1、ServerSocket API

ServerSocket 是创建TCP服务端Socket的API。

1. ServerSocket 构造方法：

方法签名	方法说明
ServerSocket(int port)	创建一个服务端流套接字Socket，并绑定到指定端口

2.ServerSocket 方法：

方法签名	方法说明
Socket accept()	开始监听指定端口（创建时绑定的端口），有客户端连接后，返回一个服务端Socket对象，并基于该Socket建立与客户端的连接，否则阻塞等待
void close()	关闭此套接字

accept：和TCP “有连接” 这样的特性密切相关！客户端尝试建立连接，首先是服务器的操作系统这一层和客户端进行一些相关的流程，把这个连接准备好，用户代码调用 accept ，才是真的把这个连接拿到用户代码中。

close：打开一个文件之后，要及时关闭，如果关闭，就会出现文件资源泄露的情况。

2、Socket API

Socket 是客户端 Socket，或服务端中接收到客户端建立连接（accept方法）的请求后，返回的服务端Socket。

不管是客户端还是服务端Socket，都是双方建立连接以后，保存的对端信息，及用来与对方收发数据的。

Socket 构造方法：

方法签名	方法说明
Socket(String host, int port)	创建一个客户端流套接字Socket，并与对应IP的主机上，对应端口的进程建立连接

Socket 方法：

方法签名	方法说明
InetAddress getInetAddress()	返回套接字所连接的地址
InputStream getInputStream()	返回此套接字的输入流
OutputStream getOutputStream()	返回此套接字的输出流

3、代码实现

1. 服务器

1.创建 ServerSocket 关联上一个端口号，称为 listenSocket

2.调用 ServerSocket 的 accept 方法

accept 的功能就是把一个内核建立好的连接拿到代码中处理

accept 返回一个Socket实例，称为 clientSocket

3.使用 clientSocket 的 getInputStream 和 getOutputStream 得到字节流对象，就可以进行读取和写入了

4.当客户端断开连接之后，服务器应该及时的关闭 cliengSocket，（否则可能会出现文件资源泄露的情况）

```
1  public class TcpEchoServer {
2   private ServerSocket listenSocket = null;
3
4   public TcpEchoServer(int port) throws IOException {
5   this.listenSocket = new ServerSocket(port);
6   }
7
8   public void start() throws IOException {
9   System.out.println("服务器启动!");
10   while (true) {
11   // UDP 的服务器进入主循环,就直接尝试 receive 读取请求
12   // 但是 TCP 是连接的,先需要做的是,就是建立好连接
13   // 当服务器运行的时候,当前是否有客户端建立连接,不确定
```

```
14  // 如果客户端没有建立连接,accept 就会阻塞等待
15  // 如果客户端建立连接了,此时的 accept 就会返回一个 Socket 对象
16  // 进一步的服务器和客户端之间的交,就交给 clientSocket 来完成
17  Socket clientSocket = listenSocket.accept();
18  processConnection(clientSocket);
19  }
20  }
21  private void processConnection(Socket clientSocket) throws IOE
xception {
22  // 处理一个连接，在这个连接中可能会涉及到客户端和服务器之间的交互
23  String log = String.format("[%s:%d 客户端上线! ]",clientSocket.
etInetAddress().toString(),
24  clientSocket.getPort());
25  System.out.println(log);
26
27  try (InputStream is = clientSocket.getInputStream();
28  OutputStream os = clientSocket.getOutputStream()) {
29  // 1.读取请求并解析
30  // 可以直接通过 inputStream 的 read 把数据读到一个Byte[].然后在装
成 String
31  while(true){
32  Scanner sc = new Scanner(is);
33  if(!sc.hasNext()){
34  log = String.format("[%s:%d 客户端下线! ]",clientSocket.getInet
ddress().toString(),
35  clientSocket.getPort());
36  System.out.println(log);
37  break;
38  }
39  String request = sc.next();
40  // 2.根据请求来计算响应
41  String response = process(request);
42  // 3.将响应写给客户端
43  PrintWriter writer = new PrintWriter(os);
44  writer.println(response);
45  writer.flush();
```

```
46   // 4.打印日志文件
47   log = String.format("[%s:%d] req: %s;resp: %s",clientSocket.ge
tInetAddress().toString(),
48   clientSocket.getPort(),request,response);
49   System.out.println(log);
50   }
51
52   }catch(Exception e){
53   e.printStackTrace();
54   }finally {
55   clientSocket.close();
56   }
57   }
58
59   private String process(String request) {
60   return request;
61   }
62 }
```

2. 客户端

1.创建一个 Socket 对象，创建的同时指定服务器的 IP 和 端口（这个操作会让客户端和服务器奖励Tcp连接，这个连接的过程就是传说中的“三次握手”，是在内核中完成的，用户代码感知不到）

2. 客户端就可以通过 Socket 对象的 getInputStream 和 getOutputStream 和服务器进行通信。

```
1 public class TcpClient {
2   private int serverPort;
3   private String serverIP;
4   private Socket socket = null;
5
6   public TcpClient( String serverIP,int serverPort) throws IOExce
ption {
7   this.serverPort = serverPort;
8   this.serverIP = serverIP;
```

```
9   // 创建 socket 的同时就和服务器进行了连接
10  this.socket = new Socket(serverIP,serverPort);
11  }
12
13  public void start(){
14  Scanner sc = new Scanner(System.in);
15  try (InputStream is = socket.getInputStream();
16  OutputStream os = socket.getOutputStream()){
17  while (true){
18  // 1.从键盘读取用户的输入
19  System.out.println("--->");
20  String request = sc.nextLine();
21  if(request.equals("exit")){
22  System.out.println("exit");
23  break;
24  }
25
26  // 2.将读取的内容构造成请求,发送给服务器
27  PrintWriter writer = new PrintWriter(os);
28  writer.println(request);
29  writer.flush();
30  // 3.读取服务器发来的响应并解析
31  Scanner responseScan = new Scanner(is);
32  String response = responseScan.next();
33  // 4. 将结果显示到界面上
34  String log = String.format("req: %s,resp:
%s",request,response);
35  System.out.println(log);
36  }
37  } catch (IOException e) {
38  e.printStackTrace();
39  } finally {
40  }
41  System.out.println("--->");
42  }
```

```
43
44  public static void main(String[] args) throws IOException {
45  TcpClient client = new TcpClient("127.0.0.1", 8090);
46  client.start();
47  }
48 }
```

当服务器启动之后，此时的服务器就会阻塞在 accept 方法（此时的客户端还没有建立连接），当服务器和客户端建立连接之后，服务器的 accept 就返回一个 Socket 对象。

客户端启动，调用 Socket 构造方法，在构造方法中，会尝试和服务器建立连接。

```
1 String request = sc.next();// 服务器
2 String response = process(request);
```

服务器的sc.next（）阻塞，直至客户端发送请求为止。当下的状态是。客户端阻塞等待用户从键盘输入，服务器阻塞在等待客户端的请求。接下来，当客户端输入数据之后，客户端的阻塞就结束了，然后发送给服务器一个数据，同时服务器就从等待读取客户端请求的状态中恢复过来，执行后面的 process（）逻辑。

```
1 String request = sc.next();// 客户端
```

客户端阻塞等待用户从键盘输入。

```
1  String response = responseScan.next();// 客户端
```

当客户端发送完请求之后，会在第二个 scnner.next 这里阻塞，等待服务器的响应的数据。

在网络通通信程序中，涉及到很多的”阻塞“这样的情况，这些阻塞其实就是客户端和服务器之间流程的交替。

多线程，锁，也会导致阻塞

一旦阻塞，意味着线程要被挂起（放在等待队列中，等待时机成熟，才会被唤醒，然后进一步的往下执行）。一旦出现阻塞，整个程序的执行效率会受到很大的影响。

停止程序的时候，先停止客户端，后停止服务器（推荐的做法）