目录

1 实验目的与要求

网络编程是通过使用套接字来达到进程间通信目的的编程，Socket 编程是网络编程的主流工具，Socket API 是实现进程间通信的一种编程设施，也是一种为进程间提供底层抽象的机制，提供了访问下层通信协议的大量系统调用和相应的数据结构。本实验利用 Socket API 编写网络通信程序

1. 掌握 C++、JAVA 或 Python 等集成开发环境编写网络程序的方法
2. 掌握客户/服务器（C/S）应用的工作方式；
3. 学习网络中进程之间通信的原理和实现方法；
4. 要求本机既是客户端又是服务器端；

2 实验原理与实验内容

2.1 实验原理

2.1.1 Socket 编程接口

要实现 Web 服务器，需使用套接字 Socket 编程接口来使用操作系统提供的网络通信功能。Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，是一组编程接口。它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。使用 Socket 后，无需深入理解 TCP/UDP 协议细节（因为 Socket 已经为我们封装好了），只需要遵循 Socket 的规定去编程，写出的程序自然就是遵循 TCP/UDP 标准的。Socket 的地位如下图所示：

从某种意义上说，Socket 由地址 IP 和端口 Port 构成。IP 是用来标识互联网中的一台主机的位置，而 Port 是用来标识这台机器上的一个应用程序，IP 地址是配置到网卡上的，而 Port 是应用程序开启的，IP 与 Port 的绑定就标识了互联网中独一无二的一个应用程序。

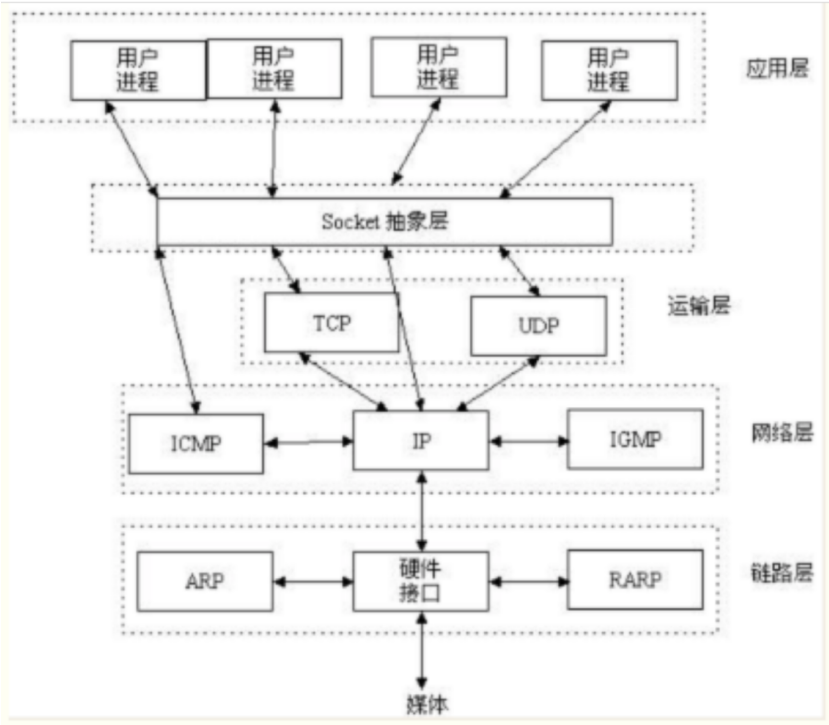

图 1: Socket 的地位

套接字类型流式套接字 (SOCK_STREAM): 用于提供面向连接、可靠的数据传输服务。数据报套接字(SOCK_DGRAM): 提供了一种无连接的服务。该服务并不能保证数据传输的可靠性, 数据有可能在传输过程中丢失或出现数据重复, 且无法保证顺序地接收到数据。原始套接字(SOCK_RAW): 主要用于实现自定义协议或底层网络协议。

在本 WEB 服务器程序实验中，采用流式套接字进行通信。其基本模型如下图所示:

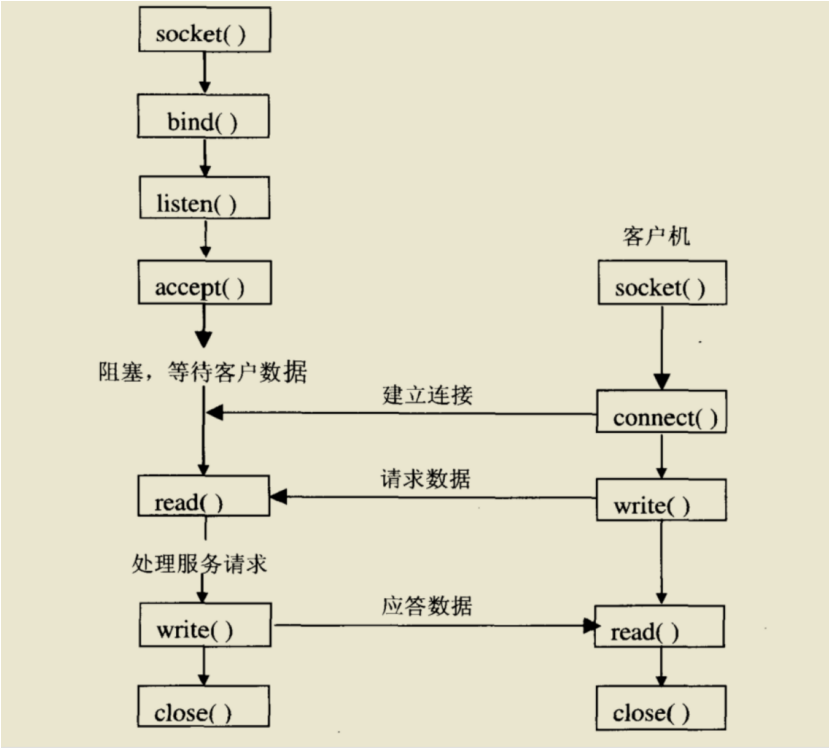

图 2: Socket 的基本模型

其工作过程如下：服务器首先启动，通过调用 socket () 建立一个套接字，然后调用绑定方法 bind () 将该套接字和本地网络地址联系在一起，再调用 listen () 使套接字做好侦听连接的准备，并设定的连接队列的长度。客户端在建立套接字后，就可调用连接方法 connect () 向服务器端提出连接

请求。服务器端在监听到连接请求后，建立和该客户端的连接，并放入连接队列中，并通过调用 accept () 来返回该连接，以便后面通信使用。客户端和服务器连接一旦建立，就可以通过调用接收方法 recv () / recvfrom () 和发送方法 send () / sendto () 来发送和接收数据。最后，待数据传送结束后，双方调用 close () 关闭套接字。

2.1.2 端口扫描

端口号的主要作用是表示一台计算机中的特定进程所提供的服务。网络中的计算机是通过 IP 地址来代表其身份的，它只能表示某台特定的计算机，但是一台计算机上可以同时提供很多个服务，如数据库服务、FTP 服务、Web 服务等，我们就通过端口号来区别相同计算机所提供的这些不同的服务，如常见的端口号 21 表示的是 FTP 服务，端口号 23 表示的是 Telnet 服务，端口号 25 指的是 SMTP 服务等。

TCP 与 UDP 段结构中端口地址都是 16 比特，可以有在 0—65535 范围内的端口号。

TCP 是面向连接的运输层协议。应用程序在使用 TCP 协议之前，必须先建立 TCP 连接。在传送数据完毕后，必须释放已经建立的 TCP 连接。TCP 面向字节流，“流”指的是流入到进程或从进程流出的字节序列。

UDP 是一个简单的面向消息的传输层协议，尽管 UDP 提供标头和有效负载的完整性验证（通过校验和），但它不保证向上层协议提供消息传递，并且 UDP 层在发送后不会保留 UDP 消息的状态。因此，UDP 有时被称为不可靠的数据报协议。如果需要传输可靠性，则必须在用户应用程序中实现。

2.2 实验内容

编写一个端口扫描器（类似 nmap 的简易版本）。给定目的 ip 地址，可以扫描目的 IP 地址在哪个端口上可以接受 tcp 连接和 udp 连接

3 实验具体设计实现及结果

3.1 实验具体设计实现

3.1.1 程序结构图

根据实验相关要求，设计了本程序，程序的框架图如下所示：

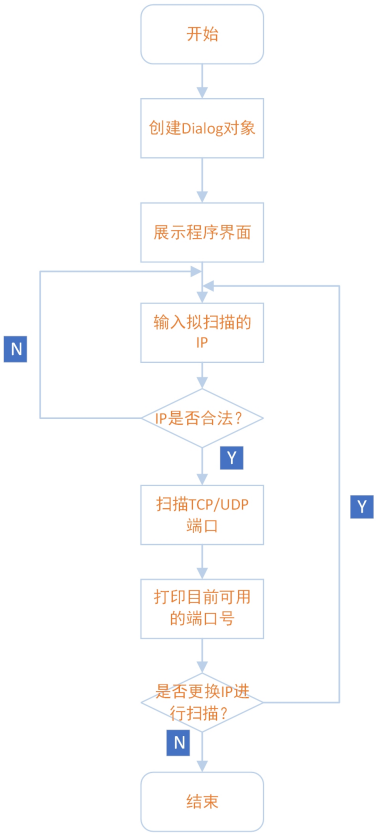

图 3: 程序框图

在 Main 函数中，首先实例化 Dialog 对象，并调用 show 函数以展示 GUI 界面。在 Dialog 类中，创建页面，读取输入的 IP 地址并判断其是否合法。若地址不合法，则提醒用户重新输入；若地址合法，则扫描对应 IP

的 TCP 端口或 UDP 端口。在扫描完毕后，用户可重新输入合法 IP 地址，并选择扫描方式，程序将按需返回对应结果。

3.1.2 Scan 类

Scan 类中主要包括 findTCP 和 findUDP 两个方法，分别用以实现 TCP 端口的扫描和 UDP 端口的扫描，本类的 UML 图如下所示：

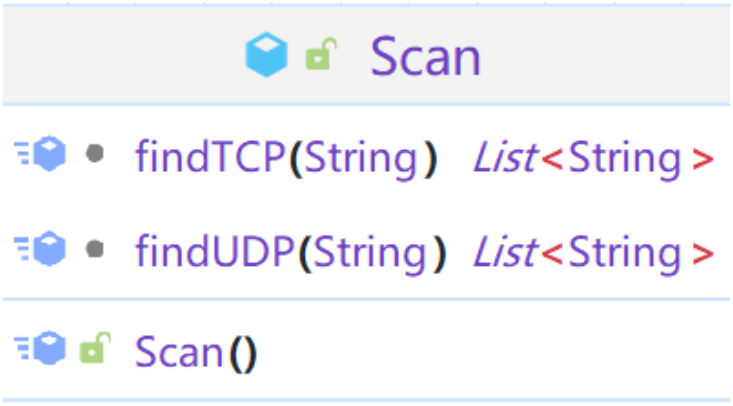

图 4: Scan 类 UML 图

findTCP 方法的关键，是基于 TCP 连接特性，采用判断指定端口是否连接成功，进而来判断对应端口是否可用。

首先创建一个 Socket 对象，然后指定目标主机的 IP 地址和端口号，采用 Socket 的 connect() 方法来建立与目标主机的连接；随后创建一个 InetSocketAddress 对象，表示要连接的 IP 地址和端口号；之后尝试连接到目的主机的指定端口，若连接超时，则判定为不可连接，即端口不可用，反之则对应 TCP 端口可用。

findUDP 方法的关键，是基于 UDP 不面向连接的特性，采用判断数据包发送是否超时，进而判断对应 UDP 端口是否可用.

首先创建一个 DatagramSocket 对象，并根据 IP 地址获取 InetAddress 对象；随后创建一个 DatagramPacket 对象，并封装要发送的数据，目的主

机 IP 地址及端口号等。然后我们发送该 UDP 数据包，若成功发送，即判断该 IP 地址的对应端口可用；反之则认为发生异常，则显示对应端口闭合。

以下是 TCP 扫描的 Java 代码：

```
List<String> findTCP(String ip) throws IOException {
        int timeout = 20;
        System.out.println(ip);
        List<String> tcps = new ArrayList<>();
        for (int port=0;port<1000;port++) {
            try (Socket socket = new Socket()) {
                // 创建一个Socket对象

                InetSocketAddress SocketAddress = new
                    InetSocketAddress(ip, port);
                //指定目标主机的IP地址和端口号, Socket的connect
                    ()方法来建立与目标主机的连接
                // 创建一个InetSocketAddress对象, 表示要连接的IP
                    地址和端口号

                socket.connect(SocketAddress, timeout);
                // 连接到目标主机的指定端口

                tcps.add("port " + port + ":open");
                // 将开放的端口添加到结果列表中
            } catch (SocketTimeoutException e) {
                tcps.add("port " + port + ":close");
                // 将关闭的端口添加到结果列表中
            } catch (SocketException e2) {
                tcps.add(e2.getMessage());
                // 将异常信息添加到结果列表中
            }
        }
        return tcps;
        // 返回TCP端口扫描的结果列表
    }
```

以下是UDP扫描的代码:

```
\begin{lstlisting}[title=findUDP,frame=shadowbox]
 List<String> findUDP(String ip) throws IOException {
        List<String> findUDP(String ip) throws
            UnknownHostException {
        List<String> udps = new ArrayList<>();
        System.out.println(ip);
        for (int port=0;port<1000;port++) {
            String udp;
            try (DatagramSocket socket = new DatagramSocket()) {
                // 创建一个DatagramSocket对象

                InetAddress address = InetAddress.getByName(ip);
                // 根据IP地址获取InetAddress对象

                byte[] sendData = new byte[1];
                DatagramPacket sendPacket = new DatagramPacket(
                    sendData, sendData.length, address, port);
                // 创建一个DatagramPacket对象, 用于发送UDP数据包
                // DatagramPacket类表示UDP数据包, 它包含了要发送
                    的数据、目标主机的IP地址和端口号等信息。通过
                    创建一个DatagramPacket对象,
                // 我们可以设置要发送的数据、目标主机的地址和端
                    口号, 然后将该数据包发送到目标主机
                socket.send(sendPacket);
                // 发送UDP数据包

                udp = "port " + port + ":open";
                // 如果没有发生异常, 则表示端口开放
            } catch (Exception e) {
                udp = "port " + port + ":close";
                // 发生异常, 则表示端口关闭
            }
            udps.add(udp);
            // 将UDP端口扫描结果添加到结果列表中
```

```
59          }
            return udps;
61          // 返回UDP端口扫描的结果列表
```

findTCP

4 实验结果

4.1 登录界面

用户运行程序后，即进入到登录后的界面，如图所示

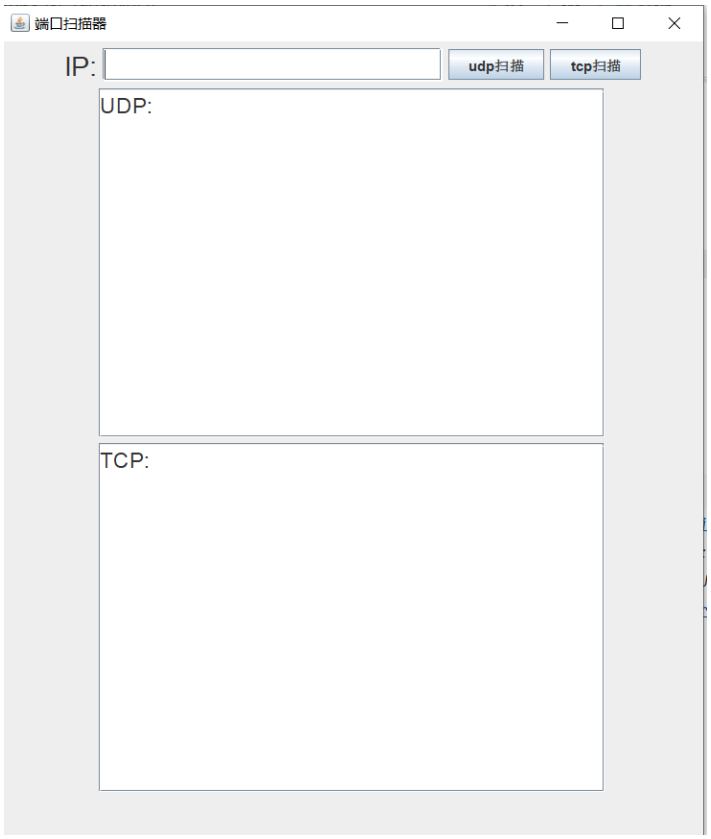

图 5: 登录界面

4.2 IP 合法性检查

程序会对输入的 IP 地址进行合法性检查，如未输入网址，而直接点击扫描端口，则程序会提醒你，请你输入正确的 IP 地址，以下功能如图所示：

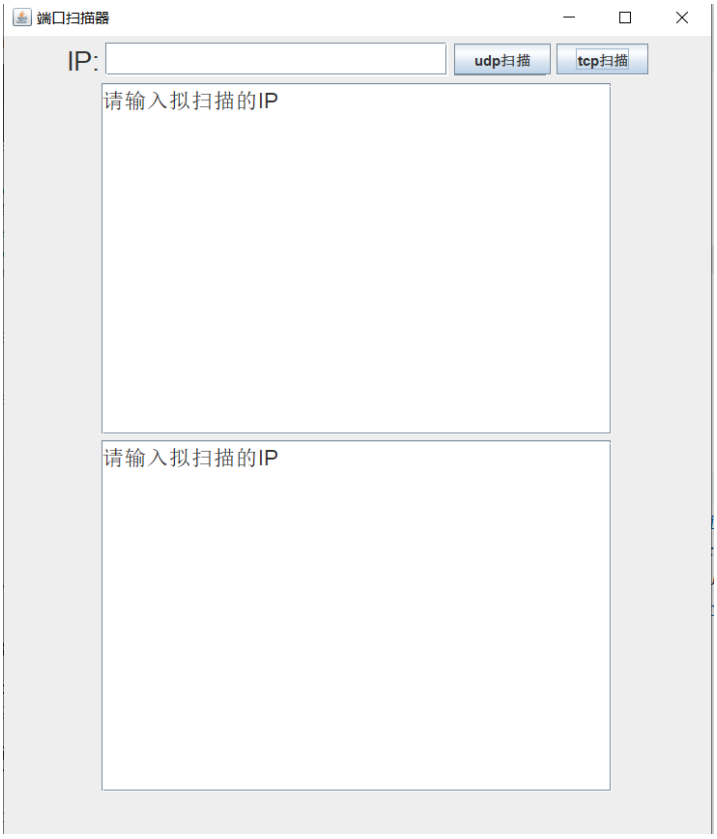

图 6: IP 合法性检查

4.3 端口检查与反馈

若输入 IP 正确，则程序会检查哪些端口是开放的，哪些端口是闭合的，并给出结果反馈。以上功能如图所示：

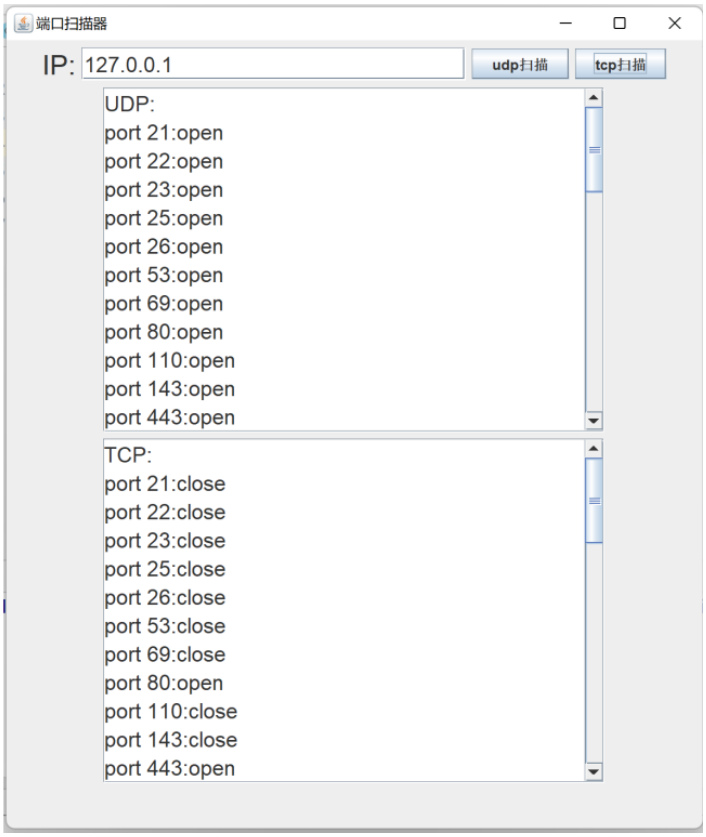

图 7: 端口检查与反馈

5 实验总结

5.1 实验环境

1. 编程语言：Java

2. 编程环境：IDEA，windows10 操作系统

5.2 实验总结

在本次实验中，程序实现了一个端口扫描器（类似 nmap 的简易版本）。给定目的 ip 地址，可以扫描目的 IP 地址在哪个端口上可以接受 tcp 连接和 udp 连接。实现过程中遇到的困难：在这个实验中，主要的困难是理解和处理 TCP,UDP 协议相关的内容。需要熟悉 TCP 和 UDP 的相关特性，另外，还需要处理文件的读取和异常情况。

通过这个实验，我对传输层的两个协议 TCP 和 UDP，有了更深入的理解。我学会了使用 Java 的 socket 库创建服务器套接字、绑定地址和端口，监听端口，并能判断某个 IP 下的哪些端口可以使用。

这个简单的扫描端口还有很大的改进空间。例如，可以引入并发处理多个客户端连接的能力，提高服务器的并发性能。此外，还可以考虑对请求参数进行解析和处理，增加服务器的功能和灵活性。

总体而言，这个实验帮助我更好地理解了传输层协议的基本原理和实现方式。通过实际编码和调试的过程，我对 TCP，UDP 协议和 socket 编程有了更深入的了解，为以后进一步探索计算机网络编程打下了坚实的基础。

6 附录

6.1 B3 源码

```
public class Main {
    public static void main(String[] args) {
        Dialog dialog = new Dialog();
        dialog.show();
    }
}

import java.io.IOException;
import java.net.*;
import java.util.ArrayList;
import java.util.List;

public class Scan {
    int[] UDP_ports = new int[] { 1, 2, 21, 22, 23, 25, 26,
        53,69, 80, 110, 143,
            443,465,69,161,162,135,995,1080,1158,1433,1521,2100,
                3128, 3306, 3389,
            7001, 8080, 8081, 9080, 9090, 43958
                ,135,445,1025,1026,1027,1028,1055,5357,138,12316};

    int[] TCP_PORT = new int[]{135, 136, 445, 446, 902, 912,
        2179, 2383, 3306, 3580, 5040, 10020, 49664, 49667,
        59110};

    List<String> findTCP(String ip) throws IOException {
        int timeout = 20;
        System.out.println(ip);
        List<String> tcps = new ArrayList<>();
        for (int port=0;port<1000;port++) {
            try (Socket socket = new Socket()) {
```

```
26                  // 创建一个Socket对象
28                  InetSocketAddress SocketAddress = new
                        InetSocketAddress(ip, port);
                    //指定目标主机的IP地址和端口号, Socket的connect
                        ()方法来建立与目标主机的连接
30                  // 创建一个InetSocketAddress对象, 表示要连接的IP
                        地址和端口号
32                  socket.connect(SocketAddress, timeout);
                    // 连接到目标主机的指定端口
34
                    tcps.add("port " + port + ":open");
36                  // 将开放的端口添加到结果列表中
                } catch (SocketTimeoutException e) {
38                  tcps.add("port " + port + ":close");
                    // 将关闭的端口添加到结果列表中
40              } catch (SocketException e2) {
                    tcps.add(e2.getMessage());
42                  // 将异常信息添加到结果列表中
                }
44          }
            return tcps;
46          // 返回TCP端口扫描的结果列表
        }
48
        List<String> findUDP(String ip) throws UnknownHostException
            {
50          List<String> udps = new ArrayList<>();
            System.out.println(ip);
52          for (int port=0;port<1000;port++) {
                String udp;
54              try (DatagramSocket socket = new DatagramSocket()) {
                    // 创建一个DatagramSocket对象
56
```

```
                InetAddress address = InetAddress.getByName(ip);
58              // 根据IP地址获取InetAddress对象

60              byte[] sendData = new byte[1];
                DatagramPacket sendPacket = new DatagramPacket(
                    sendData, sendData.length, address, port);
62              // 创建一个DatagramPacket对象，用于发送UDP数据包
                // DatagramPacket类表示UDP数据包，它包含了要发送
                    的数据、目标主机的IP地址和端口号等信息。通过
                    创建一个DatagramPacket对象，
64              // 我们可以设置要发送的数据、目标主机的地址和端
                    口号，然后将该数据包发送到目标主机
                socket.send(sendPacket);
66              // 发送UDP数据包

68              udp = "port " + port + ":open";
                // 如果没有发生异常，则表示端口开放
70          } catch (Exception e) {
                udp = "port " + port + ":close";
72              // 发生异常，则表示端口关闭
            }
74          udps.add(udp);
            // 将UDP端口扫描结果添加到结果列表中
76      }
        return udps;
78      // 返回UDP端口扫描的结果列表
    }
80  }

82  import javax.swing.*;
    import java.awt.*;
84  import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
86  import java.io.IOException;
    import java.net.UnknownHostException;
```

```
88 import java.util.List;

90 public class Dialog {
       public void show() {
92         Scan scan = new Scan(); // 创建扫描对象

94         // 创建窗口
           JFrame jf = new JFrame("端口扫描器");
96         jf.setSize(600, 700);
           jf.setLocation(350, 50);
98         JPanel jp = new JPanel();

100        // 创建标签和文本框用于输入IP地址
           JLabel jl = new JLabel("IP:");
102        jl.setFont(new Font(null, Font.PLAIN, 23));
           JTextField jTextField = new JTextField(20);
104        jTextField.setSize(300, 23);
           jTextField.setFont(new Font(null, Font.PLAIN, 18));
106
           // 创建按钮和文本区域
108        JButton jb = new JButton("udp扫描");
           JButton jbb = new JButton("tcp扫描");
110        JTextArea textArea1 = new JTextArea(12, 30);
           textArea1.setLineWrap(true);
112        textArea1.setFont(new Font(null, Font.PLAIN, 18));
           JScrollPane scrollPane1 = new JScrollPane(textArea1);
114        textArea1.append("UDP:\n");
           JTextArea textArea2 = new JTextArea(12, 30);
116        textArea2.setLineWrap(true);
           textArea2.setFont(new Font(null, Font.PLAIN, 18));
118        JScrollPane scrollPane2 = new JScrollPane(textArea2);
           textArea2.append("TCP:\n");
120
           // 设置文本区域的滚动窗格位置和大小
122        Dimension size1 = textArea1.getPreferredSize();
```

```
        scrollPane1.setBounds(120, 50, size1.width, size1.height
            );
124     Dimension size2 = textArea2.getPreferredSize();
        scrollPane2.setBounds(120, 50, size2.width, size2.height
            );
126
        // 将组件添加到面板
128     jp.add(jl);
        jp.add(jTextField);
130     jp.add(jb);
        jp.add(jbb);
132     jp.add(scrollPane1);
        jp.add(scrollPane2);
134
        // 设置窗口的内容面板和关闭操作
136     jf.setContentPane(jp);
        jf.setDefaultCloseOperation(WindowConstants.
            EXIT_ON_CLOSE);
138     jf.setVisible(true);

140     // 添加按钮的事件监听器
        jbb.addActionListener(new ActionListener() {
142         @Override
            public void actionPerformed(ActionEvent e) {
144             String ip = jTextField.getText();
                if (!ip.isEmpty()) {
146                 try {
                        List<String> portsT = scan.findTCP(ip);
                            // 进行TCP扫描
148                     for (String s : portsT) {
                            textArea2.append(s + "\n"); // 将扫
                                描结果添加到文本区域
150                     }
                        textArea2.append("结束");
152                     textArea2.append("\n");
```

```
                    } catch (IOException ex) {
                        ex.printStackTrace();
                    }
                } else {
                    textArea2.setText("请输入拟扫描的IP"); // IP地址为空时显示提示信息
                }
            }
        });

        jb.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String ip = jTextField.getText();
                if (!ip.isEmpty()) {
                    try {
                        List<String> portsU = scan.findUDP(ip); // 进行UDP扫描
                        for (String s : portsU) {
                            textArea1.append(s + "\n"); // 将扫描结果添加到文本区域
                        }
                        textArea1.append("结束");
                        textArea1.append("\n");
                    } catch (UnknownHostException ex) {
                        ex.printStackTrace();
                    }
                } else {
                    textArea1.setText("请输入拟扫描的IP"); // IP地址为空时显示提示信息
                }
            }
        });
    }
}
```

B3SourceCode