转至： <https://www.bysocket.com/?p=615>

**一、什么大小端？**

大小端在计算机业界，**Endian**表示数据在存储器中的存放顺序。百度百科如下叙述之：

***大端模式****，是指****数据的高字节保存在内存的低地址中，而数据的低字节保存在内存的高地址中****，这样的存储模式有点儿类似于把数据当作字符串顺序处理：地址由小向大增加，而数据从高位往低位放；*

***小端模式****，是指****数据的高字节保存在内存的高地址中，而数据的低字节保存在内存的低地址中****，这种存储模式将地址的高低和数据位权有效地结合起来，高地址部分权值高，低地址部分权值低，和我们的逻辑方法一致。*

这两种模式，泥瓦匠记忆宫殿：“**小端低低**”。这样就知道小端的模式，反之大端的模式。

比如**整形**十进制数字：305419896 ，转化为十六进制表示 : 0x12345678 。其中按着十六进制的话，每两位占8个位，及一个字节。如图

[](http://www.bysocket.com/wp-content/uploads/2015/10/iostream1.png)

**二、为什么有大小端模式之分呢？**

*如果统一使用大端或者小端，那么何来三国演义，何来一战二战呢？还有大小端也来源于战争。所以存在即是合理。*

在操作系统中，x86和一般的OS（如windows，FreeBSD,Linux）使用的是小端模式。但比如Mac OS是大端模式。

在计算机系统中，我们是以字节为单位的，每个地址单元都对应着一个字节，一个字节为8bit。但是在C语言中除了8bit的char之外，还有16bit的short型，32bit的long型（要看具体的编译器）。另外，对于位数大于8位的处理器，例如16位或者32位的处理器，由于**寄存器宽度大于一个字节**，那么必然存在着一个如果将多个字节安排的问题。因此就导致了大端存储模式和小端存储模式。

知道为什么有模式的存在，下面需要了解下具有有什么**应用场景**：

1、不同端模式的处理器进行数据传递时必须要考虑端模式的不同

2、在网络上传输数据时，由于数据传输的两端对应不同的硬件平台，采用的存储字节顺序可能不一致。所以在TCP/IP协议规定了在网络上必须采用**网络字节顺序**，也就是**大端模式**。对于char型数据只占一个字节，无所谓大端和小端。而对于**非char类型数据**，必须在数据发送到网络上之前将其转换成大端模式。接收网络数据时按符合接受主机的环境接收。

**三、java中的大小端**

存储量大于1字节，非char类型，如int，float等，要考虑字节的顺序问题了。java由于虚拟机的关系,屏蔽了大小端问题,需要知道的话可用 ByteOrder.nativeOrder() 查询。在操作ByteBuffer中，也可以使用 ByteBuffer.order() 进行设置：

**import** java.nio.ByteBuffer;

**import** java.nio.ByteOrder;

**import** java.util.Arrays;

/\*\*

\* **@author** Jeff Lee

\* **@since** 2015-10-13 20:40:00

\* ByteBuffer中字节存储次序

\*/

**public** **class** Endians {

**public** **static** **void** main(String[] args) {

// 创建12个字节的字节缓冲区

ByteBuffer bb = ByteBuffer.*wrap*(**new** **byte**[12]);

// 存入字符串

bb.asCharBuffer().put("abdcef");

System.*out*.println(Arrays.*toString*(bb.array()));

// 反转缓冲区

bb.rewind();

// 设置字节存储次序

bb.order(ByteOrder.*BIG\_ENDIAN*);

bb.asCharBuffer().put("abcdef");

System.*out*.println(Arrays.*toString*(bb.array()));

// 反转缓冲区

bb.rewind();

// 设置字节存储次序

bb.order(ByteOrder.*LITTLE\_ENDIAN*);

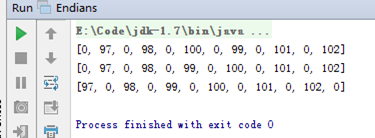
bb.asCharBuffer().put("abcdef");

System.*out*.println(Arrays.*toString*(bb.array()));

}

}

run下结果如图所示：

[](http://www.bysocket.com/wp-content/uploads/2015/10/image.png)

前两句打印说明了，ByteBuffer存储字节次序默认为大端模式。最后一段设置了字节存储次序，然后会输出，可以看出存储次序为小端模式。

转至：http://www.ruanyifeng.com/blog/2016/11/byte-order.html

# 理解字节序

作者： [阮一峰](http://www.ruanyifeng.com/)

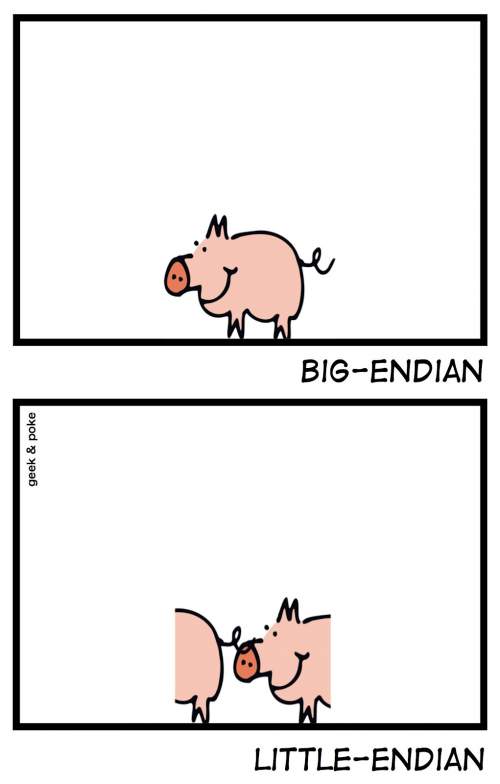
日期： [2016年11月22日](http://www.ruanyifeng.com/blog/2016/11/)

1.

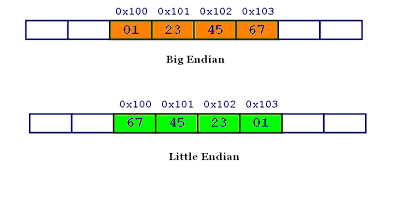
计算机硬件有两种储存数据的方式：大端字节序（big endian）和小端字节序（little endian）。

举例来说，数值0x2211使用两个字节储存：高位字节是0x22，低位字节是0x11。

* **大端字节序**：高位字节在前，低位字节在后，这是人类读写数值的方法。
* **小端字节序**：低位字节在前，高位字节在后，即以0x1122形式储存。



同理，0x1234567的大端字节序和小端字节序的写法如下图。



2.

我一直不理解，为什么要有字节序，每次读写都要区分，多麻烦！统一使用大端字节序，不是更方便吗？

上周，我读到了一篇[文章](http://blog.erratasec.com/2016/11/how-to-teach-endian.html" \t "_blank)，解答了所有的疑问。而且，我发现原来的理解是错的，字节序其实很简单。

3.

首先，为什么会有小端字节序？

答案是，计算机电路先处理低位字节，效率比较高，因为计算都是从低位开始的。所以，计算机的内部处理都是小端字节序。

但是，人类还是习惯读写大端字节序。所以，除了计算机的内部处理，其他的场合几乎都是大端字节序，比如网络传输和文件储存。

4.

计算机处理字节序的时候，不知道什么是高位字节，什么是低位字节。它只知道按顺序读取字节，先读第一个字节，再读第二个字节。

如果是大端字节序，先读到的就是高位字节，后读到的就是低位字节。小端字节序正好相反。

理解这一点，才能理解计算机如何处理字节序。

5.

字节序的处理，就是一句话：

**"只有读取的时候，才必须区分字节序，其他情况都不用考虑。"**

处理器读取外部数据的时候，必须知道数据的字节序，将其转成正确的值。然后，就正常使用这个值，完全不用再考虑字节序。

即使是向外部设备写入数据，也不用考虑字节序，正常写入一个值即可。外部设备会自己处理字节序的问题。

6.

举例来说，处理器读入一个16位整数。如果是大端字节序，就按下面的方式转成值。

x = buf[offset] \* 256 + buf[offset+1];

上面代码中，buf是整个数据块在内存中的起始地址，offset是当前正在读取的位置。第一个字节乘以256，再加上第二个字节，就是大端字节序的值，这个式子可以用逻辑运算符改写。

x = buf[offset]<<8 | buf[offset+1];

上面代码中，第一个字节左移8位（即后面添8个0），然后再与第二个字节进行或运算。

如果是小端字节序，用下面的公式转成值。

x = buf[offset+1] \* 256 + buf[offset];

32位整数的求值公式也是一样的。

/\* 大端字节序 \*/

i = (data[3]<<0) | (data[2]<<8) | (data[1]<<16) | (data[0]<<24);

/\* 小端字节序 \*/

i = (data[0]<<0) | (data[1]<<8) | (data[2]<<16) | (data[3]<<24);

（完）