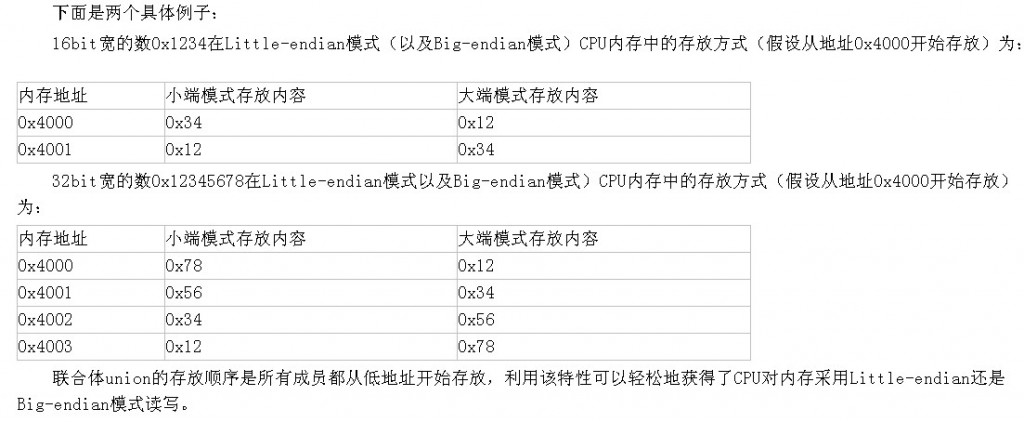
## <https://blog.csdn.net/aaronin/article/details/7700771>

开头讲个有关大端小端的故事：

　　端模式（Endian）的这个词出自Jonathan Swift书写的《格列佛游记》。这本书根据将鸡蛋敲开的方法不同将所有的人分为两类，从圆头开始将鸡蛋敲开的人被归为Big Endian，从尖头开始将鸡蛋敲开的人被归为Littile Endian。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。在计算机业Big Endian和Little Endian也几乎引起一场战争。

　　我们知道在内存中数据是以字节为单位进行存储的，每个地址单元对应着一个字节（byte），一个字节为8位（bite）。但是很多时候数据除了8bit额char外，还有16bit的short，32位的long型（要看具体的编译器），必然存在多字节安排的问题。不同的计算机存放多字节值的顺序不同，有些机器在起始地址存放低位字节（低位先存），即小端模式；有的机器在起始地址存放高位字节（高位先存），即大端模式。基于Intel的CPU，采用的是低位先存。而KEIL C51则为大端模式。大端小端对应着数据在存储器中的存放顺序。

　　同时，在网络传输中，网络协议需要指定网络字节顺序，TCP/IP协议中使用16位整数和32位整数的高位先存模式，对应我们的大端模式。



写程序判断处理器是Little-endian模式，还是Big-endian模式，可以通过以下程序：

　　1>.通过将int强制类型转换成char单字节，通过判断起始存储位置。

**[cpp]** [view plain](https://blog.csdn.net/aaronin/article/details/7700771) [copy](https://blog.csdn.net/aaronin/article/details/7700771)



**[cpp]** [view plain](https://blog.csdn.net/aaronin/article/details/7700771) [copy](https://blog.csdn.net/aaronin/article/details/7700771)

1. **void** main(**int** argc, **char** \*\*argv)
2. {
3. **int** i = 1;
5. **char** \*cp = (**char** \*)&i; //前面是指针运算符\*，前值类型转换。后面是取地址符号。
7. **if** (\*cp) //如果此时cp指向的内存为1的话，则为小端，否则为大端。
8. printf("Little Endian\n");
9. **else**
10. printf("Big Endian\n");
11. exit(EXIT\_SUCCESS);
12. }