地址信息

- 很多时候，有必要对应用层和TCP层之间交换寻址信息。
- 客户端和服务器所需要的套接字创建之后做到这一点。
  - 客户需要通过联系信息服务器的连接原始调用之前。
  - 服务器需要告诉它要监听的地址的TCP。这是由原始绑定使用。
- 偶尔有必要在相反方向传递信息
  即TCP到应用层。这会在另一个时间来照顾。
- 不管方向的所有寻址信息必须是正确的字节顺序（稍后讨论）的，并且只能通过传递 *参考* 通过标准化的地址结构。
- 这种结构是由套接字API指定，并在下一幻灯片进行了讨论。

套接字地址结构

结构组in_addr

{in_addr_t s_addr;

/ * 32位的IPv4地址的网络字节

责令* /};

结构SOCKADDR_IN

{uint8_t sin_len;

/ *结构（16）的长度* /

sa_family_t sin_family; / * * AF_INET /

在-port_t sin_port; / * 16位的TCP或UDP端口号的网络字节排序* /

结构组in_addr sin_addr; / * 32位的IPv4地址

网络字节命令* /

炭的sin_zero; [8] /* 没用过 */

};

- 只关注结构的三个成员：
  - *sin_family，sin_addr，属于* 和 *罪端口。*
  - 该 *的sin_zero* 构件垫结构，以在尺寸上至少16个字节

套接字地址结构

- 套接字地址结构是具有本地意义
  - 也就是说，它们是 不 不同的主机之间传送
- 套接字地址结构总是通过引用传递

- 然而 *插座功能* 是专门用来对付套接字地址结构从许多支持的协议族

- 这些功能做 不 明白了 通用
  指针类型 (*无效* *)
- 因此一 *通用* 以下形式的套接字地址结构的创建：

该 *通用* 套接字地址结构

结构sockaddr {

```
uint8_t         sa_len;
sa_family_t上sa_family;          / *地址族：AF_xxx值* /
烧焦            sa_data [14];    / *特定于协议的地址* /
```

};

- 套接字函数被定义为采用指针到这个通用套接字地址结构

- 例如，在 *捆绑* 功能：

```
INT绑定（listenfd，（SA *）＆servaddr，的sizeof（servaddr））；
```

其中，SA被定义为 *结构sockaddr* 和serveraddr被宣布为（ *结构SOCKADDR_IN* ）。

字节顺序

- 两种方式来存储一个16位的整数，它是由2个字节组成：
  - 在起始地址，称为存储低位字节 *小尾数* 字节顺序
  - 在起始地址，称为存储高位字节 *大端* 字节顺序
- 由给定的系统中使用的字节顺序是被称为 *主机字节顺序*。 不幸的是，这两个字节序之间没有标准

字节顺序

- 客户端和服务器应用程序通常会跨使用两种格式的主机系统扩展。
- 因此对网络应用程序的程序员必须处理 *字节序* 区别如下：
  - TCP使用16位端口号和一个32位的IPv4地址。
  - 两端协议栈必须同意这些字节的数量级上
  - TCP / IP使用 *大端* 字节顺序而被称为 *网络字节顺序*
- 套接字地址结构内的某些字段必须从被转换 *主机字节顺序* 至 *网络字节顺序*

字节顺序和操作函数

- 根据转化率水平有两组的可以使用的功能;
  - 字节排序功能是在他们处理与字符串到数字到字符串转换的最简单
  - 字节处理函数较为复杂，它们即应对更复杂的字符串操作，从点分十进制形式到数字到十进制记数法
- 有四个字节排序功能：

  htons（ ）- 主机的16位值转换为网络字节顺序htonl（ ）- 转换主机32位的值，以网络字节顺序ntohs和（ ）- 转换16位网络值到主机字节顺序再用ntohl（ ）- 转换的32位网络值到主机字节顺序

字节操作函数

- **inet_pton**
  - 这个函数接受一个ASCII字符串（*介绍*）代表该目的地地址（点分十进制表示法），并将其转换成一个二进制值（*数字*）输入到套接字地址结构（即 *网络字节顺序*）

- **inet_ntop**
  - 这个函数的逆转换，从一个即
    数字 二进制值为ASCII字符串表示（*演示文稿），即点分十进制表示法*

- 这两个函数与IPv4的工作 和 IPv6地址

  #包括<ARPA / inet.h> INT inet_pton（INT *家庭*，为const char * *strptr*，无效* *addrptr*）；

  - 返回：1，如果OK，0，如果输入不是一个有效的呈现格式，-1错误
  - 转换的字符串指向 *strptr*，通过该指针存储该二进制结果 *addrptr*

字节操作函数

  #包括<ARPA / inet.h> const char * inet_ntop（INT *家庭*，常量无效* *addrptr*，字符* *strptr*，为size_t *LEN*）；

  - 返回：造成如果正常，出错返回NULL指针

- 该 *家庭* 参数两种功能是AF _INET因为我们只用IPv4地址有关。
- 该 *LEN* 参数是 尺寸 目标缓冲区的和被传递，以防止功能溢出缓冲器

该 *接受* 功能 - 捕捉客户端地址

- 接受由服务器调用返回从连接队列中的下一个连接：
  - 如果队列为空，则进程进入睡眠

```
#包含<sys / socket.h>中

INT接受（INT *的sockfd*，结构 *SOCKADDR * cliaddr*， socklen_t的* *addrlen中* ）；
```

  - 返回：非负描述符，如果OK，-1错误

- *接受* 返回多达三个值：
  - 一个整数返回代码，或者是一个新的套接字描述符或错误指示，
  - 该 *协议* 客户端进程的地址（通过 *cliaddr* 指针）
  - 这个地址的大小（通过 *addrlen中*
    指针）的 *接受* 功能 - 捕捉客户端地址

- 该 *cliaddr* 和 *addrlen中* 参数用于返回 *协议* 客户端进程的地址：
  - 在调用之前 *接受* 即发：
    - * *addrlen中* 被设置成客户端地址结构的大小（ *cliaddr* ）
- 该 *接受* 功能 - 捕捉客户端地址
  - 在返回该整型值包含在套接字地址结构的内核存储字节的实际数量
- 如果我们不感兴趣的最后两个参数设置为客户端的地址 **空值** 指针

该 *接受* 功能 - 一个例子

```
3       INT主（INT的argc，焦炭** argv的）
五      {
        。
7       socklen_t的clntAddrLen; //新变量来保存地址结构的长度
8       结构SOCKADDR_IN servaddr，cliaddr; //新的地址结构
9       炭clntName [INET_ADDRSTRLEN]; //缓冲区来存放客户端地址
        。
19      clntAddrLen =的sizeof（cliaddr）; //确定客户端地址结构长度
20      connfd =接受（listenfd，（SA *）＆cliaddr，＆clntAddrLen）; //调用接受
21      的printf（，inet_ntop（AF_lNET，＆cliaddr.sin_addr，clntName，的sizeof（clntNam
        e）），ntohs和（cliaddr.sin_port）“来自％s，端口％d \ n连接”）; //打印出的客户端
        地址
        。
```