


题目介绍

✓ 题目1：将基于UDP协议的Client-Server通信程序示例的服务器端程序改造成多线程版。（4.1）

✓ 题目2：将基于TCP协议的Client-Server通信程序示例的服务器端程序改造成线程池版。（4.1）

 西安电子科技大学
XIDIAN UNIVERSITY

计算机科学与技术学院
School of Computer Science and Technology

第一次作业!

■ 准备:

安装jdk, 安装maven, 为maven设置阿里镜像仓库, 用maven编译Helloworld版的java程序

■ 题目1:

将基于UDP协议的Client-Server通信程序示例的服务器端程序改造成多线程版。

■ 题目2:

将基于TCP协议的Client-Server通信程序示例的服务器端程序改造成线程池版。

■ 提交要求:

1. 4月1日前将源程序发送至邮箱: xd_distri_comp@163.com

2. 邮件标题风格: 第1次作业+学号+姓名

3. 源程序打包文件命名方式: 第1次作业+学号+姓名.zip

1. udp协议是一种无连接, 可以直接通信的协议方式。

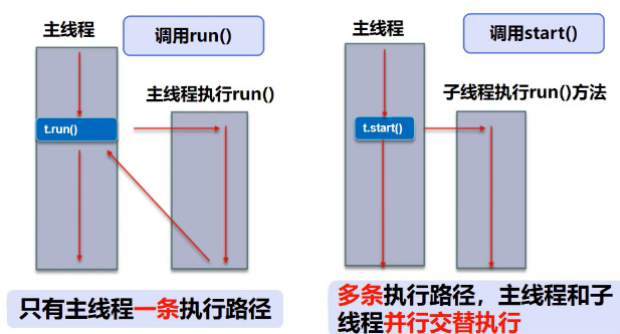
2. tcp是三次握手进行

类比

打电话: ---连接---接了---通话 TCP

发短信: ---发送了就完事了---接收 UDP

3. 多线程问题, 在网上查询了start() 和run()的区别



题目一 将基于UDP协议的Client-Server通信程序示例的服务器端程序改造成多线程版

```
EchoClient.java x UDPServer.java x UDPServerThread.java x
public class UDPServer{
    public static void main(String args[]){
        DatagramSocket aSocket = null;
        DatagramPacket request;
        try{
            aSocket = new DatagramSocket( port: 6789);
            byte[] buffer = new byte[1000];

            while(true){
                request = new DatagramPacket(buffer,buffer.length);
                aSocket.receive(request);
                UDPServerThread thread1 = new UDPServerThread(buffer, aSocket, request, number: 1);
                UDPServerThread thread2 = new UDPServerThread(buffer, aSocket, request, number: 2);
                UDPServerThread thread3 = new UDPServerThread(buffer, aSocket, request, number: 3);
                UDPServerThread thread4 = new UDPServerThread(buffer, aSocket, request, number: 4);
                thread1.start();
                thread2.start();
                thread3.start();
                thread4.start();
                System.out.println("主进程: "+ "Server receive: " + new String(buffer));
                DatagramPacket reply = new DatagramPacket(request.getData(),
                    request.getLength(), request.getAddress(), request.getPort());
                aSocket.send(reply);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public class UDPServerThread extends Thread{
    private DatagramSocket socket;
    private DatagramPacket request;
    private byte[] buffer;
    private int number;

    public UDPServerThread(byte[] buffer, DatagramSocket socket, DatagramPacket request, int number) {
        this.buffer = buffer;
        this.socket = socket;
        this.request = request;
        this.number = number;
    }

    @Override
    public void run() {
        try {
            System.out.println("线程"+number+": "+ "Server receive: " + new String(buffer));
            System.out.println();
            DatagramPacket reply = new DatagramPacket(request.getData(),
                request.getLength(), request.getAddress(), request.getPort());
            socket.send(reply);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

udpserver 创建了四个进程，进程被udpserverthread封装

运行结果来观察

```
D:\JDK\jdk8\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2020.2.3\lib\idea_rt.jar=7885:D:\IntelliJ IDEA 2020.2.3\bin" -Dfile.encoding=UTF-8
线程2: Server receive: abcdefg

线程1: Server receive: abcdefg

线程4: Server receive: abcdefg
线程3: Server receive: abcdefg

主进程: Server receive: abcdefg

EchoServerPool x EchoClient x UDPServer x UDPCClient x
D:\JDK\jdk8\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2020.2.3\lib\idea_rt.jar=7890:D:\IntelliJ IDEA 2020.2.3\bin" -Dfile.encoding=UTF-8
Reply: abcdefg

Process finished with exit code 0
```

```
EchoServerPool x EchoClient x UDPServer x UDPCClient x
D:\JDK\jdk8\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2020.2.3\lib\idea_rt.jar=7890:D:\IntelliJ IDEA 2020.2.3\bin" -Dfile.encoding=UTF-8
Reply: abcdefg

Process finished with exit code 0
```

四个线程抢占cpu资源，顺序为2-》1-》4-》3

每次运行结果都不一样

比如

```
D:\JDK\jdk8\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2020.2.3\lib\idea_rt.jar=7890:D:\IntelliJ IDEA 2020.2.3\bin" -Dfile.encoding=UTF-8
主进程: Server receive: abcdefg
线程1: Server receive: abcdefg

线程3: Server receive: abcdefg

线程2: Server receive: abcdefg

线程4: Server receive: abcdefg
```

题目二 将基于TCP协议的Client-Server通信程序示例的服务器端程序改造成线程池版

```

public class EchoServerPool {
    public static void main(String[] args) throws Exception {
        Socket clientSocket;
        // 建立连接, 创建服务
        ServerSocket listenSocket = new ServerSocket(port: 8189); // 开放端口
        System.out.println("Server listening at 8189");

        // 线程池
        ThreadPoolExecutor executor = new ThreadPoolExecutor(
            corePoolSize: 5, maximumPoolSize: 10,
            keepAliveTime: 200, TimeUnit.MILLISECONDS, new ArrayBlockingQueue<Runnable>(capacity: 5));
        clientSocket = listenSocket.accept();
        executor.execute(new MyThreadPool(name: "线程1", clientSocket));
        executor.execute(new MyThreadPool(name: "线程2", clientSocket));
        executor.execute(new MyThreadPool(name: "线程3", clientSocket));
        executor.execute(new MyThreadPool(name: "线程4", clientSocket));
        executor.shutdown();
        listenSocket.close();
    }
}

```

先是开放连接端口，等待连接。然后创建线程池，开放四个线程，让他们抢占资源。

```

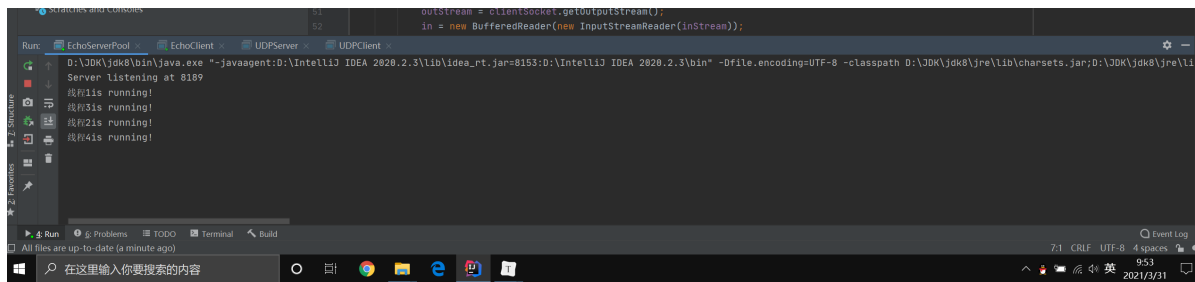
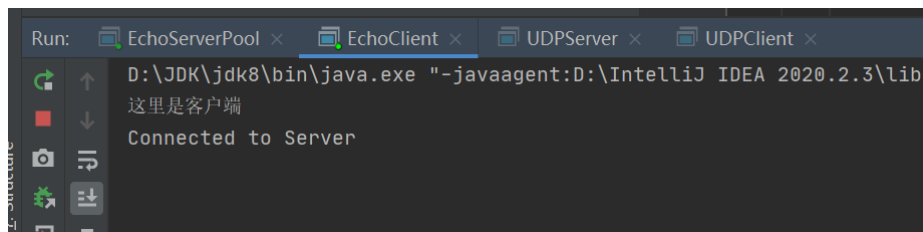
28 }
29
30 class MyThreadPool implements Runnable {
31     private String taskNum;
32     private Socket clientSocket;
33     InputStream inStream = null;
34     OutputStream outStream = null;
35     BufferedReader in = null;
36     PrintWriter out = null;
37     String line = null;
38
39     public MyThreadPool(String name, Socket clientSocket) {
40         this.taskNum = name;
41         this.clientSocket = clientSocket;
42     }
43
44     @Override
45     public void run() {
46
47         System.out.println(taskNum + "is running!");
48
49         try {
50             inStream = clientSocket.getInputStream();
51             outStream = clientSocket.getOutputStream();
52             in = new BufferedReader(new InputStreamReader(inStream));
53             out = new PrintWriter(outStream);
54
55             line = null;
56             while((line=in.readLine())!=null) {
57                 System.out.println("Message from client:" + line);
58                 out.println(line);
59                 out.flush();
60             }
61         } catch (IOException e) {
62             e.printStackTrace();
63         } finally {
64             try {
65                 if(out!=null)

```

用构造有参来导入数值。将tcp连接封装进线程中。

运行结果

1. 初始连接



线程抢占cpu资源。

2. 发送信息

