**几种编码格式简述**

**班级：软工11502**

**学号：201503592**

**姓名：罗荔豪**

**一.unicode**

*①起源与发展*

*②描述*

*③编码方式*

*④字集的转换*

*⑤解决的问题*

*⑥兼容性*

*⑦引用的书籍以及参考的资料*

**二.GB2312**

*①起源*

*②描述*

*③编码方式*

*④兼容性和转换方式*

*⑤引用的文献和参考资料*

**三.Big-5**

*①描述*

*②产生和发展的原因*

*③存储方式和编码方式*

*④遗留下的问题和弊端*

*⑤兼容性*

*⑥编码方式*

*⑦引用参考资料*

**四.Utf-8**

*①产生的原因*

*②描述和编码方式*

*③转换方式*

*④兼容性*

**五.utf-16**

*①发展缘由*

*②编码和存储方式*

*③描述*

*④兼容性*

*⑤.参考文献*

**一.unicode**

*①起源与发展*

Unicode是为了解决传统的字元编码方案的局限而产生的，例如ISO 8859-1所定义的字元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相容的情况。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），但却无法同时支援多语言环境（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（详见中日韩统一表意文字主题）。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字元保留给ISO 8859-1所定义的字元，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元重复编到不同的字元码中去，使得旧有纷杂的编码方式得以和Unicode编码间互相直接转换，而不会遗失任何资讯。举例来说，全形格式区段包含了主要的拉丁字母的全形格式，在中文、日文、以及韩文字形当中，这些字元以全形的方式来呈现，而不以常见的半形形式显示，这对竖排文字和等宽排列文字有重要作用。

*②描述*

在表示一个Unicode的字元时，通常会用“U+”然后紧接着一组十六进位的数字来表示这一个字元。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）里的所有字元，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元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。

*③编码方式*

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。实际上目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。Unicode的实现方式不同于编码方式。一个字符的Unicode编码是确定的。但是在实际传输过程中，由于不同系统平台的设计不一定一致，以及出于节省空间的目的，对Unicode编码的实现方式有所不同。Unicode的实现方式称为Unicode转换格式（Unicode Transformation Format，简称为UTF）

*④字集的转换*

例如，如果一个仅包含基本7位ASCII字符的Unicode文件，如果每个字符都使用2字节的原Unicode编码传输，其第一字节的8位始终为0。这就造成了比较大的浪费。对于这种情况，可以使用UTF-8编码，这是一种变长编码，它将基本7位ASCII字符仍用7位编码表示，占用一个字节（首位补0）。而遇到与其他Unicode字符混合的情况，将按一定算法转换，每个字符使用1-3个字节编码，并利用首位为0或1进行识别。这样对以7位ASCII字符为主的西文文档就大幅节省了编码长度（具体方案参见UTF-8）。类似的，对未来会出现的需要4个字节的辅助平面字符和其他UCS-4扩充字符，2字节编码的UTF-16也需要通过一定的算法进行转换。

再如，如果直接使用与Unicode编码一致（仅限于BMP字符）的UTF-16编码，由于每个字符占用了两个字节，在麦金塔电脑（Mac）机和个人电脑上，对字节顺序的理解是不一致的。这时同一字节流可能会被解释为不同内容，如某字符为十六进制编码4E59，按两个字节拆分为4E和59，在Mac上读取时是从低字节开始，那么在Mac OS会认为此4E59编码为594E，找到的字符为“奎”，而在Windows上从高字节开始读取，则编码为U+4E59的字符为“乙”。就是说在Windows下以UTF-16编码保存一个字符“乙”，在Mac OS环境下开启会显示成“奎”。此类情况说明UTF-16的编码顺序若不加以人为定义就可能发生混淆，于是在UTF-16编码实现方式中使用了大端序（Big-Endian，简写为UTF-16 BE）、小端序（Little-Endian，简写为UTF-16 LE）的概念，以及可附加的位元组顺序记号解决方案，目前在PC机上的Windows系统和Linux系统对于UTF-16编码默认使用UTF-16 LE。（具体方案参见UTF-16）

此外Unicode的实现方式还包括UTF-7、Punycode、CESU-8、SCSU、UTF-32、GB18030等，这些实现方式有些仅在一定的国家和地区使用，有些则属于未来的规划方式。目前通用的实现方式是UTF-16小端序（LE）、UTF-16大端序（BE）和UTF-8。在微软公司Windows XP附带的记事本（Notepad）中，“另存为”对话框可以选择的四种编码方式除去非Unicode编码的ANSI（对于英文系统即ASCII编码，中文系统则为GB2312或Big5编码）外，其余三种为“Unicode”（对应UTF-16 LE）、“Unicode big endian”（对应UTF-16 BE）和“UTF-8”。

*⑤解决的问题*

目前辅助平面的工作主要集中在第二和第三平面的中日韩统一表意文字中，因此包括GBK、GB18030、Big5等简体中文、繁体中文、日文、韩文以及越南喃字的各种编码与Unicode的协调性被重点关注。考虑到Unicode最终要涵盖所有的字符。从某种意义而言，这些编码方式也可视作Unicode的出现于其之前的既成事实的实现方式，如同ASCII及其扩展Latin-1一样，后两者的字符在16位Unicode编码空间中的编码第一字节各位全为0，第二字节编码与原编码完全一致。但上述东亚语言编码与Unicode编码的对应关系要复杂得多。

XML及其子集XHTML采用UTF-8作为标准字集，理论上我们可以在各种支持XML标准的浏览器上显示任何地区文字的网页，只要电脑本身安装有合适的字体即可。可以利用&#nnn;的格式显示特定的字符。nnn代表该字符的十进制Unicode代码。如果采用十六进制代码，在编码之前加上x字符即可。但部分旧版本的浏览器可能无法识别十六进制代码。

*⑥兼容性*

过去电脑编码的8位标准，使每个国家都只按国家使用的字符而编定各自的编码系统；而对于部份字符系统比较复杂的语言，如越南语，又或者东亚国家的大型字符集，都不能在8位的环境下正常显示。只是最近才有在文本中对十六进制的支持，那么旧版本的浏览器显示那些字符或许可能有问题-大概首先会遇到的一个问题只是在对于大于8位Unicode字符的显示。解决这个问题的普遍做法仍然是将其中的十六进制码转换成一个十进制码（例如：♠用&#9824;代替&#x2660;）。

也有一些字符集标准将一些常用的标志存放在字符内码外面，那么你可能使用象—这样的文本标志来表示一个长划（—）的情况，即使它的字符内码已经被使用，这些标准也不包含那个字符。

然而部分由于Unicode版本发展原因，很多浏览器只能显示UCS-2完整字符集，也即现在使用的Unicode版本中的一个小子集。下表可以检验您的浏览器如何显示各种Unicode代码：



一些多语言支持的网页浏览器，比如微软Windows系统的Internet Explorer 5.5及以上版本，以及跨平台的浏览器Mozilla／Netscape 6，可以在安装时根据需要动态地使用相应的字符集，预先安装了合适的语言包，就可以同时显示页面上的各种Unicode字符。Internet Explorer 5.5还提出用户可以在需要新字体时，即装即用。另外的浏览器如Netscape Navigator 4.77，则只能显示跟页面编码相应字符集中的文字。当你使用后一种浏览器时，你不大可能预先安装所有的字体，即使有了字体，浏览器也不一定能将这些字体完全应用起来。可能遇到的情况是，这种浏览器只能够显示部分文字，因为它们是按照标准进行编码尽管理论上在兼容的系统中，只要有了相应的Code2000字体，就可以正确显示。一种变通的办法，是将某些少见的字符，通过“名称实体引用”的方式来使用。

*⑦引用的书籍以及参考的资料*

1. The Unicode Standard 第五版. Addison-Wesley Professional. ISBN 0321480910.

2. 至： 2.0 2.1 Unicode 10.0.0. Unicode Consortium. [2017-06-20].

3. Unicode

4. Unicode Data 1.0.0. [2010-03-16].

5. Unicode Data 1.0.1. [2010-03-16].

6. Unicode Data 1995. [2010-03-16].

7. Unicode Data-2.0.14. [2010-03-16].

8. Unicode Data-2.1.2. [2010-03-16].

9. Unicode Data-3.0.0. [2010-03-16].

10. Unicode Data-3.1.0. [2010-03-16].

11. Unicode Data-3.2.0. [2010-03-16].

12. Unicode Data-4.0.0. [2010-03-16].

13. Unicode Data. [2010-03-16].

14. Unicode Data 5.0.0. [2010-03-17].

15. Unicode Data 5.1.0. [2010-03-17].

16. Unicode Data 5.2.0. [2010-03-17].

17. Unicode Data 6.0.0. [2010-10-11].

18. Unicode Data 6.1.0. [2012-01-31].

19. Unicode Data 6.2.0. [2012-09-26].

20. Unicode Data 6.3.0. [2013-09-30].

21. Unicode Data 7.0.0. [2014-06-15].

22. Unicode 8.0.0. Unicode Consortium. [2015-06-17].

23. Unicode Data 8.0.0. [2015-06-17].

24. Unicode 9.0.0. Unicode Consortium. [2016-06-21].

25. Unicode Data 9.0.0. [2016-06-21].

26. Unicode Data 10.0.0. [2017-06-20].

27. The Unicode Standard, Version 6.2, 2012 Unicode, Inc. Chapter 2 General Structure 2.2 Unicode Design Principles

28. Why doesn't the Unicode Standard adopt a compositional model for encoding Han ideographs? Wouldn't that save a large number of code points?GB2312

**二.GB2312**

*①起源*

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。

*②描述*

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。

*③编码方式*

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。例如“啊”字在大多数程序中，会以两个字节，0xB0（第一个字节）0xA1（第二个字节）储存。（与区位码对比：0xB0=0xA0+16,0xA1=0xA0+1）。

*④兼容性和转换方式*

有两种不同的GB 2312实现，在它们之间存在少量的差别，其中至少有一个是错误的。字节序GBK子集GB2312.TXT字符名称A1A4 U+00B7 “·” MIDDLE DOT U+30FB “・” KATAKANA MIDDLE DOT 间隔点 A1AA U+2014 “—” EM DASH U+2015 “―” HORIZONTAL BAR 破折号 GBK子集与GBK/GB 18030兼容，GB2312.TXT则不兼容。后者基于ftp.unicode.org曾经提供的GB2312.TXT实现，于2011年由官方弃用，[4]2016年9月时已无原文件踪迹。此外还有很多种厂商实现。截至2015年，微软.NET使用的是“GBK子集”实现。ICU、libiconv-1.14、php-5.6、ActivePerl-5.20、Java 1.7、Python 3.4都使用“GB2312.TXT”实现。Ruby 2.2兼容两者编码，但内部使用“GBK子集”实现。W3C的编码技术指南规定，应将gb2312字节流视为GBK编码，与GB18030一并使用同一解码器解码。

*⑤引用的文献和参考资料*

1、 1965年《印刷通用汉字字形表》将“锺”（使用简化偏旁）注明为“作为人名地名及引用文言文的时候仍有需要”；2013年《通用规范汉字表》收录“锺”，可用於姓名

2.GB 2312-1980: Information technology—Chinese ideogram coded character set for information interchange (basic set). [2 October 2016].

3. 3 Haible, Bruno. GB2312 (Conversion Tables). [29 September 2016].

4. Readme - MAPPINGS/OBSOLETE/EASTASIA. 9 August 2001 [29 September 2016].

5. java-EUC\_CN-1.3\_P.ucm. [29 September 2016].

6. libiconv:lib/gb2312.h. GNU Savannah. [29 September 2016].

7. Issue 24036. Python Bug Tracker.

8. Encoding § Names and labels. W3C. [29 September 2016].

**三.Big-5**

*①描述*

“大五码”（Big5）是由台湾财团法人资讯工业策进会为五大中文套装软体所设计的中文共通内码，在1983年12月完成公告[2][3]，隔年3月，资讯工业策进会与台湾13家厂商签定“16位元个人电脑套装软体合作开发（BIG-5）计画（五大中文套装软体）”[4]，因为此中文内码是为台湾自行制作开发之“五大中文套装软体”所设计的，所以就称为Big5中文内码[5][6][7][8]。五大中文套装软体虽然并没有如预期的取代国外的套装软体，但随著采用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码深远地影响正体中文电脑内码，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。

*②产生和发展的原因*

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体无法推广，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资讯时代，所采行的一个计画；同时，这个计画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产生前，研发中文电脑的朱邦复认为内码字集应该广纳所有的正异体字，以顾及如户政等应用上的需要，故在当时的内码会议中，建议希望采用他的五万多字的字库。工程师认为虽其技术可行，但是三个位元组（超过两个位元组）长度的内码却会造成英文萤幕画面映射成中文画面会发生文字无法对齐的问题，因为当时盛行之倚天中文系统画面系以两个位元组文字宽度映射成一个中文字图样，英文软体中只要以两个英文字宽度去显示一个中文字，画面就不会乱掉，造成中文系统业者偏爱二个位元组长度的内码[9]；此外以仓颉输入码压缩成的内码不具排序等功能，因此未被采用。1983年有人诬指朱邦复为共产党，其研究成果更不可能获采用。

在Big5码诞生后，大部分台湾的电脑软体都使用了Big5码，加上后来倚天中文系统的高度普及，使后来的微软Windows 3.x等亦予以采用。虽然后来台湾还有各种想要取代Big5码，像是倚天中文系统所推行的倚天码、台北市电脑公会所推动的公会码等，但是由于Big5字码已沿用多年，因此在习惯不易改变的情况下，始终无法成为主流字码。而台湾后来发展的国家标准CNS 11643中文标准交换码由于非一般的内码系统，是以交换使用为目的，受先天所限，必须使用至少三个位元组来表示一个汉字，所以普及率远远不及Big5码。

在1990年代初期，当中国大陆的电邮和转码软体还未普遍之时，在深圳的港商和台商公司亦曾经使用Big5系统，以方便与总部的文件交流、以及避免为大陆的办公室再写一套不同内码的系统。使用简体中文的社群，最常用的是GB 2312、GBK及其后续的国标码（GB 18030）。

除了台湾外，其他使用繁体汉字的地区，如香港（香港增补字符集）、澳门，及使用繁体汉字的海外华人，都曾普遍使用Big5码做为中文内码及交换码。

*③存储方式和编码方式*

Big5码是一套双位元组字符集，使用了双八码储存方法，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。在Big5的分区中：0x8140-0xA0FE 保留给使用者自定义字元（造字区） 0xA140-0xA3BF 标点符号、希腊字母及特殊符号，包括在0xA259-0xA261，安放了九个计量用汉字：兙兛兞兝兡兣嗧瓩糎。 0xA3C0-0xA3FE 保留。此区没有开放作造字区用。 0xA440-0xC67E 常用汉字，先按笔划再按部首排序。

0xC6A1-0xC8FE 保留给使用者自定义字元（造字区） 0xC940-0xF9D5 次常用汉字，亦是按笔划再按部首排序。 0xF9D6-0xFEFE 保留给使用者自定义字元（造字区） 值得留意的是，Big5重复收录了两个相同的字：“兀、兀”（0xA461[U+5140]及0xC94A[U+FA0C]）、“嗀、嗀”（0xDCD1[U+55C0]及0xDDFC[U+FA0D]）。此外“十”、“卅”也在符号区又重复了一次，在检索系统中常会造成查询不到字。

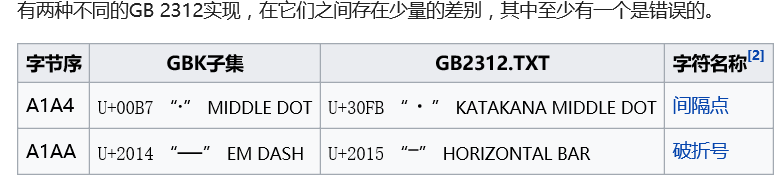
*④遗留下的问题和弊端*

自中文电脑流行后，由于很多日常用字被视为异体字而未收录。很多人，甚至电视台的字幕、报纸的用字习惯都被改变。例如台湾教育部视“着”为“著”的异体字，故没有收录“着”字。康熙字典中的一些部首用字（如“亠”、“疒”、“辵”、“癶”等）、常见的人名用字（如“堃”（台湾前行政院长游锡堃）、“煊”（台湾前监察院院长、前财政部长王建煊）、“栢”（歌手张柏芝）、“峯”（歌手吴青峯、林峯）、“喆”（歌手陶喆）等），虽被中文社会广泛采用，也没有收录到Big5之中。另外像台湾的廍，闽南语指制糖所，常见于乡间地名。但由于大五码未收此字，也被被“廓”、“部”代替。在互联网上，实在不难看到人们把游锡堃、王建煊、张柏芝、陶喆等名字，写成为“游锡方方土”、“王建火宣”、“张木百芝”和“陶吉吉”等写法。电视上日本动画的中文字幕中也会看到像“木堅”（樫）这样的字

*⑤兼容性*

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，近年的作业系统和应用程式（如苹果电脑Mac OS X和以Cocoa API撰写之程式、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。可惜现时仍有一些旧的软件（如Visual Basic 6、部分Telnet或BBS软件），未能支援Unicode编码，故相信Big5缺字的问题仍会困扰用户一段时间，直至所有程式都能改用Unicode为止。

*⑥编码方式*



GBK子集与GBK/GB 18030兼容，GB2312.TXT则不兼容。后者基于ftp.unicode.org曾经提供的GB2312.TXT实现，于2011年由官方弃用，[4]2016年9月时已无原文件踪迹。此外还有很多种厂商实现。截至2015年，微软.NET使用的是“GBK子集”实现。ICU[5]、libiconv-1.14、[6]php-5.6、ActivePerl-5.20、Java 1.7、Python 3.4[7]都使用“GB2312.TXT”实现。Ruby 2.2兼容两者编码，但内部使用“GBK子集”实现。W3C的编码技术指南规定，应将gb2312字节流视为GBK编码，与GB18030一并使用同一解码器解码。

*⑦引用参考资料*

1.1965年《印刷通用汉字字形表》将“锺”（使用简化偏旁）注明为“作为人名地名及引用文言文的时候仍有需要”；2013年《通用规范汉字表》收录“锺”，可用於姓名

2.GB 2312-1980: Information technology—Chinese ideogram coded character set for information interchange (basic set). [2 October 2016].

3.Haible, Bruno. GB2312 (Conversion Tables). [29 September 2016].

4.Readme - MAPPINGS/OBSOLETE/EASTASIA. 9 August 2001 [29 September 2016].

5.java-EUC\_CN-1.3\_P.ucm. [29 September 2016].

6.libiconv:lib/gb2312.h. GNU Savannah. [29 September 2016].

7.Issue 24036. Python Bug Tracker.

8. Encoding § Names and labels. W3C. [29 September 2016].

**四.Utf-8**

*①产生的原因*

1992年初，为建立良好的位元组串编码系统以供多位元组字元集使用，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位元的字元使用的位元组串编码系统。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以相容旧系统的概念。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。它具备可更快速实作的特性，并引入一项新的改进。其中，7位元的ASCII符号只代表原来的意思，所有多位元组序列则会包含第8位元的符号，也就是所谓的最高有效位元。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，贝尔实验室九号计画作业系统工作小组的肯·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计画当中，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作这方面的规格。

*②描述和编码方式*

UTF-8使用一至六个位元组为每个字符编码（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字符只需一个位元组编码（Unicode范围由U+0000至U+007F）。

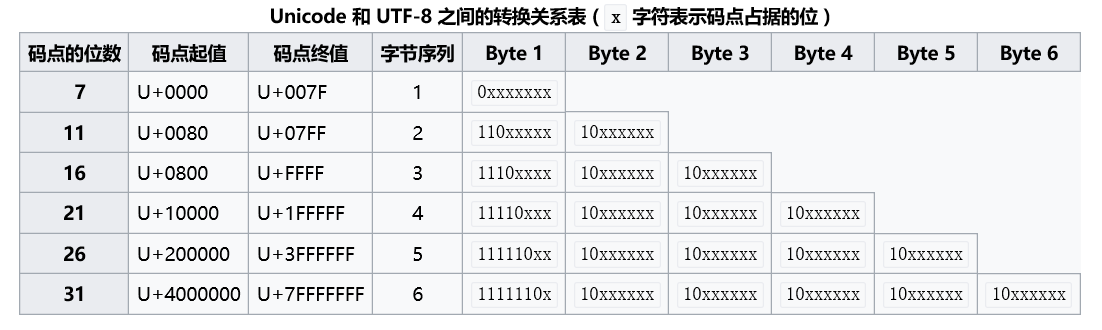
2.带有附加符号的拉丁文、希腊文、西里尔字母、亚美尼亚语、希伯来文、阿拉伯文、叙利亚文及它拿字母则需要两个位元组编码（Unicode范围由U+0080至U+07FF）。

3.其他基本多文种平面（BMP）中的字元（这包含了大部分常用字，如大部分的汉字）使用三个位元组编码（Unicode范围由U+0800至U+FFFF）。4.其他极少使用的Unicode 辅助平面的字元使用四至六位元组编码（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元而言，UTF-8使用四至六个位元组来编码似乎太耗费资源了。但UTF-8对所有常用的字元都可以用三个位元组表示，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位元组来编码，所以要决定UTF-8或UTF-16哪种编码比较有效率，还要视所使用的字元的分布范围而定。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）。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个字元（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元。基本多文种平面中其余的字元使用三个字节，剩余字符使用四个字节。根据这种方式可以处理更大数量的字元。原来的规范允许长达6字节的序列，可以覆盖到31位元（通用字符集原来的极限）。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。根据这些规范，以下字节值将无法出现在合法UTF-8序列中：



*③转换方式*

UTF-8是UNICODE的一种变长度的编码表达方式《一般UNICODE为双位元组（指UCS2）》，它由肯·汤普逊（Ken Thompson）于1992年建立，现在已经标准化为RFC 3629。UTF-8就是以8位为单元对UCS进行编码，而UTF-8不使用大尾序和小尾序的形式，每个使用UTF-8储存的字符，除了第一个字节外，其馀字节的头两个位元都是以"10"开始，使文字处理器能够较快地找出每个字符的开始位置。但为了与以前的ASCII码相容（ASCII为一个位元组），因此UTF-8选择了使用可变长度字节来储存Unicode：

在ASCII码的范围，用一个位元组表示，超出ASCII码的范围就用位元组表示，这就形成了我们上面看到的UTF-8的表示方法，这様的好处是当UNICODE文件中只有ASCII码时，储存的文件都为一个位元组，所以就是普通的ASCII文件无异，读取的时候也是如此，所以能与以前的ASCII文件相容。大于ASCII码的，就会由上面的第一位元组的前几位表示该unicode字元的长度，比如110xxxxx前三位的二进位表示告诉我们这是个2BYTE的UNICODE字元；1110xxxx是个三位的UNICODE字元，依此类推；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。

ASCII字母继续使用1字节储存，重音文字、希腊字母或西里尔字母等使用2字节来储存，而常用的汉字就要使用3字节。辅助平面字元则使用4字节。

在UTF-8文件的开首，很多时都放置一个U+FEFF字符（UTF-8以EF,BB,BF代表），以显示这个文字档案是以UTF-8编码。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元组序列的特质：单位元组字符的最高有效位元永远为0。多位元组序列中的首个字元组的几个最高有效位元决定了序列的长度。最高有效位为110的是2位元组列，而1110的是三位元组序列，如此类推。多位元组序列中其馀的位元组中的首两个最高有效位元为10。保了以位元组为基础的部份字串比对（sub-string match）方法可以适用于在文字中搜寻字或词。有些比较旧的可变长度8位元编码（如Shift JIS）没有这个特质，故字串比对的算法变得相当复杂。虽然这增加了UTF-8编码的字串的信息冗馀，但是利多于弊。另外，资料压缩并非Unicode的目的，所以不可混为一谈。即使在传送过程中有部份位元组因错误或干扰而完全遗失，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，由于其位元组序列设计，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。一段两位元组随机序列碰巧为合法的UTF-8而非ASCII的机率为32分1。对于三位元组序列的机率为256分1，对更长的序列的机率就更低了。

*④兼容性*

XML及其子集XHTML采用UTF-8作为标准字集，理论上我们可以在各种支持XML标准的浏览器上显示任何地区文字的网页，只要电脑本身安装有合适的字体即可。可以利用&#nnn;的格式显示特定的字符。nnn代表该字符的十进制Unicode代码。如果采用十六进制代码，在编码之前加上x字符即可。但部分旧版本的浏览器可能无法识别十六进制代码。过去电脑编码的8位标准，使每个国家都只按国家使用的字符而编定各自的编码系统；而对于部份字符系统比较复杂的语言，如越南语，又或者东亚国家的大型字符集，都不能在8位的环境下正常显示。只是最近才有在文本中对十六进制的支持，那么旧版本的浏览器显示那些字符或许可能有问题-大概首先会遇到的一个问题只是在对于大于8位Unicode字符的显示。解决这个问题的普遍做法仍然是将其中的十六进制码转换成一个十进制码（例如：♠用&#9824;代替&#x2660;）。也有一些字符集标准将一些常用的标志存放在字符内码外面，那么你可能使用象—这样的文本标志来表示一个长划（—）的情况，即使它的字符内码已经被使用，这些标准也不包含那个字符。然而部分由于Unicode版本发展原因，很多浏览器只能显示UCS-2完整字符集，也即现在使用的Unicode版本中的一个小子集。下表可以检验您的浏览器如何显示各种Unicode代码：一些多语言支持的网页浏览器，比如微软Windows系统的Internet Explorer 5.5及以上版本，以及跨平台的浏览器Mozilla／Netscape 6，可以在安装时根据需要动态地使用相应的字符集，预先安装了合适的语言包，就可以同时显示页面上的各种Unicode字符。Internet Explorer 5.5还提出用户可以在需要新字体时，即装即用。另外的浏览器如Netscape Navigator 4.77，则只能显示跟页面编码相应字符集中的文字。当你使用后一种浏览器时，你不大可能预先安装所有的字体，即使有了字体，浏览器也不一定能将这些字体完全应用起来。可能遇到的情况是，这种浏览器只能够显示部分文字，因为它们是按照标准进行编码，尽管理论上在兼容的系统中，只要有了相应的Code2000字体，就可以正确显示。一种变通的办法，是将某些少见的字符，通过“名称实体引用”的方式来使用。。

UTF-8是UNICODE的一种变长度的编码表达方式《一般UNICODE为双位元组（指UCS2）》，它由肯·汤普逊（Ken Thompson）于1992年建立，现在已经标准化为RFC 3629。UTF-8就是以8位为单元对UCS进行编码，而UTF-8不使用大尾序和小尾序的形式，每个使用UTF-8储存的字符，除了第一个字节外，其馀字节的头两个位元都是以"10"开始，使文字处理器能够较快地找出每个字符的开始位置。

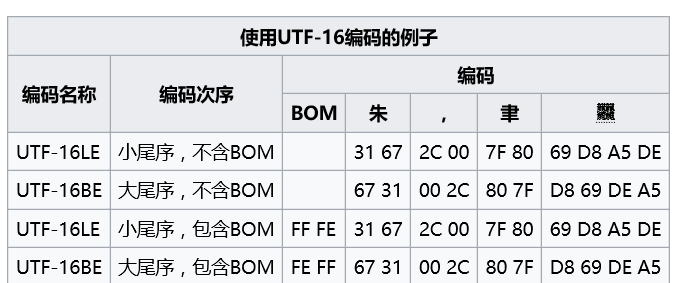
**五.utf-16**

*①发展缘由*

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。

*②编码和存储方式*

UTF-16的大尾序和小尾序储存形式都在用。一般来说，以Macintosh制作或储存的文字使用大尾序格式，以Microsoft或Linux制作或储存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），以显示这个文字档案是以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。UTF-16的大尾序和小尾序储存形式都在用。一般来说，以Macintosh制作或储存的文字使用大尾序格式，以Microsoft或Linux制作或储存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），以显示这个文字档案是以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。以下的例子有四个字符：“朱”（U+6731）、半角逗号（U+002C）、“聿”（U+807F）、“𪚥”（U+2A6A5）。



*③描述*

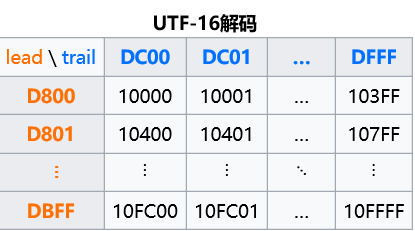
Unicode的编码空间从U+0000到U+10FFFF，共有1,112,064个码位（code point）可用来映射字符. Unicode的编码空间可以划分为17个平面（plane），每个平面包含216（65,536）个码位。17个平面的码位可表示为从U+xx0000到U+xxFFFF，其中xx表示十六进制值从0016到1016，共计17个平面。第一个平面称为基本多语言平面（Basic Multilingual Plane, BMP），或称第零平面（Plane 0）。其他平面称为辅助平面（Supplementary Planes）。基本多语言平面内，从U+D800到U+DFFF之间的码位区段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。从U+0000至U+D7FF以及从U+E000至U+FFFF的码位

第一个Unicode平面（码位从U+0000至U+FFFF）包含了最常用的字符。该平面被称为基本多语言平面，缩写为BMP（Basic Multilingual Plane, BMP）。UTF-16与UCS-2编码这个范围内的码位为16比特长的单个码元，数值等价于对应的码位. BMP中的这些码位是仅有的可以在UCS-2中表示的码位。

从U+10000到U+10FFFF的码位辅助平面（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）。上述算法可理解为：辅助平面中的码位从U+10000到U+10FFFF，共计FFFFF个，即220=1,048,576个，需要20位来表示。如果用两个16位长的整数组成的序列来表示，第一个整数（称为前导代理）要容纳上述20位的前10位，第二个整数（称为后尾代理）容纳上述20位的后10位。还要能根据16位整数的值直接判明属于前导整数代理的值的范围（210=1024)，还是后尾整数代理的值的范围（也是210=1024）。因此，需要在基本多语言平面中保留不对应于Unicode字符的2048个码位，就足以容纳前导代理与后尾代理所需要的编码空间。这对于基本多语言平面总计65536个码位来说，仅占3.125%.由于前导代理、后尾代理、BMP中的有效字符的码位，三者互不重叠，搜索是简单的：一个字符编码的一部分不可能与另一个字符编码的不同部分相重叠。这意味着UTF-16是自同步（self-synchronizing）:可以通过仅检查一个码元就可以判定给定字符的下一个字符的起始码元. UTF-8也有类似优点，但许多早期的编码模式就不是这样，必须从头开始分析文本才能确定不同字符的码元的边界。

由于最常有的字符都在基本多文种平面中，许多软件的处理代理对的部分往往得不到充分的测试。这导致了一些长期的bug与潜在安全漏洞，甚至在广为流行得到良好评价的应用软件从U+D800到U+DFFF的码位Unicode标准规定U+D800..U+DFFF的值不对应于任何字符。但是在使用UCS-2的时代，U+D800..U+DFFF内的值被占用，用于某些字符的映射。但只要不构成代理对，许多UTF-16编码解码还是能把这些不符合Unicode标准的字符映射正确的辨识、转换成合规的码元[2].按照Unicode标准，这种码元序列本来应算作编码错误。

*④.兼容性*

Microsoft Windows操作系统内核对Unicode的支持Windows操作系统内核中的字符表示为UTF-16小尾序，可以正确处理、显示以4字节存储的字符。但是Windows API实际上仅能正确处理UCS-2字符，即仅以2字节存储的，码位小于U+FFFF的Unicode字符。其根源是Microsoft C++语言把wchar\_t数据类型定义为16比特的unsigned short，这就与一个wchar\_t型变量对应一个宽字符，可以存储一个Unicode字符的规定相矛盾。相反，Linux平台的GCC编译器规定一个wchar\_t是4字节长度，可以存储一个UTF-32字符，宁可浪费了很大的存储空间。Windows 9x系统的API仅支持ANSI字符集，只支持部分的UCS-2转换。1996年发布的Windows NT 4.0的API支持UCS-2。Windows 2000开始，Windows系统API开始支持UTF-16，并支持Surrogate Pair；但许多系统控件比如文本框和label等还不支持surrogate pair表示的字符，会显示成两个字符。Windows 7及更新的系统已经良好地支持了UTF-16，包括Surrogate Pair。Windows API支持在UTF-16LE（wchar\_t类型）与UTF-8（代码页CP\_UTF8）之间的转码。

*⑤.参考文献*

1. Code in Apache Xalan 2.7.0 which can fail on surrogate pairs. Apache Foundation. “The code wrongly assumes it is safe to use substring on the input”

2. Python 2.6 decode of UTF16 does this on Linux, and it correctly handles surrogate pairs. All "CESU" decoders do it too, though they also mistranslate correct surrogate pairs into 2 characters