

## 10. 网络编程

### 10.1. 网络编程基本概念

### 10.2. 网络编程TCP协议

### 10.3. TCP实现ECHO程序

### 10.4. 服务器和多客户端通信

### 10.5. 多客户端之间的通信

### 10.6. 网络编程UDP协议

### 10.7. URL





### 10.8. MINA框架

## 10. 网络编程

### 10.1. 网络编程基本概念

- 什么是计算机网络
  - 把分布在不同地理区域的计算机与专门的外部设备用通信线路互连成一个规模大、功能强的网络系统，从而使众多的计算机可以方便地互相传递信息，共享硬件、软件、数据信息等资源
- 计算机网络的主要功能
  - 资源共享
  - 信息传输与集中处理
  - 均衡负荷与分布处理
  - 综合信息服务（www/综合业务数字网络ISDN）等
- 网络通信协议
  - 要使计算机连成的网络能够互通信息，需要对数据输出速率、传输代码、代码结构、传输控制步骤、出错控制等指定一组标准，这一组共同遵守的通信标准及时网络通信协议，不同的计算机之间必须使用相同的通讯协议才能进行通信
- 网络通信接口
  - 为了使两个结点之间能进行对话，必须在它们之间建立通信工具（即接口），使彼此之间能进行信息交换。接口包括两部分：
    - 硬件装置：实现结点之间的信息传送
    - 软件装置：规定双方进行通信的约定协议
- TCP/IP
  - 传输控制协议/因特网互联协议，又叫网络通信协议，这个协议是Internet最基本的协议、Internet国际互联网络的基础，简单地说，就是由网络层的IP协议和传输层的TCP协议组成的
  - IP地址：网络中每台计算机的一个标识号，本地IP：127.0.0.1 localhost
    - 端口号（PORT）：端口号的范围：0~65535之间，0~1023之间的端口数是用于一些知名的网络服务和

- 网络模型

OSI七层网络模型		TCP/IP四层模型	各层网络协议
应用层		应用层	TFTP、FTP、NFS、WAIS
表示层			Telnet、rlogin、SNMP、Gopher
会话层			SMTP、DNS
传输层		传输层	TCP、UDP
网络层		网络层	IP、ICMP、IGMP、ARP、RARP
数据链路层		网络接口层（链路层）	FDDI、Ethernet、SLP、PPP、Arpanet
物理层			IEEE 802.1A、IEEE802.2-IEEE802.11

- 程序开发结构

- 网络编程主要是指完成C/S程序的开发，程序的开发结构有两种：
  - C/S（客户端/服务器）：开发两套程序，两套程序需要同时维护。C/S程序一般比较稳定
  - B/S（浏览器/服务器）：开发一套程序，客户端使用浏览器进行访问。B/S程序一般稳定性很差，而且安全性较差
- C/S程序主要可以完成以下两种程序的开发：
  - TCP（Transmission Control Protocol）：传输控制协议，采用三次握手的方式，保证准确的连接操作
  - UDP（User Datagram Protocol）：数据报协议，发送数据报

## 10.2. 网络编程TCP协议

- TCP程序概述

- TCP是一个可靠的协议，面向连接的协议
- 实现TCP协议，需要编写服务器端和客户端，Java API为我们提供了java.net包，为实现网络应用程序提供类
- ServerSocket：此类实现服务器套接字
- Socket：此类实现客户端套接字
- Socket是网络驱动层提供给应用程序编程的接口和一种机制

- 实现服务器端与客户端

- 服务器端

```

1 // 此类实现服务器套接字。服务器套接字等待请求通过网络接入。它基于该请求执行某些
  操作，然后可能向请求者返回结果
2 public class ServerSocket extends Object:
3 // 创建绑定到特定端口的服务器套接字
4 ServerSocket(int port);
5 // 通过指定超时值启动/禁用SO_TIMEOUT，以毫秒为单位
6 void setSoTimeout(int timeout);
7 // 返回此服务器套接字的本地地址
8 InetAddress getInetAddress();
9 // 侦听并接受到此套接字的连接
10 Socket accept();
  
```

## 。 客户端

```
1 // 此类实现客户端套接字（也可以就叫做“套接字”）。套接字是两台机器间通信的端点
2 public class Socket extends Object;
3 // 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
4 Socket(String host, int port);
5 // 返回此套接字的输入流
6 InputStream getInputStream();
7 // 返回此套接字的输出流
8 OutputStream getOutputStream();
9 // 启用/禁用带有指定超时值的SO_TIMEOUT，以毫秒为单位
10 void setSoTimeout(int timeout);
```

### 10.3. TCP实现ECHO程序

- Echo，意为应答，程序的功能是客户端向服务器发送一个字符串，服务器不做任何处理，直接把字符串返回给客户端，Echo程序是最基本的客户/服务器程序
- 示例：

```
1 // EchoServerDemo
2 import java.io.*;
3 import java.net.ServerSocket;
4 import java.net.Socket;
5
6 /**
7  * @author xiao儿
8  * @date 2019/9/12 9:19
9  * @Description EchoServerDemo
10  */
11 public class EchoServerDemo {
12     public static void main(String[] args) {
13         try {
14             // 创建一个服务器端的Socket (2014~65535)
15             ServerSocket server = new ServerSocket(6666);
16             System.out.println("服务器已启动，正在等待客户端连接...");
17             // 等待客户端的连接，造成阻塞，如果有客户端连接成功，立即返回一个
18             // Socket对象
19             Socket accept = server.accept();
20             System.out.println("客户端连接成功" +
21             accept.getInetAddress().getHostAddress());
22             BufferedReader br = new BufferedReader(new
23             InputStreamReader(accept.getInputStream()));
24             // 通过输入流来读取网络数据，如果没有消息，会造成阻塞
25             String info = br.readLine();
26             System.out.println(info);
27             // 获取输出流，向客户端返回消息
28             PrintStream ps = new PrintStream(new
29             BufferedOutputStream(accept.getOutputStream()));
30             ps.println("echo: " + info);
31             ps.flush();
32             // 关闭流
33             ps.close();
34             br.close();
35         } catch (IOException e) {
36             e.printStackTrace();
37         }
38     }
39 }
```

```

33     }
34 }
35 }
36
37 // EchoClientDemo
38 import java.io.*;
39 import java.net.Socket;
40
41 /**
42  * @author xiao儿
43  * @date 2019/9/12 9:25
44  * @Description EchoClientDemo
45  */
46 public class EchoClientDemo {
47     public static void main(String[] args) {
48         try {
49             // 创建一个Socket对象，指定要连接的服务器
50             Socket socket = new Socket("localhost", 6666);
51             // 获取socket输入输出流
52             PrintStream ps = new PrintStream(new
BufferedOutputStream(socket.getOutputStream()));
53             BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
54             ps.println("hello,my name is Tom");
55             ps.flush();
56             // 读取服务器端返回的数
57             String info = br.readLine();
58             System.out.println(info);
59             ps.close();
60             br.close();
61         } catch (IOException e) {
62             e.printStackTrace();
63         }
64     }
65 }

```

#### 10.4. 服务器和多客户端通信

- 服务器同时支持多个客户端的连接，就必须加入多线程的处理机制，将每一个连接的客户端都创建一个线程对象
- 示例：

```

1 // MutilServerDemo
2 import java.io.*;
3 import java.net.ServerSocket;
4 import java.net.Socket;
5 import java.util.concurrent.ExecutorService;
6 import java.util.concurrent.Executors;
7
8 /**
9  * @author xiao儿
10  * @date 2019/9/12 9:49
11  * @Description MutilServerDemo
12  *
13  * 处理多个客户端
14  * 主线程用于监听客户端的连接，每次有连接成功时开启一个线程来处理该客户端的消息

```

```

15  */
16  public class MutilServerDemo {
17      public static void main(String[] args) {
18          // 开启线程池
19          ExecutorService es = Executors.newFixedThreadPool(3);
20          try {
21              ServerSocket server = new ServerSocket(6666);
22              System.out.println("服务器已启动, 正在等待连接...");
23              while (true) {
24                  Socket accept = server.accept();
25
26                  System.out.println(accept.getInetAddress().getHostAddress());
27                  es.execute(new UserThread(accept));
28              } catch (IOException e) {
29                  e.printStackTrace();
30              }
31          }
32      }
33
34      /**
35       * 用来处理客户端请求的线程任务
36       */
37      class UserThread implements Runnable {
38          private Socket socket;
39
40          public UserThread(Socket socket) {
41              this.socket = socket;
42          }
43
44          @Override
45          public void run() {
46              try {
47                  BufferedReader br = new BufferedReader(new
48                      InputStreamReader(socket.getInputStream()));
49                  PrintStream ps = new PrintStream(new
50                      BufferedOutputStream(socket.getOutputStream()));
51                  String info = br.readLine();
52                  System.out.println(info);
53                  ps.println("echo: " + info);
54                  ps.flush();
55                  ps.close();
56                  br.close();
57              } catch (IOException e) {
58                  e.printStackTrace();
59              }
60          }
61      }
62  }

```

### 10.5. 多客户端之间的通信

- 客户端的数据包通过服务器中转, 发送到另一个客户端
- 示例:

```

1  // MessageType
2  /**

```

```

3      * @author xiao儿
4      * @date 2019/9/12 10:44
5      * @Description MessageType
6      */
7      public class MessageType {
8          public static final int TYPE_LOGIN = 0x1; // 登录消息的内容
9          public static final int TYPE_SEND = 0x2; // 发送消息的类型
10     }
11
12     // Message
13     import java.io.Serializable;
14
15     /**
16      * @author xiao儿
17      * @date 2019/9/12 10:42
18      * @Description Message
19      *
20      * 消息包
21      */
22     public class Message implements Serializable {
23         private String from; // 发送者
24         private String to; // 接收者
25         private int type; // 消息类型
26         private String info; // 消息
27
28         public String getFrom() {
29             return from;
30         }
31
32         public void setFrom(String from) {
33             this.from = from;
34         }
35
36         public String getTo() {
37             return to;
38         }
39
40         public void setTo(String to) {
41             this.to = to;
42         }
43
44         public int getType() {
45             return type;
46         }
47
48         public void setType(int type) {
49             this.type = type;
50         }
51
52         public String getInfo() {
53             return info;
54         }
55
56         public void setInfo(String info) {
57             this.info = info;
58         }
59
60         public Message() {

```

```

61     }
62
63     public Message(String from, String to, int type, String info) {
64         this.from = from;
65         this.to = to;
66         this.type = type;
67         this.info = info;
68     }
69 }
70
71 // Server
72 import java.io.*;
73 import java.net.ServerSocket;
74 import java.net.Socket;
75 import java.util.Vector;
76 import java.util.concurrent.ExecutorService;
77 import java.util.concurrent.Executors;
78
79 /**
80  * @author xiao儿
81  * @date 2019/9/12 10:15
82  * @Description Server
83  * <p>
84  * 服务器端
85  */
86 public class Server {
87     public static void main(String[] args) {
88         // 保存客户端处理的线程
89         Vector<UserThread> vector = new Vector<>();
90         ExecutorService es = Executors.newFixedThreadPool(5);
91         try {
92             // 创建服务器端的Socket
93             ServerSocket server = new ServerSocket(8888);
94             System.out.println("服务器已启动, 正在等待连接...");
95             while (true) {
96                 Socket socket = server.accept();
97                 UserThread userThread = new UserThread(socket,
vector);
98                 es.execute(userThread);
99             }
100         } catch (IOException e) {
101             e.printStackTrace();
102         }
103     }
104 }
105
106 /**
107  * 客户端处理的线程
108  */
109 class UserThread implements Runnable {
110     private String name; // 客户端的用户名称 (唯一)
111     private Socket socket;
112     private Vector<UserThread> vector; // 客户端处理线程的集合
113     private ObjectInputStream ois;
114     private ObjectOutputStream oos;
115     private boolean flag = true;
116
117     public UserThread(Socket socket, Vector<UserThread> vector) {

```

```

118         this.socket = socket;
119         this.vector = vector;
120         vector.add(this);
121     }
122
123     @Override
124     public void run() {
125         try {
126             System.out.println("客户端: " +
socket.getInetAddress().getHostAddress() + "已连接");
127             ois = new ObjectInputStream(socket.getInputStream());
128             oos = new ObjectOutputStream(socket.getOutputStream());
129             while (flag) {
130                 // 读取消息对象
131                 Message message = (Message) ois.readObject();
132                 int type = message.getType();
133                 switch (type) {
134                     case MessageType.TYPE_SEND:
135                         String to = message.getTo();
136                         UserThread ut;
137                         int size = vector.size();
138                         for (int i = 0; i < size; i++) {
139                             ut = vector.get(i);
140                             if (to.equals(ut.name) && ut != this) {
141                                 ut.oos.writeObject(message);
142                                 break;
143                             }
144                         }
145                         break;
146                     case MessageType.TYPE_LOGIN:
147                         name = message.getFrom();
148                         message.setInfo("欢迎你: ");
149                         oos.writeObject(message);
150                         break;
151                 }
152             }
153             ois.close();
154             oos.close();
155         } catch (IOException e) {
156             e.printStackTrace();
157         } catch (ClassNotFoundException e) {
158             e.printStackTrace();
159         }
160     }
161 }
162
163 // Client
164 import java.io.*;
165 import java.net.Socket;
166 import java.util.Scanner;
167 import java.util.concurrent.ExecutorService;
168 import java.util.concurrent.Executors;
169
170 /**
171  * @author xiao儿
172  * @date 2019/9/12 10:16
173  * @Description Client
174  */

```



```

175 public class Client {
176     public static void main(String[] args) {
177         Scanner sc = new Scanner(System.in);
178         ExecutorService es = Executors.newSingleThreadExecutor();
179         try {
180             Socket socket = new Socket("localhost", 8888);
181             System.out.println("服务器连接成功");
182             ObjectOutputStream oos = new
ObjectOutputStream(socket.getOutputStream());
183             ObjectInputStream ois = new
ObjectInputStream(socket.getInputStream());
184             // 向服务器发送登录消息
185             System.out.println("请输入名称: ");
186             String name = sc.nextLine();
187             Message message = new Message(name, null,
MessageType.TYPE_LOGIN, null);
188             oos.writeObject(message);
189             message = (Message) ois.readObject();
190             System.out.println(message.getInfo() + message.getFrom());
191             // 启动读取消息的线程
192             es.execute(new ReadInfoThread(ois));
193             // 使用主线程来实现发送消息
194             boolean flag = true;
195             while (flag) {
196                 message = new Message();
197                 System.out.println("To: ");
198                 message.setTo(sc.nextLine());
199                 message.setFrom(name);
200                 message.setType(MessageType.TYPE_SEND);
201                 System.out.println("Info: ");
202                 message.setInfo(sc.nextLine());
203                 oos.writeObject(message);
204             }
205         } catch (IOException e) {
206             e.printStackTrace();
207         } catch (ClassNotFoundException e) {
208             e.printStackTrace();
209         }
210     }
211 }
212
213 /**
214  * 读取消息的线程
215  */
216 class ReadInfoThread implements Runnable {
217     private ObjectInputStream in;
218     private boolean flag = true;
219
220     public ReadInfoThread(ObjectInputStream in) {
221         this.in = in;
222     }
223
224     public void setFlag(boolean flag) {
225         this.flag = flag;
226     }
227
228     @Override
229     public void run() {

```

```

230         try {
231             while (flag) {
232                 Message message = (Message) in.readObject();
233                 System.out.println "[" + message.getFrom() + "]"对我
说: " + message.getInfo());
234             }
235             if (in != null) {
236                 in.close();
237             }
238         } catch (IOException e) {
239             e.printStackTrace();
240         } catch (ClassNotFoundException e) {
241             e.printStackTrace();
242         }
243     }
244 }

```

## 10.6. 网络编程UDP协议

### • UDP协议概述

- UDP是User Datagram Protocol的简称，是一种无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传目的地，因此是否能到大目的地，到达目的地的时间以及内容的正确性都是不能被保证的，每个传输的数据报必须限定在64KB之内
- 主要使用以下两个类：
  - DatagramPacket：此类表示数据包
  - DatagramSocket：此类表示用来发送和接收数据报包的套接字

### • 示例：

```

1 // UDPServerDemo
2 import java.io.IOException;
3 import java.net.*;
4
5 /**
6  * @author xiaoJL
7  * @date 2019/9/12 15:10
8  * @Description UDPServerDemo
9  */
10 public class UDPServerDemo {
11     public static void main(String[] args) {
12         String info = "good good 学习, 天天 up";
13         byte[] bytes = info.getBytes();
14         try {
15             // 封装一个数据报包
16             DatagramPacket dp = new DatagramPacket(bytes, 0,
bytes.length, InetAddress.getByName("127.0.0.1"), 8000);
17             // 本程序的端口
18             DatagramSocket socket = new DatagramSocket(9000);
19             socket.send(dp);
20             socket.close();
21         } catch (UnknownHostException e) {
22             e.printStackTrace();
23         } catch (SocketException e) {

```

```

24         e.printStackTrace();
25     } catch (IOException e) {
26         e.printStackTrace();
27     }
28 }
29 }
30
31 // UDPClientDemo
32 import java.io.IOException;
33 import java.net.DatagramPacket;
34 import java.net.DatagramSocket;
35 import java.net.SocketException;
36
37 /**
38  * @author xiaoJL
39  * @date 2019/9/12 15:18
40  * @Description UDPClientDemo
41  *
42  * 客户端
43  */
44 public class UDPClientDemo {
45     public static void main(String[] args) {
46         byte[] bytes = new byte[1024];
47         DatagramPacket dp = new DatagramPacket(bytes, bytes.length);
48         try {
49             DatagramSocket socket = new DatagramSocket(8000);
50             System.out.println("正在接收数据中...");
51             socket.receive(dp); // 接收数据
52             String s = new String(dp.getData(), 0, dp.getLength());
53             System.out.println(s);
54             socket.close();
55         } catch (SocketException e) {
56             e.printStackTrace();
57         } catch (IOException e) {
58             e.printStackTrace();
59         }
60     }
61 }

```

## 10.7. URL

- URL概述

- URL (uniform resource location) 类URL代表一个统一资源定位符，它是指向互联网“资源”的指针。
- 抽象类URLConnection是所有类的超类，它代表应用程序和URL之间的通信链接

- 示例：

```

1  import java.io.BufferedReader;
2  import java.io.BufferedOutputStream;
3  import java.io.FileOutputStream;
4  import java.io.IOException;
5  import java.net.HttpURLConnection;
6  import java.net.MalformedURLException;
7  import java.net.URL;

```

```

8
9  /**
10 * @author xiao儿
11 * @date 2019/9/12 16:10
12 * @Description URLEDemo
13 */
14 public class URLEDemo {
15     public static void main(String[] args) {
16         try {
17             URL url = new
18             URL("http://pic.netbian.com/uploads/allimg/190905/231431-
19             15676964715fb0.jpg");
20             HttpURLConnection httpURLConnection = (HttpURLConnection)
21             url.openConnection();
22             BufferedInputStream in = new
23             BufferedInputStream(httpURLConnection.getInputStream());
24             BufferedOutputStream out = new BufferedOutputStream(new
25             FileOutputStream("./Java入门/src/day10_网络编程/url/photo.jpg"));
26             byte[] bytes = new byte[1024];
27             int len = -1;
28             while ((len = in.read(bytes)) != -1) {
29                 out.write(bytes, 0, len);
30                 out.flush();
31             }
32             in.close();
33             out.close();
34             System.out.println("下载成功");
35         } catch (MalformedURLException e) {
36             e.printStackTrace();
37         } catch (IOException e) {
38             e.printStackTrace();
39         }
40     }
41 }

```

#### 10.8. MINA框架

- MINA是一个简洁易用的基于TCP/IP通信的MINA框架
- 下载地址: [http://mina.apache.org/downloads-mina\\_2\\_0.html](http://mina.apache.org/downloads-mina_2_0.html)
- 一个简单的网络程序需要的最少jar包: mina-core-2.0.16.jar、slf4j-api-1.7.21.jar
- 开发一个MINA应用, 简单的说, 就是创建连接, 设定过滤规则, 编写自己的消息处理器
- 示例:

```

1  // 服务器端
2  import org.apache.mina.core.service.IoHandlerAdapter;
3  import org.apache.mina.core.session.IoSession;
4
5  /**
6   * @author xiao儿
7   * @date 2019/9/12 16:41
8   * @Description MinaServerHandler

```

```

9      *
10     * 服务器端的消息处理器
11     */
12     public class MinaServerHandler extends IoHandlerAdapter {
13         // 一次会话被打开
14         @Override
15         public void sessionOpened(IoSession session) throws Exception {
16             super.sessionOpened(session);
17             System.out.println("Welcome client :" +
session.getRemoteAddress());
18         }
19
20         // 会话关闭
21         @Override
22         public void sessionClosed(IoSession session) throws Exception {
23             super.sessionClosed(session);
24             System.out.println("client closed");
25         }
26
27         // 接收消息
28         @Override
29         public void messageReceived(IoSession session, Object message)
throws Exception {
30             super.messageReceived(session, message);
31             String msg = (String) message; // 接收到的消息对象
32             System.out.println("收到客户端发来的消息: " + msg);
33             // 向客户端发送消息对象
34             session.write("echo: " + msg);
35         }
36     }
37
38     import org.apache.mina.core.filterchain.DefaultIoFilterChainBuilder;
39     import org.apache.mina.filter.codec.ProtocolCodecFilter;
40     import org.apache.mina.filter.codec.textline.TextLineCodecFactory;
41     import org.apache.mina.transport.socket.SocketAcceptor;
42     import org.apache.mina.transport.socket.nio.NioSocketAcceptor;
43
44     import java.io.IOException;
45     import java.net.InetSocketAddress;
46
47     /**
48      * @author xiao儿
49      * @date 2019/9/12 16:35
50      * @Description Server
51      */
52     public class Server {
53         public static void main(String[] args) {
54             // 创建一个非阻塞的Server端Socket NIO
55             SocketAcceptor acceptor = new NioSocketAcceptor();
56             DefaultIoFilterChainBuilder chain = acceptor.getFilterChain();
57             chain.addLast( "myChain", new ProtocolCodecFilter(new
TextLineCodecFactory()));
58             // 设置服务器端的消息处理器
59             acceptor.setHandler(new MinaServerHandler());
60             int port = 9999; // 服务器的端口号
61             try {
62                 // 绑定端口号, 并启动服务器 (不会阻塞, 立即返回)
63                 acceptor.bind(new InetSocketAddress(port));

```

```

64         } catch (IOException e) {
65             e.printStackTrace();
66         }
67         System.out.println("Mina Server running, listener on :" +
port);
68     }
69 }
70
71 // 客户端
72 import org.apache.mina.core.service.IoHandlerAdapter;
73 import org.apache.mina.core.session.IoSession;
74
75 /**
76  * @author xiao儿
77  * @date 2019/9/13 10:01
78  * @Description MinaClientHandler
79  */
80 public class MinaClientHandler extends IoHandlerAdapter {
81     @Override
82     public void sessionOpened(IoSession session) throws Exception {
83         super.sessionOpened(session);
84         System.out.println("sessionOpened");
85     }
86
87     @Override
88     public void sessionClosed(IoSession session) throws Exception {
89         super.sessionClosed(session);
90         System.out.println("sessionClosed");
91     }
92
93     @Override
94     public void messageReceived(IoSession session, Object message)
throws Exception {
95         super.messageReceived(session, message);
96         String msg = (String) message;
97         System.out.println(msg);
98     }
99 }
100
101 import org.apache.mina.core.filterchain.DefaultIoFilterChainBuilder;
102 import org.apache.mina.core.future.ConnectFuture;
103 import org.apache.mina.filter.codec.ProtocolCodecFilter;
104 import org.apache.mina.filter.codec.textline.TextLineCodecFactory;
105 import org.apache.mina.transport.socket.nio.NioSocketConnector;
106
107 import java.net.InetSocketAddress;
108 import java.util.Scanner;
109
110 /**
111  * @author xiao儿
112  * @date 2019/9/13 10:01
113  * @Description Client
114  */
115 public class Client {
116     public static void main(String[] args) {
117         // 创建连接
118         NioSocketConnector connector = new NioSocketConnector();

```

```
119         DefaultIoFilterChainBuilder chain =
connector.getFilterChain();
120         chain.addLast("myChain", new ProtocolCodecFilter(new
TextLineCodecFactory()));
121         connector.setHandler(new MinaClientHandler());
122         connector.setConnectTimeoutMillis(10000);
123         // 连接服务器
124         ConnectFuture cf = connector.connect(new
InetSocketAddress("localhost", 9999));
125         cf.awaitUninterruptibly();// 等待连接成功
126         Scanner input = new Scanner(System.in);
127         while (true) {
128             System.out.println("请输入: ");
129             String info = input.nextLine();
130             // 发送消息
131             cf.getSession().write(info);
132         }
133         // 等待服务器连接关闭, 结束长连接
134         // cf.getSession().getCloseFuture().awaitUninterruptibly();
135         // connector.dispose();// 关闭连接
136     }
137 }
```