链滴

Android：这是一份很详细的 Socket 使用攻略

作者：woyehua

原文链接：https://ld246.com/article/1541824545532

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20180106.jpg?imageView2/1/w/960/h/520/interlace/1/q/100">
前言
Socket 的使用在 Android 网络编程中非常重要
今天我将带大家全面了解 Socket 及 其使用方法
目录
1.网络基础
1.1 计算机网络分层
计算机网络分为五层：物理层、数据链路层、网络层、运输层、应用层
其中：
网络层：负责根据 IP 找到目的地址的主机
运输层：通过端口把数据传到目的主机的目的进程，来实现进程与进程之间的通信
1.2 端口号（PORT）
端口号规定为 16 位，即允许一个 IP 主机有 2 的 16 次方 65535 个不同的端口。其中：
0~1023：分配给系统的端口号
我们不可以乱用
1024~49151：登记端口号，主要是让第三方应用使用
但是必须在 IANA（互联网数字分配机构）按照规定手续登记，
49152~65535：短暂端口号，是留给客户进程选择暂时使用，一个进程使用完就可以供其他进
使用。
在 Socket 使用时，可以用 1024~65535 的端口号
1.3 C/S 结构
定义：即客户端/服务器结构，是软件系统体系结构
作用：充分利用两端硬件环境的优势，将任务合理分配到 Client 端和 Server 端来实现，降低了系统
通讯开销。
Socket 正是使用这种结构建立连接的，一个套接字接客户端，一个套接字接服务器。
如图：
可以看出，Socket 的使用可以基于 TCP 或者 UDP 协议。
1.4 TCP 协议
定义：Transmission Control Protocol，即传输控制协议，是一种传输层通信协议
基于 TCP 的应用层协议有 FTP、Telnet、SMTP、HTTP、POP3 与 DNS。
特点：面向连接、面向字节流、全双工通信、可靠
面向连接：指的是要使用 TCP 传输数据，必须先建立 TCP 连接，传输完成后释放连接，就像打
话一样必须先拨号建立一条连接，打完后挂机释放连接。
全双工通信：即一旦建立了 TCP 连接，通信双方可以在任何时候都能发送数据。
可靠的：指的是通过 TCP 连接传送的数据，无差错，不丢失，不重复，并且按序到达。
面向字节流：流，指的是流入到进程或从进程流出的字符序列。简单来说，虽然有时候要传输的
据流太大，TCP 报文长度有限制，不能一次传输完，要把它分为好几个数据块，但是由于可靠性保证
接收方可以按顺序接收数据块然后重新组成分块之前的数据流，所以 TCP 看起来就像直接互相传输
节流一样，面向字节流。
TCP 建立连接
必须进行三次握手：若 A 要与 B 进行连接，则必须
第一次握手：建立连接。客户端发送连接请求报文段，将 SYN 位置为 1，Sequence Number 为 x
然后，客户端进入 SYN_SEND 状态，等待服务器的确认。即 A 发送信息给 B
第二次握手：服务器收到客户端的 SYN 报文段，需要对这个 SYN 报文段进行确认。即 B 收到连接
息后向 A 返回确认信息
第三次握手：客户端收到服务器的（SYN+ACK）报文段，并向服务器发送 ACK 报文段。即 A 收到
认信息后再次向 B 返回确认连接信息
此时，A 告诉自己上层连接建立；B 收到连接信息后告诉上层连接建立。
这样就完成 TCP 三次握手 = 一条 TCP 连接建立完成 = 可以开始发送数据
三次握手期间任何一次未收到对面回复都要重发。
最后一个确认报文段发送完毕以后，客户端和服务器端都进入 ESTABLISHED 状态。
为什么 TCP 建立连接需要三次握手？
答：防止服务器端因为接收了早已失效的连接请求报文从而一直等待客户端请求，从而浪费资源

“已失效的连接请求报文段”的产生在这样一种情况下：Client 发出的第一个连接请求报文段并有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达 server。
这是一个早已失效的报文段。但 Server 收到此失效的连接请求报文段后，就误认为是 Client 再次发的一个新的连接请求。
于是就向 Client 发出确认报文段，同意建立连接。
假设不采用“三次握手”：只要 Server 发出确认，新的连接就建立了。
由于现在 Client 并没有发出建立连接的请求，因此不会向 Server 发送数据。
但 Server 却以为新的运输连接已经建立，并一直等待 Client 发来数据。>- 这样，Server 的资源白白浪费掉了。
采用“三次握手”的办法可以防止上述现象发生：
Client 不会向 Server 的确认发出确认
Server 由于收不到确认，就知道 Client 并没有要求建立连接
所以 Server 不会等待 Client 发送数据，资源就没有被浪费
TCP 释放连接
TCP 释放连接需要四次挥手过程，现在假设 A 主动释放连接：（数据传输结束后，通信的双方都可放连接）
第一次挥手：A 发送释放信息到 B；（发出去之后，A->B 发送数据这条路径就断了）
第二次挥手：B 收到 A 的释放信息之后，回复确认释放的信息：我同意你的释放连接请求
第三次挥手：B 发送“请求释放连接“信息给 A
第四次挥手：A 收到 B 发送的信息后向 B 发送确认释放信息：我同意你的释放连接请求
B 收到确认信息后就会正式关闭连接；
A 等待 2MSL 后依然没有收到回复，则证明 B 端已正常关闭，于是 A 关闭连接
为什么 TCP 释放连接需要四次挥手？
为了保证双方都能通知对方“需要释放连接”，即在释放连接后都无法接收或发送消息给对方
需要明确的是：TCP 是全双工模式，这意味着是双向都可以发送、接收的
释放连接的定义是：双方都无法接收或发送消息给对方，是双向的
当主机 1 发出“释放连接请求”（FIN 报文段）时，只是表示主机 1 已经没有数据要发送 / 数据已经部发送完毕；
但是，这个时候主机 1 还是可以接受来自主机 2 的数据。
当主机 2 返回“确认释放连接”信息（ACK 报文段）时，表示它已经知道主机 1 没有数据发送了

但此时主机 2 还是可以发送数据给主机 1
当主机 2 也发送了 FIN 报文段时，即告诉主机 1 我也没有数据要发送了
此时，主机 1 和 2 已经无法进行通信：主机 1 无法发送数据给主机 2，主机 2 也无法发送数据给主机 1，此时，TCP 的连接才算释放
1.5 UDP 协议
定义：User Datagram Protocol，即用户数据报协议，是一种传输层通信协议。
基于 UDP 的应用层协议有 TFTP、SNMP 与 DNS。
特点：无连接的、不可靠的、面向报文、没有拥塞控制
无连接的：和 TCP 要建立连接不同，UDP 传输数据不需要建立连接，就像写信，在信封写上收人名称、地址就可以交给邮局发送了，至于能不能送到，就要看邮局的送信能力和送信过程的困难程了。
不可靠的：因为 UDP 发出去的数据包发出去就不管了，不管它会不会到达，所以很可能会出现包现象，使传输的数据出错。
面向报文：数据报文，就相当于一个数据包，应用层交给 UDP 多大的数据包，UDP 就照样发送不会像 TCP 那样拆分。
没有拥塞控制：拥塞，是指到达通信子网中某一部分的分组数量过多，使得该部分网络来不及处，以致引起这部分乃至整个网络性能下降的现象，严重时甚至会导致网络通信业务陷入停顿，即出现锁现象，就像交通堵塞一样。TCP 建立连接后如果发送的数据因为信道质量的原因不能到达目的地，会不断重发，有可能导致越来越塞，所以需要一个复杂的原理来控制拥塞。而 UDP 就没有这个烦恼发出去就不管了。
应用场景
很多的实时应用（如 IP 电话、实时视频会议、某些多人同时在线游戏等）要求源主机以很定的速率

送数据，并且允许在网络发生拥塞时候丢失一些数据，但是要求不能有太大的延时，UDP 就刚好适
这种要求。所以说，只有不适合的技术，没有真正没用的技术。</p>
<p>1.6 HTTP 协议
详情请看我写的另外一篇文章你需要了解的 HTTP 知识都在这里了！</p>
<ol start="2">
Socket 定义
即套接字，是一个对 TCP / IP 协议进行封装 的编程调用接口（API）

<p>即通过 Socket，我们才能在 Andorid 平台上通过 TCP/IP 协议进行开发
Socket 不是一种协议，而是一个编程调用接口（API），属于传输层（主要解决数据如何在网络中传
）
成对出现，一对套接字：</p>
<p>Socket ={(IP 地址 1:PORT 端口号)，(IP 地址 2:PORT 端口号)}
1
3. 原理
Socket 的使用类型主要有两种：</p>
<p>流套接字（streamsocket） ：基于 TCP 协议，采用 流的方式 提供可靠的字节流服务
数据报套接字(datagramsocket)：基于 UDP 协议，采用 数据报文 提供数据打包发送的服务
具体原理图如下：</p>
<ol start="4">
Socket 与 Http 对比
Socket 属于传输层，因为 TCP / IP 协议属于传输层，解决的是数据如何在网络中传输的问题
HTTP 协议 属于 应用层，解决的是如何包装数据
由于二者不属于同一层面，所以本来是没有可比性的。但随着发展，默认的 Http 里封装了下面几层
使用，所以才会出现 Socket & HTTP 协议的对比：（主要是工作方式的不同）：

<p>Http：采用 请求—响应 方式。</p>
<p>即建立网络连接后，当 客户端 向 服务器 发送请求后，服务器端才能向客户端返回数据。
可理解为：是客户端有需要才进行通信
Socket：采用 服务器主动发送数据 的方式</p>
<p>即建立网络连接后，服务器可主动发送消息给客户端，而不需要由客户端向服务器发送请求
可理解为：是服务器端有需要才进行通信
5. 使用步骤
Socket 可基于 TCP 或者 UDP 协议，但 TCP 更加常用
所以下面的使用步骤 & 实例的 Socket 将基于 TCP 协议</p>
<p>// 步骤 1：创建客户端 & 服务器的连接</p>
<pre><code class="highlight-chroma"><span class="highlight
cl">// 创建Socket对象 & 指定服务端的IP及端口号
Socket socket = n
w Socket("192.168.1.32", 1989);

// 判断客户端和服
器是否连接成功
socket.isConnecte
());
</code></pre>
<p>// 步骤 2：客户端 & 服务器 通信
// 通信包括：客户端 接收服务器的数据 & 发送数据 到 服务器</p>
<pre><code class="highlight-chroma"><span class="highlight
cl"><-- 操作1：接收服务器的数据 -->

   // 步骤1：创
输入流对象InputStream
<span class="highlight-line"><span class="highlight-cl">   InputStream

```
s = socket.getInputStream()
                // 步骤2：创
输入流读取器对象 并传入输入流对象
                // 该对象作
：获取服务器返回的数据
                InputStream
eader isr = new InputStreamReader(is);
                BufferedRea
er br = new BufferedReader(isr);

                // 步骤3：通
输入流读取器对象 接收服务器发送过来的数据
                br.readLine(
;


&lt;-- 操作2：发
数据 到 服务器 --&gt;

                // 步骤1：从
ocket 获得输出流对象OutputStream
                // 该对象作
：发送数据
                OutputStrea
 outputStream = socket.getOutputStream();

                // 步骤2：写
需要发送的数据到输出流对象中
                outputStream
write（（"Carson_Ho"+"\n"）.getBytes("utf-8"））；
                // 特别注意
数据的结尾加上换行符才可让服务器端的readline()停止阻塞

                // 步骤3：发
数据到服务端
                outputStream
flush();
</code></pre>
<p>// 步骤 3：断开客户端 &amp;amp; 服务器 连接</p>
<pre><code class="highlight-chroma">        os.close();
                // 断开 客户
发送到服务器 的连接，即关闭输出流对象OutputStream

                br.close();
                // 断开 服务
发送到客户端 的连接，即关闭输入流读取器对象BufferedReader

                socket.close()

                // 最终关闭
个Socket连接
</code></pre>
```

<p>1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
6. 具体实例
实例 Demo 代码包括：客户端 & 服务器
本文着重讲解客户端，服务器仅采用最简单的写法进行展示

6.1 客户端 实现
步骤 1：加入网络权限

1

步骤2：主布局界面设置

包括创建 Socket 连接、客户端 & 服务器通信的按钮
<pre><code class="highlight-chroma"><span class="highlight
cl"><Button
    android:id="@+
d/connect"
    android:layout
width="match_parent"
    android:layout_
eight="wrap_content"
    android:text="c
nnect" />

<Button
    android:id="@+
d/disconnect"
    android:layout
width="match_parent"
    android:layout_
eight="wrap_content"
    android:text="d
sconnect" />

<TextView
    android:id="@+
d/receive_message"
    android:layout
width="match_parent"
    android:layout_
eight="wrap_content" />

<Button
    android:id="@+
d/Receive"
    android:layout
width="match_parent"
    android:layout_
eight="wrap_content"
    android:text="
eceive from message" />

<EditText
    android:id="@+
d/edit"
    android:layout
width="match_parent"
    android:layout_
eight="wrap_content" />

<Button
    android:id="@+
d/send"
    android:layout
width="match_parent"
    android:layout_
eight="wrap_content"
    android:text="s
nd"/>
</code></pre>
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
步骤 3：创建 Socket 连接、客户端 & 服务器通信
具体请看注释
MainActivity.java
package scut.carson_ho.socket_carson;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.view.View;

import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class MainActivity extends AppCompatActivity {
<pre><code class="highlight-chroma"><span class="highlight
cl">/**
 * 主 变量
 */

// 主线程Handler
// 用于将从服务器
取的消息显示出来
private Handler
MainHandler;

// Socket变量
private Socket soc
et;

// 线程池
// 为了方便展示,
处直接采用线程池进行线程管理,而没有一个个开线程
private ExecutorSe
vice mThreadPool;

/**
 * 接收服务器消息
变量
 */
// 输入流对象
InputStream is;

// 输入流读取器对

InputStreamReade
 isr ;
BufferedReader br
;

// 接收服务器发送
来的消息
String response;

/**
 * 发送消息到服务

```
 变量
</span></span><span class="highlight-line"><span class="highlight-cl"> */
</span></span><span class="highlight-line"><span class="highlight-cl">// 输出流对象
</span></span><span class="highlight-line"><span class="highlight-cl">OutputStream ou
putStream;
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">/**
</span></span><span class="highlight-line"><span class="highlight-cl"> * 按钮 变量
</span></span><span class="highlight-line"><span class="highlight-cl"> */
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">// 连接 断开连接
送数据到服务器 的按钮变量
</span></span><span class="highlight-line"><span class="highlight-cl">private Button bt
Connect, btnDisconnect, btnSend;
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">// 显示接收服务器
息 按钮
</span></span><span class="highlight-line"><span class="highlight-cl">private TextView
eceive,receive_message;
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">// 输入需要发送的
息 输入框
</span></span><span class="highlight-line"><span class="highlight-cl">private EditText m
dit;
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">@Override
</span></span><span class="highlight-line"><span class="highlight-cl">protected void on
reate(Bundle savedInstanceState) {
</span></span><span class="highlight-line"><span class="highlight-cl">    super.onCreate(
avedInstanceState);
</span></span><span class="highlight-line"><span class="highlight-cl">    setContentView
R.layout.activity_main);
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    /**
</span></span><span class="highlight-line"><span class="highlight-cl">     * 初始化操作
</span></span><span class="highlight-line"><span class="highlight-cl">     */
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    // 初始化所有
钮
</span></span><span class="highlight-line"><span class="highlight-cl">    btnConnect = (
utton) findViewById(R.id.connect);
</span></span><span class="highlight-line"><span class="highlight-cl">    btnDisconnect
 (Button) findViewById(R.id.disconnect);
</span></span><span class="highlight-line"><span class="highlight-cl">    btnSend = (But
on) findViewById(R.id.send);
</span></span><span class="highlight-line"><span class="highlight-cl">    mEdit = (EditTe
t) findViewById(R.id.edit);
</span></span><span class="highlight-line"><span class="highlight-cl">    receive_messag
 = (TextView) findViewById(R.id.receive_message);
</span></span><span class="highlight-line"><span class="highlight-cl">    Receive = (Butt
n) findViewById(R.id.Receive);
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    // 初始化线程池
</span></span><span class="highlight-line"><span class="highlight-cl">    mThreadPool =
```

```
Executors.newCachedThreadPool();
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    // 实例化主线程
用于更新接收过来的消息
</span></span><span class="highlight-line"><span class="highlight-cl">    mMainHandler
= new Handler() {
</span></span><span class="highlight-line"><span class="highlight-cl">        @Override
</span></span><span class="highlight-line"><span class="highlight-cl">        public void h
ndleMessage(Message msg) {
</span></span><span class="highlight-line"><span class="highlight-cl">            switch (ms
.what) {
</span></span><span class="highlight-line"><span class="highlight-cl">                case 0:
</span></span><span class="highlight-line"><span class="highlight-cl">                    receiv
_message.setText(response);
</span></span><span class="highlight-line"><span class="highlight-cl">                    break;
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">    };
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    /**
</span></span><span class="highlight-line"><span class="highlight-cl">     * 创建客户端 &
mp; 服务器的连接
</span></span><span class="highlight-line"><span class="highlight-cl">     */
</span></span><span class="highlight-line"><span class="highlight-cl">    btnConnect.set
nClickListener(new View.OnClickListener() {
</span></span><span class="highlight-line"><span class="highlight-cl">        @Override
</span></span><span class="highlight-line"><span class="highlight-cl">        public void o
Click(View v) {
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">            // 利用线
池直接开启一个线程 &amp;amp; 执行该线程
</span></span><span class="highlight-line"><span class="highlight-cl">            mThreadP
ol.execute(new Runnable() {
</span></span><span class="highlight-line"><span class="highlight-cl">                @Overr
de
</span></span><span class="highlight-line"><span class="highlight-cl">                public v
id run() {
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">                    try {
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">                        //
建Socket对象 &amp;amp; 指定服务端的IP 及 端口号
</span></span><span class="highlight-line"><span class="highlight-cl">                        soc
et = new Socket("192.168.1.172", 8989);
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">                        //
断客户端和服务器是否连接成功
</span></span><span class="highlight-line"><span class="highlight-cl">                        Sys
em.out.println(socket.isConnected());
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">                    } catc
 (IOException e) {
```

```
                    e.printStackTrace();
                }

            }
        });

    }
});

/**
 * 接收 服务器信息
 */
Receive.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        // 利用线池直接开启一个线程 &amp; 执行该线程
        mThreadPool.execute(new Runnable() {
            @Override
            public void run() {

                try {
                    // 步骤1：创建输入流对象InputStream
                    is = socket.getInputStream();

                    // 步骤2：创建输入流读取器对象 并传入输入流对象
                    // 该对象作用：获取服务器返回的数据
                    isr = new InputStreamReader(is);
                    br = new BufferedReader(isr);

                    // 步骤3：通过输入流读取器对象 接收服务器发送过来的数据
                    response = br.readLine();

                    // 步骤4:通知主线程,将接收的消息显示到界面
                    Message msg = Message.obtain();
                    m
```

```
g.what = 0;
</span></span><span class="highlight-line"><span class="highlight-cl">
MainHandler.sendMessage(msg);
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">                } catc
 (IOException e) {
</span></span><span class="highlight-line"><span class="highlight-cl">                    e.pr
ntStackTrace();
</span></span><span class="highlight-line"><span class="highlight-cl">                }
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">        });
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">});
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">/**
</span></span><span class="highlight-line"><span class="highlight-cl"> * 发送消息 给
务器
</span></span><span class="highlight-line"><span class="highlight-cl"> */
</span></span><span class="highlight-line"><span class="highlight-cl">btnSend.setOnC
ickListener(new View.OnClickListener() {
</span></span><span class="highlight-line"><span class="highlight-cl">    @Override
</span></span><span class="highlight-line"><span class="highlight-cl">    public void o
Click(View v) {
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">        // 利用线
池直接开启一个线程 &amp;amp; 执行该线程
</span></span><span class="highlight-line"><span class="highlight-cl">        mThreadP
ol.execute(new Runnable() {
</span></span><span class="highlight-line"><span class="highlight-cl">            @Overr
de
</span></span><span class="highlight-line"><span class="highlight-cl">            public v
id run() {
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">                try {
</span></span><span class="highlight-line"><span class="highlight-cl">                    //
骤1：从Socket 获得输出流对象OutputStream
</span></span><span class="highlight-line"><span class="highlight-cl">                    //
对象作用：发送数据
</span></span><span class="highlight-line"><span class="highlight-cl">                    ou
putStream = socket.getOutputStream();
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">                    //
骤2：写入需要发送的数据到输出流对象中
</span></span><span class="highlight-line"><span class="highlight-cl">                    ou
putStream.write((mEdit.getText().toString()+"\n").getBytes("utf-8"));
</span></span><span class="highlight-line"><span class="highlight-cl">                    //
别注意：数据的结尾加上换行符才可让服务器端的readline()停止阻塞
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">                    //
骤3：发送数据到服务端
</span></span><span class="highlight-line"><span class="highlight-cl">                    ou
```

```
putStream.flush();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }

                }
            });

        }
    });


    /**
     * 断开客户端 &amp;amp; 服务器的连接
     */
    btnDisconnect.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            try {
                // 断开客户端发送到服务器 的连接，即关闭输出流对象OutputStream
                outputStream.close();

                // 断开服务器发送到客户端 的连接，即关闭输入流读取器对象BufferedReader
                br.close();

                // 最终闭整个Socket连接
                socket.close();

                // 判断客户端和服务器是否已经断开连接
                System.out.println(socket.isConnected());

            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    });
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>}</p>
<p>1<br>
2<br>
3<br>
4<br>
5<br>
6<br>
7<br>
8<br>
9<br>
10<br>
11<br>
12<br>
13<br>
14<br>
15<br>
16<br>
17<br>
18<br>
19<br>
20<br>
21<br>
22<br>
23<br>
24<br>
25<br>
26<br>
27<br>
28<br>
29<br>
30<br>
31<br>
32<br>
33<br>
34<br>
35<br>
36<br>
37<br>
38<br>
39<br>
40<br>
41<br>
42<br>
43<br>
44<br>
45<br>
46<br>
47<br>
48<br>
49<br>
```

50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103

104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157

158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211

212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
6.2 服务器 实现
因本文主要讲解客户端，所以服务器仅仅是为了配合客户端展示；
为了简化服务器使用，此处采用 Mina 框架
服务器代码请在 eclipse 平台运行
按照我的步骤一步步实现就可以无脑运行了
步骤 1：导入 Mina 包
请直接移步到百度网盘：下载链接（密码: q73e）
步骤 2：创建服务器线程
TestHandler.java
package mina;
// 导入包
public class TestHandler extends IoHandlerAdapter {
<pre><code class="highlight-chroma"><span class="highlight
cl">@Override
public void except
onCaught(IoSession session, Throwable cause) throws Exception {
    System.out.print
n("exceptionCaught: " + cause);
}

@Override
public void messa
eReceived(IoSession session, Object message) throws Exception {
    System.out.print

```
n("recieve : " + (String) message);
</span></span><span class="highlight-line"><span class="highlight-cl">    session.write("h
llo I am server");
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">@Override
</span></span><span class="highlight-line"><span class="highlight-cl">public void messa
eSent(IoSession session, Object message) throws Exception {
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">@Override
</span></span><span class="highlight-line"><span class="highlight-cl">public void sessio
Closed(IoSession session) throws Exception {
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.print
n("sessionClosed");
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">@Override
</span></span><span class="highlight-line"><span class="highlight-cl">public void sessio
Opened(IoSession session) throws Exception {
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.print
n("sessionOpen");
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">@Override
</span></span><span class="highlight-line"><span class="highlight-cl">public void sessio
Idle(IoSession session, IdleStatus status) throws Exception {
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>}<br>
1<br>
2<br>
3<br>
4<br>
5<br>
6<br>
7<br>
8<br>
9<br>
10<br>
11<br>
12<br>
13<br>
14<br>
15<br>
16<br>
17<br>
18<br>
19<br>
20<br>
21<br>
22<br>
23<br>
```

24
25
26
27
28
29
30
31
32
33
34
35
36
37
步骤 3：创建服务器主代码
TestHandler.java
package mina;
import java.io.IOException;
import java.net.InetSocketAddress;
import org.apache.mina.filter.codec.ProtocolCodecFilter;
import org.apache.mina.filter.codec.textline.TextLineCodecFactory;
import org.apache.mina.transport.socket.nio.NioSocketAcceptor;
public class TestServer {
public static void main(String[] args) {
NioSocketAcceptor acceptor = null;
try {
acceptor = new NioSocketAcceptor();
acceptor.setHandler(new TestHandler());
acceptor.getFilterChain().addLast("mFilter", new ProtocolCodecFilter(new TextLineCodecFacto
y()));
acceptor.setReuseAddress(true);
acceptor.bind(new InetSocketAddress(8989));
} catch (Exception e) {
e.printStackTrace();
}
}
}
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

18
19
20
21
22
23
24
至此，客户端 & 服务器的代码均实现完毕。
6.3 测试结果
点击 Connect 按钮： 连接成功
输入发送的消息，点击 Send 按钮发送
服务器接收到客户端发送的消息
点击 Receive From Message 按钮，客户端 读取 服务器返回的消息
点击 DisConnect 按钮，断开 客户端 & 服务器的连接
<h2 id="6-4-源码地址Carson-Ho的Github地址-Socket具体实例">6.4 源码地址
Carson_Ho 的 Github 地址：Socket 具体实例</h2>
作者：Carson_Ho
来源：CSDN
原文：https://blog.csdn.et/carson_ho/article/details/53366856
