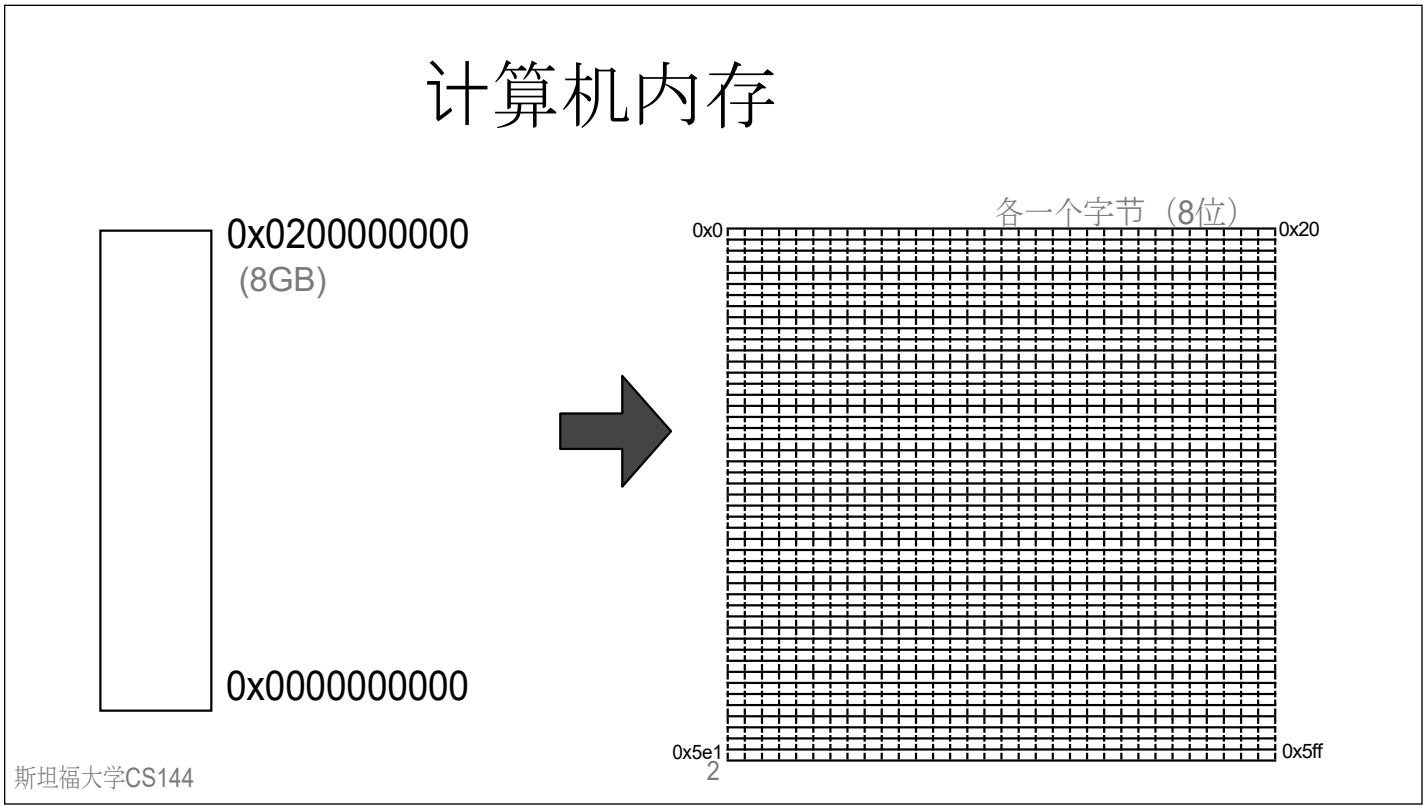
内存、字节顺序和数据包格式

对于双方的沟通，他们需要就他们所交换的信息达成一致。如果一方认为信息是西班牙语，而另一方认为是柬埔寨语，他们将无法沟通。对于计算机通信来说，这意味着同意信息有哪些字段，如何安排和格式化，以及如何表示它们。为了生成一个要发送的信息，软件通常要在内存中创建一个副本，然后将其传递给网络卡。同样，当计算机收到一个信息时，网卡会把这个信息放在内存中，然后软件就可以访问它。如果你想了解网络协议和编写网络协议软件，了解这些工作原理和你可能遇到的一些陷阱是很重要的。



因此，让我们从计算机内存的一个简单模型开始。在今天的大多数计算机中，内存是以字节为单位组织的：8比特的内存块。一个程序有一个地址空间，从地址0开始。今天的大多数计算机都是64位的：这意味着内存地址是64位的，所以一台计算机最多拥有2到64个字节，或18个六亿字节。实际上，今天的计算机没有这么多内存：它们有千兆字节，也就是2比30。在这个例子中，我们的计算机有8个千兆字节的内存，所以它的最大地址是所示的十六进制值。软件可以访问这个内存的每一个字节，也可以成组访问字节，比如用一条指令从8个连续的内存字节单元加载一个64位的整数。

内联性

1,024 = 0x0400 =

?	?

- 多字节词：你如何安排字节？
- 小恩典：最小有效字节位于最低地址
  - 从寻址/计算的角度看，这是最合理的。

0x00	0x04

- 大恩典：最重要的字节位于最低地址
  - 对人类读者来说是最有意义的

0x04	0x00

但计算机如何表示一个多字节的值呢？比方说，我们想表示数字1,024，在十六进制中是0x0400，或4次256。这个值需要16位，或两个字节。哪个字节在前：0x00还是0x04？你如何在内存中布置一个多字节的值，这被称为字节性，有两种选择。在小恩典中，最不重要的字节位于最低地址。因此，在内存中，最不重要的字节排在第一位。事实证明，从计算和架构的角度来看，这是最合理的。另一个选择是大恩典（big endian），其中最有意义的字节是最低地址。对人类读者来说，大恩典更有意义，因为这就是我们写数字的方式，最重要的数字在前。

测验

对于每一个数字，标记出十六进制的表示方法是大恩典还是小恩典。

不要使用计算器或其他工具!

宽度	小数	字节	大端数	小尾数
16位	53	0x3500		
16位	4116	0x1014		
32位	5	0x00000005		
32位	83,886,080	0x00000005		
32位	305,414,945	~~0x21433412~~		

这里有一个小测验。对于每个数字，请标出其十六进制表示法是大恩典还是小恩典。不要使用计算器或其他工具!

53是用小恩典表示的。53是3乘以16加5，0x35在第一个字节。

4116是大恩典。4116等于4096加20。所以这两个字节是0x10和0x14，其中0x10是代表更重要的比特，即4096的比特。由于十六进制是0x1014，这意味着最重要的字节在前，而且是大恩典。

5是big endian -- 最不重要的字节在最后，并且有最高的地址。

83,886,080是小字节：它是5乘以2的24次方，所以这意味着0x05是最重要的字节。

最后，305,414,945是小恩典。我没有尝试找出所有的数字，只是看了看最小有效位。最小有效位是0x21或0x12的一部分。 如果是0x21，最小有效位是1，那么这个数字是奇数。如果它是0x12，并且最小有效位是0，那么这个数字是偶数。由于305,414,945是奇数，这意味着0x21是最没有意义的字节，该数字被存储为小方位的。

网络字节顺序

- 不同的处理器有不同的编码方式
  - Little endian: x86, big endian:ARM
- 为了实现互操作，他们需要就如何表示多字节字段达成一致。
- 网络字节顺序为大殷商

1,024 = 0x400 =


```
uint16_t val = 0x400;
uint8 t* ptr = (uint8 t*)&val;
```

```
如果 (ptr[0] == 0x40) {
  printf("big endian\n")
}
else if (ptr[1] == 0x40) {
  printf("little endian\n")。
}
否则 {
  printf("unknown endianness!\n")
  。
}
```


那么，为什么这很重要呢？如果两台计算机要进行通信，它们需要就它们是用大恩典还是小恩典格式表示数字达成一致。由于不同的处理器使用不同的编码方式，这就很复杂了。例如，英特尔和AMD的x86处理器是小恩典：最小有效字节在前。相比之下，ARM处理器，如iPhone中的处理器，是大恩典，即最有意义的字节在先。

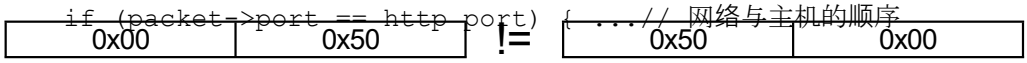
我们不希望两台计算机关心或知道对方是大恩典还是小恩典。因此，协议规范机构通常会选择一个，并坚持使用它。对于互联网来说，这意味着大恩迪安。所有作为互联网规范的协议都使用大 endian 格式。

这里有一个C语言代码的例子，它可以告诉你你的计算机是大英数还是小英数。它需要一个16字节的值，并向其投掷一个指针，让代码单独查看这些字节。如果索引0的字节是0x40，最重要的字节就在前面，它是大恩典。如果索引1的字节是0x40，那么它就是小字节。如果两者都不是，那就说明发生了奇怪的事情。

便携式代码

- 你必须将网络字节顺序值转换为你的主机顺序
- 例如，数据包有一个按网络字节顺序排列的16位端口，你用一个结构来访问它，你想在你的X86处理器上检查该端口是否为80

```
uint16_t http_port = 80; // 主机顺序
if (packet->port == http_port) { ...// 网络与主机的顺序
```



- 辅助函数：htons(), ntohs(), htonl(), ntohl()
  - htons:"主机到网络短线", ntohl: "网络到主机长线"

  #include <arpa/inet.h>

```
uint16_t http_port = 80; // Host order
uint16_t packet_port = ntohs(packet->port);
if (packet_port == http_port) { ...// OK
```


但是，等等 -- 这产生了一个复杂的问题。你需要一个数据包是大编码的格式，但是如果你的处理器是小编码的呢？比如说，你想把一个TCP段的端口号设置为80，即HTTP端口。一个简单的方法是创建一个C结构，在正确的偏移量上有一个字段端口。但是，如果你使用一个80的值与端口字段进行比较，它将被存储为小字节，第一个字节为0x50。大endian需要0x50存储在第二个字节中。所以尽管段中的端口字段是80，这个测试也会失败。

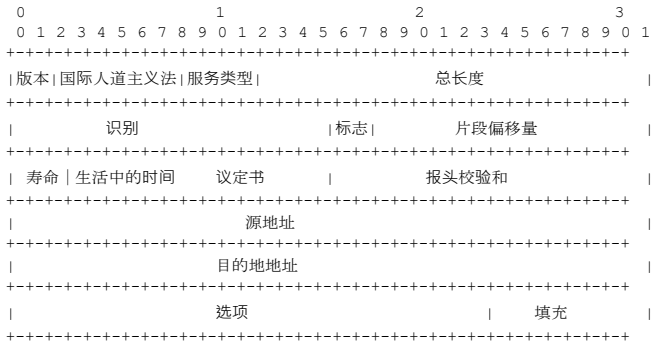
为了使之更容易，C网络库提供了在主机顺序和网络顺序之间转换的实用函数。例如，函数htons()接收一个16位的主机短值作为参数，并返回一个网络顺序的值。还有一些将网络短码转换为主机短码的函数，以及用于长码、32位数值的函数。所以测试数据包端口是否为80的正确方法是读取数据包结构的端口字段，并调用ntohs将其从网络顺序转换为主机顺序。然后你可以将其与80进行比较，得到正确的结果。在小恩典结构的情况下，ntohs和htons将两个字节的顺序颠倒。如果是大的endian结构，它们只是返回不变的值。

每当你处理网络数据时都要小心谨慎!

否则你会因为忘记转换或转换两次而浪费
很多（可避免的）时间来调试你的代码。

这些函数为你提供了机制，通过这些机制你可以编写独立于处理器架构的网络代码。但是要小心！我怎么强调都不为过。这一点我怎么强调都不为过。当你处理网络数据时要小心。如果你在主机和网络顺序之间的转换上没有原则性和严谨性，你会给自己带来巨大的头痛，因为你忘记了转换，或者不小心转换了两次，突然间你的协议表现错误，或者触发了各种奇怪的错误。

数据包格式

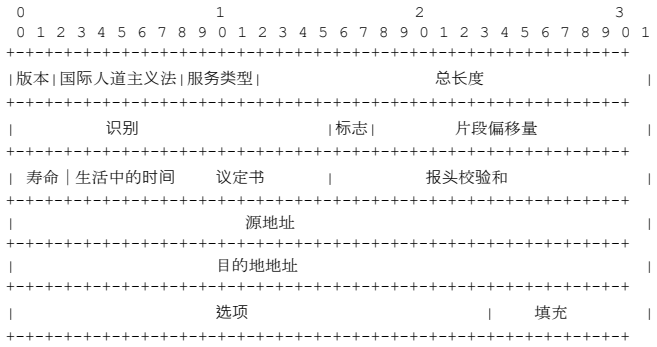


互联网数据报头的例子

现在我们知道了互联网规范是如何按网络顺序（或大恩典）排列多字节值的，我们可以看看互联网规范是如何描述其数据包格式的。由于历史原因，互联网规范是以纯ASCII文本编写的。左边的文本块是从Request for Comments (RFC) 791中逐字摘录的，它规定了IP协议的第四版，即IPv4。上面显示了从0到31的位数 -- 数据包的宽度为4字节。由于IPv4有5行必填字段，这意味着一个IPv4头至少有20字节长。尼克和我在展示数据包时经常使用更简单的视觉格式，就像右边的那个。

以此为例，一个IPv4数据包的总长度字段是2个字节，即16位长。这意味着一个IPv4数据包的长度不能超过65,535字节。数据包中的那个字段被存储为大字节。一个长度为1400字节的数据包被存储为0x0578。所以这个长度的IP包的第三个字节是0x05。

数据包格式



互联网数据报头的例子

让我们在wireshark中看看这个。我只是要启动wireshark并监听数据包。第一个数据包是关于一个叫TLS的东西，即传输层安全。它是网络浏览器用于安全连接（https）的东西。TLS向我们隐藏了数据包的内容，但我们仍然可以看到它的标题。使用wireshark，我们可以看到TLS的有效载荷在一个TCP段内，该TCP段为443端口，是标准的TLS端口。这个TCP段是在一个IPv4头里。详细看一下IPv4头，我们可以看到数据包的总长度是1230。1230的十六进制是0x04ce：1024，或0x04乘以256加106，或0xce。在底部，Wireshark向我们显示了该数据包的实际字节数。就是这个，04ce，以大恩典或网络顺序显示。

你已经看到不同的处理器是如何以不同的方式排列数字的。但由于网络协议需要达成一致，协议规范决定了数字的排布方式，这可能与你的处理器不同。为了帮助解决这个问题，C网络库提供了在主机和网络顺序之间转换的辅助函数。但要小心使用它们!胡乱使用它们很容易导致你失去许多调试的时间，而这些时间可以通过在开始时的谨慎和在代码中决定一个有原则的转换方法来避免。