

# 深圳大学实验报告

课程名称： 计算机网络

实验项目名称： Socket 网络编程

学院： 计算机与软件学院

专业： 计算机科学与技术

指导教师： 李雪亮

报告人： 林宪亮 学号： 2022150130 班级： 国际班

实验时间： 2024 年 4 月 3 日

实验报告提交时间： 2024 年 4 月 4 日

教务处制

### 实验目的：

掌握 Socket 的 TCP 通信、 Socket 的 UDP 通信。

### 实验环境：

使用 Windows 操作系统； Internet 连接  
Eclipse+Java 开发环境

### 实验内容：

Socket、ServerSocket 类和 DatagramPacket 、DatagramSocket 类的使用。

### 实验步骤：

（用文字描述实验过程，可用截图辅助说明）

1. 利用 Socket 类和 ServerSocket 类编写一个 C/S 程序，实现 C/S 通信。

**C/S 通信：**C/S 通信是指 **Client**（客户端）和 **Server**（服务器）的通信，这两个角色之间通过网络进行通信，完成数据交换和信息传递。

核心代码：

• 客户端：

```
1.    // 创建 Socket 对象，指定连接的服务器地址和端口号
2.    Socket socket = new Socket("localhost", 8888);
3.
4.    // 获取输入流和输出流
5.    BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
6.    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
7.
8.    // 发送消息给服务器
```

```
9.    out.println("我是用户，我需要使用服务器!");
10.
11.    // 接收服务器返回的消息
12.    String response = in.readLine();
13.    System.out.println("服务器返回的消息: " + response);
14.
15.    // 关闭资源
16.    in.close();
17.    out.close();
18.    socket.close();
```

#### • 服务器端

```
1.    // 创建 ServerSocket 对象，指定端口号为 8888
2.    ServerSocket serverSocket = new ServerSocket(8888);
3.
4.    System.out.println("等待客户端连接...");
5.    // 调用 accept() 方法接收客户端连接，程序将会阻塞在这里直到有
    客户端连接进来
6.    Socket clientSocket = serverSocket.accept();
7.    System.out.println("客户端已连接");
8.
9.    // 获取输入流和输出流
10.   BufferedReader in = new BufferedReader(new InputStreamReader(
    clientSocket.getInputStream()));
11.   PrintWriter output = new PrintWriter(clientSocket.getOutputStream(),
    true);
12.
13.   String line;
14.   while ((line = in.readLine()) != null) {
15.       System.out.println("客户端发送的消息: " + line);
16.       // 向客户端发送消息
17.       output.println(line);
18.   }
19.
20.   // 关闭资源
21.   in.close();
22.   output.close();
23.   clientSocket.close();
24.   serverSocket.close();
```

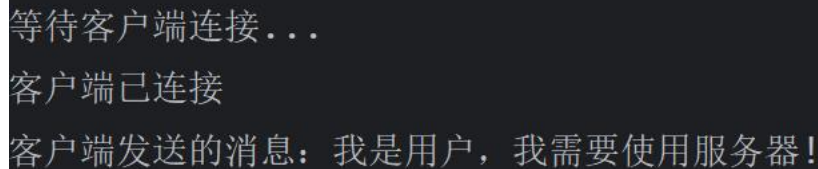
说明：

服务器端创建一个 `ServerSocket` 对象，指定端口号为 8888，之后使用 `accept` 方法接收客户端发送来的信息。使用 `BufferedReader` 类获取输入流，使用 `PrintWriter` 类向客户端返回信息。

客户端使用 `Socket` 类指定连接本地服务器的 8888 端口，同样的使用 `PrintWriter` 和 `BufferedReader` 发送信息给服务器端并且接收服务器端返回的信息。

输出结果：

- 服务器端：



```
等待客户端连接...
客户端已连接
客户端发送的消息：我是用户，我需要使用服务器！
```

图 1 服务器端输出

服务器等待客户连接，并在接收到客户发送的信息之后返回客户发送的信息，依次告知客户我已经接收到了信息。

- 客户端：



```
服务器返回的消息：我是用户，我需要使用服务器！
```

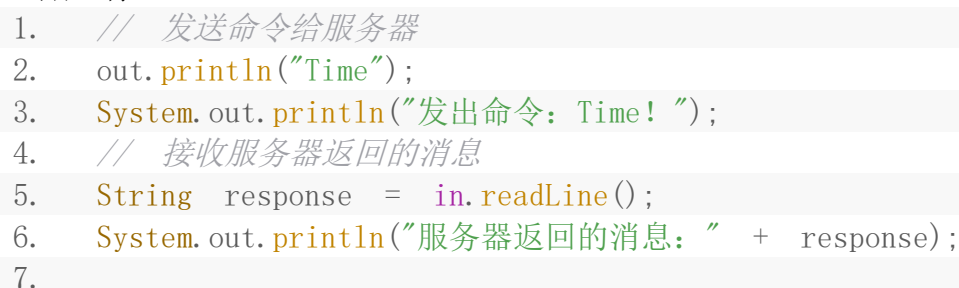
图 2 客户端输出

客户端接收到服务器发送的确认信息，确认自己发送的信息客户端已经接收到了。

2. 客户端向服务器端发送 `Time` 命令，服务器端接受到该字符串后将服务器端当前时间返回给客户端；客户端向服务器端发送 `Exit` 命令，服务器端向客户端返回 “Bye” 后退出。

核心代码：

- 客户端：



```
1. // 发送命令给服务器
2. out.println("Time");
3. System.out.println("发出命令：Time！");
4. // 接收服务器返回的消息
5. String response = in.readLine();
6. System.out.println("服务器返回的消息：" + response);
7.
```

```

8.    // 发送 Exit 命令给服务器
9.    out.println("Exit");
10.   // 接收服务器返回的消息
11.   System.out.println("发出命令: Exit");
12.   response = in.readLine();
13.   System.out.println("服务器返回的消息: " + response);

• 服务器端:
1.   while ((line = in.readLine()) != null) {
2.       System.out.println("客户端发送的命令: " + line);
3.
4.       if (line.equalsIgnoreCase("Time")) {
5.           // 获取当前时间并向客户端发送
6.           SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
7.           String currentTime = dateFormat.format(new Date());
8.           out.println("服务器当前时间: " + currentTime);
9.           System.out.println("返回给客户端信息: "+currentTime);
10.      } else if (line.equalsIgnoreCase("Exit")) {
11.          // 向客户端发送 "Bye" 并退出循环
12.          out.println("Bye");
13.          System.out.println("返回给客户端的信息: Bye!");
14.          break;
15.      } else {
16.          // 未识别的命令
17.          out.println("未识别的命令: " + line);
18.      }
19.  }

```

说明:

大体步骤和第一题一致。客户端只需要注意在发送命令的同时打印出命令信息即可，其它部分和第一题一致。服务器端则使用分支语句，根据客户端输入的不同的命令，返回不同的信息（判断是 Time 命令则返回当前时间，判断是 Bye 命令则退出程序）。注意输出接收的命令和返回的信息即可。

输出结果：

- 客户端：

```
发出命令: Time!  
服务器返回的消息: 服务器当前时间: 2024-04-03 22:41:21  
发出命令: Exit  
服务器返回的消息: Bye
```

图 3 客户端打印信息

当客户端发送 Time 指令时，可以接收到服务器返回的当前时间信息。当发出 Exit 指令则接收到服务器发送的 Bye 语句并终止通信。

- 服务器端：

```
等待客户端连接...  
客户端已连接  
客户端发送的命令: Time  
返回给客户端信息: 2024-04-03 22:41:21  
客户端发送的命令: Exit  
返回给客户端的信息: Bye!
```

图 4 服务器端打印信息

服务器端接收到 Time 指令时会返回当前的时间信息，接收到 Exit 指令时，会返回 Bye 语句并终止此次通信。

3. 运行数据报通信样例程序，理解程序所表达的意思。

- 服务器端：

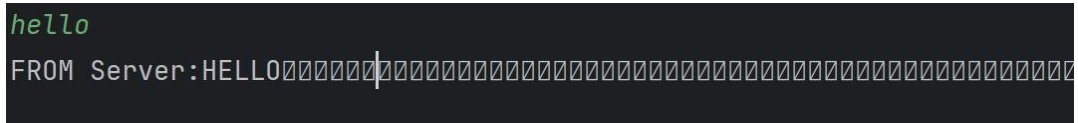
代码实现了一个简单的 UDP 服务器，它会不断地接收客户端发送的数据报，并将收到的数据报转换成大写形式后发送回客户端。服务器端使用 `DatagramSocket` 对象监听指定的端口，然后通过 `DatagramPacket` 接收和发送数据报。

- 客户端：

客户端从控制台读取输入，然后将输入发送给服务器端。客户端使用 `DatagramSocket` 对象来发送和接收数据报。发送的数据通过 `DatagramPacket` 发送到服务器端的指定端口，接收到的数据则通过另一个 `DatagramPacket` 接收。最后，客户端关闭。

运行结果：

- 客户端：



```
hello
FROM Server:HELLO
```

图 5 客户端

客户端会输出服务器端返回的大写字符串。

客户端数据报接收功能：

```
// 创建用于接收数据的数据报
DatagramPacket receivePacket =
    new DatagramPacket(receiveData, receiveData.length);

// 从服务器接收数据报
clientSocket.receive(receivePacket);

// 将接收到的数据转换为字符串
String modifiedSentence =
    new String(receivePacket.getData());

// 打印服务器返回的数据
System.out.println("FROM Server:" + modifiedSentence);
```

图 6 客户端数据报接收

## 实验结果:

(本页附完成的实验结果、并给出个人对结果的分析、结论)

本次实验,我学会了 Socket、ServerSocket 类和 DatagramPacket, DatagramSocket 类的使用。使用这些类实现了客户端和服务端通信,以及客户端和服务端数据包的通信,实验完成成功。

Socket 类提供了一种双向通信通道,允许应用程序通过网络发送和接收数据。客户端通过创建 Socket 对象与服务端建立连接,然后通过该连接发送和接收数据。

### • 分析

ServerSocket 类用于在服务端监听并接受客户端的连接请求。服务端应用程序通过 ServerSocket 对象监听指定的端口,等待客户端连接。一旦客户端连接成功,ServerSocket 就会为该连接创建一个新的 Socket 对象,使得服务端和客户端之间可以进行通信。

DatagramSocket 类提供了使用 UDP 协议进行数据传输的功能。UDP 是一种无连接的协议,它不保证数据包的顺序和可靠性,但传输速度比 TCP 快。DatagramSocket 类用于发送和接收 DatagramPacket 对象,这些对象包含了要发送或接收的数据以及目标地址和端口信息。

DatagramPacket 类用于在 UDP 协议中封装数据。DatagramPacket 对象包含了数据本身、数据的长度以及目标地址和端口信息。客户端通过 DatagramSocket 发送 DatagramPacket 对象,服务端通过 DatagramSocket 接收 DatagramPacket 对象。

### • 结论:

Socket 和 ServerSocket 类主要用于 TCP 协议,提供可靠的、面向连接的通信机制。

DatagramSocket 和 DatagramPacket 类主要用于 UDP 协议,提供不可靠的、面向数据包的通信机制,适用于实时性要求高、对数据传输顺序和可靠性要求不高的场景。



## 实验小结:

(实验中出现问题解决方法, 实验心得体会等)

本次实验主要为了掌握四个关于客户端服务器端通信的类的使用, 类的概念, 使用方法复杂, 深究需要很多时间, 但是根据提供的示例代码进行理解, 知道代码的每一行都做了什么, 可以更块的上手。

## 指导教师批阅意见:

## 成绩评定:

指导教师签字:

年 月 日

备注: