XML作业

UTF-8/16，GB 2312，Unicode字符集，big5简介

学生姓名：朱厚强

学 号：201503650

专业班级：软件工程1502

2017年9月9号

目 录

一：UTF-8................................................4

来历...............................................................4

产生的原因.........................................................4

解决的问题.........................................................4

字节数.............................................................5

兼容的字符.........................................................5

描述...............................................................5

二：UTF-16...............................................7

来历...............................................................7

产生的原因.........................................................7

解决的问题.........................................................7

字节数.............................................................8

兼容的字符.........................................................8

描述...............................................................8

三：Unicode字符集......................................10

来历...............................................................10

产生的原因.........................................................10

解决的问题.........................................................11

字节数.............................................................11

兼容的字符.........................................................11

描述...............................................................11

四：big5.................................................13

来历...............................................................13

产生的原因.........................................................13

解决的问题.........................................................13

字节数.............................................................13

兼容的字符.........................................................14

描述...............................................................15

五：GB 2312..............................................15

来历...............................................................15

产生的原因.........................................................15

解决的问题.........................................................15

字节数.............................................................15

兼容的字符.........................................................15

描述...............................................................15

六：参考文献..............................................16

UTF-8

来历：

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502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。

产生的原因：

用[ASCII](https://baike.baidu.com/item/ASCII" \t "https://baike.baidu.com/item/UTF-8/_blank)表示的[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "https://baike.baidu.com/item/UTF-8/_blank)使用UNICODE并不高效，因为UNICODE比ASCII占用大一倍的空间，而对ASCII来说高[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82" \t "https://baike.baidu.com/item/UTF-8/_blank)的0对他毫无用处。为了解决这个问题，就出现了一些中间格式的[字符集](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6%E9%9B%86" \t "https://baike.baidu.com/item/UTF-8/_blank)，他们被称为通用转换格式，即[UTF](https://baike.baidu.com/item/UTF" \t "https://baike.baidu.com/item/UTF-8/_blank)（Unicode Transformation Format）。常见的UTF格式有：UTF-7, UTF-7.5, UTF-8,[UTF-16](https://baike.baidu.com/item/UTF-16" \t "https://baike.baidu.com/item/UTF-8/_blank), 以及 [UTF-32](https://baike.baidu.com/item/UTF-32" \t "https://baike.baidu.com/item/UTF-8/_blank)。

解决的问题：

TF-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位元组序列，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。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

字节数：

UTF-8最小码位0000最大码位10FFFF每字节所占位数8 bitsByte orderN/A每个字符最小字节数1每个字符最大字节数4。

兼容的字符：

UCS字符U+0000到U+007F（ASCII）被编码为字节0x00到0x7F（ASCII兼容），这也意味著只包含7位ASCII字符的文件在ASCII和UTF-8两种编码方式下是一样的。

描述：

UTF-8（8-bit Unicode Transformation Format）是一种针对Unicode的可变长度字元编码，也是一种前缀码。它可以用来表示Unicode标准中的任何字元，且其编码中的第一个位元组仍与ASCII相容，这使得原来处理ASCII字元的软体无须或只须做少部份修改，即可继续使用。因此，它逐渐成为电子邮件、网页及其他储存或传送文字的应用中，优先采用的编码。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）：

128个US-ASCII字符只需一个位元组编码（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位元组编码（Unicode范围由U+0080至U+07FF）。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）。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）。网际网路工程工作小组（IETF）要求所有网际网路协议都必须支持UTF-8编码。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

UTF-16

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。

产生的原因：  
 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。

解决的问题：

创建了一个良好的字节串编码系统可以供多字节字符集使用。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.

字节数：

在Unicode[基本多文种平面](https://baike.baidu.com/item/%E5%9F%BA%E6%9C%AC%E5%A4%9A%E6%96%87%E7%A7%8D%E5%B9%B3%E9%9D%A2" \t "https://baike.baidu.com/item/UTF-16/_blank)定义的[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "https://baike.baidu.com/item/UTF-16/_blank)（无论是[拉丁字母](https://baike.baidu.com/item/%E6%8B%89%E4%B8%81%E5%AD%97%E6%AF%8D" \t "https://baike.baidu.com/item/UTF-16/_blank)、汉字或其他文字或符号），一律使用2[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82" \t "https://baike.baidu.com/item/UTF-16/_blank)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。

兼容的字符：

UTF-16却无法兼容于[ASCII](https://baike.baidu.com/item/ASCII" \t "https://baike.baidu.com/item/UTF-16/_blank)编码。

描述：

TF-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。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：More information: DC00, DC01 …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。

Unicode字符集

来历：

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。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。

产生的原因：

Unicode是为了解决传统的字元编码方案的局限而产生的。

解决的问题：

解决传统的字元编码方案的局限。

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。

兼容的字符：

兼容ASCII

描述：

Unicode伴随著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涵盖的资料除了视觉上的字形、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体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程式语言以及现代的作业系统，都采用Unicode编码。

Big5

来历：

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。

产生的原因：

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。

解决的问题：

使个人电脑有了共通的内码，促进了厂商推出的中文应用软体的推广

字节数：

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，亦是先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。

兼容的字符：

IBM 5550、王安码等内码。1983年“通用汉字标准交换码”试用版发行，包括13,053个字与441个符号，分为二个字面，先笔画数，后部首序排列。12月：Big-5大五码，包括13,053个字与441个符号，字集与字序与交换码试用版完全相同，仅字码定义不同。

描述：

Big5，又称为大五码或五大码，是使用繁体中文（正体中文）社群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，衍生成多种不同版本。2003年，Big5被收录到CNS11643中文标准交换码的附录当中，取得了较正式的地位。这个最新版本被称为Big5-2003。

GB 2312

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。

产生的原因：

解决的问题：

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，GB 2312收录了包括[拉丁字母](https://baike.baidu.com/item/%E6%8B%89%E4%B8%81%E5%AD%97%E6%AF%8D" \t "https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E4%BA%A4%E6%8D%A2%E7%94%A8%E6%B1%89%E5%AD%97%E7%BC%96%E7%A0%81%E5%AD%97%E7%AC%A6%E9%9B%86/_blank)、[希腊字母](https://baike.baidu.com/item/%E5%B8%8C%E8%85%8A%E5%AD%97%E6%AF%8D" \t "https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E4%BA%A4%E6%8D%A2%E7%94%A8%E6%B1%89%E5%AD%97%E7%BC%96%E7%A0%81%E5%AD%97%E7%AC%A6%E9%9B%86/_blank)、日文平假名及片假名字母、俄语西里尔字母在内的682个[全角字符](https://baike.baidu.com/item/%E5%85%A8%E8%A7%92%E5%AD%97%E7%AC%A6" \t "https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E4%BA%A4%E6%8D%A2%E7%94%A8%E6%B1%89%E5%AD%97%E7%BC%96%E7%A0%81%E5%AD%97%E7%AC%A6%E9%9B%86/_blank)。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。

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）。

兼容的字符：

GBK子集与GBK/GB 18030兼容，GB2312.TXT则不兼容。

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。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）。

参考文献

维基百科

百度百科