XML中的几种编码集

姓名：吴静文

学号：201503450

单位：软工11501班

指导老师：向华

[第 1 章 4](#_Toc2211)

[第 2 章 6](#_Toc10053)

[2.1GB2312字符集 6](#_Toc12202)

[2.1.1GB2312的来历及原因及解决的问题 6](#_Toc11212)

[2.1.2字节数 6](#_Toc28903)

[2.1.3描述方式 6](#_Toc18627)

[2.1.4兼容性关系 7](#_Toc4247)

[3.1BIG5字符集 7](#_Toc15372)

[3.1.1BIG5的来历及原因及解决的问题 7](#_Toc7501)

[3.1.2字节数 8](#_Toc18781)

[3.1.3描述方式 8](#_Toc9635)

[4.1Unicode 8](#_Toc16287)

[4.1.1Unicode 9](#_Toc13492)

[4.1.2字节数 9](#_Toc24075)

[4.1.3描述方式 10](#_Toc4915)

[5.1UTF-8 11](#_Toc16226)

[5.1.1UTF-8的来历及原因及解决的问题 11](#_Toc31315)

[5.1.2字节数 12](#_Toc25524)

[5.1.3描述方式 12](#_Toc3810)

[6.1UTF-16 14](#_Toc30098)

[6.1.1字节数 15](#_Toc28227)

[6.1.2描述方式 15](#_Toc3489)

[第 3章 参考文献 17](#_Toc9255)

XML中的几种编码集

摘要

编码是信息从一种形式或格式转换为另一种形式的过程。解码，是编码的逆过程。 对于特定的上下文，编码有一些更具体的意义。 编码（Encoding）在认知上是解释传入的刺激的一种基本知觉的过程。技术上来说，这是一个复杂的、多阶段的转换过程，从较为客观的感觉输入（例如光、声）到主观上有意义的体验。

关键字：GB2312，Big5，Unicode，UTF-8/UTF-16

第 1 章 引言

在计算机中，所有的数据只可能是0或者1(用高电平和低电平分别表示1和0)，那么我们通常看到的字符也就只能用0和1来表示呀。于是科学家们(这里指的是美国的科学家)就想出一个办法，把一个特定的数字对应一个特定的字母进行存储和传输，比如我需要存储字母a，那么我存入一个数字97(即在计算机中存入二进制(01100001)，这个过程叫做编码(encode)，而我们在读取数据的时候，当遇到97时，我们就让计算机显示字母a，这个过程叫做解码(decode)。为了大家在数据传输的时候不至于产生误会，那么我们需要让所有的人都使用数字97来代表字母a，所以需要制定一份标准(即码表)，最开始的这个标准叫做ASCII码表。计算机技术到了欧洲，欧洲人发现怎么我们的那么多符号没有编进去啊！所以欧洲"砖家"坐到了一起，开始讨论。发现既然美国人把第一位流出来了，那么我们就用128-255的位置好了。随着通讯越来越多，而老美发现在自己公司需要国际化的时候，自己原来埋的这个雷真的害了自己。于是乎，开始研讨把世界上几乎所有文字全部放在一个码表中，而这个包罗万象的码表就叫做Unicode，即万国码。Unicode是国际组织制定的可以容纳世界上所有文字和符号的字符编码方案。Unicode用数字0-0x10FFFF来映射这些字符，最多可以容纳1114112个字符，或者说有1114112个码位。码位就是可以分配给字符的数字。实际上，在软件制造商的协会（unicode.org）在做这个工作时，国际标准化组织（ISO）在做同样的事情，最后大家都意识到世界上并不需要两个不同的万国码，于是大家坐在一起合并研究的成果，最后的结果就是现在的Unicode。

# 第 2 章

## 2.1GB2312字符集

### 2.1.1GB2312的来历及原因及解决的问题

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。

### 2.1.2字节数

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。 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）。

### 2.1.3描述方式

GB 2312中对所收汉字进行了“分区”处理，每区含有94个汉字／符号。这种表示方式也称为区位码。 01–09区为特殊符号。 16–55区为一级汉字，按拼音排序。 56–87区为二级汉字，按部首／笔画排序。 举例来说，“啊”字是GB 2312之中的第一个汉字，它的区位码就是1601。 10–15区及88–94区则未有编码。但在附录3，则在第10区推荐作为 GB 1988–80 中的94个图形字符区域（即第3区字符之半形版本）。

### 2.1.4兼容性关系

国家简体中文字符集，兼容ASCII。

## 3.1BIG5字符集

### 3.1.1BIG5的来历及原因及解决的问题

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[1]。 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。 Big5虽普及于台湾、香港与澳门等繁体中文通行区，但长期以来并非当地的国家/地区标准或官方标准，而只是业界标准。倚天中文系统、Windows繁体中文版等主要系统的字符集都是以Big5为基准，但厂商又各自增加不同的造字与造字区，派生成多种不同版本。 2003年，Big5被收录到CNS11643中文标准交换码的附录当中，获取了较正式的地位。这个最新版本被称为Big5-2003。

“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[2][3]，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）”[4]，因为此中文内码是为台湾自行制作开发之“五大中文套装软件”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，但是三个字节（超过两个字节）长度的内码却会造成英文显示屏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字节文字宽度映射成一个中文字图样，英文软件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字节长度的内码[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。[10] 在Big5码诞生后，大部分台湾的电脑软件都使用了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，必须使用至少三个字节来表示一个汉字，所以普及率远远不及Big5码。 在1990年代初期，当中国大陆的电子邮件和转码软件还未普遍之时，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。使用简体中文的社区，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。

### 3.1.2字节数

Big5码是一套[双字节字符集](https://zh.wikipedia.org/wiki/%E5%8F%8C%E5%AD%97%E8%8A%82%E5%AD%97%E7%AC%A6%E9%9B%86" \o "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。在Big5的分区中：

|  |  |
| --- | --- |
| 0x8140-0xA0FE | 保留给用户自定义字符（[造字](https://zh.wikipedia.org/wiki/%E9%80%A0%E5%AD%97" \o "造字)区） |
| 0xA140-0xA3BF | [标点符号](https://zh.wikipedia.org/wiki/%E6%A8%99%E9%BB%9E%E7%AC%A6%E8%99%9F" \o "标点符号)、[希腊字母](https://zh.wikipedia.org/wiki/%E5%B8%8C%E8%85%8A%E5%AD%97%E6%AF%8D" \o "希腊字母)及特殊符号， 包括在0xA259-0xA261，安放了九个[计量用汉字](https://zh.wikipedia.org/wiki/%E8%A8%88%E9%87%8F%E7%94%A8%E6%BC%A2%E5%AD%97" \o "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](https://zh.wikipedia.org/wiki/%E5%B8%B8%E7%94%A8%E6%BC%A2%E5%AD%97" \o "常用汉字)，先按[笔划](https://zh.wikipedia.org/wiki/%E7%AD%86%E5%8A%83" \o "笔划)再按[部首](https://zh.wikipedia.org/wiki/%E9%83%A8%E9%A6%96" \o "部首)排序。 |
| 0xC6A1-0xC8FE | 保留给用户自定义字符（造字区） |
| 0xC940-0xF9D5 | [次常用汉字](https://zh.wikipedia.org/wiki/%E6%AC%A1%E5%B8%B8%E7%94%A8%E5%AD%97" \o "次常用字)，亦是先按笔划再按部首排序。 |
| 0xF9D6-0xFEFE | 保留给用户自定义字符（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。

### 3.1.3描述方式

在Big5的分区中：0x8140-0xA0FE保留给用户自定义字符（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 保留给用户自定义字符（造字区0xC940-0xF9D5次常用汉字，亦是先按笔划再按部首排序。 0xF9D6-0xFEFE 保留给用户自定义字符（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。

## 4.1Unicode

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。

### 4.1.1Unicode

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（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。

Unicode伴随着通用字符集的标准而发展，同时也以书本的形式[1]对外发表。Unicode至今仍在不断增修，每个新版本都加入更多新的字符。目前最新的版本为2017年6月20日公布的10.0.0[2]，已经收录超过十万个字符（第十万个字符在2005年获采纳）。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发展由非营利机构统一码联盟负责，该机构致力于让Unicode方案取代既有的字符编码方案。因为既有的方案往往空间非常有限，亦不适用于多语环境。 Unicode备受认可，并广泛地应用于电脑软件的国际化与本地化过程。有很多新科技，如可扩展置标语言(Extensible Markup Language，简称：XML)、Java编程语言以及现代的操作系统，都采用Unicode编码。

### 4.1.2字节数

Unicode编码长度可达到32位，即4字节。第一字节称为组，第二字节称为面，第三字节称为行，第四字节称为点。第0组第0面里的字符可以只用2个字节表示，且涵盖了绝大部分的常用字，所以备受青睐。为了方便称呼，Unicode给它了一个名称——基本多文种平面（BMP）。基本多文种平面值域和上域都是0到FFFF，共计65535个码点。8 9基本多文种平面是Unicode的子集。

虽然Unicode的表示长度是32位，但因为通常表示的是常用字，且常用字在BMP的字库里，所以表示长度往往缩减为16位。

### 4.1.3描述方式

Unicode是国际组织制定的可以容纳世界上所有文字和符号的字符编码方案。目前的Unicode字符分为17组编排，0x0000 至 0xFFFF，每组称为平面（Plane），而每平面拥有65536个码位，共1114112个。然而目前只用了少数平面。UTF-8、UTF-16、UTF-32都是将数字转换到程序数据的编码方案。

通用字符集（Universal Character Set, UCS）是由ISO制定的ISO 10646（或称ISO/IEC 10646）标准所定义的标准字符集。UCS-2用两个字节编码，UCS-4用4个字节编码。

历史上存在两个独立的尝试创立单一字符集的组织，即国际标准化组织（ISO）和多语言软件制造商组成的统一码联盟。前者开发的 ISO/IEC 10646 项目，后者开发的统一码项目。因此最初制定了不同的标准。

1991年前后，两个项目的参与者都认识到，世界不需要两个不兼容的字符集。于是，它们开始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。从Unicode 2.0开始，Unicode采用了与ISO 10646-1相同的字库和字码；ISO也承诺，ISO 10646将不会替超出U+10FFFF的UCS-4编码赋值，以使得两者保持一致。两个项目仍都存在，并独立地公布各自的标准。但统一码联盟和ISO/IEC JTC1/SC2都同意保持两者标准的码表兼容，并紧密地共同调整任何未来的扩展。在发布的时候，Unicode一般都会采用有关字码最常见的字型，但ISO 10646一般都尽可能采用Century字型。

UCS-4根据最高位为0的最高字节分成27=128个group。每个group再根据次高字节分为256个平面（plane）。每个平面根据第3个字节分为256行 （row），每行有256个码位（cell）。group 0的平面0被称作BMP（Basic Multilingual Plane）。如果UCS-4的前两个字节为全零，那么将UCS-4的BMP去掉前面的两个零字节就得到了UCS-2。每个平面有216=65536个码位。Unicode计划使用了17个平面，一共有17×65536=1114112个码位。在Unicode 5.0.0版本中，已定义的码位只有238605个，分布在平面0、平面1、平面2、平面14、平面15、平面16。其中平面15和平面16上只是定义了两个各占65534个码位的专用区（Private Use Area），分别是0xF0000-0xFFFFD和0x100000-0x10FFFD。所谓专用区，就是保留给大家放自定义字符的区域，可以简写为PUA。

平面0也有一个专用区：0xE000-0xF8FF，有6400个码位。平面0的0xD800-0xDFFF，共2048个码位，是一个被称作代理区（Surrogate）的特殊区域。代理区的目的用两个UTF-16字符表示BMP以外的字符。在介绍UTF-16编码时会介绍。

如前所述在Unicode 5.0.0版本中，238605-65534\*2-6400-2048=99089。余下的99089个已定义码位分布在平面0、平面1、平面2和平面14上，它们对应着Unicode定义的99089个字符，其中包括71226个汉字。平面0、平面1、平面2和平面14上分别定义了52080、3419、43253和337个字符。平面2的43253个字符都是汉字。平面0上定义了27973个汉字。

在Unicode中：汉字“字”对应的数字是23383（十进制），十六进制表示为5B57。在Unicode中，我们有很多方式将数字23383表示成程序中的数据，包括：UTF-8、UTF-16、UTF-32。UTF是“Unicode Transformation Format”的缩写，可以翻译成Unicode字符集转换格式，即怎样将Unicode定义的数字转换成程序数据。

这里用char、char16\_t、char32\_t分别表示无符号8位整数，无符号16位整数和无符号32位整数。UTF-8、UTF-16、UTF-32分别以char、char16\_t、char32\_t作为编码单位。（注： char16\_t 和 char32\_t 是 C++ 11 标准新增的关键字。如果你的编译器不支持 C++ 11 标准，请改用 unsigned short 和 unsigned long。）“汉字”的UTF-8编码需要6个字节。“汉字”的UTF-16编码需要两个char16\_t，大小是4个字节。“汉字”的UTF-32编码需要两个char32\_t，大小是8个字节。根据字节序的不同，UTF-16可以被实现为UTF-16LE或UTF-16BE，UTF-32可以被实现为UTF-32LE或UTF-32BE。下面介绍UTF-8、UTF-16、UTF-32、字节序和BOM。

## 5.1UTF-8

### 5.1.1UTF-8的来历及原因及解决的问题

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。

原因：

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。

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。

### 5.1.2字节数

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。

### 5.1.3描述方式

目前有好几份关于UTF-8详细规格的文件，但这些文件在定义上有些许的不同：

* [RFC 3629](https://tools.ietf.org/html/rfc3629) / STD 63（2003），这份文件制定了UTF-8是标准的[互联网](https://zh.wikipedia.org/wiki/%E7%B6%B2%E9%9A%9B%E7%B6%B2%E8%B7%AF" \o "互联网)[协议](https://zh.wikipedia.org/wiki/%E7%BD%91%E7%BB%9C%E5%8D%8F%E8%AE%AE" \o "网络协议)元素
* *第四版，The Unicode Standard*，§3.9－§3.10（2003）
* ISO/IEC 10646-1:2000附加文件D（2000）

它们取代了以下那些被淘汰的定义：

* ISO/IEC 10646-1:1993修正案2／附加文件R（1996）
* *第二版，The Unicode Standard*，附录A（1996）
* [RFC 2044](https://tools.ietf.org/html/rfc2044)（1996）
* [RFC 2279](https://tools.ietf.org/html/rfc2279)（1998）
* *第三版，The Unicode Standard*，§2.3（2000）及勘误表#1：UTF-8 Shortest Form（2000）
* *Unicode Standard附加文件#27: Unicode 3.1*（2001）

事实上，所有定义的基本原理都是相同的，它们之间最主要的不同是支持的字符范围及无效输入的处理方法。

Unicode[字符](https://zh.wikipedia.org/wiki/UCS" \o "UCS)的[比特](https://zh.wikipedia.org/wiki/%E4%BD%8D%E5%85%83" \o "比特)被分区为数个部分，并分配到UTF-8的[字节](https://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82" \o "字节)串中较低的[比特](https://zh.wikipedia.org/wiki/%E4%BD%8D%E5%85%83" \o "比特)的位置。在U+0080的以下[字符](https://zh.wikipedia.org/wiki/UCS" \o "UCS)都使用内含其字符的单[字节](https://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82" \o "字节)编码。这些编码正好对应7比特的ASCII字符。在其他情况，有可能需要多达4个字符组来表示一个字符。这些多字节的[最高有效比特](https://zh.wikipedia.org/wiki/%E6%9C%80%E9%AB%98%E6%9C%89%E6%95%88%E4%BD%8D%E5%85%83" \o "最高有效比特)会设置成1，以防止与7比特的ASCII字符混淆，并保持标准的字节主导字符串运作顺利。

|  |  |  |  |
| --- | --- | --- | --- |
| **代码范围 [十六进制](https://zh.wikipedia.org/wiki/%E5%8D%81%E5%85%AD%E9%80%B2%E5%88%B6" \o "十六进制)** | **标量值（scalar value） [二进制](https://zh.wikipedia.org/wiki/%E4%BA%8C%E9%80%B2%E5%88%B6" \o "二进制)** | **UTF-8 [二进制](https://zh.wikipedia.org/wiki/%E4%BA%8C%E9%80%B2%E5%88%B6" \o "二进制)／[十六进制](https://zh.wikipedia.org/wiki/%E5%8D%81%E5%85%AD%E9%80%B2%E5%88%B6" \o "十六进制)** | **注释** |
| 000000 - 00007F 128个代码 | 00000000 00000000 0zzzzzzz | 0zzzzzzz（00-7F） | ASCII字符范围，字节由零开始 |
| 七个z | 七个z |
| 000080 - 0007FF 1920个代码 | 00000000 00000yyy yyzzzzzz | 110yyyyy（C0-DF) 10zzzzzz(80-BF） | 第一个字节由110开始，接着的字节由10开始 |
| 三个y；二个y；六个z | 五个y；六个z |
| 000800 - 00D7FF 00E000 - 00FFFF 61440个代码 [[Note 1]](https://zh.wikipedia.org/wiki/UTF-8" \l "endnote_D800Note_1) | 00000000 xxxxyyyy yyzzzzzz | 1110xxxx(E0-EF) 10yyyyyy 10zzzzzz | 第一个字节由1110开始，接着的字节由10开始 |
| 四个x；四个y；二个y；六个z | 四个x；六个y；六个z |
| 010000 - 10FFFF 1048576个代码 | 000wwwxx xxxxyyyy yyzzzzzz | 11110www(F0-F7) 10xxxxxx 10yyyyyy 10zzzzzz | 将由11110开始，接着的字节由10开始 |
| 三个w；二个x；四个x；四个y；二个y；六个z | 三个w；六个x；六个y；六个z |

**[Note 1](https://zh.wikipedia.org/wiki/UTF-8" \l "ref_D800Note_1)** Unicode在范围D800-DFFF中不存在任何字符，[基本多文种平面](https://zh.wikipedia.org/wiki/%E5%9F%BA%E6%9C%AC%E5%A4%9A%E6%96%87%E7%A8%AE%E5%B9%B3%E9%9D%A2" \o "基本多文种平面)中约定了这个范围用于UTF-16扩展标识[辅助平面](https://zh.wikipedia.org/wiki/%E8%BE%85%E5%8A%A9%E5%B9%B3%E9%9D%A2" \o "辅助平面)（两个UTF-16表示一个[辅助平面](https://zh.wikipedia.org/wiki/%E8%BE%85%E5%8A%A9%E5%B9%B3%E9%9D%A2" \o "辅助平面)字符）。当然，任何编码都是可以被转换到这个范围，但在unicode中他们并不代表任何合法的值。

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，*110*yyyyy *10*zzzzzz.
* [十六进制](https://zh.wikipedia.org/wiki/%E5%8D%81%E5%85%AD%E8%BF%9B%E5%88%B6" \o "十六进制)的0x05D0换算成[二进制](https://zh.wikipedia.org/wiki/%E4%BA%8C%E8%BF%9B%E5%88%B6" \o "二进制)就是101-1101-0000.
* 这11位数按顺序放入"y"部分和"z"部分：110**10111** 10**010000**.
* 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](https://zh.wikipedia.org/wiki/%E9%99%84%E5%8A%A0%E7%AC%A6%E5%8F%B7" \o "附加符号)的[拉丁字母](https://zh.wikipedia.org/wiki/%E6%8B%89%E4%B8%81%E5%AD%97%E6%AF%8D" \o "拉丁字母)，[希腊字母](https://zh.wikipedia.org/wiki/%E5%B8%8C%E8%85%8A%E5%AD%97%E6%AF%8D" \o "希腊字母)，[西里尔字母](https://zh.wikipedia.org/wiki/%E8%A5%BF%E9%87%8C%E5%B0%94%E5%AD%97%E6%AF%8D" \o "西里尔字母)，[科普特语](https://zh.wikipedia.org/wiki/%E7%A7%91%E6%99%AE%E7%89%B9%E8%AF%AD" \o "科普特语)字母，[亚美尼亚语](https://zh.wikipedia.org/wiki/%E4%BA%9A%E7%BE%8E%E5%B0%BC%E4%BA%9A%E8%AF%AD" \o "亚美尼亚语)字母，[希伯来文](https://zh.wikipedia.org/wiki/%E5%B8%8C%E4%BC%AF%E6%9D%A5%E6%96%87" \o "希伯来文)字母和[阿拉伯字母](https://zh.wikipedia.org/wiki/%E9%98%BF%E6%8B%89%E4%BC%AF%E5%AD%97%E6%AF%8D" \o "阿拉伯字母)的字符。[基本多文种平面](https://zh.wikipedia.org/wiki/%E5%9F%BA%E6%9C%AC%E5%A4%9A%E6%96%87%E7%A8%AE%E5%B9%B3%E9%9D%A2" \o "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。

根据这种方式可以处理更大数量的字符。原来的规范允许长达6字节的序列，可以覆盖到31位（[通用字符集](https://zh.wikipedia.org/wiki/%E9%80%9A%E7%94%A8%E5%AD%97%E7%AC%A6%E9%9B%86" \o "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：

|  |  |  |
| --- | --- | --- |
| **编码（[二进制](https://zh.wikipedia.org/wiki/%E4%BA%8C%E8%BF%9B%E5%88%B6" \o "二进制)）** | **编码（[十六进制](https://zh.wikipedia.org/wiki/%E5%8D%81%E5%85%AD%E8%BF%9B%E5%88%B6" \o "十六进制)）** | **注释** |
| 1100000x | C0, C1 | 过长编码：双字节序列的头字节，但码点 <= 127 |
| 1111111x | FE, FF | 无法达到：7或8字节序列的头字节 |
| 111110xx 1111110x | F8, F9, FA, FB, FC, FD | 被RFC 3629规范：5或6字节序列的头字节 |
| 11110101 1111011x | F5, F6, F7 | 被RFC 3629规范：码点超过10FFFF的头字节 |

## 6.1UTF-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。

### 6.1.1字节数

在Unicode基本多文种平面定义的字符（无论是拉丁字母、汉字或其他文字或符号），一律使用2字节储存。而在辅助平面定义的字符，会以代理对（surrogate pair）的形式，以两个2字节的值来储存。

UTF-16比起UTF-8，好处在于大部分字符都以固定长度的字节 (2字节) 储存，但UTF-16却无法兼容于ASCII编码。

### 6.1.2描述方式

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。 从U+0000至U+D7FF以及从U+E000至U+FFFF的码位[编辑] 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[编辑] 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： UTF-16解码 lead \ trail DC00 DC01    …    DFFF D800 10000 10001 … 103FF D801 10400 10401 … 107FF   ⋮ ⋮ ⋮ ⋱ ⋮ DBFF 10FC00 10FC01 … 10FFFF 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[1]. 从U+D800到U+DFFF的码位[编辑] 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匹配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。

# 第 3章 参考文献

1. 维基百科 <https://en.wikipedia.org/wiki/Wiki>。
2. 百度百科 <https://baike.baidu.com/>。