**JAVA网络字节序转换**

字节序指的是数据存储的顺序问题，分为Big-Endian和Little-Endian，Big-Endian指的是数据中的高位存储在存储器的低位，Little-Endian正好相反。Big-Endian也叫大头在前，Little-Endian叫做小头在前。  
举例而言，有一个4字节的数据0x01020304，要存储在内存中或文件中编号0~3字节的位置，两种字节序的排列方式分别如下：  
Big Endian  
 低地址                           高地址  
 ----------------------------------------->  
地址编号  
|     0      |      1     |     2       |      3    |  
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  
 |     01     |      02    |     03      |     04    |  
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  
   
Little Endian  
   
 低地址                           高地址  
 ----------------------------------------->  
地址编号  
|     0      |      1     |     2       |      3    |  
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  
  |     04     |      03    |     02      |     01    |  
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  
   
如果按照我们一般的书写习惯来说，Big-Endian似乎更好一点。  
具体应用中是用big还是little就要取决于系统平台了。不同的CPU会有不同的情况，比如说x86架构的cpu使用的是little模式，而PowerPC则是big。C/C++语言对数据的处理取决于其运行的CPU，而Java则统一使用big模式。  
如果单纯仅是使用Java的话也就不存在大小字节序的处理问题，但有时候Java程序可能还要使用到C/C++这类语言生成的数据，尤其是在我们常用x86环境下，两种语言的字节序就会出现不一致的问题，必须做一定的处理。  
处理的方法很简单，就是把数据按字节前后调换一下就可以了。  
在这里是将长度为len的数据转换成字节数组str[len]，带入以下函数，如果原始数据是little，也就是big=false的情况下，函数将对数据进行高低位的调换，并返回。  
publicstaticbyte[] ReversEndian(byte str[],int len, boolean big)  
    {  
       byte b;  
       byte res[]=newbyte[len];  
       for(int i=0;i<len;i++)  
       {  
           res[i]=str[i];             
       }  
       if(big==false)  
       {  
           for(int i=0;i<len;i++)  
           {  
              b=str[i];  
              res[len-i-1]=b;               
           }  
       }        
       return res;  
    }  
   
假设现在我们需要从一个C/C++语言生成的二进制文件中读出一个float数据，可以如下处理。  
//文件名  
String fileName="file";  
//float变量是4字节长  
byte b[]=newbyte[4];  
ByteBuffer bb;  
File file=new File(fileName);  
FileInputStream fis=new FileInputStream(file);  
//从文件中读取一个4字节数据  
fis.read(b);  
//反转  
b=ReversEndian(b,4,false);  
//将字节数组还原成浮点数值  
bb=ByteBuffer.wrap(b);  
float value=bb.getFloat();

/\*\*    
\* 通信格式转换    
\*    
\* Java 和一些windows编程语言如c、c++、delphi所写的网络程序进行通讯时，需要进行相应的转换    
\* 高、低字节之间的转换    
\* windows的字节序为低字节开头    
\* linux,unix的字节序为高字节开头    
\* java则无论平台变化，都是高字节开头    
   \*/      
     
     
public   class  FormatTransfer {    
/\*\*    
   \* 将 int转为低字节在前，高字节在后的byte数组    
   \* @param n int    
   \* @return byte[]    
   \*/     
public   static   byte [] toLH( int  n) {    
   byte [] b =  new   byte [ 4 ];    
   b[0 ] = ( byte ) (n &  0xff );    
   b[1 ] = ( byte ) (n >>  8  &  0xff );    
   b[2 ] = ( byte ) (n >>  16  &  0xff );    
   b[3 ] = ( byte ) (n >>  24  &  0xff );    
   return  b;    
}     
     
     
/\*\*    
   \* 将 int转为高字节在前，低字节在后的byte数组    
   \* @param n int    
   \* @return byte[]    
   \*/     
public   static   byte [] toHH( int  n) {    
   byte [] b =  new   byte [ 4 ];    
   b[3 ] = ( byte ) (n &  0xff );    
   b[2 ] = ( byte ) (n >>  8  &  0xff );    
   b[1 ] = ( byte ) (n >>  16  &  0xff );    
   b[0 ] = ( byte ) (n >>  24  &  0xff );    
   return  b;    
}     
     
/\*\*    
   \* 将 short转为低字节在前，高字节在后的byte数组    
   \* @param n short    
   \* @return byte[]    
   \*/     
public   static   byte [] toLH( short  n) {    
   byte [] b =  new   byte [ 2 ];    
   b[0 ] = ( byte ) (n &  0xff );    
   b[1 ] = ( byte ) (n >>  8  &  0xff );    
   return  b;    
}  
  
     
/\*\*    
   \* 将 short转为高字节在前，低字节在后的byte数组    
   \* @param n short    
   \* @return byte[]    
   \*/     
public   static   byte [] toHH( short  n) {    
   byte [] b =  new   byte [ 2 ];    
   b[1 ] = ( byte ) (n &  0xff );    
   b[0 ] = ( byte ) (n >>  8  &  0xff );    
   return  b;    
}     
     
     
     
/\*\*    
   \* 将将int转为高字节在前，低字节在后的byte数组     
public static byte[] toHH(int number) {    
   int temp = number;    
   byte[] b = new byte[4];    
   for (int i = b.length - 1; i > -1; i--) {    
     b = new Integer(temp & 0xff).byteValue();    
     temp = temp >> 8;    
   }    
   return b;    
}     
public static byte[] IntToByteArray(int i) {    
     byte[] abyte0 = new byte[4];    
     abyte0[3] = (byte) (0xff & i);    
     abyte0[2] = (byte) ((0xff00 & i) >> 8);    
     abyte0[1] = (byte) ((0xff0000 & i) >> 16);    
     abyte0[0] = (byte) ((0xff000000 & i) >> 24);    
     return abyte0;    
}     
\*/      
     
/\*\*    
   \* 将 float转为低字节在前，高字节在后的byte数组    
   \*/     
public   static   byte [] toLH( float  f) {    
   return  toLH(Float.floatToRawIntBits(f));    
}     
/\*\*    
   \* 将 float转为高字节在前，低字节在后的byte数组    
   \*/     
public   static   byte [] toHH( float  f) {    
   return  toHH(Float.floatToRawIntBits(f));    
}  
  
     
/\*\*    
   \* 将 String转为byte数组    
   \*/     
public   static   byte [] stringToBytes(String s,  int  length) {    
   while  (s.getBytes().length < length) {    
     s += " " ;    
   }    
   return  s.getBytes();    
}     
     
     
/\*\*    
   \* 将字节数组转换为String    
   \* @param b byte[]    
   \* @return String    
   \*/     
public   static  String bytesToString( byte [] b) {    
   StringBuffer result = new  StringBuffer( "" );    
   int  length = b.length;    
   for  ( int  i= 0 ; i< length; i++) {    
     result.append((char )(b[i] &  0xff ));    
   }    
   return  result.toString();    
}  
  
     
/\*\*    
   \* 将字符串转换为byte数组    
   \* @param s String    
   \* @return byte[]    
   \*/     
public   static   byte [] stringToBytes(String s) {    
   return  s.getBytes();    
}  
  
     
/\*\*    
   \* 将高字节数组转换为int    
   \* @param b byte[]    
   \* @return int    
   \*/     
public   static   int  hBytesToInt( byte [] b) {    
   int  s =  0 ;    
   for  ( int  i =  0 ; i <  3 ; i++) {    
     if  (b[i] >=  0 ) {    
     s = s + b[i];    
     } else  {    
     s = s + 256  + b[i];    
     }    
     s = s \* 256 ;    
   }    
   if  (b[ 3 ] >=  0 ) {    
     s = s + b[3 ];    
   } else  {    
     s = s + 256  + b[ 3 ];    
   }    
   return  s;    
}  
  
     
/\*\*    
   \* 将低字节数组转换为int    
   \* @param b byte[]    
   \* @return int    
   \*/     
public   static   int  lBytesToInt( byte [] b) {    
   int  s =  0 ;    
   for  ( int  i =  0 ; i <  3 ; i++) {    
     if  (b[ 3 -i] >=  0 ) {    
     s = s + b[3 -i];    
     } else  {    
     s = s + 256  + b[ 3 -i];    
     }    
     s = s \* 256 ;    
   }    
   if  (b[ 0 ] >=  0 ) {    
     s = s + b[0 ];    
   } else  {    
     s = s + 256  + b[ 0 ];    
   }    
   return  s;    
}     
     
     
/\*\*    
   \* 高字节数组到short的转换    
   \* @param b byte[]    
   \* @return short    
   \*/     
public   static   short  hBytesToShort( byte [] b) {    
   int  s =  0 ;    
   if  (b[ 0 ] >=  0 ) {    
     s = s + b[0 ];    
     } else  {    
     s = s + 256  + b[ 0 ];    
     }    
     s = s \* 256 ;    
   if  (b[ 1 ] >=  0 ) {    
     s = s + b[1 ];    
   } else  {    
     s = s + 256  + b[ 1 ];    
   }    
   short  result = ( short )s;    
   return  result;    
}  
  
     
/\*\*    
   \* 低字节数组到short的转换    
   \* @param b byte[]    
   \* @return short    
   \*/     
public   static   short  lBytesToShort( byte [] b) {    
   int  s =  0 ;    
   if  (b[ 1 ] >=  0 ) {    
     s = s + b[1 ];    
     } else  {    
     s = s + 256  + b[ 1 ];    
     }    
     s = s \* 256 ;    
   if  (b[ 0 ] >=  0 ) {    
     s = s + b[0 ];    
   } else  {    
     s = s + 256  + b[ 0 ];    
   }    
   short  result = ( short )s;    
   return  result;    
}  
  
     
/\*\*    
   \* 高字节数组转换为float    
   \* @param b byte[]    
   \* @return float    
   \*/     
public   static   float  hBytesToFloat( byte [] b) {    
   int  i =  0 ;    
   Float F = new  Float( 0.0 );    
   i = ((((b[0 ]& 0xff )<< 8  | (b[ 1 ]& 0xff ))<< 8 ) | (b[ 2 ]& 0xff ))<< 8  | (b[ 3 ]& 0xff );    
   return  F.intBitsToFloat(i);    
}  
  
     
/\*\*    
   \* 低字节数组转换为float    
   \* @param b byte[]    
   \* @return float    
   \*/     
public   static   float  lBytesToFloat( byte [] b) {    
   int  i =  0 ;    
   Float F = new  Float( 0.0 );    
   i = ((((b[3 ]& 0xff )<< 8  | (b[ 2 ]& 0xff ))<< 8 ) | (b[ 1 ]& 0xff ))<< 8  | (b[ 0 ]& 0xff );    
   return  F.intBitsToFloat(i);    
}  
  
     
/\*\*    
   \* 将 byte数组中的元素倒序排列    
   \*/     
public   static   byte [] bytesReverseOrder( byte [] b) {    
   int  length = b.length;    
   byte [] result =  new   byte [length];    
   for ( int  i= 0 ; i< length; i++) {    
     result[length-i-1 ] = b[i];    
   }    
   return  result;    
}  
  
     
/\*\*    
   \* 打印byte数组    
   \*/     
public   static   void  printBytes( byte [] bb) {    
   int  length = bb.length;    
   for  ( int  i= 0 ; i< length; i++) {    
     System.out.print(bb + " " );    
   }    
   System.out.println("" );    
}     
     
public   static   void  logBytes( byte [] bb) {    
   int  length = bb.length;    
   String out = "" ;    
   for  ( int  i= 0 ; i< length; i++) {    
     out = out + bb + " " ;    
   }     
     
}     
     
/\*\*    
   \* 将 int类型的值转换为字节序颠倒过来对应的int值    
   \* @param i int    
   \* @return int    
   \*/     
public   static   int  reverseInt( int  i) {    
   int  result = FormatTransfer.hBytesToInt(FormatTransfer.toLH(i));    
   return  result;    
}     
     
/\*\*    
   \* 将 short类型的值转换为字节序颠倒过来对应的short值    
   \* @param s short    
   \* @return short    
   \*/     
public   static   short  reverseShort( short  s) {    
   short  result = FormatTransfer.hBytesToShort(FormatTransfer.toLH(s));    
   return  result;    
}     
     
/\*\*    
   \* 将 float类型的值转换为字节序颠倒过来对应的float值    
   \* @param f float    
   \* @return float    
   \*/     
public   static   float  reverseFloat( float  f) {    
   float  result = FormatTransfer.hBytesToFloat(FormatTransfer.toLH(f));    
   return  result;    
}     
   }