**字符集**

**摘要：**字符编码（英语：Character encoding）、字集码是把字符集中的字符编码为指定集合中某一对象（例如：比特模式、自然数序列、8位组或者电脉冲），以便文本在计算机中存储和通过通信网络的传递计算机中储存的信息都是用二进制数表示的；而我们在屏幕上看到的英文、汉字等字符是二进制数转换之后的结果。常见字符集名称：ASCII字符集、GB2312字符集、BIG5字符集、GB18030字符集、Unicode字符集等。计算机要准确的处理各种字符集文字，需要进行字符编码，以便计算机能够识别和存储各种文字。

**关键词：**Unicode,GB-2312,BIG5,GB2312,utf8/16

1. **字符集**
   1. **Unicode**

Unicode（中文：万国码、国际码、统一码、单一码）是计算机科学领域里的一项业界标准。它对世界上大部分的文字系统进行了整理、编码，使得电脑可以用更为简单的方式来呈现和处理文字。Unicode涵盖的数据除了视觉上的字形、编码方法、标准的字符编码外，还包含了字符特性，如大小写字母。

Unicode是为了解决传统的字符编码方案的局限而产生的，例如ISO 8859-1所定义的字符虽然在不同的国家中广泛地使用，可是在不同国家间却经常出现不兼容的情况。很多传统的编码方式都有一个共同的问题，即容许电脑处理双语环境（通常使用拉丁字母以及其本地语言），但却无法同时支持多语言环境（指可同时处理多种语言混合的情况）。

在文字处理方面，统一码为每一个字符而非字形定义唯一的代码（即一个整数）。换句话说，统一码以一种抽象的方式（即数字）来处理字符，并将视觉上的演绎工作（例如字体大小、外观形状、字体形态、文体等）留给其他软件来处理，例如网页浏览器或是文字处理器。

在表示一个Unicode的字符时，通常会用“U+”然后紧接着一组十六进制的数字来表示这一个字符。在基本多文种平面（英文：Basic Multilingual Plane，简写BMP。又称为“零号平面”、plane 0）里的所有字符，要用四个数字（即两个char,16bit ,例如U+4AE0，共支持六万多个字符）；在零号平面以外的字符则需要使用五个或六个数字。旧版的Unicode标准使用相近的标记方法，但却有些微小差异：在Unicode 3.0里使用“U-”然后紧接着八个数字，而“U+”则必须随后紧接着四个数字。

统一码的编码方式与ISO 10646的通用字符集概念相对应。目前实际应用的统一码版本对应于UCS-2，使用16位的编码空间。也就是每个字符占用2个字节。这样理论上一共最多可以表示216（即65536）个字符。基本满足各种语言的使用。实际上当前版本的统一码并未完全使用这16位编码，而是保留了大量空间以作为特殊使用或将来扩展。

上述16位统一码字符构成基本多文种平面。最新（但未实际广泛使用）的统一码版本定义了16个辅助平面，两者合起来至少需要占据21位的编码空间，比3字节略少。但事实上辅助平面字符仍然占用4字节编码空间，与UCS-4保持一致。未来版本会扩充到ISO 10646-1实现级别3，即涵盖UCS-4的所有字符。UCS-4是一个更大的尚未填充完全的31位字符集，加上恒为0的首位，共需占据32位，即4字节。理论上最多能表示231个字符，完全可以涵盖一切语言所用的符号。基本多文种平面的字符的编码为*U+hhhh*，其中每个*h*代表一个十六进制数字，与UCS-2编码完全相同。而其对应的4字节UCS-4编码后两个字节一致，前两个字节则所有位均为0。

* 1. **大五码**

Big5，又称为大五码或五大码，是使用繁体中文（正体中文）社区中最常用的电脑汉字字符集标准，共收录13,060个汉字。中文码分为内码及交换码两类，Big5属中文内码，知名的中文交换码有CCCII、CNS11643。

“大五码”（Big5）是由台湾财团法人信息产业策进会为五大中文套装软件所设计的中文共通内码，在1983年12月完成公告，隔年3月，信息产业策进会与台湾13家厂商签定“16位个人电脑套装软件合作开发（BIG-5）项目（五大中文套装软件）”，因为此中文内码是为台湾自行制作开发之“五大中文套装软件”所设计的，所以就称为Big5中文内码。五大中文套装软件虽然并没有如预期的取代国外的套装软件，但随着采用Big5码的国乔中文系统及倚天中文系统先后在台湾市场获得成功，使得Big5码深远地影响繁体中文电脑内码，直至今日。“五大码”的英文名称“Big5”后来被人按英文字序译回中文，以致现在有“五大码”和“大五码”两个中文名称。

Big5码的产生，是因为当时个人电脑没有共通的内码，导致厂商推出的中文应用软件无法推广，并且与IBM 5550、王安码等内码，彼此不能兼容；另一方面，台湾当时尚未推出中文编码标准。在这样的时空背景下，为了使台湾早日进入信息时代，所采行的一个项目；同时，这个项目对于以台湾为核心的亚洲繁体汉字圈也产生了久远的影响。

Big5码是一套双字节字符集，使用了双八码存储方法，以两个字节来安放一个字。第一个字节称为“高位字节”，第二个字节称为“低位字节”。“高位字节”使用了0x81-0xFE，“低位字节”使用了0x40-0x7E，及0xA1-0xFE。

由于各厂商及政府推出的Big5延伸，彼此互不兼容，造成乱码问题。鉴于Unicode能正确地处理七万多个汉字，近年的操作系统和应用程序（如苹果电脑Mac OS X和以Cocoa API撰写之程序、Microsoft Windows 2000及之后版本、Microsoft Office 2000及之后版本、Mozilla浏览器、Internet Explorer浏览器、Java语言等等），已改用Unicode编码

* 1. **Utf 8**

UTF-8（8-bit Unicode Transformation Format）是一种针对Unicode的可变长度字符编码，也是一种前缀码。它可以用来表示Unicode标准中的任何字符，且其编码中的第一个字节仍与ASCII兼容，这使得原来处理ASCII字符的软件无须或只须做少部分修改，即可继续使用。因此，它逐渐成为电子邮件、网页及其他存储或发送文字的应用中，优先采用的编码。

UTF-8使用一至六个字节为每个字符编码（尽管如此，2003年11月UTF-8被RFC 3629重新规范，只能使用原来Unicode定义的区域，U+0000到U+10FFFF，也就是说最多四个字节）：

1. 128个US-ASCII字符只需一个字节编码（Unicode范围由U+0000至U+007F）。
2. 带有附加符号的拉丁文、希腊文、西里尔字母、亚美尼亚语、希伯来文、阿拉伯文、叙利亚文及它拿字母则需要两个字节编码（Unicode范围由U+0080至U+07FF）。
3. 其他基本多文种平面（BMP）中的字符（这包含了大部分常用字，如大部分的汉字）使用三个字节编码（Unicode范围由U+0800至U+FFFF）。
4. 其他极少使用的Unicode 辅助平面的字符使用四至六字节编码（Unicode范围由U+10000至U+1FFFFF使用四字节，Unicode范围由U+200000至U+3FFFFFF使用五字节，Unicode范围由U+4000000至U+7FFFFFFF使用六字节）。

对上述提及的第四种字符而言，UTF-8使用四至六个字节来编码似乎太耗费资源了。但UTF-8对所有常用的字符都可以用三个字节表示，而且它的另一种选择，UTF-16编码，对前述的第四种字符同样需要四个字节来编码，所以要决定UTF-8或UTF-16哪种编码比较有效率，还要视所使用的字符的分布范围而定。

1992年初，为创建良好的字节串编码系统以供多字节字符集使用，开始了一个正式的研究。ISO/IEC 10646的初稿中有一个非必须的附录，名为UTF。当中包含了一个供32比特的字符使用的字节串编码系统。这个编码方式的性能并不令人满意，但它提出了将0-127的范围保留给ASCII以兼容旧系统的概念。1992年7月，X/Open委员会XoJIG开始寻求一个较佳的编码系统。Unix系统实验室（USL）的Dave Prosser为此提出了一个编码系统的建议。它具备可更快速实现的特性，并引入一项新的改进。其中，7比特的ASCII符号只代表原来的意思，所有多字节序列则会包含第8比特的符号，也就是所谓的最高有效比特。1992年8月，这个建议由IBMX/Open的代表流传到一些感兴趣的团体。与此同时，贝尔实验室九号项目操作系统工作小组的肯·汤普逊对这编码系统作出重大的修改，让编码可以自我同步，使得不必从字符串的开首读取，也能找出字符间的分界。1992年9月2日，肯·汤普逊和罗勃·派克一起在美国新泽西州一架餐车的餐桌垫上描绘出此设计的要点。接下来的日子，Pike及汤普逊将它实现，并将这编码系统完全应用在九号项目当中，及后他将有关成果回馈X/Open。1993年1月25-29日的在圣地牙哥举行的USENIX会议首次正式介绍UTF-8。

自1996年起，微软的CAB（MS Cabinet）规格在UTF-8标准正式落实前就明确容许在任何地方使用UTF-8编码系统。但有关的编码器实际上从来没有实现这方面的规格。

* 对于UTF-8编码中的任意字节B，如果B的第一位为0，则B独立的表示一个字符(ASCII码)；
* 如果B的第一位为1，第二位为0，则B为一个多字节字符中的一个字节(非ASCII字符)；
* 如果B的前两位为1，第三位为0，则B为两个字节表示的字符中的第一个字节；
* 如果B的前三位为1，第四位为0，则B为三个字节表示的字符中的第一个字节；
* 如果B的前四位为1，第五位为0，则B为四个字节表示的字符中的第一个字节；

因此，对UTF-8编码中的任意字节，根据第一位，可判断是否为ASCII字符；根据前二位，可判断该字节是否为一个字符编码的第一个字节；根据前四位（如果前两位均为1），可确定该字节为字符编码的第一个字节，并且可判断对应的字符由几个字节表示；根据前五位（如果前四位为1），可判断编码是否有错误或数据传输过程中是否有错误。

UTF-8的设计有以下的多字符组序列的特质：

* 单字节字符的最高有效比特永远为0。
* 多字节序列中的首个字符组的几个最高有效比特决定了序列的长度。最高有效位为110的是2字节序列，而1110的是三字节序列，如此类推。
* 多字节序列中其余的字节中的首两个最高有效比特为10。

UTF-8的这些特质，保证了一个字符的字节序列不会包含在另一个字符的字节序列中。这确保了以字节为基础的部分字符串比对（sub-string match）方法可以适用于在文字中搜索字或词。有些比较旧的可变长度8位编码（如Shift JIS）没有这个特质，故字符串比对的算法变得相当复杂。虽然这增加了UTF-8编码的字符串的信息冗余，但是利多于弊。另外，数据压缩并非Unicode的目的，所以不可混为一谈。即使在发送过程中有部分字节因错误或干扰而完全丢失，还是有可能在下一个字符的起点重新同步，令受损范围受到限制。

另一方面，由于其字节序列设计，如果一个疑似为字符串的序列被验证为UTF-8编码，那么我们可以有把握地说它是UTF-8字符串。一段两字节随机序列碰巧为合法的UTF-8而非ASCII的概率为32分1。对于三字节序列的概率为256分1，对更长的序列的概率就更低了。

* 1. **Utf 16**

UTF-16是Unicode字符编码五层次模型的第三层：字符编码表（Character Encoding Form，也称为"storage format"）的一种实现方式。即把Unicode字符集的抽象码位映射为16位长的整数（即码元）的序列，用于数据存储或传递。Unicode字符的码位，需要1个或者2个16位长的码元来表示，因此这是一个变长表示。

UTF是"Unicode/UCS Transformation Format"的首字母缩写，即把Unicode字符转换为某种格式之意。UTF-16正式定义于ISO/IEC 10646-1的附录C，而RFC2781也定义了相似的做法。

Unicode的编码空间从U+0000到U+10FFFF，共有1,112,064个码位（code point）可用来映射字符. Unicode的编码空间可以划分为17个平面（plane），每个平面包含216（65,536）个码位。17个平面的码位可表示为从U+xx0000到U+xxFFFF，其中xx表示十六进制值从0016到1016，共计17个平面。第一个平面称为基本多语言平面（Basic Multilingual Plane, BMP），或称第零平面（Plane 0）。其他平面称为辅助平面（Supplementary Planes）。基本多语言平面内，从U+D800到U+DFFF之间的码位区块是永久保留不映射到Unicode字符。UTF-16就利用保留下来的0xD800-0xDFFF区段的码位来对辅助平面的字符的码位进行编码。

UTF-16的大尾序和小尾序存储形式都在用。一般来说，以Macintosh制作或存储的文字使用大尾序格式，以Microsoft或Linux制作或存储的文字使用小尾序格式。

为了弄清楚UTF-16文件的大小尾序，在UTF-16文件的开首，都会放置一个U+FEFF字符作为Byte Order Mark（UTF-16LE以FF FE代表，UTF-16BE以FE FF代表），以显示这个文本文件是以UTF-16编码，其中U+FEFF字符在UNICODE中代表的意义是ZERO WIDTH NO-BREAK SPACE，顾名思义，它是个没有宽度也没有断字的空白。

* 1. **GB 2312**

GB 2312 或 GB 2312–80 是中华人民共和国国家标准简体中文字符集，全称《信息交换用汉字编码字符集·基本集》，又称GB0，由中国国家标准总局发布，1981年5月1日实施。GB 2312编码通行于中国大陆；新加坡等地也采用此编码。中国大陆几乎所有的中文系统和国际化的软件都支持GB 2312。

GB 2312标准共收录6763个汉字，其中一级汉字3755个，二级汉字3008个；同时收录了包括拉丁字母、希腊字母、日文平假名及片假名字母、俄语西里尔字母在内的682个字符。

GB 2312的出现，基本满足了汉字的计算机处理需要，它所收录的汉字已经覆盖中国大陆99.75%的使用频率。但对于人名、古汉语等方面出现的罕用字和繁体字，GB 2312不能处理，因此后来GBK及GB 18030汉字字符集相继出现以解决这些问题。

在使用GB 2312的程序通常采用EUC储存方法，以便兼容于ASCII。这种格式称为EUC-CN。浏览器编码表上的“GB2312”就是指这种表示法。

每个汉字及符号以两个字节来表示。第一个字节称为“高位字节”，第二个字节称为“低位字节”。

“高位字节”使用了0xA1–0xF7（把01–87区的区号加上0xA0），“低位字节”使用了0xA1–0xFE（把01–94加上0xA0）。 由于一级汉字从16区起始，汉字区的“高位字节”的范围是0xB0–0xF7，“低位字节”的范围是0xA1–0xFE，占用的码位是72\*94=6768。其中有5个空位是D7FA–D7FE。

1. **关系**

B2312中GB是国标，顾名思义，这是我们中国的汉字编码标准。国标中2个字节表示一个汉字，“GB2312的原文”是指国家1980年的一个标准《中华人民共和国国家标准 信息交换用汉字编码字符集 基本集 GB 2312-80》。这个标准用两个数来编码汉字和中文符号。第一个数称为“区”，第二个数称为“位”。所以也称为区位码。1-9区是中文符号，16-55区是一级汉字，56-87区是二级汉字。现在Windows也还有区位输入法，例如输入1601得到“啊”。（这个区位输入法可以自动识别16进制的GB2312和10进制的区位码，也就是说输入B0A1同样会得到“啊”。）

BIG-5码是通行于台湾、香港地区的一个繁体字编码方案，俗称“大五码”。地区标准号为：CNS11643，这就是人们讲的BIG-5码。

然而GB2312只是中国的标准，Big5只是台湾和香港的标准，那国际上使用汉字该如何处理，除了英语外其他的文字又如何处理，不同语言之间又如何共存？这就需要一个统一的编码方式，能够兼容全世界所有的语言，这个编码方式就是Unicode。

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。

UCS有两种格式：UCS-2和UCS-4。顾名思义，UCS-2就是用两个字节编码，UCS-4就是用4个字节（实际上只用了31位，最高位必须为0）编码。下面让我们做一些简单的数学游戏：

UCS-2有2^16=65536个码位，UCS-4有2^31=2147483648个码位。

UCS-4根据最高位为0的最高字节分成2^7=128个group。每个group再根据次高字节分为256个plane。每个plane根据第3个字节分为256行 (rows)，每行包含256个cells。当然同一行的cells只是最后一个字节不同，其余都相同。

group 0的plane 0被称作Basic Multilingual Plane, 即BMP。或者说UCS-4中，高两个字节为0的码位被称作BMP。

将UCS-4的BMP去掉前面的两个零字节就得到了UCS-2。在UCS-2的两个字节前加上两个零字节，就得到了UCS-4的BMP。而目前的UCS-4规范中还没有任何字符被分配在BMP之外。

UTF8,UTF16就是Unicode的不同的存储方式，UTF-8就是以8位为单元对UCS进行编码。从UCS-2到UTF-8的编码方式如下：

UCS-2编码(16进制) UTF-8 字节流(二进制)  
0000 – 007F 0xxxxxxx  
0080 – 07FF 110xxxxx 10xxxxxx  
0800 – FFFF 1110xxxx 10xxxxxx 10xxxxxx

例如“汉”字的Unicode编码是6C49。6C49在0800-FFFF之间，所以肯定要用3字节模板了：1110xxxx 10xxxxxx 10xxxxxx。将6C49写成二进制是：0110 110001 001001， 用这个比特流依次代替模板中的x，得到：11100110 10110001 10001001，即E6 B1 89。

UTF-16以16位为单元对UCS进行编码。对于小于0×10000的UCS码，UTF-16编码就等于UCS码对应的16位无符号整数。对于不小于0×10000的UCS码，定义了一个算法。不过由于实际使用的UCS2，或者UCS4的BMP必然小于0×10000，所以就目前而言，可以认为UTF-16和UCS-2基本相同。但UCS-2只是一个编码方案，UTF-16却要用于实际的传输，所以就不得不考虑字节序的问题

电子计算机技术是从美国开始发展起来的，因为美国使用的文字为英文，美国规定的计算机信息交换用的字符编码集是人们熟知的扩展的ASCII码，它以8bit字节为单位存储，ASCII的0-31及127为控制符，32-126为可见字符，包括所有的英文字母，阿拉伯数字和其他一些常见符号，128-255的ASCII码则没有定义。   
 ASCII对英语国家是够用了，但对其他西欧国家却不够用，因此,人们将ASCII扩展到0-255的范围，形成了ISO-8859-1字符集。值得一提的是，因为考虑到程序中处理的信息大多是西文信息，因此有些WEB容器（如：Tomcat4.x）在处理所接收到的request字符串时，如果您没指定request的编码方式则系统就缺省地采用ISO-8859-1，明白这一点对理解后面的问题会有帮助。   
 相比西方的拼音文字，东方的文字（如中文）的字符数要大得多，根本不可能在一个字节内将它们表示出来，因此，它们以两个字节为单位存储，以中文国标字符集GB2312为例，它的第一个字节为128-255。系统可以据此判断，若第一个字节大于127，则把与该字节后紧接着的一个字节结合起来共两个字节组成一个中文字符。这种由多个字节存储一个字符的字符集叫多字节字符集（MultiByte Charsets），对应的象ASCII这种用一个字节存储一个字符的字符集叫单字节字符集（SingleByte Charsets）。在GB2312字符集中，ASCII字符仍然用一个字节存储，换句话说该ASCII是该字符集的子集。   
 GB2312只包含数千个常用汉字，往往不能满足实际需要，因此，人们对它进行扩展，这就有了我们现在广泛使用的GBK字符集，GBK是现阶段Windows及其他一些中文操作系统的缺省字符集。它包含2万多个字符，除了保持和GB2312兼容外，还包含繁体中文字，日文字符和朝鲜字符。值得注意的是GBK只是一个规范而不是国家标准，新的国家标准是GB18030-2000，它是比GBK包含字符更多的字符集。   
 我国的台湾地区使用的文字是繁体字，其字符集是BIG5，而日本采用的字符集则是SJIS。它们的编码方法与GB2312类似，它们的ASCII字符部分是兼容的，但扩展部分的编码则是不兼容的，比如这几种字符集中都有"中文"这两个字符，但他们在各自的字符集中的编码并不相同，这就是用GB2312写成的网页用BIG5浏览时，看到的是乱糟糟的信息的原因。   
 可见，在字符集的世界里，呈现给我们的是一个群雄割据的局面，各字符集拥有一块自己的地盘。这给各国和各地区交换信息带来了很大的困难，同时，也给国际化（本地化）编程造成了很大的麻烦。   
 常言道："分久必合"，随着国际标准ISO10646定义的通用字符集（Universal Character Set即UCS）的出现，使这种局面发生了彻底的改观。UCS 是所有其他字符集标准的一个超集. 它保证与其他字符集是双向兼容的. 就是说, 如果你将任何文本字符串翻译到 UCS格式, 然后再翻译回原编码, 你不会丢失任何信息。UCS 包含了用于表达所有已知语言的字符。不仅包括拉丁语、希腊语、 斯拉夫语、希伯来语、阿拉伯语、亚美尼亚语和乔治亚语的描述、还包括中文、日文和韩文这样的象形文字、 以及平假名、片假名、 孟加拉语、 旁遮普语果鲁穆奇字符(Gurmukhi)、泰米尔语、印.埃纳德语(Kannada)、Malayalam、泰国语、 老挝语、 汉语拼音(Bopomofo)、Hangul、 Devangari、Gujarati、Oriya、Telugu 以及其他数也数不清的语。对于还没有加入的语言，由于正在研究怎样在计算机中最好地编码它们， 因而最终它们都将被加入。   
 ISO 10646 定义了一个 31 位的字符集。然而， 在这巨大的编码空间中， 迄今为止只分配了前 65534 个码位 (0x0000 到 0xFFFD)。 这个 UCS 的 16位子集称为基本多语言面 (Basic Multilingual Plane, BMP)。 将被编码在 16 位 BMP 以外的字符都属于非常特殊的字符(比如象形文字)， 且只有专家在历史和科学领域里才会用到它们。   
 UCS 不仅给每个字符分配一个代码， 而且赋予了一个正式的名字。 表示一个 UCS 值的十六进制数， 通常在前面加上 "U+", 就象 U+0041 代表字符"拉丁大写字母A"。 UCS 字符 U+0000 到 U+007F 与 US-ASCII(ISO 646) 是一致的， U+0000 到 U+00FF 与 ISO 8859-1(Latin-1) 也是一致的。这里要注意的是它是以16bit为单位存储，即便对字母"A"也是用16bit，这是与前面介绍的所有字符集不同的地方。   
 历史上，在国际标准化组织研究ISO10646标准的同时，另一个由多语言软件制造商组成的协会也在从事创立单一字符集的工作，这就是现在人们熟知的 Unicode。幸运的是，1991年前后ISO10646和Unicode的参与者都认识到，世界上不需要两个不同的单一字符集。他们合并双方的工作成果，并为创立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，都同意保持ISO10646和Unicode的码表兼容，并紧密地共同调整任何未来的扩展。这与当年在PC机上的操作系统MS-dos与PC-dos的情形有些相象。后面，我们将视ISO10646和Unicode为同一个东西。

有了Unicode，字符集问题接近了完美的解决，但不要高兴得过早。由于历史的原因：一些操作系统如：Unix、Linux等都是基于ASCII设计的。此外，还有一些数据库管理系统软件如：Oracle等也是围绕ASCII来设计的(从其8i的白皮书上介绍的设置系统字符集和字段的字符集中可以间接地看到这一点)。在这些系统中直接用Unicode会导致严重的问题。用这些编码的字符串会包含一些特殊的字符， 比如 '/0' 或 '/'， 它们在 文件名和其他 C 库函数参数里都有特别的含义。 另外， 大多数使用 ASCII 文件的 UNIX 下的工具，如果不进行重大修改是无法读取 16 位的字符的。 基于这些原因， 在文件名, 文本文件, 环境变量等地方，直接使用Unicode是不合适的。   
  
 在 ISO 10646-1 Annex R 和 RFC 2279 里定义的 UTF-8 （Unicode Transformation Form 8-bit form）编码没有这些问题。   
UTF-8 有以下一些特性：   
 CS 字符 U+0000 到 U+007F (ASCII) 被编码为字节 0x00 到 0x7F (ASCII 兼容)。 这意味着只包含 7 位 ASCII 字符的文件在 ASCII 和 UTF-8 两种编码方式下是一样的。   
 所有 >U+007F 的 UCS 字符被编码为一个多个字节的串， 每个字节都有标记位集。 因此，ASCII 字节 (0x00-0x7F) 不可能作为任何其他字符的一部分。   
表示非 ASCII 字符的多字节串的第一个字节总是在 0xC0 到 0xFD 的范围里, 并指出这个字符包含多少个字节。 多字节串的其余字节都在 0x80 到 0xBF 范围里。 这使得重新同步非常容易， 并使编码无国界，且很少受丢失字节的影响。

UTF-8 编码字符理论上可以最多到 6 个字节长， 然而 16 位 BMP 字符最多只用到 3 字节长。

字节 0xFE 和 0xFF 在 UTF-8 编码中从未用到。

通过，UTF-8这种形式，Unicode终于可以广泛的在各种情况下使用了.

**第二部分: 从Unicode到UTF-8的转换**

Unicode是一个字符集，而UTF-8是 Unicode的其中一种，Unicode是定长的都为双字节，而UTF-8是可变的，对于汉字来说Unicode占有的字节比UTF-8占用的字节少1 个字节。Unicode为双字节，而UTF-8中汉字占三个字节。  
    UTF-8编码字符理论上可以最多到6个字节长,然而16位BMP（Basic Multilingual Plane）字符最多只用到3字节长。下面看一下UTF-8编码表：  
  
       U-00000000 - U-0000007F: 0xxxxxxx   
        U-00000080 - U-000007FF: 110xxxxx 10xxxxxx   
        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: 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx   
  
    xxx 的位置由字符编码数的二进制表示的位填入， 越靠右的 x 具有越少的特殊意义，只用最短的那个足够表达一个字符编码数的多字节串。注意在多字节串中, 第一个字节的开头"1"的数目就是整个串中字节的数目。而第一行中以0开头，是为了兼容ASCII编码，为一个字节，第二行就为双字节字符串，第三行为3 字节，如汉字就属于这种，以此类推。  
    为了要将Unicode转换为UTF-8，当然要知道他们的区别到底在什么地方。下面来看一下，在Unicode中的编码是怎样转换成UTF-8的，在UTF-8中，如果一个字符的字节小于0x80（128）则为ASCII字符，占一个字节，可以不用转换，因为UTF-8兼容ASCII编码。假如在Unicode中汉字“你”的编码为“u4F60”，把它转换为二进制为100111101100000，然后按照UTF-8的方法进行转换。可以将Unicode二进制从地位往高位取出二进制数字，每次取6位，如上述的二进制就可以分别取出为如下所示的格式，前面按格式填补，不足8位用0填补。

       unicode: 100111101100000                  4F60  
  
           utf-8:    11100100,10111101,10100000       E4BDA0

    从上面就可以很直观的看出Unicode到UTF-8之间的转换，当然知道了UTF-8的格式后，就可以进行逆运算，就是按照格式把它在二进制中的相应位置上取出，然后在转换就是所得到的Unicode字符了（这个运算可以通过“位移”来完成）。  
    如上述的“你”的转换，由于其值大于0x800小于0x10000，因此可以判断为三字节存储，则最高位需要向右移“12”位再根据三字节格式的最高位为 11100000（0xE0）求或（|）就可以得到最高位的值了。同理第二位则是右移“6”位，则还剩下最高位和第二位的二进制值，可以通过与 111111（0x3F）求按位于（&）操作，再和11000000（0x80）求或（|）。第三位就不用移位了，只要直接取最后六位（与 111111（ox3F）取&），在与11000000(0x80)求或（|）。OK了，转换成功！在VC++中的代码如下所示（Unicode到UTF-8的转换）。

        1 const wchar\_t pUnicode = L"你";  
        2 char utf8[3+1];  
        3 memset(utf8,0,4);  
        4 utf8[0] = 0xE0|(pUnicode>>12);  
        5 utf8[1] = 0x80|((pUnicode>>6)&0x3F);  
        6 utf8[2] = 0x80|(pUnicode&0x3F);  
        7 utf8[3] = "/0";  
        8 //char[4]就是UTF-8的字符“你”了。

    当然在UTF-8到Unicode的转换也是通过移位等来完成的，就是把UTF-8那些格式相应的位置的二进制数给揪出来。在上述例子中“你”为三个字节，因此要每个字节进行处理，有高位到低位进行处理。在UTF-8中“你”为11100100,10111101,10100000。从高位起即第一个字节11100100就是把其中的"0100"给取出来，这个很简单只要和11111（0x1F）取与（&），由三字节可以得知最到位肯定位于12位之前，因为每次取六位。所以还要将得到的结果左移12位，最高位也就这样完成了0100,000000,000000。而第二位则是要把“111101”给取出来，则只需将第二字节10111101 和111111(0x3F)取与（&）。在将所得到的结果左移6位与最高字节所得的结果取或（|），第二位就这样完成了，得到的结果为 0100,111101,000000。以此类推最后一位直接与111111（0x3F）取与（&），再与前面所得的结果取或（|）即可得到结果 0100,111101,100000。OK。

**参考文献：**

1. **维基百科https://zh.wikipedia.org/wiki/UTF-8.**
2. **维基百科 https://zh.wikipedia.org/wiki/Unicode.**
3. **维基百科https://zh.wikipedia.org/wiki/UTF-16.**
4. **维基百科https://zh.wikipedia.org/wiki/GB\_2312.**
5. **维基百科https://zh.wikipedia.org/wiki/%E5%A4%A7%E4%BA%94%E7%A2%BC.**
6. **字符集和字符编码http://www.cnblogs.com/skynet/archive/2011/05/03/2035105.html**
7. **JACK47 http://www.cnblogs.com/Jack47/p/unicode-utf8-character\_set-encoding.html**