

Simple DNS relay

- 一、引言
- 二、目的
- 三、环境
- ▼ 四、代码结构与功能
 - 0. 代码层进次序
 - 1. 配置解析函数 `config(path)`
 - 2. 问题部分解析与打包类 `query_part`
 - 3. DNS 报文解析与生成类 `message`
 - 4. DNS 中继服务器类 `relay_server`
 - 5. 主程序入口
- ▼ 五、报文处理流程
 - 1. 接收报文
 - 2. 创建线程处理报文
 - 3. 解析报文
 - 4. 处理查询报文
 - 5. 处理响应报文
 - 6. 生成并发送响应报文
 - 7. 记录日志
- 六、报文生成细节
- ▼ 七、执行流程分析
 - 1. 接收到查询报文
 - 2. 接收到查询报文，域名未配置
 - 3. 接收到响应报文
- ▼ 八、实验结果
 - 1.准备工作
 - 2.命令行测试基本功能
 - 3.访问知乎测试屏蔽效果
- 未来改进方向：

一、引言

域名系统（DNS）作为互联网的重要基础设施，其通过转发和缓存 DNS 请求，能够提升解析效率，减轻上游 DNS 服务器的负载，并增强网络的安全性。此外，个人用户还可以配置自己的 DNS 中继以实现自己的个性化需求（例如为自己已有 ip 的网站配置个性域名、屏蔽不想看到的网站等）。本项目旨在通过实现一个简单可用的 DNS 中继服务器，深入理解 DNS 报文的结构及其处理流程。

二、目的

1. 理解 DNS 协议的基本结构和工作原理
2. 学习 DNS 服务器如何处理 UDP 与 IPv4
3. 掌握 DNS 报文的解析与生成
4. 实现 DNS 中继功能，包括本地解析与转发上游 DNS 服务器
5. 通过实验验证 DNS 中继服务器的功能

三、环境

- 编程语言: Python 3.10.10
- 操作系统: Windows11
- 依赖库: `socket`, `struct`, `time`, `threading`, `logging`
 - `logging` 库仅用于日志的打印, 与核心功能无关

四、代码结构与功能

0. 代码层进次序

运行示例请看第七节

- 启动
- 实例化 `relay_server` 类
 - 调用 `config` 读取配置
 - 配置套接字、公共 DNS 服务器地址, 绑定地址与端口
- 调用 `relay_server.run` 循环接受报文
 - 调用 `recvfrom` 接受数据与地址
 - 为每个任务创建线程
- 调用 `relay_server.process` 开启任务
 - 实例化 `message` 类并传入报文数据 `data`
 - `message.unpack` 解析并拆分报文
 - 通过实例化后传回的 `msg.qr` 判断是查询报文还是响应报文
 - 如果是查询报文, 优先查找是否存在本地配置
 - 如果本地已经对这个域名有配置, 则针对具体的配置内容生成回复 (对于配置为 `0.0.0.0` 会进行拦截, 其它 `ip` 直接返回)
 - 如果本地未配置该域名, 则将请求转发到上游 DNS 服务器并加入任务字典, 本地作为该请求的中继
 - 如果是响应报文, 则从任务字典列中弹出对应 `id` 的任务
 - 随后直接将上游 DNS 发来的响应内容发向原先发起请求的地址即可

对于有本地配置的域名, 上面提到我们要手动生成回复, 这里就需要调用 `message` 类中的 `r_pack` 方法。

由于我们的本地配置里包含 禁用域名 操作, 所以这里除了修改头部的回答数 `self.answers` 为 1 (表示我们做出了响应), 还需要修改 `self.flag` 以表示查询结果类型

```
# self.id: 事务 ID
# response_flags: ban type
# 0x8180: 表示这是一个标准查询响应, 无错误
# 0x8183: 表示域名不存在 (一般都是手动封禁)
# 1000 0001 1000 0011
# QR = 1 (响应)
# Opcode = 0000 (标准查询)
# AA = 0 (非权威回答)
# TC = 0 (未截断)
# RD = 1 (期望递归)
# RA = 0 (不支持递归)
# Z = 000 (保留字段)
# RCODE = 0011 (名称错误, 表示域名不存在)
# self.quests: 问题数
# response_answers: 回答数
# self.author: 授权记录数
# self.addition: 附加记录数
```

构造好头部后, 还需要补充几个字段

```

# 回答部分:如题
# 0xC00C: 指针, 指向问题部分的域名
# 1: 类型字段, 表示 A 记录
# 1: 类字段, 表示 IN 类
# 666: 生存时间 (TTL)
#     TTL 是 Time To Live 的缩写, 表示资源记录在缓存中的有效时间
#     设高点可以减少查询次数, 减小网络负载
# 4: 数据长度, 表示 IPv4 地址的长度
# ip: IPv4 地址

```

其中, 问题部分需要调用 `query_part` 类中的打包方法 `pack`, 其会将之前解析报文时拆出的 `name`、`type` 和 `class` 一起打包起来返回

对于 `name` 部分, 由于之前解析时得到的是字符域名, 所以这里要对那个字符域名按 . 拆分为多个部分, 每个部分先计算长度并将长度与该部分内容的字节编码一起打包到结果中 `struct.pack("B", len(part)) + part.encode()`, 接着循环处理完所有的部分即可。最后还需要补上终结符 `\x00`, 便达成了对问题部分的还原。

至此, 答复报文生成完毕, 随后将报文发回请求者即可。

整个 `DNS` 中继服务器的代码主要分为以下几个部分, 具体实现请看对应注释:

1. 配置解析函数 `config(path)`

这部分由助教提供, 未做改动

```

def config(path):
    """根据配置文件提取"""
    name2ip = {}
    with open(path, "r", encoding="utf-8") as file:
        for line in file:
            if line.strip(): # 判断行非空
                ip, domain = line.rstrip().split(" ", 1)
                name2ip[domain] = ip
    return name2ip

```

功能: 读取指定路径的配置文件, 将域名与对应的 IP 地址存储在字典 `name2ip` 中返回。配置文件格式为 IP 域名, 例如 0.0.0.0 www.test1.com。

2. 问题部分解析与打包类 `query_part`

假设 DNS 查询报文中的域名部分为 03 77 77 77 06 65 78 61 6D 70 6C 65 03 63 6F 6D 00

03 表示长度为 3, 标签为 www
 06 表示长度为 6, 标签为 example
 03 表示长度为 3, 标签为 com
 00 表示域名部分结束。

所以读取域名只需读取长度字节并往后读对应的长度, 循环直到读到终结符为止

```

class query_part:
    """一条问题记录, 解析+打包"""

    def __init__(self) -> None:
        self.name = ""
        self.idx = 0
        self.type = None
        self.classify = None

```

```

def unpack(self, data):
    """解析二进制查询报文中的问题节, data -> name, type, class"""
    # 公共变量复位
    self.name = ""
    self.idx = 0
    while True:
        length = data[self.idx] # 读取当前索引位置的长度字节
        if length == 0: # 如果长度为0, 说明域名解析完毕
            break
        self.idx += 1
        self.name += data[self.idx : self.idx + length].decode() + "."
        self.idx += length
    self.name = self.name.rstrip(".") # 去掉最后的点
    # struct.unpack()将字节数据解包为py的数据类型
    ## >HH表示两个无符号短整型数据, 大端字节序 (每个2字节, 共4字节)
    self.type, self.classify = struct.unpack(">HH", data[self.idx : self.idx + 4])
    self.idx += 4 # 上面读取了四个字节, 所以索引位置加4

```

```

def pack(self):
    """将问题节打包回二进制查询报文, name, type, class -> data"""
    parts = self.name.split(".")
    data = b"" # 初始化为空字节串
    # 打包域名
    for part in parts:
        # 字节表示的长度 + 编码后的字节数据
        # B 为一个字节, H 为两个字节。每个标签最长63字节
        data += struct.pack("B", len(part)) + part.encode()
    data += b"\x00" # 追加结束字节
    # 打包type和classify
    data += struct.pack(">HH", self.type, self.classify)
    return data

```

功能:

- `unpack`: 将接收到的二进制 `DNS` 查询报文中的问题部分解析为域名、类型和类。
- `pack`: 将解析后的域名、类型和类重新打包成二进制格式, 用于生成响应报文。

3. DNS 报文解析与生成类 `message`

```

class message:
    """一封DNS报文, 解析头部, 若是查询报文则进一步解析问题节"""

    def __init__(self, data) -> None:
        self.data = data
        self.unpack(data)

    def unpack(self, data):
        # 处理报文的头部
        self.id, self.flags, self.quests, self.answers, self.author, self.addition = (
            struct.unpack(">HHHHHH", data[0:12])
        )
        self.qr = data[2] >> 7
        # 这种方式提取出来的值将是0 (查询报文) 或1 (响应报文)
        if self.qr == 0: # 是查询报文
            self.query = query_part()
            self.query.unpack(data[12:]) # 生成问题节
        else:
            self.query = None

```

```

def r_pack(self, ip):
    """根据ip资源和当前查询报文内容生成回复报文，注意哪些头部字段要修改"""
    # 仿照上面的unpack方法，先生成data[0:12]，再调库生成data[12:]

    response_flags = 0x8183 if ip == "0.0.0.0" else 0x8180
    response_answers = 1
    response = struct.pack(
        ">HHHHH",
        self.id,
        response_flags,
        self.quests,
        response_answers,
        self.author,
        self.addition,
    )
    response += self.query.pack() # 将问题部分打包并追加到响应中
    response += struct.pack(">HHHLH", 0xC00C, 1, 1, 666, 4)
    ip_parts = [int(part) for part in ip.split(".")]
    response += struct.pack("BBBB", *ip_parts)
    return response

```

功能：

- 初始化：被实例化后首先进行解压
- `unpack`：解析 `DNS` 报文的头部信息，并根据报文类型（查询或响应）进一步解析问题部分
- `r_pack`：根据查询内容和本地解析结果生成响应报文。若本地配置中 IP 为 `0.0.0.0`，则表示拦截该请求；否则，返回配置中的 IP 地址

4. DNS 中继服务器类 `relay_server`

这里需要注意中继的第一段也属于报文的等待时间，刚开始写的时候没有往 `self.transaction` 存开始时间，存了 `(msg.query.name, addr)` 导致测试结果出来的处理时间一大片 `0.0000s`

此外，这里使用 `pop` 方法弹出任务字典，因为 `pop` 方法会从字典中删除指定的键，并返回该键对应的值，以便节省内存与避免重复处理请求

```

class relay_server:
    """中继器，接收DNS报文并处理"""

    def __init__(self, path) -> None:
        self.config = config(path) # 解析配置文件，存储为字典{name: ip}
        print(self.config)
        self.s = socket.socket(
            socket.AF_INET, socket.SOCK_DGRAM
        ) # AF_INET表示使用IPv4地址族 SOCK_DGRAM 表示UDP
        self.s.bind(("0.0.0.0", 53))
        # 将套接字绑定到本地地址 0.0.0.0 和端口 53
        # 0.0.0.0 表示绑定到所有可用的网络接口
        # 端口 53 是 DNS 服务的标准端口
        self.s.setblocking(False)
        # 在非阻塞模式下，套接字操作（如接收数据）不会阻塞程序的执行
        self.nameserver = (
            "114.114.114.114",
            53,
        ) # 一个公共DNS服务器，也可选择其他，注意事先测试

        self.transaction = {}
        # 空字典，用于存储事务ID和对应的查询报文，以便在收到回复报文时找到对应的查询报文

    def process(self, data, addr):
        """报文处理"""
        start_time = time()
        msg = message(data)
        domain_name = "unknown"
        handled_as = "unknown"
        flag = 0

        # 解析报文后，需要检查是否有匹配的配置。
        # 如果有匹配的配置，则生成相应的回复报文；
        # 否则，将查询转发到上游 DNS 服务器。
        if msg.qr == 0: # 是查询报文
            domain_name = msg.query.name
            # 存在本地配置的就优先从本地找，否则转发到上游DNS服务器
            if domain_name in self.config:
                ip = self.config[domain_name]
                response = msg.r_pack(ip)
                self.s.sendto(response, addr)
                if ip == "0.0.0.0":
                    handled_as = "intercept"
                else:
                    handled_as = "local resolve"
            else:
                transaction_id = msg.id
                if transaction_id not in self.transaction: # 避免重复修改任务字典
                    self.transaction[transaction_id] = (
                        msg.query.name,
                        addr,
                        start_time,
                    )
                self.s.sendto(data, self.nameserver)
                flag = 1

        elif msg.qr == 1: # 响应报文
            transaction_id = msg.id
            if transaction_id in self.transaction:
                domain_name, original_addr, start_time = self.transaction.pop(
                    transaction_id
                )
                self.s.sendto(data, original_addr)
                handled_as = "relay"
            else:
                handled_as = "[ERROR] unknown transaction id"
        else:
            handled_as = "[ERROR] unknown transaction id"

```

```

        return -1

    end_time = time()
    duration = end_time - start_time
    if flag == 0: # 简化输出, 第一段中继不打印日志
        logging.info(
            f"query to {domain_name:>50},     handled as {handled_as:>20},     takes {duration:.4f}s"
        )
    return 0

def run(self):
    """循环接收DNS报文"""
    while True:
        try:
            data, addr = self.s.recvfrom(1024)
            # 接收最多1024字节数据, 返回值是一个元组(接收数据, 发送数据的地址)
            threading.Thread(
                target=self.process,
                args=(data, addr),
                # 为process函数创建一个线程, 传入参数data和addr
            ).start()
        except Exception:
            pass

```

功能:

- **初始化:**

- 解析配置文件, 获取本地域名与 IP 的映射关系
- 创建并绑定 UDP 套接字到本地 53 端口, 设置为非阻塞模式
- 设置上游 DNS 服务器地址
- 初始化事务 ID 与查询信息的字典 transaction

- **process 方法:**

- **查询报文处理:**

- 如果查询的域名在本地配置中, 生成相应的响应报文并返回给请求方
- 若域名未在本地配置中, 记录事务信息并将查询报文转发给上游 DNS 服务器

- **响应报文处理:**

- 根据事务 ID 查找原始请求信息, 将上游 DNS 服务器的响应报文转发给原始请求方
- 记录并日志化每个查询的处理方式和耗时

- **run 方法:**

- 循环接收 DNS 报文, 每收到一个报文便创建一个新线程处理该报文

5. 主程序入口

```

if __name__ == "__main__":
    path = r"D:\Study_Work\Electronic_data\CS\AAAUUniversity\CN\Lab\1\example.txt"
    r = relay_server(path)
    r.run()

```

功能: 在非外部调用启动时指定配置文件路径, 初始化 DNS 中继服务器并启动运行

五、报文处理流程

以下详细描述 DNS relay 在接收到每个 DNS 报文后的处理流程:

1. 接收报文

- 服务器通过 UDP 套接字接收来自客户端的 DNS 查询报文或来自上游 DNS 服务器的响应报文
- 接收的数据包括报文内容 data 和发送方地址 addr

2. 创建线程处理报文

- 为每个接收到的报文创建一个新的线程, 调用 process 方法进行处理, 确保服务器能够并发处理多个请求

3. 解析报文

- 使用 `message` 类解析报文头部信息
- 判断报文类型：
 - **查询报文 (qr == 0):**
 - 进一步解析问题部分，获取查询的域名、类型和类
 - **响应报文 (qr == 1):**
 - 不需解析问题部分，直接处理响应

4. 处理查询报文

- **本地解析：**
 - 检查查询的域名是否在本地配置文件中
 - **若存在：**
 - 根据配置生成相应的响应报文：
 - 如果配置的 `IP` 为 `0.0.0.0`，表示拦截该请求
 - 否则，返回配置中的 `IP` 地址作为解析结果
 - 将生成的响应报文发送回客户端
 - 记录处理方式（拦截或本地解析）及耗时
 - **若不存在：**
 - 将事务 `ID`、域名、客户端地址和开始时间记录在 `transaction` 字典中
 - 将查询报文转发给上游 `DNS` 服务器
 - 标记该报文为需要中继处理

5. 处理响应报文

- **查找事务记录：**
 - 根据响应报文中的事务 `ID`，在 `transaction` 字典中查找对应的查询信息
 - **若找到：**
 - 将响应报文转发给原始请求客户端
 - 从 `transaction` 字典中移除该事务记录
 - 记录 处理方式为中继 及 耗时
 - **若未找到：**
 - 记录错误信息，表示收到未知的事务 `ID`

6. 生成并发送响应报文

- **本地生成响应：**
 - 使用 `message` 类的 `r_pack` 方法，根据查询内容和配置生成相应的响应报文
 - 发送给客户端。
- **转发响应：**
 - 将从上游 `DNS` 服务器接收到的响应报文直接转发给原始请求客户端

7. 记录日志

- 使用 `logging` 模块记录并格式化输出每个查询的详细信息，包括查询域名、处理方式和耗时

```
logging.info(  
    f"query to {domain_name:>50},     handled as {handled_as:>20},     takes {duration:.4f}s"  
)
```

六、报文生成细节

在生成响应报文时，主要涉及以下几个部分：

1. **报文头部：**
 - **事务 ID (id)：**保持与查询报文相同，以便客户端识别响应
 - **标志 (flags)：**
 - `0x8180`：标准查询响应，无错误
 - `0x8183`：名称错误，域名不存在

- **问题数 (`quests`)**: 保持与查询报文相同
 - **回答数 (`response_answers`)**: 通常为 `1`, 表示有一个回答记录
 - **授权记录数 (`author`) 和 附加记录数 (`addition`)**: 保持与查询报文相同
2. **问题部分**:
- 使用 `query_part` 类的 `pack` 方法将域名、类型和类重新打包成二进制格式
3. **回答部分**:

- **名称指针 (`0xC00C`)**: 指向问题部分的域名, 节省报文空间
- **类型 (`1`)**: 表示 `A` 记录 (`IPv4` 地址)
- **类 (`1`)**: 表示 `IN` 类 (互联网)
- **TTL (`666`)**: 资源记录在缓存中的生存时间, 单位为秒
- **数据长度 (`4`)**: `IPv4` 地址长度为 `4` 字节
- **IP 地址**: 将 `IP` 地址从字符串格式转换为二进制格式

七、执行流程分析

以下是每份报文在接收到之后代码的执行流程:

1. 接收到查询报文

- **示例**: 客户端查询 `www.test1.com`
- **执行流程**:
 - i. `relay_server` 的 `run` 方法接收到查询报文, 创建新线程调用 `process` 方法
 - ii. `process` 方法解析报文, 识别为查询报文, 获取域名 `www.test1.com`
 - iii. 检查配置文件, 发现 `www.test1.com` 映射为 `0.0.0.0`
 - iv. 使用 `message.r_pack` 方法生成拦截响应报文
 - v. 发送响应报文回客户端
 - vi. 记录日志: `handled as intercept`

2. 接收到查询报文, 域名未配置

- **示例**: 客户端查询 `www.test3.com`。
- **执行流程**:
 - i. `relay_server` 的 `run` 方法接收到查询报文, 创建新线程调用 `process` 方法
 - ii. `process` 方法解析报文, 识别为查询报文, 获取域名 `www.test3.com`
 - iii. 检查配置文件, 未找到匹配条目
 - iv. 记录事务 `ID` 与查询信息, 转发查询报文到上游 `DNS` 服务器
 - v. 记录日志: `handled as relay` (稍后接收到响应时输出)

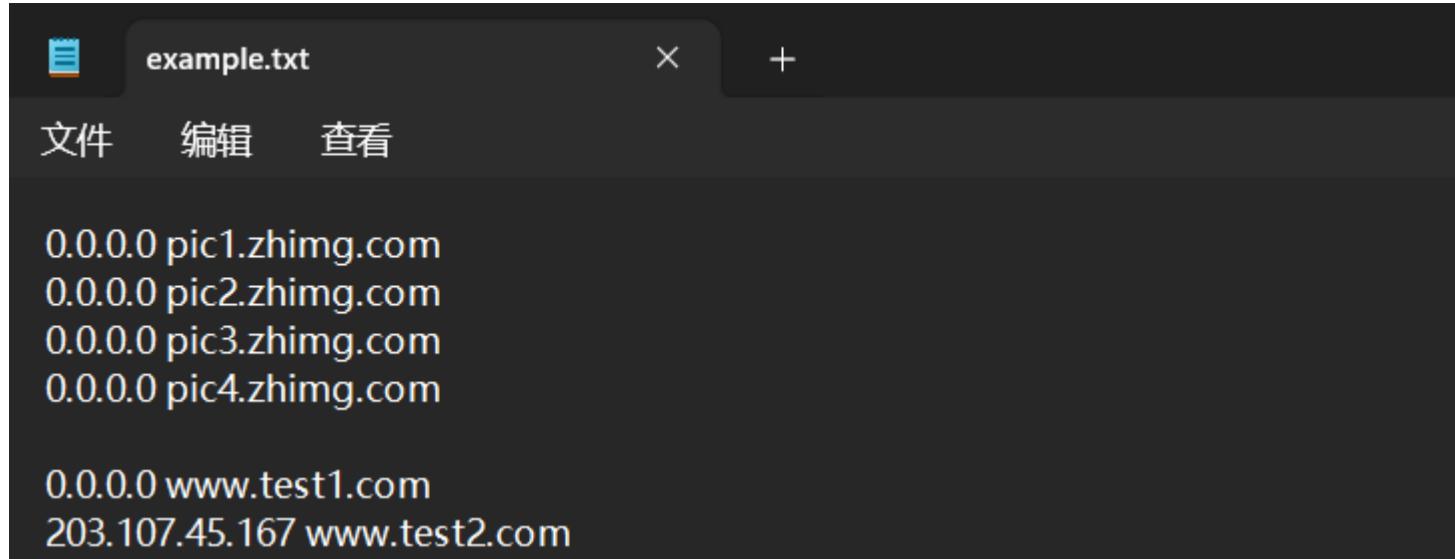
3. 接收到响应报文

- **示例**: 上游 `DNS` 服务器响应 `www.test3.com` 的查询
- **执行流程**:
 - i. `relay_server` 的 `run` 方法接收到响应报文, 创建新线程调用 `process` 方法
 - ii. `process` 方法解析报文, 识别为响应报文, 获取事务 `ID`
 - iii. 在 `transaction` 字典中查找对应的查询信息
 - iv. 找到后, 转发响应报文给原始请求客户端, 并移除事务记录
 - v. 记录日志: `handled as relay`

八、实验结果

1. 准备工作

配置文件未作修改



```
example.txt
X + 
文件 编辑 查看

0.0.0.0 pic1.zhimg.com
0.0.0.0 pic2.zhimg.com
0.0.0.0 pic3.zhimg.com
0.0.0.0 pic4.zhimg.com

0.0.0.0 www.test1.com
203.107.45.167 www.test2.com
```

```
& D:/Python/python.exe d:/Study_Work/Electronic_data/CS/AAAUniversity/CN/Lab/1/demo.py
{'pic1.zhimg.com': '0.0.0.0', 'pic2.zhimg.com': '0.0.0.0', 'pic3.zhimg.com': '0.0.0.0', 'pic4.zhimg.com': '0.0.0.0', 'www.test1.com': '0.0.0.0', 'www.test2.com': '203.107.45.167'}
```

程序完成初始化后首先输出了配置域名

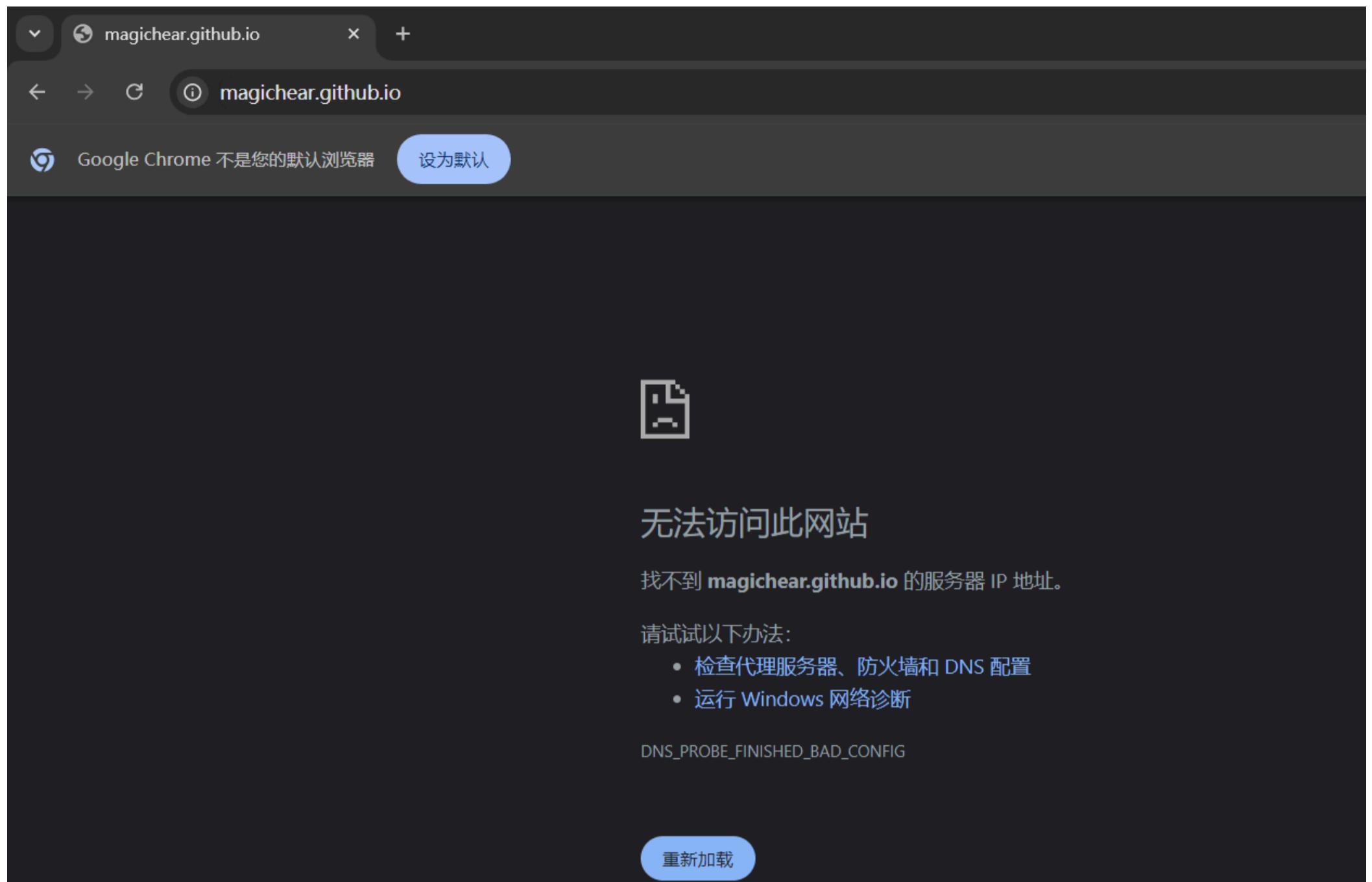
说明：

- pic1/2/3/4.zhimg.com 被配置为拦截 (IP 为 0.0.0.0)
- www.test1.com 被配置为拦截 (IP 为 0.0.0.0)
- www.test2.com 被本地解析为指定 IP (203.107.45.167)
- www.test3.com 未在本地配置，故通过中继转发到上游 DNS 服务器，具体返回结果取决于上游服务器

首先在设置->Internet->高级网络设置->WIFI-编辑->双击 IPv4(或单击后点击属性)->手动将 DNS 服务器首选地址设为本地->确定退出



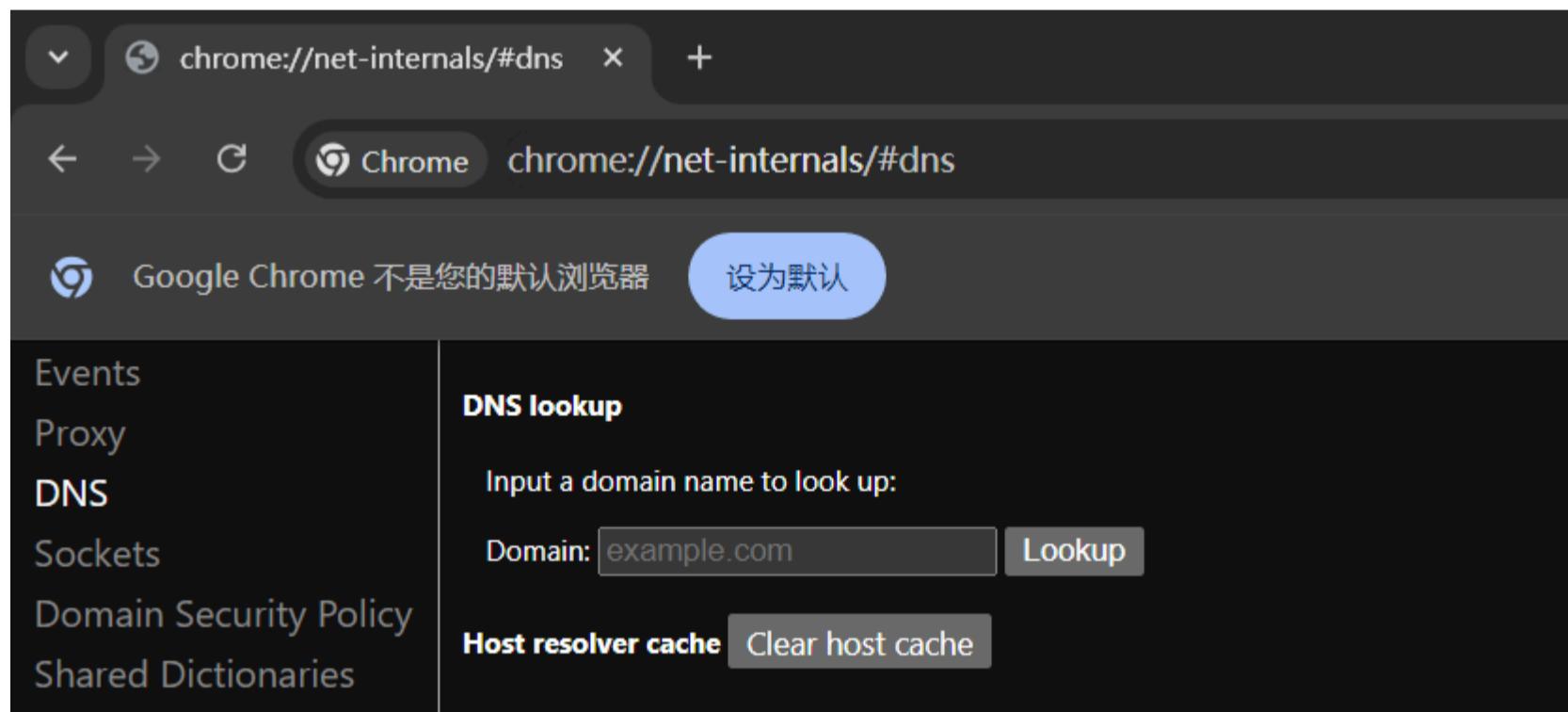
这时候由于我们的程序没有启动，机器是无法联网的



接下来，由于我日常使用 `EDGE` 较多，这里使用 `google chrome` 进行测试

在地址栏里输入 `chrome://net-internals` 后进入以下页面，随后选择 `DNS` 选项并单击 `Clear host cache` 清空本地缓存，同时可在工具栏中找到清除浏览器数据选项

这里清空图片缓存是为了后续测试对知乎广告域名的屏蔽排除干扰（测试发现即使多次清除也会这么显示）



删除浏览数据

基本

高级

时间范围

过去 7 天

浏览记录

删除历史记录，包括搜索框中的历史记录

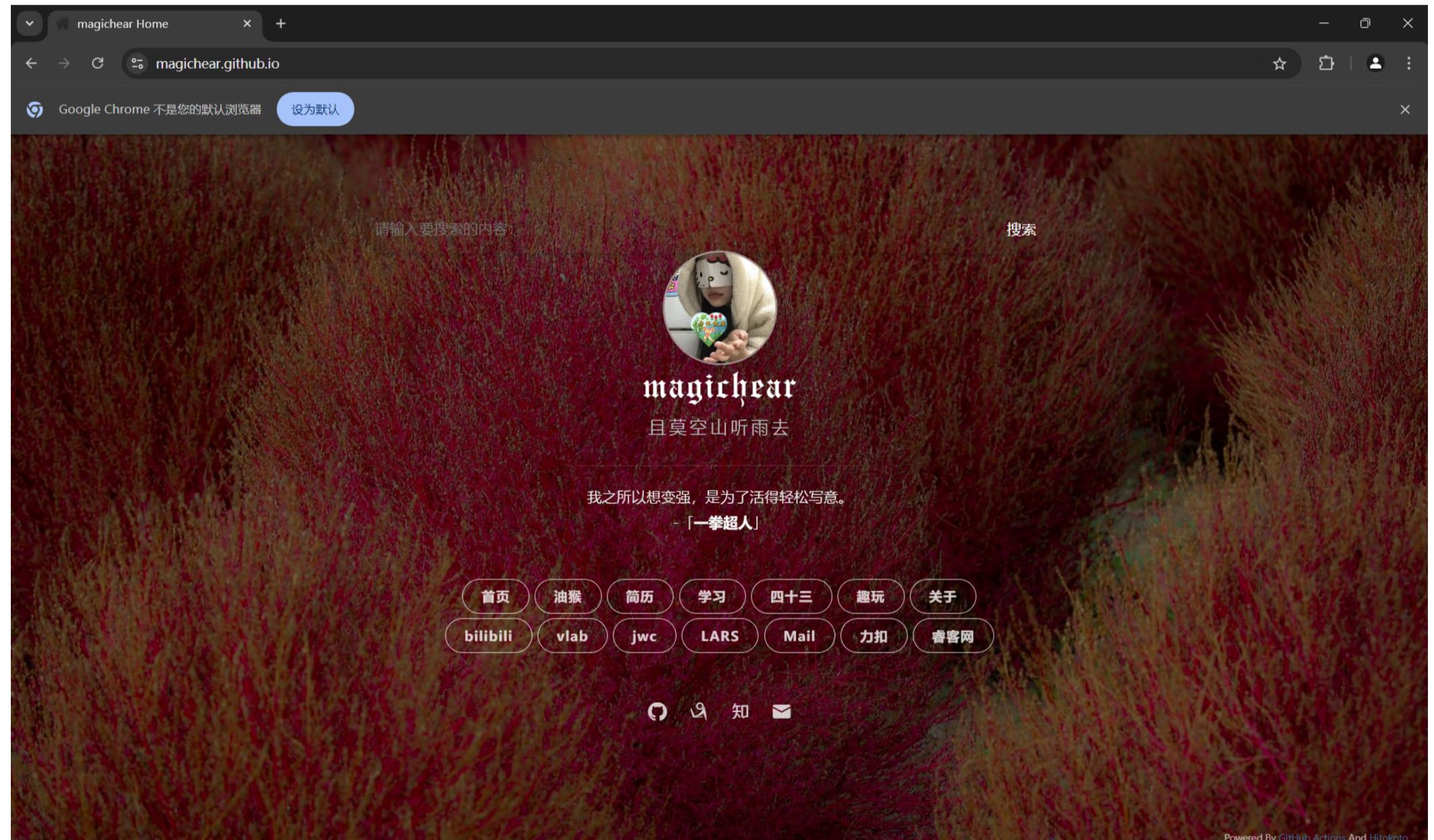
Cookie 及其他网站数据

会致使您从大多数网站退出账号

缓存的图片和文件

释放不到 36.3 MB 空间。当您下次访问时，某些网站的加载速度可能会更慢。

这时候启动我们的程序，发现已经可以正常联网了



Powered By Github Actions And Hitokoto

query to	magicbear.github.io,	handled as	relay,	takes 0.0581s
query to	mobile.events.data.microsoft.com,	handled as	relay,	takes 0.0200s
query to	mobile.events.data.microsoft.com,	handled as	relay,	takes 0.0200s
query to	mobile.events.data.microsoft.com,	handled as	relay,	takes 0.0200s
query to	magicbear.github.io,	handled as	relay,	takes 0.0179s
query to	unpkg.com,	handled as	relay,	takes 0.0179s
query to	unpkg.com,	handled as	relay,	takes 0.0189s
query to	unpkg.com,	handled as	relay,	takes 0.0189s
query to	magicbear.github.io,	handled as	relay,	takes 0.0570s
query to	magicbear.github.io,	handled as	relay,	takes 0.0606s
query to	magicbear.github.io,	handled as	relay,	takes 0.0189s
query to	magicbear.github.io,	handled as	relay,	takes 0.0189s
query to	unpkg.com,	handled as	relay,	takes 0.0179s
query to	unpkg.com,	handled as	relay,	takes 0.0200s
query to	unpkg.com,	handled as	relay,	takes 0.0210s
query to	unpkg.com,	handled as	relay,	takes 0.0210s

2. 命令行测试基本功能

打开 powershell 刷新 DNS，接着依次对三个网站使用 nslookup 进行测试，可以看到都达到了我们要的效果

```
PS D:\Study_Work\Electronic_data\CS\AAAUniversity\CN\Lab\1> ipconfig /flushdns
```

Windows IP 配置

已成功刷新 DNS 解析缓存。

```
PS D:\Study_Work\Electronic_data\CS\AAAUniversity\CN\Lab\1> nslookup -q=A www.test1.com
```

服务器: UnKnown

Address: 127.0.0.1

```
*** UnKnown 找不到 www.test1.com: Non-existent domain
```

```
PS D:\Study_Work\Electronic_data\CS\AAAUniversity\CN\Lab\1> nslookup -q=A www.test2.com
```

服务器: UnKnown

Address: 127.0.0.1

非权威应答:

名称: www.test2.com

Address: 203.107.45.167

```
PS D:\Study_Work\Electronic_data\CS\AAAUniversity\CN\Lab\1> nslookup -q=A www.test3.com
```

服务器: UnKnown

Address: 127.0.0.1

非权威应答:

名称: www.test3.com

Addresses: 45.33.30.197

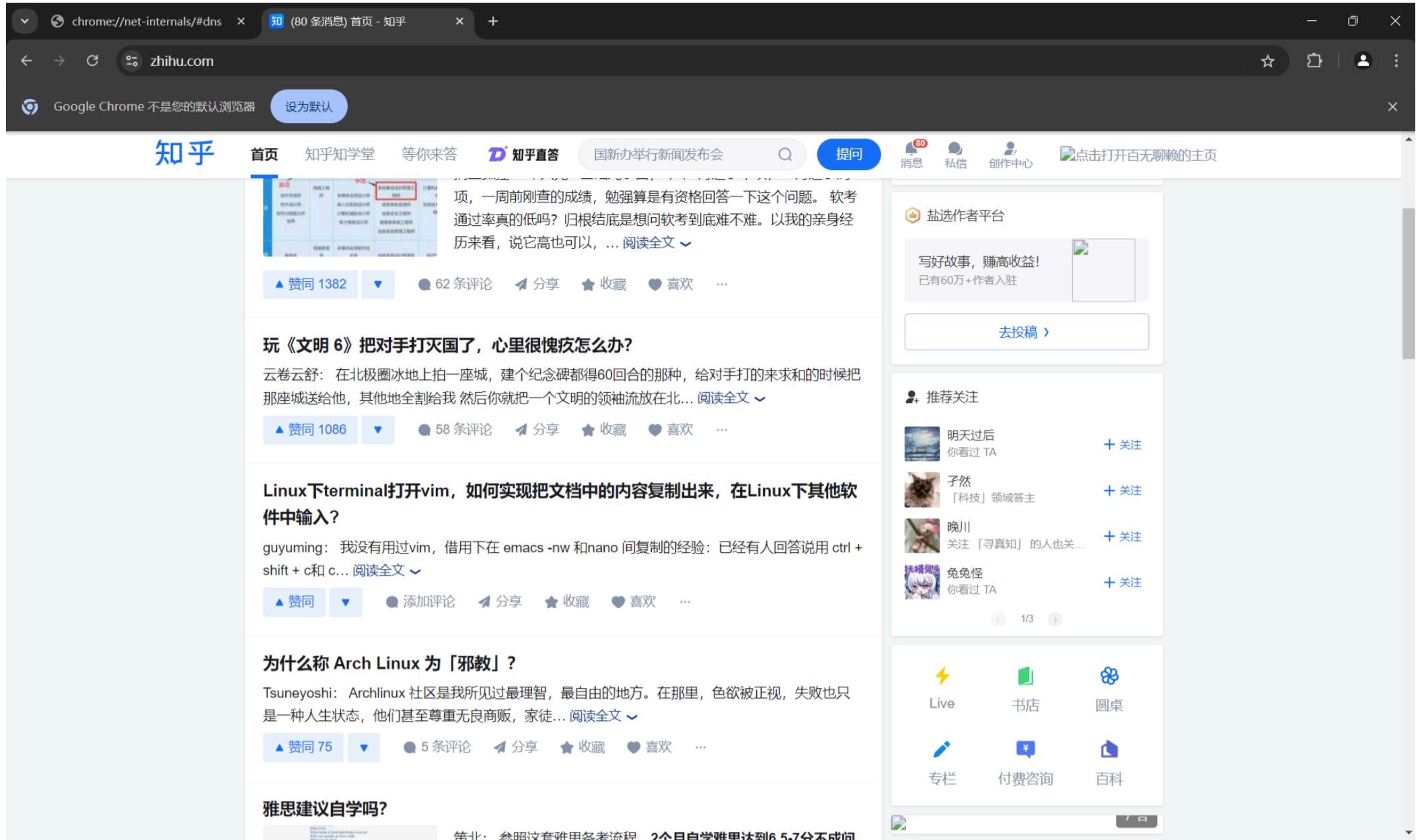
45.33.2.79

45.33.20.235

```
PS D:\Study_Work\Electronic_data\CS\AAAUniversity\CN\Lab\1>
```

query to	1.0.0.127.in-addr.arpa,	handled as	relay,	takes 0.0818s
query to	www.test1.com,	handled as	intercept,	takes 0.0000s
query to	www.test1.com,	handled as	intercept,	takes 0.0000s
query to	1.0.0.127.in-addr.arpa,	handled as	relay,	takes 0.0388s
query to	www.test2.com,	handled as	local resolve,	takes 0.0010s
query to	1.0.0.127.in-addr.arpa,	handled as	relay,	takes 0.0459s
query to	www.test3.com,	handled as	relay,	takes 0.0575s

3. 访问知乎测试屏蔽效果



Google Chrome 不是您的默认浏览器

设为默认

知乎

首页 知乎知学堂 等你来答 知乎直答 国新办举行新闻发布会 提问

消息 私信 创作中心

点击打开百无聊赖的主页

盐选作者平台

写好故事，赚高收益！

已有60万+作者入驻

去投稿 >

推荐关注

明天过后 你看过 TA +关注

子然 「科技」领域答主 +关注

晚川 关注「寻真知」的人也关注 +关注

兔兔怪 你看过 TA +关注

1/3

Live 书店 圆桌

专栏 付费咨询 百科

项，一周前刚查的成绩，勉强算是有资格回答一下这个问题。软考通过率真的低吗？归根到底是想问软考到底难不难。以我的亲身经历来看，说它高也可以，... 阅读全文 ▾

▲ 赞同 1382 ▾ 62 条评论 分享 收藏 喜欢 ...

玩《文明 6》把对手打灭了，心里很愧疚怎么办？

云卷云舒：在北极圈冰地上拍一座城，建个纪念碑都得60回合的那种，给对手打的来求和的时候把那座城送给他，其他地全割给我 然后你就把一个文明的领袖流放在北... 阅读全文 ▾

▲ 赞同 1086 ▾ 58 条评论 分享 收藏 喜欢 ...

Linux下terminal打开vim，如何实现把文档中的内容复制出来，在Linux下其他软件中输入？

guyuming：我没有用过vim，借用下在 emacs -nw 和 nano 间复制的经验：已经有人回答说用 ctrl + shift + c 和 c... 阅读全文 ▾

▲ 赞同 1086 ▾ 58 条评论 分享 收藏 喜欢 ...

为什么称 Arch Linux 为「邪教」？

Tsuneyoshi：Archlinux 社区是我所见过最理智，最自由的地方。在那里，色欲被正视，失败也只是一种人生状态，他们甚至尊重无良商贩，家徒... 阅读全文 ▾

▲ 赞同 75 ▾ 5 条评论 分享 收藏 喜欢 ...

雅思建议自学吗？

笨牛 - 雅思口语自学备考流程 20天自学雅思口语到6.5-7分不成功

query to	www.zhihu.com,	handled as	relay,	takes 0.0190s
query to	www.zhihu.com,	handled as	relay,	takes 0.0180s
query to	www.zhihu.com,	handled as	relay,	takes 0.0180s
query to	zhihu.com,	handled as	relay,	takes 0.0180s
query to	zhihu.com,	handled as	relay,	takes 0.0180s
query to	zhihu.com,	handled as	relay,	takes 0.0310s
query to	play.google.com,	handled as	relay,	takes 0.0180s
query to	pic1.zhimg.com,	handled as	intercept,	takes 0.0000s
query to	pic1.zhimg.com,	handled as	intercept,	takes 0.0000s
query to	pic1.zhimg.com,	handled as	intercept,	takes 0.0000s
query to	static.zhimg.com,	handled as	relay,	takes 0.0170s
query to	static.zhimg.com,	handled as	relay,	takes 0.0190s
query to	static.zhimg.com,	handled as	relay,	takes 0.0200s
query to	pica.zhimg.com,	handled as	relay,	takes 0.0190s
query to	pica.zhimg.com,	handled as	relay,	takes 0.0200s
query to	pica.zhimg.com,	handled as	relay,	takes 0.0200s
query to	picx.zhimg.com,	handled as	relay,	takes 0.0200s
query to	picx.zhimg.com,	handled as	relay,	takes 0.0200s
query to	pic4.zhimg.com,	handled as	relay,	takes 0.0210s
query to	pic4.zhimg.com,	handled as	intercept,	takes 0.0000s
query to	pic4.zhimg.com,	handled as	intercept,	takes 0.0000s
query to	pic3.zhimg.com,	handled as	intercept,	takes 0.0000s
query to	pic3.zhimg.com,	handled as	intercept,	takes 0.0000s
query to	pic2.zhimg.com,	handled as	intercept,	takes 0.0000s
query to	pic2.zhimg.com,	handled as	intercept,	takes 0.0000s
query to	pic2.zhimg.com,	handled as	intercept,	takes 0.0010s
query to	static.zhimg.com,	handled as	relay,	takes 0.0185s
query to	static.zhimg.com,	handled as	relay,	takes 0.0195s
query to	static.zhihu.com,	handled as	relay,	takes 0.0185s
query to	static.zhihu.com,	handled as	relay,	takes 0.0195s
query to	static.zhihu.com,	handled as	relay,	takes 0.0195s
query to	hm.baidu.com,	handled as	relay,	takes 0.0220s
query to	hm.baidu.com,	handled as	relay,	takes 0.0230s
query to	hm.baidu.com,	handled as	relay,	takes 0.0220s
query to	sugar.zhihu.com,	handled as	relay,	takes 0.0190s
query to	sugar.zhihu.com,	handled as	relay,	takes 0.0190s
query to	sugar.zhihu.com,	handled as	relay,	takes 0.0195s

从知乎界面与程序的输出日志来看，配置里屏蔽的几个域名都被成功拦截了

未来改进方向：

- 增加缓存机制，提升解析效率。
- 实现更多 DNS 记录类型的处理。
- 增强错误处理和日志记录的详细程度。
- 支持 IPv6 地址解析。