# 字符编码：ASCII，Unicode和UTF-8以及GBK

作者：东坡网搜集整理 来源：互联网

本文原地址：http://www.dp1037.com/dpinfo-7-43-0.html

*东坡网，为帝国cms加油*

字符必须编码后才能被计算机处理。计算机使用的缺省编码方式就是计算机的内码。早期的计算机使用7位的ASCII编码，为了处理汉字，程序员设计了用于简体中文的GB2312和用于繁体中文的big5。

## 1. ASCII码

我们知道，在计算机内部，所有的信息最终都表示为一个二进制的字符串。每一个二进制位（bit）有0和1两种状态，因此八个二进制位就可以组合出256种状态，这被称为一个字节（byte）。也就是说，一个字节一共可以用来表示256种不同的状态，每一个状态对应一个符号，就是256个符号，从0000000到11111111。

上个世纪60年代，美国制定了一套字符编码，对英语字符与二进制位之间的关系，做了统一规定。这被称为ASCII码，一直沿用至今。

ASCII码一共规定了128个字符的编码，比如空格"SPACE"是32（二进制00100000），大写的字母A是65（二进制01000001）。这128个符号（包括32个不能打印出来的控制符号），只占用了一个字节的后面7位，最前面的1位统一规定为0。

## 2、非ASCII编码

英语用128个符号编码就够了，但是用来表示其他语言，128个符号是不够的。比如，在法语中，字母上方有注音符号，它就无法用ASCII码表示

。于是，一些欧洲国家就决定，利用字节中闲置的最高位编入新的符号。比如，法语中的 é 的编码为130（二进制10000010）。这样一来，这些欧洲国家使用的编码体系，可以表示最多256个符号。

但是，这里又出现了新的问题。不同的国家有不同的字母，因此，哪怕它们都使用256个符号的编码方式，代表的字母却不一样。比如，130在法语编码中代表了 é，在希伯来语编码中却代表了字母Gimel ( ג )，在俄语编码中又会代表另一个符号。但是不管怎样，所有这些编码方式中，0--127表示的符号是一样的，不一样的只是128--255的这一段。

至于亚洲国家的文字，使用的符号就更多了，汉字就多达10万左右。一个字节只能表示256种符号，肯定是不够的，就必须使用多个字节表达一个符号。比如，简体中文常见的编码方式是GB2312，使用两个字节表示一个汉字，所以理论上最多可以表示256x256=65536个符号。

GB类的汉字编码与后文的Unicode和UTF-8是毫无关系的。

## 3.Unicode

正如上一节所说，世界上存在着多种编码方式，同一个二进制数字可以被解释成不同的符号。因此，要想打开一个文本文件，就必须知道它的编码方式，否则用错误的编码方式解读，就会出现乱码。为什么电子邮件常常出现乱码？就是因为发信人和收信人使用的编码方式不一样。

可以想象，如果有一种编码，将世界上所有的符号都纳入其中。每一个符号都给予一个独一无二的编码，那么乱码问题就会消失。这就是Unicode，就像它的名字都表示的，这是一种所有符号的编码。

Unicode当然是一个很大的集合，现在的规模可以容纳100多万个符号。每个符号的编码都不一样，比如，U+0639表示阿拉伯字母Ain，U+0041表示英语的大写字母A，U+4E25表示汉字"严"。

## 4. Unicode的问题

需要注意的是，Unicode只是一个符号集，它只规定了符号的二进制代码，却没有规定这个二进制代码应该如何存储。

比如，汉字"严"的unicode是十六进制数4E25，转换成二进制数足足有15位（100111000100101），也就是说这个符号的表示至少需要2个字节。表示其他更大的符号，可能需要3个字节或者4个字节，甚至更多。

这里就有两个严重的问题，第一个问题是，如何才能区别Unicode和ASCII？计算机怎么知道三个字节表示一个符号，而不是分别表示三个符号呢？第二个问题是，我们已经知道，英文字母只用一个字节表示就够了，如果Unicode统一规定，每个符号用三个或四个字节表示，那么每个英文字母前都必然有二到三个字节是0，这对于存储来说是极大的浪费，文本文件的大小会因此大出二三倍，这是无法接受的。

它们造成的结果是：1）出现了Unicode的多种存储方式，也就是说有许多种不同的二进制格式，可以用来表示Unicode。2）Unicode在很长一段时间内无法推广，直到互联网的出现。

## 5.UTF-8

互联网的普及，强烈要求出现一种统一的编码方式。UTF-8就是在互联网上使用最广的一种Unicode的实现方式。其他实现方式还包括UTF-16（字符用两个字节或四个字节表示）和UTF-32
（字符用四个字节表示），不过在互联网上基本不用。**重复一遍，这里的关系是，UTF-8是Unicode的实现方式之一。**

UTF-8最大的一个特点，就是它是一种变长的编码方式。它可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度。

UTF-8的编码规则很简单，只有二条：

1）对于单字节的符号，字节的第一位设为0，后面7位为这个符号的unicode码。因此对于英语字母，UTF-8编码和ASCII码是相同的。

2）对于n字节的符号（n>1），第一个字节的前n位都设为1，第n+1位设为0，后面字节的前两位一律设为10。剩下的没有提及的二进制位，全部为这个符号的unicode码。

下表总结了编码规则，字母x表示可用编码的位。

Unicode符号范围 | UTF-8编码方式
(十六进制) | （二进制）
----------+---------------------------------------------
0000 0000-0000 007F | 0xxxxxxx
0000 0080-0000 07FF | 110xxxxx 10xxxxxx

0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx
0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

跟据上表，解读UTF-8编码非常简单。如果一个字节的第一位是0，则这个字节单独就是一个字符；如果第一位是1，则连续有多少个1，就表示当前字符占用多少个字节。

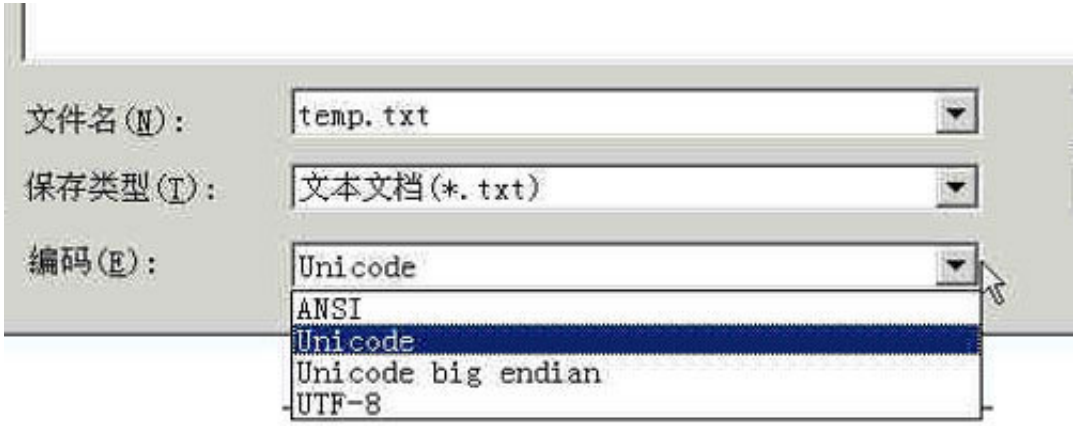
下面，还是以汉字"严"为例，演示如何实现UTF-8编码。

已知"严"的unicode是4E25（100111000100101），根据上表，可以发现4E25处在第三行的范围内（0000 0800-0000 FFFF），因此"严"的UTF-8编码需要三个字节，即格式是"1110xxxx 10xxxxxx 10xxxxxx"。然后，从"严"的最后一个二进制位开始，依次从后向前填入格式中的x，多出的位补0。这样就得到了，"严"的UTF-8编码是"11100100 10111000 10100101"，转换成十六进制就是E4B8A5。

## 6. Unicode与UTF-8之间的转换

通过上一节的例子，可以看到"严"的Unicode码是4E25，UTF-8编码是E4B8A5，两者是不一样的。它们之间的转换可以通过程序实现。

在Windows平台下，有一个最简单的转化方法，就是使用内置的记事本小程序Notepad.exe。打开文件后，点击"文件"菜单中的"另存为"命令，会跳出一个对话框，在最底部有一个"编码"的下拉条。




里面有四个选项：ANSI，Unicode，Unicode big endian 和 UTF-8。

1）ANSI是默认的编码方式。对于英文文件是ASCII编码，对于简体中文文件是GB2312编码（只针对Windows简体中文版，如果是繁体中文版会采用Big5码）。

2）Unicode编码指的是UCS-2编码方式，即直接用两个字节存入字符的Unicode码。这个选项用的little endian格式。

3）Unicode big endian编码与上一个选项相对应。我在下一节会解释little endian和big endian的涵义。

4）UTF-8编码，也就是上一节谈到的编码方法。

选择完"编码方式"后，点击"保存"按钮，文件的编码方式就立刻转换好了。

## 7. Little endian和Big endian

上一节已经提到，Unicode码可以采用UCS-2格式直接存储。以汉字"严"为例，Unicode码是4E25，需要用两个字节存储，一个字节是4E，另一个

字节是25。存储的时候，4E在前，25在后，就是Big endian方式；25在前，4E在后，就是Little endian方式。

这两个古怪的名称来自英国作家斯威夫特的《格列佛游记》。在该书中，小人国里爆发了内战，战争起因是人们争论，吃鸡蛋时究竟是从大头(Big-Endian)敲开还是从小头(Little-Endian)敲开。为了这件事情，前后爆发了六次战争，一个皇帝送了命，另一个皇帝丢了王位。

因此，第一个字节在前，就是"大头方式"（Big endian），第二个字节在前就是"小头方式"（Little endian）。

那么很自然的，就会出现一个问题：计算机怎么知道某一个文件到底采用哪一种方式编码？

Unicode规范中定义，每一个文件的最前面分别加入一个表示编码顺序的字符，这个字符的名字叫做"零宽度非换行空格"（ZERO WIDTH NO-BREAK SPACE），用FEFF表示。这正好是两个字节，而且FF比FE大1。

如果一个文本文件的头两个字节是FE FF，就表示该文件采用大头方式；如果头两个字节是FF FE，就表示该文件采用小头方式。

## 8. 实例

下面，举一个实例。

打开"记事本"程序Notepad.exe，新建一个文本文件，内容就是一个"严"字，依次采用ANSI，Unicode，Unicode big endian 和UTF-8编码方式保存。

然后，用文本编辑软件UltraEdit中的"十六进制功能"，观察该文件的内部编码方式。

1）ANSI：文件的编码就是两个字节"D1 CF"，这正是"严"的GB2312编码，这也暗示GB2312是采用大头方式存储的。

2）Unicode：编码是四个字节"FF FE 25 4E"，其中"FF FE"表明是小头方式存储，真正的编码是4E25。

3）Unicode big endian：编码是四个字节"FE FF 4E 25"，其中"FE FF"表明是大头方式存储。

4）UTF-8：编码是六个字节"EF BB BF E4 B8 A5"，前三个字节"EF BB BF"表示这是UTF-8编码，后三个"E4B8A5"就是"严"的具体编码，它的存储顺序与编码顺序是一致的。

## 8、GBK编码

汉字内码扩展规范，称GBK，全名为《汉字内码扩展规范(GBK)》1.0版，由中华人民共和国全国信息技术标准化技术委员会1995年12月1日制订，国家技术监督局标准化司和电子工业部科技与质量监督司1995年12月15日联合以《技术标函[1995]229号》文件的形式公布。

字符有一字节和双字节编码，00－7F范围内是第一个字节，和ASCII保持一致，此范围内严格上说有96个文字和32个控制符号。

之后的双字节中，前一字节是双字节的第一位。总体上说第一字节的范围是81－FE（也就是不含80和FF），第二字节的一部分领域在40－7E，其他领域在80－FE。

GBK向下完全兼容GB2312-80编码。支持GB2312-80编码不支持的部分中文姓，中文繁体，日文假名，还包括希腊字母以及俄语字母等字母。不过这种编码不支持韩国字，也是其在实际使用中与unicode编码相比欠缺的部分。

上述GBK/1和GBK/2的领域即GB 2312-80用通常方法编码的区域。GB 2312（正确说法是其根据EUC-CN的编码）和ISO/IEC 2022中调用GR其他的94?字符集一样，A1－FE的范围开始读取字节对。这是上图中右下角的部分。但是，GB 2312中对于AA－AF和F8－FE区域是空的，没有赋予编码。于是GBK就在这些领域里进行拓展。二者剩余部分作为用户定义区。

更重要的是，GBK进行了字节范围的扩展。ISO/IEC 2022中GR区域的字数有94?=8,836字的限制。只要放弃ISO/IEC 2022中针对图形文字和控制文字赋予严格的范围的模式，下位字节为单字节文字，上位字节对保留对应字符的功能，潜在的128?=16,384的代码位置就可以使用。GBK采用其中的一部分，第一个字节从A1－FE（每个字节有94个选项）扩展成81－FE（126个选项），第二字节的范围是40－FE（191个选项），总共有24066（126*191）个位置。

微软的CP936通常被视为等同GBK，连IANA也以“CP936”为“GBK”之别名[1]。事实上比较起来，GBK定义之字符较CP936多出95字（15个非汉字及80个汉字），皆为其时未收入ISO 10646 / Unicode之符号。

## 9、常用概念

内码是指操作系统内部的字符编码。早期操作系统的内码是与语言相关的。现在的Windows在系统内部支持Unicode，然后用代码页适应各种语言，“内码”的概念就比较模糊了。微软一般将缺省代码页指定的编码说成是内码。

内码这个词汇，并没有什么官方的定义，代码页也只是微软这个公司的叫法。作为程序员，我们只要知道它们是什么东西，没有必要过多地考证这些名词。

所谓代码页(code page)就是针对一种语言文字的字符编码。例如GBK的code page是CP936，BIG5的code page是CP950，GB2312的code page是CP20936。

Windows中有缺省代码页的概念，即缺省用什么编码来解释字符。例如Windows的记事本打开了一个文本文件，里面的内容是字节流：BA、BA、D7、D6。Windows应该去怎么解释它呢？

是按照Unicode编码解释、还是按照GBK解释、还是按照BIG5解释，还是按照ISO8859-1去解释？如果按GBK去解释，就会得到“汉字”两个字。按照其它编码解释，可能找不到对应的字符，也可能找到错误的字符。所谓“错误”是指与文本作者的本意不符，这时就产生了乱码。

答案是Windows按照当前的缺省代码页去解释文本文件里的字节流。缺省代码页可以通过控制面板的区域选项设置。记事本的另存为中有一项ANSI，其实就是按照缺省代码页的编码方法保存。

Windows的内码是Unicode，它在技术上可以同时支持多个代码页。只要文件能说明自己使用什么编码，用户又安装了对应的代码页，Windows就能正确显示，例如在HTML文件中就可以指定charset。

有的HTML文件作者，特别是英文作者，认为世界上所有人都使用英文，在文件中不指定charset。如果他使用了0x80-0xff之间的字符，中文Windows又按照缺省的GBK去解释，就会出现乱码。这时只要在这个html文件中加上指定charset的语句，例如：

<meta http-equiv="Content-Type" content="text/html; charset=ISO8859-1">

如果原作者使用的代码页和ISO8859-1兼容，就不会出现乱码了。

再说区位码，啊的区位码是1601，写成16进制是0x10,0x01。这和计算机广泛使用的ASCII编码冲突。为了兼容00-7f的ASCII编码，我们在区位码

的高、低字节上分别加上A0。这样“ 啊 ”的编码就成为B0A1。我们将加过两个A0的编码也称为GB2312编码，虽然GB2312的原文根本没提到这一点。

## 10、php编码相关的常用函数

```
int ord ( string $string ) //返回字符串 string 第一个字符的 ASCII 码值。
```

```
string chr ( int $ascii ) //返回相对应于 ascii所指定的单个字符。
```

```
string mb_convert_encoding ( string str, string to_encoding [, mixed from_encoding] )
```

```
string iconv ( string in_charset, string out_charset, string str )
```

iconv第二个参数，除了可以指定要转化到的编码以外，还可以增加两个后缀：//TRANSLIT 和 //IGNORE，其中：//TRANSLIT 会自动将不能直接转化的字符变成一个或多个近似的字符，//IGNORE 会忽略掉不能转化的字符，而默认效果是从第一个非法字符截断。

执行效率mb_convert_encoding
比iconv差。一般情况下用iconv，只有当遇到无法确定原编码是何种编码，或者iconv转化后无法正常显示时才用mb_convert_encoding 函数。

string decbin ( int $number ) //十进制转换为二进制，所能转换的最大数值为十进制的 4294967295，其结果为 32 个 1 的字符串
。

string decoct ( int $number ) //十进制转换为八进制，所能转换的最大数值为十进制的 4294967295，其结果为 "37777777777"。

string dechex ( int $number ) //十进制转换为十六进制，所能转换的最大数值为十进制的 4294967295，其结果为 "ffffffff"。

string bin2hex ( string $str ) //将二进制数据转换成十六进制表示，返回 ASCII 字符串，为参数 str的十六进制表示。转换使用字节
方式，高四位字节优先。

number bindec ( string $binary_string ) //将一个二进制数转换成 integer。可转换的最大的数为 31 位 1 或者说十进制的 2147483647。

string base_convert ( string $number , int $frombase , int $tobase ) //在任意进制之间转换数字，number本身的进制由 frombase 指定。frombase和 tobase 都只能在 2 和 36 之间（包括 2 和 36）。高于十进制的数字用字母 a-z 表示，例如 a 表示 10，b 表示 11 以及 z 表示 35。

```
//汉字转换为16进制编码
function hexEncode($s){
 return preg_replace('/(.)/es',"str_pad(dechex(ord('\\1')),2,'0',STR_PAD_LEFT)",$s);
}
```

```
//16进制编码转换为汉字
function hexDecode($s){
 return preg_replace('/(\w{2})/e',"chr(hexdec('\\1'))",$s);
}
```

```
//将汉字切割成数组
function mb_string_to_array($str,$charset,$num){
 $strlen=mb_strlen($str,$charset);
 $array=array();
 while($strlen){
  $array[]=mb_substr($str,0,$num,$charset);
  $str=mb_substr($str,$num,$strlen,$charset);
  $strlen=mb_strlen($str);
```

```
 }
 return $array;
}
```

```
//截取汉字
function hz_substr($str,$start,$length=null){
 $res=substr($str,$start,$length);
 $strlen=strlen($str);
 if($start>=0){
  $next_start=$start+$length;
  $next_len=$next_start+6<=$strlen?6:$strlen-$next_start;
  $next_segm=substr($str,$next_start,$next_len);
  $prev_start=$start-6>0?$start-6:0;
  $prev_segm=substr($str,$prev_start,$start-$prev_start);
 }
 else{
```

```
 $next_start=$strlen+$start+$length;
 $next_len=$next_start+6<=$strlen?6:$strlen-$next_start;
 $next_segm=substr($str,$next_start,$next_len);
 $start=$strlen+$start;
 $prev_start=$start-6>0?$start-6:0;
 $prev_segm=substr($str,$prev_start,$start-$prev_start);
}
if(preg_match('@^([\x80-\xBF]{0,5})[\xC0-\xFD]?@',$next_segm,$bytes)){
 if(!empty($bytes[1])){
  $bytes=$bytes[1];
  $res.=$bytes;
 }
}
$ord0=ord($res[0]);
if(128<=$ord0&&191>=$ord0){
 if(preg_match('@[\xC0-\xFD][\x80-\xBF]{0,5}$@',$prev_segm,$bytes)){
  if(!empty($bytes[0])){
   $bytes=$bytes[0];
```

```
    $res=$bytes.$res;
   }
  }
 }
 return $res;
}
```