字符编码

姓　　名：_____

学　　号：_____

班　　级：_____

指导老师：_____

目录

前言 1
为什么需要编码？ 1
一、Unicode 字符 1
1、起源与发展 1
2、有重要作用 2
3、编码方式 2
4、优点与缺点 2
二、big5 2
1、历史及名称 2
2、字节结构 3
3、优点与缺点 3
三、GB2312 4
1、来历与原因 4
2、字节结构 4
3、优点与缺点 5
四、UTF-8/16 5
1、历史 5
2、含义 6
3、编码方式 6
UTF-8 6
UTF-16 6
4、特性 7
5、优点与缺点 7
六、总结 8

前言

在解决昨天的问题时，又引出了很多新的问题，如为什么要进行编码，这些编码的关系如何，如 ASCII，IOS-8859-1，GB2312，GBK，Unicode 之间的关系，笔者想要彻底理解字符编码背后的故事，遂进行了探索，具体如下。

为什么需要编码？

我们知道，所有的信息最终都表示为一个二进制的字符串，每一个二进制位（bit）有 0 和 1 两种状态。当我们需要把字符'A'存入计算机时，应该对应哪种状态呢，存储时，我们可以将字符'A'用 01000010（这个随便编的）二进制字符串表示，存入计算机；读取时，再将 01000010 还原成字符'A'。那么问题来了，存储时，字符'A'应该对应哪一串二进制数呢，是 01000010？或者是 10000000 11110101？说白了，就是需要一个规则。这个规则可以将字符映射到唯一一种状态(二进制字符串)，这就是编码。而最早出现的编码规则就是 ASCII 编码，在 ASCII 编码规则中，字符'A'既不对应 01000010，也不对应 1000 0000 11110101，而是对应 01000001。

一、Unicode 字符

1、起源与发展

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。

Unicode 编码包含了不同写法的字，如“α / a”、“強 / 强”、“戶 / 户 / 戶”。然而在汉字方面引起了一字多形的认定争议。

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。换句话说，统一码以一种抽象的方式来处理字符，并将视觉上的演绎工作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

2、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。在基本多文种平面里的所有字符，要用四个数字；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。

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。

统一码这种为数万汉字逐一编码的方式很浪费资源，且要把汉字增加到标准中也并不容易，因此去研究以汉字部件产生汉字的方法，期望取代为汉字逐一编码的方法。Unicode 委员会在关于中文和日语的常用问题列表[28]里回答了此问题。主要问题是汉字中各个组件的相对大小不是固定的。比如“员”字，由“口”和“贝”组成，而“呗”也是由“口”和“贝”组成，但其相对位置和大小并不一致。还有一些其他原因，比如字符比较和排序时需要先对编码流进行分析后才能得到各个字符，增加处理程序复杂性等。

4、优点与缺点

优点：
适用国际化环境，可以做字符的内部表示和存储形式，来实现软件的国际化、本地化；

缺点：
目前支持较少，与其他中文字符集不兼容。

二、big5

1、历史及名称

“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项目‘，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。

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；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。

除了台湾外，其他使用繁体汉字的地区，如香港、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。

2、字节结构

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。

3、优点与缺点

优点：

适用于繁体中文环境，属于台湾官方标准，为繁体 Windows 所使用，在台湾和香港得到广泛支持，而且，由于台湾软件业发展较早，国外软件支持 big5 的比支持 gb2312/gbk 的要多；

缺点：

不兼容简体中文环境，和 gb2312 之间需要转换。

三、GB2312

1、来历与原因

当中国开始使用计算机时，没有可以利用的字节状态来表示汉字，有 6000 多个常用汉字需要保存。中国人就不客气地把那些 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（他称之为高字节）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。于是就把这种汉字方案叫做“GB2312”。GB2312 是对 ASCII 的中文扩展。

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。

2、字节结构

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。 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）。

GB2312 其对所收录字符进行了"分区"处理，共 94 个区，区从 1（十进制）开始，一直到 94（十进制），每区含有 94 个位，位从 1（十进制）开始，一直到 94（十进制），共 8836（94 * 94）个码位，这种表示方式也称为区位码，GB2312 是双字节编码，其中高字节表示区，低字节表示位。

那么根据区位码如何算出 GBK2312 编码呢？区位码的表示范围为 0101 - 9494（包含了空的区位码）。点击这里 ， 查看中 GB2312 编码区位码。之后只需要按照如下规则进行转化即可。

1. 将区（十进制）转化为十六进制。
2. 将转化的十六进制加上 A0，得到 GB2312 编码的高字节。

3. 将位（十进制）转化为十六进制。
4. 将转化的十六进制加上 A0，得到 GB2312 编码的低字节。
5. 组合区和位，区在高字节，位在低字节。
6. 得到 GB2312 编码。

GB2312 用两个字节编码，采用分区编码，总共编码的中文个数为 6763（3755 + 3008 ）。这些汉字只是最常用的汉字，已经覆盖中国大陆 99.75%的使用频率。但是，还有一些汉字在 GB2312 中没有被编码，如'镕'字，在 GB2312 中就没有被编码，这样就导致了问题，随之就出现了主流的 GBK 编码。

3、优点与缺点

优点：
适用于简体中文环境，属于中国国家标准，在大陆得到广泛支持；

缺点
不兼容繁体中文，其汉字集合过少。

四、UTF-8/16

1、历史

UNICODE 来到时，一起到来的还有计算机网络的兴起，UNICODE 如何在网络上传输也是一个必须考虑的问题，于是面向传输的众多 UTF 标准出现了，顾名思义，UTF8 就是每次 8 个位传输数据，而 UTF16 就是每次 16 个位，只不过为了传输时的可靠性，从 UNICODE 到 UTF 时并不是直接的对应，而是要过一些算法和规则来转换。

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。

2、含义

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，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，可判断编码是否有错误或数据传输过程中是否有错误。

3、编码方式

UTF-8

UTF-8 是一种变长编码方式，使用 1-4 个字节进行编码。UTF-8 完全兼容 ASCII，对于 ASCII 中的字符，UTF-8 采用的编码值跟 ASCII 完全一致。UTF-8 是 Unicode 一种具体的编码实现 。UTF-8 是在互联网上使用最广的一种 Unicode 的编码规则，因为这种编码有利于节约网络流量（因为变长编码，而非统一长度编码）。关于 Unicode 码点如何转化为 UTF-8 编码，可以参照如下规则：

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

UTF-8 是不定长的编码，使用 1、2、3、4 个字节编码，而 UTF-16 则只使用 2 或 4 个字节编码。UTF-16 也是 Unicode 一种具体的编码实现。关于 Unicode 如何转化为 UTf-16 编码规则如下

① 若 Unicode 码点在第一平面（BPM）中，则使用 2 个字节进行编码。

② 若 Unicode 码点在其他平面（辅助平面），则使用 4 个字节进行编码。

关于辅助平面的码点编码更详细解析如下：辅助平面码点被编码为一对 16 比特（四个字节）长的码元，称之为代理对 (surrogate pair)，第一部分称为高位代理 (high surrogate)或

前导代理 (lead surrogates), 码位范围为 :D800-DBFF. 第二部分称为低位代理(low surrogate)或后尾代理(trail surrogates)， 码位范围为：DC00-DFFF。注意，高位代理的码位从 D800 到 DBFF，而低位代理的码位从 DC00 到 DFFF，总共恰好为 D800-DFFF，这部分码点在第一平面内是保留的，不映射到任何字符，所以 UTF-16 编码巧妙的利用了这点来进行码点在辅助平面内的 4 字节编码。

4、特性

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

所有>U+007F 的 UCS 字符被编码为一个多个字节的串, 每个字节都有标记位集。因此, ASCII 字节（0x00-0x7F）不可能作为任何其他字符的一部分。

表示非 ASCII 字符的多字节串的第一个字节总是在 0xC0 到 0xFD 的范围里，并指出这个字符包含多少个字节。多字节串的其余字节都在 0x80 到 0xBF 范围里，这使得重新同步非常容易，并使编码无国界，且很少受丢失字节的影响。

可以编入所有可能的 231 个 UCS 代码

UTF-8 编码字符理论上可以最多到 6 个字节长，然而 16 位 BMP 字符最多只用到 3 字节长。

Bigendian UCS-4 字节串的排列顺序是预定的。

字节 0xFE 和 0xFF 在 UTF-8 编码中从未用到，同时，UTF-8 以字节为编码单元，它的字节顺序在所有系统中都是一样的，没有字节序的问题，也因此它实际上并不需要 BOM。

与 UTF-16 或其他 Unicode 编码相比，对于不支持 Unicode 和 XML 的系统，UTF-8 更不容易造成问题。

5、优点与缺点

优点：

总体来说，在 Unicode 字符串中不可能由码点数量决定显示它所需要的长度，或者显示字符串之后在文本缓冲区中光标应该放置的位置；组合字符、变宽字体、不可打印字符和从右至左的文字都是其归因。

所以尽管在 UTF-8 字符串中字符数量与码点数量的关系比 UTF-32 更为复杂，在实际中很少会遇到有不同的情形。

更详细的说，UTF-8 编码具有以下几点优点：

ASCII 是 UTF-8 的一个子集。因为一个纯 ASCII 字符串也是一个合法的 UTF-8 字符串，所以现存的 ASCII 文本不需要转换。为传统的扩展 ASCII 字符集设计的软件通常可以不经修改或很少修改就能与 UTF-8 一起使用。

使用标准的面向字节的排序例程对 UTF-8 排序将产生与基于 Unicode 代码点排序相同的结果。

UTF-8 和 UTF-16 都是可扩展标记语言文档的标准编码。所有其它编码都必须通过显式或文本声明来指定。

任何面向字节的字符串搜索算法都可以用于 UTF-8 的数据。但是，对于包含字符记数的正则表达式或其它结构必须小心。

UTF-8 字符串可以由一个简单的算法可靠地识别出来。就是，一个字符串在任何其它编码中表现为合法的 UTF-8 的可能性很低，并随字符串长度增长而减小。举例说，字符值 C0,C1,F5 至 FF 从来没有出现。为了更好的可靠性，可以使用正则表达式来统计非法过长和替代值。

与 UCS-2 的比较：ASCII 转换成 UCS-2，在编码前插入一个 0x0。用这些编码，会含括一些控制符，比如"或 '/'，这在 UNIX 和一些 C 函数中，将会产生严重错误。因此可以肯定，UCS-2 不适合作为 Unicode 的外部编码，也因此诞生了 UTF-8。

缺点：

编写不良的解析器：

一份写得很差（并且与当前标准的版本不兼容）的 UTF-8 解析器可能会接受一些不同的伪 UTF-8 表示并将它们转换到相同的 Unicode 输出上。这为设计用于处理八位表示的校验例程提供了一种遗漏信息的方式。

不利于正则表达式检索

正则表达式可以进行很多英文高级的模糊检索。例如，[a-h]表示 a 到 h 间所有字母。

同样 GBK 编码的中文也可以这样利用正则表达式，比如在只知道一个字的读音而不知道怎么写的情况下，也可用正则表达式检索，因为 GBK 编码是按读音排序的。只是 UTF-8 不是按读音排序的，所以会对正则表达式检索造成不利影响。但是这种使用方式并未考虑中文中的破音字，因此影响不大。Unicode 是按部首排序的，因此在只知道一个字的部首而不知道如何发音的情况下，UTF-8 可用正则表达式检索而 GBK 不行。

六、总结

在计算机中字符通常并不是保存为图像，每个字符都是使用一个编码来表示的，而每个字符究竟使用哪个编码代表，要取决于使用哪个字符集。

由于每种语言都制定了自己的字符集，导致最后存在的各种字符集实在太多，在国际交流中要经常转换字符集非常不便。因此，提出了 Unicode 字符集，它固定使用 16 bits 来表示一个字符，共可以表示 65536 个字符。将世界上几乎所有语言的常用字符收录其中，方便了信息交流。标准的 Unicode 称为 UTF-16。后来为了双字节的 Unicode 能够在现存的处理单字节的系统上正确传输，出现了 UTF-8，使用类似 MBCS 的方式对 Unicode 进行编码。注意 UTF-8 是编码，它属于 Unicode 字符集。Unicode 字符集有多种编码形式，而 ASCII 只有一种， GB-2312 也只有一种。