名称及地址转换

- 所有实施例中检查至今已有意使用 *IP地址：*
  - 这些IP地址用于通过连接到服务器的客户端，
  - 然而，最终用户很少知道的，更不用说使用，他们希望与之通信的服务器的IP地址（ES），
  - 相反，我们使用 ***主机名。***
- 远程主机被称为正常 **人力阅读** *名称：*
  - 运用 *名* 不是数字的是对人类更容易，
  - 它还允许 *数字IP地址* 远程主机以改变，而不改变 *主机名。*
- 但是，连接到远程服务器应用程序的客户端应用程序仍需要一个IP地址：
  - 需要一些方法来主机名映射到IP地址。

名称及地址转换

- 一个 *名称服务* 是 用于之间进行映射 *主机名* 和 *IP地址（* 除其他事项外）：
  - 映射A的处理 *主机名* 一个数，如IP地址或端口号被称为 ***解析度，***
  - 当从获得用于特定主机名的IP地址
    名称服务 主机名被认为是 ***解决。***
- 两个主名称服务来源：
  - 该 *域名系统（* DNS）。这是一种分布式名称服务需要使用DNS协议。和，
  - 本地配置数据库是特定于操作系统。
- 好在从编程的角度看名称服务的细节被隐藏：
  - 程序员只需要知道如何提出一个名字是
    解决。

解析器和名称服务器

- 组织通常运行一个或多个 *域名服务器*
  （DNS服务器）。
- 应用在被称为库联系DNS服务器通过调用函数 ***解析：***

  - 该 *分解器* 代码包含在系统库并且在被编辑的链路到应用程序 *建立* 处理，

- 调用到 *分解器* 代码使用的功能，例如由 ***的getaddrinfo（ ）*** 和 ***则getnameinfo（ ）***

  - 前者映射一个 *主机名* 到它的IP地址，后者则反向映射

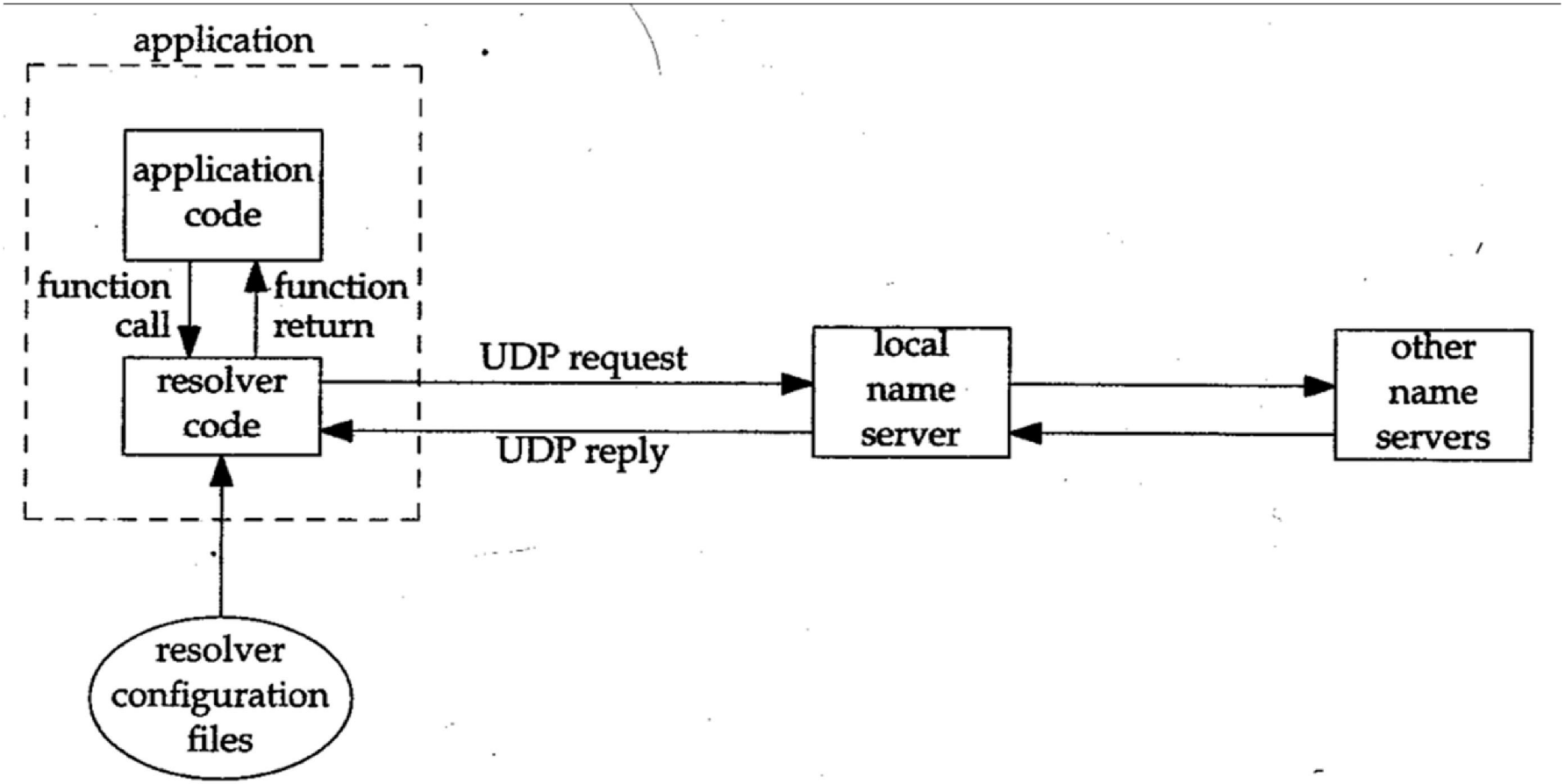
解析器和名称服务器

- 解析器代码指 *配置文件* 确定这个名称服务器的位置（一个或多个）
  - /etc/resolv.conf文件通常包含的IP地址 本地 域名服务器
- 解析器发送查询到的本地域名服务器：
  - 如果需要本地服务器将查询另一个名称服务器

名称及地址转换

- 在DNS条目被称为 *资源记录（* 的RR）：
  - *一个* 记录主机名映射到一个32位的IPv4地址，
  - *AAAA* 记录主机名映射到一个128位的IPv6地址，
  - *PTR* 记录图 *IP地址* 成 *主机名，*
  - *MX* 记录指定主机充当 *邮件交换器，*
  - CNAME记录映射共同服务，如 *FTP* 和 *万维网* 实际的主机提供服务
    - 例 www.dit.ie 有 *典范* 名称 *remus.dit.ie。*
- 我们感兴趣的是在的RR ***一个记录。***

解析器和名称服务器


application
application code
function call
function return
resolver code
UDP request
UDP reply
local name server
other name servers
resolver configuration files

该 *的getaddrinfo（ ）* 功能

- ***的getaddrinfo（ ）*** 执行用于一个查询 一个 记录：

  INT **的getaddrinfo（** 为const char * **hostStr，** 为const char * **serviceStr，**
  常量结构addrinfo中* **提示，** 结构addrinfo中** **结果）；**

  返回：NULL，如果OK和非错误代码，如果不成功。

  例如 **的getaddrinfo（** “www.google.com”，0，NULL，&addrList）；

- ***hostStr*** **指向表示一个**NULL结尾的字符串 **主机名** 如 *aisling，* **或者，一个完全合格域名（**FQDN），例如：
  aisling.student.dit.ie
- ***serviceStr*** *暂时将被忽略*
- ***提示*** 描述了要返回的信息类型
- ***结果*** **是**a的位置 *结构addrinfo中* 指针指向一 链表 包含结果即协议地址

该 *的getaddrinfo（ ）* 功能

- 链接列表中的每个条目是在举行 *addrinfo中结构*
  其声明如下：

结构addrinfo中{INT

填上ai_flags; //标志来控制信息解析

INT **ai_family;** //家庭：AF_INET，AF_UNSPEC等。

INT **ai_socktype; //** 接口类型：SOCK_STREAM，
SOCK_DGRAM

INT **ai_protocol; //** 协议：0（默认值）或IPPROTO_XXX

socklen_t的 **ai_addrlen; //** 套接字地址ai_addr的长度

结构sockaddr * **ai_addr; //** 套接字地址插座焦炭

***ai_canonname; //** 规范名称

结构addrinfo中* **ai_next; //** 在链表接下来addrinfo中};

该 *的getaddrinfo（ ）* 功能

- 该 **ai_addr** 字段包含 *SOCKADDR* 适当类型的结构中，填充了（数字）地址和端口信息。

- 在TCP / IP的情况下，适当的类型是一个 *SOCKADDR_IN* 结构，其具有以下成员：

```
结构SOCKADDR_IN
{uint8_t
                    sin_len;        //结构的长度（16）
sa_family_t         sin_family;     // AF_INET
在-port_t           sin_port;       // 16位的TCP或UDP端口号的网络字节命令

结构组in_addr       sin_addr;       // 32位的IPv4地址的网络字节
                                    命令
烧焦                的sin_zero; [8] // 没用过
};
```

该 *的getaddrinfo（ ）* 功能

- 该 链表 结果被返回 ***的getaddrinfo（ ）*** 必须释放：
  - 这需要使用辅助功能， ***freeaddrinfo（ ）***
  - 给定一个指针链表的头它释放分配给列表中的所有存储。如果不调用此方法可能会导致内存泄漏
- 下面的示例程序（GetAddrInfo.c）示出了使用的 ***的getaddrinfo（ ）*** 和 ***freeaddrinfo（***
  - 该方案需要两个命令行参数，主机名和服务名称（或端口号），并打印该IP地址（ES）

。/的getaddrinfo www.dit.ie HTTP

使用的样品程序 *的getaddrinfo（ ）*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <netdb.h>
5  #include "Practical.h"
6
7  int main(int argc, char *argv[]) {
8
9    if (argc != 3) // Test for correct number of arguments
10     DieWithUserMessage("Parameter(s)", "<Address/Name> <Port/Service>");
11
12   char *addrString = argv[1];   // Server address/name
13   char *portString = argv[2];   // Server port/service
14
15   // Tell the system what kind(s) of address info we want
16   struct addrinfo addrCriteria;                   // Criteria for address match
17   memset(&addrCriteria, 0, sizeof(addrCriteria)); // Zero out structure
18   addrCriteria.ai_family = AF_INET;              // The TCP/IP address family
19   addrCriteria.ai_socktype = SOCK_STREAM;         // Only stream sockets
20   addrCriteria.ai_protocol = IPPROTO_TCP;         // Only TCP protocol
21
22   // Get address(es) associated with the specified name/service
23   struct addrinfo *addrList; // Holder for list of addresses returned
24   // Modify servAddr contents to reference linked list of addresses
25   int rtnVal = getaddrinfo(addrString, portString, &addrCriteria, &addrList);
26   if (rtnVal != 0)
27     DieWithUserMessage("getaddrinfo() failed", gai_strerror(rtnVal));
28
29   // Display returned addresses
30   for (struct addrinfo *addr = addrList; addr != NULL; addr = addr->ai_next) {
31     PrintSocketAddress(addr->ai_addr, stdout);
32     fputc('\n', stdout);
33   }
34
35   freeaddrinfo(addrList); // Free addrinfo allocated in getaddrinfo()
36
37   exit(0);
38 }
```

该 *的getaddrinfo（ ）* 功能

- 第16行是 *结构* 该用于限制要返回的信息的类型：
  - 在这种情况下，我们只对TCP / IP地址感兴趣
  - **这样做的位置 *结构* 作为第三个参数被传递（ 提示 ）至 *的getaddrinfo（ ）***
- 第25行是调用 ***的getaddrinfo（ ）***
- 线30个遍历链表中的每个节点
- 31行，打印从IP地址和端口号 *ai_addr* 当前链接列表节点的成员：
  - ***ai_addr*** **指向 *结构* 类型 *SOCKADDR_IN***
  - **地址会被取自 *sin_addr* 和 *sin_port***
    这个成员 *SOCKADDR_IN* 结构体