聊天室

本节目标

1. Java Socket API简介
2. 基于单线程的服务端与客户端实现
3. 基于多线程的服务端与客户端实现

1. Socket API

1.1 Socket编程

套接字使用TCP提供了两台计算机之间的通信机制。 客户端程序创建一个套接字，并尝试连接服务器的套接字。

当连接建立时，服务器会创建一个 Socket 对象。客户端和服务器现在可以通过对 Socket 对象的写入和读取来进行通信。

java.net.Socket 类代表一个套接字，并且 java.net.ServerSocket 类为服务器程序提供了一种来监听客户端，并与他们建立连接的机制。

以下步骤在两台计算机之间使用套接字建立TCP连接时会出现：

- 服务器实例化一个 ServerSocket 对象，表示通过服务器上的端口通信。
- 服务器调用 ServerSocket 类的 accept() 方法，该方法将一直等待，直到客户端连接到服务器上给定的端口。
- 服务器正在等待时，一个客户端实例化一个 Socket 对象，指定服务器名称和端口号来请求连接。
- Socket 类的构造函数试图将客户端连接到指定的服务器和端口号。如果通信被建立，则在客户端创建一个 Socket 对象能够与服务器进行通信。
- 在服务器端，accept() 方法返回服务器上一个新的 socket 引用，该 socket 连接到客户端的 socket。

连接建立后，通过使用 I/O 流在进行通信，每一个socket都有一个输出流和一个输入流，客户端的输出流连接到服务器端的输入流，而客户端的输入流连接到服务器端的输出流。

TCP 是一个双向的通信协议，因此数据可以通过两个数据流在同一时间发送.以下是一些类提供的一套完整的有用的方法来实现 socket。

1.2 ServerSocket

服务器应用程序通过使用 java.net.ServerSocket 类以获取一个端口,并且侦听客户端请求。

ServerSocket 类有四个构造方法：

序号	方法描述
1	**public ServerSocket(int port) throws IOException** 创建绑定到特定端口的服务器套接字。
2	**public ServerSocket(int port, int backlog) throws IOException** 利用指定的 backlog 创建服务器套接字并将其绑定到指定的本地端口号。
3	**public ServerSocket(int port, int backlog, InetAddress address) throws IOException** 使用指定的端口、侦听 backlog 和要绑定到的本地 IP 地址创建服务器。
4	**public ServerSocket() throws IOException** 创建非绑定服务器套接字。

创建非绑定服务器套接字。 如果 ServerSocket 构造方法没有抛出异常，就意味着你的应用程序已经成功绑定到指定的端口，并且侦听客户端请求。

服务端socket处理客户端socket连接是需要一定时间的。ServerSocket有一个队列，存放还没有来得及处理的客户端Socket，这个队列的容量就是backlog的含义。如果队列已经被客户端socket占满了，如果还有新的连接过来，那么ServerSocket会拒绝新的连接。也就是说backlog提供了容量限制功能，避免太多的客户端socket占用太多服务器资源。

这里有一些 ServerSocket 类的常用方法：

序号	方法描述
1	**public int getLocalPort()** 返回此套接字在其上侦听的端口。
2	**public Socket accept() throws IOException** 侦听并接受到此套接字的连接。
3	**public void setSoTimeout(int timeout)** 通过指定超时值启用/禁用 SO_TIMEOUT，以毫秒为单位。timeout指的是InputStream的读取超时时间
4	**public void bind(SocketAddress host, int backlog)** 将 ServerSocket 绑定到特定地址（IP 地址和端口号）。

1.3 Socket

java.net.Socket 类代表客户端和服务器都用来互相沟通的套接字。客户端要获取一个 Socket 对象通过实例化 ，而服务器获得一个 Socket 对象则通过 accept() 方法的返回值。

Socket 类有五个构造方法.

序号	方法描述
1	**public Socket(String host, int port) throws UnknownHostException, IOException.** 创建一个流套接字并将其连接到指定主机上的指定端口号。
2	**public Socket(InetAddress host, int port) throws IOException** 创建一个流套接字并将其连接到指定 IP 地址的指定端口号。
3	**public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.** 创建一个套接字并将其连接到指定远程主机上的指定远程端口。
4	**public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.** 创建一个套接字并将其连接到指定远程地址上的指定远程端口。
5	**public Socket()** 通过系统默认类型的 SocketImpl 创建未连接套接字

当 Socket 构造方法返回，并没有简单的实例化了一个 Socket 对象，它实际上会尝试连接到指定的服务器和端口。

下面列出了一些感兴趣的方法，注意客户端和服务器端都有一个 Socket 对象，所以无论客户端还是服务端都能够调用这些方法。

序号	方法描述
1	**public void connect(SocketAddress host, int timeout) throws IOException** 将此套接字连接到服务器，并指定一个超时值。
2	**public InetAddress getInetAddress()** 返回套接字连接的地址。
3	**public int getPort()** 返回此套接字连接到的远程端口。
4	**public int getLocalPort()** 返回此套接字绑定到的本地端口。
5	**public SocketAddress getRemoteSocketAddress()** 返回此套接字连接的端点的地址，如果未连接则返回 null。
6	**public InputStream getInputStream() throws IOException** 返回此套接字的输入流。
7	**public OutputStream getOutputStream() throws IOException** 返回此套接字的输出流。
8	**public void close() throws IOException** 关闭此套接字。

1.4 InetAddress

这个类表示互联网协议(IP)地址。下面列出了 Socket 编程时比较有用的方法：

序号	方法描述
1	**static InetAddress getByAddress(byte[] addr)** 在给定原始 IP 地址的情况下，返回 InetAddress 对象。
2	**static InetAddress getByAddress(String host, byte[] addr)** 根据提供的主机名和 IP 地址创建 InetAddress。
3	**static InetAddress getByName(String host)** 在给定主机名的情况下确定主机的 IP 地址。
4	**String getHostAddress()** 返回 IP 地址字符串（以文本表现形式）。
5	**String getHostName()** 获取此 IP 地址的主机名。
6	**static InetAddress getLocalHost()** 返回本地主机。
7	**String toString()** 将此 IP 地址转换为 String。

2. 单线程模式

2.1 服务端

```
package bitekeji;

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

/**
 * 单线程聊天室服务器端
 * @author yuisama
 */
public class SingleServer {
    public static void main(String[] args) throws Exception{
        // 创建服务端Socket，端口号为6666
        ServerSocket serverSocket = new ServerSocket(6666);
        try {
            System.out.println("等待客户端连接ing...");
            // 等待客户端连接，有客户端连接后返回客户端的Socket对象，否则线程将一直阻塞于此处
            Socket client = serverSocket.accept();
            System.out.println("有新的客户端连接，端口号为："+client.getPort());
            // 获取客户端的输入输出流
            Scanner clientInput = new Scanner(client.getInputStream());
            clientInput.useDelimiter("\n");
            PrintStream clientOut = new PrintStream(client.getOutputStream(),
                    true,"UTF-8");
            // 读取客户端输入
            if (clientInput.hasNext()) {
                System.out.println(client.getInetAddress()+"说："+clientInput.next());
            }
            // 向客户端输出
            clientOut.println("Hello I am Server,Welcome! ");
            // 关闭输入输出流
            clientInput.close();
            clientOut.close();
            serverSocket.close();
        }catch (IOException e) {
            System.err.println("服务端通信出现异常，错误为"+e);
        }
```

```
    }
}
```

2.2 客户端

```
package bitekeji;

import java.io.IOException;
import java.io.PrintStream;
import java.net.Socket;
import java.util.Scanner;

/**
 * 单线程聊天室客户端
 * @author yuisama
 */
public class SingleClient {
    public static void main(String[] args) throws Exception{
        String serverName = "127.0.0.1";
        Integer port = 6666;
        try {
            // 创建客户端Socket连接服务器
            Socket client = new Socket(serverName,port);
            System.out.println("连接上服务器，服务器地址为: "+client.getInetAddress());
            // 获取输入输出流
            PrintStream out = new PrintStream(client.getOutputStream(),
                    true,"UTF-8");
            Scanner in = new Scanner(client.getInputStream());
            in.useDelimiter("\n");
            // 向服务器输出内容
            out.println("Hi I am Client!!");
            // 读取服务器输入
            if (in.hasNext()) {
                System.out.println("服务器发送消息为: "+in.next());
            }
            in.close();
            out.close();
            client.close();
        }catch (IOException e) {
            System.err.println("客户端通信出现异常，错误为"+e);
        }
    }
}
```

3. 多线程模式

3.1 客户端

```
package bitekeji;

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

```java
/**
 * 读取服务器信息线程
 */
class ReadFromServerThread implements Runnable {
    private Socket client;

    public ReadFromServerThread(Socket client) {
        this.client = client;
    }
    @Override
    public void run() {
        try {
            // 获取客户端输入流
            Scanner in = new Scanner(client.getInputStream());
            in.useDelimiter("\n");
            while (true) {
                if (in.hasNext()) {
                    System.out.println("从服务器发来的消息为: "+in.next());
                }
                // 此客户端退出
                if (client.isClosed()) {
                    System.out.println("客户端已关闭");
                    break;
                }
            }
            in.close();
        } catch (IOException e) {
            System.err.println("客户端读线程异常, 错误为 "+e);
        }
    }
}

/**
 * 将信息发送给服务器线程
 */
class WriteToServerThread implements Runnable {
    private Socket client;

    public WriteToServerThread(Socket client) {
        this.client = client;
    }

    @Override
    public void run() {
        try{
            // 获取键盘输入
            Scanner scanner = new Scanner(System.in);
            scanner.useDelimiter("\n");
            // 获取客户端输出流
            PrintStream out = new PrintStream(client.getOutputStream());
            while (true) {
                System.out.println("请输入要发送的消息..");
                String strToServer;
                if (scanner.hasNextLine()) {
                    strToServer = scanner.nextLine().trim();
                    out.println(strToServer);
                    // 客户端退出标志
                    if (strToServer.equals("byebye")) {
                        System.out.println("关闭客户端");
                        scanner.close();
```

```
                        out.close();
                        client.close();
                        break;
                    }
                }
            }
        }catch (IOException e) {
            System.out.println("客户端写线程异常，错误为 "+e);
        }
    }
}

/**
 * 多线程聊天室客户端
 * @author yuisama
 */
public class MultiThreadClient {
    public static void main(String[] args) {
        try {
            Socket client = new Socket("127.0.0.1",6666);
            // 读取服务器消息线程
            Thread readFromServer  = new Thread(new ReadFromServerThread(client));
            Thread writeToServer = new Thread(new WriteToServerThread(client));
            readFromServer.start();
            writeToServer.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3.2 服务端

```
package bitekeji;

import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Iterator;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * 多线程聊天室服务端
 * @author yuisama
 */
public class MultiThreadServer {
    // 存储所有注册的客户端
    private static Map<String, Socket> clientMap = new ConcurrentHashMap<String, Socket>();

    // 具体处理与每个客户端通信的内部类
    private static class ExecuteClient implements Runnable {
        private Socket client;
```

```
public ExecuteClient(Socket client) {
    this.client = client;
}

@Override
public void run() {
    try {
        // 获取客户端输入流
        Scanner in = new Scanner(client.getInputStream());
        String strFromClient;
        while (true) {
            if (in.hasNextLine()) {
                strFromClient = in.nextLine();
                // windows下将默认换行/r/n中的/r替换为空字符串
                Pattern pattern = Pattern.compile("\r");
                Matcher matcher = pattern.matcher(strFromClient);
                strFromClient = matcher.replaceAll("");
                // 注册流程
                if (strFromClient.startsWith("userName")) {
                    String userName = strFromClient.split("\\:")[1];
                    registerUser(userName,client);
                    continue;
                }
                // 群聊流程
                if (strFromClient.startsWith("G")) {
                    String msg = strFromClient.split("\\:")[1];
                    groupChat(msg);
                    continue;
                }
                // 私聊流程
                if (strFromClient.startsWith("P")) {
                    String userName = strFromClient.split("\\:")[1]
                            .split("-")[0];
                    String msg = strFromClient.split("\\:")[1]
                            .split("-")[1];
                    privateChat(userName,msg);
                }
                // 用户退出
                if (strFromClient.contains("byebye")) {
                    String userName = null;
                    // 根据Socket找到UserName
                    for (String keyName : clientMap.keySet()) {
                        if (clientMap.get(keyName).equals(client)) {
                            userName = keyName;
                        }
                    }
                    System.out.println("用户"+userName+"下线了!");
                    clientMap.remove(userName);
                    continue;
                }
            }
        }
    }catch (IOException e) {
        System.err.println("服务器通信异常，错误为 "+e);
    }
}
// 注册方法
private void registerUser(String userName,Socket client) {
    System.out.println("用户姓名为: "+userName);
```

```
            System.out.println("用户"+userName+"上线了！");
            System.out.println("当前群聊人数为："+(clientMap.size()+1)+"人");
            // 将用户信息保存到map中
            clientMap.put(userName,client);
            try {
                PrintStream out  = new PrintStream(client.getOutputStream(),
                        true,"UTF-8");
                // 告知用户注册成功
                out.println("用户注册成功!");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        // 群聊流程
        private void groupChat(String msg) {
            // 取出clientMap中所有Entry遍历发送群聊信息
            Set<Map.Entry<String,Socket>> clientSet = clientMap.entrySet();
            for (Map.Entry<String,Socket> entry : clientSet) {
                try {
                    Socket socket = entry.getValue();
                    // 取得每个客户端的输出流
                    PrintStream out = new PrintStream(socket.getOutputStream(),
                            true,"UTF-8");
                    out.println("群聊信息为："+msg);
                }catch (IOException e) {
                    System.err.println("群聊异常，错误为 "+e);
                }
            }
        }
        // 私聊流程
        private void privateChat(String userName,String msg) {
            Socket privateSocket = clientMap.get(userName);
            try {
                PrintStream out = new PrintStream(privateSocket.getOutputStream(),
                        true,"UTF-8");
                out.println("私聊信息为："+msg);
            }catch (IOException e) {
                System.err.println("私聊异常，错误为"+e);
            }
        }

    }

    public static void main(String[] args) throws Exception{
        ExecutorService executorService = Executors.newFixedThreadPool(20);
        ServerSocket serverSocket = new ServerSocket(6666);
        for (int i = 0 ; i < 20 ; i++) {
            System.out.println("等待客户端连接...");
            Socket client = serverSocket.accept();
            System.out.println("有新的客户端连接，端口号为："+client.getPort());
            executorService.submit(new ExecuteClient(client));
        }
        executorService.shutdown();
        serverSocket.close();
    }
}
```