



**计算机网络课程设计**

**DNS中继服务器**

2019年7月

班级：2016211310

学号：2016211181

组员：裴 家 亮

**目录**

[一、 系统功能设计 3](#_Toc13999244)

[1. 基本任务与要求 3](#_Toc13999245)

[2. 整体流程图 3](#_Toc13999246)

[3. 问题考虑 4](#_Toc13999247)

[二、 软件设计与模块划分 4](#_Toc13999248)

[1. 技术选型与环境依赖 4](#_Toc13999249)

[2. 模块划分 5](#_Toc13999250)

[三、 软件流程图 6](#_Toc13999251)

[1. 主流程图 6](#_Toc13999252)

[2. 子流程图 7](#_Toc13999253)

[A. 获取请求:dns\_server\_listener 8](#_Toc13999254)

[B. 响应回发:dns\_server\_sender 8](#_Toc13999255)

[C. 远程查询:dns\_server\_helper 8](#_Toc13999256)

[四、 软件测试 9](#_Toc13999257)

[1. 测试用例设计 9](#_Toc13999258)

[2. 测试结果分析 9](#_Toc13999259)

[A. 功能测试 9](#_Toc13999260)

[B. 性能测试 11](#_Toc13999261)

# 系统功能设计

## 基本任务与要求

设计一个DNS中继服务器程序，读入“域名-IP地址”对照表，当客户端查询域名对应的IP地址时，用域名检索该对照表，需要实现三种功能，即对于以下三种检索结果进行不同处理：

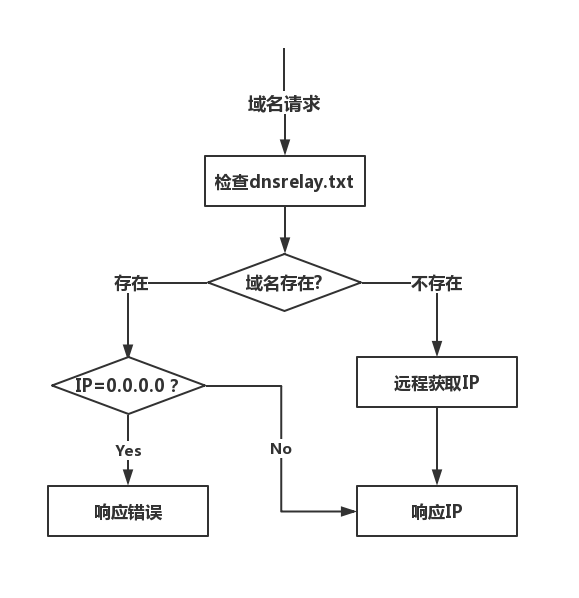
* **不良网站拦截功能**：检索结果为IP地址0.0.0.0，则向客户端返回“域名不存在”的报错信息；
* **服务器功能**：检索结果为普通IP地址，则向客户返回这个地址；
* **中继器功能**：表中未检索到该域名，则向因特网DNS服务器发出查询，并将结果返回给客户端；

设计程序时需要考虑以下两个问题：

* **多客户端并发**：允许多个客户端（可能会位于不同计算机）的并发查询，即：允许第一个查询尚未得到答案前就启动处理另外一个客户端查询请求；
* **超时处理**：由于UDP的不可靠性，考虑求助外部DNS服务器时却不能得到应答或者收到迟到应答的情形如何处理。

## 整体流程图

对于基本任务提到的三个功能，可以分为向表/向远程服务器获取IP两种方案，其主要流程如图一所示，具体实现参见后文。



**图 1** 整体流程图

## 问题考虑

针对高并发、超时处理两个问题，本次采取了以下解决方案：

* 多客户端并发：
  + 采用UDP无连接传输协议，较易支持服务端与多个客户端通信；
  + 解耦功能模块，采用多线程/多进程技术并发执行，由于Python GIL全局锁的原因，多线程效率比较低，本次采用多进程与管道通信技术；
* 超时处理：
  + 设置超时时间，降低远程查询对服务性能的影响；
  + 记录发送包ID，持续监听远程回复，避免迟到回复/永久不回复的问题；

# 软件设计与模块划分

## 技术选型与环境依赖

本次DNS服务器开发技术主要是基于struct工具库的字节流序列化/反序列化的socket通信，DNS协议封装参考RFC1034及RFC1035，相关说明如下：

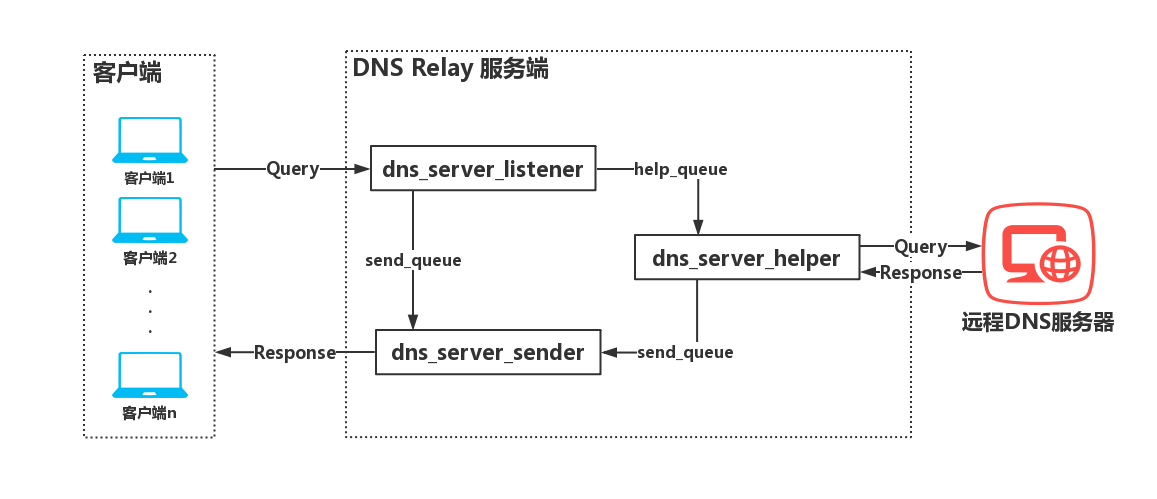
|  |  |
| --- | --- |
| **操作系统** | Windows10 家庭版 |
| **开发语言** | Python 3.6 |
| **主要依赖包** | multiprocessing：多进程库 |
| socket：socket通信库 |
| struct：字节流解析/序列化工具库 |

## 模块划分

从文件上划分，服务由入口程序start.py、服务器类DNSRelayServer.py、协议解析工具函数DNSProtocol.py三部分构成，下面是具体介绍：

* **入口程序start.py**：解析命令行参数并启动服务，-r参数指定远程DNS服务器、-t参数指定本地IP-域名表、-b参数指定日志开关。
* **服务器类DNSRelayServer.py**
  + 成员变量
    - remote\_dns：远程DNS服务器地址
    - local\_dns\_file：本地IP-域名表数据
    - s\_listener：负责与客户端通信的socket（localhost:53）
    - id\_addr：DNS报文ID与客户端地址映射表
    - id\_data：DNS报文ID与客户端发送数据映射表
    - id\_dname：DNS报文ID与客户端请求域名映射表
    - printSwitch：全局日志输出开关，建议使用时关闭，开启会造成一定程度上的性能损耗
  + 成员函数
    - dns\_load()：加载本地IP-域名表数据至成员变量local\_dns\_file
    - dns\_update\_buffer()：更新本地缓存表，到达一定量后自动清除
    - dns\_server\_listener(send\_queue, help\_queue)：持续运行进程函数之一，负责监听53端口里客户端发送的DNS域名解析请求，它首先会校验请求合法性（必须是查询报文QR=0， 查询必须是A类型QTYPE=1），对于合法请求会根据本地IP-域名表/缓存检索结果来分发任务，如果在IP-域名表/缓存中存在请求域名，则连带对应IP加入发送队列send\_queue，准备发回客户端（0.0.0.0非法拦截延后至发送时处理）；如果不存在，则将请求报文加入查询队列help\_queue，准备向远程DNS查询
    - dns\_server\_sender(send\_queue)：持续运行进程函数之一，负责通过53端口向客户端发送回包，它首先检查发送队列send\_queue是否为空，若不为空则将队列内所有请求包重组为响应包，再不断发回客户端
    - dns\_server\_helper(send\_queue, help\_queue)：持续运行进程函数之一，负责向远程服务器查询请求解析未知域名，它首先检查查询队列help\_queue是否为空，若不为空则将队列内所有客户端请求包重组后发送给远程DNS服务器（这里设计上的难点是ID的重新分配，本次实验中采取在不重复的约束下随机生成ID方案），同时不断监听远程服务器的响应，在获得正确响应后将相关数据加入发送队列send\_queue
    - start()：类运行统一入口，负责启动进程与socket初始化，在创建DNSRelayServer对象后，可通过此成员方法启动服务器
* **协议解析工具函数DNSProtocol.py**
  + getDomainName(data)：解析请求包的查询域名
  + getPacketId(data)：解析DNS报文的ID
  + getPacketIp(data)：解析响应包的IP
  + createID(addr, mapping)：构造用于请求包的ID，约束目前有效的ID相同
  + createResponsePacket(addr, data, ip)：构造DNS响应报文，用于发向客户端，如果发现给定的ip是0.0.0.0，则构造错误码报文进行非法域名拦截
  + createQueryPacket(data, id, dname)：构造DNS请求报文，用于发向远程DNS服务器

在程序运行期间，主要依赖三大处理进程以及相关辅助函数、共享变量，三大处理进程分别指客户端请求处理分发进程dns\_server\_listener、客户端回包进程dns\_server\_sender以及远程DNS查询进程dns\_server\_helper，三进程架构保证了服务的高并发性能，其中dns\_server\_listener只负责快速收集客户端发送来的查询请求，分类（已有答案直接发回客户端或者需要向远程DNS服务器求助）后分发至下游任务，dns\_server\_sender专门将发送队列send\_queue中已有ip的相关数据重组为正确的响应报文后，持续发回客户端，dns\_server\_helper则不断从请求队列help\_queue中取出数据向远程DNS服务器查询，获得正确的响应后再放入发送队列send\_queue，交由dns\_server\_sender发回客户端；三大进程共享了相关功能函数、全局队列，功能函数诸如请求包/回包构造、DNS报文解析等都是线程安全函数，十分方便地为进程提供功能支持，全局队列是指发送队列send\_queue以及请求队列help\_queue，其中send\_queue存储将要发回客户端的有关数据，help\_queue存储将要发向远程DNS服务器的查询报文有关数据，它们通过管道技术实现进程间安全、高效通信，服务功能架构如图2所示。

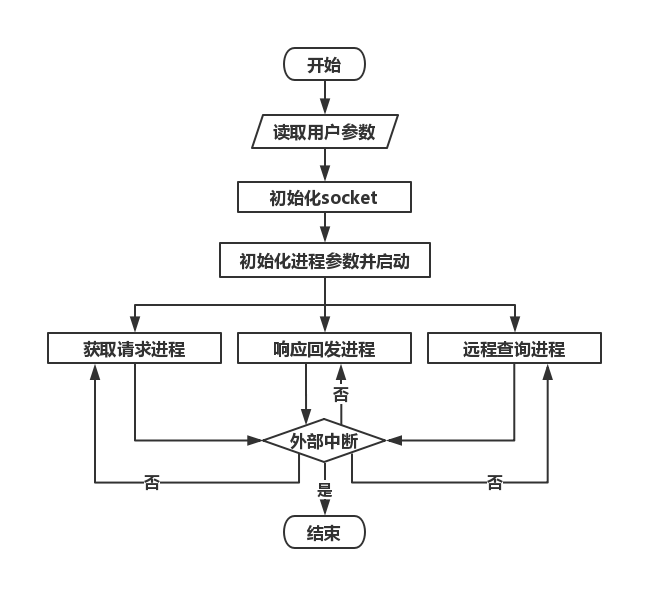


**图** 2 功能架构图

# 软件流程图

## 主流程图

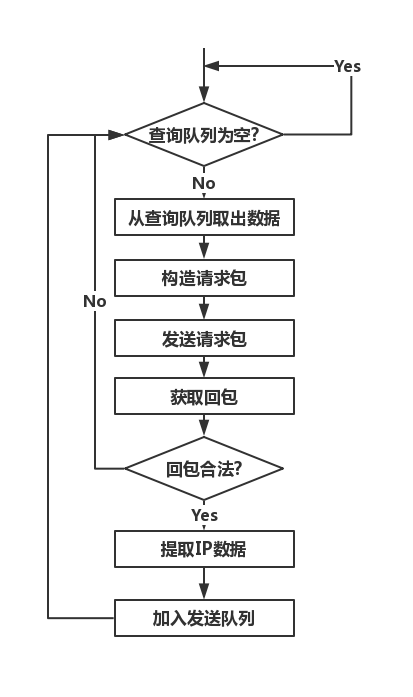
如图3所示，服务主流程由读取用户参数🡪初始化(socket以及进程)开始，之后三大进程（获取请求进程:dns\_server\_listener、响应回发进程:dns\_server\_sender、远程查询进程:dns\_server\_helper）持续运行直到外部给出中断指令。



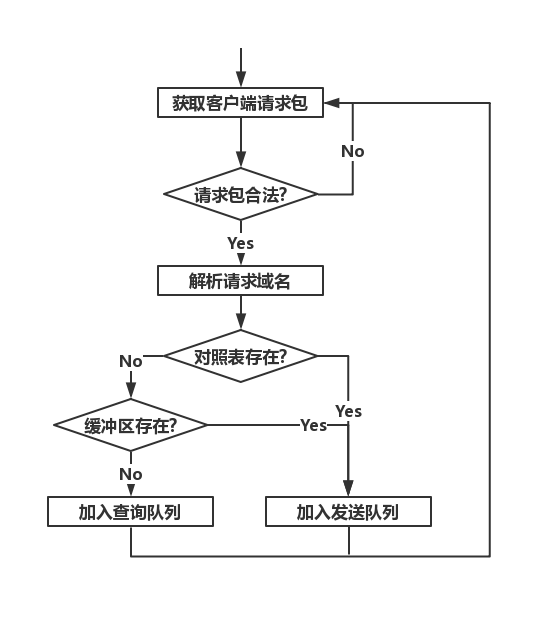
**图 3** 软件主流程图

## 子流程图

**图4** 响应回发子流程



**图 5** 远程查询子流程



**图 6** 获取请求子流程

### 获取请求:dns\_server\_listener

如图6所示，dns\_server\_listener不断接收客户端请求包，在校验请求合法后（必须是查询类型、QTYPE必须是A类型）取出其需要查询的域名，之后首先在本地IP-域名对照表中进行检索，如果域名或者缓冲区存在该域名则将对应IP以及请求包、客户端信息写入发送队列send\_queue，准备进行写回，否则将请求包、客户端信息写入请求队列help\_queue，准备进行查询。

### 响应回发:dns\_server\_sender

如图4所示，dns\_server\_sender不断从发送队列中取得数据，根据这些数据（响应IP、客户端信息）来构造响应包并发回对应客户端。

### 远程查询:dns\_server\_helper

如图5所示，dns\_server\_helper不断从请求队列中取得数据，根据这些数据来构造请求包向远端DNS服务器查询（ID要重新构造且不能和未完成），之后便不断接受远端DNS服务器的响应，如果合法则取出对应IP，与客户端信息一起写入发送队列，准备发回。

# 软件测试

## 测试用例设计

本次测试分为功能测试与性能测试，其中功能测试围绕运行场景、用例特殊性展开，两大运行场景分别为单机测试（本地使用）以及联机测试（远程机使用），用例特殊性考虑对照表中已有且正常的域名、对照表中已有但非法的域名以及对照表中不存在的域名，衡量标准分为nslookup请求解析正常与否以及浏览器能否正常使用。性能测试主要围绕并发量展开，编写脚本利用多线程的数量来模拟并发水平，观察DNS服务器的平均响应时间来评估其并发能力，整理如下：

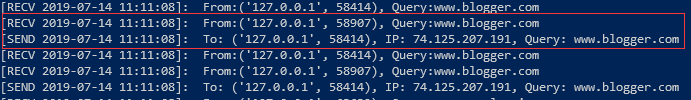
|  |  |  |
| --- | --- | --- |
| 功能测试 | 单机测试 | 表中正常域名 |
| 表中非法域名 |
| 表中不存在的域名 |
| 联机测试 | 表中不存在的域名 |
| 性能测试 | 不同并发量 | 随机生成的域名 |

## 测试结果分析

### 功能测试

1. 单机测试

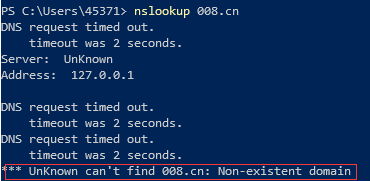
* 表中正常域名：[www.blogger.com](http://www.blogger.com)
  + 调试信息（RECV指接收到，SEND指发回）



* Wireshark抓包验证
* 表中非法域名：008.cn
  + 调试信息



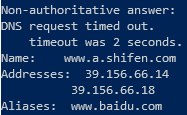
* + Nslookup工具



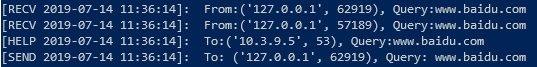
* 表中不存在的域名：[www.baidu.com](http://www.baidu.com)
  + 浏览器测试



* + Nslookup工具

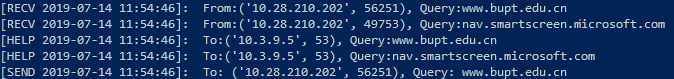


* + 调试信息（HELP是指向远程DNS服务器查询）



1. 联机测试（本机ip：10.201.8.84，测试对象：10.28.210.202，宿舍校园网）

* 测试用例：www.bupt.edu.cn
  + 本机日志



* + 测试机表现



* 测试用例：www.qq.com
  + 本机日志



* + 测试机表现



### 性能测试

为了模拟并发查询，本次实验专门开发了并发测试程序test.py，其通过调整多线程值t以及每个线程内查询次数m实现不同的并发量m\*t。测试程序每次均匀增加并发量，计算该轮次内响应时间（得到回包时间-查询时间）的相关统计指标（均值、分位数等）来评估并发水平。下图是本次性能测试结果，可以发现尽管随着并发量提高，平均耗时也在提高，不过仍在接受范围内。