

云南大学数学与统计学院

《计算机网络实验》上机实践报告

课程名称：计算机网络实验	年级：2015 级	上机实践成绩：
指导教师：陆正福	姓名：刘鹏	专业：信息与计算科学
上机实践名称：基于 TCP 协议与 Socket 接口的网络通信编程实验	学号：20151910042	上机实践日期：2018-10-25
上机实践编号：No.03	组号：	

一、 实验目的

1. 熟悉基于 TCP 协议与 Socket 接口的网络通信编程实验
2. 熟悉教材第二章的基本概念

二、 实验内容

1. 掌握基于 TCP 协议与 Socket 接口的网络通信编程的流程；
2. 使用 Java 实现基于 TCP 协议与 Socket 接口的 PC 网络通信编程；
3. 使用 Java 和 Android 实现基于 TCP 协议与 Socket 接口的移动网络通信编程（**选做**）；
4. 使用 Python 实现基于 TCP 协议与 Socket 接口的网络通信编程（**选做**）；

说明：3 和 4 至少选做 1 项；调试所用实例可以源自书本和网络，但是应有属于自己的修改，不可以照搬照抄。

三、 实验平台

Windows 10 Pro 1803;
MacOS Mojave 10.14;
Cygwin GCC 编译器;
Android Studio 3.2.1 for MacOS;
Android Studio 3.2.1 for Windows;

四、 程序代码

1.1 基于 TCP 协议与 Socket 接口的网络编程流程（叙述内容）

- （1）**配置服务器** 网络编程中比较经典的场景是 Client-Server 模式，P2P 模式可以视为 Client-Server 模式的一种拓展。把服务器的 IP 确定下来，保持平稳运行、网络畅通、无故障。
- （2）**配置服务器端代码** 服务端的负责通讯的代码比较简单。TCP 通信时，需要创建一个 Socket，然后时刻监听。具体步骤如下：

```
1. 新建serverSocket对象, 并指定端口
serverSocket = new ServerSocket(port)

2. 进行监听
socket = serverSocket.accept()

3. 拿到输入流 (客户端发来的消息)
InputStream = socket.getInputStream()

4. 解析数据
reader = new InputStreamReader(inputStream)
bufReader = new BufferedReader(reader)
一系列操作

5. 关闭输入流
socket.shutdownInput()

6. 拿到输出流
OutputStream = socket.getOutputStream()
OutputStream.someOperations ()
OutputStream.flush()

7. 关闭输出流
socket.shutdownOutput()
OutputStream.close()

8. 关闭IO资源
bufReader.close()
reader.close()
InputStream.close()

9. 关闭socket
socket.close()
serverSocket.close()
```

- (3) **编写客户端程序** 客户端的程序需要精心编写, 而且要在获知服务器地址 (或者有解析的域名) 的情况下进行。由于客户端需要设计好用的 UI, 所以在 Java 中牵扯到了创建进程对象, 并调用其 start 方法。具体实现细节可以参看具体的代码。

这里的客户端使用 Android 模拟了一个手机 App, 仅仅是 echo 一个字符串。除此之外没有运算。

```

1. 直接创建socket对象
socket = new Socket(ip_address, port)

2. 拿到客户端要发送的数据流
outputStream = socket.getOutputStream()

3. 写入要发给服务器的数据
outputStream.someOperations ()

4. 关闭输出流
outputStream.flush()
socket.shutdownOutput()

5. 经过些许等待，拿到服务器返回的数据，即输入流
inputStream = socket.getInputStream()

6. 解析服务器返回的数据

7. 关闭IO资源

8. 关闭socket
inputStream.close()
outputStream.close()
socket.close()

```

(4) 测试与客户端的通信是否正常。

1.2 使用 Java 实现基于 TCP 协议与 Socket 接口的网络编程（给出实例）

主要把 Java 给出的 Socket 用好就可以了。详见 1.1。

1.3 使用 Java 实现基于 TCP 协议与 Socket 接口的网络编程

本选做题选择 Java，因为 Java 比较全面，各种库比较完善，而且在应用中，Java 也是核心语言，相比 Python 就比较弱势。本例主要参考 <https://www.jianshu.com/p/fb4dfab4eecd> 的相关指导与说明。应用层的编程，不需要考虑如何建立路由表、防止比特错误以及纠错，只需要考虑建立连接以及通过连接收发消息。这类似于通过邮件系统收发邮件，需要做的仅仅是知道对方的邮箱地址、邮编。

本次使用的 Socket 是由 TCP 协议规定的、由 Java 语言实现的一种面向应用层的接口。

1.3.1 应用设计

服务器开启一个端口号为 2222 的 TCP 进程，一直监听来自外部的访问，然后经过简单的计算给出回答。而这里假设的外部客户端是一台手机上的 App。本例没有采取任何复杂的计算，只是单纯地回复一个字符串。

1.3.2 实验过程分析

由于实验中需要设计一个 App，所以比较理想的状态是采取开源的 Android 平台，用 Java 语言进行开发。这个 App 十分简易，但是需要连接网络，所以在 App 的设计中，需要赋予其连接网络的权限。Android 应用需要用触摸交互式操作，所以设计对 UI 对象的操作，即把对基本 UI 组件的操作映射到相应的代码块上。

1.3.3 Server 端 Java 代码

```
1 import java.io.BufferedReader;
```

```
2  import java.io.IOException;
3  import java.io.InputStream;
4  import java.io.InputStreamReader;
5  import java.io.OutputStream;
6  import java.net.ServerSocket;
7  import java.net.Socket;
8
9  public class Server {
10
11      public static void main(String[] args) {
12
13          try {
14              // 为了看流程, 我就把所有的代码都放在 main 函数里了,
15              // 也没有捕捉异常, 直接抛出去了。实际开发中不可取。
16              // 1.新建 ServerSocket 对象, 创建指定端口的连接
17              ServerSocket serverSocket = new ServerSocket(2222);
18              System.out.println("服务端监听开始了~~~~");
19
20              // 2.进行监听
21              Socket socket = serverSocket.accept();
22
23              // 3.拿到输入流 (客户端发送的信息就在这里)
24              InputStream is = socket.getInputStream();
25
26              // 4.解析数据
27              InputStreamReader reader = new InputStreamReader(is);
28              BufferedReader bufReader = new BufferedReader(reader);
29              String s = null;
30              StringBuffer sb = new StringBuffer();
31              while ((s = bufReader.readLine()) != null) {
32                  sb.append(s);
33              }
34              System.out.println("服务器: " + sb.toString());
35
36              // 关闭输入流
37              socket.shutdownInput();
38
39              OutputStream os = socket.getOutputStream();
40              os.write(("我是服务端, 客户端发给我的数据就是: " + sb.toString()).getBytes());
41              os.flush();
42              // 关闭输出流
43              socket.shutdownOutput();
44              os.close();
45
46              // 关闭 IO 资源
47              bufReader.close();
48              reader.close();
49              is.close();
50
51              socket.close();// 关闭 socket
52              serverSocket.close();// 关闭 ServerSocket
53
54          } catch (IOException e) {
55              e.printStackTrace();
56          } catch (Exception e) {
57              e.printStackTrace();
58          }
59      }
```

在后来的 Windows 平台的调试过程中，我发现由于 JDF for Windows 是只识别 GBK 编码，直接从 MacOS 拷贝过来的 Java 源代码是 UTF-8 编码，是无法编译成字节码的。然而，用 Visual Studio Code 将字符集更改为 GB2312 之后，虽然可以编译成功，但是客户端返回的中文字符又变成了乱码。可见，服务端的输出在 Windows 平台下是 GBK 输出。

解决这个问题的方法有很多，下面试列举两个。（1）所有的代码、注释都用英文，这样可以绝对保证不会出问题；（2）统一平台的编码，更改 Android Studio 的字符集比较困难，最好是令 JDK 的字符集更改为默认 UTF-8，这可以通过添加环境变量：变量名 `JAVA_TOOL_OPTIONS`，变量值 `-Dfile.encoding=UTF-8`，改完之后就不存在这种问题了。

最后，由于这个工程的系统代码太多，有很多 XML 文件，所以这里仅仅贴出核心代码，具体工程文件参见 GitHub 仓库。

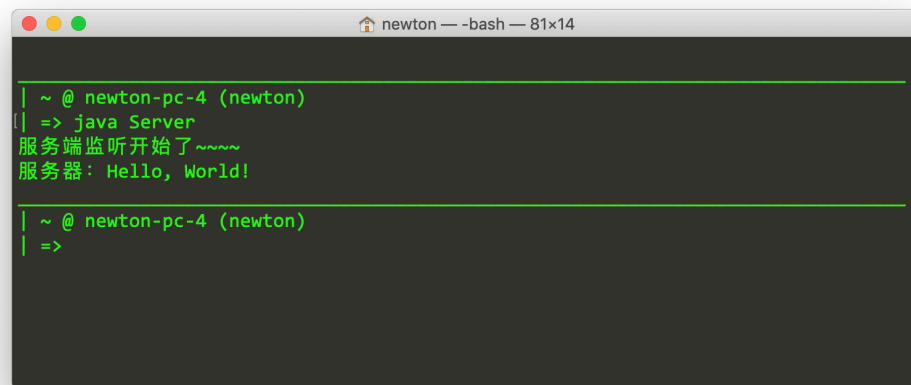
1.3.4 Android App 端 Java 代码

```
1  package com.example.newton.myapplication;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5
6  import java.io.BufferedReader;
7  import java.io.IOException;
8  import java.io.InputStream;
9  import java.io.InputStreamReader;
10 import java.io.OutputStream;
11 import java.net.Socket;
12
13 import android.view.View;
14
15 import android.widget.Button;
16 import android.widget.EditText;
17 import android.widget.TextView;
18
19 import java.net.UnknownHostException;
20
21 public class MainActivity extends AppCompatActivity {
22
23     EditText et;
24     TextView tv;
25     Button button;
26
27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.activity_main);
31
32         et = findViewById(R.id.et);
```

```
33     tv = findViewById(R.id.tv);
34     button = findViewById(R.id.button);
35 }
36
37 public void onClick(View view){
38     new Thread(){
39         @Override
40         public void run() {
41             super.run();
42             try {
43                 // 1.创建监听指定服务器地址以及指定服务器监听的端口号
44                 Socket socket = new Socket("192.168.1.79", 2222);
45
46                 // 2.拿到客户端的 socket 对象的输出流发送给服务器数据
47                 OutputStream os = socket.getOutputStream(); // 客户机的 output
48
49                 // 写入要发送给服务器的数据
50                 os.write(et.getText().toString().getBytes());
51
52                 // System.out.println(os.toString());
53
54                 os.flush();
55                 socket.shutdownOutput();    // OK, 不再需要写字了, 准备发送吧!
56
57                 // 拿到 socket 的输入流, 这里存储的是服务器返回的数据
58                 InputStream is = socket.getInputStream();
59
60                 // 解析服务器返回的数据
61                 InputStreamReader reader = new InputStreamReader(is);
62                 BufferedReader bufReader = new BufferedReader(reader);
63                 String s = null;
64                 final StringBuffer sb = new StringBuffer();
65                 while((s = bufReader.readLine()) != null){
66                     sb.append(s);
67                 }
68                 runOnUiThread(new Runnable() {
69                     @Override
70                     public void run() {
71                         tv.setText(sb.toString());
72                     }
73                 });
74
75                 // 3、关闭 IO 资源
76                 bufReader.close();
77                 reader.close();
78                 is.close();
79                 os.close();
80                 socket.close();
81             } catch (UnknownHostException e) {
```

```
82         e.printStackTrace();
83     } catch (IOException e) {
84         e.printStackTrace();
85     }
86 }
87 }.start();
88
89 }
90 }
```

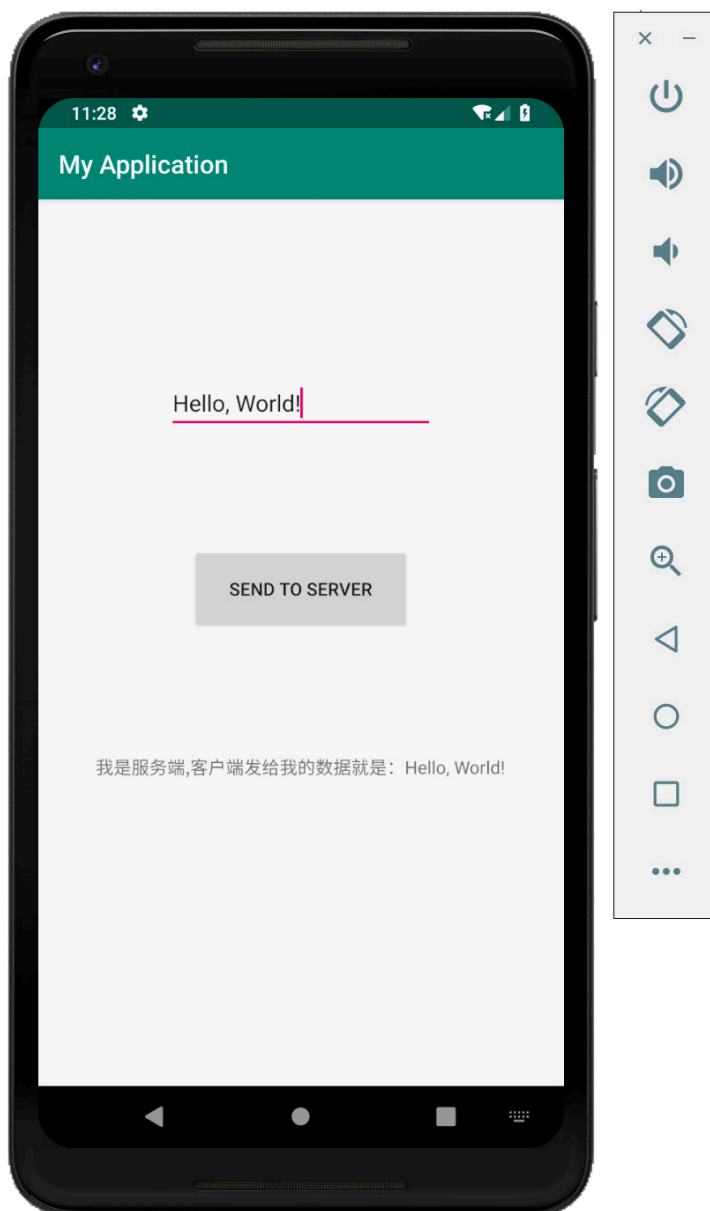
1.3.5 Server 端运行结果



```
newton -- -bash -- 81x14
| ~ @ newton-pc-4 (newton)
| => java Server
服务端监听开始了~~~~~
服务器: Hello, World!
| ~ @ newton-pc-4 (newton)
| =>
```

本次实验没有采用多机环境，一是考虑到多机环境配置网络可能比较繁琐，再者是单机的不同进程之间也可以通过 Socket 进行网络通信，所以在一台机器上进行了实验。

1.3.6 Android 客户端运行结果



五、 实验体会

Android 编程比较面向应用，对于实现某个特定的功能考虑较多，最初对于安卓的开发环境非常不熟悉，后来逐步完成代码与 UI 设计，中间遇到了一些困难，但是还可以通过考虑系统的可能设计加上搜索由此而得到的问题逐一解决。

六、 参考文献

- [1] <https://www.jianshu.com/p/fb4dfab4eec1>
- [2] java NIO: <https://www.jianshu.com/p/093b7c408dba>
- [3] java NIO: <https://docs.oracle.com/javase/7/docs/api/java/nio/package-summary.html>
- [4] java NET: <https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html>