链滴

SocketServer 复盘

作者：luomuren

原文链接：https://ld246.com/article/1661068893552

来源网站：链滴

公司有需求，需要我们的项目作为Socket服务端来连接网关设备，所以自己从网上搜了些教程做了个ocketServer服务端

环境

jdk：jdk1.8.0_51

框架：springBoot

项目gitee：https://gitee.com/lmr-replay/replay-socketServer

寻找SocketServer服务端教程

Java Socket实现多个客户端连接同一个服务端

从网上找了很多SocketServer服务端教程，最后选了这篇

集成服务端

1. 项目中新建SocketServer工具类，拷贝教程中服务端代码到工具类中

```
/**
 * Socket服务端<br>
 * 功能说明:
 *
 * @author 大智若愚的小懂
 * @Date 2016年8月30日
 * @version 1.0
 */
public class Server {

    /**
     * 入口
     *
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {
        // 为了简单起见，所有的异常信息都往外抛
        int port = 8899;
        // 定义一个ServiceSocket监听在端口8899上
        ServerSocket server = new ServerSocket(port);
        System.out.println("等待与客户端建立连接...");
        while (true) {
            // server尝试接收其他Socket的连接请求，server的accept方法是阻塞式的
            Socket socket = server.accept();
            /**
             * 我们的服务端处理客户端的连接请求是同步进行的， 每次接收到来自客户端的连接请求后，
             * 都要先跟当前的客户端通信完之后才能再处理下一个连接请求。 这在并发比较多的情况下严重影响程序的性能，
             * 为此，我们可以把它改为如下这种异步处理与客户端通信的方式
             */
```

```
            // 每接收到一个Socket就建立一个新的线程来处理它
            new Thread(new Task(socket)).start();

        }
        // server.close();
    }

    /**
     * 处理Socket请求的线程类
     */
    static class Task implements Runnable {

        private Socket socket;

        /**
         * 构造函数
         */
        public Task(Socket socket) {
            this.socket = socket;
        }

        @Override
        public void run() {
            try {
                handlerSocket();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        /**
         * 跟客户端Socket进行通信
         *
         * @throws IOException
         */
        private void handlerSocket() throws Exception {
            // 跟客户端建立好连接之后，我们就可以获取socket的InputStream，并从中读取客户端发
来的信息了
            /**
             * 在从Socket的InputStream中接收数据时，像上面那样一点点的读就太复杂了，
             * 有时候我们就会换成使用BufferedReader来一次读一行
             *
             * BufferedReader的readLine方法是一次读一行的，这个方法是阻塞的，直到它读到了一行
据为止程序才会继续往下执行，
             * 那么readLine什么时候才会读到一行呢？直到程序遇到了换行符或者是对应流的结束符read
ine方法才会认为读到了一行，
             * 才会结束其阻塞，让程序继续往下执行。
             * 所以我们在使用BufferedReader的readLine读取数据的时候一定要记得在对应的输出流里
一定要写入换行符（
             * 流结束之后会自动标记为结束，readLine可以识别），写入换行符之后一定记得如果输出
不是马上关闭的情况下记得flush一下，
             * 这样数据才会真正的从缓冲区里面写入。
             */
            BufferedReader br = new BufferedReader(
```

```
                new InputStreamReader(socket.getInputStream(), "UTF-8"));
        StringBuilder sb = new StringBuilder();
        String temp;
        int index;
        while ((temp = br.readLine()) != null) {
            if ((index = temp.indexOf("eof")) != -1) { // 遇到eof时就结束接收
                sb.append(temp.substring(0, index));
                break;
            }
            sb.append(temp);
        }
        System.out.println("Form Cliect[port:" + socket.getPort()
                + "] 消息内容:" + sb.toString());

        // 回应一下客户端
        Writer writer = new OutputStreamWriter(socket.getOutputStream(),
                "UTF-8");
        writer.write(String.format("Hi,%d.天朗气清，惠风和畅！", socket.getPort()));
        writer.flush();
        writer.close();
        System.out.println(
                "To Cliect[port:" + socket.getPort() + "] 回复客户端的消息发送成功");

        br.close();
        socket.close();
    }

  }

}
```

2. 设置网关连接到服务端

因为设备挂在互联网上，就将项目部署到了自己的服务器上，开放了8899端口

然后设置网关连接到服务端

- 无法收到客户端确认信息问题

此时发现问题，客户端连接成功，但是收不到客户端发送的确认信息

先来说一下客户端连接成功验证机制：

客户端配置的规则是连接成功之后会自动发送一条数据到服务端，发送的数据是以字符串方式发送的并且是以'01'结尾

于是查看拷贝过来的代码，发现教程中是以**readLine()**方法接收数据，并且是已"eof"来判断是否束的

于是修改代码，使用**read()**方法读取流数据到char数组中，再将其转换成字符串，并已“01”判是否结束。

此处没有搞清楚流的使用，只是写了修改过程，表述不清楚，麻烦大神指正

此时设备连接已没有问题，并且可以正常收到客户端的确认数据

因为项目需求是要连接多个设备获取数据，所以和客户端约定，通过客户端确认信息来区分不同的设备

handlerSocket()方法修改如下：

```
private void handlerSocket() throws Exception {
        char[] charArray = new char[10];
        //获取客户端发送的数据的输入流
        InputStream inputStream = socket.getInputStream();
        //读取输入流
        InputStreamReader inputStreamReader = new InputStreamReader(inputStream);
        int readLength = inputStreamReader.read(charArray);
        String ret = "";
        while (readLength != -1) {
            String newString = new String(charArray, 0, readLength);
            ret += newString;
            if (ret.indexOf("01")!=-1) {
                ret = ret.replaceAll("01", "");
                break;
            }
            readLength = inputStreamReader.read(charArray);
        }
        System.out.println("Form Cliect[port:" + socket.getPort()
                + "] 消息内容:" + ret.toString());
        inputStream.close();
        inputStreamReader.close();
        socket.close();
    }
```

- 收到客户端信息后连接关闭问题

此时又发现一个问题，服务端收到客户端确认信息之后，连接就关闭了，无法再发送和接收数据

这块尝试了一下，关闭流也会导致socket连接关闭，不清楚原理，因为时间问题还没来得及详细研究
有大神知道可以在评论区说明一下，谢谢！

所以修改**handlerSocket()**方法，删除流关闭代码及socket关闭代码

```
// inputStream.close();
// inputStreamReader.close();
// socket.close();
```

3. 修改代码，完成多台设备连接需求

首先新建一个全局静态Map对象，来保存连接到的客户端

```
public static Map<String, Socket> socketMap = new HashMap<>();
// key:从客户端接收到的确认信息，用来标识唯一的客户端
// value:socket客户端对象
```

然后修改**handlerSocket()**方法，与客户端建立连接之后保存连接对象

```
socketMap.put(ret, socket);
```

4. 编写数据发送及接收方法

此处的需求是要给客户端发送指令来获取客户端连接的设备的数据，指令是由设备定义的

此时连接已经建立完成，接下来要做的就是给客户端发送数据，并从客户端接收数据

- 判断连接是否正常

在发送数据之前，首先判断要设备连接是否正常

```
/**
     * 验证socket是否连接
     * @param key
     * @return
     */
    public static boolean checkConnect(String key) {
    // 从socketMap中获取要连接的设备socket对象
        Socket socket = socketMap.get(key);
        if (socket != null) {
            if (socket.isConnected()&&!socket.isClosed()) {
                return true;
            } else {
                // 设备连接失败，移除连接池中的socket
                socketMap.remove(key);
            }
        }
        return false;
    }
```

- 发送数据

因为此处发送的数据是16进制的字符串，发送前先将其转换为字节数组

```
/**
     * 16进制表示的字符串转换为字节数组
     *
     * @param hexString 16进制表示的字符串
     * @return byte[] 字节数组
     */
    public static byte[] hexStringToByteArray(String hexString) {
        hexString = hexString.replaceAll(" ", "");
        int len = hexString.length();
        byte[] bytes = new byte[len / 2];
        for (int i = 0; i < len; i += 2) {
            // 两位一组，表示一个字节,把这样表示的16进制字符串，还原成一个字节
            bytes[i / 2] = (byte) ((Character.digit(hexString.charAt(i), 16) << 4) + Character
                    .digit(hexString.charAt(i + 1), 16));
        }
        return bytes;
    }
```

发送数据到客户端

```
/**
     * 发送消息到客户端
     *
     * @param sendmessage
     * @return
     */
    public static boolean sendData(String sendmessage, String key) {
        PrintWriter pw = null;
        OutputStream outputStream = null;
        Socket socket = null;
        try {
```

```
            if (checkConnect(key)) {
                //发送数据到服务端
                socket = socketMap.get(key);
                byte[] bytes = hexStringToByteArray(sendmessage);
                outputStream = socket.getOutputStream();
                outputStream.write(bytes);
                outputStream.flush();
                socket.shutdownOutput();
                System.out.println("发送成功");
                return true;
            } else {
        return false;
            }
        } catch (IOException e) {
            return false;
        } catch (BaseException e){
            throw e;
        }
    // 此处如果关闭资源，会导致客户端连接中断
    /*finally {
            //关闭资源
            System.out.println(socket.isClosed());
            if (pw != null) {
                pw.close();
                System.out.println(socket.isClosed());
            }
            System.out.println(socket.isClosed());
            if (outputStream != null) {
                try {
                    outputStream.close();
                    System.out.println(socket.isClosed());
                } catch (IOException e) {
                }
            }
        }*/
    }
```

- 接收数据

因为客户端发送的数据是16进制的，此处需要将收到的数据转换为16进制字符串

```
/**
     * 将接收到的数据转换为16进制
     * @param bytes
     * @return
     */
    public static String bytesToHexString(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < bytes.length; i++) {
            String hex = Integer.toHexString(0xFF & bytes[i]);
            if (hex.length() == 1) {
                sb.append('0');
            }
            sb.append(hex);
        }
```

```
        return sb.toString();
    }

// 接收数据
public static String getData(String key) {
        InputStream inputStream = null;
        BufferedInputStream bis = null;
        DataInputStream dis = null;
        try {
            if (checkConnect(key)) {
                System.out.println("获取数据");
                Socket socket = socketMap.get(key);
                inputStream = socket.getInputStream();
                bis = new BufferedInputStream(inputStream);
                dis = new DataInputStream(bis);
                byte[] bytes = new byte[1];
                String ret = "";
        // 此处因为客户端发送的消息格式是确定的，每条长度都是20
                while (dis.available() > 0 && ret.length() < 20) {
                    dis.read(bytes);
                    ret += bytesToHexString(bytes) + " ";
                }
                System.out.println("获取成功：" + ret);
                return ret;
            } else {
                return "";
            }
        } catch (IOException e) {
            e.printStackTrace();
            return "";
        } catch (BaseException e){
            e.printStackTrace();
            throw e;
        } /*finally {
            //关闭输出流，关闭socket
            if (StringUtils.isNotNull(inputStream)) {
                try {
                    inputStream.close();
                } catch (IOException e) {
                }
            }
        }*/
    }
```

5. 项目启动时自动开启服务端

修改代码，编写服务端初始化方法及服务端关闭方法

- 初始化方法

```
public static boolean initServer(int port) throws IOException {
        server = new ServerSocket(port);
        System.out.println("等待与客户端建立连接...");
        while (true) {
            // server尝试接收其他Socket的连接请求，server的accept方法是阻塞式的
```

```
            Socket socket = server.accept();

            /**
             * 我们的服务端处理客户端的连接请求是同步进行的， 每次接收到来自客户端的连接请求后，
             * 都要先跟当前的客户端通信完之后才能再处理下一个连接请求。 这在并发比较多的情况下
严重影响程序的性能，
             * 为此，我们可以把它改为如下这种异步处理与客户端通信的方式
             */
            // 每接收到一个Socket就建立一个新的线程来处理它
            new Thread(new Task(socket)).start();
        }
    }
```

- 关闭socketServer方法

```
/**
     * 关闭socketServer
     *
     * @return
     */
    public static boolean closeServer() throws IOException {
        Set<String> keys = socketMap.keySet();
        for (String key : keys) {
            System.out.println("---关闭连接：" + key);
            Socket socket = socketMap.get(key);
            if (socket != null && !socket.isClosed()) {
                socket.close();
            }
        }
        if (server != null) {
            server.close();
        }
        System.out.println("------关闭成功----------");
        return true;
    }
```

项目是springboot项目，在启动类中调用初始化及关闭方法

- Application

启动应用时启动SocketServer

```
public static void main(String[] args)
    {
        SpringApplication.run(VhetuApplication.class, args);
        System.out.println("\n---  智能集控管理系统   ---\n");
        try {
            SocketServer.initServer(8899);
        } catch (IOException e) {
        }
    }
```

关闭应用时关闭连接

```
@PreDestroy
    public void destory() {
```

```
        // 关闭socket连接
        log.info("\n---"+ DateUtils.getTime() +"==========关闭SocketServer连接-START==
=========");
        try {
            SocketServer.closeServer();
        } catch (IOException e) {
            e.printStackTrace();
        }
        log.info("\n---"+ DateUtils.getTime() +"==========关闭SocketServer连接-END===
========");
    }
```

总结

到此SocketServer服务端集成就结束了，目前数据可以正常发送及返回，但还是有一些问题没有解决最主要的是所有的流都没有关闭，目前没发现影响，如果有大神知道这块的原理，麻烦在评论区告诉一下，最近太忙没时间研究这块，等有时间了研究一下，再回来补充解决步骤！

联系方式

作者：永夜

邮箱：Evernight@aliyun.com

以上内容有不正确的地方烦请指正！pray
ray pray