# Unicode

来历（来自维基百科，自由的百科全书）：Unicode的明确目标是超越传统的字符编码的局限性,比如定义的[ISO 8859](https://en.wikipedia.org/wiki/ISO_8859" \o "ISO 8859)标准,在世界各国得到广泛使用但仍很大程度上互相矛盾。 许多传统的字符编码共享一个共同的问题,它们允许双语计算机处理(通常使用[拉丁字符](https://en.wikipedia.org/wiki/Latin_character" \o "Latin character)和当地的脚本),但不是多语言的计算机处理(计算机处理任意脚本相互混合)。

意图,Unicode编码——底层字符字母和grapheme-like单位,而变异符号(效果图)这样的人物。 在的情况下汉字,这有时会导致争议区别底层字符从它的变体(见的符号韩统一)。

Unicode文本处理,需要提供一个独特的角色*代码点*——一个数量,而不是glyph-for每个字符。 换句话说,Unicode代表一个字符在一种抽象的方式,使视觉呈现(尺寸、形状、[字体](https://en.wikipedia.org/wiki/Font" \o "Font)其他软件或风格),如web浏览器或文字处理器。 这个简单的目标变得复杂,然而,由于让步由Unicode的设计者希望鼓励更快速的采用Unicode。

256年第一个代码点都是相同的内容iso - 8859 - 1使其简单转换现有的西方文本。 许多本质上相同的字符编码多次在不同的代码点保护遗产编码,因此使用的区别,允许从这些编码转换到Unicode(来回),而不丢失任何信息。 例如,“fullwidth形式”部分的代码点包含一个完整的拉丁字母分开主要拉丁字母部分因为在中国,日本和韩国(CJK)字体,这些拉丁字符宽度一样CJK呈现象形文字,而不是在宽度的一半。 其他的例子,请参阅在Unicode字符复制。

产生原因：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）。

Unicode编码包含了不同写法的字，如“ɑ／a”、“強／强”、“戶／户／戸”。然而在汉字方面引起了一字多形的认定争议。

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。

目前，几乎所有电脑系统都支持基本拉丁字母，并各自支持不同的其他编码方式。Unicode为了和它们相互兼容，其首256字符保留给ISO 8859-1所定义的字符，使既有的西欧语系文字的转换不需特别考量；并且把大量相同的字符重复编到不同的字符码中去，使得旧有纷杂的编码方式得以和Unicode编码间互相直接转换，而不会丢失任何信息。举例来说，全角格式区块包含了主要的拉丁字母的全角格式，在中文、日文、以及韩文字形当中，这些字符以全角的方式来呈现，而不以常见的半角形式显示，这对竖排文字和等宽排列文字有重要作用。

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。

##### 解决的问题：

Unicode的成功统一字符集导致其广泛和主要用途国际化和本地化的计算机软件。 标准已经在许多最近的技术实现,包括现代操作系统,XML,Java(和其他编程语言),。 微软网络框架。

Unicode可以实现由不同的字符编码。 Unicode标准定义了utf - 8,utf - 16,utf - 32,和其他几个编码使用。 最常用的编码utf - 8,utf - 16ucs - 2,前兆utf - 16。

##### 字节数：

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。

**描述**：**nicode**是一个计算行业标准一致的吗[编码](https://en.wikipedia.org/wiki/Character_encoding" \o "Character encoding)、表示和处理的[文本](https://en.wikipedia.org/wiki/Character_(computing))在大多数世界上的表达书写系统。 136755年的最新版本包含一个曲目[字符](https://en.wikipedia.org/wiki/Character_(computing)" \o "Character (computing))覆盖139现代和历史[脚本](https://en.wikipedia.org/wiki/Script_(Unicode)" \o "Script (Unicode))以及多个符号集。*Unicode标准*保持在一起吗ISO / IEC 10646,都是code-for-code相同。

*Unicode标准*由一组代码图表视觉参考,一组编码方法和标准字符编码,一组参考数据文件,和一些相关的项目,如字符属性、规则[归一化](https://en.wikipedia.org/wiki/Unicode_normalization" \o "Unicode normalization)分解,排序呈现,双向正确显示的显示顺序(从右到左的文本包含脚本,如阿拉伯语和希伯来语,从左到右的脚本)。[[1]](https://en.wikipedia.org/wiki/Unicode" \l "cite_note-1)截至2017年6月最新版本*Unicode 10.0*。 维护的标准Unicode协会。

**兼容哪些字符集**：Unicode只与ASCII兼容

**6.是否能相互转换：**

可以相互转换

##### GB2312来历（来自维基百科，自由的百科全书）：

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。

##### 产生原因：

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。

##### 解决的问题字节数：

EUC-CN通常是用作什么字符编码(例如外部存储器)程序中处理GB2312,从而保持兼容美国信息交换标准代码。 两个字节用来表示每一个字符都没有发现美国信息交换标准代码。 第一个字节的值从0 xa1-0xf7(161 - 247),而第二个字节的值从0 xa1-0xfe(161 - 254)。 由于所有这些范围超出ASCII,utf - 8,可以检查是否一个字节是使用EUC-CN时一个多字节构造的一部分,但如果第一个或最后一个字节。

相比utf - 8,GB2312(无论是本地或编码EUC-CN)是存储效率更高:utf - 8用三个字节[(一)](https://en.wikipedia.org/wiki/GB_2312#cite_note-4)每CJK象形文字,GB2312只使用两个。 然而,GB2312并不覆盖尽可能多的象形文字Unicode。 和GB2312更节能,它需要部门到94年相对昂贵,而utf - 8使用便宜移位操作。

将代码点映射到字节,158(0 x98)添加到行号的代码点形成高字节,并添加158列号的代码点形成了低字节。 行号是代码点整数除以94,列代码点模94年。

例如,如果你有GB2312代码点4566(“外”,这意味着外国),高字节将4566/94 + 158 = 206 = 0 xce,和低字节将来自94 + 158 = 158 = 0 xd4 4566%。 所以,完整的编码是0 xced4 = 52948。

**描述：**

有两种实现GB2312在几个不同代码点。

|  |  |  |  |
| --- | --- | --- | --- |
| **EUC-CN** | **GBK / GB18030子集** | **GB2312.TXT** | **角色名称**[[4]](https://en.wikipedia.org/wiki/GB_2312" \l "cite_note-gb2312-80-5):3 |
| A1A4 | U + 00 b7 · 中间点 | U + 30 fb ・ 片假名中间点 | 间隔点; ”[分隔符点](https://en.wikipedia.org/wiki/Chinese_punctuation#Middle_dot)” |
| A1AA | U + 2014 - - - - - - 长破折号 | U + 2015 - - - - - - 单杠 | 破折号; ”[长破折号](https://en.wikipedia.org/wiki/Chinese_punctuation#Em_dash)” |

的**GBK / GB18030子集**兼容两种GBK和GB18030;**GB2312.TXT**从ftp.unicode.org then-official实现,[[5]](https://en.wikipedia.org/wiki/GB_2312#cite_note-many-mappings-6)2011年8月以来已经过时了吗[[6]](https://en.wikipedia.org/wiki/GB_2312" \l "cite_note-7)2016年9月和失踪。 更多的供应商映射存在。[[5]](https://en.wikipedia.org/wiki/GB_2312#cite_note-many-mappings-6)

在2015年,微软。 净框架是使用*子集*。加护病房,[[7]](https://en.wikipedia.org/wiki/GB_2312#cite_note-8)iconv - 1.14,[[8]](https://en.wikipedia.org/wiki/GB_2312#cite_note-9)activeperl php - 5.6——5.20,Java 1.7和Python 3.4[[9]](https://en.wikipedia.org/wiki/GB_2312#cite_note-10)正在使用*GB2312.TXT*。 Ruby 2.2兼容的实现; 内部矛盾的角色转换*子集*。 指定一个W3C的技术建议*GBK*编码为流标签推断gb2312,进而使用*GB18030*译码器。[[10]](https://en.wikipedia.org/wiki/GB_2312" \l "cite_note-11)

**兼容哪些字符集：**

可以通过uincode转换

**UTF-8来历**（来自维基百科，自由的百科全书）**：**

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）：

128个US-ASCII字符只需一个字节编码（Unicode范围由U+0000至U+007F）。

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）。

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）。

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。不过，如果使用一些传统的压缩系统，比如DEFLATE，则这些不同编码系统间的的差异就变得微不足道了。若顾及传统压缩算法在压缩较短文字上的效果不大，可以考虑使用Unicode标准压缩格式（SCSU）。

互联网工程工作小组（IETF）要求所有互联网协议都必须支持UTF-8编码[1]。互联网邮件联盟（IMC）建议所有电子邮件软件都支持UTF-8编码。

**产生原因：**

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。

**字节数描述：**

例如，希伯来语字母aleph（א）的Unicode代码是U+05D0，按照以下方法改成UTF-8：

例如，希伯来语字母aleph（א）的Unicode代码是U+05D0，按照以下方法改成UTF-8：

它属于U+0080到U+07FF区域，这个表说明它使用双字节，110yyyyy 10zzzzzz.

十六进制的0x05D0换算成二进制就是101-1101-0000.

这11位数按顺序放入"y"部分和"z"部分：11010111 10010000.

最后结果就是双字节，用十六进制写起来就是0xD7 0x90，这就是这个字符aleph（א）的UTF-8编码。

所以开始的128个字符（US-ASCII）只需一字节，接下来的1920个字符需要双字节编码，包括带附加符号的拉丁字母，希腊字母，西里尔字母，科普特语字母，亚美尼亚语字母，希伯来文字母和阿拉伯字母的字符。基本多文种平面中其余的字符使用三个字节，剩余字符使用四个字节。

**兼容哪些字符集：**兼容ASCii 可以相互转换

**UTF-16来历(维基百科)：**

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。

**产生原因（来源于知乎）：**兼容 ASCII  
2. 能适应许多 C 库中的 \0 结尾惯例  
3. 没有字节序问题  
4. 良好的多语种支持（相对 GBK 等跟语种绑定的编码方式）  
5. 以英文和西文符号比较多的场景下（例如 HTML/XML），编码较短

**解决的问题：**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。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

**字节数描述：**辅助平面（Supplementary Planes）中的码位，在UTF-16中被编码为一对16比特长的码元（即32bit,4Bytes），称作代理对（surrogate pair），具体方法是：

码位减去0x10000,得到的值的范围为20比特长的0..0xFFFFF.

高位的10比特的值（值的范围为0..0x3FF）被加上0xD800得到第一个码元或称作高位代理（high surrogate），值的范围是0xD800..0xDBFF.由于高位代理比低位代理的值要小，所以为了避免混淆使用，Unicode标准现在称高位代理为前导代理（lead surrogates）。

低位的10比特的值（值的范围也是0..0x3FF）被加上0xDC00得到第二个码元或称作低位代理（low surrogate），现在值的范围是0xDC00..0xDFFF.由于低位代理比高位代理的值要大，所以为了避免混淆使用，Unicode标准现在称低位代理为后尾代理（trail surrogates）。

**兼容哪些字符集：**兼容UCS-2可以互相转换