`

常见编码集学习报告

系: 计算机科学学院

班级: 软工11502

姓名: 周锐

学号: 201503427

序号: 31

目录

[一.引言: 3](#_Toc493011245)

[二.GB2312： 3](#_Toc493011246)

[产生的原因和来历: 3](#_Toc493011247)

[解决的问题: 4](#_Toc493011248)

[字节以及兼容哪些字符集： 4](#_Toc493011249)

[二.Big5 5](#_Toc493011250)

[产生的原因和来历: 5](#_Toc493011251)

[解决的问题: 6](#_Toc493011252)

[字节以及兼容哪些字符集: 6](#_Toc493011253)

[三.Unicode 6](#_Toc493011254)

[产生的原因和来历: 6](#_Toc493011255)

[解决的问题: 7](#_Toc493011256)

[字节以及兼容哪些字符集: 8](#_Toc493011257)

[四. UTF 8/16 10](#_Toc493011258)

[产生的原因和来历: 10](#_Toc493011259)

[解决的问题: 11](#_Toc493011260)

[字节以及兼容哪些字符集: 11](#_Toc493011261)

# 一.引言:

本文我们主要来了解如下字符集:Unicode，GB2312，big5UTF 8/16这几个编码集的来历，产生原因，解决问题，字节数，描述兼容那些字符集以及相互的转换。

关键词为：Unicode GB2312 big5 UTF 8/16

# 二.GB2312：

## 产生的原因和来历:

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。

GB2312编码用两个字节(8位2进制)表示一个汉字，所以理论上最多可以表示256×256=65536个汉字。但这种编码方式也仅仅在中国行得通，如果您的网页使用的GB2312编码，那么很多外国人在浏览你的网页时就可能无法正常显示，因为其浏览器不支持GB2312编码。当然，中国人在浏览外国网页(比如日文)时，也会出现乱码或无法打开的情况，因为我们的浏览器没有安装日文的编码表。

　但由于中国的汉字太多了，我们很快就就发现有许多人的人名没有办法在这里打出来，特别是某些很会麻烦别人的国家领导人。于是我们不得不继续把 GB2312 没有用到的码位找出来老实不客气地用上。

　GBK:后来还是不够用，于是干脆不再要求低字节一定是127号之后的内码，只要第一个字节是大于127就固定表示这是一个汉字的开始，不管后面跟的是不是扩展字 符集里的内容。结果扩展之后的编码方案被称为 GBK 标准，GBK 包括了 GB2312 的所有内容，同时又增加了近20000个新的汉字（包括繁体字）和符号。

　GB18030:后来少数民族也要用电脑了，于是我们再扩展，又加了几千个新的少数民族的字，GBK 扩成了GB18030。从此之后，中华民族的文化就可以在计算机时代中传承了。

GB 18030，全称：国家标准GB 18030-2005《信息技术 中文编码字符集》，是中华人民共和国现时最新的内码字集，是GB 18030-2000《信息技术 信息交换用汉字编码字符集 基本集的扩充》的修订版。与GB 2312完全兼容，与GBK基本兼容，支持GB 13000及Unicode的全部统一汉字，共收录汉字70244个。GB 18030主要有以下特点：

. 与UTF-8相同，采用多字节编码，每个字可以由1个、2个或4个字节组成。

. 编码空间庞大，最多可定义161万个字符。

. 支持中国国内少数民族的文字，不需要动用造字区。

. 汉字收录范围包含繁体汉字以及日韩汉字

　中国的程序员们看到这一系列汉字编码的标准是好的，于是通称他们叫做 "DBCS"（Double Byte Charecter Set 双字节字符集）。在DBCS系列标准里，最大的特点是两字节长的汉字字符和一字节长的英文字符并存于同一套编码方案里，因此他们写的程序为了支持中文处 理，必须要注意字串里的每一个字节的值，如果这个值是大于127的，那么就认为一个双字节字符集里的字符出现了。那时候凡是受过加持，会编程的计算机僧侣 们都要每天念下面这个咒语数百遍："一个汉字算两个英文字符！一个汉字算两个英文字符……"

## 解决的问题:

信息交换用汉字编码字符集和汉字输入编码之间的关系是，根据不同的汉字输入方法，通过必要的设备向计算机输入汉字的编码，计算机接收之后，先转换成信息交换用汉字编码字符，这时计算机就可以识别并进行处理；汉字输出是先把机内码转成汉字编码，再发送到输出设备。国家标准《信息交换用汉字编码字符集·基本集》已于1981年5月发布实施。《基本集》规定了汉字信息交换用的基本图形字符及其二进制编码,收汉字6763个。它适用于一般汉字处理、汉字通信等系统之间的信息交换。随着我国汉字信息处理技术的发展,计算机的应用范围不断扩大,使用汉字字数较多的部门迫切需要在《基本集》的基础上继续制定信息交换用汉字编码字符集各辅助集的国家标准。[1]

## 字节以及兼容哪些字符集：

GB2312一个汉字两个字节。兼容GB18030，GBK，

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）。

# 二.Big5

## 产生的原因和来历:

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。

## 解决的问题:

Big5码的产生一是解决了厂商推出的中文应用软件无法推广的问题。二是另一方面，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。

## 字节以及兼容哪些字符集:

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。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。

# 三.Unicode

## 产生的原因和来历:

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。

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。

## 解决的问题:

我们知道计算机其实挺笨的,它只认识0101这样的字符串,当然了我们看这样的01串时肯定会比较头晕的,所以很多时候为了描述简单都用十进制,十六进制,八进制表示.实际上都是等价的,没啥太多不一样.其他啥文字图片之类的其他东东计算机不认识.那为了在计算机上表示这些信息就必须转换成一些数字.你肯定不能想怎么转换就怎么转,必须得有定些规则.于是刚开始的时候就有ASCII字符集(American Standard Code for Information Interchange， "美国信息交换标准码),它使用7 bits来表示一个字符,总共表示128个字符,我们一般都是用字节(byte,即8个01串)来作为基本单位.那么怎么当用一个字节来表示字符时第一个bit总是0,剩下的七个字节就来表示实际内容.后来IBM公司在此基础上进行了扩展，用8bit来表示一个字符，总共可以表示256个字符.也就是当第一个bit是0时仍表示之前那些常用的字符.当为1时就表示其他补充的字符.

英文字母再加一些其他标点字符之类的也不会超过256个.一个字节表示主足够了.但其他一些文字不止这么多 ,像汉字就上万个.于是又出现了其他各种字符集.这样不同的字符集交换数据时就有问题了.可能你用某个数字表示字符A,但另外的字符集又是用另外一个数字表示A.这样交互起来就麻烦了.于是就出现了Unicode和ISO这样的组织来统一制定一个标准,任何一个字符只对应一个确定的数字.ISO取的名字叫UCS(Universal Character Set),Unicode取的名字就叫unicode了.

总结起来为啥需要Unicodey就是为了适应全球化的发展,便于不同语言之间的兼容交互,而ASCII不再能胜任此任务了.

## 字节以及兼容哪些字符集:

当讲解和谈论统一码和UCS时，经常提到“兼容字符”这个字眼。兼容字符是指统一码联盟主张不要使用的图像式字元，正如统一码联盟所说： 若不是为了与其他标准间兼容和能够双向对应转换，就不会在统一码中编码的字符。 然而，其定义实在远比这话语有更深层和复杂的意味。每个字符都有一个标准分解（canonical decomposition）的属性，可是大部分字符这个属性的值是这个字符自身，但超过五千个字符的标准分解属性的值不是该字符自身。这个标准分解属性值能将兼容字符映射到一个或多个的其他的非兼容字符，并借此定义这五千多个字符为统一码中的兼容字符。指定某字符为兼容字符的理由各异，下面有更详细的说明。“分解”这个用语有时会令人感到困惑，因为有些字符“分解”后还是单个字符的形态，这时该字符的分解值就是另一个相等或近似相等的字符。兼容字符与其非空值的标准分解（语义上）是确切等价的。

兼容对应种类编辑

字形代换和组合

对于遵循统一码标准的文字处理和显示软件来说，有些兼容字符是不必要的。这些包含：

合字：在拉丁字母老旧的字符集中，合字如‘ffi’经常被作为一个字符来编码。Unicode的方法是把合字当作一种文本绘制的艺术，编码时用普通字母序列来代替。

预组合的罗马数字

譬如，罗马数十二（“Ⅻ”：U+216B）可以分解成一个罗马数十（“Ⅹ”：U+2169）和两个罗马数一 (‘Ⅰ’: U+2160)。Unicode认为字符序列'U+2169 U+2160 U+2160'是否被组合为单个字符'Ⅻ'，这是文本绘制软件要处理的问题。

预组合的分数：这些预组合字符具有<fraction>关键字。完全遵从Unicode规范的文本处理器应当绘制字符¼ (U+00BC)完全同于字符序列1⁄4 (数字1、分数斜线U+2044、数字4).

预组合的变音字母.

如 Å（U+00C5），统一码倾向视为两个分开的字符，一个拉丁字母 A （“Latin letter A”）结合一个“Combining Ring Above”（U+030A）。

依上下文环境而变的字形：主要出现在阿拉伯文中。使用具有字形替换能力的字体，如OpenType、TrueTypeGX，遵从Unicode规范的软件能把出现在不同上下文环境（词的开头、词的中部、词的结尾、单独使用）的同一字符替换为适当的字形来绘制。这种字形替换能力对于垂直文本绘制的东亚语言也是必需的。

总结而言，上述的预组合的字符的使用是不必要的，应该用普通字符的序列来表示这些预组合的字符；由遵从Unicode规范的字体或者文本绘制软件来决定选用哪个字形。

Unicode字符集（即UCS）, Unicode字符的属性、Unicode算法为文本处理软件实现提供了所需的一切来适当地绘制处于分解等价状态的字符。因此那些分解兼容字符变成了冗余与不必要。字符集中分解兼容字符的存在导致了文本处理时额外的代价用于正确地比较、排序（参见统一码等价性）。此外，分解兼容字符并没有提供补充的或不同的语义。分解兼容字符也没有提供绘制时视觉上的不同，如果文本布局与字体遵循Unicode规范。分解兼容字符也不是与其他字符集往返转换所需要的，因为可以把一个字符集的分解字符序的列来映射到另一字符集的预组合的字符。 上下文中字形选择，如阿拉伯字母可以根据它在单词内的位置而映射到传统字符集的具有特定字形的某个字符上。

为了处理兼容字符，文本软件必须遵从几个Unicode协议。软件必须能够：

从字母字符与一个或多个单独的附加符号组合出变音字母字符；

根据用户的判断，替换出合字或上下文相关的字形变种；

根据用户的判断，CJVK文本垂直布局时，把字形替换为小的、垂直的、窄的、宽方块等字形变种；

使用分数斜线字符 (⁄ U+2044)与其它任意字符来组合出分数的字形。

能把组合字符（Combining character）正确绘制到其前的普通字符上。例如把表示现金的长斜线组合字符‘Combining Long Solidus Overlay’ ( ̸ U+0338)加到其它字符上，如∄

Unicode的5,402个兼容字符中，上述这些不应该使用在文本中的字符共计3,779个。这包括所有具有关键字 <initial>、<medial>、<final>、<isolated>、<wide>, <narrow>、<small>、<square>、<vertical>、<fraction>的兼容字符。还包括大多数具有<compat>关键字的兼容字符（例外情况是有<compat>关键字的被封闭的字母数字、被封闭的象形文字、以及下文所提及)

格式化文本

用于格式化文本的兼容字符，不是Unicode与UCS所考虑的目标。格式化文本所用的兼容字符，可能与普通字符在语义上有出入。例如，作为上角标的数字4，与普通字符数字4，可能语义上不完全等价。

格式化文本兼容字符包括：

数学字母数字符号：拉丁字母、希腊字母、阿拉伯数字的字符，有14种不同的字体，用于数学文本。

被封闭的字母、数字、象形文字。如：⑪ ⑫ ⒄ ⒅ ⒌ ⒍ ⒲ ⒳ Ⓐ Ⓑ

不同宽度的空格与不换行空格。这些字符是核心的空格符(U+0020)与不换行空格符(U+00A0)的变种。 如不换行窄空格（Narrow No-Break Space, U+202F）、词连接（Word joiner, U+2060）。

上角标、下角标字符。这里包括国际音标所用的那些看似是上下角标的字符。这里所指是拉丁字母、希腊字母、数字作为上下角标的字符。

CJK定义的全宽拉丁字母、阿拉伯数字，以及半宽日文、韩文字母。在Unicode的Half-width and Fullwidth Forms块中，U+FF00至U+FFEF。

格式化文本兼容字符共有1,451个，包括所有具有关键字<circle>与<font>（除了下文列出3个语义不同的例外），11个空格符的变种具有关键字<compat>，具有关键字<superscript>或<subscript>的定义在基本多语言平面的“上标及下标”块中的字符。 定义在U+2100至U+214F定义的Letterlike Symbols块中的字符，都是具有字体格式的兼容字符。

语义上相异字符

Unicode对于用于科学或数学的希伯来字母、希腊字母符号，作为兼容字符定义。如：

希伯来字母符号：alef (ℵ U+2135), bet (ℶ U+2136), gimel (ℷ U+2137), dalet (ℸ U+2138)

希腊字母符号：beta (ϐ U+03D0), theta (ϑ U+03D1), phi (ϕ U+03D5), pi (ϖ U+03D6), kappa (ϰ U+03F0), rho (ϱ U+03F1), 大写theta (ϴ U+03F4)

6个作为度量单位的兼容字符，Unicode建议使用其等价分解的字符序列：

Angstrom (“Å”（angstrom sign，U+212B），应该用“Å”（latin capital letter a with ring above，U+00C5） )

Ohm (“Ω”（ohm sign，U+2126）, 应该用“Ω”（greek capital letter omega，U+03A9）)

Kelvin (“K”（，U+212A）, 应该用“K”（，U+004B）)

Fahrenheit (“℉”（，U+2109）， 应该用“°”（，U+00B0）与“F”（，U+0046）)

Celsius (“℃”（，U+2103），应该用“°”（，U+00B0）与“C”（，U+0043）)

Micro Sign (“µ”（，U+00B5），应该用“μ”（，U+03BC）)

Unicode规定了22个类似字母的兼容字符。

基于希腊字母的: lunate epsilon (ϵ U+03F5), lunate sigma (ϲ U+03F2), capital lunate sigma (Ϲ U+03F9), upsilon with hook (ϒ U+03D2)

数学常量： Euler constant (ℇ U+2107), Planck constant (ℎ U+210E), reduced Planck constant (ℏ U+210F),

货币符号: rupee sign (₨ U+20A8), rial sign (﷼ U+FDFC)

标点符号: one dot leader (U+2024), no-break space (U+00A0), non-breaking hyphen (U+2011), Tibetan mark delimiter tsheg bstar (U+0F0C)

其它类字母符号: information source (ℹ U+2139), account of (℀ U+2100), addressed to the subject (℁ U+2101), care of (℅ U+2105), cada una (℆ U+2106), numero (№ U+2116), telephone sign (℡ U+2121), facsimile sign (℻ U+213B), trademark (™ U+2122), service mark (℠ U+2120)

一些语言中，语义与字形的位置有关的字符，共计130个。

罗马数字：

大写罗马数字7个：1 (Ⅰ U+2160), 5(Ⅴ U+2164), 10(Ⅹ U+2169), 50(Ⅼ U+216C), 100 (Ⅽ U+216D), 500 (Ⅾ U+216E), 1000 (Ⅿ U+216F)

小写罗马数字7个：1 (Ⅰ U+2170), 5(Ⅴ U+2174), 10(Ⅹ U+2179), 50(ⅼ U+217C), 100(ⅽ U+217D), 500 (ⅾ U+217E) ，1000 (ⅿ U+217F)

预组合罗马数字18个：2–4, 6–9, 11–12的大小写版本。

“ↀ”（one thousand c d，U+2180）

# 四. UTF 8/16

## 产生的原因和来历:

UTF 8：1992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。

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

## 解决的问题:

1、UTF-16：是任何字符对应的数字都用两个字节来保存，但如果都是英文字母(一个字节能表示一个字符)这样做有点浪费。

2、UTF-8：是任何字符对应的数字保存时所占的空间是可变的，可能用一个、两个或三个字节表示一个字符。

如果全部英文或英文与其他文字混合(英文占绝大部分)，用UTF-8就比UTF-16节省了很多空间。而如果全部是中文这样类似的字符或者混合字符(中文占绝大多数),UTF-16就可以节省很多空间，另外还有个容错问题(比如：UTF-8需要判断每个字节中的开头标志信息,所以如果一当某个字节在传送过程中出错了,就会导致后面的字节也会解析出错；而UTF-16不会判断开头标志,即使错也只会错一个字符,所以容错能力强)。

## 字节以及兼容哪些字符集:

对代码点进行编码后的二进制的比特序列，是由一个或者多个编码单元按照一定的序列组成的。一个编码单元由若干个比特构成，例如7比特、8比特等(最常用的是8/16/32比特，便于存储和传输)。

比如说，UTF-8编码，采用1~4个8比特的编码单元；

UTF-16编码，采用1~2个16比特的编码单元；

前两个属于变长编码，后一个属于等长编码。

对于ASCII字符集，编码不变，从0x00~0x7F

UTF-16比较好理解,就是任何字符对应的数字都用两个字节来保存.我们通常对Unicode的误解就是把Unicode与UTF-16等同了.但是很显然如果都是英文字母这做有点浪费.明明用一个字节能表示一个字符为啥整两个啊.

于是又有个UTF-8,这里的8非常容易误导人,8不是指一个字节,难道一个字节表示一个字符?实际上不是.当用UTF-8时表示一个字符是可变的,有可能是用一个字节表示一个字符,也可能是两个,三个.当然最多不能超过3个字节了.反正是根据字符对应的数字大小来确定.

于是UTF-8和UTF-16的优劣很容易就看出来了.如果全部英文或英文与其他文字混合,但英文占绝大部分,用UTF-8就比UTF-16节省了很多空间.而如果全部是中文这样类似的字符或者混合字符中中文占绝大多数.UTF-16就占优势了,可以节省很多空间.另外还有个容错问题,等会再讲

看的有点晕了吧,举个例子.假如中文字"汉"对应的unicode是6C49(这是用十六进制表示,用十进制表示是27721为啥不用十进制表示呢?很明显用十六进制表示要短点.其实都是等价的没啥不一样.就跟你说60分钟和1小时一样.).你可能会问当用程序打开一个文件时我们怎么知道那是用的UTF-8还是UTF-16啊.自然会有点啥标志,在文件的开头几个字节就是标志.

EF BB BF 表示UTF-8,FE FF 表示UTF-16.

用UTF-16表示"汉"

假如用UTF-16表示的话就是01101100 01001001(共16 bit,两个字节).程序解析的时候知道是UTF-16就把两个字节当成一个单元来解析.这个很简单.

用UTF-8表示"汉"

用UTF-8就有复杂点.因为此时程序是把一个字节一个字节的来读取,然后再根据字节中开头的bit标志来识别是该把1个还是两个或三个字节做为一个单元来处理.

0xxxxxxx,如果是这样的01串,也就是以0开头后面是啥就不用管了XX代表任意bit.就表示把一个字节做为一个单元.就跟ASCII完全一样.

110xxxxx 10xxxxxx.如果是这样的格式,则把两个字节当一个单元

1110xxxx 10xxxxxx 10xxxxxx 如果是这种格式则是三个字节当一个单元.

这是约定的规则.你用UTF-8来表示时必须遵守这样的规则.我们知道UTF-16不需要用啥字符来做标志,所以两字节也就是2的16次能表示65536个字符.

而UTF-8由于里面有额外的标志信息,所有一个字节只能表示2的7次方128个字符,两个字节只能表示2的11次方2048个字符.而三个字节能表示2的16次方,65536个字符.

由于"汉"的编码27721大于2048了所有两个字节还不够,只能用三个字节来表示.

所有要用1110xxxx 10xxxxxx 10xxxxxx这种格式.把27721对应的二进制从左到右填充XXX符号(实际上不一定从左到右,也可以从右到左,这是涉及到另外一个问题.等会说.

刚说到填充方式可以不一样,于是就出现了Big-Endian,Little-Endian的术语.Big-Endian就是从左到右,Little-Endian是从右到左.

由上面我们可以看出UTF-8需要判断每个字节中的开头标志信息,所以如果一当某个字节在传送过程中出错了,就会导致后面的字节也会解析出错.而UTF-16不会判断开头标志,即使错也只会错一个字符,所以容错能力强.