**Unicode字符集和多字节字符集关系**

在计算机中字符通常并不是保存为图像，每个字符都是使用一个编码来表示的，而每个字符究竟使用哪个编码代表，要取决于使用哪个字符集(charset)。   
在最初的时候，Internet上只有一种字符集——ANSI的ASCII字符集，它使用7 bits来表示一个字符，总共表示128个字符，其中包括了英文字母、数字、标点符号等常用字符。之后，又进行扩展，使用8 bits表示一个字符，可以表示256个字符，主要在原来的7 bits字符集的基础上加入了一些特殊符号例如制表符。   
后来，由于各国语言的加入，ASCII已经不能满足信息交流的需要，因此，为了能够表示其它国家的文字，各国在ASCII的基础上制定了自己的字符集，这些从ANSI标准派生的字符集被习惯的统称为**ANSI字符集，它们正式的名称应该是MBCS(Multi-Byte Chactacter System**，即多字节字符系统)。这些派生字符集的特点是以ASCII 127 bits为基础，兼容ASCII 127，他们**使用大于128的编码作为一个Leading Byte，紧跟在Leading Byte后的第二（甚至第三）个字符与Leading Byte一起作为实际的编码**。这样的字符集有很多，我们常见的GB-2312就是其中之一。   
例如在GB-2312字符集中，“连通”的编码为C1 AC CD A8，其中C1和CD就是Leading Byte。前127个编码为标准ASCII保留，例如“0”的编码是30H（30H表示十六进制的30）。软件在读取时，如果看到30H，知道它小于128就是标准ASCII，表示“0”，看到C1大于128就知道它后面有一个另外的编码，因此C1 AC一同构成一个整个的编码，在GB-2312字符集中表示“连”。   
由于每种语言都制定了自己的字符集，导致最后存在的各种字符集实在太多，在国际交流中要经常转换字符集非常不便。因此，提出了Unicode字符集，它**固定使用16 bits（两个字节、一个字）来表示一个字符，共可以表示65536个字符**。将世界上几乎所有语言的常用字符收录其中，方便了信息交流。标准的Unicode称为UTF-16。后来**为了双字节的Unicode能够在现存的处理单字节的系统上正确传输，出现了UTF-8**，使用类似MBCS的方式对Unicode进行编码。注意**UTF-8是编码，它属于Unicode字符集**。Unicode字符集有多种编码形式，而ASCII只有一种，大多数MBCS（包括GB-2312）也只有一种。Unicode的最初目标，是用1个16位的编码来为超过65000字符提供映射。但这还不够，它不能覆盖全部历史上的文字，也不能解决传输的问题 (implantation head-ache's)，尤其在**那些基于网络的应用中。已有的软件必须做大量的工作来程序16位的数据**。因此，**Unicode用一些基本的保留字符制定了三套编码方式。它们分别是UTF-8,UTF-16和UTF-32**。正如名字所示，在UTF－8中，字符是以8位序列来编码的，用一个或几个字节来表示一个字符。这种方式的最大好处，是UTF－8保留了ASCII字符的编码做为它的一部分，例如，在UTF－8和ASCII中，“A”的编码都是0x41.UTF－16和UTF－32分别是Unicode的16位和32位编码方式。考虑到最初的目的，通常说的Unicode就是指UTF-16。   
例如“连通”两个字的Unicode标准编码UTF-16 (big endian)为：DE 8F 1A 90   
而其UTF-8编码为：E8 BF 9E E9 80 9A   
最后，当一个软件打开一个文本时，它要做的第一件事是决定这个文本究竟是使用哪种字符集的哪种编码保存的。软件有三种途径来决定文本的字符集和编码：   
最标准的途径是检测文本最开头的几个字节，如下表：   
  
开头字节 Charset/encoding   
**EF BB BF UTF-8   
FE FF UTF-16/UCS-2, little endian   
FF FE UTF-16/UCS-2, big endian   
FF FE 00 00 UTF-32/UCS-4, little endian.**  
00 00 FE FF UTF-32/UCS-4, big-endian.例如插入标记后，连通”两个字的UTF-16 (big endian)和UTF-8码分别为：   
FF FE DE 8F 1A 90   
EF BB BF E8 BF 9E E9 80 9A   
但是MBCS文本没有这些位于开头的字符集标记，更不幸的是，一些早期的和一些设计不良的软件在保存Unicode文本时不插入这些位于开头的字符集标记。因此，软件不能依赖于这种途径。这时，软件可以采取一种比较安全的方式来决定字符集及其编码，那就是弹出一个对话框来请示用户，例如将那个“连通”文件拖到MS Word中，Word就会弹出一个对话框。   
如果软件不想麻烦用户，或者它不方便向用户请示，那它只能采取自己“猜”的方法，软件可以根据整个文本的特征来猜测它可能属于哪个charset，这就很可能不准了。使用记事本打开那个“连通”文件就属于这种情况。   
我们可以证明这一点：在记事本中键入“连通”后，选择“Save As”，会看到最后一个下拉框中显示有“ANSI”，这时保存。当再当打开“连通”文件出现乱码后，再点击“File”->“Save As”，会看到最后一个下拉框中显示有“UTF-8”，这说明记事本认为当前打开的这个文本是一个UTF-8编码的文本。而我们刚才保存时是用ANSI字符集保存的。这说明，记事本猜测了“连通”文件的字符集，认为它更像一个UTF-8编码文本。这是因为“连通”两个字的GB-2312编码看起来更像UTF-8编码导致的，这是一个巧合，不是所有文字都这样。可以使用记事本的打开功能，在打开“连通”文件时在最后一个下拉框中选择ANSI，就能正常显示了。反过来，如果之前保存时保存为UTF-8编码，则直接打开也不会出现问题。   
如果将“连通”文件放入MS Word中，Word也会认为它是一个UTF-8编码的文件，但它不能确定，因此会弹出一个对话框询问用户，这时选择“简体中文（GB2312）”，就能正常打开了。记事本在这一点上做得比较简化罢了，这与这个程序的定位是一致的。   
  
需要提醒大家的是，部分Windows 2000字型无法显示所有的Unicode字符。如果发现文件中缺少了某些字符，只需将其变更为其它字型即可。   
  
big endian和little endian   
  
big endian和little endian是CPU处理多字节数的不同方式。**例如“汉”字的Unicode编码是6C49。那么写到文件里时，究竟是将6C写在前面，还是将49写在前面？如果将6C写在前面，就是big endian。还是将49写在前面，就是little endian。**  
  
“endian”这个词出自《格列佛游记》。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，由此曾发生过六次叛乱，其中一个皇帝送了命，另一个丢了王位。   
  
我们一般将endian翻译成“字节序”，将big endian和little endian称作“大尾”和“小尾”。   
  
Unicode big endian：在Big-endian处理器（如苹果Macintosh电脑）上建立的Unicode文件中的文字位元组（存放单位）排列顺序，与在Intel处理器上建立的文件的文字位元组排列顺序相反。最重要的位元组拥有最低的地址，且会先储存文字中较大的一端。为使这类电脑的用户能够存取你的文件，可选择Unicode big-endian格式。

**#####################################################################**

**ANSI字符，UNICODE，宽字符，窄字符,多字节字符集**

Unicode ：宽字节字符集   
1. 如何取得一个既包含单字节字符又包含双字节字符的字符串的字符个数？   
可以调用Microsoft Visual C++的运行期库包含函数\_mbslen来操作多字节（既包括单字节也包括双字节）字符串。   
**调用strlen函数，无法真正了解字符串中究竟有多少字符，它只能告诉你到达结尾的0之前有多少个字节**。   
2. 如何对DBCS（双字节字符集）字符串进行操作？   
函数 描述   
PTSTR CharNext （ LPCTSTR ）; 返回字符串中下一个字符的地址   
PTSTR CharPrev （ LPCTSTR, LPCTSTR ）； 返回字符串中上一个字符的地址   
BOOL IsDBCSLeadByte( BYTE )； 如果该字节是DBCS字符的第一个字节，则返回非0值   
3. 为什么要使用Unicode？   
（1） 可以很容易地**在不同语言之间进行数据交换**。   
（2） 使你能够**分配支持所有语言的单个二进制.exe文件或DLL文件**。   
（3） 提高应用程序的**运行效率**。   
Windows 2000是使用Unicode从头进行开发的，如果调用任何一个Windows函数并给它传递一个ANSI字符串，那么系统首先要将字符串转换成Unicode，然后将Unicode字符串传递给操作系统。如果希望函数返回ANSI字符串，系统就会首先将Unicode字符串转换成ANSI字符串，然后将结果返回给你的应用程序。进行这些字符串的转换需要占用系统的时间和内存。通过从头开始用Unicode来开发应用程序，就能够使你的应用程序更加有效地运行。   
**Windows CE 本身就是使用Unicode的一种操作系统，完全不支持ANSI Windows函数**  
Windows 98 只支持ANSI，只能为ANSI开发应用程序。   
Microsoft公司将COM从16位Windows转换成Win32时，公司决定需要字符串的所有COM接口方法都只能接受Unicode字符串。   
4. 如何编写Unicode源代码？   
Microsoft公司为Unicode设计了WindowsAPI，这样，可以尽量减少代码的影响。实际上，可以编写单个源代码文件，以便使用或者不使用Unicode来对它进行编译。只需要**定义两个宏（UNICODE和\_UNICODE），就可以修改然后重新编译该源文件。   
\_UNICODE宏用于C运行期头文件，而UNICODE宏则用于Windows头文件。当编译源代码模块时，通常必须同时定义这两个宏。**5. Windows定义的Unicode数据类型有哪些？   
数据类型 说明   
WCHAR Unicode字符   
PWSTR 指向Unicode字符串的指针   
PCWSTR 指向一个恒定的Unicode字符串的指针   
对应的ANSI数据类型为CHAR，LPSTR和LPCSTR。   
ANSI/Unicode通用数据类型为TCHAR，PTSTR,LPCTSTR。   
6. 如何对Unicode进行操作？   
字符集 特性 实例   
**ANSI 操作函数以str开头 strcpy   
Unicode 操作函数以wcs开头 wcscpy   
MBCS 操作函数以\_mbs开头 \_mbscpy**  
ANSI/Unicode 操作函数以\_tcs开头 \_tcscpy**（C运行期库）**ANSI/Unicode 操作函数以lstr开头 lstrcpy**（Windows函数）**  
所有新的和未过时的函数在Windows2000中都同时拥有ANSI和Unicode两个版本。ANSI版本函数结尾以A表示；Unicode版本函数结尾以W表示。Windows会如下定义：   
#ifdef UNICODE   
#define CreateWindowEx CreateWindowExW   
#else   
#define CreateWindowEx CreateWindowExA   
#endif // !UNICODE   
7. 如何表示Unicode字符串常量？   
字符集 实例   
ANSI “string”   
Unicode L“string”   
ANSI/Unicode T(“string”)或\_TEXT(“string”)if( szError[0] == \_TEXT(‘J’) ){ }   
8. 为什么应当尽量使用操作系统函数？   
这将有助于稍稍提高应用程序的运行性能，因为操作系统字符串函数常常被大型应用程序比如操作系统的外壳进程Explorer.exe所使用。由于这些函数使用得很多，因此，在应用程序运行时，它们可能已经被装入RAM。   
如：StrCat，StrChr，StrCmp和StrCpy等。   
9.**如何编写符合ANSI和Unicode的应用程序？   
（1） 将文本串视为字符数组，而不是chars数组或字节数组。   
（2） 将通用数据类型（如TCHAR和PTSTR）用于文本字符和字符串。   
（3） 将显式数据类型（如BYTE和PBYTE）用于字节、字节指针和数据缓存。   
（4） 将TEXT宏用于原义字符和字符串。   
（5） 执行全局性替换（例如用PTSTR替换PSTR）。   
（6） 修改字符串运算问题。例如函数通常希望在字符中传递一个缓存的大小，而不是字节。**这意味着不应该传递sizeof(szBuffer),而应该传递（sizeof(szBuffer)/sizeof(TCHAR)。另外，如果需要为字符串分配一个内存块，并且拥有该字符串中的字符数目，那么请记住要按字节来分配内存。这就是说，应该调用malloc(nCharacters \*sizeof(TCHAR)),而不是调用malloc(nCharacters)。

10. 如何对字符串进行有选择的比较？   
通过调用CompareString来实现。   
标志 含义   
NORM\_IGNORECASE 忽略字母的大小写   
NORM\_IGNOREKANATYPE 不区分平假名与片假名字符   
NORM\_IGNORENONSPACE 忽略无间隔字符   
NORM\_IGNORESYMBOLS 忽略符号   
NORM\_IGNOREWIDTH 不区分单字节字符与作为双字节字符的同一个字符   
SORT\_STRINGSORT 将标点符号作为普通符号来处理   
11. 如何判断一个文本文件是ANSI还是Unicode？   
判断如果文本文件的开头**两个字节是0xFF和0xFE，那么就是Unicode**，否则是ANSI。   
12. 如何判断一段字符串是ANSI还是Unicode？   
用IsTextUnicode进行判断。IsTextUnicode使用一系列统计方法和定性方法，以便猜测缓存的内容。由于这不是一种确切的科学方法，因此 IsTextUnicode有可能返回不正确的结果。   
13. 如何在Unicode与ANSI之间转换字符串？   
**Windows函数MultiByteToWideChar用于将多字节字符串转换成宽字符串；函数WideCharToMultiByte将宽字符串转换成等价的多字节字符串。**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

UCS,UNICODE和UTF-8

UCS,UNICODE和UTF-8

本文简单介绍UCS,UNICODE和UTF-8，并利用C语言实现了UTF-8与UCS2之间的互相转化。  
1.什么是UCS和ISO10646？  
国际标准ISO10646定义了通用字符集(Universal Character Set, UCS). UCS是所有其它字符集标准的一个超集，它保证也其它字符集双向兼容，即编码间相互转换不会丢失任何信息。UCS字符集U+0000到U+007F与US-ASCII是一致的。   
2.什么是UNICODE  
历史上, 有两个独立的, 创立单一字符集的尝试. 一个是国际标准化组织(ISO)的 ISO 10646 项目, 另一个是由(一开始大多是美国的)多语言软件制造商组成的协会组织的 Unicode 项目. 幸运的是, 1991年前后, 两个项目的参与者都认识到, 世界不需要两个不同的单一字符集. 它们合并双方的工作成果, 并为创立一个单一编码表而协同工作. 两个项目仍都存在并独立地公布各自的标准, 但 Unicode 协会和 ISO/IEC JTC1/SC2 都同意保持 Unicode 和 ISO 10646 标准的码表兼容, 并紧密地共同调整任何未来的扩展.  
3.什么是UTF-8(**一种传送和存储格式**)  
**UCS和UNICODE为每个字符分配了一个对应的整数，但并没有明确说明其实现机制.故存在多种编码方式,**其中以**两个字节和四个字节来存储一个字符的方法分别叫UCS-2, UCS-4**,要将一个ASCII文件转换成一个UCS-2文件只要在每个字节前加一个字节0X00,转换成UCS-4只要在每个字节前加三个0X00。  
而**internet上大量的信息是以ASCII码存在的，如果都用两个字节来存储将浪费大量的资源,同时Unix和Linux下使用USC-2和USC-4会导致严重问题，于是出现了UTF-8(定义于ISO10646-1).**UTF-8

(UTF-8 stands for Unicode Transformation Format-8. It is an octet (8-bit) lossless encoding of Unicode characters.)  
**UNICODE(UCS)和UTF-8的对应关系**。  
**U-00000000 - U-0000007F**: **0**xxxxxxx （ASCII码 最重用）  
U-00000080 - U-000007FF: 110xxxxx 10xxxxxx （第二优先级常用）  
U-00000800 - U-0000FFFF: 1110xxxx 10xxxxxx 10xxxxxx   
U-00010000 - U-001FFFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx   
U-00200000 - U-03FFFFFF: 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx   
**U-04000000 - U-7FFFFFFF**: **1111110**x **10**xxxxxx **10**xxxxxx **10**xxxxxx **10**xxxxxx **10**xxxxxx （很少用）  
在多字节串中，第一个字节的开头**‘1’的数目就是整个串中字节的数目**.  
下面举**UCS-2与UTF-8的对应关系**,并利用C语言实现之间的互相转化。  
-----------------------------------------------------------------------------------------------------------  
| UCS2 | UTF-8 |  
|----------------------------------------------------------------------------------------------------------  
| | code | 1st Byte | 2nd byte | 3rd Byte |  
|--------------------------------------------------------------------------------------------------------  
| 000000000**aaaaaaa** | 0000 - 007F | 0**aaaaaaa** | | |  
|--------------------------------------------------------------------------------------------------------  
| 00000**bbbbbaaaaaa**| 0080 - 07FF | 110**bbbbb**| 10**aaaaaa**| |  
|--------------------------------------------------------------------------------------------------------  
| **ccccbbbbbbaaaaaa** | 0800 - FFFF | 1110cccc | 10bbbbbb | 10aaaaaa |  
|--------------------------------------------------------------------------------------------------------

alphajay的疑问： 这里是a b c应该是从0 1中取的一个bit位吧

因为USC-2 每个字符用两个字节 16bit表示

在这儿我只实现了**单个字符**的转换，串的转换也是一样的道理。  
1，把一个UTF-8字符转换成一个UCS-2字符。  
如果转换成功返回1，如果UTF-8字符是一个unrecognized字符，则返回0，且存一个blackbox(U+22e0)到ucs2\_code\_ptr中。  
  
typedef unsigned short UINT16;  
typedef unsigned char UINT8;  
typedef unsigned char BOOL;  
#define TRURE (BOOL)(1)  
#define FALSE (BOOL)(0)  
BOOL UTF8toUCS2Code(const UINT8\* utf8\_code, UINT16\* ucs2\_code){  
UINT16 temp1, temp2;  
BOOL is\_unrecognized = FALSE ;  
UINT16 \* in = utf8\_code;  
if(!utf8\_code || !ucs2\_code){  
return is\_unrecognized;  
}  
  
if(0x00 == (\*in & 0x80)){  
/\* 1 byte UTF-8 Charater.\*/  
\*ucs2\_code= (UINT16)\*in;  
is\_unrecognized = TRUE;  
}  
else if(0xc0 == (\*in & 0xe0) &&  
0x80 == (\*(in + 1) & 0xc0)  
){  
/\* 2 bytes UTF-8 Charater.\*/   
temp1 = (UINT16)(\*in & 0x1f);  
temp1 <<= 6;  
temp1 |= (UINT16)(\*(in + 1) & 0x3f);  
\*ucs2\_code = temp1;  
is\_unrecognized = TRUE;  
}  
else if( 0xe0 == (\*in & 0xf0) &&  
0x80 == (\*(in +1) & 0xc0) &&  
0x80 == (\*(in + 2) & 0xc0)  
){  
/\* 3bytes UTF-8 Charater.\*/  
temp1 = (UINT16)(\*in &0x0f);  
temp1 <<= 12;  
temp2 = (UINT16)(\*(in+1) & 0x3F);  
temp2 <<= 6;  
temp1 = temp1 | temp2 | (UINT16)(\*(in+2) & 0x3F);  
\*ucs2\_code = temp1;  
is\_unrecognized = TRUE;  
}  
else{  
/\* unrecognize byte. \*/   
\*ucs2\_code = 0x22e0;  
is\_unrecognized = FALSE;  
}  
return is\_unrecognized;  
}  
2,把一个UCS-2字符转换成UTF-8字符。函数返回转换成UTF-8的长度(字节1 -- 3)，如果目标指针为空，返回0。  
UINT8 UCS2toUTF8Code(UINT16 ucs2\_code, UINT8\* utf8\_code){  
int length = 0;  
UINT8\* out = utf8\_code;  
if(!utf8\_code){  
return length;  
}  
if(0x0080 > ucs2\_code){  
/\* 1 byte UTF-8 Character.\*/  
\*out = (UINT8)ucs2\_code;  
length++;   
}  
else if(0x0800 > ucs2\_code){  
/\*2 bytes UTF-8 Character.\*/  
\*out = ((UINT8)(ucs2\_code >> 6)) | 0xc0;  
\*(out+1) = ((UINT8)(ucs2\_code & 0x003F)) | 0x80;  
length += 2;  
}  
else{  
/\* 3 bytes UTF-8 Character .\*/  
\*out = ((UINT8)(ucs2\_code >> 12)) | 0xE0;  
\*(out+1) = ((UINT8)((ucs2\_code & 0x0FC0)>> 6)) | 0x80;  
\*(out+2) = ((UINT8)(ucs2\_code & 0x003F)) | 0x80;  
length += 3;   
}  
return length;  
}  
字符串间的转换也是一样的。

|  |
| --- |
| [概述] 计算机中的一切都是以数字来表示的，字符同样如此。字符编码就是将字符集编码成为数字序列，以便能让计算机识别。各个地区和国家使用的语言有别，将本地使用的语言符号进行编码就得到本地编码字符集。例如西欧国家使用的本地编码是ISO8859-1，中国大陆和新加坡等地区使用本地编码是GB2312或GBK，中国港台地区使用的本地编码是BIG5，韩国和日本的本地编码分别是euc-kr和Shift\_JIS。电脑的操作系统支持各种本地编码字符集，操作系统默认的本地编码和你所安装的操作系统语言版本是一致的。本地集只对本地使用的文字符号进行了编码，并不包括其他地区使用的文字，即使两个本地集中包含了相同的字符，这个字符的编码值也是不同的。例如“中”的GB2312或GBK编码值为“0xD6D0”，而BIG5编码值为“0xA4A4”。 全球信息交流与融合的趋势要求实现对本地字符集的统一，1984年4月ISO成立了工作组，针对各国文字、符号进行统一编码，这种编码成为Unicode。Unicode于1992年6月通过DIS(DrafInternationalStandard)，V2.0版本于1996年发布。Unicode编码包括了符号6811个、汉字20902个、韩文11172个、等等。Unicode虽然实现了全球统一编码，但是在字符集数量和编码效率方面显然存在着不足，而UTF-8、UTF-16就是针对Unicode编码进行转换或扩充形成的编码，UTF是Unicode Translation Format的缩写。  [细节] 关于ASCII编码 ASCII编码是美国标准信息交换码，这种**编码方式针对的是英文字符**。ASCII编码使用一个字节对字符进行编码，而且字节的最高位都为0，因此ASCII编码的字符集大小是128个。由于英文字母仅有26个，再加上其他一些常用符号，总大小也不会超过128个，因此ASCII编码的空间是足够的。例如，字符“a”被编码为0x61，字符“b”被编码为0x62等等。注意，在有的时候ASCII泛指本地编码，例如文本编辑器UltraEdit中有诸如“ASCII转Unicode”的功能，这里的ASCII就泛指本地编码，如果本地编码是GBK，这个功能执行的就是GBK编码到Unicode编码的转换。  关于ISO8859-1编码 ISO8859-1是西欧语系国家通用的字符集编码，ISO8859-1使用一个字节对字符进行编码，编码值范围是0x00-0xFF。其中，0x00-0x1F用作控制字，0x20-0x7F表示字母、数字和符号这些图形字符，0xA0-0xFF作为附加部分使用。由于ASCII编码只使用了一个字节中的低7位，编码范围仅为0-127，虽然可以容纳英文字符和其他的一些符号，但是却不能包含除英文以外的其他西欧语言的字母，因此ASCII编码在西欧国家并不通用。针对这个问题ISO在ASCII编码的基础上进行了扩充，制定了ISO8859-1编码，ISO8859-1编码使用了一个字节的全部8位，编码范围是0-255，能包含西欧语系的所有字母和符号。  关于GB2312、GBK和BIG5编码 **GB2312码是中华人民共和国国家汉字信息交换使用码**，全称《信息交换使用汉字编码字符集－基本集》，由国家标准总局发布，1981年5月1日实施，中国大陆和新加坡等地使用此编码。GB2312收录了简化汉字、符号、字母、日文假名等共计7445个字符，其中汉字占6763个。**GB2312将代码表分区94个区(0xA1-0xFE)，对应第一个字节，每个区94个位(0xA1-0xFE)，对应了第二字节，两个字节的值分别为区号的值和位号的值加32(0x20)，因此也被称为区位码。GB2312的编码范7围是0x2121-0x777E，与ASCII有重叠，通常方法是将GB码的两个字节的最高位置1区别**。 **GBK是GB2312-80的扩展，向上兼容**，包含了20902个汉字，编码范围是0x8140-0xFEFE，剔除高位0x80的字位，其他字符都可以一一映射到Unicode2.0。GB18030-2000(GBK2K)在GBK的基础上增加了藏、蒙等少数民族的字符，GBK2K从根本上解决了字位不够、字形不足的问题。GBK2K首先要求实现能够完全映射到Unicode3.0标准的所有字形，现在还没有任何一个操作系统支持GBK2K。 **BIG5码被称为大五码，是中国港台地区使用的字符编码方式**。TW-BIG5码将所有字分为两大群，即常用字区和次常用字区，每个字区分都采用笔画排序，同笔画的字依部首排序。TW-BIG5每个字由两个字节组成 |