编码集

一些相关中文字符集的分析

张文倩 | XML | 2017.9

编码发展史

从头讲讲编码的故事。那么就让我们找个草堆坐下，先抽口烟，看看夜晚天空上的银河，然后想一想要从哪里开始讲起。嗯，也许这样开始比较好……

很久很久以前，有一群人，他们决定用 8 个可以开合的晶体管来组合成不同的状态，以表示世界上的万物。他们看到 8 个开关状态是好的，于是他们把这称为"字节"。

再后来，他们又做了一些可以处理这些字节的机器，机器开动了，可以用字节来组合出很多状态，状态开始变来变去。他们看到这样是好的，于是它们就这机器称为"计算机"。

开始计算机只在美国用。八位的字节一共可以组合出 256(2 的 8 次方)种不同的状态。

他们把其中的编号从 0 开始的 32 种状态分别规定了特殊的用途，一但终端、打印机遇上约定好的这些字节被传过来时，就要做一些约定的动作。遇上 00x10, 终端就换行，遇上 0x07, 终端就向人们嘟嘟叫，例好遇上 0x1b, 打印机就打印反白的字，或者终端就用彩色显示字母。他们看到这样很好，于是就把这些 0x20 以下的字节状态称为"控制码"。

他们又把所有的空格、标点符号、数字、大小写字母分别用连续的字节状态表示，一直编到了第 127 号，这样计算机就可以用不同字节来存储英语的文字了。大家看到这样，都感觉很好，于是大家都把这个方案叫做 ANSI 的"Ascii"编码（American Standard Code for Information Interchange，美国信息互换标准代码）。当时世界上所有的计算机都用同样的 ASCII 方案来保存英文文字。

后来，就像建造巴比伦塔一样，世界各地的都开始使用计算机，但是很多国家用的不是英文，他们的字母里有许多是 ASCII 里没有的，为了可以在计算机保存他们的文字，他们决定

采用 127 号之后的空位来表示这些新的字母、符号，还加入了很多画表格时需要用下到的横线、竖线、交叉等形状，一直把序号编到了最后一个状态 255。从 128 到 255 这一页的字

符集被称"扩展字符集"。从此之后，贪婪的人类再没有新的状态可以用了，美帝国主义可能没有想到还有第三世界国家的人们也希望可以用到计算机吧！

等中国人得到计算机时，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呢。但是这难不倒智慧的中国人民，我们不客气地把那些 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。

中国人民看到这样很不错，于是就把这种汉字方案叫做 "GB2312"。GB2312 是对 ASCII 的中文扩展。

但是中国的汉字太多了，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。

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。

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。

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……"

因为当时各个国家都像中国这样搞出一套自己的编码标准，结果互相之间谁也不懂谁的编码，谁也不支持别人的编码，连大陆和台湾这样只相隔了 150 海里，使用着同一种语言的兄弟地区，也分别采用了不同的 DBCS 编码方案——当时的中国人想让电脑显示汉字，就必须装上一个"汉字系统"，专门用来处理汉字的显示、输入的问题，但是那个台湾的愚昧封建人士写的算命程序就必须加装另一套支持 BIG5 编码的什么"倚天汉字系统"才可以用，装错了字符系统，显示就会乱了套！这怎么办？而且世界民族之林中还有那些一时用不上电脑的穷苦人民，他们的文字又怎么办？

真是计算机的巴比伦塔命题啊！

正在这时，大天使加百列及时出现了——一个叫 ISO （国际标谁化组织）的国际组织决定着手解决这个问题。他们采用的方法很简单：废了所有的地区性编码方案，重新搞一个包括了地球上所有文化、所有字母和符号的编码！他们打算叫它"Universal Multiple-Octet Coded Character Set"，简称 UCS, 俗称 "UNICODE"。

UNICODE 开始制订时，计算机的存储器容量极大地发展了，空间再也不成为问题了。于是 ISO 就直接规定必须用两个字节，也就是 16 位来统一表示所有的字符，对于 ascii 里的那些“半角”字符，UNICODE 保持其原编码不变，只是将其长度由原来的 8 位扩展为 16 位，而其他文化和语言的字符则全部重新统一编码。由于"半角"英文符号只需要用到低 8 位，所以其高 8 位永远是 0，因此这种大气的方案在保存英文文本时会多浪费一倍的空间。

这时候，从旧社会里走过来的程序员开始发现一个奇怪的现象：他们的 strlen 函数靠不住了，一个汉字不再是相当于两个字符了，而是一个！是的，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。

如前所述，UNICODE 是用两个字节来表示为一个字符，他总共可以组合出 65535 不同的字符，这大概已经可以覆盖世界上所有文化的符号。如果还不够也没有关系，ISO 已经

准备了 UCS-4 方案，说简单了就是四个字节来表示一个字符，这样我们就可以组合出 21 亿个不同的字符出来（最高位有其他用途），这大概可以用到银河联邦成立那一天吧！

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（UCS Transfer Format）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。

受到过网络编程加持的计算机僧侣们都知道，在网络里传递信息时有一个很重要的问题，就是对于数据高低位的解读方式，一些计算机是采用低位先发送的方法，例如我们 PC 机采用的 INTEL 架构，而另一些是采用高位先发送的方式，在网络中交换数据时，为了核对双方对于高低位的认识是否是一致的，采用了一种很简便的方法，就是在文本流的开始时向对方发送一个标志符——如果之后的文本是高位在位，那就发送"FEFF"，反之，则发送"FFFE"。不信你可以用二进制方式打开一个 UTF-X 格式的文件，看看开头两个字节是不是这两个字节？

讲到这里，我们再顺便说说一个很著名的奇怪现象：当你在 windows 的记事本里新建一个文件，输入"联通"两个字之后，保存，关闭，然后再次打开，你会发现这两个字已经消失了，代之的是几个乱码！呵呵，有人说这就是联通之所以拼不过移动的原因。

其实这是因为 GB2312 编码与 UTF8 编码产生了编码冲撞的原因。

从网上引来一段从 UNICODE 到 UTF8 的转换规则：

Unicode

UTF-8

0000 - 007F

0xxxxxxx

0080 - 07FF

110xxxxx 10xxxxxx

0800 - FFFF

1110xxxx 10xxxxxx 10xxxxxx

例如"汉"字的 Unicode 编码是 6C49。6C49 在 0800-FFFF 之间，所以要用 3 字节模板：1110xxxx 10xxxxxx 10xxxxxx。将 6C49 写成二进制是：0110 1100 0100 1001，将这个比特流按三字节模板的分段方法分为 0110 110001 001001，依次代替模板中的 x，得到：1110-0110 10-110001 10-001001，即 E6 B1 89，这就是其 UTF8 的编码。而当你新建一个文本文件时，记事本的编码默认是 ANSI, 如果你在 ANSI 的编码输入汉字，那么他实际就是 GB 系列的编码方式，在这种编码下，"联通"的内码是：

c1 1100 0001

aa 1010 1010

cd 1100 1101

a8 1010 1000

注意到了吗？第一二个字节、第三四个字节的起始部分的都是"110"和"10"，正好与 UTF8 规则里的两字节模板是一致的，于是再次打开记事本时，记事本就误认为这是一个 UTF8 编码的文件，让我们把第一个字节的 110 和第二个字节的 10 去掉，我们就得到了"00001 101010"，再把各位对齐，补上前导的 0，就得到了"0000 0000 0110 1010"，不好意思，这是 UNICODE 的 006A，也就是小写的字母"j"，而之后的两字节用 UTF8 解码之后是 0368，这个字符什么也不是。这就是只有"联通"两个字的文件没有办法在记事本里正常显示的原因。

而如果你在"联通"之后多输入几个字，其他的字的编码不见得又恰好是 110 和 10 开始的字节，这样再次打开时，记事本就不会坚持这是一个 utf8 编码的文件，而会用 ANSI 的方式解读之，这时乱码又不出现了。

好了，终于可以回答 NICO 的问题了，在数据库里，有 n 前缀的字串类型就是 UNICODE 类型，这种类型中，固定用两个字节来表示一个字符，无论这个字符是汉字还是英文字母，或是别的什么。

如果你要测试"abc 汉字"这个串的长度，在没有 n 前缀的数据类型里，这个字串是 7 个字符的长度，因为一个汉字相当于两个字符。而在有 n 前缀的数据类型里，同样的测试串长度的函数将会告诉你是 5 个字符，因为一个汉字就是一个字符。