# BigEndian和LittleEndian

这里简单介绍不同计算机体系结构中表示数据的字节序（BigEndian和LittleEndian）问题。

一些基本概念：

MSB(Most Significant Bit)：高字节。

对于一个多字节的数据，高字节也就是它的高位（最低位应该是“个位”）。

LSB(Least Significant Bits) ：低字节。

对于一个多字节的数据，低字节也就是它的低位（最低位应该是“个位”）。

这里"B"是大写的表示字节，小写"b"表示位。

高地址和低地址：

对于内存的地址，从0x000000开始增加，也就是从低地址向高地址增加的过程。

BigEndian:指低地址存放最高有效字节（MSB）。又叫降序。

这里，是“正”序的体现符合我们的思维习惯。例如，我们写一个数字0x1234,从左向右写（高位->低位），那么在内存上也从小向大来存放（低->高）,所以内存中看到的就依次是12 34.

LittleEndian:是低地址存放最低有效字节（LSB）。又叫开序。

这里，是“反”序的体现和我们的思维习惯相反。例如，我们写一个数字0x1234,从左向右写（高位->低位），那么在内存上却从大向小来存放（高->低）,所以内存中看到的就依次是34 12.

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.

所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中提供了四个转换字节序的宏。

一个简单的例子：

//判断是little endian还是big endian

#include <stdio.h>

int main(int argc, char \*argv[])

{

int a = 0x11223344;

if((char)a == 0x11)

{//低地址存高字节

printf("big endian!\n");

}

printf("%x\n",(int)(char)a);

short b = 0x1122;

if((char)b == 0x22)

{//低地址存低字节

printf("little endian!\n");

}

//强转之后的地址，一定是从低开始算。

printf("%x\n",(int)(char)b);

printf("hello\n");

}

总结：

自己总结了两种记忆方式：除了前面说的那种，还一个：

Endian就是“端”，内存的端确定的就是最开始的0地址（最低地址），所以BigEndian就是大的存在低地址。大的，当然就是最高位。LittleEndian类似.

*作者：QuietHeart*

[*Email：quiet\_heart000@126.com*](mailto:Email：quiet_heart000@126.co)

*日期：2009年10月*