高低字节序转换（htonl、ntohl、htons、ntohs函数）

Part 1: htons函数具体解释

在Linux和Windows网络编程时需要用到htons和htonl函数，用来将主机字节顺序转换为网络字节顺序。

在Intel机器下，执行以下程序

```
int main()

...{

printf("%d /n",htons(16));

return 0;

}
```

得到的结果是4096，初一看感觉很怪。

解释如下，数字16的16进制表示为0x0010，数字4096的16进制表示为0x1000。 由于Intel机器是小尾端，存储数字16时实际顺序为1000，存储4096时实际顺序为0010。因此在发送网络包时为了报文中数据为0010，需要经过htons进行字节转换。如果用IBM等大尾端机器，则没有这种字节

顺序转换，但为了程序的可移植性，也最好用这个函数。

另外用注意，数字所占位数小于或等于一个字节（8 bits）时，不要用htons转换。这是因为对于主机来说，大小尾端的最小单位为字节(byte)。

Part 2: 大小端模式

不同的CPU有不同的字节序类型 这些字节序是指整数在内存中保存的顺序 这个叫做主机序

最常见的有两种

1． Little endian：将低序字节存储在起始地址

2． Big endian：将高序字节存储在起始地址

LE little-endian

最符合人的思维的字节序

地址低位存储值的低位

地址高位存储值的高位

怎么讲是最符合人的思维的字节序，是因为从人的第一观感来说

低位值小，就应该放在内存地址小的地方，也即内存地址低位

反之，高位值就应该放在内存地址大的地方，也即内存地址高位

BE big-endian

最直观的字节序

地址低位存储值的高位

地址高位存储值的低位

为什么说直观，不要考虑对应关系

只需要把内存地址从左到右按照由低到高的顺序写出

把值按照通常的高位到低位的顺序写出

两者对照，一个字节一个字节的填充进去

例子：在内存中双字0x01020304(DWORD)的存储方式

内存地址

4000 4001 4002 4003

LE 04 03 02 01

BE 01 02 03 04

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 0x23 0xab

0x0002 0xab 0x34

0x0003 0xcd 0x12

x86系列CPU都是little-endian的字节序.

网络字节顺序是TCP/IP中规定好的一种数据表示格式，它与具体的CPU类型、操作系统等无关，从而可以保证数据在不同主机之间传输时能够被正确解释。网络字节顺序采用big endian排序方式。

为了进行转换 bsd socket提供了转换的函数 有下面四个

htons 把unsigned short类型从主机序转换到网络序

htonl 把unsigned long类型从主机序转换到网络序

ntohs 把unsigned short类型从网络序转换到主机序

ntohl 把unsigned long类型从网络序转换到主机序

在使用little endian的系统中 这些函数会把字节序进行转换

在使用big endian类型的系统中 这些函数会定义成空宏

同样 在网络程序开发时 或是跨平台开发时 也应该注意保证只用一种字节序 不然两方的解释不一样就会产生bug.

注:

1、网络与主机字节转换函数:htons ntohs htonl ntohl (s 就是short l是long h是host n是network)

2、不同的CPU上运行不同的操作系统，字节序也是不同的，参见下表。

处理器 操作系统 字节排序

Alpha 全部 Little endian

HP-PA NT Little endian

HP-PA　UNIX　Big endian

Intelx86　全部　Little endian <-----x86系统是小端字节序系统

Motorola680x()　全部　Big endian

MIPS　NT　Little endian

MIPS　UNIX　Big endian

PowerPC　NT　Little endian

PowerPC　非NT　Big endian <-----PPC系统是大端字节序系统

RS/6000　UNIX　Big endian

SPARC　UNIX　Big endian

IXP1200 ARM核心　全部　Little endian

本文来自CSDN博客，转载请标明出处：http://blog.csdn.net/zouxinfox/archive/2007/10/07/1814088.aspx

Part 3: 模拟htonl、ntohl、htons、ntohs函数实现

今天在如鹏网里讨论htonl、ntohl在不同机器的区别，特意模拟了htonl、ntohl、htons、ntohs函数实现。

实现如下：

```
typedef unsigned short int uint16;

typedef unsigned long int uint32;

// 短整型大小端互换

#define BigLittleSwap16(A) ((((uint16)(A) & 0xff00) >> 8) | /

(((uint16)(A) & 0x00ff) << 8))

// 长整型大小端互换

#define BigLittleSwap32(A) ((((uint32)(A) & 0xff000000) >> 24) | /

(((uint32)(A) & 0x00ff0000) >> 8) | /

(((uint32)(A) & 0x0000ff00) << 8) | /

(((uint32)(A) & 0x000000ff) << 24))
```

```
// 本机大端返回1，小端返回0
int checkCPUendian()
{
union{
unsigned long int i;
unsigned char s[4];
}c;
c.i = 0x12345678;
return (0x12 == c.s[0]);
}
// 模拟htonl函数，本机字节序转网络字节序
unsigned long int HtoNl(unsigned long int h)
{
// 若本机为大端，与网络字节序同，直接返回
```

```
// 若本机为小端，转换成大端再返回
return checkCPUendian() ? h : BigLittleSwap32(h);
}
// 模拟ntohl函数，网络字节序转本机字节序
unsigned long int NtoHl(unsigned long int n)
{
// 若本机为大端，与网络字节序同，直接返回
// 若本机为小端，网络数据转换成小端再返回
return checkCPUendian() ? n : BigLittleSwap32(n);
}
// 模拟htons函数，本机字节序转网络字节序
unsigned short int HtoNs(unsigned short int h)
{
// 若本机为大端，与网络字节序同，直接返回
```

```
// 若本机为小端，转换成大端再返回
return checkCPUendian() ? h : BigLittleSwap16(h);
}
// 模拟ntohs函数，网络字节序转本机字节序
unsigned short int NtoHs(unsigned short int n)
{
// 若本机为大端，与网络字节序同，直接返回
// 若本机为小端，网络数据转换成小端再返回
return checkCPUendian() ? n : BigLittleSwap16(n);
}
```