**北京邮电大学课程设计报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课程设计**  **名称** | **计算机网络**  **课程设计** | | **学 院** | **计算机** | **指导教师** | **吴起凡** |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2018211316** | **15** | **2018211582** | | **李志毅** |  | |
| **2018211317** | **7** | **2018211605** | | **李泊昊** |  | |
| **2018211317** | **12** | **2018211610** | | **赵浩天** |  | |
| **课**  **程**  **设**  **计**  **内**  **容** | （简要介绍课程设计的主要内容，包括课程设计教学目的、基本内容、实验方法和团队分工等）  本次课程设计主要内容为利用C语言实现DNS中继服务器程序。  最终实现的DNS中继服务程序功能包括但不限于本地查询返回、外部中继响应、特定域名屏蔽、LRU算法缓存、多客户并发ID转换。  主要考察了关于计算机网络通信流程、DNS报文的解析、DNS服务器工作原理、socket通信原理以及其他编程相关等知识。通过团队形式，经过研讨、设计、开发、测试、验收阶段流程，充分研究了DNS报文构成、计算机网络通信流程、通信及服务程序相关编程手段、报文处理及缓存算法等内容。项目过程中由李志毅同学负责socket通信、报文解析、本地处理服务部分；由赵浩天同学负责中继处理服务、多客户并发、ID转换部分；  由李泊昊同学负责Cache缓存配置、LRU算法、超时反馈处理。其他项目相关工作均由小组成员协同合作完成。 | | | | | |
| **学生**  **课程设计**  **报告**  （附页） | （见附页） | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | **评语**:  **成绩**:  指导教师签名：  年 月 日 | | | | | |

注：评语要体现每个学生的工作情况，可以加页。

# 目录

[目录 2](#_Toc49976184)

[一、功能设计 3](#_Toc49976185)

[二、开发环境 4](#_Toc49976186)

[三、模块设计 5](#_Toc49976187)

[模块一 初始化模块 5](#_Toc49976188)

[模块二 监听模块 14](#_Toc49976189)

[模块三 请求处理模块 16](#_Toc49976190)

[模块四、响应处理模块 21](#_Toc49976191)

[模块五 缓存模块 23](#_Toc49976192)

[四、流程图 29](#_Toc49976193)

[1、程序总体流程 29](#_Toc49976194)

[2、ID转换流程 30](#_Toc49976195)

[3、LRU-Cache缓存流程 31](#_Toc49976196)

[五、测试用例与测试结果 32](#_Toc49976197)

[六、问题解决 40](#_Toc49976198)

[七、心得总结 42](#_Toc49976199)

[附件 43](#_Toc49976200)

[1、版本记录 43](#_Toc49976201)

[2、源代码 44](#_Toc49976202)

# 一、功能设计

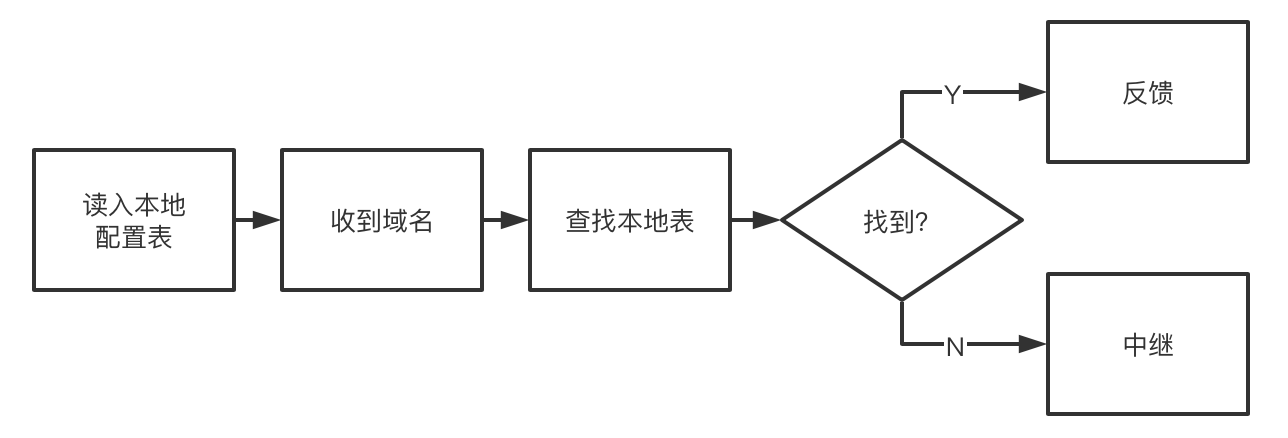
本次课程设计需要完成一项DNS中继服务器程序。

客户机基于DNS协议通过UDP字节流报文的形式与服务器通信，向服务器传输DNS报文提供域名，而服务器需要对该报文进行处理并反馈给客户主机响应对DNS响应报文。由此推出课程设计中DNS中继服务器需要主要实现的两项功能为：

本地检索查询、外部中继反馈。

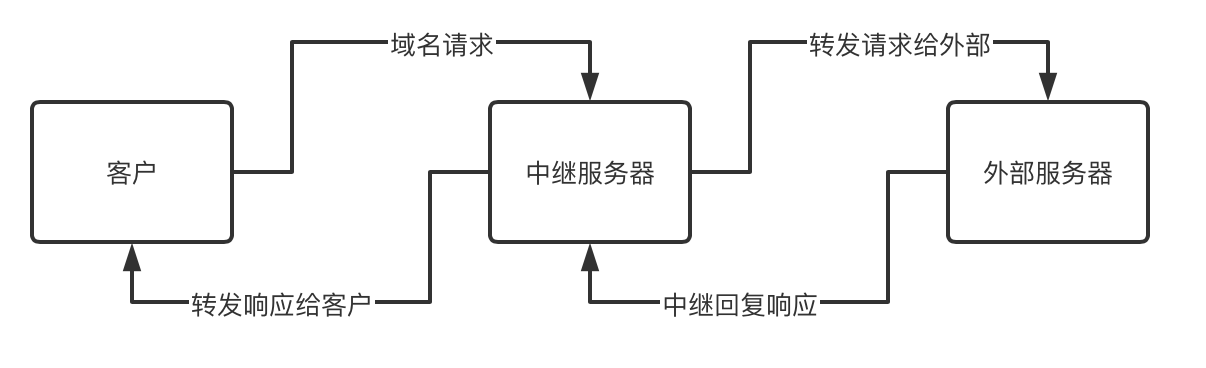
**1、本地检索查询：**

首先需要实现本地配置文件的读入。其次对于Client传入服务器端的域名在已经读入的表中进行本地查询检索，若查询成功则向Client返回其所查找到的ip地址（屏蔽功能）；若查询失败则向外部中继。



**2、外部中继反馈**

对于本地配置表中检索不到的项目，需要向外部域名服务器再次发起中继请求（注意这一过程中的多客户ID转换问题）；而后外部域名服务器反馈给中继服务器响应，中继服务器再回发给对应客户Client主机。



主要功能之外需要额外注意实现其他必要细节功能：

**1、通信功能：**

需要能够实现客户机与服务器之间的socket通信功能。在服务器与客户主机之间通过套接字手段建立socket通信，采用UDP方式传输DNS报文。

**2、屏蔽功能：**

需要能够实现屏蔽特定设置的域名，使客户机无法访问对应域名。当在本地表项中查找到请求的域名后，若域名内容为0.0.0.0表示为屏蔽，返回0.0.0.0使客户机无法访问到对应域名地址从而实现屏蔽功能。

**3、类型辨析/iPv6功能：**

对于客户机发出iPv4以及iPv6请求时均需要作出应答。iPv4查找本地，iPv6在非屏蔽情况下中继转发。

**4、多客户/ID转换功能：**

需要支持多个客户接入服务器时服务器正常运行。DNS报文采用ID编号进行标示，对于多个客户机接入同一服务器时可能会出现ID相同的情况。因此中继过程中需要对ID编号进行转换，确保每一中继出去给外部的报文编号不能重复。并且需要在接收到响应后转换回客户所发送的ID从而保证多个客户同时接入时能够不发生冲突。

**5、多端并发功能：**

需要允许某一客户收到回复之前就能够接受处理其他用户的请求。

**6、超时处理功能：**

对于接受外部服务器响应报文超时的情况下不再接收超时到达/未到达报文。每次中继过程中通过计时刷新中继转发表，删除超时表项避免空间浪费溢出，抛弃超时报文。

**7、使用LRU算法的Cache缓存功能：**

对于客户向服务器所发起的请求进行缓存，在中继过程中对用户请求的域名URL以及返回的IP地址进行缓存。在用户下次进行调用的过程中，可以直接先在本地缓存表中进行搜索判定，直接对缓存中的ip进行使用而省去中继步骤，有效提高程序效率。

另一方面，对于缓存表设置固定大小，使用LRU最少使用替换算法，在缓存表满的情况下对最少使用的表项进行替换，更进一步优化DNS服务。

# 二、开发环境

**操作系统：** Windows10

**IDE：** Visual Studio 2017、

Visual Studio 2019

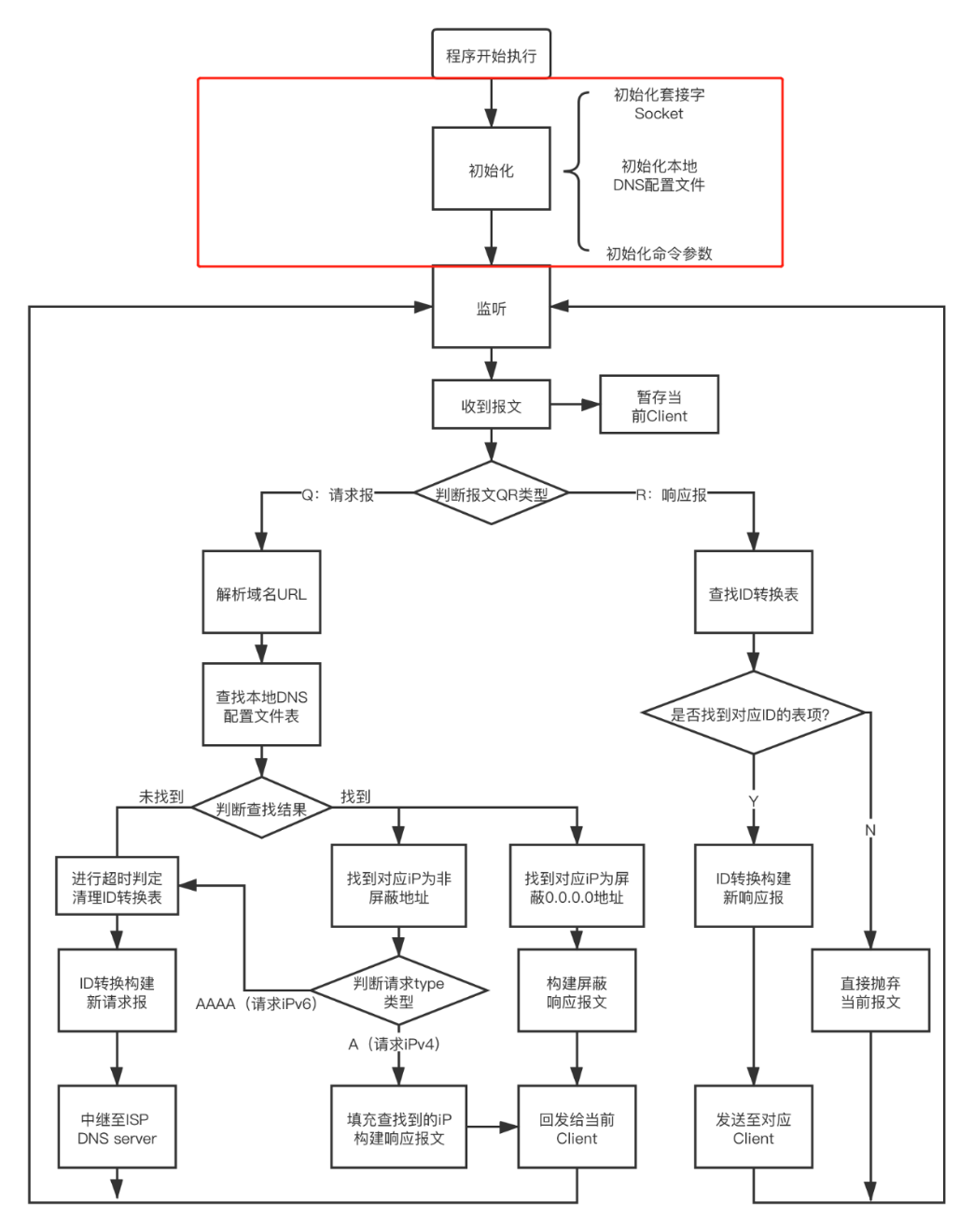
# 三、模块设计

## 模块一 初始化模块

* 1. **功能介绍**

初始化模块主要完成系统初始化部分，包含初始化命令参数，初始化套接字socket，以及初始化本地DNS配置文件三个主要部分。对于初始化命令参数及初始化本地DNS配置文件两个部分，通过两个函数得以实现，而初始化套接字socket在main函数中体现。

函数deal\_parameters用于实现设置执行可执行文件时输入的命令参数；函数local\_Init用于实现“域名-IP”对照表的读取，即初始化本地DNS配置文件，以及ID转换链表的初始化，其中ID转换链表用于，多个未在“域名-IP“对照表查询到的域名请求，将该请求转发至官方DNS服务器时ID转换时使用，具体功能介绍和功能实现在请求处理和相应处理模块有所体现。



* 1. **初始化套接字socket**
     1. **介绍及功能实现**

Socket是使用标准Unix文件描述符(file descriptor)和其他程序通讯的方式，文件描述符是指和打开的文件相关联的整数，在和Internet上别的程序通讯的时候，需要使用文件描述符，利用系统调用socket()，系统返回套接字描述符，再通过此套接字进行send()和recv()调用，在Windows系统中，Windows下的socket程序依赖Winsock.dll或者ws2\_32.dll,使用类似于“文件描述符”的“文件句柄”概念，在Windows环境下，socket()函数的返回值不再是整数，而是SOCKET类型，即句柄。

Internet套接字有两种类型，“SOCK\_STREAM”（流格式）和“SOCK\_DGRAM”（数据报格式），流式套接字是可靠的双向通讯的数据流，它使用的是TCP协议进行数据传输，数据报套接字是无连接的数据流，它使用的是UDP协议进行数据传输。在本次课设中，由于DNS大部分是运行在UDP协议上的，因此我们使用的套接字均是无连接的SOCK\_DGRAM格式。

* + 1. **全局变量及数据结构字典**

**·sockaddr**

struct sockaddr {

unsigned short sa\_family; /\*地址家族，AF\_XXX\*/

char sa\_data[14]; /\*14字节协议地址\*/

}

Sockaddr为许多类型的套接字存储套接字地址信息，sa\_family有多种类型，代表了套接字对应的协议族，常见的有AF\_UNIX（本机通信），AF\_INET（TCP/IP——IPv4），AF\_INET6（TCP/IP——IPv6）等。sa\_data包含套接字中的目标地址和端口信息。

**·sockaddr\_in**

struct sockaddr\_in {

short int sin\_family; /\*通信类型\*/

unsigned short int sin\_port; /\*端口\*/

struct in\_addr sin\_addr; /\*Internet地址\*/

unsigned char sin\_zero[8]; /\*与sockaddr结构的长度相同\*/

};

sockaddr\_in是与sockaddr并列的一个结构，用于处理sockaddr，用sockaddr\_in可以轻松的处理套接字地址的基本元素，其中sin\_zero应该使用bzero()或memset()来全部置零，sin\_family和struct addr中的sa\_family一致并能够设置为“AF\_INET”，sin\_port和sin\_addr必须是网络字节顺序的。

**·in\_addr**

struct in\_addr {

unsigned long s\_addr;

};

存储网络地址的结构体，使用的是网络字节顺序

**·WSAStartup**

int WSAStartup{

WORD wVersionRequested /\*用户调用Winsock的版本号\*/

LPWSADATA lpWSAData /\*指向WSADATA结构体的指针\*/

};

WSAStartup函数用于初始化供进程调用的Winsock相关的dll，调用成功WSAStartup函数返回0，否则返回五种错误代码之一

**·Socket()函数**

SOCKET socket(int af, int type, int protocol)

socket函数用于创建指定传输服务的socket，它有三个参数

**af:** address family，指明地址簇类型，例如 AF\_INET

**type:** 指明socket类型，例如SOCK\_STREAM，SOCK\_DGRAM

**protocol:** 指明数据传输协议，该参数取决于af和type的类型，常见的有IPPROTO\_TCP,IPPROTO\_UDP

如果创建成功，socke函数返回socket的描述符(句柄)，否则返回INVALID\_SOCKET

**·bind**

int bind(

SOCKET s,

const struct sockaddr\* name,

int namelen

);

bind函数将socket关联一个本地地址, 它有三个参数

**s**：指定一个未绑定的socket。

**name**：指向sockaddr地址的指针，该结构含有IP和PORT

**namelen：** 参数name的字节数。

无错误返回0，有错误返回SOCKET\_ERROR。

**·listen**

int listen(

SOCKET s,

int backlog

);

用于服务端网络监听

s: socket描述符，该socket是一个未连接状态的socket

backlog:挂起连接的最大长度，如果该值设置为SOMAXCONN，负责socket的底部服务提供商将设置该值为最大合理值。并没有该值的明确规定。

没有错误发生将返回0，否则返回SOCKET\_ERROR

**·accept**

SOCKET accept(

SOCKET s,

struct sockaddr\* addr,

int\* addrlen

);

服务端connect接收

**s:** listen函数用到的socket。accept函数将创建连接。

**addr:** 指向通信层连接实体地址的指针。addr 的格式取决于bind函数内地址簇的类型。

**addrlen:** addr的长度。

如果不发生错误，accept将返回一个新的SOCKET描述符，即新建连接的socket句柄。否则，将返回INVALID\_SOCKET。传进去的addrlen应该是参数addr的长度，返回的addrlen是实际长度。

**·connect**

int connect(

SOCKET s,

const struct sockaddr\* name,

int namelen

);

客户端请求连接

s：与bind函数的name参数类似，指明待连接的地址

namelen: Length of name, in bytes.

0表示正确，否则，将返回SOCKET\_ERROR。如果是阻塞式的socket连接，返回值代表了连接正常与失败。

**·sendto()和recvfrom()**

int sendto(

SOCKET s,

const char FAR \* buf,

int len,

int flags,

const struct sockaddr FAR \* to,

int tolen

);

**s:**已建立的socket套接字

**buf:** 数据buffer

**len:** 待发送数据长度

**flags:** sendto函数的发送（接收）数据方式。

**to:** 发往的sockaddr指针

**tolen:** int 型的局部指针，它的初始值为 sizeof(struct sockaddr)。函数调用返回后，tolen 保存着实际储存在 to 中的地址的长度。

int recvfrom(

SOCKET s,

char FAR \* buf,

int len,

int flags,

struct sockaddr FAR \* from,

int FAR \* fromlen

);

**s:**已建立的socket套接字

**buf:** 数据buffer

**len:** 待接受数据长度

**flags:** recvfrom函数的发送（接收）数据方式。

**from:** 源sockaddr指针

**fromlen：**int 型的局部指针，它的初始值为 sizeof(struct sockaddr)。函数调用返回后，fromlen 保存着实际储存在 from 中的地址的长度。

recvfrom的返回值标识已接收数据的长度。如果连接已关闭，返回值将是0,返回SOCKET\_ERROR标识recvfrom出错。

**·closesocket()**

int closesocket(

SOCKET s

);

如果无错误发生，函数返回0。否则，返回SOCKET\_ERROR。

**1.2.3 地址转换**

**·htons和htonl**

u\_short htons(u\_short hostshort);

u\_long htonl(u\_long hostlong);

各个机器cpu对数据存储和表示的方法不通，intel机器用littele-endian存数据，而IBM机器用big-endian存数据。网络协议为取消这种差异，一致采用big-endian方式。htons用于将unsigned short的数值从littele-endian转换为big-endian，由于short只有2字节，常用语port数值的转换。htons用于将unsigned long的数值从littele-endian转换为big-endian，long有4字节，常用于ipv4地址的转换。

**·inet\_addr和inet\_ntoa**

unsigned long inet\_addr(const char\* cp);

char\* FAR inet\_ntoa( struct in\_addr in);

inet\_addr用于将ipv4格式的字符串转换为unsigned long的数值。inet\_ntoa用于将struct in\_addr的地址转换为ipv4格式的字符串。

**·INADDR\_ANY**

用INADDR\_ANY来配置IP地址，意味着不需要知道当前服务器的IP地址。对于多网卡的服务器，INADDR\_ANY允许你的服务接收一个服务器上所有网卡发来的数据。

* 1. **初始化命令参数**
     1. **介绍及功能实现**

在执行exe文件时，我们将根据输入的指令执行相关程序，其中输入的指令可以配置的内容包含：DNS中继服务器调试信息输出显示的详细程度、DNS中继服务器是否开启缓存功能、官方DNS中继服务器的IP地址、本地“域名-IP“对照表的名称。因此输入指令dns后，根据继续输入的指令，可以配置以上内容。

对于配置DNS中继服务器运行过程显示的详细程度，该配置是一个必选项，其中可供用户选择的指令包含：-n、-d、-dd，其中，-n表示Debug等级为0，DNS中继服务器调试信息不输入显示；-d表示Debug等级为1，DNS中继服务器调试信息输出显示内容较精简；-dd表示Debug等级为2，DNS中继服务器调试信息输出显示内容详细。

对于配置DNS中继服务器是否开启缓存功能，当输入-c时，服务器开启缓存功能，不输入时或输入-nc时，服务器关闭缓存功能。缓存功能主要保存本地查询不到的域名IP对应关系，当下次查询时，直接从缓存区构建响应报文进行回复。

对于配置官方DNS中继服务器的IP地址，配置该地址作为第二项参数，默认该官方DNS中继服务器IP地址为223.5.5.5，该IP地址是一个Ali提供的官方中继服务器，若选择使用默认IP地址，则该参数无需输入，默认选择为官方DNS中继服务器。若需要更换官方中继服务器IP地址，则输入需要替换的官方DNS中继服务器地址即可。

对于配置本地“域名-IP“对照表，该配置作为第三项参数，默认该”域名-IP“对照表名称为dnsrelay.txt，若选择使用默认的对照表，则该项参数无需输入，默认选择名称为dnsrelay.txt的对照表，若需要修改对照表，则输入对照表名称即可。

* + 1. **字典介绍**
       1. **函数字典**

【函数原型】：void deal\_parameters(int argc, char\* argv[])

【作用】：初始化命令参数，该函数在main函数启动后即调用，完成初始化命令参数的全部工作，根据输入的参数个数和参数中的内容，可以完成DNS中继服务器的调试信息输出详细程度配置、官方DNS服务器地址配置、“域名-IP“对照表配置。

【返回值】：无

【参数】：argc表示输入参数的个数，argv是一个字符串数组，表示了对应参数下输入的参数字符串

* + - 1. **全局变量字典**

【变量名】：srv\_ip

【类型】：char字符串

【作用】：表示官方的DNS服务器，默认IP地址为223.5.5.5，是Ali的官方DNS服务器，可以通过参数配置进行修改。

【变量名】：host\_file

【类型】：char字符串

【作用】：表示本地DNS配置文件，即“域名-IP“地址对照表，默认文件名称为” dnsrelay.txt “，可以通过参数配置进行修改。

【变量名】：dbg\_flag  
【类型】：int

【作用】：表示指令输入的第一个参数的类型，用于表示DNS中继服务器调试信息输出的详细程度。

* 1. **初始化本地DNS配置文件及ID转换链表**
     1. **介绍及功能实现**

根据需求，当DNS中继服务器收到查询信息后，首先查询“域名-IP“对照表，若”域名-IP“对照表中包含对应信息，直接回复用户主机反馈信息，因此需要初始化本地DNS配置文件。

根据配置的本地DNS配置文件名称，打开文件，依次读取域名IP地址对照信息，并保存，当中继服务器收到查询信息后，根据读入的“域名-IP“对照表进行查询。

而ID转换链表用于，多个未在“域名-IP“对照表查询到的域名请求，将该请求转发至官方DNS服务器时ID转换时使用，在这里进行初始化后，在请求处理和相应处理中添加ID转换链结，进行信息存储。

* + 1. **函数字典**

【函数原型】：void local\_Init()

【作用】：初始化本地dns存储文件、ID转换链表

【返回值】：无

【参数】：无

* + 1. **读入本地DNS配置文件部分字典**

**全局变量：**

【变量名】：local\_url

【类型】：char字符串数组

【作用】：记录从本地DNS配置文件中读取的域名

【变量名】：local\_ip

【类型】：char字符串数组

【作用】：记录从本地DNS配置文件中读取的ip地址

【变量名】：local\_number

【类型】：int

【作用】：记录从本地DNS配置文件中读取的域名-ip对应信息

* + 1. **ID转换链表部分字典**

**数据结构：**

1. **typedef** **struct** ConId {   //对中继转发的报文进行ID转换
2. **struct** sockaddr\_in addr;//保存Client地址
3. **char** clientID[2];       //Client发给本机的ID（允许重复）
4. **char** convertID[2];      //本机中继转发的ID（不允许重复）
5. **clock\_t** starttime;      //转发出去的时间，2秒后超时自动清除该链结
6. **int** LRU\_cache;
7. **struct** ConId\* nextptr;
8. }ConId;

【数据结构名称】：ConId

【作用】：对中继转发的报文进行ID转换，保证中转后数据传输的可靠性。

【元素】：

【元素名称】：addr

【元素类型】：**struct** sockaddr\_in

【元素作用】：保存Client地址，用于保存中继出的请求报文的发送方，当中继服务器收到外部DNS服务器后，可以正确发送至用户。

【元素名称】：clientID

【元素类型】：char数组

【元素作用】：记录该Client端发给本机的报文中的ID，由于中继服务器需要接收多用户的请求报文，当该请求需要中继至官方DNS服务器查询时，中继服务器可能记录多个用户发来的ID，因此该ID可能出现重复的情况，需要替换。

【元素名称】：convertID

【元素类型】：char数组

【元素作用】：记录经过ID转换后替换的ID，该ID作为新的请求报文中的ID，将该报文发送给官方DNS服务器。

【元素名称】：starttime

【元素类型】：clock\_t

【元素作用】：记录创建该链结的时间，若创建后2s内仍未收到官方服务器发来的应答报文，则视为超时，删除该链表。这里判断超时的方式是，当有新的链结添加至链表中时，对于链表中的每一个链结，也就是每一个还未收到回复的请求报文，检查其是否超时，若超时，则删除。随着中继出去的报文的频率增加，对于链表中超时链结的检查频率越高，也就是当请求信息增加时，可能造成的链表过长，而该设定大大降低了链表超长的现象。

【元素名称】：LRU\_cache

【元素类型】：int

【元素作用】：指向该响应报文对应的中继请求报文中域名记录的缓存位置。

【元素名称】：nextptr

【元素类型】：**struct** ConId\*

【元素作用】：由于该数据结构应用为一个链表，该元素表示下一个链结。

**全局变量**

【变量名】：headptr

【类型】：ConId

【作用】：指向ID转换链表表头

【变量名】：tailptr

【类型】：ConId

【作用】：指向ID转换链表表尾

## 模块二 监听模块

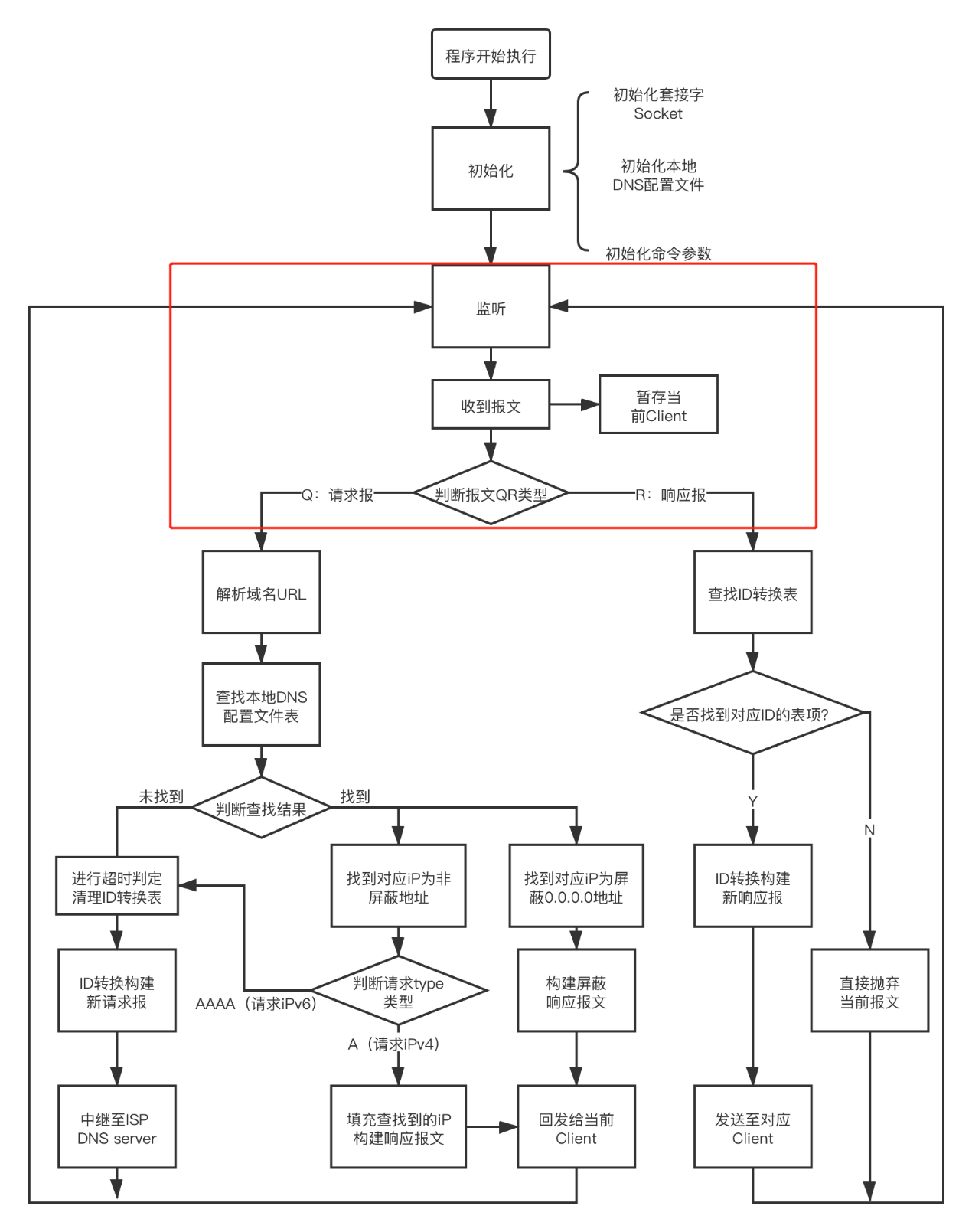
**2.1 功能介绍**

监控报文，作为DNS中继服务器需要监控全部报文，并判断报文类型。

当监控到的报文为请求报文时，该报文是用户发送的DNS请求，需要DNS中继服务器通过查找当前“域名-IP“对照表，或转发至官方DNS服务器进行查询。

当监控到的报文为响应报时，该报文是官方DNS服务器响应回复的报文，需要DNS中继服务器修改信息后转发至用户。

监控模块通过函数monitor得以实现，而监控到的报文为请求报文时，调用函数deal\_Question，该部分具体细节详见请求处理模块；监控到的报文为响应报文时，调用函数deal\_Respond，该部分具体细节详见响应处理模块。



**2.2 函数字典**

【函数原型】：void monitor()

【作用】：实现监控模块全部功能。监控全部报文，并判断报文类型。当监控到的报文为请求报文时，该报文是用户发送的DNS请求，需要DNS中继服务器通过查找当前“域名-IP“对照表，或转发至官方DNS服务器进行查询。当监控到的报文为响应报时，该报文是官方DNS服务器响应回复的报文，需要DNS中继服务器修改信息后转发至用户。

【返回值】：无

【参数】：无

**2.3 其他字典**

**宏：**

【名称】：BUF\_SIZE

【作用】：设定应用层信息的最长字节数

**全局变量：**

【名称】：temp

【类型】：struct sockaddr\_in

【作用】：用于记录监控到的报文的信息。

## 模块三 请求处理模块

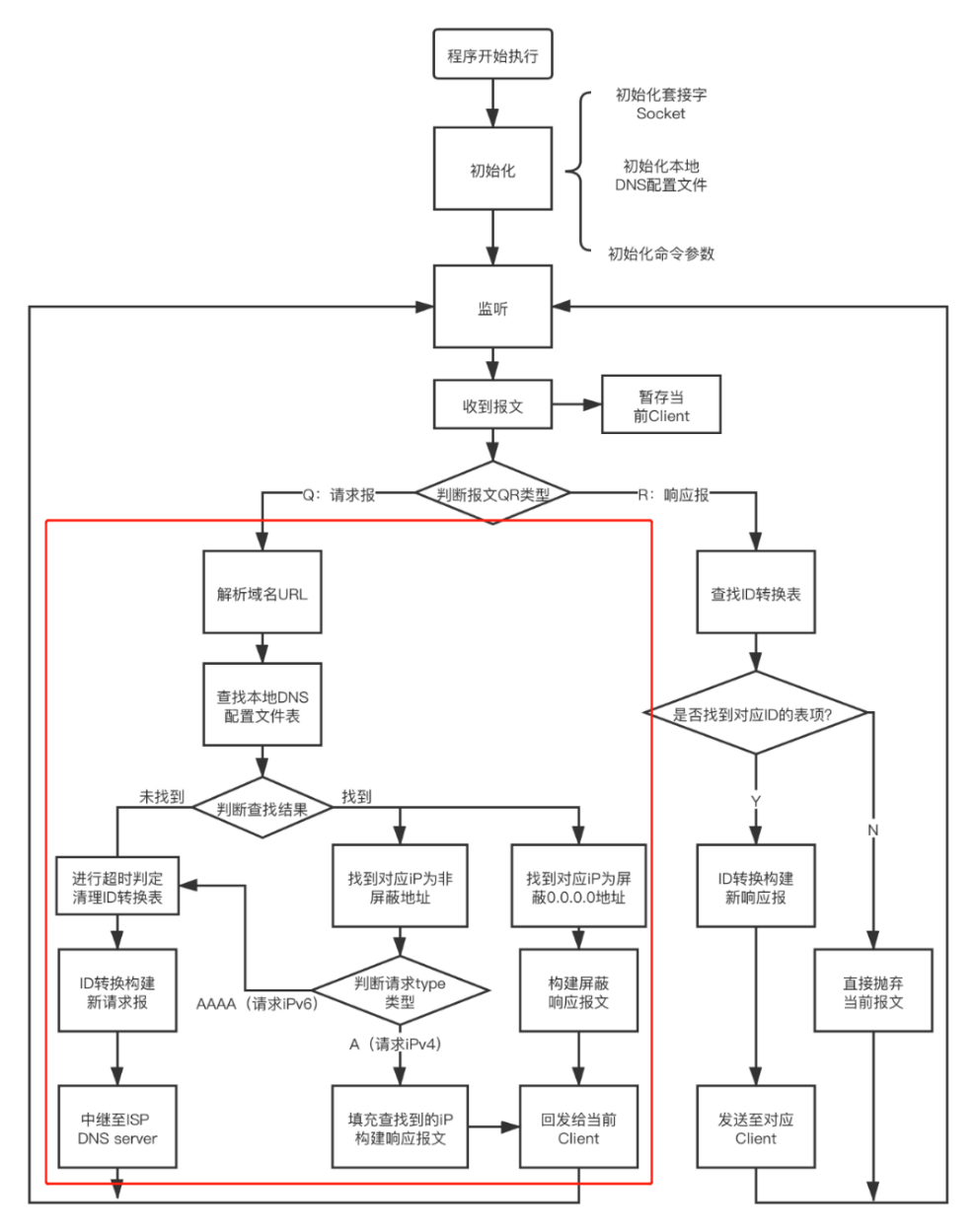
**3.1 功能介绍**

当监控到请求报文时，调用此模块，需要完成域名URL解析，查找本地“域名-IP“对应表，若在表中可以找到对应信息，则发送一个响应报文至用户端。若未找到对应信息，或请求类型非ipv4，并且其域名未被屏蔽，则中继至官方DNS服务器。

若”域名-IP“对应表中可以查找到对应信息，查找到的信息是屏蔽信息，则发送屏蔽响应报文；若对应表中查找到对应信息，查找到的信息不是屏蔽信息，但请求类型是A，则发送响应报文。

若未找到对应信息，或请求类型非ipv4，并且其域名未被屏蔽，则中继至官方DNS服务器。

该模块实现可以分为四部分，其中包含：域名URL解析，该功能通过函数get\_url实现；查找本地“域名-IP“对应表，该功能通过函数local\_find实现；直接回复响应报文和中继转发两种处理方式，该功能在函数deal\_Question中实现。而整个模块是通过函数deal\_Question实现，也就是函数get\_url和函数local\_find均包含在函数deal\_Question中。



**3.2 函数字典**

【函数原型】：

void deal\_Question(char buf[], int length, struct sockaddr\_in temp)

【作用】：完成请求响应模块的实现，需要完成域名URL解析，查找本地“域名-IP“对应表，若在表中可以找到对应信息，则发送一个响应报文至用户端。若未找到对应信息，或请求类型非ipv4，并且其域名未被屏蔽，则中继至官方DNS服务器。

【返回值】：无

【参数】：

【参数名】：buf

【类型】：char字符串

【作用】：记录应用层消息内容

【参数名】：length

【类型】：int

【作用】：记录应用层消息长度

【参数名】：temp

【类型】：struct sockaddr\_in

【作用】：记录客户端信息

**3.3 各功能介绍**

**3.3.1 域名URL解析**

由于应用层中的URL信息与用户使用的格式略有差异，因此需要将网络中传输的格式转化至正常的使用格式，该功能实现通过函数get\_url实现。

其中函数get\_url，不仅实现了从应用层消息中截取URL部分，并完成转换，同时识别了应用层消息的请求类型，并返回，便于进行后续功能的判断。

**3.3.1.1 函数字典**

【函数原型】：int get\_url(char\* buf, char\* url,int length)

【作用】：实现了从应用层消息中截取URL部分，并完成转换，将转换后的url保存在url中，同时识别了应用层消息的请求类型，并返回，便于进行后续功能的判断。

【返回值】：请求消息的请求类型

【参数】：

【参数名】:buf

【参数类型】：char指针

【参数作用】：应用层消息内容

【参数名】:url

【参数类型】：char指针

【参数作用】：解析后的url

【参数名】:length

【参数类型】：int

【参数作用】：应用层消息长度

**3.3.1.2 数据结构及全局变量字典**

重命名：typedef unsigned short U16;

1. **typedef** **struct** DNS\_HDR {  //DNS报文头部
2. U16 ID;         //头部ID，2字节
3. U16 Flags;      //Flags
4. U16 QDCOUNT;    //question计数
5. U16 ANCOUNT;    //answer计数
6. U16 NSCOUNT;    //authority计数
7. U16 ARCOUNT;    //additional计数
8. }DNS\_HDR;

【数据结构名称】：DNS\_HDR

【作用】：DNS报文头部，为了从应用层消息中截取头部，需要剥离头部。

【元素】：

【元素名】：ID

【元素类型】：unsigned short

【元素作用】：头部ID

【元素名】：Flags

【元素类型】：unsigned short

【元素作用】：Flags

【元素名】：question

【元素类型】：unsigned short

【元素作用】：question计数

【元素名】：answer

【元素类型】：unsigned short

【元素作用】：answer计数

【元素名】：authority

【元素类型】：unsigned short

【元素作用】：authority计数

【元素名】：additional

【元素类型】：unsigned short

【元素作用】：additional计数

**3.3.2 查找本地“域名-IP”对照表**

域名解析后，需要判断该域名是否在本地“域名-IP”对照表中，根据对照表一一检查，并返回查找结果，此功能通过函数local\_find实现。

该函数实现，通过参数url进入本地“域名-IP”对照表中的查找，返回查找结果，同时将查找到的ip地址记录到参数ip中。

函数字典：

【函数原型】：int local\_find (char \*url,char \*\*ip)

【作用】： 通过参数url进入本地“域名-IP”对照表中的查找，返回查找结果，同时将查找到的ip地址记录到参数ip中。

【返回值】：是否在本地“域名-IP“对照表中找到对应信息

【参数】：

【参数名】:url

【参数类型】：char指针

【参数作用】：记录待查找的域名

【参数名】:ip

【参数类型】：char字符串指针

【参数作用】：保存查找到的IP地址

**3.3.3 回复响应报文**

当查询到信息后，若该IP地址是0.0.0.0，意味着该地址被屏蔽，因此无论查询类型为何，都需要回复屏蔽响应报文；若该IP地址是正常的非屏蔽地址，并且该查询类型为A时，意味着可以找到查询类型为Ipv4的域名，需要回复正确的响应报文。

通过Wireshark工具分析回复报文，可以得到回复报文是建立在请求报文的基础上的，因此我们需要在请求报文的基础上，额外添加响应报文的内容，包含Name，Type，Class，TTL，data\_length，和回复的ip地址几个字段。

对于屏蔽报文需要将回复的ip地址设置为网络传输格式的0.0.0.0，其他的报文也需要将ip地址进行转换。

补充几个字段后，将回复报文发送至用户。

**3.3.4 中继转发请求报文**

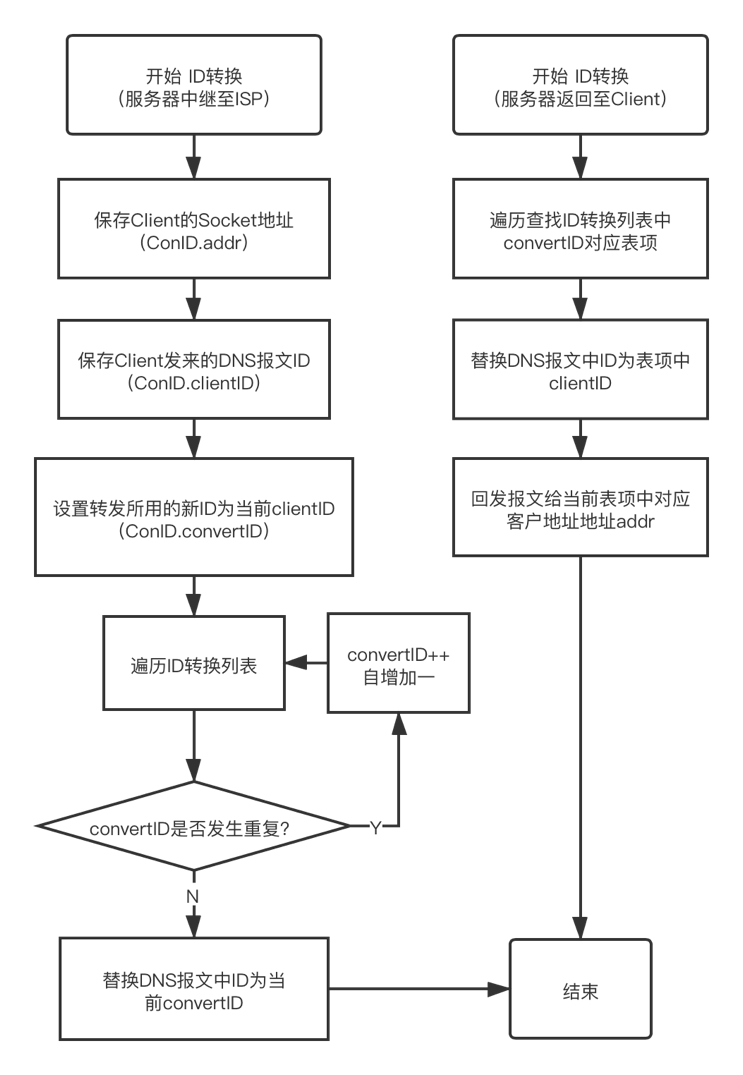
若未找到对应信息，或请求类型非ipv4，并且其域名未被屏蔽，则中继至官方DNS服务器。

由于DNS中继服务器可能需要为多个用户提供中继转发请求报文，而不同用户之间的请求报文id可能一致，直接转发可能出现错误，因此需要设置ID转换，将ID转换为唯一的ID，发送至官方的中继服务器。

同时在ID转换过程中，一旦由于udp协议的不稳定性，使得报文未接收成功，或者官方的DNS服务器失效，需要保证ID转换中保存信息的时效性，避免信息存储过多，造成服务器异常，因此需要清除ID转化过程中的超时信息。该部分通过函数delete\_TTL\_ConID实现。

**3.3.4.1 ID转换（服务器中继至ISP）**

首先保存Client的Socket地址至addr字段，保存Client发来的DNS报文ID至clientID字段，保存建立该链结的时间至starttime字段，用于检测超时。随后设置转发所用的新ID为当前的clientID至convertID，设置过程为：遍历整个ID转换链表，一旦出现相同的convertID，则自加，直到找到不重复的ID。



**3.3.4.2 清除超时信息**

同时在ID转换过程中，一旦由于udp协议的不稳定性，使得报文未接收成功，或者官方的DNS服务器失效，需要保证ID转换中保存信息的时效性，避免信息存储过多，造成服务器异常，因此需要清除ID转化过程中的超时信息。

每当有ID转换时，调用函数delete\_TTL\_ConID清除超时信息，随着ID转换频率的提升，清除超时信息的频率也将伴随提高，因此大大降低了出现超时数据积累的情况。

当调用该函数时，计算当前时间，与各个链结初始化的时间求差，一旦该时间超过设置的TTL时间，则删除该链结。

**函数字典：**

【函数原型】：void delete\_TTL\_ConID()

【作用】：当调用该函数时，计算当前时间，与各个链结初始化的时间求差，一旦该时间超过设置的TTL时间，则删除该链结。

【返回值】：无

【参数】：无

**宏字典**

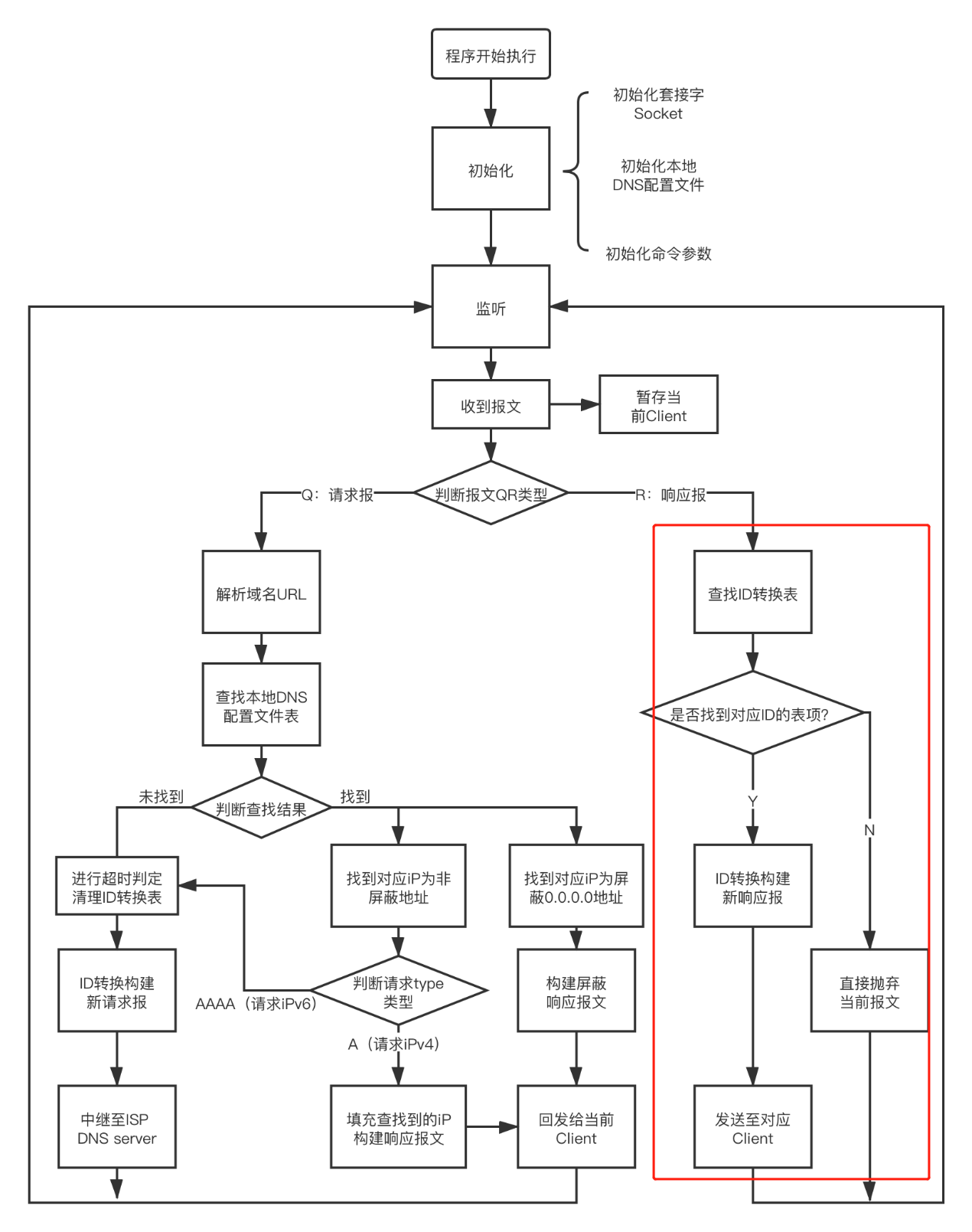
【宏名称】： TTL\_TIME

【作用】：用于设置超时时间，进行超时判断，单位为毫秒。

## 模块四、响应处理模块

**4.1 功能介绍**

该模块主要用于接收到响应报文后的处理，主要过程包含：通过ID转换表，将发送给官方DNS服务器的ID转换至回复给Client端的ID，随后，删除该链结，并将转换后的ID发送给Client。若没有找到对应ID，则说明已经超时，直接删除当前报文。



**函数字典**

【函数原型】：void deal\_Respond(char buf[], int length)

【作用】：完成响应处理模块的功能，通过ID转换表，将发送给官方DNS服务器的ID转换至回复给Client端的ID，随后，删除该链结，并将转换后的ID发送给Client。若没有找到对应ID，则说明已经超时，直接删除当前报文。

【返回值】：无

【参数】：

【参数名称】：buf

【参数类型】：char数组

【参数作用】：应用层消息内容

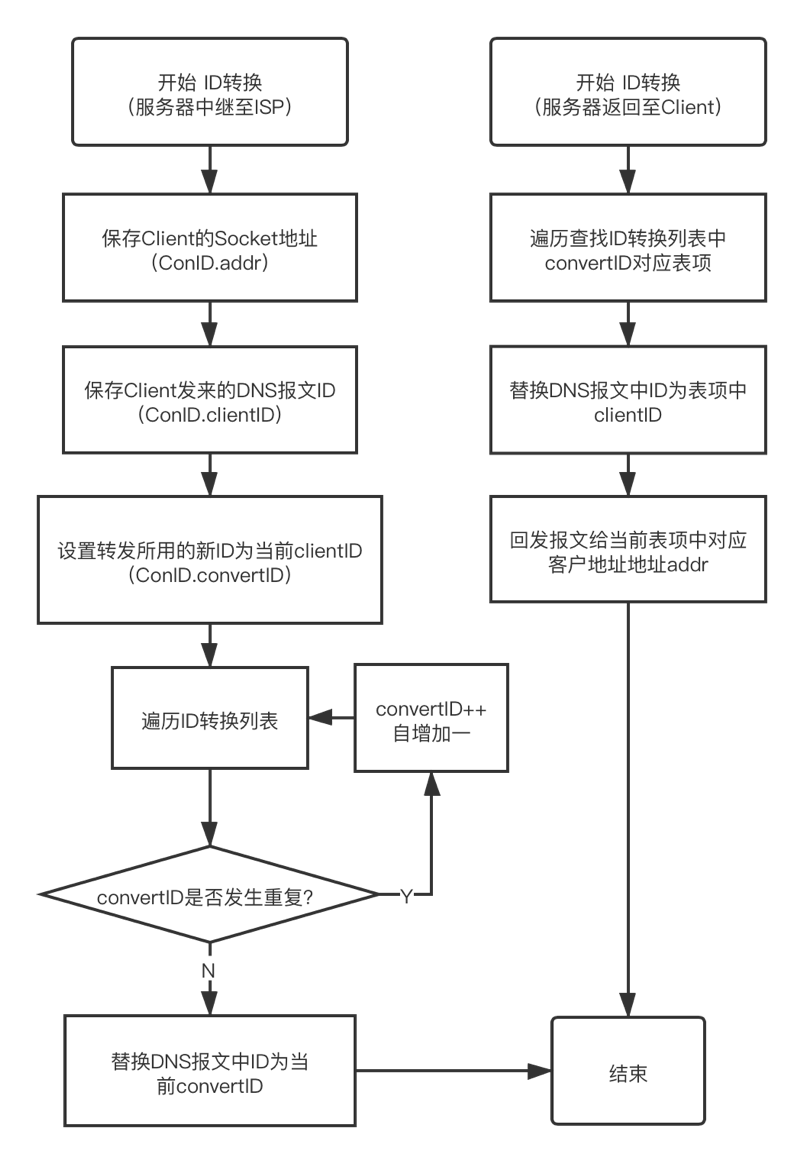
【参数名称】：length

【参数类型】：int

【参数作用】：应用层内容长度

**4.2 ID转换（服务器返回至Client）**

首先遍历查找ID转换链表中convertID对应表项，找到后，替换DNS报文中ID为表项中的clientID，随后回发报文给当前表项中对应客户地址地址addr。



## 模块五 缓存模块

**5.1 功能介绍**

当DNS中继服务器收到来自客户端的请求后，首先检测本地“域名-IP”对照表，一旦本地“域名-IP”对照表中没有请求的域名，开始查找缓存中的“域名-IP”对照表，这部分对照表是当DNS中继服务器启动后，将中继至官方DNS服务器的请求报文中的域名，以及对应的回发至DNS中继服务器的响应报文中的IP，截取出来，进行缓存。当用户在这两个表中都没有找到响应的域名对应，再发送一个请求报文至中继服务器。

以上是完整的DNS中继服务器的工作顺序，对于缓存模块而言，主要是帮助用户更快的获取到IP信息，缓存模块中记录了用户最近访问，且未被文件记录的信息，是通过中继至官方DNS服务器的请求报文中的域名，以及对应的回发至DNS中继服务器的响应报文中的IP组合而成，减少了将报文中继出去以及回收响应报文的工作。

再缓存模块中，使用了LRU算法，可以使缓存区中一直保存使用频率较高的域名IP对应信息，对于不经常使用的域名和IP将进行替换，加快了查询的速度。

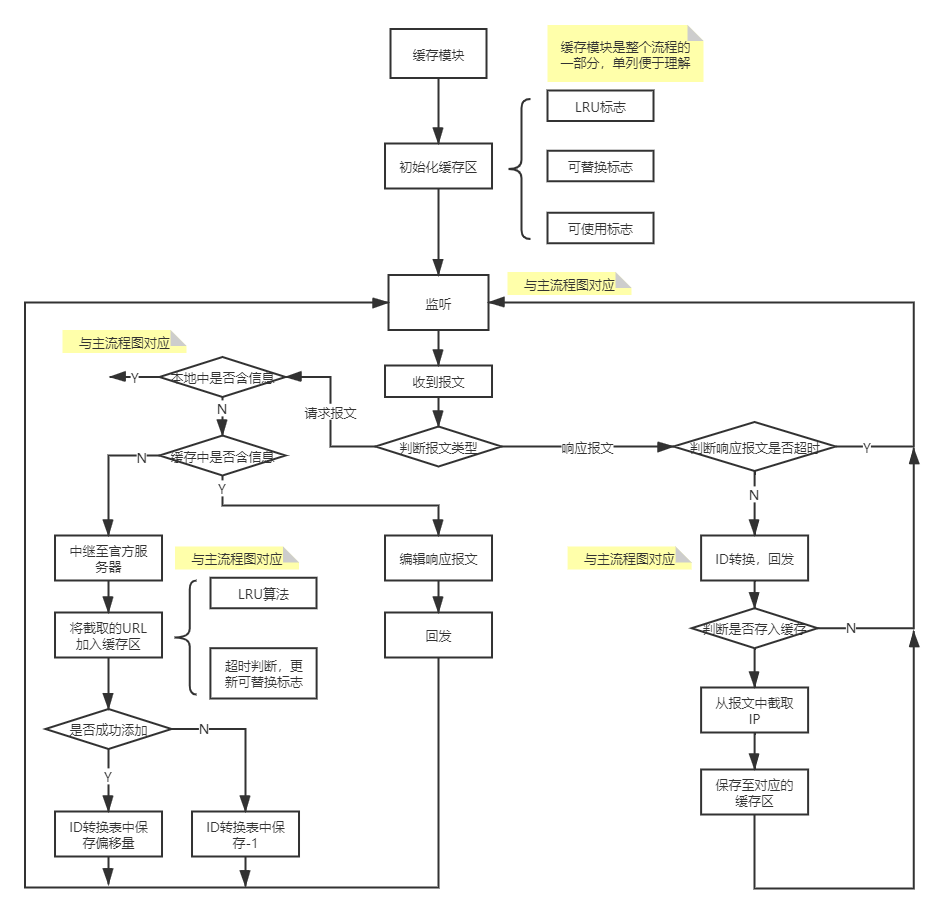
缓存模块中，主要通过以下几个功能实现整个缓存，包括：初始化缓存、添加URL至缓存、截取IP并添加至缓存以及根据缓存信息构造响应报文。

对于初始化缓存，首先初始化每一个缓存位置的LRU信息、该缓存位置是否可被替换、该缓存位置是否可被使用。当刚添加URL进入缓存位置后，该位置不可被替换，否则一旦被替换，将造成错误，只有到该存储位置超时，也就是未能在TTL时间接收到响应报文时，该位置可以被替换，此时ID转换链表也应同步超时，因此不会把超时收到的响应报文再添加进来，当收到IP后此位置可以被替换。该缓存位置是否可以使用是根据域名-IP对应关系是否完整来确定，当IP被添加进去后，该缓存位置可以被使用，中继服务器可以直接从缓存区中取出该数据进行响应报文的编辑，并回发。

对于将URL添加至缓存，首先从中继至官方服务器中的报文中截取URL，其次在缓存区中根据LRU算法选择一个被替换的“域名-IP”对应关系，该对应关系在所有对应关系中被使用的频率最低，将URL添加至缓存，最后再将添加的缓存位置记录到ID转换链表中，作为添加IP时的参照。同时在添加URL中也会修改每一个缓存位置是否超时，若超时，则该缓存位置可以被替换。

对于截取IP至缓存，需要首先将IP从响应报文中截取出来，依次剥离报头，请求报文部分，并在响应部分找到类型为A，也就是Ipv4的IP信息进行读取。随后根据ID转换中的缓存区偏移量将该IP信息保存至缓存区，并更新该缓存位置的状态。

对于根据缓存信息构造响应报文，当中继服务器检测到缓存区中包含需要的“域名-IP”对照信息，则可以通过IP信息直接构造响应报文，进行回发。



**5.2 初始化缓存**

**5.2.1 功能介绍**

初始化缓存区首先初始化每一个缓存位置的LRU信息、该缓存位置是否可被替换、该缓存位置是否可被使用。当刚添加URL进入缓存位置后，该位置不可被替换，否则一旦被替换，将造成错误，只有到该存储位置超时，也就是未能在TTL时间接收到响应报文时，该位置可以被替换，此时ID转换链表也应同步超时，因此不会把超时收到的响应报文再添加进来，当收到IP后此位置可以被替换。该缓存位置是否可以使用是根据域名-IP对应关系是否完整来确定，当IP被添加进去后，该缓存位置可以被使用，中继服务器可以直接从缓存区中取出该数据进行响应报文的编辑，并回发。

该部分在函数local\_Init中。

**5.2.2 数据结构字典**

1. **typedef** **struct** LRU\_Cache {//缓存中继出去的报文,只记录类型是A的，ipv4
2. **char** url[200];
3. **char** ip[20];
4. **clock\_t** starttime;
5. **int** flag;          //初始为0，意味着不能替换，当超时或ip已经被写入，则可以替换
6. **int** lru;              //LRU参数
7. **int** flag\_success;     //可以使用
8. LRU\_Cache;

【名称】：LRU\_Cache

【作用】：每一个结构作为一个缓存位置，构成整个缓存区。

【参数】：

【参数名称】：url

【参数类型】：char数组

【参数作用】：记录域名

【参数名称】：ip

【参数类型】：char数组

【参数作用】：记录IP地址

【参数名称】：starttime

【参数类型】：clock\_t

【参数作用】：记录该缓存位置创建时间

【参数名称】：flag

【参数类型】：int

【参数作用】：表示该位置是否可以被替换，1表示可以被替换，0表示不能被替换，用于LRU算法，当刚添加URL进入缓存位置后，该位置不可被替换，否则一旦被替换，将造成错误，只有到该存储位置超时，也就是未能在TTL时间接收到响应报文时，该位置可以被替换，此时ID转换链表也应同步超时，因此不会把超时收到的响应报文再添加进来，当收到IP后此位置可以被替换。

【参数名称】：lru

【参数类型】：int

【参数作用】：记录该缓存位置的LRU参数，作为lru算法判断替换关系的标志，反映了该缓存位置被使用的频率。

【参数名称】：flag\_success

【参数类型】：int

【参数作用】：反映了该位置是否可以被使用，根据域名-IP对应关系是否完整来确定，当IP被添加进去后，该缓存位置可以被使用，中继服务器可以直接从缓存区中取出该数据进行响应报文的编辑，并回发。

**5.2.3 其他字典**

**宏：**

【名称】：#define CACHE\_NUMBER 100

【作用】：表示缓存区长度

**全局变量：**

【名称】：cache

【类型】：LRU\_Cache数组

【作用】：表示整个缓存区

**5.3 URL添加**

**5.3.1 功能介绍**

将URL添加至缓存，首先从中继至官方服务器中的报文中截取URL，其次在缓存区中根据LRU算法选择一个被替换的“域名-IP”对应关系，该对应关系在所有对应关系中被使用的频率最低，将URL添加至缓存，最后再将添加的缓存位置记录到ID转换链表中，作为添加IP时的参照。同时在添加URL中也会修改每一个缓存位置是否超时，若超时，则该缓存位置可以被替换。

**5.3.2 函数字典**

【函数原型】：

int add\_Cache\_URL(char url[100])

【作用】：将URL添加至缓存，被截取的URL作为参数进入函数，在缓存区中根据LRU算法选择一个被替换的“域名-IP”对应关系，该对应关系在所有对应关系中被使用的频率最低，将URL添加至缓存，最后再将添加的缓存位置记录到ID转换链表中，作为添加IP时的参照。同时在添加URL中也会修改每一个缓存位置是否超时，若超时，则该缓存位置可以被替换。

【返回值】：表示是否成功添加至缓冲区，若成功，则返回1，只有当整个缓冲区都不能被替换时才返回0，若未添加，则ID转换表中记录的时-1，则当DNS中继服务器收到响应报时，不将ip添加至缓存区。

【参数】：

【参数名称】：url

【参数类型】：char数组

【参数作用】：保存域名

**5.4 IP添加**

**5.4.1 功能介绍**

当收到从官方中继服务器发送回来的响应报文，根据ID转换链表中记录的缓存区偏移量，判断是否保存了正确的URL信息，若未保存，则不添加IP，否则需要首先将IP从响应报文中截取出来，依次剥离报头，请求报文部分，并在响应部分找到类型为A，也就是Ipv4的IP信息进行读取。随后根据ID转换中的缓存区偏移量将该IP信息保存至缓存区，并更新该缓存位置的状态。

其中从报文中剥离报头的部分在函数get\_IP\_From\_Buf中实现，将IP添加至缓存区通过函数add\_Cache\_IP实现，函数print\_inetIp可以打印网络格式的IP，用于调试。

**5.4.2 函数字典**

【函数原型】：

int get\_IP\_From\_Buf(char\* buf, int length, char \*ip)

【作用】：将IP从响应报文中截取出来，依次剥离报头，请求报文部分，并在响应部分找到类型为A，也就是Ipv4的IP信息进行读取，保存至参数ip中

【返回值】：返回是否获取成功，若不成功说明该报文未保存Ipv4的数据信息，不保存至缓冲区，做返回成功，则将ip中记录的IP信息保存至缓冲区中。

【参数】：

【参数名称】：buf

【参数类型】：char\*

【参数作用】：保存从官方中继服务器返回的响应报文

【参数名称】：length

【参数类型】：int

【参数作用】：保存从官方中继服务器返回的响应报文长度

【参数名称】：ip

【参数类型】：char\*

【参数作用】：保存截取出来的IP地址

【函数原型】：

void add\_Cache\_IP(char \*ip, int lru\_cache)

【作用】：将剥离出来的IP地址保存到缓存区中偏移量为lru\_cache的缓存位置。

【返回值】：无

【参数】：

【参数名称】：ip

【参数类型】：char\*

【参数作用】：保存剥离出来的IP地址

【参数名称】：lru\_cache

【参数类型】：int

【参数作用】：保存该IP地址需要保存到的缓冲区偏移量。

【函数原型】：void print\_inetIp(char ip[])

【作用】：打印网络类型的IP地址，用于调试信息的输出

【返回值】：无

【参数】：

【参数名称】：ip

【参数类型】：char\*

【参数作用】：保存网络类型的IP地址

**5.5 缓存区读取**

**5.5.1 功能介绍**

当DNS中继服务器收到来自客户端的请求后，首先检测本地“域名-IP”对照表，一旦本地“域名-IP”对照表中没有请求的域名，开始查找缓存中的“域名-IP”对照表，这部分对照表是当DNS中继服务器启动后，将中继至官方DNS服务器的请求报文中的域名，以及对应的回发至DNS中继服务器的响应报文中的IP，截取出来，进行缓存。当用户在这两个表中都没有找到响应的域名对应，再发送一个请求报文至中继服务器。

其中，查找缓存中的“域名-IP”对照表的功能通过该小模块实现，一旦用户请求的报文类型是A，并且可以在缓冲区中找到请求的域名，则将对应存放的IP取出，添加至响应报中进行返回。

其中域名判断时，只能选择可以被使用的缓存位置进行域名判断，因为只有可以被使用的缓存区域才包含正确的IP地址。

查找IP的功能通过函数Cache\_find实现，将IP地址添加至响应报文的功能通过函数deal\_Respond实现，这部分内容在函数deal\_Respond的说明中未包含，编辑响应报文的方式与从本地获取IP并发送响应报文的方式一致，唯一的不同是，本地保存的IP类型是正常的IP类型，而缓冲区中保存的是网络类型的IP，需要一位一位进行数据填充。

**5.5.2 函数字典**

【函数原型】：int Cache\_find(char \*url, char \*\*ip, int type)

【作用】：若该请求类型是A，则根据参数url中保存的URL信息，进入缓存区寻找对应的IP，若找到则将IP保存至ip中，并返回1，若未找到，返回0，表示未找到对应IP，需要中继至中继服务器。

【返回值】：返回是否在缓存区中找到

【参数】：

【参数名称】：url

【参数类型】：char\*

【参数作用】：保存需要查找的URL

【参数名称】：ip

【参数类型】：char\*\*

【参数作用】：指向查找到的保存IP地址的位置

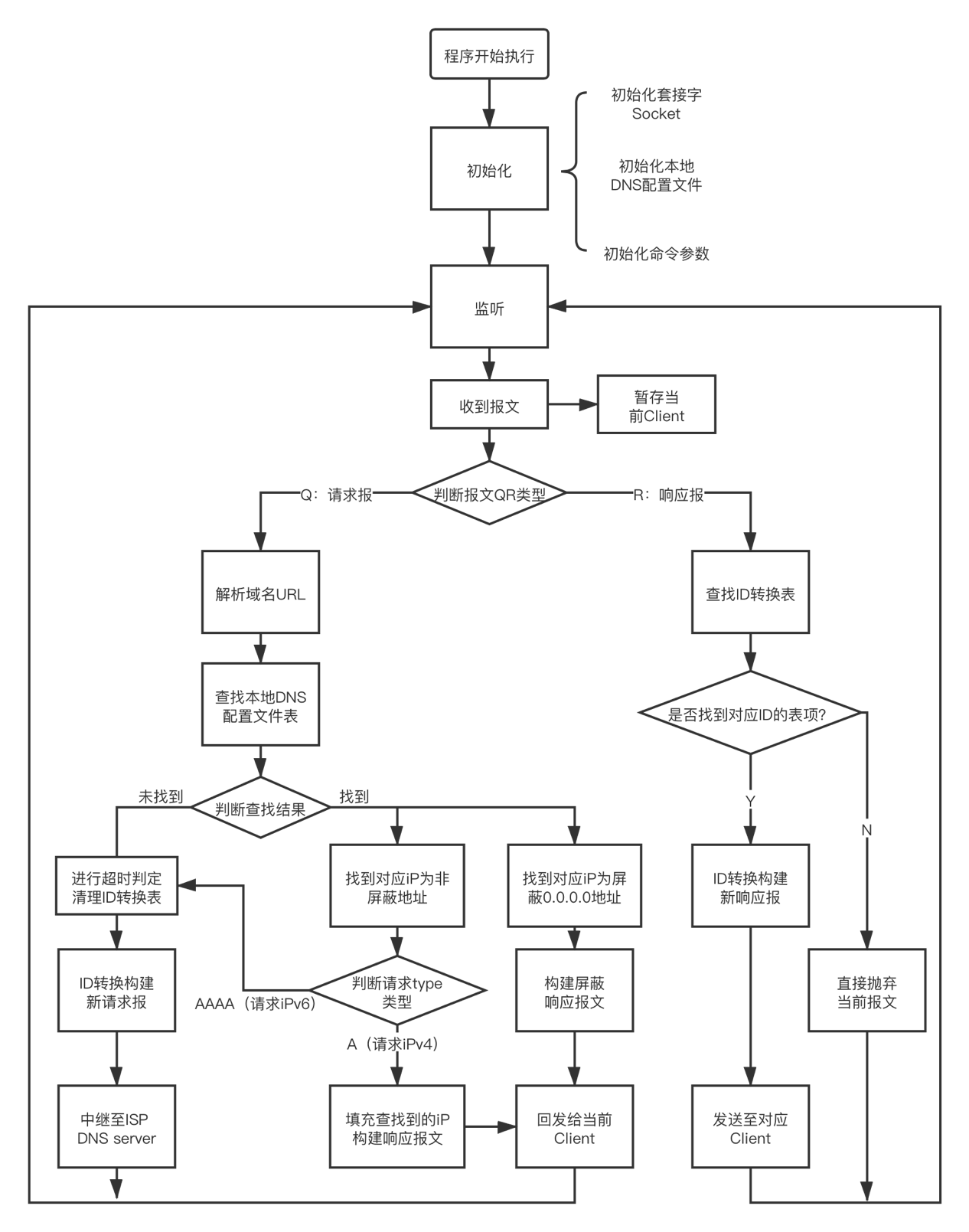
【参数名称】：type

【参数类型】：int

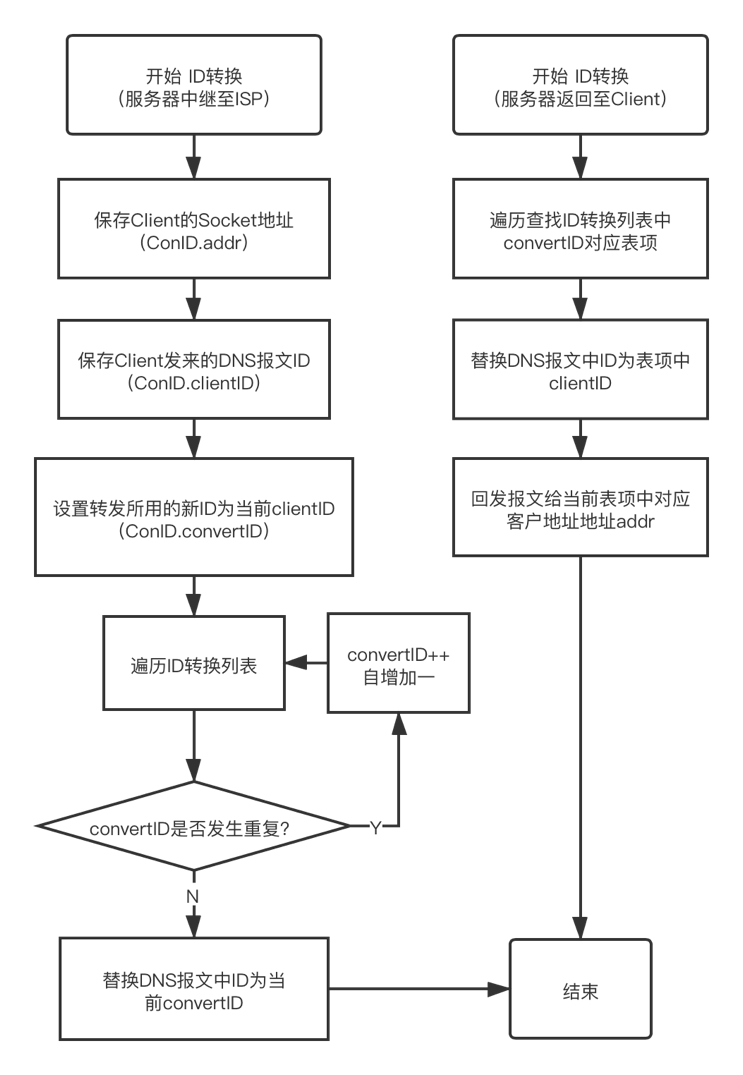
【参数作用】：表示请求类型

# 四、流程图

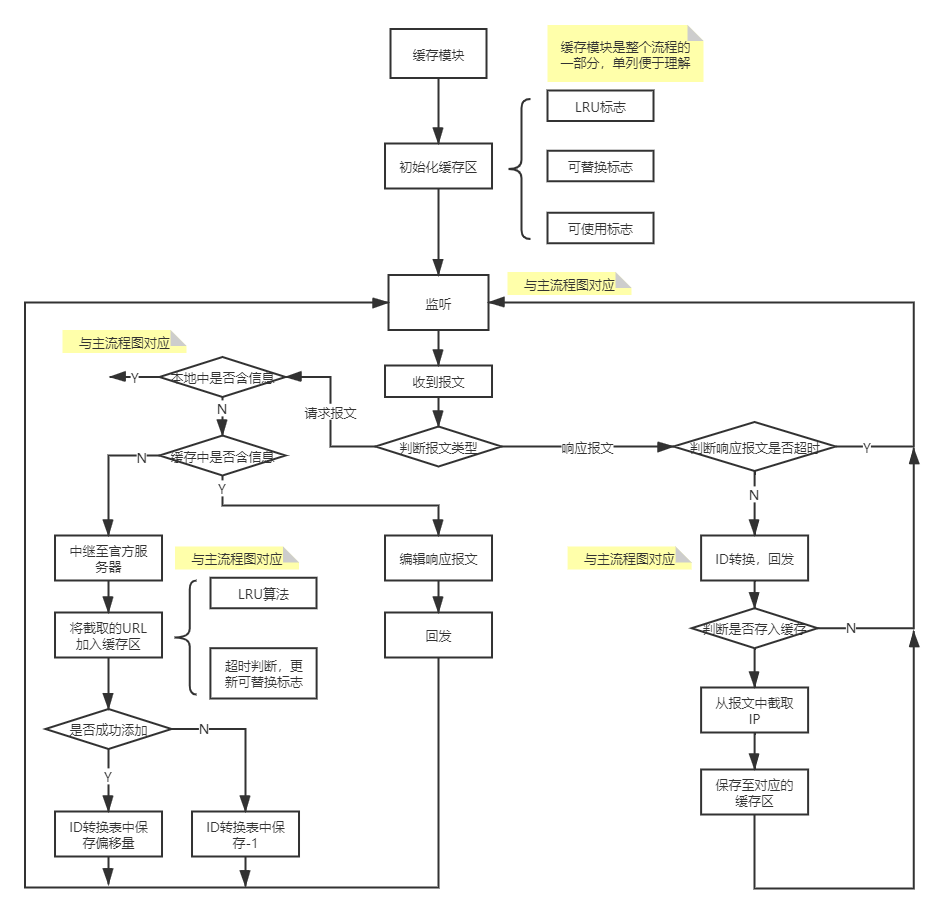
## 1、程序总体流程



## 2、ID转换流程



## 3、LRU-Cache缓存流程



# 五、测试用例与测试结果

**1、本地查找功能测试：**

**·测试内容：**

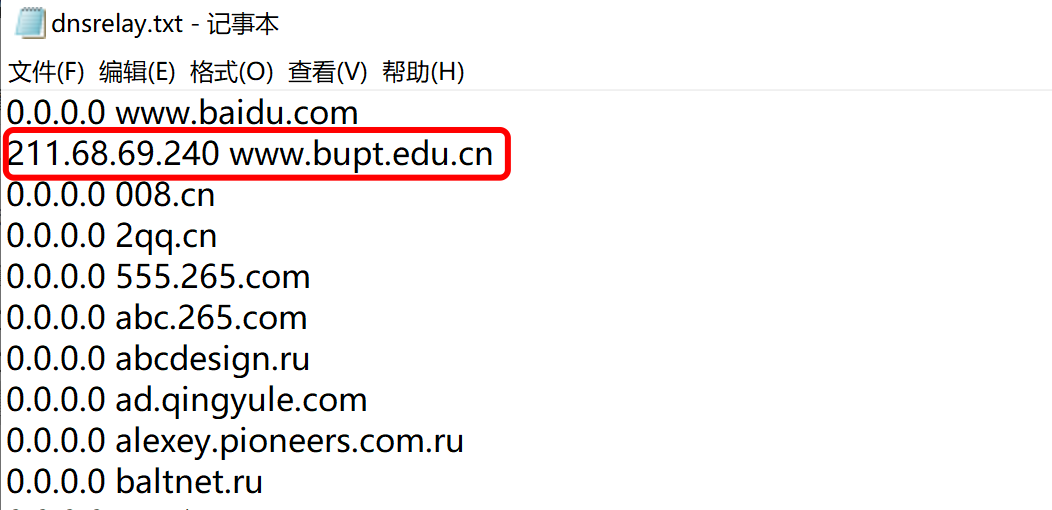
测试利用客户端向服务器端发起域名请求，服务器能否正确解析报文以及域名并在对应配置文件表中查找到相应的ip地址并构建返回。

**·测试方法：**

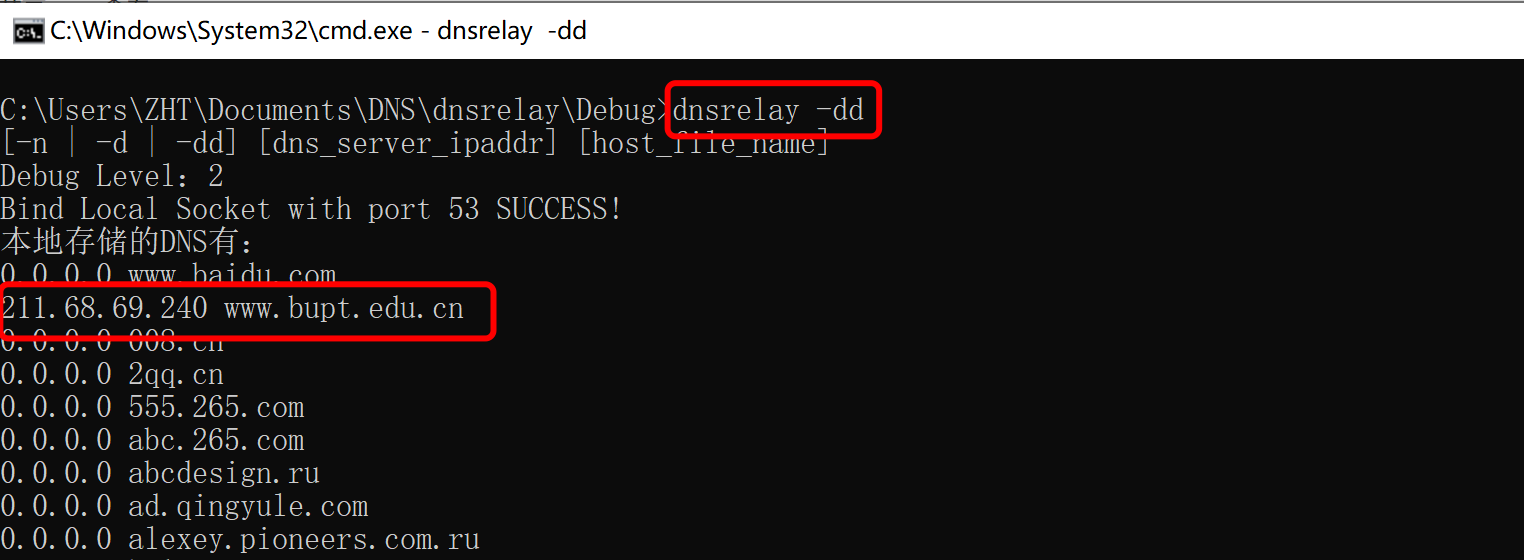
向本地配置文件中写入特定网站的ip地址与域名；使用-dd选项调用程序启动；利用nslookup向本机发起请求，查询刚刚写入的域名；观察是否返回正确的ip地址结果，观察服务器端是否正确处理了相关报文。

**·测试结果：**

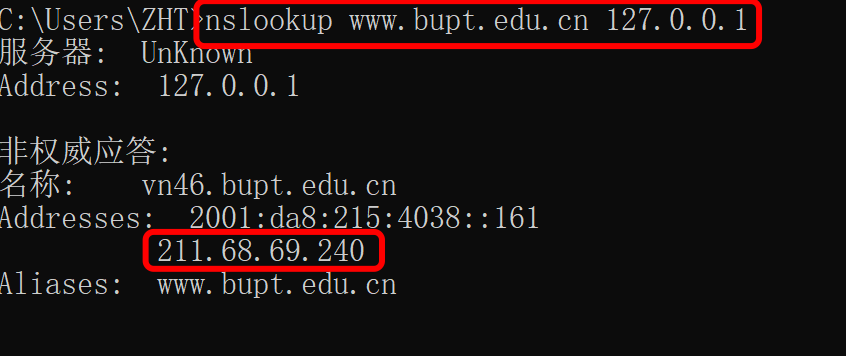
向配置文件中输入北邮官网的ip地址以及域名www.bupt.edu.cn



使用dnsrelay -dd选项启动dns中继服务程序（默认选用233.5.5.5作为外部DNS服务器地址，当前目录下dnsrelay.txt作为配置文件），观察到程序正确加载并读入配置文件。

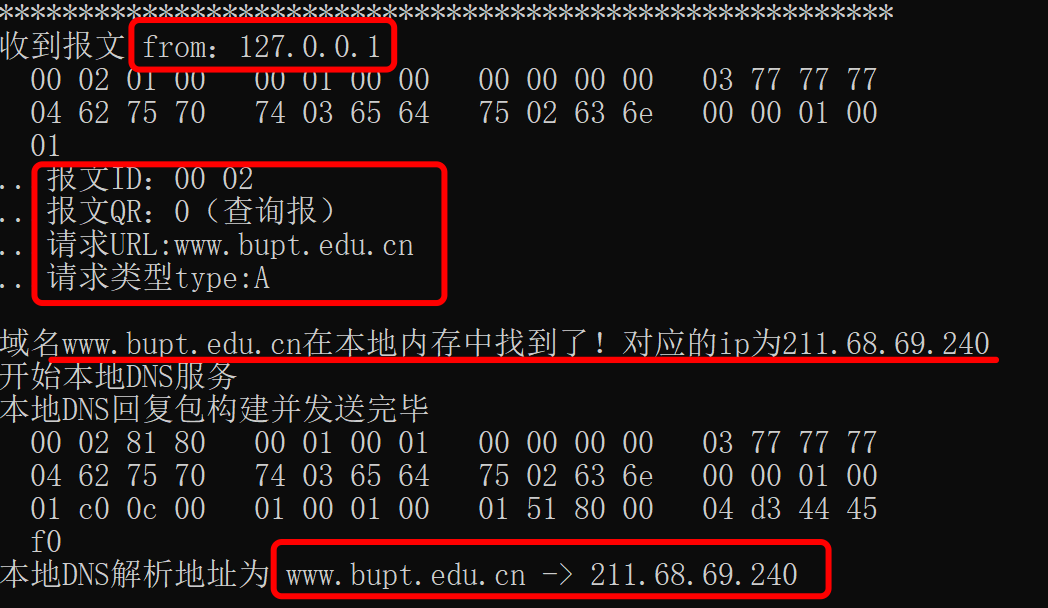


打开新控制台窗口，利用nslookup www.bupt.edu.cn 127.0.0.1指令向127.0.0.1本机发出指令，查询www.bupt.edu.cn对应的ip地址。



可以观察到客户端成功收到了我们刚刚输入的ip地址（此处仅观察iPv4地址的收发情况，忽略iPv6的处理情况）。

再观察服务器端输出情况，



服务器端成功接收到客户端发出的域名请求报文，成功对报文关键字段进行了解析，并在本地配置表中成功查找到域名字段对应的ip信息，成功构建响应报文后返回正确的响应报，最终客户端成功接收响应报显示出如上上图所示结果。

**2、中继服务功能测试：**

**·测试内容：**

测试程序遇到不存在于本地配置文件表中的域名时，能否正确解析报文并且在匹配查找失败的情况下送出中继请求。并且在送出中继请求后返回接收响应报文，并且将响应转发回发起请求的客户主机。

**·测试方法：**

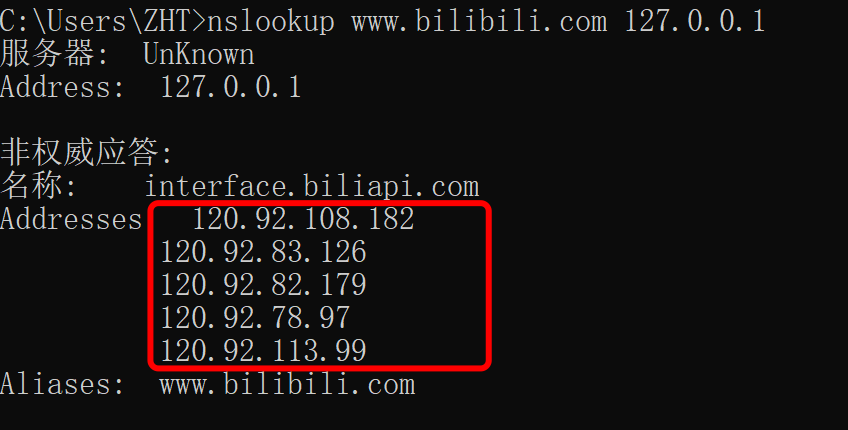
使用-dd选项调用程序启动；利用nslookup指令向服务器端发起域名查询请求，查询内容设置为配置表中不包含的网址；观察是否返回正确的ip地址结果，观察服务器端是否正确处理了相关报文。

**·测试结果：**

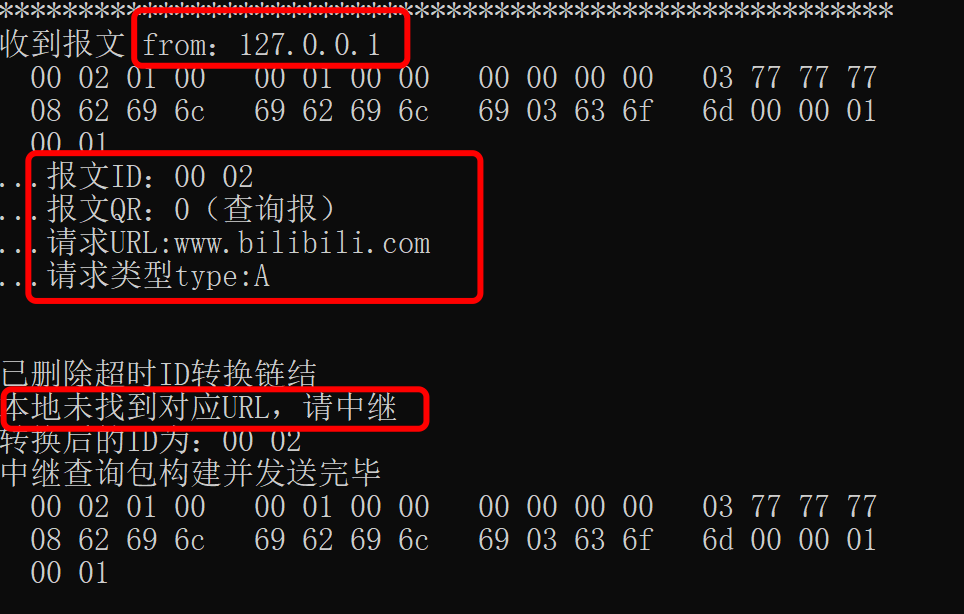
利用nslookup www.bilibili.com 127.0.0.1指令查询哔哩哔哩域名对应的ip地址，



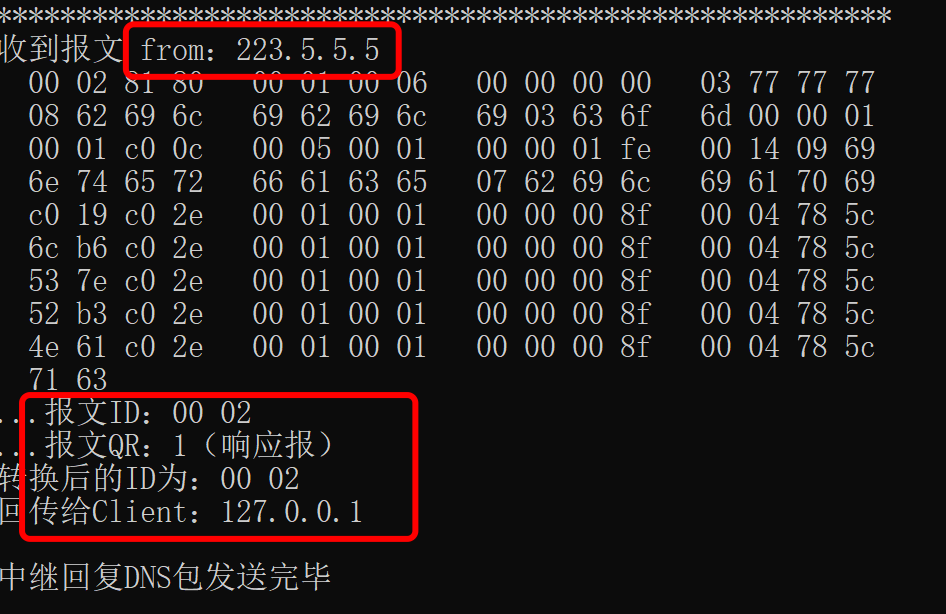
观察到客户端正确返回多个ip地址，



观察服务器端输出信息，



服务器端正确收到客户端发出的域名查询请求并且成功进行了解析，在本地进行查找对应域名失败后进行中继服务，（经ID转换后）构建中继查询报文转发至外部服务器。而后收到外部服务器应答回复如下，



服务器端收到外部回复信息，（经ID转换后）构建回复报文，回传给对应正确Client客户，客户端收到后，最终成功接收响应报显示出如上上图所示结果。

**3、iPv6服务功能测试：**

**·测试内容：**

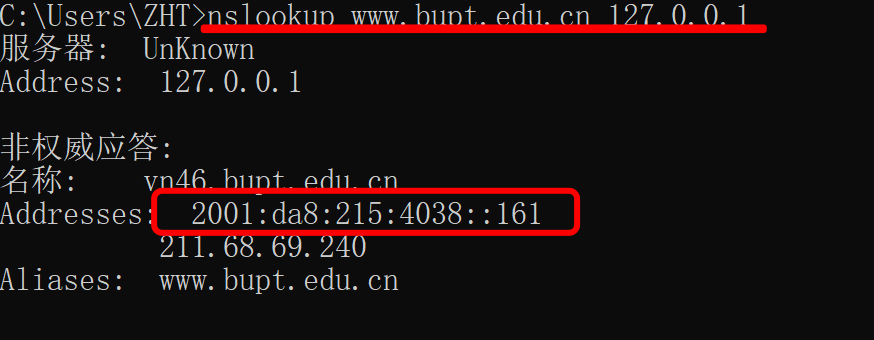
注意利用nslookup指令进行域名查询端过程中客户端会自动向服务器端发送两种类型的请求（A/AAAA）代表iPv4地址请求与iPv6地址请求，在这两种地址请求下我们的中继服务器应当均能做出对应资源类型的反馈（不能均返回iPv4地址）。

**·测试方法：**

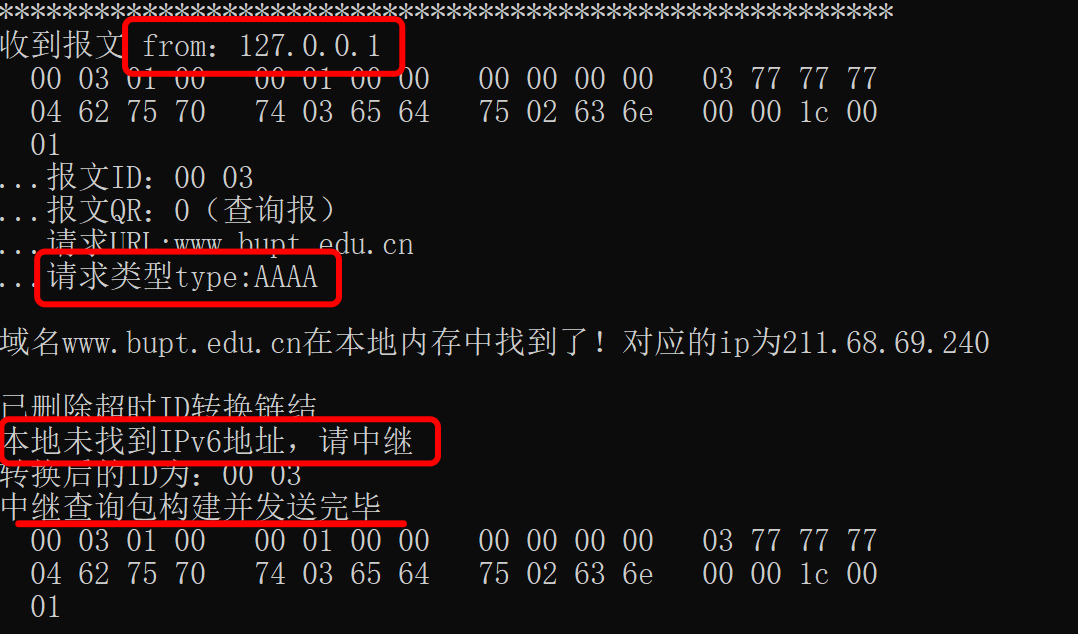
使用-dd选项调用程序启动；利用nslookup向本机发起请求，查询拥有iPv6地址的域名（如www.bupt.edu.cn）；观察是否返回正确解析报文中资源类型type字段内容，观察客户端是否得到了正确的ip地址结果，观察服务器端是否正确处理了相关报文。

**·测试结果：**

在客户端利用nslookup www.bupt.edu.cn 127.0.0.1指令向服务器端发起查询请求如下图所示，



客户端成功收到两种不同资源类型的回复，显示了www.bupt.edu.cn对应的两类地址。再观察服务器端所显示的调试信息如下，



服务器端收到了来自客户端的请求报文后成功对其进行了解析处理，对于AAAA类型的iPv6请求直接向外部服务器进行中继。而后服务器工作过程与中继测试中相同。



收到来自外部服务器回复的响应报文后，转发回原始发出请求的客户端。

**4、屏蔽功能测试：**

**·测试内容：**

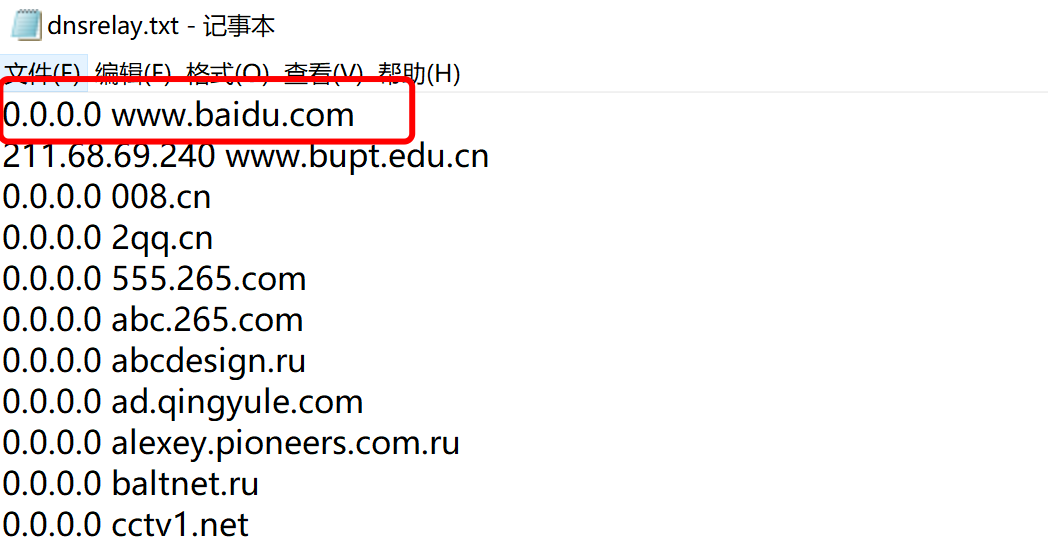
测试当客户端请求某一域名时，能否对特定域名进行屏蔽，使用户无法获取其ip并且无法访问到对应网址。（对于屏蔽应当屏蔽整个域名的访问，若iPv4得到屏蔽而仍返回iPv6地址则没有真正起到屏蔽效果）

**·测试方法：**

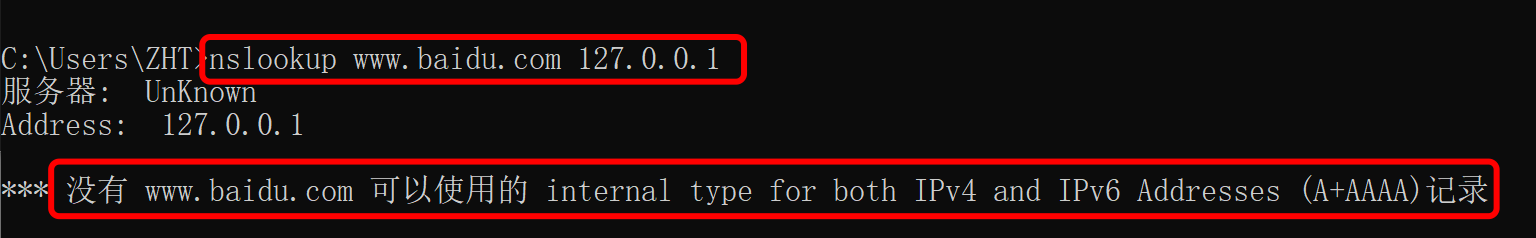
向dnsrelay.txt的配置文件中写入需要进行屏蔽的域名，并且设置该域名对应的ip地址为0.0.0.0（屏蔽）；利用-dd选项启动dns中继服务器服务程序；首先利用nslookup指令向服务器端发起查询请求；再设置本机网络dns地址为127.0.0.1再发起域名访问请求；理论上正确的屏蔽效果应为nslookup无法查询到该网站ip（包括iPv4与iPv6），浏览器无法登录对应网址。

**·测试结果：**

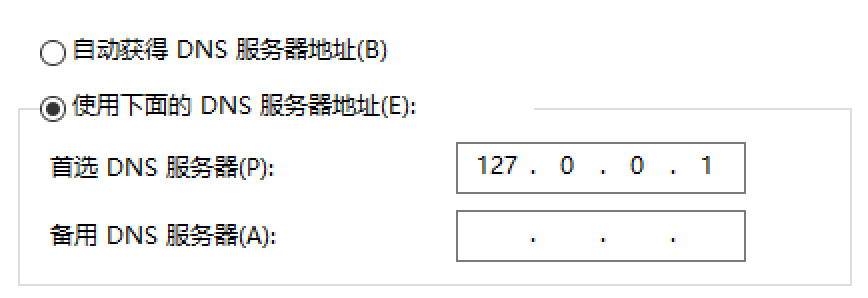
向本地配置文件中写入屏蔽网址www.baidu.com,

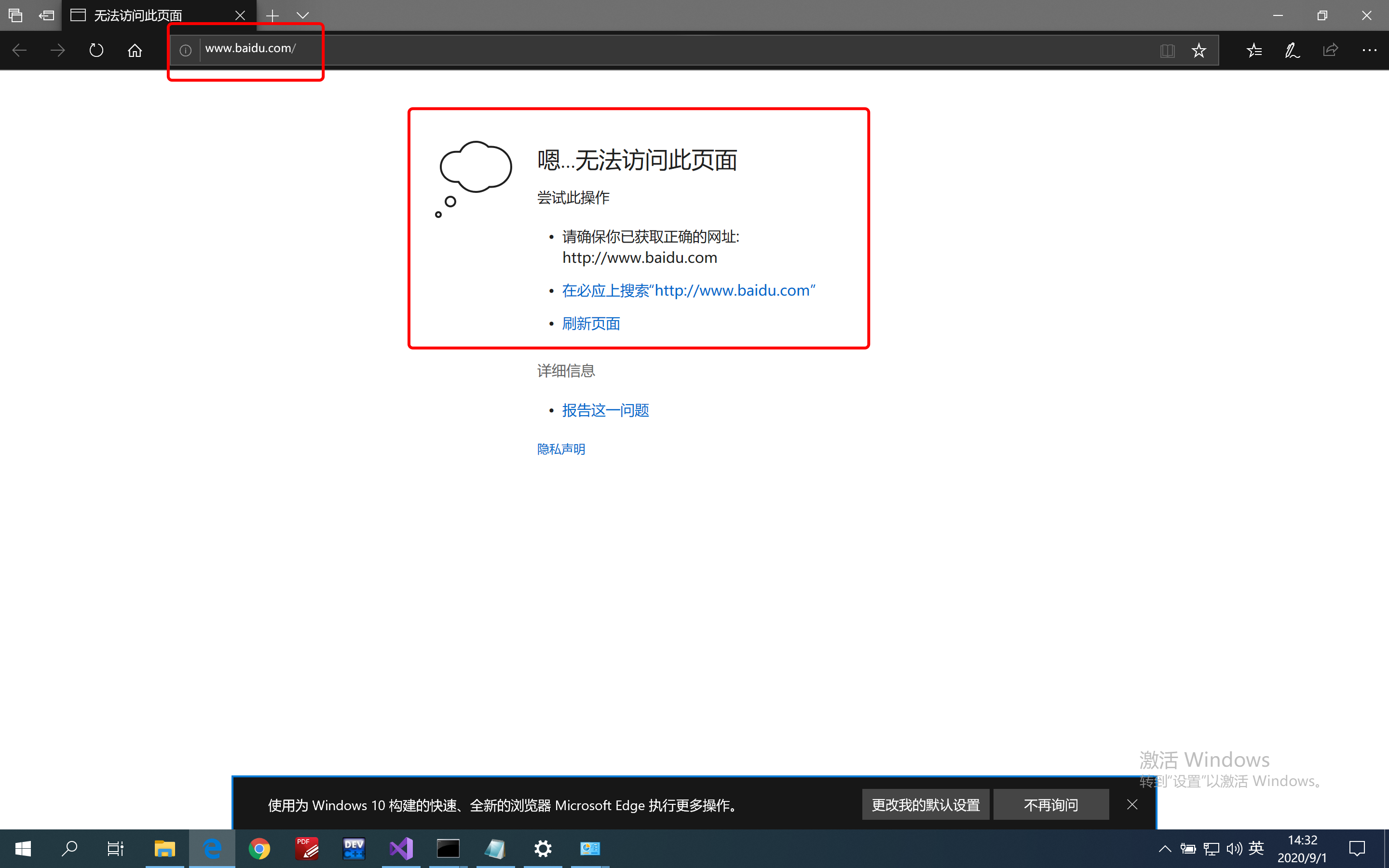


利用-dd选项启动dns中继服务程序，利用nslookup指令查询这条需要被屏蔽的域名，



查询结果显示没有可以使用的ip地址，对于iPv4和iPv6均没有可用的iP地址。再修改本机网络dns服务器地址为127.0.0.1，使用浏览器访问[www.baidu.com](http://www.baidu.com)





可以看到通过浏览器也无法访问该域名，成功对该网址进行了屏蔽。再访问其他网页，可以看到其他网页依旧能正常进行访问浏览。

继续在服务器端对整个过程进行观察，



可以看到服务器收到了来自客户端对于www.baidu.com这一域名的解析请求，在本地表项中查找得到其ip为0.0.0.0为屏蔽地址，因此在调试输出中显示为屏蔽，并且利用0.0.0.0作为ip地址构建响应报文，使客户端只能够得到0.0.0.0作为地址，无法访问。

**5、实际网络访问测试：**

**·测试内容；**

设置网络dns服务器地址为本机，利用浏览器进行实际测试，查看是否能支持浏览器的正确访问，是否能够满足主机正常的网络使用。

**·测试方法：**

在本机上利用-dd选项启动dns服务程序，设置本机网络dns服务地址为127.0.0.1；在本机上正常浏览网页，查看是否能够正常访问。

**·测试结果：**

网络访问正常，机器能够正常通过浏览器以及其他应用程序进行网络访问以及网络使用。

**6、多用户接入测试：**

**·测试内容：**

多客户接入状态下，主要测试ID转换功能，多客户并发功能。

启动dns域名服务程序，将多台机器的dns服务器设置为启动服务程序的主机，其他多台客户向服务器主机发出解析请求。测试在多客户的情况下，dns程序是否能够良好运行。

**·测试方法：**

在一台主机上启动dns服务程序，将其视为服务器主机，利用ipconfig获取其ip地址，将同一局域网内的其他设备的网络dns服务器设置为服务器主机。使用其他设备进行上网，查看服务器主机是否能够良好运行，其他设备并发状况下是否能够良好运行进行网站访问。

**·测试结果：**

每一接入本dns服务器的设备均能够正常进行网络访问，能够正常通过浏览器以及其他应用程序进行网络访问以及网络使用。多客户端并发状态下依旧运行良好，经由ID转换后发出的中继报文ID无重复现象，且在接收后均能实现回转并回发到正确主机。

**7、采用LRU算法的Cache缓存功能测试：**

**·测试内容：**

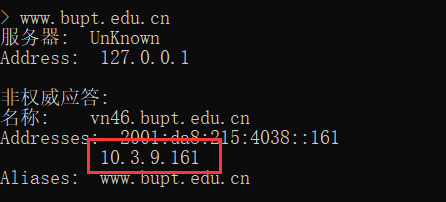
测试在网络对外部进行中继的情况下能否正确缓存返回的ip地址信息；测试再次请求已经进行缓存过ip地址的域名时能否正确调用缓存；服务器能否正确构建返回报文，客户机能否收到正确的响应的报文并且解析出与缓存信息以及上一次用于进行缓存的请求所返回的报文相同的ip地址信息。

**·测试方法：**

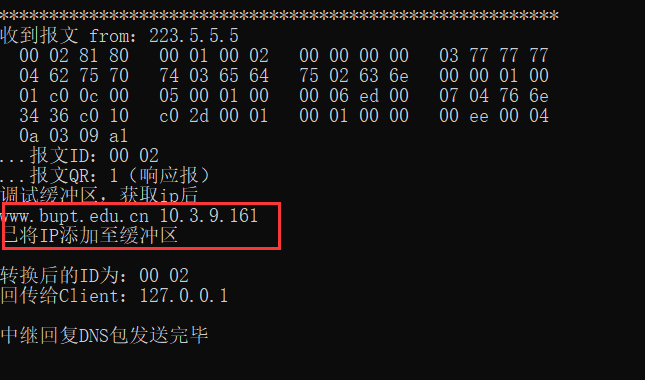
启动dns服务程序；利用nslookup指令向服务器请求本地配置文件中不存在的域名；观察得到的反馈信息，记录首次请求得到的ip地址，观察服务器对相应报文处理时的调试信息输出；再次利用nslookup向服务器请求相同域名；观察服务器对于相应报文处理时的调试信息输出，观察客户端得到的反馈信息是否与第一次请求时获得的ip一致，报文是否成功构建发送并完成解析。

**·测试结果：**

利用nslookup向dns中继服务器请求配置文件中不存在的网址www.bupt.edu.cn

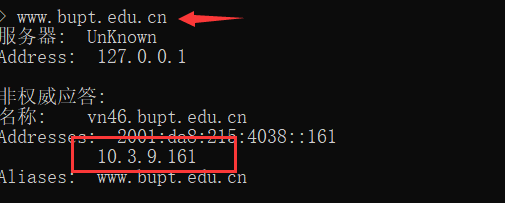


可以看到在客户端返回的ip地址为10.3.9.161；

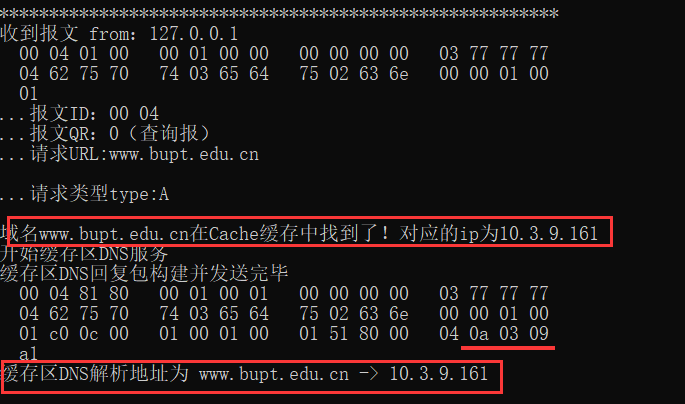


在服务器端可以看到服务器已经成功添加刚刚所请求的域名，及其对应返回的ip到cache缓存当中。

利用客户端nslookup指令再次向服务器请求同一域名www.bupt.edu.cn



客户端获得与先前所请求得到的相同ip地址10.3.9.161；



在服务器端通过输出的调试信息可以看到，本次返回的ip地址没有进行中继。而实通过域名在Cache缓存当中进行查找，从而得到刚刚缓存的ip地址，再于本地构建新的响应报文返回给客户端。

综上所述，关于使用LRU算法的Cache缓存功能测试完毕。

# 六、问题解决

**1、**抓包过程中发现许多响应报文中answer部分的name域存放的并不是与question中类似的方式进行域名的表示。通过查阅资料得知，在这一域中可以通过指针的形式来表示重复出现过的域名，当前两位比特为11时，其后面的内容代表一个指针，其指向的为从头开始向后第n位起的内容。当报文中出现重复的名字需要表示时，仅需要在第一次出现时表示一次，而后出现的名字均可用如上所属的指针方式来进行表示。

**2、**绑定53端口失败，无法绑定到53端口。需要通过setsocketopt进行强制53端口复用，否则可能会出现无法绑定到53端口的问题。

**3、**在命令语法设计过程中需要读入对应的命令参数，但程序仅能在运行过程中实现输入输出。而后通过调用main函数中的参数argc与argv来实现参数的读取，再在程序中对其进行判定分析从而实现期望功能。

**4、**使用nslookup – 127.0.0.1指令时，服务器端程序会接收到一条以.in-addr.arpa为结尾的域名请求，服务器端无法解析并且客户端连续超时。经查阅资料得知此类请求为逆向解析域名，可以直接通过送中继解决问题，在程序加入中继功能后问题解决消失。

**5、**进行中继的过程中，等待ISP服务器返回相应报过程中，如果又接收到一条其他主机发来的请求报，将会无法返回到正确的主机。在逻辑设计上出现了错误，旧版本中仅用单个变量来存储客户机到地址信息，将收到请求，中继请求，收到回复，中继回复这四个过程错误地视为了必然连续的过程。而在实际情况下，收到请求到中继请求和收到回复到中继回复之间是必然连续，而在中继请求和收到回复之间可能又会收到其他请求。

对逻辑进行修改，收到报文后首先对报文的QR字段进行判定，针对请求报与响应报分别进行处理，同时存储报文ID以及其对应的客户主机地址，以便在收到响应报后返回给正确的客户主机。

**6、**当多个主机接入当前中继服务器时，向服务器发送的报文ID可能会出现重复，再接收到回复的响应报后将无法分辨需要返回给哪一具体客户。

由于中继服务器的存在，外部ISP服务器不能够直接响应客户的请求，而将我们的中继服务器作为客户。对于每一台客户机，其报文ID可以唯一对应到发出的请求报文，因此对于我们的服务器也需要保证每一条发出的报文（未得到响应前）都能够有唯一的报文ID与之对应。需要通过ID转换将收到的客户报文ID转换为服务器发出的报文ID，通过结构链表对应存储三项要素：客户地址、客户发来的报文ID、服务器中继发出的报文ID。在发送请求过程中检查报文ID是否重复，进行转换；在接受响应过程中将报文ID更换为原ID并发送到对应客户Client的地址。

**7、**收到的报文类型为字符串，但是在处理数据的过程中无法直接用字符串进行观察判定与处理，因此涉及到对报文当中每一个字段的提取和转化。首先考虑使用逻辑与运算提取特定位的方法实现，后考虑通过强制类型转换直接将字符串拷贝入结构体当中利用结构进行判定。报头解析过程中通过类型定义实现，不要根据协议规定逐个数字节或比特，用C语言的结构体类型定义。

**8、**读入本地的DNS配置文件过程中，在自行编写的文件下可以正常读入，在老师所提供的范例中文件下无法正常读入，且出现字符串跳跃乱码的情况。后找到原因为由于设置固定的静态空间用于存储本地的DNS配置文件信息，因此在进行本地文件读入时不能够读入过长的文件，否则将会出现错误。通过调整扩大了存放域名的静态空间后解决问题。

**9、**在利用字符串直接给结构体赋值的过程中，利用nslookup测试查询过程中，可能无法正确屏蔽某一域名，出现屏蔽iPv4但仍显示iPv6地址的情况。而后发现在处理屏蔽的域名过程中存在问题，原逻辑为先判定资源类型（iPv4/6）而后进行本地解析判断，对于iPv6的报文没有经过本地判断直接送出中继，因此导致iPv6地址正确到达，未起到屏蔽效果。将逻辑改为先本地解析，若是屏蔽则直接返回，若非屏蔽再进行资源类型（iPv4/6）的判定。

**10、**对于超时情况未进行任何处理的问题。首先考虑利用多线程判断，但会消耗占用大量运算资源得不偿失；而后考虑在ID转换表中加入一项TTL作为超时生存时间的标志，每次在monitor监控收到新报文后，检查刷新ID转换表，若ID转换表中某项超时则删除该表项。对于收到延迟超时的报文则无法在ID转换表中查找到，而后直接抛弃。

**11、**在编写缓存功能部分代码时，难以将网络上截取到的报文字符串直接转换为点分十进制形式以字符格式存储，进行输出和再利用inet\_addr()调用。选择使用unsigned char转换后以%u形式输出，每个字符char恰好八位一个字节，中间只需要通过加入”.”的输出即可以点分十进制形式输出调试信息。

但再次匹配到可以利用的缓存时，需要把缓存的ip置入响应报文中构建新的响应报，但是ip字段的长度为32位，现在在缓存中使用char[4]的形式进行存储，考虑到可以直接存储原有的网络字节顺序报文ip，在再次利用是方便直接拷入。首先尝试通过memcpy利用char\*指针直接拷贝整段数据到unsigned long格式对象中，再将unsigned long数据接到报文末尾，但这样的作法失败。依旧为了避免转换过程的繁琐，最终使用将每个八位的char独立连接在报文末尾，连接四次共32位最终尝试可行。由此完成从缓存读出构建报文ip字段的功能任务。

# 七、心得总结

本次课程设计过程中通过成员之间协作探讨，梳理并学习到关于DNS协议的相关内容、充分分析软件实现要求与流程、合理分工进行代码编写与测试。首先，在合作完成形式上由于疫情影响，从传统的线下交流方式改为线上交流方式，这更加考验团队的分工合作意识与交流能力，小组成员的共同努力和团队协作精神是完成本次课程设计的基础。

在计算机网络的理论课程课程当中也有学到过DNS相关的知识，但是对于其中众多的细节仅仅略知一二。在课程设计过程中，通过wireshark抓包分析报文结构、结合RFC1035的协议文档阅读、以及结合课堂中老师所讲述的内容，三者充分结合思考，让我们对于DNS的理解有了新的认识，同时再一次以直观的方式理解了计算机在网络上的工作流程。在对于DNS的学习过程中，我们再次回顾了先前理论课上所讲述的内容，例如在抓包过程中可以看到报文包含的物理层以太网头部，网络层的iPv4头部，传输层的UDP头部，而后是应用层的DNS报文；在socket通信过程中，需要我们的程序实际进行处理的就是应用层的DNS报文部分。正是在这样的新旧知识结合分析，理论实践相互联系的过程中更进一步加深了我们对计算机网络整体知识的梳理把握以及扎实巩固。

在程序设计开发的编码过程中，一方面是遇到了许多问题，另一方面是不断解决问题和知识学习。以socket通信为例，关于socket通信部分的知识在理论课中仅大概了解socket的含义，而深层的实现过程以及代码方面的知识更是一无所有。通过网上资料查阅，理论书本分析，我们通过自学的方式最终解决了关于socket通信有关的知识以及实现手段。这一过程充分锻炼了我们的自主学习能力，丰富了主动解决问题的手段，主动汲取课堂外的关键知识。

由于采用纯C语言进行的设计，因此许多函数以及方法都不能够通过简单的调用库函数实现，均需要自己在编程过程中通过不断的调试与修正，这十分考验我们对于DNS协议及其工作流程的深刻透彻的认识与掌握，同时需要充分考虑到许多代码在各类情况下是否均能够正常执行、底层基础问题考虑是否足够全面。编码完成后还需要利用抓包观察调试的方法测试程序，通过多机器大流量的各类测试来确保程序的稳定与健壮，最终实现整个课程设计。在此过程中，也需要小组成员不断的分工讨论、集思广益，对于许多关键部分代码的实现需要小组成员分析讨论，例如ID转换、超时处理、监听处理逻辑等等。小组成员提出方案、评估方案性能与可行性，最终择优编码实现。

总而言之，在本次课程设计的过程中，一方面是理论联系实际增强了尤其是以DNS协议为主的计算机网络相关知识与实践编码能力；另一方面是通过小组成员之间的交流沟通协作增加了我们的团队意识与实际项目能力。

# 附件

## 1、版本记录

**V0.1**

满足端口绑定、socket通信；能够完成单个Client的本地查询，无法中继；

**V0.2**

加入参数选择功能；

**V0.3**

在简单逻辑下完成单个Client的本地查询以及中继服务；

**V0.4**

在复杂逻辑下完成单个Client的本地查询以及中继服务；改善了输出格式；

**V0.5**

实现对于type字段的辨识，区分iPv4（A）与iPv6（AAAA）请求；

**V0.6**

加入多Client情况下的id转换，在当个Client的测试下运行良好；未进行多个Client环境下的测试；

**V0.7**

简单调整了读文件的规则顺序；但是仍存在无法正确读取范例文件的问题；

**V0.8**

修正了由于存储本地url-ip的数组大小，而引起的无法正确读取范例文件的问题；

**V1.0**

当前版本能够成功完成所有题目的基本要求；可以实现本地查询功能、实现中继服务功能、屏蔽功能，网络正常使用、nslookup正常查询，多客户端并发查询正常；

**V1.1**

增加超时处理功能，能够有效规避外部服务器失效或返回响应报文丢失时，转换表存储过大的问题；转换表中增加超时表项自检删除功能；

**V1.2**

针对iPv4和iPv6的屏蔽逻辑进行更新；当判断出需要屏蔽时对iPv4以及iPv6报文均进行屏蔽处理；

**V1.3**

增加Cache缓存功能；缓存收到过请求的域名URL以及对应的IP地址；通过LRU算法更新cache缓存表；

**V2.0**

验收版本

## 2、源代码

1. #define \_CRT\_SECURE\_NO\_WARNINGS
2. #define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS
4. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Header \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <winsock2.h>
8. #include <windows.h>
9. #include <sys/types.h>
10. #include <string.h>
11. #include <WS2tcpip.h>
12. #include<time.h>
13. #pragma comment(lib,"wsock32.lib")
14. #pragma comment(lib,"ws2\_32.lib")
16. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Macro \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
17. #define BUF\_SIZE 1024
18. #define SRV\_PORT 53
19. #define TTL\_TIME 2000    //超时时间2s
20. #define CACHE\_NUMBER 100 //cache长度
21. **typedef** unsigned **short** U16;
23. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Struct \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
24. **typedef** **struct** DNS\_HDR {    //DNS报文头部
25. U16 ID;         //头部ID，2字节
26. U16 Flags;      //Flags
27. U16 QDCOUNT;    //question计数
28. U16 ANCOUNT;    //answer计数
29. U16 NSCOUNT;    //authority计数
30. U16 ARCOUNT;    //additional计数
31. }DNS\_HDR;
33. **typedef** **struct** DNS\_QUE {    //DNS的Queries部分(查询问题部分)
34. U16 type;       //查询类型
35. U16 classes;    //查询类
36. }DNS\_QUE;
38. **typedef** **struct** ConId {  //对中继转发的报文进行ID转换
39. **struct** sockaddr\_in addr;//保存Client地址
40. **char** clientID[2];       //Client发给本机的ID（允许重复）
41. **char** convertID[2];      //本机中继转发的ID（不允许重复）
42. **clock\_t** starttime;      //转发出去的时间，2秒后超时自动清除该链结
43. **int** LRU\_cache;          //指向LRU中的偏移量
44. **struct** ConId\* nextptr;
45. }ConId;
47. **typedef** **struct** LRU\_Cache {//缓存中继出去的报文,只记录类型是A的，ipv4
48. **char** url[200];
49. **char** ip[20];
50. **clock\_t** starttime;
51. **int** flag;             //初始为0，意味着不能替换，当超时或ip已经被写入，则可以替换
52. **int** lru;              //LRU参数
53. **int** flag\_success;     //可以使用
54. }LRU\_Cache;
56. /\*
57. struct sockaddr\_in {
58. short int sin\_family;       //通信类型
59. unsigned short int sin\_port;//端口
60. struct in\_addr sin\_addr;    //Internet地址
61. unsigned char sin\_zero[8];  //与sockaddr结构的长度相同
62. };
64. struct in\_addr {    //Internet 地址
65. unsigned long s\_addr;
66. };
67. \*/
69. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
70. **char** srv\_ip[] = "223.5.5.5"; //AliDNS\_server
71. **char** host\_file[] = "dnsrelay.txt";
72. **int** flag\_whether\_cache = 0;
73. **char** local\_url[1000][200] = {0};
74. **char** local\_ip[1000][20] = {0};
75. **int** local\_number = 0;
77. WSADATA wsaData;
78. SOCKET local\_sock, outside\_sock;
79. **struct** sockaddr\_in local\_addr, outside\_addr;
80. **struct** sockaddr\_in temp;    //for monitor
81. **struct** sockaddr\_in client,test;
82. **int** len\_addr;
84. **int** dbg\_flag = 0;   //use to flag the Debug information output.
85. //0:no output 1:simple output 2:complex output
87. ConId\* headptr,\* tailptr;   //id转换链表
89. LRU\_Cache cache[CACHE\_NUMBER];      //缓存区

92. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
94. //打印网络类型的ip
95. **void** print\_inetIp(**char** ip[]) {
96. **int** i;
97. **for** (i = 0; i < 3; i++) {
98. printf("%u", (unsigned **char**)ip[i]);
99. printf(".");
100. }
101. printf("%u\n", (unsigned **char**)ip[i]);
102. }
104. //添加url至缓存，lru算法，成功添加返回偏移量，不成功返回-1
105. **int** add\_Cache\_URL(**char** url[100]) {
106. **int** max\_cache;             //LRU最大项
107. **int** max\_LRU;
108. **int** i;
109. **clock\_t** nowtime = clock();
110. max\_cache = 0;
111. max\_LRU = cache[0].lru;
112. //尝试找到最大可替换lru
113. **for** (i = 0; i < CACHE\_NUMBER; i++) {
114. **if** (nowtime - cache[i].starttime > TTL\_TIME) {//若超时可以替换
115. cache[i].flag = 1;
116. }
117. **if** (cache[i].flag == 1) {
118. **if** (cache[i].lru > cache[max\_cache].lru) {
119. max\_cache = i;
120. }
121. }
122. }
124. **if** (cache[max\_cache].flag) {//找到可替换项
125. //替换
127. cache[max\_cache].flag = 0;
128. cache[max\_cache].starttime = clock();
129. cache[max\_cache].lru = 0;
130. cache[max\_cache].flag\_success = 0;
131. strcpy(cache[max\_cache].url, url);
133. //其他所有项目lru自加
134. **for** (i = 0; i < CACHE\_NUMBER; i++) {
135. **if** (i != max\_cache) {
136. cache[i].lru++;
137. }
138. }
139. **return** max\_cache;
140. }
142. //未找到替换项目
143. **else** {
144. printf("未找到可替换项\n");
145. **return** -1;
146. }
148. }
150. //添加ip至下标为lru\_cache的缓存区
151. **void** add\_Cache\_IP(**char** \*ip, **int** lru\_cache) {
152. memcpy(cache[lru\_cache].ip, ip, **sizeof**(ip));
153. cache[lru\_cache].flag\_success = 1;
154. cache[lru\_cache].flag = 1;
155. printf("\n调试缓冲区，获取ip后\n");
156. **for** (**int** i = 0; i < CACHE\_NUMBER; i++) {
157. **if** (cache[i].flag\_success == 1) {
158. printf("%s ", cache[i].url);
159. print\_inetIp(cache[i].ip);
160. }
161. }
162. }
164. //从buf中获取IPv4
165. //成功：返回1
166. //不成功：返回0
167. **int** get\_IP\_From\_Buf(**char**\* buf, **int** length, **char** \*ip) {
168. **int** answer\_number = (unsigned)(buf[6]) \* 16 + (unsigned)buf[7];
169. **int** offset = **sizeof**(DNS\_HDR);
171. **char** url\_number[1024];
172. **char** url[100];//用于跳过name
173. memcpy(url\_number, &(buf[**sizeof**(DNS\_HDR)]), length);//跳过url头，下面保存的是请求报文
174. **int** i = 0, j = 0, k = 0;
175. **while** (url\_number[i] != 0) {
176. **if** (url\_number[i] > 0 && url\_number[i] <= 63) //如果是个计数
177. {
178. **for** (j = url\_number[i], i++; j > 0; j--, i++, k++) //j是计数是几，k是目标位置下标，i是报文里的下标
179. url[k] = url\_number[i];
180. }
182. **if** (url\_number[i] != 0)    //如果没结束就在dest里加个'.'
183. {
184. url[k] = '.';
185. k++;
186. }
187. }
188. url[k] = '\0';

191. i++;
192. i += 4;
193. //此时i中保存answer字段第一项
195. offset += i;
196. //此时offset中保存answer字段第一项


200. **for** (**int** p = 0; p < answer\_number; p++) {
202. memcpy(url\_number, &(buf[offset]), length);//跳过url头以及Queries部分
203. //跳过name
204. i = 1;

207. **if** (url\_number[i + 1] == 0x00 && url\_number[i + 2] == 0x01) {//如果此answer类型是A

210. memcpy(ip, &(buf[offset + i + 11]), 4);//截取

213. **return** 1;
214. }
215. **else** {//不是A
216. **int** add\_length = (16 \* (unsigned)url\_number[i + 9]) + (unsigned)url\_number[i + 10];
217. offset += 12;
218. offset += add\_length;
219. }
220. }
221. **return** 0;
222. }
224. //在cache中匹配URL
225. **int** Cache\_find(**char** \*url, **char** \*\*ip, **int** type) {
226. **if**(type!=1){
227. **return** 0;
228. }
229. **for** (**int** i = 0; i < CACHE\_NUMBER; i++) {
230. **if** (strcmp(url, cache[i].url) == 0 && cache[i].flag\_success == 1) {
231. \*ip = cache[i].ip;
232. **if** (dbg\_flag) {
233. printf("域名%s在Cache缓存中找到了！对应的ip为", url);
234. print\_inetIp(cache[i].ip);
235. }
236. **return** 1;
237. }
238. }
239. **if** (dbg\_flag) {
240. printf("域名%s在Cache缓存中未找到\n", url);
241. }
242. **return** 0;
243. }
245. //清除超时ID转换链表
246. **void** delete\_TTL\_ConID() {
247. ConId\* ptr;
248. ConId\* deleteptr;
249. ConId\* lastptr;
250. lastptr = headptr;
251. ptr = headptr->nextptr;
252. **clock\_t** nowtime = clock();
253. **while** (ptr) {
254. **if** (nowtime - ptr->starttime > TTL\_TIME) {
255. //需要删除该链表
256. deleteptr = ptr;
257. **if** (ptr == tailptr) {//删除的是尾链
258. lastptr->nextptr = NULL;
259. tailptr = lastptr;
260. free(deleteptr);
261. **break**;
262. }
263. **else** {
264. lastptr->nextptr = ptr->nextptr;
265. ptr = ptr->nextptr;
266. free(deleteptr);
267. lastptr = lastptr->nextptr;
268. }
269. }
270. **else** {
271. ptr = ptr->nextptr;
272. lastptr = lastptr->nextptr;
273. }
274. }
275. **if** (dbg\_flag) {
276. printf("已删除超时ID转换链结\n");
277. }
278. }
280. //从报文中截取URL字段 查询类型
281. **int** get\_url(**char**\* buf, **char**\* url,**int** length) {
282. **char** url\_number[100];
283. memcpy(url\_number, &(buf[**sizeof**(DNS\_HDR)]), length);//跳过url头，下面保存的是
284. **int** i = 0, j = 0, k = 0;
286. //报文中的name很有趣，他的方式是www.bupt.edu.cn转二进制后是03 www 04 bupt 03 edu 02 cn
287. **while**( url\_number[i]!=0 ) {
288. **if** (url\_number[i] > 0 && url\_number[i] <= 63) //如果是个计数
289. {
290. **for** (j = url\_number[i], i++; j > 0; j--, i++, k++) //j是计数是几，k是目标位置下标，i是报文里的下标
291. url[k] = url\_number[i];
292. }
294. **if** (url\_number[i] != 0)    //如果没结束就在dest里加个'.'
295. {
296. url[k] = '.';
297. k++;
298. }
299. }
300. url[k] = '\0';
302. //判断请求类型type
303. **if** (url\_number[i + 1] == 0x00 && url\_number[i + 2] == 0x01) {   //type字段0x01代表A类型 请求ipv4地址
304. **return** 1;
305. }
306. **if** (url\_number[i + 1] == 0x00 && url\_number[i + 2] == 0x1c) {   //type字段0x01代表AAAA类型 请求ipv6地址
307. **return** 2;
308. }
309. **return** 0;
310. }
312. //初始化：本地dns存储文件、ID转换链表、缓存表
313. **void** local\_Init() {
315. //初始化本地dns存储文件
316. **FILE**\* fp;
317. **if** ((fp = fopen(host\_file, "r")) == NULL) {
318. printf("host error.\n");
319. exit(-1);
320. }
321. **int** i = 0;
322. **while** (!feof(fp)) {
323. fscanf(fp, "%s %s", local\_ip[i], local\_url[i]);
324. i++;
325. }
326. local\_number = i;
327. **if** (dbg\_flag) {
328. printf("本地存储的DNS有：\n");
329. **for** (i = 0; i < local\_number; i++) {
330. printf("%s %s\n", local\_ip[i], local\_url[i]);
331. }
332. }
334. //初始化ID转换链表
335. headptr = (ConId\*)malloc(**sizeof**(ConId));
336. headptr->nextptr = NULL;
337. tailptr = headptr;
339. **if** (flag\_whether\_cache) {
340. //初始化缓存表，每个项目都可以被替换
341. **for** (i = 0; i < CACHE\_NUMBER; i++) {
342. cache[i].flag = 1;
343. cache[i].lru = 0;
344. cache[i].flag\_success = 0;
345. }
346. }
348. }
350. //在本地DNS文件中匹配URL
351. **int** local\_find(**char** \*url,**char** \*\*ip) {
352. **for** (**int** i = 0; i < local\_number; i++) {
353. **if** (strcmp(url, local\_url[i]) == 0) {
354. \*ip = local\_ip[i];
355. **if** (dbg\_flag) {
356. printf("域名%s在本地内存中找到了！对应的ip为%s\n", url, \*ip);
357. }
359. **return** 1;
360. }
361. }
362. **return** 0;
363. }
365. //处理查询报
366. **void** deal\_Question(**char** buf[], **int** length, **struct** sockaddr\_in temp) {
367. **char** url[100];
368. **int** i = 0, type = 0;
369. unsigned **char** tage;
371. //在报文中截取URL字段以及查询类型type
372. type = get\_url(buf, url, length);
373. **if** (dbg\_flag) {
374. printf("\n...请求URL:%s\n", url);
375. **if** (type == 1) {
376. printf("\n...请求类型type:A\n\n");
377. }
378. **else** **if**(type == 2) {
379. printf("\n...请求类型type:AAAA\n\n");
380. }
381. }
383. //在本地文件中匹配对应URL
384. **char** \*ip;
385. **int** flag = local\_find(url, &ip);//flag标识是否在本地匹配成功（1：成功 2：失败）
387. //成功：本地存储中查询到对应IP，并且该ip被屏蔽
388. //或者：本地存储中查询到对应IP，该ip未被屏蔽，且请求类型是ipv4
389. **if** ((flag == 1 && strcmp(ip, "0.0.0.0") == 0) || (flag == 1 && strcmp(ip, "0.0.0.0") != 0 && type==1)) {
390. **if** (dbg\_flag) {
391. printf("开始本地DNS服务\n");
392. }
393. **char** send\_buf[BUF\_SIZE];//构建一个response报文
394. unsigned **short** flag = htons(0x8180);//flag
395. memcpy(&send\_buf, buf, length);
396. memcpy(&send\_buf[2], &flag, **sizeof**(unsigned **short**));
397. unsigned **short** answer = htons(0x0001);
398. **if** (strcmp(ip, "0.0.0.0") == 0) {//屏蔽
399. **if** (dbg\_flag) {
400. printf("屏蔽\n");
401. }
402. answer = htons(0x0000);
403. }
404. **int** offset = 0;
405. memcpy(&send\_buf[6], &answer, **sizeof**(unsigned **short**));
406. **char** A\_Answer[16];//A类型的response报文Answer报文是16个字节
408. unsigned **short** Name = htons(0xc00c);//抓包得到的域名指针
409. memcpy(&A\_Answer, &Name, **sizeof**(Name));
411. unsigned **short** Type = htons(0x0001);
412. offset += **sizeof**(unsigned **short**);
413. memcpy(A\_Answer + offset, &Type, **sizeof**(unsigned **short**));
415. unsigned **short** Class = htons(0x0001);
416. offset += **sizeof**(unsigned **short**);
417. memcpy(A\_Answer + offset, &Class, **sizeof**(unsigned **short**));
419. unsigned **long** TTL = htonl(0x00015180);//24个小时
420. offset += **sizeof**(unsigned **short**);
421. memcpy(A\_Answer + offset, &TTL, **sizeof**(unsigned **long**));
423. unsigned **short** Data\_length = htons(0x0004);
424. offset += **sizeof**(unsigned **long**);
425. memcpy(A\_Answer + offset, &Data\_length, **sizeof**(unsigned **short**));
427. unsigned **long** Address = (unsigned **long**)inet\_addr(ip);
428. offset += **sizeof**(unsigned **short**);
429. memcpy(A\_Answer + offset, &Address, **sizeof**(unsigned **long**));
431. offset += length;
432. offset += **sizeof**(unsigned **long**);
433. memcpy(send\_buf + length, A\_Answer, **sizeof**(A\_Answer));
434. length = sendto(local\_sock, send\_buf, offset, 0, (**struct** sockaddr \*)&temp, **sizeof**(temp));
436. //反馈
437. **if** (length < 0) {
438. printf("send error\n");
439. **return**;
440. }
441. **if** (dbg\_flag) {
442. printf("本地DNS回复包构建并发送完毕\n");
443. }
444. **if** (dbg\_flag == 2) {
445. printf("  ");
446. **for** (i = 0; i < length; i++) {
447. tage = (unsigned **char**)send\_buf[i];
448. printf("%02x ", tage);
449. **if** (!((i + 1) % 4)) {
450. printf("  ");
451. }
452. **if** (!((i + 1) % 16)) {
453. printf("\n  ");
454. }
455. }
456. }
458. **char**\* p;
459. p = send\_buf + length - 4;
460. **if** (dbg\_flag) {
461. printf("\n本地DNS解析地址为 %s -> %u.%u.%u.%u\n\n\n", url, (unsigned **char**)\*p, (unsigned **char**)\*(p + 1), (unsigned **char**)\*(p + 2), (unsigned **char**)\*(p + 3));
462. }
463. }
465. **else** **if** (flag\_whether\_cache && Cache\_find(url, &ip, type) && type==1) {
466. **if** (dbg\_flag) {
467. printf("开始缓存区DNS服务\n");
468. }
469. **char** send\_buf[BUF\_SIZE];//构建一个response报文
470. unsigned **short** flag = htons(0x8180);//flag
471. memcpy(&send\_buf, buf, length);
472. memcpy(&send\_buf[2], &flag, **sizeof**(unsigned **short**));
473. unsigned **short** answer = htons(0x0001);
475. **int** offset = 0;
476. memcpy(&send\_buf[6], &answer, **sizeof**(unsigned **short**));
477. **char** A\_Answer[16];//A类型的response报文Answer报文是16个字节
479. unsigned **short** Name = htons(0xc00c);//抓包得到的域名指针
480. memcpy(&A\_Answer, &Name, **sizeof**(Name));
482. unsigned **short** Type = htons(0x0001);
483. offset += **sizeof**(unsigned **short**);
484. memcpy(A\_Answer + offset, &Type, **sizeof**(unsigned **short**));
486. unsigned **short** Class = htons(0x0001);
487. offset += **sizeof**(unsigned **short**);
488. memcpy(A\_Answer + offset, &Class, **sizeof**(unsigned **short**));
490. unsigned **long** TTL = htonl(0x00015180);//24个小时
491. offset += **sizeof**(unsigned **short**);
492. memcpy(A\_Answer + offset, &TTL, **sizeof**(unsigned **long**));
494. unsigned **short** Data\_length = htons(0x0004);
495. offset += **sizeof**(unsigned **long**);
496. memcpy(A\_Answer + offset, &Data\_length, **sizeof**(unsigned **short**));
498. offset += **sizeof**(unsigned **short**);
499. memcpy(A\_Answer + offset, ip, **sizeof**(unsigned **char**));
500. ip++;
501. offset += **sizeof**(unsigned **char**);
502. memcpy(A\_Answer + offset, ip, **sizeof**(unsigned **char**));
503. ip++;
504. offset += **sizeof**(unsigned **char**);
505. memcpy(A\_Answer + offset, ip, **sizeof**(unsigned **char**));
506. ip++;
507. offset += **sizeof**(unsigned **char**);
508. memcpy(A\_Answer + offset, ip, **sizeof**(unsigned **char**));
510. offset += length;
511. offset += **sizeof**(unsigned **char**);
512. memcpy(send\_buf + length, A\_Answer, **sizeof**(A\_Answer));
514. length = sendto(local\_sock, send\_buf, offset, 0, (**struct** sockaddr \*)&temp, **sizeof**(temp));
516. //反馈
517. **if** (length < 0) {
518. printf("send error\n");
519. **return**;
520. }
521. **if** (dbg\_flag) {
522. printf("缓存区DNS回复包构建并发送完毕\n");
523. }
524. **if** (dbg\_flag == 2) {
525. printf("  ");
526. **for** (i = 0; i < length; i++) {
527. tage = (unsigned **char**)send\_buf[i];
528. printf("%02x ", tage);
529. **if** (!((i + 1) % 4)) {
530. printf("  ");
531. }
532. **if** (!((i + 1) % 16)) {
533. printf("\n  ");
534. }
535. }
536. }
538. **char**\* p;
539. p = send\_buf + length - 4;
540. **if** (dbg\_flag) {
541. printf("\n缓存区DNS解析地址为 %s -> %u.%u.%u.%u\n\n\n", url, (unsigned **char**)\*p, (unsigned **char**)\*(p + 1), (unsigned **char**)\*(p + 2), (unsigned **char**)\*(p + 3));
542. }
543. }
545. //失败：本地存储中未查询到对应
546. //或者：查找到对应ip，且未屏蔽，请求类型是ipv6的
547. **else**{
548. //删除ID转换链表中的超时项
549. delete\_TTL\_ConID();

552. //若是ipv4，添加url至缓存cache
553. //成功：返回偏移量，之后添加至ConID中
554. //失败：返回-1，不添加，ConID中偏移量设为-1
555. **int** cache\_flag = -1;
556. **if** (type == 1) {
557. cache\_flag = add\_Cache\_URL(url);
558. }

561. **if** (dbg\_flag) {
562. **if** (type == 1) {
563. printf("本地未找到对应URL，请中继\n");
564. }
565. **else** **if** (type == 2) {
566. printf("本地未找到IPv6地址，请中继\n");
567. }
568. }
570. //ID转换：处理报文ID，进行ID转换
571. //尾插 保存client地址到addr,保存clientID；
572. //查询本地表中是否与已有convertID发生重复
573. //重复：增加id偏移值；
574. //不重复：修改buf中ID字段为convertID = clientid + id偏移;
575. ConId\* ptr;
576. ConId\* newptr;
577. **int** find\_flag = 1;
579. newptr = (ConId\*)malloc(**sizeof**(ConId));
580. tailptr->nextptr = newptr;
581. tailptr = newptr;
583. newptr->addr = temp;
584. newptr->starttime = clock();//转发时间
586. **if** (flag\_whether\_cache) {//启动缓存
587. //若已经缓存，则保存偏移量
588. **if** (cache\_flag >= 0 && type == 1) {
589. newptr->LRU\_cache = cache\_flag;
590. **if** (dbg\_flag) {
591. printf("成功：已将URL添加至LRU缓存区\n");
592. }
593. }
594. //若未缓存，则保存偏移量-1
595. **else** {
596. newptr->LRU\_cache = -1;
597. **if** (dbg\_flag) {
598. **if** (type == 1) {
599. printf("失败：未将URL添加至LRU缓存区\n");
600. }
601. **else** {
602. printf("非ipv4：不保存至缓存区\n");
603. }
604. }
605. }
606. }
608. newptr->nextptr = NULL;
610. newptr->clientID[0] = buf[0];
611. newptr->clientID[1] = buf[1];
613. i = -1;
614. **while** (find\_flag) {
615. i++;
616. find\_flag = 0;
617. ptr = headptr->nextptr;
618. **while** (ptr) {
619. **if** (ptr->convertID[0] == buf[0]+i && ptr->convertID[1] == buf[1]+i) {
620. find\_flag = 1;
621. **break**;
622. }
623. ptr = ptr->nextptr;
624. }
625. }
626. newptr->convertID[0] = buf[0] + i;
627. newptr->convertID[1] = buf[1] + i;
629. buf[0] += i;
630. buf[1] += i;
632. **if** (dbg\_flag) {
633. printf("转换后的ID为：%02x %02x\n", (unsigned **char**)buf[0], (unsigned **char**)buf[1]);
634. }
636. //转发报文到指定DNS服务器
637. length = sendto(local\_sock, buf, length, 0, (**struct** sockaddr \*)&outside\_addr, **sizeof**(outside\_addr));



642. //反馈
643. **if** (length < 0) {
644. printf("send to server error\n");
645. **return**;
646. }
647. **if** (dbg\_flag) {
648. printf("中继查询包构建并发送完毕\n");
649. }
650. **if** (dbg\_flag == 2) {
651. printf("  ");
652. **for** (i = 0; i < length; i++) {
653. tage = (unsigned **char**)buf[i];
654. printf("%02x ", tage);
655. **if** (!((i + 1) % 4)) {
656. printf("  ");
657. }
658. **if** (!((i + 1) % 16)) {
659. printf("\n  ");
660. }
661. }
662. }
664. //保存temp到client以便回送响应报
665. //client = temp;//ID转换实现后将不再需要
666. }
667. }
669. //处理响应报
670. **void** deal\_Respond(**char** buf[], **int** length) {
671. **int** i = 0;
673. //处理报文ID，进行ID转换
674. //查询本地表中convertID进行匹配
675. //一定匹配成功，将该报文buf的id修改为对应clientID;   地址变量client赋值为对应client地址;
676. //从表中删除该表项;
677. ConId\* ptr;
678. ConId\* lastptr;
679. lastptr = headptr;
680. ptr = headptr->nextptr;
681. **while** (ptr) {
682. **if** (buf[0] == ptr->convertID[0] && buf[1] == ptr->convertID[1]) {
683. //找到
684. buf[0] = ptr->clientID[0];
685. buf[1] = ptr->clientID[1];
686. client = ptr->addr;
687. //删除
688. lastptr->nextptr = ptr->nextptr;
689. **if** (ptr == tailptr) {
690. tailptr = lastptr;
691. }
693. **char** ip[20];
695. **if** (flag\_whether\_cache) {
696. **if** (ptr->LRU\_cache >= 0) {
697. **if** (get\_IP\_From\_Buf(buf, length, ip)) {//成功获取ipv4地址
698. //添加ip至缓存区
699. add\_Cache\_IP(ip, ptr->LRU\_cache);
700. **if** (dbg\_flag) {
701. printf("已将IP添加至缓冲区\n");
702. }
703. }
704. }
705. }

708. free(ptr);
709. //退出
710. **break**;
711. }
712. ptr = ptr->nextptr;
713. lastptr = lastptr->nextptr;
714. }
716. **if** (ptr) {
717. **if** (dbg\_flag) {
718. printf("\n转换后的ID为：%02x %02x\n", (unsigned **char**)buf[0], (unsigned **char**)buf[1]);
719. }

722. //回发响应报到Client
723. length = sendto(local\_sock, buf, length, 0, (**struct** sockaddr \*)&client, **sizeof**(client));
725. //反馈
726. **if** (length < 0) {
727. printf("send back error\n");
728. }
729. **if** (dbg\_flag) {
730. printf("回传给Client：%s\n", inet\_ntoa(client.sin\_addr));
731. printf("\n中继回复DNS包发送完毕\n");
732. }
733. }
734. **else** {
735. **if** (dbg\_flag) {
736. printf("该回复已超时，不进行回发\n");
737. }
738. }
740. }

743. //监控抓包
744. **void** monitor() {
745. **int** length = -1;
746. **char** buf[BUF\_SIZE];
747. memset(&buf, 0, BUF\_SIZE);
749. //进行监控接收
750. length = recvfrom(local\_sock, buf, BUF\_SIZE, 0, (**struct** sockaddr\*) &temp, &len\_addr);
752. //监控收到包
753. **if** (length > 0) {
754. unsigned **char** tage;
755. **int** i = 0;
757. //打印报文内容
758. **if** (dbg\_flag) {
759. printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n收到报文 ");
760. printf("from：%s\n", inet\_ntoa(temp.sin\_addr));
761. }
762. **if** (dbg\_flag == 2) {
763. printf("  ");
764. **for** (i = 0; i < length; i++)
765. {
766. tage = (unsigned **char**)buf[i];
767. printf("%02x ", tage);
768. **if** (!((i + 1) % 4)) {
769. printf("  ");
770. }
771. **if** (!((i + 1) % 16)) {
772. printf("\n  ");
773. }
774. }
775. }
777. //打印报文ID
778. **if** (dbg\_flag == 2) {
779. printf("\n...报文ID：");
780. **for** (i = 0; i < 2; i++)
781. {
782. tage = (unsigned **char**)buf[i];
783. **if** (dbg\_flag == 2) {
784. printf("%02x ", tage);
785. }
786. }
787. }

790. //判定报文QR
791. **if** ((unsigned **char**)buf[2] & 0x80) { //QR位为1：响应报
792. **if** (dbg\_flag) {
793. printf("\n...报文QR：1（响应报）");
794. }
795. //处理响应报
796. deal\_Respond(buf, length);
797. }
798. **else** {                              //QR位为0：查询报
799. **if** (dbg\_flag) {
800. printf("\n...报文QR：0（查询报）");
801. }
802. //处理查询报
803. deal\_Question(buf, length, temp);
804. }
805. }
806. }
808. //处理参数
809. **void** deal\_parameters(**int** argc, **char**\* argv[]) {
810. **int** i;
811. //printf("%d\n", argc);
812. printf("[-n | -d | -dd] [-c] [dns\_server\_ipaddr] [host\_file\_name]\n");
813. **if** (argc == 1) {
814. **return**;
815. }
816. **for** (i = 1; i != argc; i++) {
817. **if** (i == 1) {
818. //printf("%s\n", argv[i]);
819. **if** (!strcmp(argv[i], "-n")) {
820. dbg\_flag = 0;
821. printf("Debug Level：%d\n", dbg\_flag);
822. }
823. **else** **if** (!strcmp(argv[i], "-d")) {
824. dbg\_flag = 1;
825. printf("Debug Level：%d\n", dbg\_flag);
826. }
827. **else** **if** (!strcmp(argv[i], "-dd")) {
828. dbg\_flag = 2;
829. printf("Debug Level：%d\n", dbg\_flag);
830. }
831. **else** {
832. printf("ERROR\n");
833. }
834. }
835. **else** **if** (i == 2) {
836. **if** (!strcmp(argv[i], "-c")) {
837. flag\_whether\_cache = 1;
838. printf("服务器已启动存储区\n");
839. }
840. **else** **if** (!strcmp(argv[i], "-nc")) {
841. printf("服务器未启动存储区\n");
842. }
843. **else** {
844. printf("ERROR\n");
845. }
846. }
847. **else** **if** (i == 3) {
848. //printf("%s\n", argv[i]);
849. memcpy(srv\_ip, argv[i], strlen(argv[i]));
850. printf("已设置指定DNS服务器为：%s\n", srv\_ip);
851. }
852. **else** **if** (i == 4) {
853. //printf("%s\n", argv[i]);
854. memcpy(host\_file, argv[i], strlen(argv[i]));
855. printf("已设置指定本地配置文件为：%s\n", host\_file);
856. }
857. **else** {
858. **return**;
859. }
860. }
861. **return**;
862. }
864. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
865. **int** main(**int** argc, **char**\* argv[])
866. {
867. deal\_parameters(argc, argv);
869. WSAStartup(MAKEWORD(2, 2), &wsaData);
870. local\_sock = socket(AF\_INET, SOCK\_DGRAM, 0);
871. outside\_sock = socket(AF\_INET, SOCK\_DGRAM, 0);
873. len\_addr = **sizeof**(**struct** sockaddr\_in);
875. **if** (local\_sock < 0 || outside\_sock < 0) {
876. printf("create socket error");
877. **return** -1;
878. }
879. local\_addr.sin\_family = AF\_INET;
880. local\_addr.sin\_port = htons(SRV\_PORT);
881. local\_addr.sin\_addr.S\_un.S\_addr = INADDR\_ANY;
883. outside\_addr.sin\_family = AF\_INET;
884. outside\_addr.sin\_port = htons(SRV\_PORT);
885. outside\_addr.sin\_addr.S\_un.S\_addr = inet\_addr(srv\_ip);
887. **int** const\_one = 1;
888. ioctlsocket(outside\_sock, FIONBIO, (u\_long FAR\*) & const\_one);  //将外部套接口设置为非阻塞
889. ioctlsocket(local\_sock, FIONBIO, (u\_long FAR\*) & const\_one);    //将本地套接口设置为非阻塞
891. setsockopt(local\_sock, SOL\_SOCKET, SO\_REUSEADDR, (**const** **char**\*)&const\_one, **sizeof**(const\_one));//53端口强制复用
892. **if** (bind(local\_sock, (**struct** sockaddr\*) & local\_addr, **sizeof**(local\_addr)) < 0) {//绑定local\_sock套接字到53端口
893. printf("bind error");
894. }
895. **else** {
896. printf("Bind Local Socket with port 53 SUCCESS!\n");
897. }
899. local\_Init();
900. **while** (1) {
901. monitor();
902. }
903. }