目录

[摘要 2](#_Toc492845604)

[关键词： 2](#_Toc492845605)

[1.1来历 2](#_Toc492845606)

[1.2产生原因 2](#_Toc492845607)

[1.3解决的问题 2](#_Toc492845608)

[1.4字节数 3](#_Toc492845609)

[1.5描述 3](#_Toc492845610)

[1.6兼容 3](#_Toc492845611)

[二、big5 3](#_Toc492845612)

[2.1来历 3](#_Toc492845613)

[2.2产生的原因 4](#_Toc492845614)

[2.3解决的问题 4](#_Toc492845615)

[2.4字节数 4](#_Toc492845616)

[2.5描述 4](#_Toc492845617)

[2.6兼容 4](#_Toc492845618)

[三、UTF 5](#_Toc492845619)

[3.1来历 5](#_Toc492845620)

[3.3解决的问题 6](#_Toc492845621)

[3.4字节数 6](#_Toc492845622)

[3.5描述 6](#_Toc492845623)

[四、Unicode 8](#_Toc492845624)

[4.1来历 8](#_Toc492845625)

[4.2产生的原因 9](#_Toc492845626)

[4.3解决的问题 9](#_Toc492845627)

[4.4字节数 10](#_Toc492845628)

[4.5描述 10](#_Toc492845629)

[五、联系 11](#_Toc492845630)

[六、参考文献 12](#_Toc492845631)

各字符集的相关内容

摘要

本文对计算机中字符与编码的各种方法，编码方案的发展，相关概念进行了分析总结。

关键词：汉字字符编码, UNICODE，字符编码转换

GB 2312

# 1.1来历

GB2312或GB 2312-1980，全称《信息交换用汉字编码字符集 基本集》，由国家标准总局于1980年3月9号发布，1981年5月1日实施，通行于大陆。新加坡等地也使用此编码。它是一个简化字的编码规范，也包括其他的符号、字母、日文假名等，共7445个图形字符，其中汉字占6763个。中国大陆几乎所有的中文系统和国际化的软件都支持GB2312。

**1.2产生原因**

计算机发明之处及后面很长一段时间，只用应用于美国及西方一些发达国家，ASCII能够很好满足用户的需求。当中国人们得到计算机时，已经没有可以利用的字节状态来表示汉字，况且有6000多个常用汉字需要保存，于是想到把那些ASCII码中127号之后的奇异符号们直接取消掉, 规定：一个小于127的字符的意义与原来相同，但两个大于127的字符连在一起时，就表示一个汉字，前面的一个字节（称之为高字节）从0xA1用到0xF7，后面一个字节（低字节）从0xA1到0xFE，这样我们就可以组合出大约7000多个简体汉字了。在这些编码里，我们还把数学符号、罗马希腊的字母、日文的假名们都编进去了，连在 ASCII 里本来就有的数字、标点、字母都统统重新编了两个字节长的编码，这就是常说的"全角"字符，而原来在127号以下的那些就叫"半角"字符了。这种汉字方案叫做 "GB2312"。GB2312 是对 ASCII 的中文扩展。

**1.3解决的问题**

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。

**1.4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）。

**1.5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。

**1.6兼容**

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。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，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。

**二、big5**

**2.1来历**

“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。

**2.2产生的原因**

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。

**2.3解决的问题**

为了解决在港澳台地区使用繁体字的统一问题。

**2.4字节数**

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。

**2.5描述**

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。

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。

# 2.6兼容

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。

因为Big5也占用了ASCII的编码空间（低字节所使用的0x40-0x7E），所以Big5编码在一些环境下存在和GBK编码相同的问题，即低字节范围为0x40-0x7E的字符有可能会被误处理，尤其是低字节是0x5C（"/"）和0x7C（"|"）的字符。

**三、UTF**

**3.1来历**

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。

**3.2产生的原因**

UTF-16编码方式源于UCS-2(Universal Character Set coded in 2 octets、2-byte Universal Character Set)。而UCS-2，是早期遗留下来的历史产物。

UTF-16编码方式源于UCS-2(Universal Character Set coded in 2 octets、2-byte Universal Character Set)。而UCS-2，是早期遗留下来的历史产物。

UCS-2将字符编号直接映射为字符编码(CEF，而非CES，详见前文中对现代字符编码模型的解释)，亦即字符编号就是字符编码，中间没有经过特别的编码算法转换。因此，从现代字符编码模型的角度来看的话，此时并没有将编号字符集CCS与字符编码方式CEF作严格区分，既可以将UCS-2看作是编号字符集CCS中的字符编号，也可以看作是字符编码方式CEF中的字符编码。

**3.3解决的问题**

互联网的普及，强烈要求出现一种统一的编码方式。

**3.4字节数**

UTF-8最大的一个特点，就是它是一种变长的编码方式。它可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度。

UTF-8的编码规则很简单，只有二条：

1）对于单字节的符号，字节的第一位设为0，后面7位为这个符号的unicode码。因此对于英语字母，UTF-8编码和ASCII码是相同的。

2）对于n字节的符号（n>1），第一个字节的前n位都设为1，第n+1位设为0，后面字节的前两位一律设为10。剩下的没有提及的二进制位，全部 为这个符号的unicode码。

UTF-16的常用字符都使用两个个字节表示。

**3.5描述**

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 / STD 63（2003），这份文件制定了UTF-8是标准的互联网协议元素

•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（1996）

•RFC 2279（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字符的比特被分区为数个部分，并分配到UTF-8的字节串中较低的比特的位置。在U+0080的以下字符都使用内含其字符的单字节编码。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。

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

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：

UTF-16解码

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匹配Unicode标准的字符映射正确的辨识、转换成合规的码元。按照Unicode标准，这种码元序列本来应算作编码错误。

**四、Unicode**

**4.1来历**

世界上存在着多种编码方式，在ANSi编码下，同一个编码值，在不同的编码体系里代表着不同的字，。在简体中文系统下，ANSI 编码代表 GB2312 编码，在日文操作系统下，ANSI 编码代表 JIS 编码，可能最终显示的是中文，也可能显示的是日文。在ANSI编码体系下，要想打开一个文本文件，不但要知道它的编码方式，还要安装有对应编码表，否则就可能无法读取或出现乱码。为什么电子邮件和网页都经常会出现乱码，就是因为信息的提供者可能是日文的ANSI编码体系和信息的读取者可能是中文的编码体系，他们对同一个二进制编码值进行显示，采用了不同的编码，导致乱码。这个问题促使了unicode码的诞生。

如果有一种编码，将世界上所有的符号都纳入其中，无论是英文、日文、还是中文等，大家都使用这个编码表，就不会出现编码不匹配现象。每个符号对应一个唯一的编码，乱码问题就不存在了。这就是Unicode编码。

Unicode当然是一个很大的集合，现在的规模可以容纳100多万个符号。每个符号的编码都不一样，比如，U+0639表示阿拉伯字母Ain，U+0041表示英语的大写字母A，“汉”这个字的Unicode编码是U+6C49。

Unicode固然统一了编码方式，但是它的效率不高，比如UCS-4(Unicode的标准之一)规定用4个字节存储一个符号，那么每个英文字母前都必然有三个字节是0，这对存储和传输来说都很耗资源。

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。

位于美国加州的Unicode组织允许任何愿意支付会费的公司和个人加入，其成员包含了主要的电脑软硬件厂商，例如奥多比系统、苹果公司、惠普、IBM、微软、施乐等。

20世纪80年代末，组成Unicode组织的商业机构，和国际合作的国际标准化组织因为电脑普及和信息国际化的前提下，分别各自成立了Unicode组织[3]和ISO-10646工作小组。他们不久便发现对方的存在，大家为着相同的目的而工作。1991年，Unicode Consortium与ISO/IEC JTC1/SC2同意保持Unicode码表与ISO 10646标准保持兼容并密切协调各自标准进一步的扩展。虽然实际上两者的字集编码相同，但实质上两者确实为两个不同的标准。Unicode 1.1对应于ISO 10646-1:1993，Unicode 3.0对应于ISO 10646-1:2000，Unicode 3.2对应于ISO 10646-2:2001，Unicode 4.0对应于ISO 10646:2003，Unicode 5.0对应于ISO 10646:2003及附录1–3。

Unicode自版本2.0开始保持了向后兼容，即新的版本仅仅增加字符，原有字符不会被删除或更名。

统一码联盟在1991年首次发布了The Unicode Standard。Unicode的开发结合了国际标准化组织所制定的ISO/IEC 10646，即通用字符集。Unicode与ISO/IEC 10646在编码的运作原理相同，但The Unicode Standard包含了更详尽的实现信息、涵盖了更细节的主题，诸如比特编码（bitwise encoding）、校对以及呈现等。The Unicode Standard也枚举了诸多的字符特性，包含了那些必须支持两种阅读方向的文字（由左至右或由右至左的文字阅读方向，例如阿拉伯文是由右至左）。Unicode与ISO/IEC 10646这两个标准在术语上的使用有些微的不同。

在2005年，Unicode的第十万个字符被引入成为标准之一，该字符被用于马拉雅拉姆语。

**4.2产生的原因**

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）。

**4.3解决的问题**

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。

**4.4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。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。

**4.5描述**

从 UNICODE 开始，无论是半角的英文字母，还是全角的汉字，它们都是统一的"一个字符"！同时，也都是统一的"两个字节"，请注意"字符"和"字节"两个术语的不 同，“字节”是一个8位的物理存贮单元，而“字符”则是一个文化相关的符号。在UNICODE 中，一个字符就是两个字节。一个汉字算两个英文字符的时代已经快过去了。

从前多种字符集存在时，那些做多语言软件的公司遇上过很大麻烦，他们为了在不同的国家销售同一套软件，就不得不在区域化软件时也加持那个双字节字符集咒 语，不仅要处处小心不要搞错，还要把软件中的文字在不同的字符集中转来转去。UNICODE 对于他们来说是一个很好的一揽子解决方案，于是从 Windows NT 开始，MS 趁机把它们的操作系统改了一遍，把所有的核心代码都改成了用 UNICODE 方式工作的版本，从这时开始，WINDOWS 系统终于无需要加装各种本土语言系统，就可以显示全世界上所有文化的字符了。

但是，UNICODE 在制订时没有考虑与任何一种现有的编码方案保持兼容，这使得 GBK 与UNICODE 在汉字的内码编排上完全是不一样的，没有一种简单的算术方法可以把文本内容从UNICODE编码和另一种编码进行转换，这种转换必须通过查表来进行。

**4.6兼容**

最初，由多语言软件制造商组成了统一码联盟(The Unicode Consortium)，然后于1991年发布了The Unicode Standard(统一码标准)，定义了一个全球统一的通用字符集，习惯简称为Unicode字符集（Unicode常被称为统一码、万国码、单一码，严格来说，这种称呼不够严谨或不够准确，因为Unicode字符集只是一个编号字符集，尚未经过字符编码方式CEF和字符编码模式CES进行编码）。

接着，ISO及IEC也于1993年联合发布了称之为Universal Multiple-Octet Coded Character Set(通用多八位组编号字符集，习惯翻译为“通用多八位编码字符集”)、简称为UCS(Universal Character Set通用字符集)、标准号为ISO/IEC 10646-1的全球统一的通用字符集。

后来，统一码联盟与ISO/IEC双方都意识到世界上没有必要存在两套全球统一的通用字符集，于是进行整合，并为创立一个单一的全球统一的通用字符集而协同工作。到Unicode 2.0时，Unicode字符集和UCS字符集(ISO/IEC 10646-1)基本保持了一致。

虽然现在两个项目仍都存在，并独立地公布各自的标准，但统一码联盟和ISO/IEC都同意保持两者的通用字符集相互兼容，并共同调整未来的任何扩展。

目前，Unicode的知名度要比UCS知名度大得多，已成了全球统一的通用字符集或编码方案的代名词。并且在实践中，Unicode也要比UCS应用得更为广泛得多。

**五、联系**

GB2312是在ASCLL上发展过来的，是为了储存汉字和一些常见的字符。

Big5是为了制定一个统一的储存繁体字标准而制定的。

GB2312和BIG5都是为了储存对应的字型而建立的，两者相辅相成。

UTF-8/16是为了建立统一的互联网编码标准而建立的。

Unicode是为了统一各国国家文化而建立的一个字符集。

**六、参考文献**

1. 简书. 字符，字符集，字符编码。<http://www.jianshu.com/p/bd7a6c508c33>
2. 维基百科. 字符编码. <http://zh.wikipedia.org/wiki/%E5%AD%97%E7%AC%A6%E7%BC%96%E7%A0%81>
3. 维基百科. ASCII. <http://zh.wikipedia.org/wiki/ASCII>
4. 维基百科. GB2312. <http://zh.wikipedia.org/wiki/GB_2312>,
5. 维基百科.大五码. <https://zh.wikipedia.org/wiki/%E5%A4%A7%E4%BA%94%E7%A2%BC>
6. 维基百科. Unicode. <http://zh.wikipedia.org/wiki/Unicode>
7. 维基百科.UTF-8.<https://zh.wikipedia.org/wiki/UTF-8>
8. 维基百科. UTF-16.<https://zh.wikipedia.org/wiki/UTF-16>