Big Endian 和 Little Endian

Peter Lee 2008-04-20

一、字节序

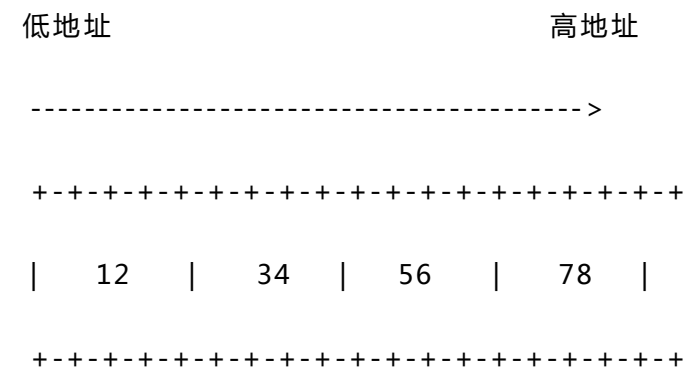
来自：http://ayazh.gjjblog.com/archives/1058846/

谈到字节序的问题，必然牵涉到两大 CPU 派系。那就是 Motorola 的 PowerPC 系列 CPU 和 Intel 的 x86 系列 CPU。PowerPC 系列采用 big endian 方式存储数据，而 x86 系列则采用 little endian 方式存储数据。那么究竟什么是 big endian，什么又是 little endian 呢？

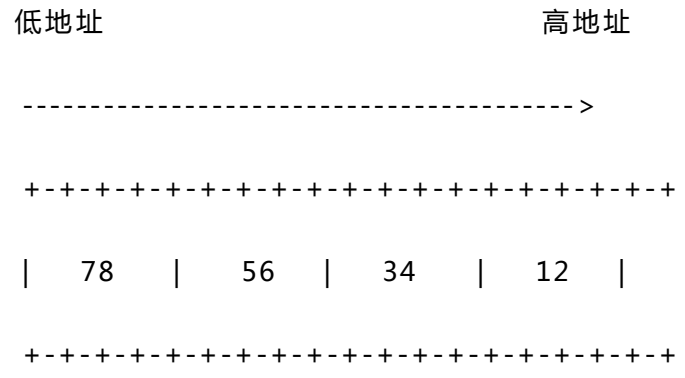
其实 big endian 是指低地址存放最高有效字节(MSB)，而 little endian 则是低地址存放最低有效字节（ LSB ）。

用文字说明可能比较抽象，下面用图像加以说明。比如数字 0x12345678 在两种不同字节序 CPU 中的存储顺序如下所示：

Big Endian



Little Endian



从上面两图可以看出，采用 big endian 方式存储数据是符合我们人类的思维习惯的。而 little endian，!@#$%^&*，见鬼去吧 -_-|||

为什么要注意字节序的问题呢？你可能这么问。当然，如果你写的程序只在单机环境下面运行，并且不和别人的程序打交道，那么你完全可以忽略字节序的存在。但是，如果你的程序要跟别人的程序产生交互呢？在这里我想说说两种语言。C/C++语言编写的程序里数据存储顺序是跟编译平台所在的 CPU 相关的，而 JAVA 编写的程序则唯一采用 big endian 方式来存储数据。试想，如果你用 C/C++语言在 x86 平台下编写的程序跟别人的 JAVA 程序互通时会产生什么结果？就拿上面的 0x12345678 来说，你的程序传递给别人的一个数据，将指向 0x12345678 的指针传给了 JAVA 程序，由于 JAVA 采取 big endian 方式存储数据，很自然的它会将你的数据翻译为 0x78563412。什么？竟然变成另外一个数字了？是的，就是这种后果。因此，在你的 C 程序传给 JAVA 程序之前有必要进行字节序的转换工作。

无独有偶，所有网络协议也都是采用 big endian 的方式来传输数据的。所以有时我们也会把 big endian 方式称之为网络字节序。当两台采用不同字节序的主机通信时，在发送数据之前都必须经过字节序的转换成为网络字节序后再进行传输。

ANSI C 中提供了下面四个转换字节序的宏。

big endian：最高字节在地址最低位，最低字节在地址最高位，依次排列。

little endian：最低字节在最低位，最高字节在最高位，反序排列。

endian 指的是当物理上的最小单元比逻辑上的最小单元小时，逻辑到物理的单元排布关系。咱们接触到的物理单元最小都是 byte，在通信领域中，这里往往是 bit，不过原理也是类似的。

一个例子：

如果我们将 0x1234abcd 写入到以 0x0000 开始的内存中，则结果为

	big-endian	little-endian
0x0000	0x12	0xcd
0x0001	0x34	0xab
0x0002	0xab	0x34
0x0003	0xcd	0x12

目前应该 little endian 是主流，因为在数据类型转换的时候（尤其是指针转换）不用考虑地址问题。

二、Big Endian 和 Little Endian 名词的由来

这两个术语来自于 Jonathan Swift 的《《格利佛游记》其中交战的两个派别无法就应该从哪一端——小端还是大端——打开一个半熟的鸡蛋达成一致。：）

“endian”这个词出自《格列佛游记》。小人国的内战就源于吃鸡蛋时是究竟从大头(Big-Endian)敲开还是从小头(Little-Endian)敲开，由此曾发生过六次叛乱，其中一个皇帝送了命，另一个丢了王位。

我们一般将 endian 翻译成“字节序”，将 big endian 和 little endian 称作“大尾”和“小尾”。

在那个时代，Swift 是在讽刺英国和法国之间的持续冲突，Danny Cohen，一位网络协议的早期开创者，第一次使用这两个术语来指代字节顺序，后来这个术语被广泛接纳了

三、Big Endian 和 Little Endian 优劣

来自：Dr. William T. Verts, April 19, 1996

Big Endian

判别一个数的正负很容易，只要取 offset0 处的一个字节就能确认。

Little Endian

长度为 1，2，4 字节的数，排列方式都是一样的，数据类型转换非常方便。

四、一些常见文件的字节序

来自：Dr. William T. Verts, April 19, 1996

Common file formats and their endian order are as follows:

- Adobe Photoshop -- Big Endian
- BMP (Windows and OS/2 Bitmaps) -- Little Endian
- DXF (AutoCad) -- Variable
- GIF -- Little Endian
- IMG (GEM Raster) -- Big Endian
- JPEG -- Big Endian
- FLI (Autodesk Animator) -- Little Endian
- MacPaint -- Big Endian
- PCX (PC Paintbrush) -- Little Endian
- PostScript -- Not Applicable (text!)
- POV (Persistence of Vision ray-tracer) -- Not Applicable (text!)
- QTM (Quicktime Movies) -- Little Endian (on a Mac!) （PeterLee 注 Big Endian in my opinion）
- Microsoft RIFF (.WAV & .AVI) -- Both
- Microsoft RTF (Rich Text Format) -- Little Endian
- SGI (Silicon Graphics) -- Big Endian
- Sun Raster -- Big Endian
- TGA (Targa) -- Little Endian
- TIFF -- Both, Endian identifier encoded into file
- WPG (WordPerfect Graphics Metafile) -- Big Endian (on a PC!)
- XWD (X Window Dump) -- Both, Endian identifier encoded into file

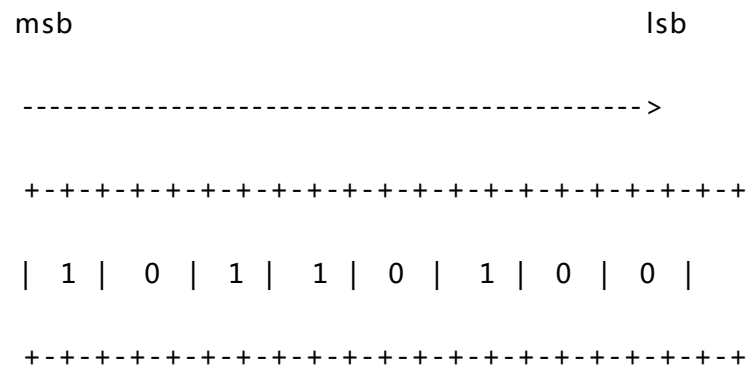
五、比特序

来自：http://ayazh.gjjblog.com/archives/1058846/

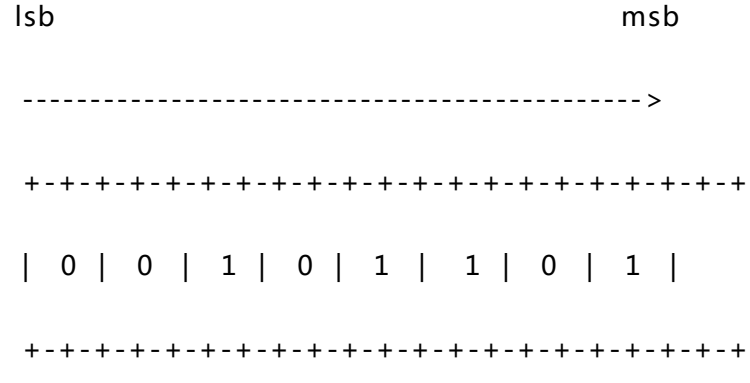
我在 8 月 9 号的《Big Endian 和 Little Endian》一文中谈了字节序的问题。可是有朋友仍然会问，CPU 存储一个字节的数据时其字节内的 8 个比特之间的顺序是否也有 big endian 和 little endian 之分？或者说是否有比特序的不同？

实际上，这个比特序是同样存在的。下面以数字 0xB4（10110100）用图加以说明。

Big Endian



Little Endian



实际上，由于 CPU 存储数据操作的最小单位是一个字节，其内部的比特序是什么样对我们的程序来说是一个黑盒子。也就是说，你给我一个指向 0xB4 这个数的指针，对于 big endian 方式的 CPU 来说，它是从左往右依次读取这个数的 8 个比特；而对于 little endian 方式的 CPU 来说，则正好相反，是从右往左依次读取这个数的 8 个比特。而我们的程序通过这个指针访问后得到的数就是 0xB4，字节内部的比特序对于程序来说是不可见的，其实这点对于单机上的字节序来说也是一样的。

那可能有人又会问，如果是网络传输呢？会不会出问题？是不是也要通过什么

函数转换一下比特序？嗯，这个问题提得很好。假设 little endian 方式的 CPU 要传给 big endian 方式 CPU 一个字节的话，其本身在传输之前会在本地就读出这个 8 比特的数，然后再按照网络字节序的顺序来传输这 8 个比特，这样的话到了接收端不会出现任何问题。而假如要传输一个 32 比特的数的话，由于这个数在 littel endian 方存储时占了 4 个字节，而网络传输是以字节为单位进行的，little endian 方的 CPU 读出第一个字节后发送，实际上这个字节是原数的 LSB，到了接收方反倒成了 MSB 从而发生混乱。