

# 计网大作业：端口扫描

姓名：张文康

学号：522021910121

日期：2024.12

## 一、项目背景

- 端口**：端口是设备与外界通讯交流的出口，可分为物理端口和虚拟端口。在计算机网络中，端口号用于标识不同的网络服务或应用程序。
- 端口扫描的重要性**：端口扫描是网络安全评估和网络管理中的重要技术，通过扫描目标主机开放的端口，可以了解目标主机上运行的服务，进而发现潜在的安全漏洞或进行网络服务的管理与优化。
- 常见端口扫描技术原理**
  - ICMP扫描**：利用ICMP Echo（即ICMP Request数据包，ping命令）来判断主机是否开机。原理是向目标主机发送ICMP Echo Request数据包，如果目标主机在线并正常响应，会返回ICMP Echo Reply数据包，从而确定主机开机。这种扫描方式简单直接，但容易被防火墙等安全设备拦截，且无法获取目标主机上运行的具体服务信息。
  - TCP端口扫描**
    - TCP connect扫描**：这是最基本的TCP扫描方式。它尝试与目标主机的指定端口建立TCP连接，如果端口处于侦听状态，connect()函数成功返回0；若端口关闭，则返回SOCKET - ERROR。这种方式扫描结果准确，但容易被目标主机记录连接信息，因为它会完整地建立TCP连接，可能会触发目标主机的安全防护机制。
    - TCP SYN扫描**：也称为“半开放”扫描。它向目标端口发送一个SYN数据包，如果目标端口处于侦听状态，会返回SYN/ACK数据包；若端口关闭，则返回RST数据包。这种扫描方式不会建立完整的TCP连接，相对隐蔽，不易被目标主机记录，但需要具备较高的权限才能在某些操作系统上发送原始的TCP SYN数据包。
    - TCP FIN扫描**：若对应一个连接，返回ACK；若端口打开且没有连接，直接丢弃；若端口关闭，返回RST。这种扫描方式利用了TCP协议的特性，通过发送FIN数据包来判断端口状态，更加隐蔽，但在一些情况下可能不太准确，例如目标主机的操作系统对FIN数据包的处理方式可能影响扫描结果。
  - UDP扫描**：通过向目标UDP端口发送UDP数据包来检测端口状态。如果目标端口处于开放状态且有应用程序在监听该端口，通常会返回一个ICMP端口不可达消息（当发送的UDP数据包格式不正确或目标端口没有应用程序处理时）；如果目标端口关闭，大多数情况下不会返回任何响应，但也可能返回ICMP端口不可达消息（取决于目标系统和防火墙的配置）。

## 二、项目实现

- 开发环境**
  - 操作系统：Windows 11
  - 编程语言：Python 3.11
  - 主要依赖库：tkinter, socket, scapy, libpcap
- 系统架构**
  - UI层：使用tkinter实现图形界面
  - 业务层：实现各种扫描方法
  - 工具层：提供IP和端口验证等通用功能
- 核心功能实现**
  - 图形界面**：使用tkinter构建直观的用户界面，包含IP输入、端口输入、扫描按钮和结果显示区
  - ICMP扫描**：使用Raw Socket发送ICMP Echo请求
  - TCP扫描**：实现Connect、SYN、FIN三种扫描方式
  - UDP扫描**：通过发送UDP数据包探测端口状态
  - 输入验证**：确保IP地址和端口号格式正确
  - 日志保存**：保存当前时刻结果显示区的内容

## 三、关键代码分析

### 1. IP和端口格式验证

- 检测IP地址是否符合IPv4的格式规范

```
def validate_ip(ip):
    if not ip:
        return False
    try:
        # 检查IP格式
        parts = ip.split('.')
        if len(parts) != 4:
            return False
        # 检查每个部分是否为0-255的整数
        return all(0 <= int(part) <= 255 for part in parts)
    except (AttributeError, TypeError, ValueError):
        return False
```

- 检测端口是否在0~65535范围内

```
def validate_port(port):
    try:
        port_num = int(port)
        return 0 <= port_num <= 65535
    except ValueError:
        return False
```

### 2. ICMP扫描

- 直接利用 ping 指令模拟ICMP扫描

```
class ICMPScanner:
    def __init__(self):
        pass

    def ping_host(self, host):
        # 使用 ping 命令来判断主机是否在线
        response = os.system(f"ping -n 1 {host}")
        return response == 0
```

### 3. TCP扫描

- TCP Connect: 直接用socket库实现连接
- TCP SYN 和 TCP FIN: 用scapy 进行侦听, 获取响应, 根据响应的结果判断具体的扫描结果
- 三种方法的比较
  - TCP Connect: 需要建立TCP连接, 比较准确但耗时、隐蔽性低
  - TCP SYN: “半开放”扫描, 准确性、速度和隐蔽性居中
  - TCP FIN: 不建立连接, 隐蔽性强、速度快, 但可靠性、准确性低
- 代码实现

```

import socket
from scapy.all import IP, TCP, sr1

class TCPScanner:
    def __init__(self, target=None):
        self.target = target

    def tcp_connect_scan(self, host, port):
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(1)
            result = sock.connect_ex((host, port))
            sock.close()
            return result == 0
        except Exception as e:
            print(f"Error: {e}")
            return False

    def tcp_syn_scan(self, host, port):
        try:
            pkt = IP(dst=host)/TCP(dport=port, flags="S")
            resp = sr1(pkt, timeout=1, verbose=0)
            if resp is None:
                return False
            if resp.haslayer(TCP):
                # 检查TCP响应中的SYN/ACK标志
                if resp.getlayer(TCP).flags == 0x12:
                    return True
                # 检查TCP响应中的RST标志
                elif resp.getlayer(TCP).flags == 0x14:
                    return False
            return False
        except Exception as e:
            print(f"Error: {e}")
            return False

    def tcp_fin_scan(self, host, port):
        try:
            pkt = IP(dst=host)/TCP(dport=port, flags="F")
            resp = sr1(pkt, timeout=1, verbose=0)
            if resp is None:
                return False
            if resp.haslayer(TCP):
                # 检查TCP响应中的ACK标志
                if resp.getlayer(TCP).flags == 0x10:
                    return True
                # 检查TCP响应中的RST标志
                elif resp.getlayer(TCP).flags == 0x14:
                    return False
            return False
        except Exception as e:
            print(f"Error: {e}")
            return False

    def scan_ports(self, ports, scan_type):
        results = {}
        for port in ports:

```

```

        if scan_type == 'connect':
            results[port] = self.tcp_connect_scan(self.target, port)
        elif scan_type == 'syn':
            results[port] = self.tcp_syn_scan(self.target, port)
        elif scan_type == 'fin':
            results[port] = self.tcp_fin_scan(self.target, port)
    return results

```

## 4. UDP扫描

- 借助scapy实现，根据响应结果具体判断

```

class UDPScanner:
    def __init__(self):
        pass
    def udp_scan(self, host, port):
        try:
            # 发送 UDP 包并等待响应
            pkt = IP(dst=host)/UDP(dport=port)
            resp = sr1(pkt, timeout=2, verbose=0)

            if resp is None:
                return True # 可能开放（无响应）
            elif resp.haslayer(ICMP):
                if int(resp[ICMP].type) == 3 and int(resp[ICMP].code) == 3:
                    return False # 端口关闭
                elif int(resp[ICMP].type) == 3 and int(resp[ICMP].code) in [1, 2, 9, 10, 13]:
                    return False # 被过滤
            else:
                return True # 端口开放

        except Exception as e:
            print(f"Error during UDP scan: {e}")
            return False

```

## 5. UI界面实现

- 完整代码参见 `src/ui.py`
- 组件
  - IP 和 Port 输入部分
  - 扫描按钮和清除按钮
  - 结果显示区域
  - 端口提示区
  - 日志保存按钮
  - 背景图片
- 扫描功能函数（以TCP Connect为例）

```

def tcp_connect_scan(self):
    try:
        host = self.host_entry.get()
        port = self.port_entry.get()

        if not validate_ip(host):
            self.show_error("Invalid IP address format")
            return
        if not validate_port(port):
            self.show_error("Invalid port number (must be between 0-65535)")
            return

        result = self.tcp_scanner.tcp_connect_scan(host, int(port))
        status = 'Open' if result else 'Closed'
        self.insert_colored_result(f"TCP Connect Scan result for {host} on port {port}: {status}\n", status)
    except Exception as e:
        self.show_error(str(e))

```

- 日志保存功能

```

def save_log(self):
    """保存日志到文件"""
    # 创建output目录(如果不存在)
    if not os.path.exists('./output'):
        os.makedirs('./output')

    # 生成文件名 (使用时间戳)
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    default_filename = f"scan_log_{timestamp}.txt"

    # 获取完整的日志内容
    log_content = self.result_text.get(1.0, tk.END)

    # 如果日志为空, 显示提示
    if not log_content.strip():
        messagebox.showwarning("Warning", "No log content to save!")
        return

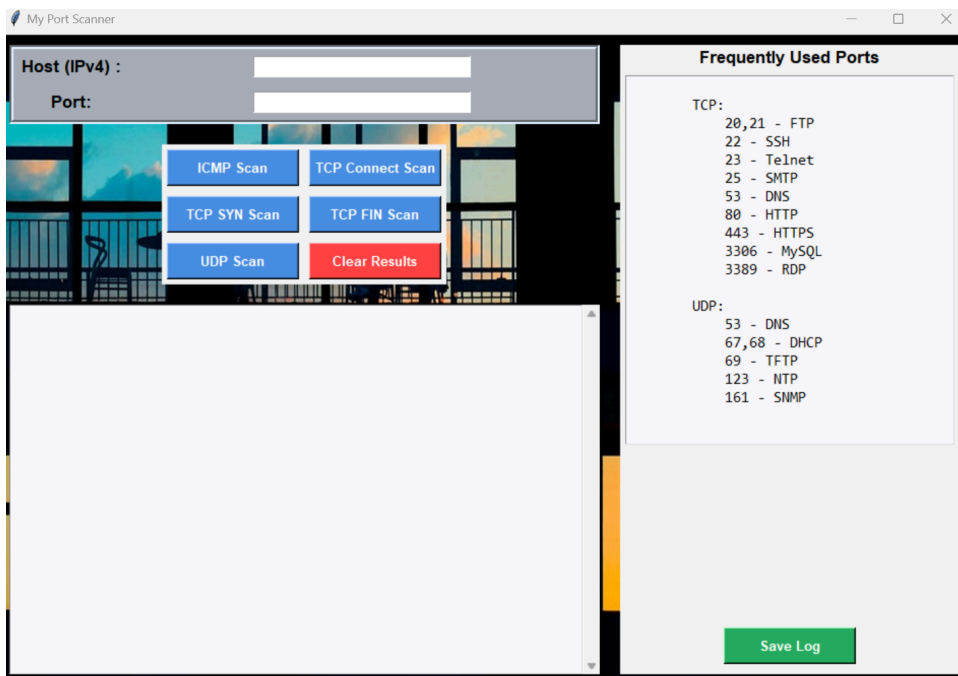
    try:
        # 保存文件
        filepath = os.path.join('./output', default_filename)
        with open(filepath, 'w', encoding='utf-8') as f:
            f.write(log_content)
        messagebox.showinfo("Success", f"Log saved successfully to:\n{filepath}")
    except Exception as e:
        messagebox.showerror("Error", f"Failed to save log: {str(e)}")

```

- 其他功能

- 不同结果用不同颜色表示
- 每条记录添加时间戳
- 显示窗口大小改变后, 组件和背景图片的自适应调节

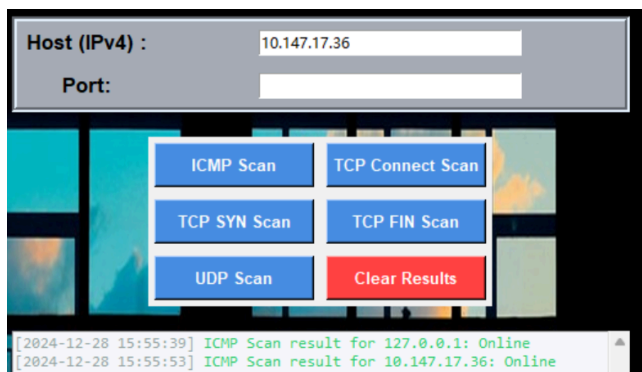
- UI完整界面



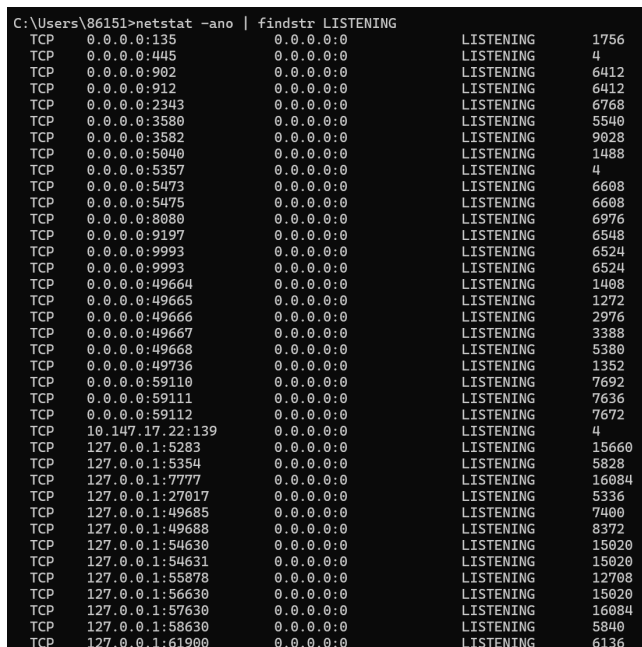
## 四、测试结果

### 1. 本地与远程测试

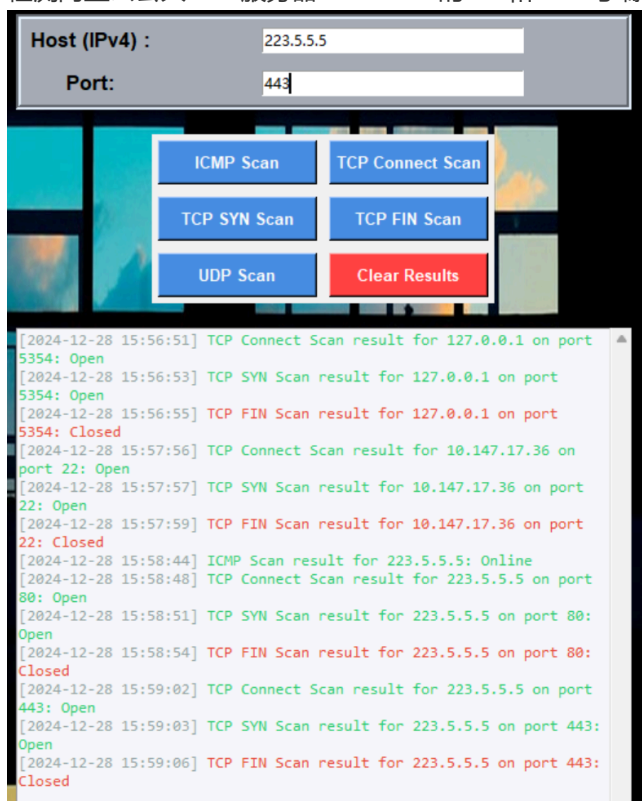
- ICMP扫描：
  - 成功检测到本机127.0.0.1 在线
  - 成功检测到远程服务器 10.147.17.36 在线



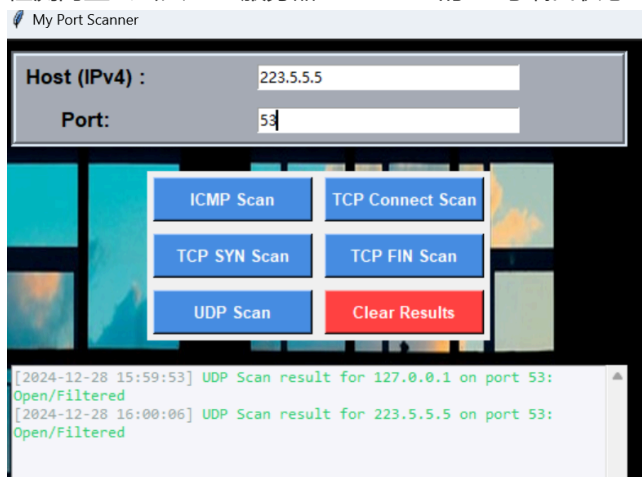
- TCP扫描：TCP Connect 和 TCP SYN 可以正确识别端口状态，TCP FIN 不能
  - 检测本机相关端口 5354



- 检测实验室远程服务器 10.147.17.36 的 22号端口 (ssh)
- 检测阿里云公共DNS服务器 223.5.5.5 的 80 和 443 号端口 (http和https)

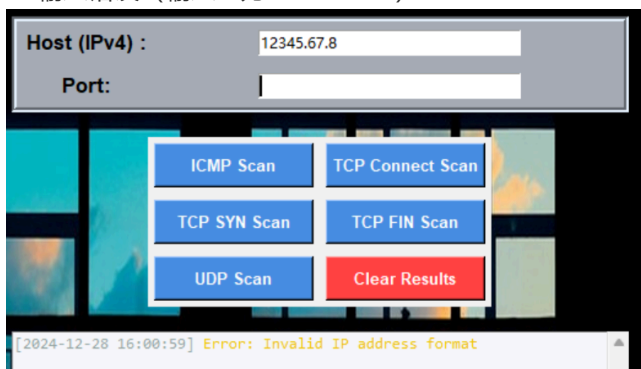


- UDP扫描:
  - 检测本机 53号端口状态 (DNS)
  - 检测阿里云公共DNS服务器 223.5.5.5 的 53号端口状态 (DNS)

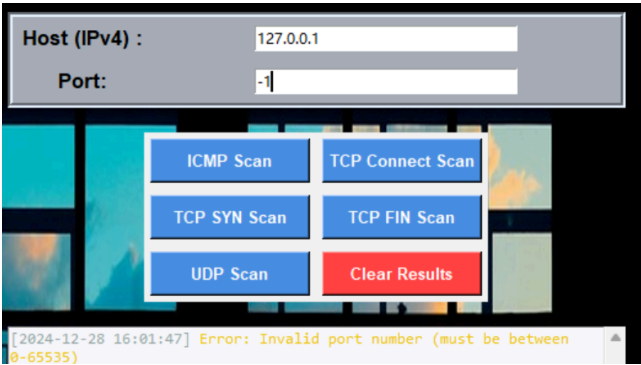


## 2. 异常输出测试

- IP 输入错误 (输入IP为12345.67.8)

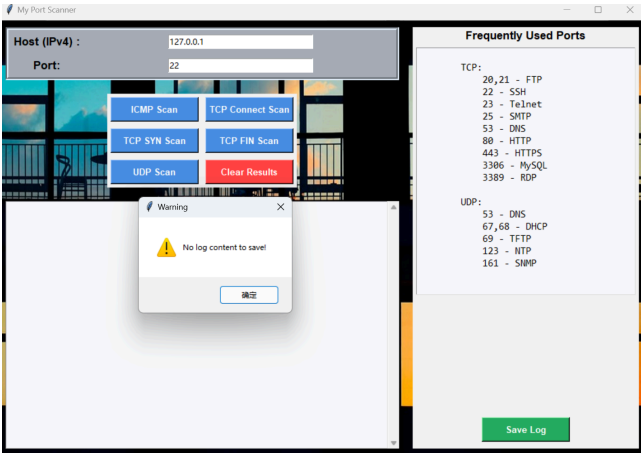


- 端口输入错误 (输入端口为-1)

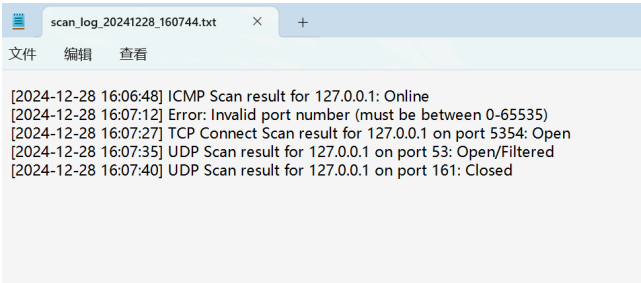
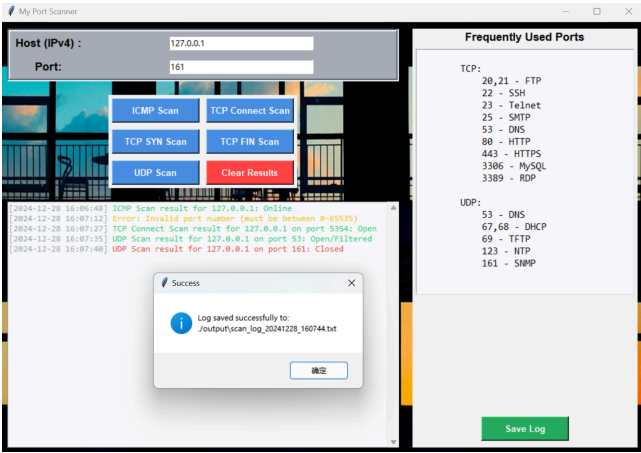


### 3. 日志保存功能测试

- 无内容无法保存



- 正常保存



## 五、项目特点

1. 用户友好
  - 直观的图形界面
  - 实时显示扫描结果
  - 内置常用端口提示
2. 功能完整



- 支持多种扫描方式
  - 提供输入验证
  - 结果可视化展示
3. 安全性考虑
- 限制扫描频率
  - 提供错误处理
  - 日志记录功能


## 六、所遇问题与解决

1. UI界面设计遇到困难，之前没有接触过python的UI库
  - 通过使用copilot等大模型，完成UI的设计实现，通过研究相关代码学习设计思路的方法
2. TCP FIN检测结果有些异常
  - 回过头研读相关函数，发现逻辑问题，查阅相关资料后进行改正

## 七、总结与展望

1. 项目总结与体会
  - 通过本次项目，我掌握网络编程的核心概念，学习了相关库的使用，成功实现基本端口扫描功能和合理的UI界面，提高了项目能力
  - 通过大模型工具的使用，提高编程效率，但不能完全依赖大模型，代码中的细节要做检查
2. 后续改进
  - 添加多线程支持提高扫描效率
  - 增加更多扫描技术
  - 优化用户界面


# 八、附录：大模型使用情况示例



Mr-Zwkid

我目前在上计算机网络课程，课程大作业是实现一个端口扫描功能，要求如下：1、基本功能：（1）ICMP扫描 使用ICMP Echo，即ICMP Request数据包（ping命令），来判断主机是否开机。（2）TCP端口扫描 使用TCP connect、TCP SYN和TCP FIN分别实现对某台主机的端口扫描，并分析这三种扫描方式的特点。•TCP connect扫描：最基本的TCP扫描。若端口处于侦听状态，connect()成功，返回0；若关闭，返回SOCKET-ERROR。•TCP SYN扫描：“半开放”扫描。若端口处于侦听状态，返回SYN/ACK；若关闭，返回RST。•TCP FIN扫描：若对应一个连接，返回ACK；若端口打开，且没连接，直接丢弃；若端口关闭，返回RST。（以SSH端口扫描、FTP端口扫描为例）2.自行扩展端口扫描器的功能，如使用其它协议（如UDP），或参考现有端口扫描工具的功能。3.使用python语言，Windows平台，可借助WinPcap、SharpPcap、LibPcap、Libnet等类库；4.能扫描到特定网络可能开启的应用服务；5.需要实现友好的用户界面；基于上述要求，完成port\_scan.py的编写

port\_scan.py:1

 Workspace

> 已使用 1 个引用

Sure, here's a proposed directory structure for a Python port scanning project:

port-scanner

src

icmp\_scan.py

tcp\_scan.py





ui.py

utils.py

requirements.txt

README.md

创建工作区...

Mr-Zwkid

下面请具体实现相关文件

icmp\_scan.py:1-8

GitHub Copilot

> 已使用 1 个引用

好的, 首先我们来完善 `icmp_scan.py` 文件中的 ICMP 扫描功能。

```
import os

class ICMPScanner:
    def __init__(self):
        pass

    def ping_host(self, host):
        # 使用 ping 命令来判断主机是否在线
        response = os.system(f"ping -n 1 {host}")
        return response == 0
```

接下来, 我们实现 `tcp_scan.py` 文件中的 TCP 端口扫描功能。

```
import socket

class TCPScanner:
    def __init__(self):
        pass

    def tcp_connect_scan(self, host, port):
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(1)
            result = sock.connect_ex((host, port))
            sock.close()
            return result == 0
        except Exception as e:
            print(f"Error: {e}")
            return False
```

Mr-Zwkid

能否在UI界面上添加一些提示信息, 如常用的熟知端口号及对应的是TCP还是UDP

ui.py

ui.py:9-54

GitHub Copilot

Enable Claude 3.5 Sonnet (Preview) for all clients

Enable access to the latest Claude 3.5 Sonnet model from Anthropic. [Learn more about how GitHub Copilot serves Claude 3.5 Sonnet.](#)

👍 🗨 📄

Mr-Zwkid

已选择“Enable”

GitHub Copilot

我将按以下步骤修改UI界面:

1. 添加一个新的Frame来显示常用端口信息
2. 使用Text widget显示端口信息, 设为只读
3. 添加一些常见端口号的说明

以下是修改后的代码:

```
import tkinter as tk
from tkinter import messagebox, scrolledtext
from icmp_scan import ICMPScanner
from tcp_scan import TCPScanner

class PortScannerUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Port Scanner")

        # ...existing code until result_frame...

        # 添加常用端口信息框架
        port_info_frame = tk.Frame(root)
```



Mr-Zwkid

增加一下ui界面的清晰度，设置背景图片？

ui.py:36-65



GitHub Copilot

> 已使用 1 个引用

计划步骤:


1. 为主窗口添加背景图片
2. 调整各个Frame的背景色和透明度
3. 美化字体和按钮样式
4. 添加一些视觉分隔

```
import tkinter as tk
from tkinter import messagebox, scrolledtext
from PIL import Image, ImageTk # 需要安装 pillow 库

class PortScannerUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Port Scanner")


        # 设置背景图片
        try:
            bg_image = Image.open("background.png") # 需要准备一张b
            bg_photo = ImageTk.PhotoImage(bg_image)
            bg_label = tk.Label(root, image=bg_photo)
            bg_label.image = bg_photo # 保持引用
            bg_label.place(x=0, y=0, relwidth=1, relheight=1)
        except:
            self.root.configure(bg='#f0f0f0') # 如果没有图片则使用纯

        # 美化Frame样式
        style = {
            'bg': '#ffffff',
            'relief': 'groove',
            'bd': 2
        }
```

 Mr-Zwkid

怎么使得背景图片适应窗口大小，并且各个组件适配窗口大小（窗口大小可改变）

ui.py:59-87

 GitHub Copilot

> 已使用 1 个引用

实现步骤:

1. 配置窗口和网格权重
2. 创建背景图片自适应调整函数
3. 绑定窗口大小改变事件
4. 修改组件布局以支持自适应

```
def __init__(self, root):
    self.root = root
    self.root.title("Port Scanner")

    # 配置窗口最小尺寸
    self.root.minsize(800, 600)

    # 配置主窗口网格权重
    self.root.grid_rowconfigure(2, weight=1)
    self.root.grid_columnconfigure(0, weight=3)
    self.root.grid_columnconfigure(1, weight=1)

    # 设置背景图片
    try:
        self.original_bg = Image.open("./assets/background.png")
        self.bg_photo = None
        self.update_background()
        self.bg_label = tk.Label(root)
        self.bg_label.place(x=0, y=0, relwidth=1, relheight=1)

        # 绑定窗口大小改变事件
        self.root.bind('<Configure>', self.on_resize)
    except:
```

询问 Copilot

project\_report.md Current file

@📎🔊

Claude 3.5 Sonnet (Preview) >