目录

[一、 系统的功能设计 3](#_Toc1573542486)

[1. 基本要求 3](#_Toc267449397)

[2. 实验环境 3](#_Toc338742208)

[二、 模块划分 3](#_Toc261141659)

[1. 数据结构的存储 3](#_Toc1698771992)

[2. “IP-域名”映射表的处理 4](#_Toc465782679)

[3. “ID-客户端”映射的处理 4](#_Toc831592638)

[4. DNS报文的处理 4](#_Toc753892190)

[三、 软件流程图 5](#_Toc512520030)

[四、 测试用例以及运行结果 6](#_Toc367236093)

[五、 调试中遇到并解决的问题 8](#_Toc269013573)

[六、 课程设计工作总结 8](#_Toc858044476)

# 系统的功能设计

1. 基本要求  
 设计一个DNS服务器程序，读入“IP地址-域名”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，有三种可能检索结果：

（1）检索结果：ip地址0.0.0.0，则向客户端返回“域名不存在”的报错消息（不良网站拦截功能）

（2）检索结果：普通IP地址，则向客户端返回该地址（服务器功能）

（3）表中未检到该域名，则向因特网DNS服务器发出查询，并将结果返给客户端（中继功能）：考虑多个计算机上的客户端会同时查询，需要进行消息ID的转换。  
注意：表中只有IPV4地址，如果客户端想要查询的是IPV6地址，则需要向因特网DNS服务器发出查询。

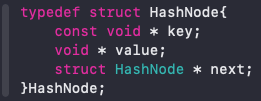
1. 实验环境

- 操作系统：Mac OS Mojave 10.14.6

- 编译环境：Xcode  
 - 编程语言：C语言

# 模块划分

## 数据结构的存储

（1）采用哈希表作为基本的存储数据结构，并采用链式法解决“冲突”。  
 - 哈希表的结构体  
   
 -哈希表中的结点的结构体  
   
（2）实现哈希表的以下操作：  
 ①创建哈希表；  
 ②插入结点；  
 - 插入结点使用的是头插法。  
 ③删除结点  
 ④销毁哈希表；  
 ⑤查找key对应的值；  
 ⑥哈希表的映射；  
 ⑦将哈希表转换为数组。  
（3）这部分内容对应的文件为：hash.h和hash.c

## 2. “IP-域名”映射表的处理

（1）ipCache是基本存储结构HashTable的扩展，以域名为key，考虑存在某个域名对应多个IP地址的情况，将ip存在另一个简化的HashTable（HashSet）中，以整个HashSet作为value。

- ipCache的哈希函数



（2）实现ipCache的以下操作：

①初始化；

②从文件中读取ip和域名的映射；

③查找域名对应的ip；

④插入新的域名节点；

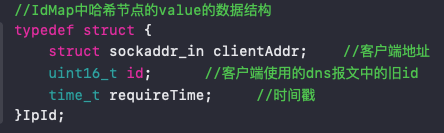
⑤销毁节点以及表。

（3）这部分内容对应的文件为：ipcache.h和ipchace.c

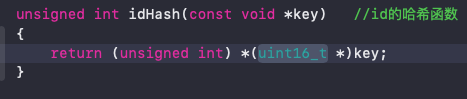
## 3. “ID-客户端”映射的处理

（1）IdMap是基本存储结构HashTable的扩展，以重新分配的id为key，以结构体IpId作为value。

- IdMap节点的value的数据结构



- IdMap的哈希函数



（2）实现IdMap的以下操作：

①初始化；

②插入新的id映射；

③查找id对应的ipid；

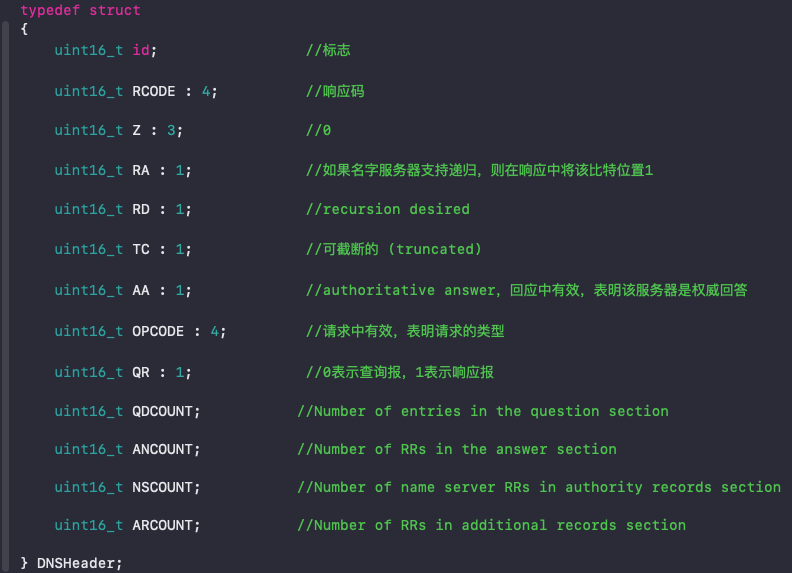
④刷新IdMap，删除超时映射；

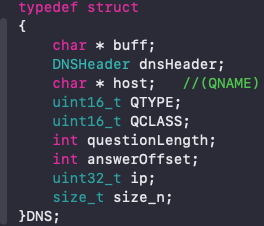
⑤销毁节点以及表。

（3）这部分内容对应的文件为：idmap.h和idmap.c

## 4. DNS报文的处理

（1）DNS报文头部的数据结构：

  
（2）DNS报文的数据结构

  
（3）实现对DNS报文的以下操作：  
 ①取出DNS头；

②得到域名；

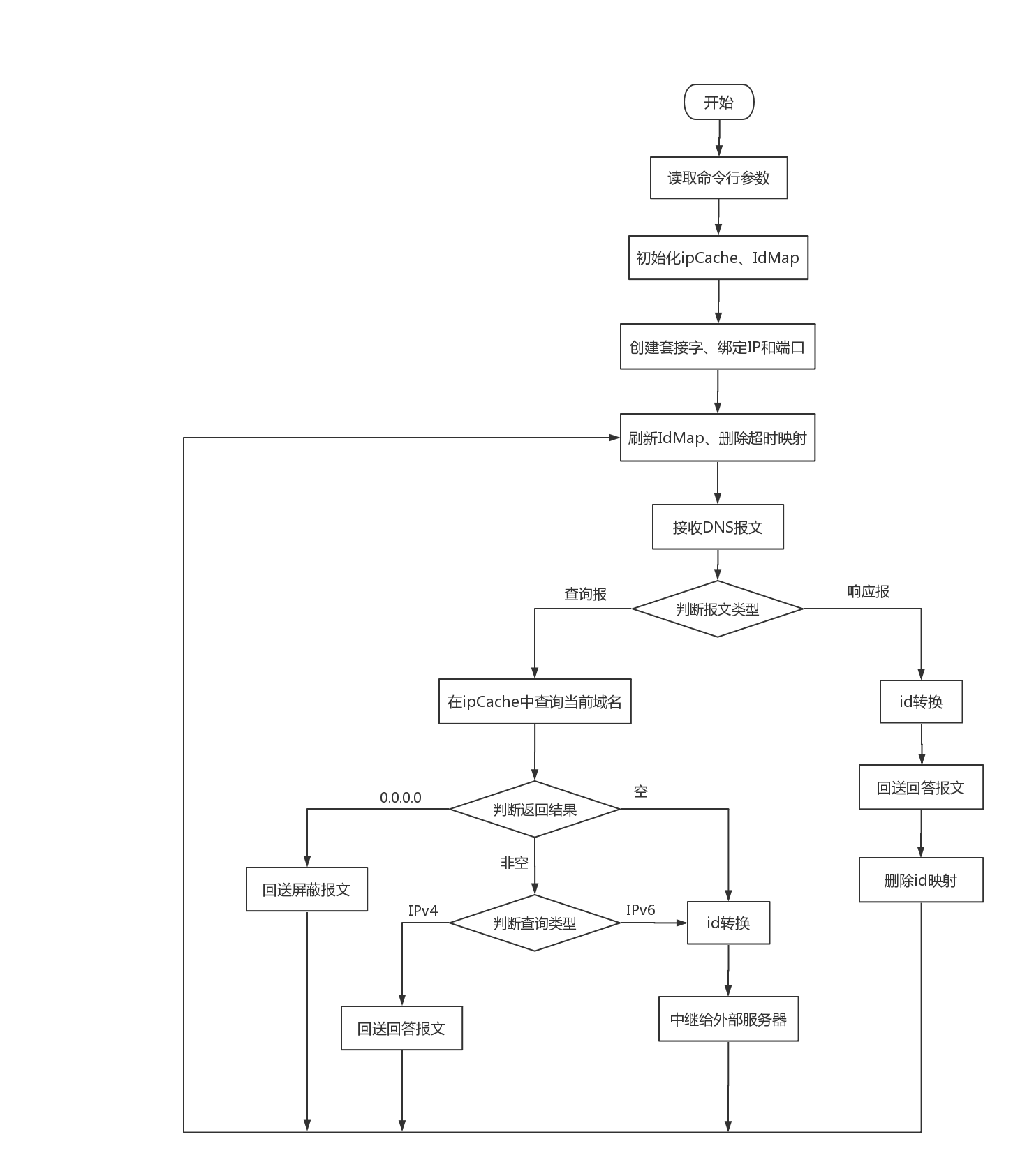
③加入回答资源；

④替换报文中的id；

⑤返回被拦截的查询。  
（4）这部分内容对应的文件为：DNS.h和DNS.c

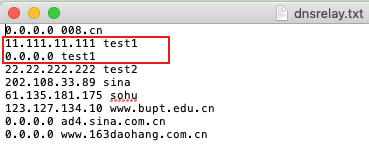
# 软件流程图

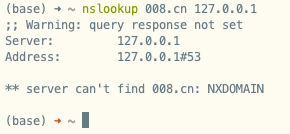
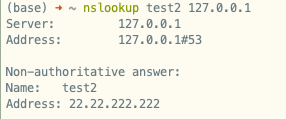
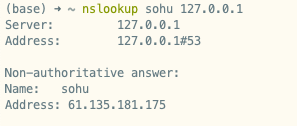
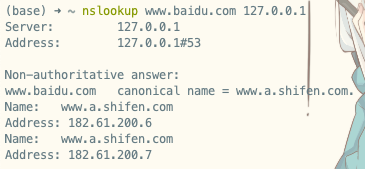
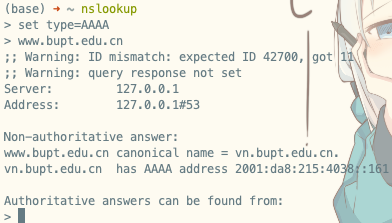
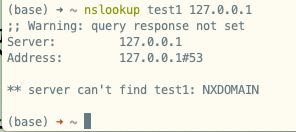
下图为main.c中的流程：



# 测试用例以及运行结果

dnsrelay.txt 如下图所示：

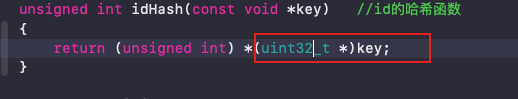


1. 基本功能测试  
   （1）不良网站拦截功能  
    选取IP地址为0.0.0.0的域名之一进行查询：  
      
   （2）服务器功能  
    选取存在于dnsrelay.txt中的域名之一，且对应IP地址不为0.0.0.0的域名进行查询：  
      
      
   （3）中继功能  
    选取不存在于dnsrelay.txt中的域名进行查询：  
    
2. IPV6地址查询测试  
    
3. 其他异常情况测试  
   （1）dnsrelay.txt中，一个域名对应了两个IP地址  
    这种情况下的返回结果应该是：只返回一个IP地址，如下图所示：  
    

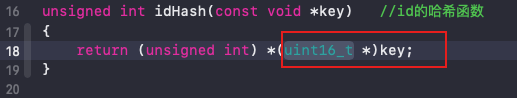
# 调试中遇到并解决的问题

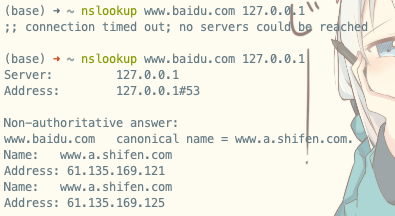
1. 查询dnsrelay.txt中不存在的域名时出现请求超时  
    错误情况如下图所示：

  
 经过检查后我们发现，是因为idmap.c中的哈希函数出了问题，错误如红框中所示：



将这里改成uint16\_t之后，程序运行正确：

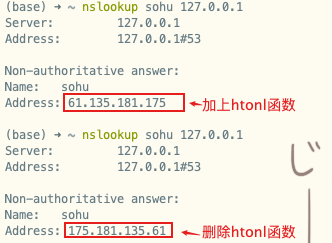




1. 查询到的IP地址错误  
    刚开始我们测试时发现，给客户端返回的IP地址与dnsrelay.txt中存储的是相反的。比如，dnsrelay.txt中存储的如下图：  
    

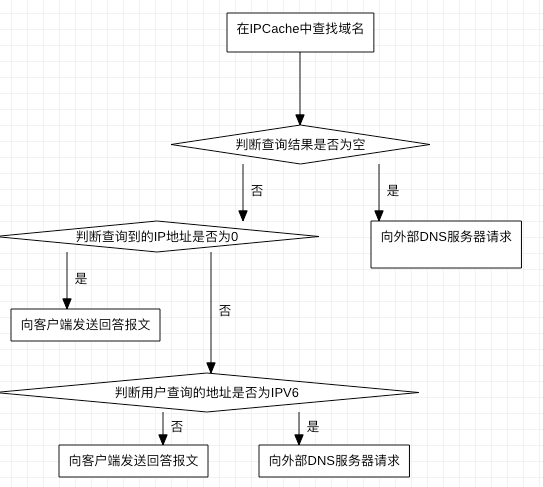
但是客户端返回的IP地址为：175.181.135.61。

经过检查后我们发现，需要把IP地址的字节序转换一下，解决方法如下图：  
 

即调用htonl()函数，将\*(ipArray[0])存储的IP地址主机字节顺序转换一下。  
 

3. 查询不到IPV6地址  
 在客户端请求的域名存在于dnsrelay.txt中的情况，我们的程序只能返回它的IPV4地址，经检查发现是因为我们在查询dnsrelay.txt时，如果检查到域名存在于dnsrelay.txt中，则不再向外部服务器请求查询。

所以我们修改了这部分逻辑为：



修改后即可在”客户端请求的域名存在于dnsrelay.txt中“这种情况下也可以成功查找到IPV6地址。

# 课程设计工作总结

关于DNS服务器这部分内容，我们在之前的计算机网络课程的学习中有过一定的了解，但对它的报文以及通信过程了解的并不十分深刻。这次的课程设计则让我们更深入的了解了DNS服务器工作的过程，并对其报文的格式与对应字段的意义有了进一步的掌握与了解。  
 同时，这次实验也让我们复习了socket编程。我们选择了UDP协议来完成这次实验，基本没有出现丢包导致超时等问题，开销也相对较小。但是可能仅限于我们测试时这种只有一个客户端的情况，在后期完成作业后我们发现，可以用多线程+TCP协议来实现更可靠的传输，并且注意线程之间的调度和锁的问题，但是时间上已经来不及了，我们觉得非常可惜。

还有，这次的编程使用了我们有段时间没有使用的C语言，虽然可能工作量大了一点，但是C语言能让我们更好的设计、实现哈希表这样基本数据结构，对于我们的能力的提升也起到了很大的帮助作用。

最后，这次实验让我们意识到模块化和数据结构的重要性，只有拥有清晰的程序架构和明确的数据结构，才能帮助我们在后续的代码编写过程中不偏离主线，也节省了很多debug的时间。