前面有关套接字的UDP和TCP网络编程中，都是使用点分十进制、的ip地址字符串（例如”192.168.0.100”）来表示主机，使用数值端口来表示服务器（例如80表示http服务器）。但是多数情况下，可以使用名字而不是数值来表示主机：名字容易记，ip地址可以变动但名字保持不变，ipv6地址过长不便于记忆和键入等。本章主要介绍：**主机名与数值地址转换（gethostbyname，gethostbyaddr）**，**服务名与端口转换（getserverbyname，getserverbyaddr）**。前面的4个函数方法仅支持IPv4相关协议，还介绍与协议无关的函数**getaddrinfo、getnameinfo**。另外还提到了与套接字地址转换类似关于**可重入函数、局域网IP地址获取**的说明。

**1、主机名与地址解析**

在使用有关地址、名字相关的转换函数前，简单说明两个简单的概念。

1. 主机名

为了方便记忆和使用，对某一台网络设备进行标识并能被其他网络发现而起的名字。这个名字可以是简单的字符串，例如”localhost”标志本机，”DESKTOP-UANHMRS”标识某台windows电脑；还可以是一个全域名，例如“www.baidu.com”，“www.microsoft.com”。

1. 地址解析

主机名转换成地址时，需要用到地址解析。通常对于域名，需要DNS服务器进行解析，提供DNS解析的运营商维护域名和地址对应关系，当进行网络访问时会将实际的网络请求传递到实际的ip地址对应的物理主机进行处理。同样，也能使用静态的方式，例如在hosts文件中进行人为的指定域名(或简单名字)和地址对应关系（linux下/etc/hosts，windows下C:\Windows\System32\drivers\etc\hosts）。

**2、主机名与地址转换**

**2.1 gethostname**

#include <unistd.h>

int gethostname(char \*hostname, size\_t len); // 返回：0成功, -1出错

函数getsockname获取本机的主机名。

先在命令行中使用hostname获取本机名:



使用如下代码进行测试

|  |
| --- |
| #include <stdio.h>  #include <unistd.h> // gethostname  void main()  {  char host[32] = {0};  if( gethostname(host, sizeof(host)) < 0){  printf("gethostname failed. %s", strerror(errno));  }  printf("hostname: %s\n", host);  } |

测试结果如下



**2.2 gethostbyname**

#include <netdb.h>

struct hostent{

char \*h\_name; /\* 主机规范名. \*/

char \*\*h\_aliases; /\* 别名列表 \*/

int h\_addrtype; /\* 地址类型，固定AF\_INET \*/

int h\_length; /\* 地址长度，固定4 \*/

char \*\*h\_addr\_list; /\* IP地址列表 \*/

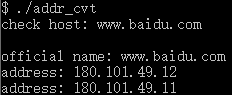
};

struct hostent \*gethostbyname (const char \*hostname);

//返回：成功返回非空指针，出错返回NULL并设置h\_errno

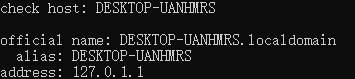
参数hostname是要查询的主机名，例如“localhost”,“www.baidu.com”，即可以是简单主机名或域名，也可能是点分十进制地址字符串，例如“192.168.0.1”。**当发生错误时会修改全局变量h\_errno，使用hstrerror函数输出错误说明**。目前**仅支持获取IPv4地址**。

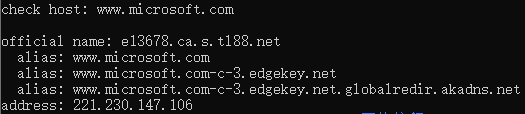
|  |
| --- |
| #include <stdio.h>  #include <unistd.h> // gethostname  #include <arpa/inet.h> // sockaddr\_in, inet\_addr  #include <netdb.h> // gethostby\*\*\*, getaddrinfo, h\_error enum  #include <errno.h>  #include <cstring>  int main()  {  const char host[] = "www.baidu.com";  printf("check host: %s\n\n", host);  struct hostent \*hptr;  // 只能返回IPv4地址，host允许是点分十进制字串  if( (hptr = gethostbyname(host)) == NULL) {  // 错误时，设置全局变量h\_errno，取值在<netdb.h>中，  // HOST\_NOTFOUND,TRY\_AGAIN, NO\_RECOVERY, NO\_DATA  printf("gethostbyname error from \"%s\": %s\n", host, hstrerror(h\_errno));  return ;  }  printf("official name: %s\n", hptr->h\_name);  char \*\*pptr;  for(pptr = hptr->h\_aliases; \*pptr != NULL; pptr++){  printf(" alias: %s\n", \*pptr);  }  char str[16];  for(pptr = hptr->h\_addr\_list; \*pptr != NULL; pptr++){  printf("address: %s\n", inet\_ntop(hptr->h\_addrtype, \*pptr, str, 16));  }  } |



其他测试如下：







**2.3 gethostbyaddr**

#include <netdb.h>

struct hostent \*gethostbyname (const char \*addr, socklen\_t len, int family);

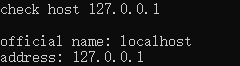
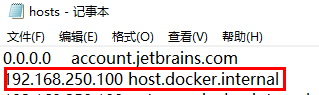
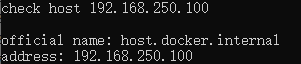
//返回：成功返回非空指针，出错返回NULL并设置h\_errno

gethostbyname用于由一个二进制的IP地址找到相应的主机名，感兴趣的是返回值hostent

中的h\_host变量。实际中网络编程用的不多，主要用在内网中查询局域网中的网络设备名称。

测试代码段

|  |
| --- |
| // char ip[] = "127.0.0.1" ; // "localhost"  char ip[] = "192.168.250.100";  in\_addr addr;  inet\_aton(ip, &addr); // ip地址转换为二进制地址  // inet\_pton(AF\_INET, ip, &addr)  if( (hptr = gethostbyaddr(&addr, sizeof(addr), AF\_INET)) == NULL) {  printf("gethostbyaddr error from: %s\n", hstrerror(h\_errno));  return ;  }  printf("official name: %s\n", hptr->h\_name);  char \*\*pptr;  for(pptr = hptr->h\_aliases; \*pptr != NULL; pptr++){  printf(" alias: %s\n", \*pptr);  }  char str[16];  for(pptr = hptr->h\_addr\_list; \*pptr != NULL; pptr++){  printf("address: %s\n", inet\_ntop(hptr->h\_addrtype, \*pptr, str, 16));  } |

**3、服务名与端口转换**

同主机一样，服务也通常靠名字来认知。在程序代码中通过其名字而不是其端口来指代按一个服务，而且从服务名字到端口号的映射关系保存在一个文件中（通常是/etc/services）。

**getserverbyname()函数用于根据名字查找相应的服务端口号**，**getserverbyaddr()函数用于根据指定端口号和可选协议查找对应的服务名**。

在端口服务映射文件/etc/services中，查看几个常见的的服务：

服务 端口/协议 服务别名 说明

domain 53/tcp # Domain Name Server

domain 53/udp

http 80/tcp www # WorldWideWeb HTTP

tftp 69/udp

服务domain使用端口号53，使用协议udp或tcp；服务http使用端口号80，仅使用tcp协议；服务tftp使用端口69，仅使用udp协议。

**3.1 getservbyname**

#include <netdb.h>

struct servent{

char \*s\_name; /\* 标准服务名 \*/

char \*\*s\_aliases; /\* 别名列表 \*/

int s\_port; /\* 端口号(网络序) \*/

char \*s\_proto; /\* 使用协议 \*/

};

struct servent \*getservbyname (const char \*servname, const char \*protoname);

//返回：成功返回非空指针，出错返回NULL并设置h\_errno

参数servname是要查询的服务名，必须指定。**如果同时指定了协议即protoname非空，那么要查询的服务必须有匹配的协议，否则出错**。如果未指定协议即传入protoname为NULL，而指定服务支持多个协议，那么返回的端口号是哪一个取决于实现。通常情况下，支持多个协议的服务使用相同的TCP和UDP端口号。

|  |
| --- |
| int main()  {  servent \*sptr = getservbyname("domain", NULL);  if(sptr == NULL){  printf("getservbyname error. %s\n", hstrerror(h\_errno));  return 0;  }  printf("service name: %s, port: %d, proto: %s", sptr->s\_name, ntohs(sptr->s\_port), sptr->s\_proto);  char \*\*pptr;  for(pptr = sptr->s\_aliases; \*pptr != NULL; pptr++){  printf(", alias: %s", \*pptr);  }  } |

下标中给出不同的测试结果，

|  |  |  |
| --- | --- | --- |
| servname | protoname | 输出 |
| “domain” | NULL | service name: domain, port: 53, proto: tcp |
| “domain” | ”udp” | service name: domain, port: 53, proto: udp |
| “http” | NULL | service name: http, port: 80, proto: tcp, alias: www |
| “http” | ”tcp” | service name: http, port: 80, proto: tcp, alias: www |
| “http” | ”udp” | getservbyname error. Resolver Error 0 (no error) |
| “tftp” | NULL | service name: tftp, port: 69, proto: udp |
| “tftp” | ”tcp” | getservbyname error. Resolver Error 0 (no error) |

实验结果情况如前函数说明一致。例如domain服务支持tcp和udp，端口号都是53，当不指定协议时，默认返回是tcp协议，端口53；指定协议udp时，同样返回端口53。又如http服务，仅支持tcp协议，使用端口80，当指定protoname为“udp”时无法解析；同样仅支持udp协议的tftp服务指定protoname为“tcp”时无法解析。

**3.2 getservbyaddr**

#include <netdb.h>

struct servent \*getservbyaddr (int port, const char \*protoname);

//返回：成功返回非空指针，出错返回NULL并设置h\_errno

参数port要求是网络字节序，protoname参数要求同getservbyport函数。这里关心返回的服务名即servent结构的s\_name。

给出如下实验表,

|  |  |  |
| --- | --- | --- |
| port | protoname | 输出 |
| htons(53) | NULL | service name: domain, port: 53, proto: tcp |
| htons(53) | ”udp” | service name: domain, port: 53, proto: udp |
| htons(80) | NULL | service name: http, port: 80, proto: tcp, alias: www |
| htons(80) | ”tcp” | service name: http, port: 80, proto: tcp, alias: www |
| htons(80) | ”udp” | getservbyport error. Resolver Error 0 (no error) |
| htons(69) | NULL | service name: tftp, port: 69, proto: udp |
| htons(69) | ”tcp” | getservbyport error. Resolver Error 0 (no error) |

处理结果和上一节完全一致。

**4、协议无关的转换**

**4.1 getaddrinfo**

前一小节中，gethostbyaddr和gethostbyaddr这两个函数仅支持IPv4，是建议废弃的（实际上目前IPv4还不能完全被IPv4替代）。新的函数getaddrinfo是协议无关的，同时支持IPv4和IPv6地址，能够处理主机名以及服务到端口这两种转换，返回结果是sockaddr结构列表而不是一个点分十进制的ip地址列表，sockaddr结构可以由套接字函数bind、listen、connect、recvfrom、sendto等直接使用。

**4.1.1 相关函数说明**

#include <netdb.h>

struct addrinfo{

int ai\_flags; /\* 标志，一个或多个AI\_xxx值 \*/

int ai\_family; /\* 套接字协议族 AF\_xxx \*/

int ai\_socktype; /\* 套接字类型 SOCK\_xxx \*/

int ai\_protocol; /\* IP协议类型 IPPROTO\_xxx \*/

socklen\_t ai\_addrlen; /\* 套接字地址结构的长度 \*/

struct sockaddr \*ai\_addr; /\* 套接字地址结构 \*/

char \*ai\_canonname; /\* 规范主机名，仅第一个结构体有值 \*/

struct addrinfo \*ai\_next; /\* 下一个addrinfo 地址，其ai\_canonname为空\*/

};

//返回：成功返回0，出错返回非0

int getaddrinfo (const char \*hostname, /\* 主机名或IPv4和IPv6点分十进制地址 \*/

const char \*service, /\* 服务名或对应的十进制端口号字串 \*/

const struct addrinfo \*hints, /\* 期望返回的信息类型的暗示 \*/

struct addrinfo \*\*res); /\* 成功时，指向addrinfo结构列表的指针\*/

参数hostname可以是主机名（包括如”localhost”简单主机名，可以是如“www.baidu.com的域名”），也可以是IPv4或IPv6的点分十进制地址数串。

参数service是需要查询的服务名或十进制服务端口数串，例如“domain”/”53”，“ssh”/”22”等。服务名或端口号的传递会受到hints参数ai\_flag的设置影响。

参数hints可以是一个空[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88)，也可以是一个指向某个addrinfo[结构体](https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E4%BD%93)的指针，调用者在这个结构中填入关于期望返回的信息类型的暗示。举例来说：指定的服务既可支持[TCP](https://baike.baidu.com/item/TCP/33012)也可支持[UDP](https://baike.baidu.com/item/UDP/571511)，所以调用者可以把hints结构中的ai\_socktype成员设置成SOCK\_DGRAM使得返回的仅仅是适用于[数据报](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%8A%A5/2194617)[套接口](https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%8F%A3)的信息。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 取值 | 值 | 说明 |
| ai\_family | AF\_UNSPEC | 0 | 协议无关 |
| AF\_INET | 2 | IPv4 |
| AF\_INET6 | 10 | IPv6 |
| ai\_protocol | IPPROTO\_IP | 0 | IP协议 |
| IPPROTO\_IPV4 | 4 | IPv4 |
| IPPROTO\_IPV6 | 41 | IPv6 |
| IPPROTO\_UDP | 17 | UDP |
| IPPROTO\_TCP | 6 | TCP |
| ai\_socktype | SOCK\_STREAM | 1 | 流式套接字 TCP socket |
| SOCK\_DGRAM | 2 | 数据报套接字 UDP socket |
| ai\_flags | AI\_PASSIVE | 1 | 被动的，用于bind，通常用于server socket |
| AI\_CANONNAME | 2 | 用于返回主机的规范名称 |
| AI\_NUMERICHOST | 4 | 禁止名字到地址映射，参数hostname必须为地址串 |
| AI\_NUMERICSERV | 1024 | 禁止名字到服务映射，参数service必须为十进制端口号串 |
| AI\_V4MAPPED | 8 | 允许返回IPv4映射的IPv6地址 |
| AI\_ALL | 16 | 返回IPv4映射（需开启AI\_V4MAPPED）、IPv6的地址 |

如果hint是一个空指针时，实际是设置hints是参数ai\_flags、ai\_socktype、ai\_protocol为0，ai\_famaly为AF\_UNSPEC。即代码块

|  |
| --- |
| struct addrinfo hints, \*res  bzero(&hints, sizeof(hints));  // hints.ai\_family = AF\_UNSPEC; // 可省略，AF\_UNSPEC值为0  getaddrinfo(hostname, service, &hints, &res) |

与 getaddrinfo(hostname, service, NULL, &res) 结果相同。

函数调用成功返回0，那么res参数指向的变量已经填入一个指针，它指的是由其中的ai\_next成员穿起来的addrinfo结构链表·。如果**出错返回非零值，使用const char \* gai\_strerror(int error);返回一个指向对应的出错信息字符串**。

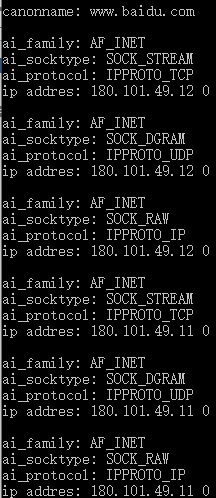
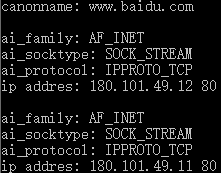
**4.1.2 使用示例与说明**

这里说明hostname、service传递参数不同情况，先直接给出以hostname为“www.baidu.com”service为NULL，仅设置ai\_flag为AI\_CANONNAME的示例代码。

|  |
| --- |
| #include <stdio.h>  #include <arpa/inet.h> // sockaddr\_in, inet\_addr  #include <netdb.h> // gethostby\*\*\*, getaddrinfo, h\_error, gai\_ strerror …  #include <cstring>  void main()  {  char host[] = "www.baidu.com";  char service[] = "http"; //http/80, ssh/22, domain/53 …  struct addrinfo hints, \*res;  bzero(&hints, sizeof(hints));  // hints.ai\_flags = AI\_PASSIVE;  hints.ai\_flags |= AI\_CANONNAME;  // hints.ai\_flags |= AI\_PASSIVE;  // hints.ai\_flags |= AI\_V4MAPPED | AI\_ALL;  // hints.ai\_family = AF\_UNSPEC;  // hints.ai\_flags |= AI\_NUMERICSERV;  // hints.ai\_socktype = SOCK\_STREAM;  // hints.ai\_socktype = SOCK\_DGRAM;  int ret;  if ((ret = getaddrinfo(host, NULL, &hints, &res)) < 0){  printf("get addrinfo error: %s\n", gai\_strerror(ret)); // getaddrinfo error  return;  }  // 返回成功且ai\_flags设置AI\_CANONNAME，有且仅有第一个addrinfo中规范主机名不为空  printf("canonname: %s\n", res->ai\_canonname);  // 打印输出  for (; res != NULL; res = res->ai\_next){  char family[16] = {0};  char socktype[16] = {0};  char protocal[16] = {0};  char ip[INET6\_ADDRSTRLEN] = {0};  int port; // 不指定服务即参数service传递NULL时，返回端口为0  if (res->ai\_family == AF\_INET){  memcpy(family, "AF\_INET", strlen("AF\_INET"));  sockaddr\_in \*addr = (sockaddr\_in \*)res->ai\_addr;  inet\_ntop(res->ai\_family, &addr->sin\_addr, ip, res->ai\_addrlen);  port = ntohs(addr->sin\_port);  }  else if (res->ai\_family == AF\_INET6){  memcpy(family, "AF\_INET6", strlen("AF\_INET6"));  sockaddr\_in6 \*addr = (sockaddr\_in6 \*)res->ai\_addr;  inet\_ntop(res->ai\_family, &addr->sin6\_addr, ip, res->ai\_addrlen);  port = ntohs(addr->sin6\_port);  }  else{  printf("unknow famaly %d\n", res->ai\_family);  continue;  }  switch (res->ai\_socktype){  case SOCK\_DGRAM: memcpy(socktype, "SOCK\_DGRAM", strlen("SOCK\_DGRAM")); break;  case SOCK\_STREAM:memcpy(socktype, "SOCK\_STREAM", strlen("SOCK\_STREAM")); break;  case SOCK\_RAW: memcpy(socktype, "SOCK\_RAW", strlen("SOCK\_RAW")); break;  default:  printf("\nunknow socktype %d\n", res->ai\_socktype);  memcpy(socktype, "UNKNOW", strlen("UNKNOW"));  }  switch (res->ai\_protocol){  case IPPROTO\_IP: memcpy(protocal, "IPPROTO\_IP", strlen("IPPROTO\_IP"));break;  case IPPROTO\_UDP: memcpy(protocal, "IPPROTO\_UDP", strlen("IPPROTO\_UDP")); break;  case IPPROTO\_TCP: memcpy(protocal, "IPPROTO\_TCP", strlen("IPPROTO\_TCP"));break;  default:  printf("unknow protocal %d\n", res->ai\_protocol);  memcpy(protocal, "UNKNOW", strlen("UNKNOW"));  }  printf(  "\nai\_family: %s\n"  "ai\_socktype: %s\n"  "ai\_protocol: %s\n"  "ip addres: %s %d\n",  family, socktype, protocal, ip, port);  }  freeaddrinfo(res); // 使用getaddrinfo后必须调用以释放动态分配的res内存  } |

上述代码的运行结果如下左侧截图，代码中设置了参数hint的ai\_flag仅为了获取规范名字，并且仅有第一个地址结构中的规范名字段ai\_canonname 有数据。遍历地址链表的结果实际与getaddrinfo(hostname, NULL, NULL, &res)相同。

在前面的gethostbyname中已经知道主机名“www.baidu.com” 关联了两个ip地址，这里返回了两个ip地址相关适应每个地址族的每个地址对应的一个结构。简单来说，当不设置service、hints时，将返回所有不同服务、不同ip地址对应的主机名的地址结构。这里没有指定任何服务，所有地址中的端口号都为0。

即如果中。修改代码，仅将service参数设置为“http”或“80”时，会从上述左侧地址中查找满足当前”http服务”查询的结果，返回结果如上右侧截图，端口号数为80。

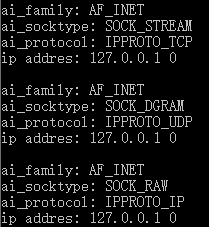
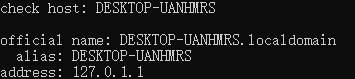
注意，由于结果**指针res指向的链表内存是动态分配的，调用完成后需要使用特定函数freeaddrinfo(res)进行内存释放**。

下面介绍传递参数的四类组合情况的对比。

**(1) service NULL，hints NULL**

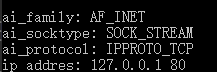
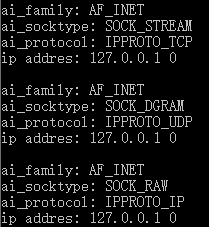
这种情况在上面的例子中说明，功能和getaddrbyname()类似。查询主机名www.baidu.com的地址时，getaddrbyname()仅返回了两个ip地址的结果，共2个结果；getaddrinfo()也返回同样两个ip地址的结果，但是将关联这两个ip地址的所有协议服务等分条列出，共6个结果。

查询主机名”localhost”的地址，getaddrbyname和getaddrinfod结果分别如下：



**(2) service 非空，hints NULL**

这种情况下，相当于从（1）service NULL，hints NULL的处理结果中，在对查询的主机名，筛选返回满足指定服务的地址信息。在上面的函数说明中已经给出了示例，这里给出调用getaddrinfo("localhost", NULL, NULL, &res)和getaddrinfo("localhost ", "http", NULL, &res)的运行结果，当指定”http”服务（SOCK\_STREAM、IPPROTO\_TCP）仅返回满足条件的信息。分别如下图。



**(3) service NULL，hints 非空**

这种情况下，这种情况下，相当于从（1）service NULL，hints NULL的处理结果（6个）中，在对查询的主机名，筛选返回满足hints条件的地址信息。

例如，我们查询主机名“”www.baicu.com”, hints设置套接字类型为SOCK\_STREAM，再将套接字类型为SOCK\_DGRAM。两次处理修改代码和结果如下

|  |  |
| --- | --- |
| struct addrinfo hints, \*res;  bzero(&hints, sizeof(hints));  hints.ai\_socktype = SOCK\_STREAM;  getaddrinfo("www.baidu.com", NULL, &hints, &res)； |  |
| struct addrinfo hints, \*res;  bzero(&hints, sizeof(hints));  hints.ai\_socktype = SOCK\_DGRAM;  getaddrinfo("www.baidu.com", NULL, &hints, &res)； |  |

**(4) service 非空，hints 非空**

相当于在（3）service NULL，hints 非空基础上再一次筛选满足服务的地址信息。这里我们以主机名”www.taobao.com”为例，其关联多个ip主机且支持IPv6协议。

示例代码和结果

|  |
| --- |
| char host[] = "www.taobao.com";  struct addrinfo hints, \*res;  bzero(&hints, sizeof(hints));  hints.ai\_flags |= AI\_CANONNAME; // 规范名  hints.ai\_flags |= AI\_NUMERICSERV； // 服务名必须使用十进制端口号数串  hints.ai\_family = AF\_INET6; // IPv6协议  hints.ai\_socktype = SOCK\_STREAM; // 流式套接字  char service[] = "80"; // 设置NUMERICSERV标志，不能使用“http”  getaddrinfo(host, service, &hints, &res)； |



通常，在使用getaddrinfo()函数时一般通过参数hints和参数service，以返回我们期望的地址信息结果：

* **指定hostname和service**

用于TCP或UDP客户端调用getaddrinfo()的常规输入。TCP客户调用返回后，遍历循环每一个返回的ip地址，逐一调用socket、connect，直到有一个连接成功，或者所有地址尝试一遍。

UDP客户由getaddrinfo()填入的套接字地址结构调用sento或connect。如果客户能判断第一个地址不工作（已连接套机字上收到出错消息，或者在未连接的套接字上接受消息超时），可以尝试其余的地址。

如果客户清楚只处理一种类型套接字，应该把hints结构的ai\_socktype成员设置成SOCK\_STREAM或SOCK\_DGRAM。

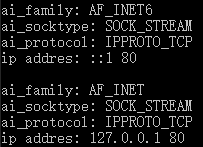
* **只指定service，不指定hostname**

典型的服务器进程使用情况，仅指定service而不指定hostname，同时在hints中指定AI\_PASSIVE标志，返回的套接字地址结构中应含有INADDR\_ANY(对与IPv4)

或IN6ANND\_ANY\_INIT（对于IPv6）的IP地址。TCP服务器随后调用socket、bind和listen，UDP服务器将调用socket、bind和recvfrom。

与典型的客户一样，如果服务器清楚自己只处理一种类型套接字，应该把hints结构的ai\_socktype成员设置成SOCK\_STREAM或SOCK\_DGRAM，避免返回多个结构，其中可能出现错误的ai\_socktype值。

调用getaddrinfo(NULL, “http”, NULL, &res)；的结果



然而实际使用中，这种获取回环地址127.0.0.1的方式只能在本机上使用，在局域网中需要获取局域网的ip地址，相关内容在第5.3节中说明。

**4.1.3 不同应用场景举例**

这里对使用getaddrinfo()函数在不同应用场景下的举例，给出演示代码

（1）Host server

|  |
| --- |
| struct addrinfo \*  Host\_serv(const char \*host, const char \*serv, int family, int socktype)  {  int n;  struct addrinfo hints, \*res;  bzero(&hints, sizeof(struct addrinfo));  hints.ai\_flags = AI\_CANONNAME; /\* always return canonical name \*/  hints.ai\_family = family; /\* 0, AF\_INET, AF\_INET6, etc. \*/  hints.ai\_socktype = socktype; /\* 0, SOCK\_STREAM, SOCK\_DGRAM, etc. \*/  if ((n = getaddrinfo(host, serv, &hints, &res)) != 0)  err\_quit("host\_serv error for %s, %s: %s",  (host == NULL) ? "(no hostname)" : host,  (serv == NULL) ? "(no service name)" : serv,  gai\_strerror(n));  return (res); /\* return pointer to first on linked list \*/  } |

（2）Tcp connect

调用getaddrinfo一次，指定地址族为AF\_UNSPEC，套接字类型为SOCK\_STREAM。函数返回之后，对每个ip地址进行调用socket和connect。当返回地址有IPv6地址而主机不支持IPv6时，调用socke失败时就应该退出函数。

|  |
| --- |
| int tcp\_connect(const char \*host, const char \*serv)  {  int sockfd, n;  struct addrinfo hints, \*res, \*ressave;  bzero(&hints, sizeof(struct addrinfo));  hints.ai\_family = AF\_UNSPEC;  hints.ai\_socktype = SOCK\_STREAM;  if ((n = getaddrinfo(host, serv, &hints, &res)) != 0)  err\_quit("tcp\_connect error for %s, %s: %s",  host, serv, gai\_strerror(n));  ressave = res;  do{  sockfd = socket(res->ai\_family, res->ai\_socktype, res->ai\_protocol);  if (sockfd < 0)  continue; /\* ignore this one \*/  if (connect(sockfd, res->ai\_addr, res->ai\_addrlen) == 0)  break; /\* success \*/  Close(sockfd); /\* ignore this one \*/  } while ((res = res->ai\_next) != NULL);  if (res == NULL) /\* errno set from final connect() \*/  err\_sys("tcp\_connect error for %s, %s", host, serv);  freeaddrinfo(ressave);  return (sockfd);  } |

（3）Tcp listen

创建一个TCP套接字，捆绑到一个总所周知的端口，并允许接收外来的连接请求。初始化addrinfo结构提供如下暗示信息：AI\_PASSIVE(本函数供服务器使用)、AF\_UNSPEC(也能明确的指定地址族为AF\_INT或AF\_INET6)、SOCK\_STREAM。调用socket和bind函数，如果任一个调用失败就尝试下一个IP地址。对于TCP服务器，总是设置SO\_REUSEADDR套接字选项。调用bind成功后继续调用listen，使的当前套接字变成监听套接字以接收外来连接请求。

|  |
| --- |
| int tcp\_listen(const char \*host, const char \*serv, socklen\_t \*addrlenp)  {  int listenfd, n;  const int on = 1;  struct addrinfo hints, \*res, \*ressave;  bzero(&hints, sizeof(struct addrinfo));  hints.ai\_flags = AI\_PASSIVE;  hints.ai\_family = AF\_UNSPEC;  hints.ai\_socktype = SOCK\_STREAM;  if ((n = getaddrinfo(host, serv, &hints, &res)) != 0)  err\_quit("tcp\_listen error for %s, %s: %s",  host, serv, gai\_strerror(n));  ressave = res;  do{  listenfd = socket(res->ai\_family, res->ai\_socktype, res->ai\_protocol);  if (listenfd < 0)  continue; /\* error, try next one \*/  Setsockopt(listenfd, SOL\_SOCKET, SO\_REUSEADDR, &on, sizeof(on));  if (bind(listenfd, res->ai\_addr, res->ai\_addrlen) == 0)  break; /\* success \*/  Close(listenfd); /\* bind error, close and try next one \*/  } while ((res = res->ai\_next) != NULL);  if (res == NULL) /\* errno from final socket() or bind() \*/  err\_sys("tcp\_listen error for %s, %s", host, serv);  Listen(listenfd, LISTENQ);  if (addrlenp)  \*addrlenp = res->ai\_addrlen; /\* return size of protocol address \*/  freeaddrinfo(ressave);  return (listenfd);  } |

（4）Udp clinet

创建一个未连接的UDP套接字。输入的是目的ip地址和端口服务，返回目标套接字、目标套接字地址结构和长度，便于直接利用在稍后的sendto函数。传递的参数saptr和lenp的空间要足够保存目标地址结构数据。

|  |
| --- |
| int udp\_client(const char \*host, const char \*serv, sockaddr \*\*saptr, socklen\_t \*lenp)  {  int sockfd, n;  struct addrinfo hints, \*res, \*ressave;  bzero(&hints, sizeof(struct addrinfo));  hints.ai\_family = AF\_UNSPEC;  hints.ai\_socktype = SOCK\_DGRAM;  if ((n = getaddrinfo(host, serv, &hints, &res)) != 0)  err\_quit("udp\_client error for %s, %s: %s",  host, serv, gai\_strerror(n));  ressave = res;  do{  sockfd = socket(res->ai\_family, res->ai\_socktype, res->ai\_protocol);  if (sockfd >= 0)  break; /\* success \*/  } while ((res = res->ai\_next) != NULL);  if (res == NULL) /\* errno set from final socket() \*/  err\_sys("udp\_client error for %s, %s", host, serv);  \*saptr = Malloc(res->ai\_addrlen);  memcpy(\*saptr, res->ai\_addr, res->ai\_addrlen);  \*lenp = res->ai\_addrlen;  freeaddrinfo(ressave);  return (sockfd);  } |

（5）udp connect

创建一个已连接的UDP套接字，不需要同udp client中保存目标地址套接字结构，将sendto改用为write。

|  |
| --- |
| int udp\_connect(const char \*host, const char \*serv)  {  int sockfd, n;  struct addrinfo hints, \*res, \*ressave;  bzero(&hints, sizeof(struct addrinfo));  hints.ai\_family = AF\_UNSPEC;  hints.ai\_socktype = SOCK\_DGRAM;  if ((n = getaddrinfo(host, serv, &hints, &res)) != 0)  err\_quit("udp\_connect error for %s, %s: %s",  host, serv, gai\_strerror(n));  ressave = res;  do{  sockfd = socket(res->ai\_family, res->ai\_socktype, res->ai\_protocol);  if (sockfd < 0)  continue; /\* ignore this one \*/  if (connect(sockfd, res->ai\_addr, res->ai\_addrlen) == 0)  break; /\* success \*/  Close(sockfd); /\* ignore this one \*/  } while ((res = res->ai\_next) != NULL);  if (res == NULL) /\* errno set from final connect() \*/  err\_sys("udp\_connect error for %s, %s", host, serv);  freeaddrinfo(ressave);  return (sockfd);  } |

（6）udp server

除了没有调用listen函数，代码几乎等同于tcp\_listen。这里设置地址族为AF\_UNSPEC，以支持IPv4和IPv6，并且明确指出了套接字类型为SOCK\_DGRAM。**对于UDP服务端套接字不设置SO\_REUSEADDR，因为UDP套接字没有TCP的TIME\_WAIT状态**，也就没有必要在启动服务器时设置该选项。

|  |
| --- |
| int udp\_server(const char \*host, const char \*serv, socklen\_t \*addrlenp)  {  int sockfd, n;  struct addrinfo hints, \*res, \*ressave;  bzero(&hints, sizeof(struct addrinfo));  hints.ai\_flags = AI\_PASSIVE;  hints.ai\_family = AF\_UNSPEC;  hints.ai\_socktype = SOCK\_DGRAM;  if ((n = getaddrinfo(host, serv, &hints, &res)) != 0)  err\_quit("udp\_server error for %s, %s: %s",  host, serv, gai\_strerror(n));  ressave = res;  do{  sockfd = socket(res->ai\_family, res->ai\_socktype, res->ai\_protocol);  if (sockfd < 0)  continue; /\* error - try next one \*/  if (bind(sockfd, res->ai\_addr, res->ai\_addrlen) == 0)  break; /\* success \*/  Close(sockfd); /\* bind error - close and try next one \*/  } while ((res = res->ai\_next) != NULL);  if (res == NULL) /\* errno from final socket() or bind() \*/  err\_sys("udp\_server error for %s, %s", host, serv);  if (addrlenp)  \*addrlenp = res->ai\_addrlen; /\* return size of protocol address \*/  freeaddrinfo(ressave);  return (sockfd);  } |

**在能明确套接字类型时，尽量给出其ai\_socktype标志值**。**设置地址族标志ai\_family为AF\_UNSPEC时，创建的socket应用是协议无关的，同时支持IPv4和IPv6**。

注意，尽管getaddrinfo比gethostbyname和getserverbyname要“好”（便于编写协议无关的程序代码，能够同时处理主机名和服务，返回信息都是动态分配的）。使用时必须先分配一个hint结构，清零后填写需要的字段，再调用getaddrinfo，然后遍历返回结果链表中的每一个地址。

**4.2 getnameinfo**

函数getaddrinfo解决把主机名和服务名转换为套接字地址结构的问题。getnamerinfo函数则相反，它把套接字地址结构转换成主机名和服务名。

#include <netdb.h>

# define NI\_NUMERICHOST 1 /\* 以数串格式返回主机字符串 \*/

# define NI\_NUMERICSERV 2 /\* 以数串格式返回服务字符串 \*/

# define NI\_NOFQDN 4 /\* 只返回QFDN的主机名部分 \*/

# define NI\_NAMEREQD 8 /\* 若不能从地址解析出名字则返回错误 \*/

# define NI\_DGRAM 16 /\* 数据报服务 \*/

//返回：成功返回0，出错返回非0

int getnameinfo(struct sockaddr \*sockaddr, socklen\_t addrlen /\* 套接字地址结构和长度 \*/

char \*host, socklen\_t hostlen /\* 主机字符串 \*/

char \*serv, socklen\_t servlen /\* 服务字符串 \*/

int flags); /\* 标志 \*/

参数sockaddr指向套接字地址结构，addrlen为该套接字地址结构长度，通常由accept、recvfrom、getsockname或getpeername返回。字符串host和serv用于接收主机名和服务，当对应参数长度hostlen、servlen设置为0表示不想返回对应字符串。例如，仅希望返回主机名，可以调用getnameinfo((struct sockaddr \*)&addr, sizeof(addr), hbuf, hbuflen, NULL, 0, flags);。类似gethostbyaddr函数使用，给出一个示例代码如下：

|  |
| --- |
| void main()  {  char host[] = "127.0.0.1";  char hbuf[NI\_MAXHOST];  char sbuf[NI\_MAXSERV];  int hbuflen = sizeof(hbuf);  int sbuflen = sizeof(sbuf);  sockaddr\_in addr;  addr.sin\_family = AF\_INET;  inet\_pton(AF\_INET, host, &addr.sin\_addr);  addr.sin\_port = htons(8080);    int flags = 0;  flags |= NI\_DGRAM;  int ret = getnameinfo((struct sockaddr \*)&addr, sizeof(addr),  hbuf, hbuflen, sbuf, sbuflen, flags);  if (ret < 0){  printf("getnameinfo error: %s\n", gai\_strerror(ret)); // getnameinfo error  return;  }  printf("hbuf: %s, sbuf: %s\n", hbuf, sbuf);  } |

结果为



**5 其他**

**4.1 可重入**

本章节中使用到的函数**gethostbyname、gethostbyaddr、getservbyname和getservbyport这四个函数都是不可重入的**，由于他们都在内部返回一个静态结构的指针。

getaddrinfo和getnameinfo可重入的前提是由它调用的函数都可以重入，因为动态分配是内部完成。

同样的，在套接字函数中的错误信息errno变量也存在类似问题，每个进程有一个该变量的副本，在多线程中使用错误混乱。通常先将全局erron的值保存，调用相关函数后再回复其值。

在**地址转换中，inet\_pton和inet\_ntop总是可重入的，而inet\_ntoa是不可重入的**。

本章节中，提供了一些可重入的版本函数，函数名增加“\_r”后缀，例如gethostbyname\_r、gethostbyaddr\_r、getservbyname\_r和getservbyport\_r。不同的平台、不同操作系统版本，可能提供的实现方式不同。这里以WSL下ubuntu16.04为例，说明gethostbyname\_r函数：

//返回：成功返回0，出错返回非0

int gethostbyname\_r (const void \*hostname, // 主机名

struct hostent \*hostbuf, // 地址结构缓冲区

char \*buf, size\_t buflen, // 缓冲区地址和长度

struct hostent \*\* res, // 地址结构链表

int \* h\_errnop); // 错误指针

相对于 struct hostent \*gethostbyname (const char \*hostname), 参数增加了多个，主要是需要预先分配保存hostent结构链表的缓冲区。

|  |
| --- |
| void main()  {  char host[] = "www.taobao.com";  printf("check host: %s\n\n", host);  struct hostent hostbuf, \*res = NULL;  char buf[8192] = {0};  int err = 0;  if( gethostbyname\_r(host, &hostbuf, buf, sizeof(buf), &res, &err) < 0){  printf("gethostbyname\_r error from \"%s\": %s\n", host, hstrerror(err));  return;  }  printf("official name: %s\n", res->h\_name);  char \*\*pptr;  for (pptr = res->h\_aliases; \*pptr != NULL; pptr++){  printf(" alias: %s\n", \*pptr);  }  char str[16];  for (pptr = res->h\_addr\_list; \*pptr != NULL; pptr++){  printf("address: %s\n", inet\_ntop(res->h\_addrtype, \*pptr, str, 16));  }  } |

运行结果完全同 gethostbyaddr()函数。

**4.3 作废的ipv6解析**

gethostbyname(const char\* hostname)仅支持IPv4地址的解析，存在支持IPv6的地址解析函数，在函数名后增加了”2”的后缀，例如gethostbyname2(const char\* hostname, int af)。

其他函数在这里不做说明。建议直接使用getaddrinfo函数。

**4.3 获取本机的局域网地址**

前面的方法获取本机地址多为回环地址127.0.0.1。我们在需要外其他机器提供服务时，就必须需要知道本机在局域网中的ip地址。

有以下方法：

（1）创建socket，和外部UDP或TCP服务端建立连接，之后使用getsockname获取本机ip地址

（2）使用ioctl()函数以及结构体 struct ifreq和结构体struct ifconf来获取网络接口的各种信息。给出指定网卡名称eth0，返回其绑定的ip。

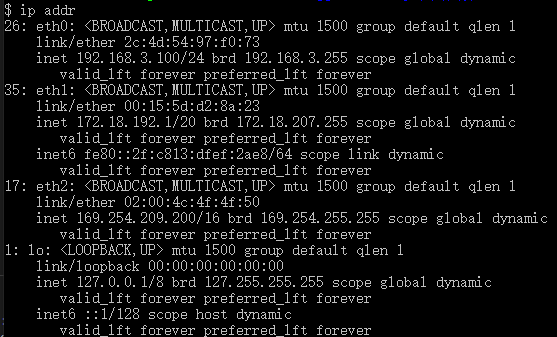
|  |
| --- |
| void main()  {  // https://www.cnblogs.com/baiduboy/p/7287026.html  int inet\_sock;  ifreq ifr;  char ip[46]={};  inet\_sock = socket(AF\_INET, SOCK\_DGRAM, 0);  strcpy(ifr.ifr\_name, "eth0"); // 指定网卡名称  ioctl(inet\_sock, SIOCGIFADDR, &ifr);  inet\_ntop(AF\_INET, &(((struct sockaddr\_in \*)&ifr.ifr\_addr)->sin\_addr), ip, sizeof(sockaddr\_in));    printf("%s IP Address %s\n", ifr.ifr\_name, ip);  } |



（3）使用getifaddrs() 函数遍历网卡绑定的ip地址

|  |
| --- |
| #include <stdio.h>  #include <errno.h>  #include <cstring>  #include <ifaddrs.h>  void main()  {  ifaddrs \*ifAddrStruct = NULL;    if (getifaddrs(&ifAddrStruct) < 0){  printf("getifaddrs failed. %s", strerror(errno));  }  for (ifaddrs \*ifa = ifAddrStruct; ifa != NULL; ifa = ifa->ifa\_next) {  if (!ifa->ifa\_addr) {  continue;  }  if (ifa->ifa\_addr->sa\_family == AF\_INET) { // check it is IP4  // is a valid IP4 Address  void\* tmpAddrPtr= &((sockaddr\_in \*)ifa->ifa\_addr)->sin\_addr;  char addressBuffer[INET\_ADDRSTRLEN];  inet\_ntop(AF\_INET, tmpAddrPtr, addressBuffer, INET\_ADDRSTRLEN);  printf("%s IP Address %s\n", ifa->ifa\_name, addressBuffer);  } else if (ifa->ifa\_addr->sa\_family == AF\_INET6) { // check it is IP6  // is a valid IP6 Address  void\* tmpAddrPtr = &((sockaddr\_in6 \*)ifa->ifa\_addr)->sin6\_addr;  char addressBuffer[INET6\_ADDRSTRLEN];  inet\_ntop(AF\_INET6, tmpAddrPtr, addressBuffer, INET6\_ADDRSTRLEN);  printf("%s IP Address %s\n", ifa->ifa\_name, addressBuffer);  }  }    freeifaddrs(ifAddrStruct);  } |

先使用命令行ip addr查询本机的所有网卡ip地址信息：



运行结果如下，同样可以在循环中根据网口名称、地址族协议，获取指定的IP数据。

