目录

[一、设计任务 1](#_Toc186894754)

[二、需求分析 2](#_Toc186894755)

[三、总体设计与理论 4](#_Toc186894756)

[原理分析： 4](#_Toc186894757)

[总体实现： 5](#_Toc186894758)

[四、具体实现 7](#_Toc186894759)

[五、设计总结 18](#_Toc186894760)

一、设计任务

网络通信是现代计算机技术的核心，ping 是网络管理和调试中常用的工具，用于测试主机与目标地址之间的网络连通性和网络延迟。为了更友好地使用 ping 功能，本项目设计了一个具有用户界面的 Ping 工具。设计的程序，使之能达到与系统自带的Ping命令相同的效果,并且有图形化的UI界面。

制定任务如下：

1.用户输入：允许用户输入目标地址（域名或 IP 地址）。

2.地址解析：将目标地址解析为 IP 地址，检测其有效性。

3.网络探测：通过 ICMP 协议模拟 ping 功能，测试网络连通性、延迟等。

4.结果展示：以图形化界面实时显示 ping 的结果，包括接收数据包、延迟、丢包率等。

5.默认网关检测：获取本地默认网关的 MAC 地址，用于构建以太网帧头。

二、需求分析

功能需求分析：

1.图形化用户界面：

a.使用 Swing 构建界面。

b.提供输入框、按钮和文本区域。

c输入框接收目标地址，按钮触发 ping 测试，文本区域实时展示结果。

2.地址解析：

a解析用户输入的目标地址（支持域名或 IP 地址）。

b验证解析结果并显示解析成功或失败的信息。

3.ICMP Ping 测试：

a使用 Jpcap 库构建 ICMP 数据包并发送。

b捕获目标地址的响应包，计算网络延迟和丢包率。

c提供实时反馈，模拟命令行中的 ping 输出格式。

4.默认网关 MAC 地址获取：

a通过抓包获取默认网关的 MAC 地址。

b在以太网层构建数据包时，设置目的地址为默认网关的 MAC 地址。

5.结果统计：

a显示数据包发送数量、接收数量、丢包数量和丢包率。

b统计和展示延迟的最小值、最大值和平均值。

技术功能需求分析：

4.1 编程语言与工具

Java：核心开发语言。

Swing：实现图形化界面。

Jpcap 库：支持网络包捕获与发送。

4.2 网络通信技术

ICMP 协议：实现 ping 功能的核心协议。

ARP 协议：获取默认网关的 MAC 地址。

IP 协议：处理 IPv4 数据包。

4.3 工具类 Util 的功能

Util 类提供了一些辅助方法，用于网络地址的转换与处理：

MAC 地址与字节数组的转换：

支持以太网层 MAC 地址的格式化显示与处理。

十六进制与字节数组的转换：

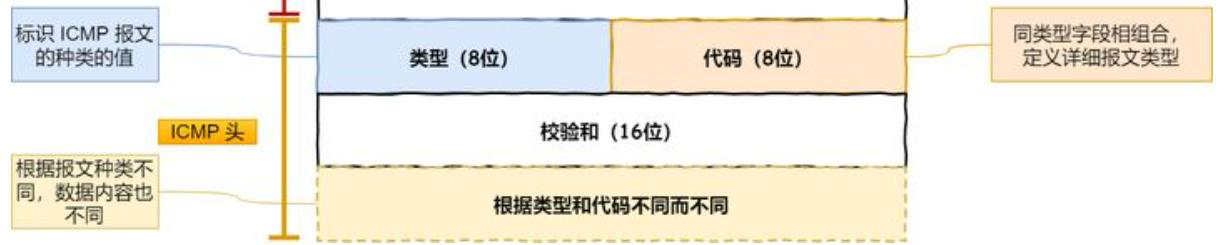
支持网络协议中常见的字节与十六进制表示的相互转换。

三、总体设计与理论

原理分析：

A.ICMP协议

ICMP协议是一个网络层协议。一个新搭建好的网络，往往需要先进行一个简单的测试，来验证网络是否畅通；但是IP协议并不提供可靠传输。如果丢包了，IP协议并不能通知传输层是否丢包以及丢包的原因。所以我们就需要一种协议来完成这样的功能–ICMP协议。ICMP报文包含在IP数据报中，IP报头在ICMP报文的最前面。一个ICMP报文包括IP报头（至少20字节）、ICMP报头（至少八字节）和ICMP报文（属于ICMP报文的数据部分）。当IP报头中的协议字段值为1时，就说明这是一个ICMP报文。



B. ping 命令的作用和原理：

简单来说，ping是用来探测本机与网络中另一主机之间是否可达的 命令，如果两台主机之间ping不通，则表明这两台主机不能建立起连接。ping 是定位网络通不通的一个重要手段。ping 命令是基于 ICMP 协议来工 作的，ICMP全称为 Internet 控制报文协议。ping 命令会发送一份ICMP 回显请求报文给目标主机，并等待目标主机返回ICMP回显应答。因为 ICMP协议会要求目标主机在收到消息之后，必须返回ICMP应答消息给源主机，如果源主机在一定时间内收到了目标主机的应答，则表明两台主机之间网络是可达的

C. ping 命令工作过程

假设有两个主机，主机A（192.1.1.1）和主机B（192.1.1.2）， 现在我们要监测主机A和主机B之间网络是否可达，那么我们在主机A上 输入命令：ping192.1.1.2。此时，ping 命令会在主机A上构建一个 ICMP 的请求数据包（数据包里的内容后面再详述），然后 ICMP协议会将这个数 据包以及目标IP（192.1.1.2）等信息一同交给IP层协议。IP层协议得到 这些信息后，将源地址（即本机IP）、目标地址（即目标IP：192.1.1.2）、 再加上一些其它的控制信息，构建成一个IP数据包。IP数据包构建完成后， 还不够，还需要加上MAC地址，因此，还需要通过ARP映射表找出目标 IP 所对应的MAC地址。当拿到了目标主机的MAC地址和本机MAC后，一并交给数据链路层，组装成一个数据帧，依据以太网的介质访问规则， 将它们传送出出去。当主机B收到这个数据帧之后，会首先检查它的目标 MAC地址是不是本机，如果是就接收下来处理，接收之后会检查这个数据 帧，将数据帧中的IP数据包取出来，交给本机的IP层协议，然后IP层协 议检查完之后，再将ICMP数据包取出来交给ICMP协议处理，当这一步 也处理完成之后，就会构建一个ICMP应答数据包，回发给主机A。在一定的时间内，如果主机A收到了应答包，则说明它与主机B之间网络可达， 如果没有收到，则说明网络不可达。除了监测是否可达以外，还可以利用 应答时间和发起时间之间的差值，计算出数据包的延迟耗时。

总体实现：

使用Java语言实现，并使用第三方库jpcap来实现整个程序的实现，实现与Windows系统自带的Ping功能的相同输出值。本实验的Java版本为17.0

需要引入jpcap包，在idea中实现。

总体流程为下述步骤：

1.获取网络设备列表

* 使用JpcapCaptor.getDeviceList()获取当前系统中可用的网络接口设备列表。这些设备通常是计算机上的网卡。

2.解析目标IP地址

* 程序首先检查命令行参数是否提供了目标IP地址或域名。如果没有提供，程序会提示用户输入目标IP地址或域名。
* 使用InetAddress.getByName()方法将输入的域名或IP地址解析为InetAddress对象。如果解析失败，程序会输出错误信息并退出。

3.选择网络接口

* 程序默认选择设备列表中的第5个网络接口（devices[4]）作为发送和接收ICMP数据包的网络接口。这个选择是硬编码的，可能需要根据实际情况调整。

4.获取源IP地址

* 程序遍历所选网络接口的地址列表，找到第一个IPv4地址作为源IP地址（srcIP）。

5.打开网络接口

* 使用JpcapCaptor.openDevice()方法打开选定的网络接口，准备捕获和发送数据包。

6.获取默认网关的MAC地址

* 程序通过向百度（www.baidu.com）发送HTTP请求，捕获返回的数据包，从中提取默认网关的MAC地址。这个步骤是为了确保ICMP数据包能够正确路由到目标地址。

7.设置ICMP过滤器

* 使用captor.setFilter()方法设置过滤器，只捕获来自目标IP地址的ICMP回复数据包。

8.发送ICMP Echo请求

* 程序构造ICMP Echo请求数据包（ICMPPacket），设置ICMP类型为ICMP\_ECHO，并填充数据字段。
* 使用JpcapSender.sendPacket()方法发送ICMP Echo请求数据包。

9.接收ICMP Echo回复

* 程序等待接收ICMP Echo回复数据包。如果收到回复，程序会计算往返时间（RTT），并输出相关信息，如数据包大小、响应时间和TTL（Time To Live）。
* 如果没有收到回复，程序会输出“请求超时”。

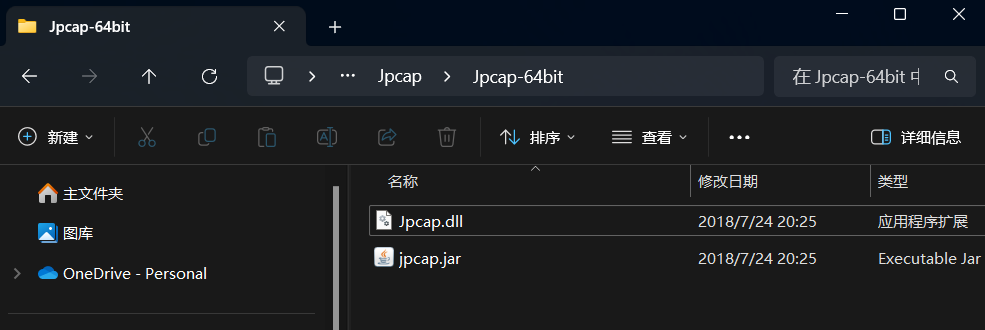
10.统计和输出结果

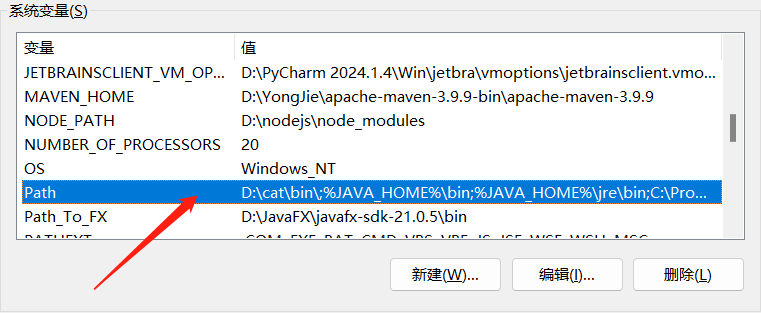
* 程序统计发送和接收的数据包数量，计算丢包率，并输出Ping的统计信息，包括最短、最长和平均往返时间。

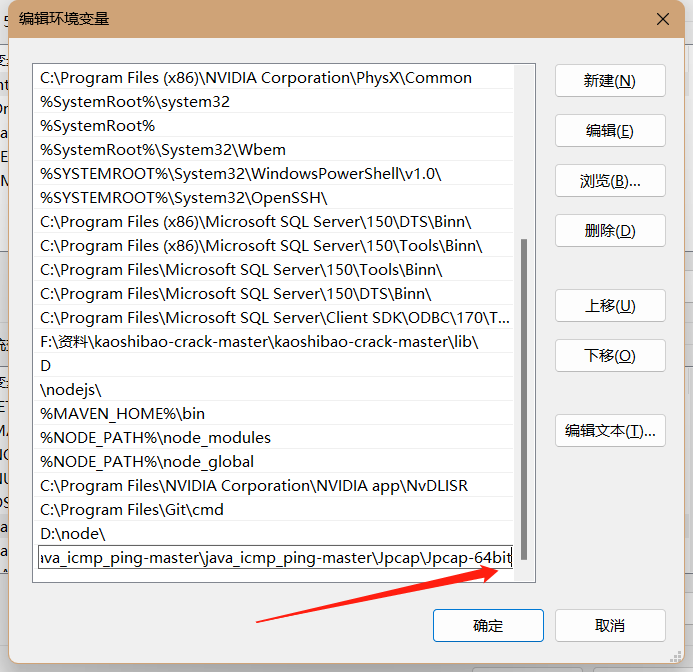
四、具体实现

在开始编程之前，需要先引入我们需要用到Jpcap包。配置如下：

下载好所要用到的Jpcap包，找到Jpcap.ddl所在的文件目录下，将其添加到系统环境变量中。







主程序PingGUI以及调用到的Ping方法如下：

package ping;

import java.io.IOException;

import java.net.Inet4Address;

import java.net.InetAddress;

import java.net.MalformedURLException;

import java.net.URL;

import java.net.UnknownHostException;

import java.util.Scanner;

import jpcap.JpcapCaptor;

import jpcap.JpcapSender;

import jpcap.NetworkInterface;

import jpcap.NetworkInterfaceAddress;

import jpcap.packet.EthernetPacket;

import jpcap.packet.ICMPPacket;

import jpcap.packet.IPPacket;

import jpcap.packet.Packet;

import util.Util;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class PingGUI extends JFrame {

    private JTextField inputField;

    private JTextArea resultArea;

    private JButton pingButton;

    public PingGUI() {

        // 设置窗口标题

        setTitle("Ping工具");

        // 设置窗口大小

        setSize(500, 400);

        // 设置关闭操作

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        // 窗口居中显示

        setLocationRelativeTo(null);

        // 初始化组件

        inputField = new JTextField(20);

        pingButton = new JButton("Ping");

        resultArea = new JTextArea();

        resultArea.setEditable(false); // 结果区域不可编辑

        // 布局

        JPanel inputPanel = new JPanel();

        inputPanel.add(new JLabel("目标地址："));

        inputPanel.add(inputField);

        inputPanel.add(pingButton);

        JScrollPane scrollPane = new JScrollPane(resultArea);

        setLayout(new BorderLayout());

        add(inputPanel, BorderLayout.NORTH);

        add(scrollPane, BorderLayout.CENTER);

        // 按钮事件

        pingButton.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                String input = inputField.getText().trim();

                if (input.isEmpty()) {

                    // 提示用户输入目标地址

                    JOptionPane.showMessageDialog(PingGUI.this, "请输入目标地址！", "错误", JOptionPane.ERROR\_MESSAGE);

                    return;

                }

                try {

                    // 解析输入的主机名或IP地址

                    InetAddress ip = InetAddress.getByName(input);

                    // 显示解析成功的IP地址

                    resultArea.append("解析成功，IP 地址：" + ip.getHostAddress() + "\n");

                    // 获取所有网络接口列表

                    NetworkInterface[] devices = JpcapCaptor.getDeviceList();

                    // 清空结果区域

                    resultArea.setText("");

                    // 执行Ping操作，默认使用第5个网络接口

                    ping(devices[4], ip, 4, resultArea);

                } catch (UnknownHostException ex) {

                    // 显示无法找到主机的错误信息

                    resultArea.append("无法找到主机：" + input + "\n");

                }

            }

        });

    }

    // 执行Ping操作的方法

    public static void ping(NetworkInterface device, InetAddress dstIP, int times, JTextArea resultArea) {

        InetAddress srcIP = null;

        // 获取设备的IPv4地址作为源IP

        for (NetworkInterfaceAddress addr : device.addresses) {

            if (addr.address instanceof Inet4Address) {

                srcIP = addr.address;

                break;

            }

        }

        int timeout = 1000;

        JpcapCaptor captor = null;

        try {

            // 打开设备进行数据包捕获

            captor = JpcapCaptor.openDevice(device, 2000, false, timeout);

        } catch (IOException e) {

            e.printStackTrace();

        }

        byte[] defaultGatewayMacAddress = null;

        InetAddress baiduIP = null;

        try {

            // 获取百度的IP地址

            baiduIP = InetAddress.getByName("www.baidu.com");

        } catch (UnknownHostException e1) {

            e1.printStackTrace();

        }

        // 获取默认网关的MAC地址

        while (true) {

            try {

                new URL("http://www.baidu.com").openStream().close();

            } catch (MalformedURLException e) {

                e.printStackTrace();

            } catch (IOException e) {

                e.printStackTrace();

            }

            Packet p = captor.getPacket();

            if (p == null) {

                continue;

            }

            IPPacket packet = (IPPacket) p;

            if (packet == null) {

                System.out.println("[调试信息]获取默认网关MAC地址失败，正在重试。。。");

            } else {

                if (packet.dst\_ip.getHostAddress().equals(baiduIP.getHostAddress())) {

                    defaultGatewayMacAddress = ((EthernetPacket) packet.datalink).dst\_mac;

                    break;

                }

            }

        }

        System.out.println("[调试信息]默认网关的MAC地址：" + Util.bytes2MacAddress(defaultGatewayMacAddress));

        System.out.println();

        System.out.println();

        try {

            // 设置捕获过滤器，只捕获来自目标IP的ICMP包

            captor.setFilter("icmp and src host " + dstIP.getHostAddress(), true);

        } catch (IOException e) {

        }

        byte[] data = "abcdefghijklmnopqrstuvwxyz123456".getBytes();

        JpcapSender sender = captor.getJpcapSenderInstance();

        short seq = 1;

        System.out.println(String.format("正在 Ping %s 具有 %d 字节的数据：", dstIP.getHostAddress(), data.length));

        int send = 0;

        int recv = 0;

        int timeSum = 0;

        int maxTime = 0;

        int minTime = 99999;

        String strip = "";

        for (int i = 0; i < times; i++) {

            ICMPPacket icmp = new ICMPPacket();

            icmp.type = ICMPPacket.ICMP\_ECHO;

            icmp.code = 0;

            icmp.seq = seq++;

            icmp.id = 1;

            icmp.data = data;

            int ttl = 128;

            icmp.setIPv4Parameter(0, false, false, false, 0, false, false, false, 0, 0, ttl, IPPacket.IPPROTO\_ICMP, srcIP, dstIP);

            EthernetPacket eth = new EthernetPacket();

            eth.frametype = EthernetPacket.ETHERTYPE\_IP;

            eth.src\_mac = device.mac\_address;

            eth.dst\_mac = defaultGatewayMacAddress;

            icmp.datalink = eth;

            long time = System.currentTimeMillis();

            icmp.sec = time / 1000;

            icmp.usec = time % 1000 \* 1000;

            sender.sendPacket(icmp);

            send++;

            ICMPPacket packet = (ICMPPacket) captor.getPacket();

            if (packet == null) {

                resultArea.append("请求超时。\n");

                System.out.println("请求超时。");

            } else {

                recv++;

                long t1 = icmp.sec \* 1000 + icmp.usec / 1000;

                long t2 = packet.sec \* 1000 + packet.usec / 1000;

                InetAddress ip = packet.src\_ip;

                int dataLength = packet.data.length;

                int hop\_limit = packet.hop\_limit;

                int t = (int) (t2 - t1);

                if (t < 0) {

                    t = 10;

                }

                String status = String.format("来自 %s 的回复：字节=%d 时间=%dms TTL=%d", ip.getHostAddress(), dataLength, t, hop\_limit);

                resultArea.append(status + "\n");

                System.out.println(status);

                timeSum += t;

                strip = ip.getHostAddress();

                if (t > maxTime) {

                    maxTime = t;

                }

                if (t < minTime) {

                    minTime = t;

                }

            }

        }

        String result1 = String.format("%s 的 Ping 统计信息：", strip);

        resultArea.append(result1 + "\n");

        System.out.println(result1);

        int lost = send - recv;

        int l = (int) (((double) lost / send) \* 100);

        String result2 = String.format("　　数据包：已发送=%d，已接收=%d，丢失=%d（%d%% 丢失）", send, recv, lost, l);

        resultArea.append(result2 + "\n");

        System.out.println(result2);

        if (recv > 0) {

            resultArea.append("往返行程的估计时间(以毫秒为单位):\n");

            System.out.println("往返行程的估计时间(以毫秒为单位):");

            int avg = timeSum / recv;

            String result3 = String.format("　　最短 = %dms，最长 = %dms，平均 = %dms", minTime, maxTime, avg);

            resultArea.append(result3 + "\n");

            System.out.println(result3);

        }

        System.out.println();

        int a = ICMPPacket.ICMP\_TIMXCEED;

        int b = ICMPPacket.ICMP\_UNREACH;

        int c = ICMPPacket.ICMP\_ECHOREPLY;

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            @Override

            public void run() {

                new PingGUI().setVisible(true);

            }

        });

    }

}

在主程序中调用到的相关类，在ping中调用它以获取网关地址。

public class Util {

    /\*\*

     \* 将字节数组转换为标准的 MAC 地址字符串，格式为 "XX:XX:XX:XX:XX:XX"。

     \* 如果输入的字节数组为空或长度不是 6，则返回默认的 MAC 地址 "00:00:00:00:00:00"。

     \*

     \* @param bs 表示 MAC 地址的字节数组

     \* @return 转换后的 MAC 地址字符串

     \*/

    public static String bytes2MacAddress(byte[] bs) {

        // 检查输入字节数组是否有效

        if (bs == null || bs.length != 6) {

            return "00:00:00:00:00:00";

        }

        StringBuilder macAddress = new StringBuilder();

        // 遍历字节数组，将每个字节转换为十六进制表示

        for (int i = 0; i < bs.length; i++) {

            // 将字节转换为十六进制字符串

            String t = Integer.toHexString(bs[i] & 0xff);

            // 如果十六进制字符串只有一位，则在前面补 0

            if (t.length() == 1) {

                macAddress.append("0");

            }

            macAddress.append(t);

            // 每个字节之间用冒号 (:) 分隔，最后一个字节后不加冒号

            if (i != bs.length - 1) {

                macAddress.append(":");

            }

        }

        return macAddress.toString();

    }

    /\*\*

     \* 将标准格式的 MAC 地址字符串（如 "XX:XX:XX:XX:XX:XX" 或 "XX-XX-XX-XX-XX-XX"）

     \* 转换为字节数组。如果输入的 MAC 地址无效，则返回 null。

     \*

     \* @param macAddress MAC 地址字符串

     \* @return 表示 MAC 地址的字节数组，如果无效则返回 null

     \*/

    public static byte[] macAddress2bytes(String macAddress) {

        // 移除 MAC 地址中的连字符 (-) 和冒号 (:)

        String mac = macAddress.replace("-", "").replace(":", "");

        // 检查 MAC 地址长度是否为 12 个字符

        if (mac.length() != 12) {

            return null;

        }

        // 将十六进制字符串转换为字节数组

        return hex2Bytes(mac);

    }

    /\*\*

     \* 将十六进制字符串转换为字节数组。

     \* 每两个十六进制字符转换为一个字节。

     \*

     \* @param src 十六进制字符串

     \* @return 转换后的字节数组

     \*/

    public static byte[] hex2Bytes(String src) {

        // 创建结果字节数组，长度为输入字符串的一半

        byte[] res = new byte[src.length() / 2];

        char[] chs = src.toCharArray(); // 将字符串转换为字符数组

        int[] b = new int[2]; // 临时存储两个十六进制字符的数值

        // 遍历字符数组，每次处理两个字符

        for (int i = 0, c = 0; i < chs.length; i += 2, c++) {

            for (int j = 0; j < 2; j++) {

                // 将每个字符转换为十六进制对应的数值

                if (chs[i + j] >= '0' && chs[i + j] <= '9') {

                    b[j] = (chs[i + j] - '0'); // 数字 0-9

                } else if (chs[i + j] >= 'A' && chs[i + j] <= 'F') {

                    b[j] = (chs[i + j] - 'A' + 10); // 大写字母 A-F

                } else if (chs[i + j] >= 'a' && chs[i + j] <= 'f') {

                    b[j] = (chs[i + j] - 'a' + 10); // 小写字母 a-f

                }

            }

            // 将两个十六进制数值合并为一个字节

            b[0] = (b[0] & 0x0f) << 4; // 左移 4 位，作为高位

            b[1] = (b[1] & 0x0f); // 保留低 4 位

            res[c] = (byte) (b[0] | b[1]); // 合并高位和低位

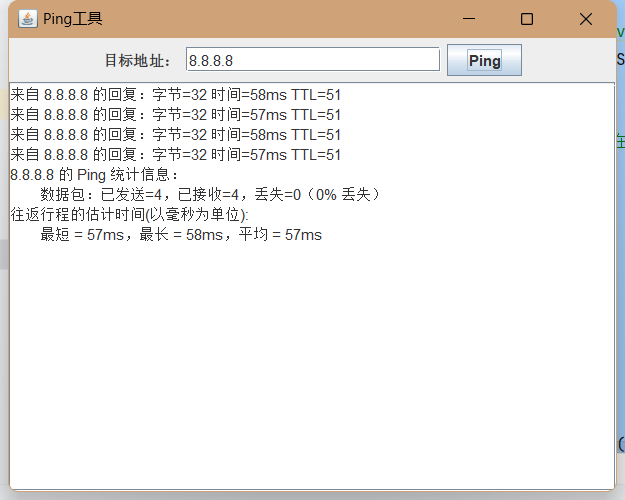
        }

        return res;

    }

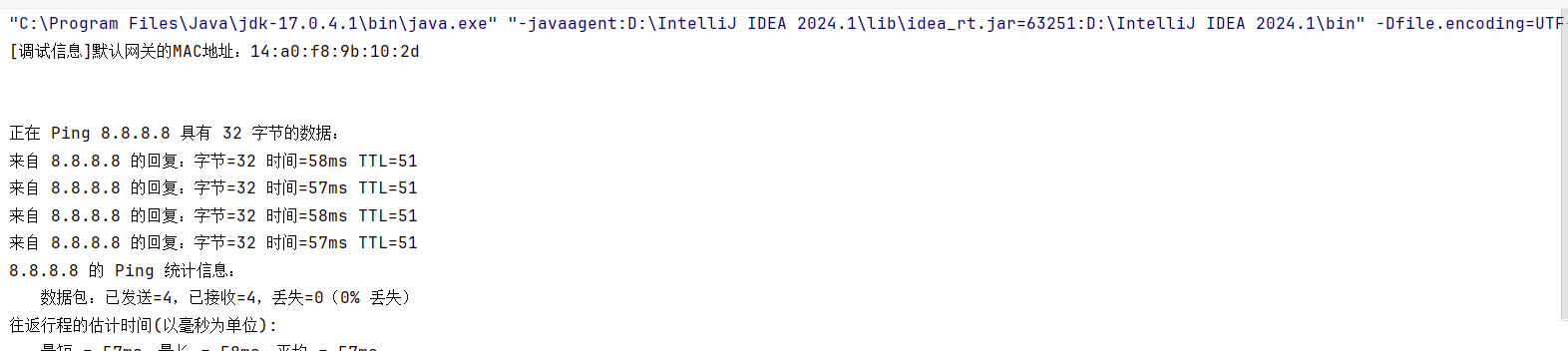
}

效果展示：

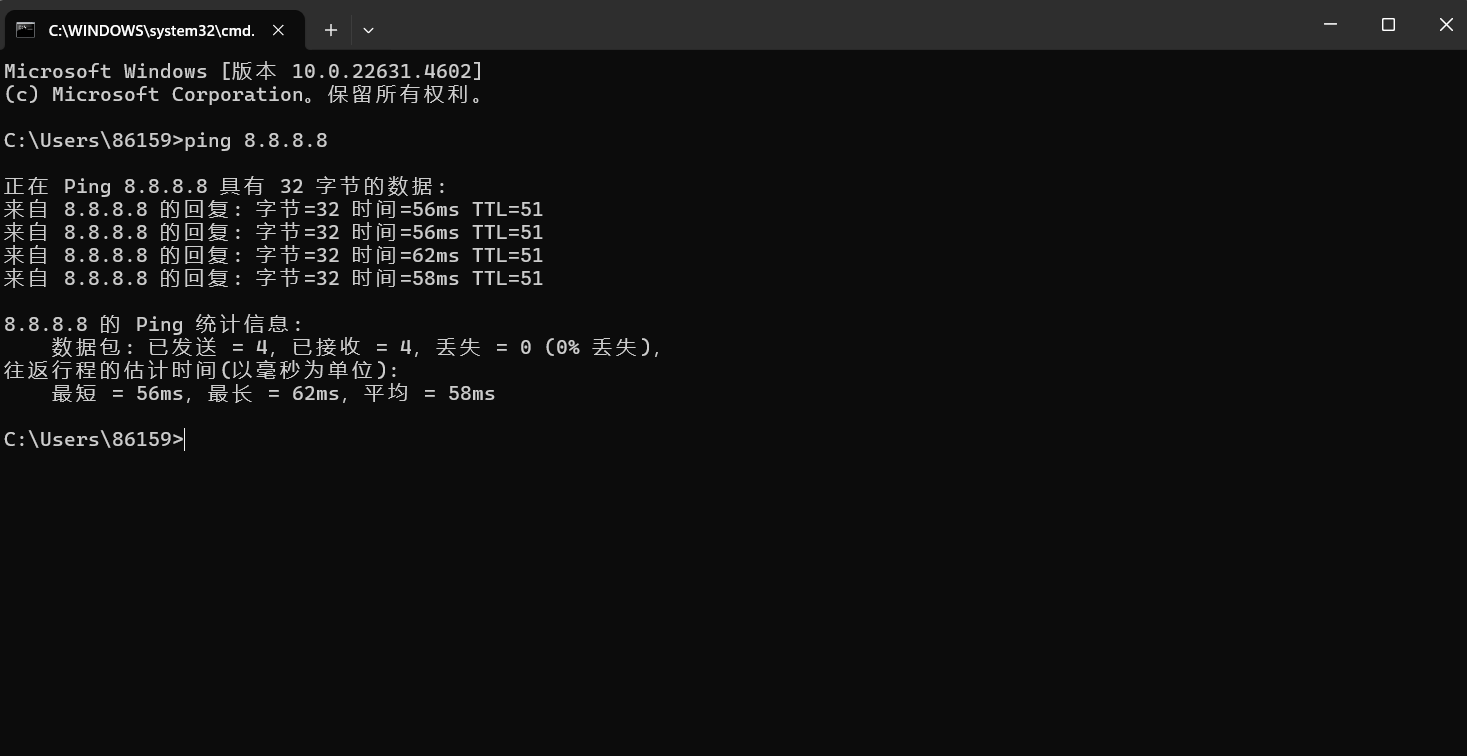


能够将Ping方法中获取的输出值在UI界面上呈现，更加方便。

同时在后台终端同样也可以输出结果，并且如果有异常，也可以在终端界面看到运行异常问题：



使用操作系统自带的Ping命令效果如下：



可以发现是效果相同的。

五、设计总结

通过使用java语言以及jpcap库和swing包来实现了一个能够平替操作系统ping命令的带有UI界面的程序。核心功能包括用户地址输入解析、ICMP 数据包发送与接收、默认网关 MAC 地址获取以及网络连接的延迟和丢包统计。通过 Jpcap 库的强大功能实现了网络包捕获与发送，同时使用 Swing 提供简洁明了的用户界面。

需要注意的是，在我们主程序中使用Ping方法进行调用时，我们要注意选择的网卡号，如果选择网卡不正确，是ping不通的，会导致Ping方法中的抓取包部分陷入死循环，跳不出来。

这里其实有些缺陷，但是通过调试最终找到了一个能用的网卡号，于是就选择了4号网卡进行操作。为了更方便调试，可以对Ping方法中抓包的循环体添加相关的条件。

对于核心方法作一下分析与总结

1. 地址解析与验证

InetAddress.getByName(input)

功能：将用户输入的目标地址（域名或 IP）解析为 InetAddress 对象，便于后续网络包的发送。

2. 默认网关 MAC 地址获取

captor.setFilter("icmp and src host " + baiduIP.getHostAddress(), true);

Packet p = captor.getPacket();

defaultGatewayMacAddress = ((EthernetPacket) p.datalink).dst\_mac;

功能：通过抓取 ICMP 数据包，从以太网层提取默认网关的 MAC 地址，用于构造网络包的帧头。

3.ICMP 数据包构造与发送

ICMPPacket icmp = new ICMPPacket();

icmp.type = ICMPPacket.ICMP\_ECHO;

icmp.seq = seq++;

icmp.id = 1;

icmp.data = data;

icmp.setIPv4Parameter(...); // 设置 IPv4 参数

EthernetPacket eth = new EthernetPacket();

eth.src\_mac = device.mac\_address;

eth.dst\_mac = defaultGatewayMacAddress;

icmp.datalink = eth;

sender.sendPacket(icmp);

这里值得注意的是：数据包的大小和格式必须符合标准，否则可能被目标主机丢弃。设置的 TTL 值决定数据包的最大跳数，过小会导致包无法到达目标。

4.ICMP 响应包接收与延迟计算

long t1 = icmp.sec \* 1000 + icmp.usec / 1000;

long t2 = packet.sec \* 1000 + packet.usec / 1000;

int t = (int) (t2 - t1);

功能：计算 ICMP 响应时间（网络延迟）。

最后，通过结合ICMP、ARP、IP等相关知识的结合，实现本程序，使得我对相关网络协议的了解更加深刻，更加清楚他们的作用以及如何将理论与实际相结合，最终实现代码。