP连接,并在构造器中实例化该socket,并指定IP和端口号

- 其中的IP可以参考1.1 IP
- 端口号可以参考1.2 什么是端口号




```
/**
 * java.net.Socket 套接字,原意是插座
```

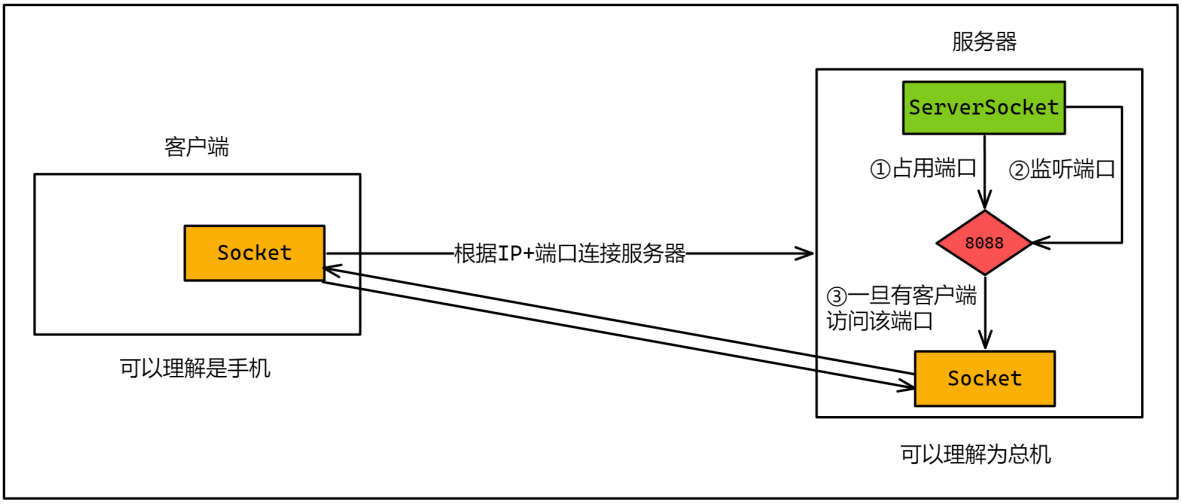
```
 * Socket封装了TCP协议的通信细节,我们使用它就可以与远端计算机建立TCP连接,并基于一堆流的IO操作完成
 * 与远端计算机的数据交互
 */
private Socket socket;

/**
 * 初始化客户端
 */
public Client() {
    try {
        System.out.println("正在连接服务器...");
        /*
         * 实例化Socket时,需要传入两个参数:
         * ①要链接的远端计算机的IP地址
         * 连接的计算机IP分为两种情况:
         *  1)如果连接的是非本机,则需要查询对方的IP地址,进行连接
         *  2)如果连接的是本机,则可以选取如下的值:
         *      i. 真实IP(通过ipconfig查询出来的)
         *      ii. 127.0.0.1(会自动映射本机的真实IP)
         *      iii. localhost(是127.0.0.1的域名)
         * ②要链接的远端计算机的端口
         */
        socket = new Socket("localhost",8088);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3.2 初始化Server

1. 仿照Client类,在Server中也定义构造器、start方法和main方法

```
package cn.tedu.socket;

/**
 * 聊天室服务器端
 */
public class Server {
    public Server() {
    }

    public void start() {

    }

    public static void main(String[] args) {
        Server server = new Server();
        server.start();
    }
}
```

2. 在Server中声明ServerSocket,并且在构造器中实例化ServerSocket,并指定占用端口号

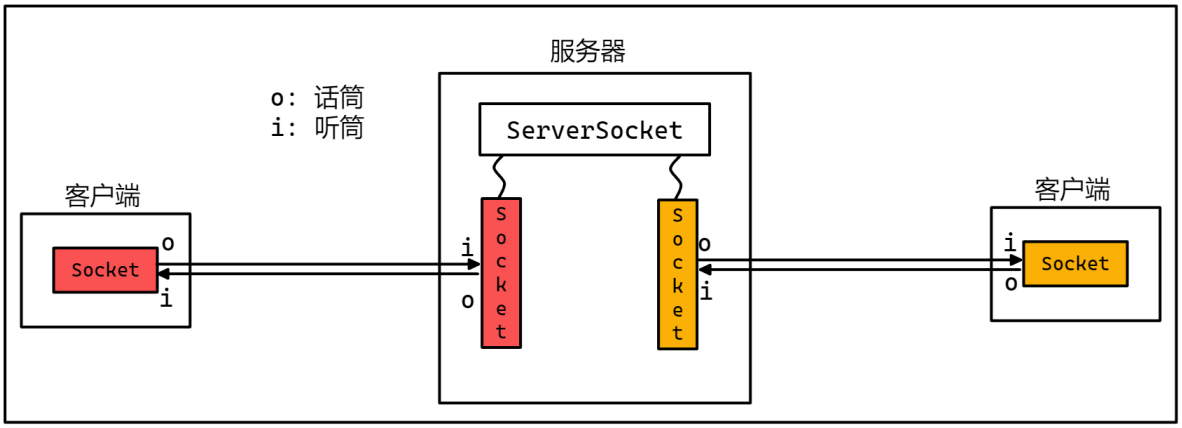



```java
/**
 * java.net.ServerSocket
 * 是运行在服务器端的,它具有以下两个作用:
 * ①打开服务器端口(占用一个端口,客户端访问服务器的端口,就是它占用)
 * ②监听该端口,一旦一个客户端访问服务器,则会立即返回一个Socket实例,并通过这个Socket实例和客户端进行交互
 */
private ServerSocket server;

public Server() {
    try {
        System.out.println("正在启动服务器...");
        /*
         * 实例化ServerSocket时,需要指定要占用的端口,而该端口就是我们Client类要连接的端口号
         */
        server = new ServerSocket(8088);
        System.out.println("服务器启动完毕!!!");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

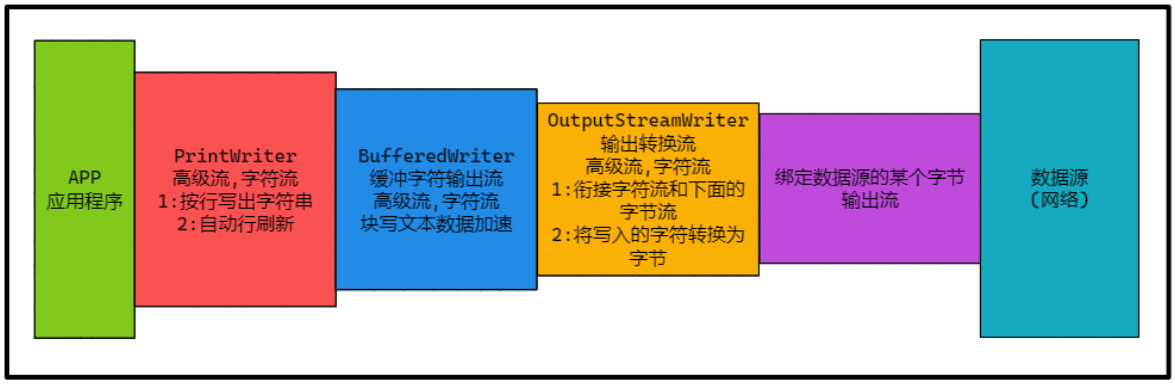
3. 在start方法中调用accept方法,监听端口

```java
public void start() {
    while (true) {
        try {
            System.out.println("等待客户端的连接...");
            /*
             * 监听当前占用的端口号,并且该方法是一个阻塞方法
             * 当端口号没有被客户端访问时,程序不会向下执行,一直阻塞在此处,
             * 当端口号被客户端访问时,会理解返回一个Socket实例,并且程序继续向下运行
             */
            Socket socket = server.accept();
            System.out.println("一个客户端连接了!!!");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3.3 客户端发一条信息

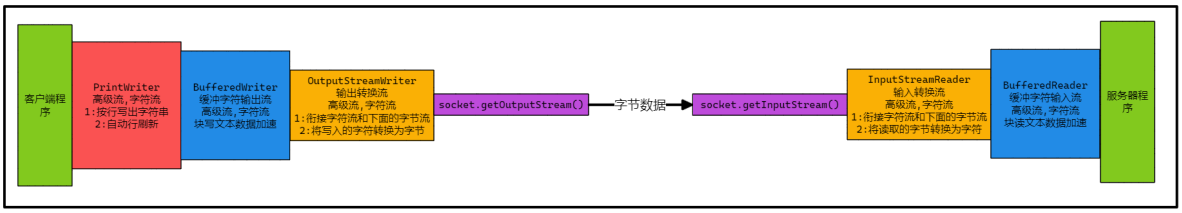


1. 在Client的start方法中建立流链接




```java
public void start() {
    try {
        /*
         * 通过Socket的getOutputStream方法
         * 获取字节输出流写出的字节会通过网络发送给远端计算机
         * 此处获取的是一个低级的字节输出流
         */
        OutputStream out = socket.getOutputStream();
        //和转换输出字符流连接,并且指定编码
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
        //和缓冲字符流连接
        BufferedWriter bw = new BufferedWriter(osw);
        //和按行写出字符流连接,并且开启自动行刷新
        PrintWriter pw = new PrintWriter(bw, true);
        pw.println("你好啊,客户端!");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

2. 在Server类中的start方法中,实现接受客户端发送的字符串功能




```java
public void start() {
    while (true) {
```

```
        try {
            System.out.println("等待客户端的连接...");
            Socket socket = server.accept();
            System.out.println("一个客户端连接了!!!");
            /*
             * Socket的getInputStream方法
             * 可以获取一个低级的字节输入流,可以读取来自远端计算机发送过来的字节数据
             */
            InputStream in = socket.getInputStream();
            //连接输入转换字符流,并指定编码
            InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
            //连接缓冲输入字符流
            BufferedReader br = new BufferedReader(isr);
            //读取客户端发送的一行字符串
            String line = br.readLine();
            System.out.println(line);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

3.4 客户端循环发送

1. 在Client类中的start方法中,仿照记事本案例,实现客户端在控制台写一行字符串,可以向服务器发送一行字符串

```
public void start() {
    try {
        OutputStream out = socket.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
        BufferedWriter bw = new BufferedWriter(osw);
        PrintWriter pw = new PrintWriter(bw, true);
        //仿照记事本实现写一行发一行
        Scanner scanner = new Scanner(System.in);
        while (true) {
            String line = scanner.nextLine();
            if ("exit".equals(line)) {
                break;
            }
            pw.println(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

2. 在Server的start方法中,循环读取客户端发送的字符串

```
public void start() {
    while (true) {
        try {
            System.out.println("等待客户端的连接...");
            Socket socket = server.accept();
            System.out.println("一个客户端连接了!!!");
            InputStream in = socket.getInputStream();
            InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
            BufferedReader br = new BufferedReader(isr);
            //循环读取客户端发送的一行字符串
            String line;
```

```
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3. 测试程序,当客户端输入"exit"时,会退出程序,但是服务器处会报如下错误:

```
java.net.SocketException Create breakpoint : Connection reset
    at java.net.SocketInputStream.read(SocketInputStream.java:210)
    at java.net.SocketInputStream.read(SocketInputStream.java:141)
```

- **原因:** 是因为基于TCP的底层协议,要求客户端断开连接时,需要和服务器挥手示意,告知服务器客户端断开了,否则就会报错

4. 无论是客户端的流资源释放,还是客户端下线了,去挥手示意,都是必须要实现的,所以适合写在start方法中的try代码块的finally中

```
public void start() {
    try {
        OutputStream out = socket.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
        BufferedWriter bw = new BufferedWriter(osw);
        PrintWriter pw = new PrintWriter(bw, true);
        Scanner scanner = new Scanner(System.in);
        while (true) {
            String line = scanner.nextLine();
            if ("exit".equals(line)) {
                break;
            }
            pw.println(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            /*
             * Socket提供的close方法
             * ①可以断开和远端计算机的连接,并挥手示意
             * ②可以将socket所连接的流进行关闭
             */
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

5. 此时再进行测试,可以发现输入exit时,服务器端不会发生异常,但是强制关闭客户端依旧还是报错,这是不可避免的,因为我们不能彻底约束客户端的行为!!!

3.5 客户端起名

1. 在Client类中声明name的全局变量

```
public class Client {
    private Socket socket;
    private static String name;
    ...
}
```

2. 然后在main方法中,为name属性赋值

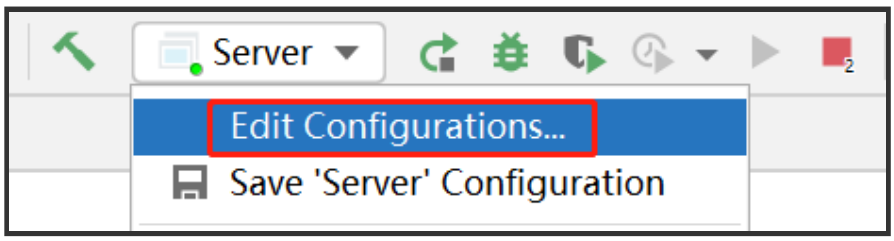
```
public static void main(String[] args) {
    System.out.println("请输入您的用户名:");
    name = new Scanner(System.in).nextLine();
    Client client = new Client();
    client.start();
}
```

3. 然后在发送字符串时,将name一同发送给服务器

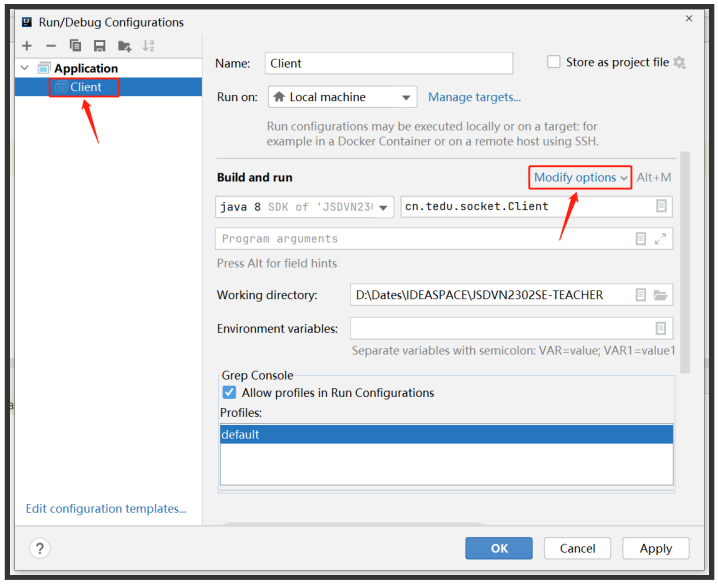
```
public void start() {
    try {
        OutputStream out = socket.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
        BufferedWriter bw = new BufferedWriter(osw);
        PrintWriter pw = new PrintWriter(bw, true);
        Scanner scanner = new Scanner(System.in);
        while (true) {
            String line = scanner.nextLine();
            if ("exit".equals(line)) {
                break;
            }
            pw.println(name + "说: " + line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3.6 客户端多开

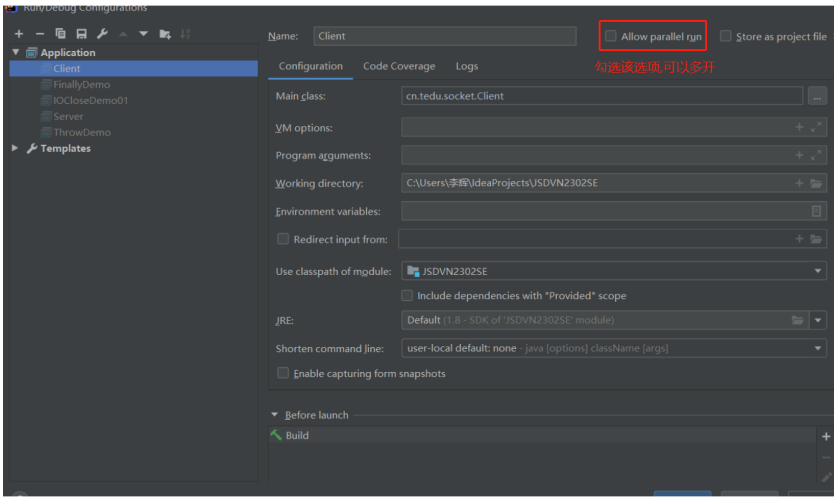
1. 点击如下内容



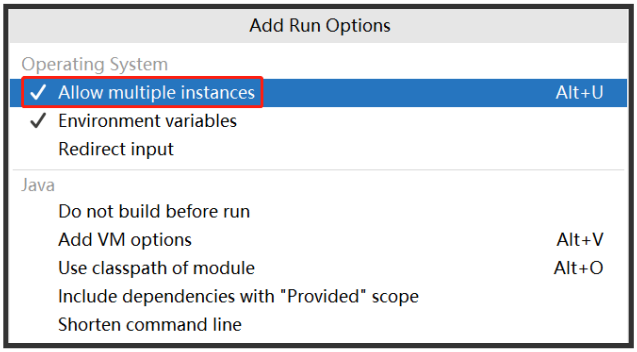
2. 在左侧栏中找到要多开的程序,然后点击右侧栏中的 `Modify options`



注意: 不同的idea,该设置会略有不同,比如如果是下图样式,可以参考如下配置

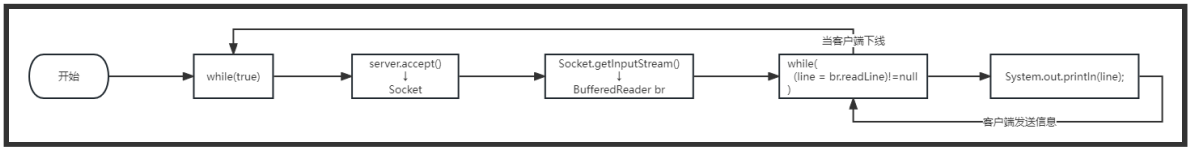


3. 勾选如下的选项,即可多开

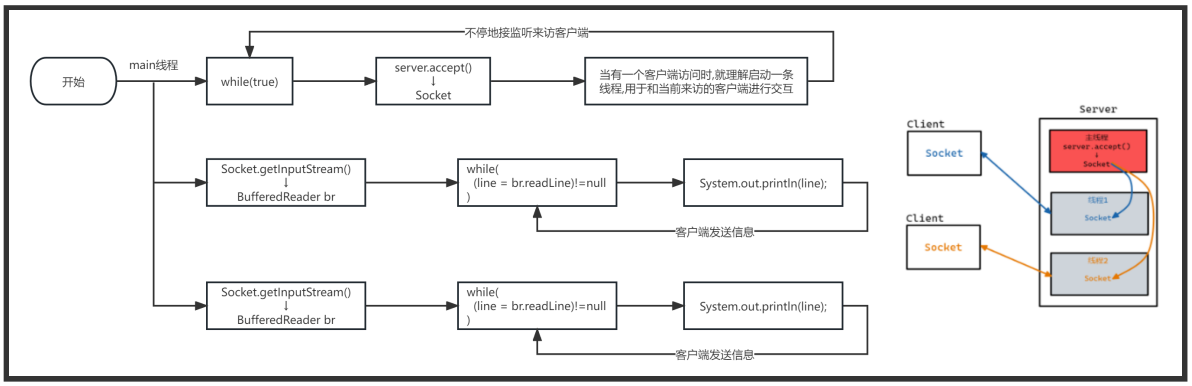


3.7 引入多线程

- 此时测试,会发现多个客户端不能同时向服务器发送内容,服务器只能同一时间和一个客户端进行交互,当多个客户端访问服务器时,需要排队
- 原因是我们目前的程序都是单线程程序(main线程),是没办法解决当前问题的



- 所以需要引入**多线程**,让主线程负责接受客户端,当客户端来访问时,再创建一条线程负责和客户端进行交互



1. 在Server类中,将start方法中和客户端进行交互的内容,移动到线程任务类的run方法中

- **注意事项**
  - ClientHandler是内部类
  - start方法中在接收一个客户端访问时,一定要创建线程,并且将对客户端生成的socket实例传递给线程任务类,此处使用的是构造器传输

①start方法

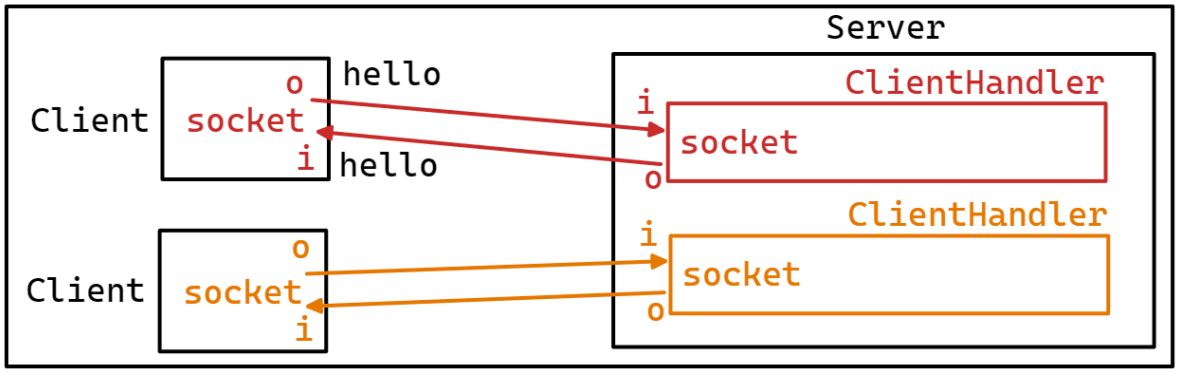
```
public void start() {
    while (true) {
        try {
            System.out.println("等待客户端的连接...");
            Socket socket = server.accept();
            System.out.println("一个客户端连接了!!!");
            //当有客户端访问时,可以创建一个线程负责和当前客户端进行交互
            ClientHandler handler = new ClientHandler(socket);
            Thread t = new Thread(handler);
            t.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

②ClientHandler类

```
/**
 * 定义线程的任务类,负责和客户端进行交互
 */
private class ClientHandler implements Runnable {
    private Socket socket;

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

    @Override
    public void run() {
        try {
            InputStream in = socket.getInputStream();
            InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
            BufferedReader br = new BufferedReader(isr);
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
```

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

3.8 服务器向客户端回信



1. 在Server类中的run方法里,定义输出流向客户端发送信息

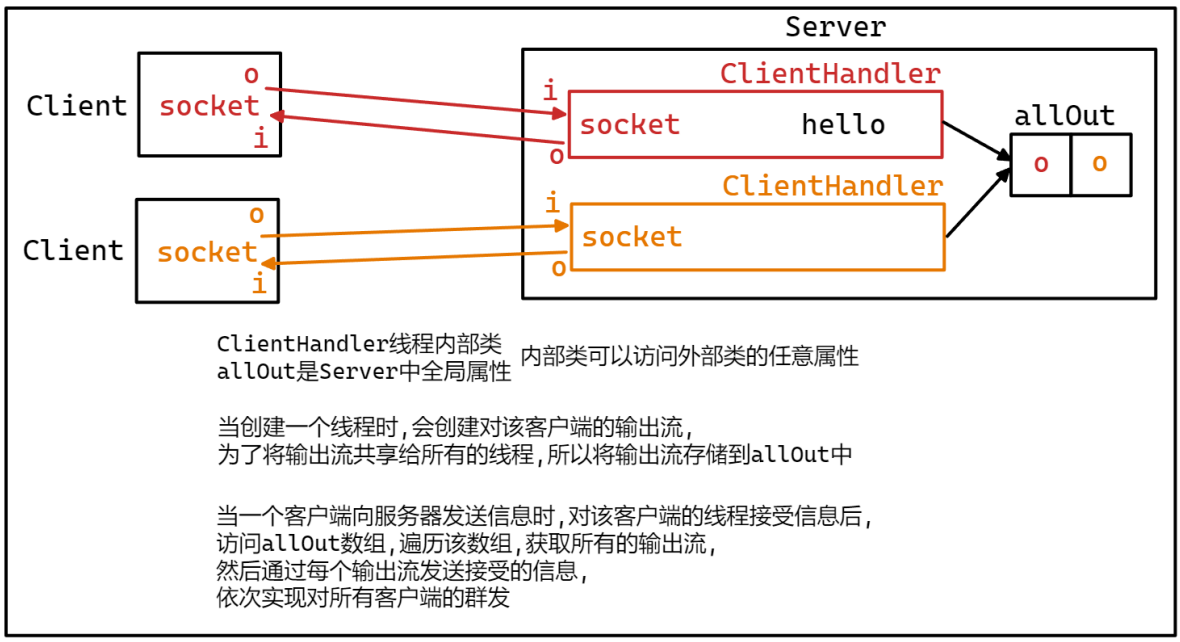
```
@Override
public void run() {
    try {
        InputStream in = socket.getInputStream();
        InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
        BufferedReader br = new BufferedReader(isr);
        //通过socket获取输出流,用于给客户端发送信息
        OutputStream out = socket.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
        BufferedWriter bw = new BufferedWriter(osw);
        PrintWriter pw = new PrintWriter(bw, true);
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
            //将客户端发送的信息回复给客户端
            pw.println(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

2. 在Client中的start方法中创建输入流,读取服务器发送的信息

```
public void start() {
    try {
        OutputStream out = socket.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
        BufferedWriter bw = new BufferedWriter(osw);
        PrintWriter pw = new PrintWriter(bw, true);
        //通过socket获取输入流读取服务器发送的信息
        InputStream in = socket.getInputStream();
        //连接输入转换字符流,并指定编码
        InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
        //连接缓冲输入字符流
```

```
        BufferedReader br = new BufferedReader(isr);
        Scanner scanner = new Scanner(System.in);
        while (true) {
            String line = scanner.nextLine();
            if ("exit".equals(line)) {
                break;
            }
            pw.println(name + "说: " + line);
            //读取服务器回复的一句话
            line = br.readLine();
            System.out.println(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3.9 实现客户端群聊



1. 在Server中添加一个**全局属性**:allOut的空数组,用于存放所有对客户端的输出流

```
private PrintWriter[] allOut = {};
```

2. 在ClientHandler中的run方法中,将对客户端的输出流,存储到allOut中,并且在收到客户端发送信息后,将信息群发给所有客户端

```
public void run() {
    try {
        InputStream in = socket.getInputStream();
        InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
        BufferedReader br = new BufferedReader(isr);
        OutputStream out = socket.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
```

```
            BufferedWriter bw = new BufferedWriter(osw);
            PrintWriter pw = new PrintWriter(bw, true);
            //将该客户端的输出流存入到共享数组allOut中
            //①对allOut扩容
            allOut = Arrays.copyOf(allOut, allOut.length + 1);
            //②将pw存储到allOut中(存储到allOut中的最后一个位置)
            allOut[allOut.length - 1] = pw;
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
                //将客户端发送的信息回复给所有客户端
                for (int i = 0; i < allOut.length; i++) {
                    allOut[i].println(line);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

3. 启动服务器,运行多个客户端,测试时,发现,多个客户端在发送信息时,出现发送一行信息,才能收到一行信息的情况,原因主要是出现在Client类中如下位置:

```
while (true) {
    String line = scanner.nextLine();
    if ("exit".equals(line)) {
        break;
    }
    pw.println(name + "说: " + line);
    //读取服务器回复的一句话
    line = br.readLine();
    System.out.println(line);
}
```

其中 `scanner.nextLine()` 这句代码是阻塞方法,只有在控制敲击回车键,程序才会继续向下执行,所以此处导致我们不在控制台敲击回车键,就不能执行 `br.readLine();` 代码,也就是不能读取服务器转发给各个客户端的信息,如果要解决该问题,需要使用线程,消除此处代码片段之间的互相影响

3.10 客户端引入多线程

1. 在Client中,创建ServerHandler负责读取服务器发送的信息,并且将start方法中读取服务器信息的内容移动到ServerHandler中的run方法内部

①ServerHandler

```
/**
 * 该线程负责读取服务器发送的信息
 */
private class ServerHandler implements Runnable {
    @Override
    public void run() {
        try {
            //通过socket获取输入流读取服务器发送的信息
            InputStream in = socket.getInputStream();
            //连接输入转换字符流,并指定编码
            InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
            //连接缓冲输入字符流
            BufferedReader br = new BufferedReader(isr);
            //创建流之后,开始循环读取服务器发送的信息
            String line;
            while ((line = br.readLine()) != null) {
```

```
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

②start方法

```
public void start() {
    try {
        OutputStream out = socket.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
        BufferedWriter bw = new BufferedWriter(osw);
        PrintWriter pw = new PrintWriter(bw, true);
        Scanner scanner = new Scanner(System.in);
        while (true) {
            String line = scanner.nextLine();
            if ("exit".equals(line)) {
                break;
            }
            pw.println(name + "说: " + line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2. 在start方法中,启动线程,并执行读取服务器信息的任务

```
//启动一个线程来读取服务器端发送的信息
ServerHandler handler = new ServerHandler();
Thread t = new Thread(handler);
t.start();
```

3. 由于服务器可能存在异常断开,会导致我们客户端报错,但是这种断开是不能控制的,所以索性就忽略就可以了,所以我们可以在ServerHandler中的run方法中捕获异常但是不需要打印异常

```
public void run() {
    try {
        InputStream in = socket.getInputStream();
        InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
        BufferedReader br = new BufferedReader(isr);
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException e) {
        //此处只有当链接异常断开时,才会触发异常,但是控制不了,所以索性就忽视该异常
        //e.printStackTrace();
    }
}
```

4. 同样在Server类中,断开连接时,也同时关闭流并且向客户端挥手示意,所以在Server中的run方法中添加finally,关闭资源

```
public void run() {
    try {
        InputStream in = socket.getInputStream();
        InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
        BufferedReader br = new BufferedReader(isr);
        OutputStream out = socket.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
        BufferedWriter bw = new BufferedWriter(osw);
        PrintWriter pw = new PrintWriter(bw, true);
        allOut = Arrays.copyOf(allOut, allOut.length + 1);
        allOut[allOut.length - 1] = pw;
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
            for (int i = 0; i < allOut.length; i++) {
                allOut[i].println(line);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

5. 启动测试,测试效果没有问题,但是目前有一点要说明,客户端启动后,会存在两个线程,一个是主线程,一个是ServerHandler的子线程,当客户端输入exit时,意味着主线程结束,但是子线程会结束吗?很明显不会,因为子线程是普通用户线程,所以我们为了实现主线程结束后,子线程也会被同步杀死,可以将子线程设置为**守护线程**,所以在启动ServerHandler线程前,将其设置为守护线程

```
ServerHandler handler = new ServerHandler();
Thread t = new Thread(handler);
//设置为守护线程
t.setDaemon(true);
t.start();
```

3.11 客户端下线

- 客户端下线包含两种情况,一种是客户端输入exit时,正常下线,另一种是直接关闭程序,总之不论是哪种方式,都需要从allOut数组中,取出对下线的客户端的输出流,所以这部分代码适合写在finally中

1. 由于我们在下线时,需要比较输出流内存地址,所以为了能够在finally中使用pw实例,所以需要将pw的作用域提高到全局

```
public void run() {
    PrintWriter pw = null;
    try {
        //此处省略
        pw = new PrintWriter(bw, true);
    }
```

2. 在finally中进行取出客户端输出流操作

> **取出思路:**
>
> **①遍历allOut数组**
> **②将要删除的元素和数组中的每一个元素进行内存地址的比较,相同的即是要删除的元素**
> **③将数组的最后一个元素替换到删除元素的位置**
> **④将数组的最后一个元素缩容**

```
public void run() {
    PrintWriter pw = null;
    try {
        //此处省略
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        //将当前客户端的输出流从allOut数组中取出
        //遍历allOut
        for (int i = 0; i < allOut.length; i++) {
            //找到要删除的元素
            if (allOut[i] == pw) {
                //将最后一个元素替换到目标元素
                allOut[i] = allOut[allOut.length - 1];
                //进行数组的缩容(将数组的最后一个元素删除)
                allOut = Arrays.copyOf(allOut, allOut.length-1);
                //由于数组中只会存储一个目标元素,所以找到目标元素取出后,就可以停止遍历了
                break;
            }
        }
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3.12 广播在线人数

- 广播通知的操作在当前案例会出现多次
  - 当用户上线时,需要广播所有客户端
  - 当用户下线时,需要广播所有客户端
  - 当一个客户端发送信息时,需要广播所有客户端

1. 在Server中的ClientHandler类中,添加一个广播通知的方法

```
/**
 * 广播通知所有客户端
 *
 * @param message 广播的信息
 */
private void sendMessage(String message) {
    for (int i = 0; i < allOut.length; i++) {
        allOut[i].println(message);
    }
}
```

2. 调用该方法

```
public void run() {
    PrintWriter pw = null;
    try {
```

```
        InputStream in = socket.getInputStream();
        InputStreamReader isr = new InputStreamReader(in,
StandardCharsets.UTF_8);
        BufferedReader br = new BufferedReader(isr);
        OutputStream out = socket.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(out,
StandardCharsets.UTF_8);
        BufferedWriter bw = new BufferedWriter(osw);
        pw = new PrintWriter(bw, true);
        allOut = Arrays.copyOf(allOut, allOut.length + 1);一个位
        allOut[allOut.length - 1] = pw;
        //广播通知所有客户端该用户上线了
        sendMessage("一个用户上线了!当前在线人数:" + allOut.length);
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
            //将客户端发送的信息回复给所有客户端
            sendMessage(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        for (int i = 0; i < allOut.length; i++) {
            if (allOut[i] == pw) {
                allOut[i] = allOut[allOut.length - 1];)
                allOut = Arrays.copyOf(allOut, allOut.length - 1);
                break;
            }
        }
        //广播通知所有客户端用户下线了
        sendMessage("一个用户下线了,当前在线人数:" + allOut.length);
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

4 常见问题

4.1 测试问题

一定要先启动服务器!!!再启动客户端!!!

- 如果服务器不启动,客户端会连接失败

```
java.net.ConnectException Create breakpoint : Connection refused: connect
    at java.net.DualStackPlainSocketImpl.connect0(Native Method)
    at java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.java:79)
```

4.2 端口号被占用问题

4.2.1 问题描述

- 当程序启动时,控制台报如下错误,一定是发送了端口号占用的问题

```
java.net.BindException Create breakpoint : Address already in use: JVM_Bind
    at java.net.DualStackPlainSocketImpl.bind0(Native Method)
    at java.net.DualStackPlainSocketImpl.socketBind(DualStackPlainSocketImpl.java:106)
    at java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java:387)
    at java.net.PlainSocketImpl.bind(PlainSocketImpl.java:190)
```

4.2.2 解决方式

①方式一: 更换端口号

注意: 如果修改了服务器的端口号,客户端连接的端口号也要一同发生改变

②方式二: 杀死现占用端口号的程序

1. 按 **WIN+R** ,敲击cmd,打开DOS窗口
2. 然后敲击如下命令,该命令会查询出占用端口号的程序

```
netstat -ano|findstr "端口号"
```

```
C:\Users\86132>netstat -ano|findstr 8088
  TCP    0.0.0.0:8088           0.0.0.0:0              LISTENING       11240
  TCP    [::]:8088              [::]:0                 LISTENING       11240
```

注意: 其中11240是当前这个程序的PID

3. 根据PID值,获取对应的程序名,来防止杀死不该杀死的程序

```
tasklist|findstr "PID值"
```

```
C:\Users\86132>tasklist|findstr 11240
java.exe                     11240 Console                    1     26,268 K
```

4. 经过检查,该程序是可以关闭的,所以关闭该程序

```
taskkill /f /pid "PID值"
```

```
C:\Users\86132>taskkill /f /pid 11240
成功: 已终止 PID 为 11240 的进程。
```