全球讯息：Unicode编码介绍?

时间： 2022-10-24 13:44:53 来源： IT专家网

最近这段时间总有小伙伴问小编Unicode编码介绍是什么，小编为此在网上搜寻了一些有关于Unicode编码介绍的知识送给大家，希望能解答各位小伙伴的疑惑。

先从ASCII说起。ASCII是用来表示英文字符的一种编码规范，每个ASCII字符占用1个字节（8bits）

【资料图】

因此，ASCII编码可以表示的最大字符数是256，其实英文字符并没有那么多，一般只用前128个（最高位为0），其中包括了控制字符、数字、大小写字母和其他一些符号。

而最高位为1的另128个字符被成为“扩展ASCII”，一般用来存放英文的制表符、部分音标字符等等的一些其他符号

这种字符编码规范显然用来处理英文没有什么问题。（实际上也可以用来处理法文、德文等一些其他的西欧字符，但是不能和英文通用），但是面对中文、阿拉伯文之类复杂的文字，255个字符显然不够用

于是，各个国家纷纷制定了自己的文字编码规范，其中中文的文字编码规范叫做“GB2312-80”，它是和ASCII兼容的一种编码规范，其实就是利用扩展ASCII没有真正标准化这一点，把一个中文字符用两个扩展ASCII字符来表示。

但是这个方法有问题，最大的问题就是，中文文字没有真正属于自己的编码，因为扩展ASCII码虽然没有真正的标准化，但是PC里的ASCII码还是有一个事实标准的（存放着英文制表符），所以很多软件利用这些符号来画表格。这样的软件用到中文系统中，这些表格符就会被误认作中文字，破坏版面。而且，统计中英文混合字符串中的字数，也是比较复杂的，我们必须判断一个ASCII码是否扩展，以及它的下一个ASCII是否扩展，然后才“猜”那可能是一个中文字。

总之当时处理中文是很痛苦的。而更痛苦的是GB2312是国家标准，台湾当时有一个Big5编码标准，很多编码和GB是相同的，所以……，嘿嘿。

这时候，我们就知道，要真正解决中文问题，不能从扩展ASCII的角度入手，也不能仅靠中国一家来解决。而必须有一个全新的编码系统，这个系统要可以将中文、英文、法文、德文……等等所有的文字统一起来考虑，为每个文字都分配一个单独的编码，这样才不会有上面那种现象出现。

于是，Unicode诞生了。

Unicode有两套标准，一套叫UCS-2(Unicode-16)，用2个字节为字符编码，另一套叫UCS-4(Unicode-32)，用4个字节为字符编码。

以目前常用的UCS-2为例，它可以表示的字符数为2^16=65535，基本上可以容纳所有的欧美字符和绝大部分的亚洲字符。

UTF-8的问题后面会提到。

在Unicode里，所有的字符被一视同仁。汉字不再使用“两个扩展ASCII”，而是使用“1个Unicode”，注意，现在的汉字是“一个字符”了，于是，拆字、统计字数这些问题也就自然而然的解决了。

但是，这个世界不是理想的，不可能在一夜之间所有的系统都使用Unicode来处理字符，所以Unicode在诞生之日，就必须考虑一个严峻的问题：和ASCII字符集之间的不兼容问题。

我们知道，ASCII字符是单个字节的，比如“A”的ASCII是65。而Unicode是双字节的，比如“A”的Unicode是0065，这就造成了一个非常大的问题：以前处理ASCII的那套机制不能被用来处理Unicode了。

另一个更加严重的问题是，C语言使用""作为字符串结尾，而Unicode里恰恰有很多字符都有一个字节为0，这样一来，C语言的字符串函数将无法正常处理Unicode，除非把世界上所有用C写的程序以及他们所用的函数库全部换掉。

于是，比Unicode更伟大的东东诞生了，之所以说它更伟大是因为它让Unicode不再存在于纸上，而是真实的存在于我们大家的电脑中。那就是：UTF。

UTF=UCSTransformationFormatUCS转换格式

它是将Unicode编码规则和计算机的实际编码对应起来的一个规则。现在流行的UTF有2种：UTF-8和UTF-16。

其中UTF-16和上面提到的Unicode本身的编码规范是一致的，这里不多说了。而UTF-8不同，它定义了一种“区间规则”，这种规则可以和ASCII编码保持最大程度的兼容。

UTF-8有点类似于Haffman编码，它将Unicode编码为00000000-0000007F的字符，用单个字节来表示；

00000080-000007FF的字符用两个字节表示

00000800-0000FFFF的字符用3字节表示

因为目前为止Unicode-16规范没有指定FFFF以上的字符，所以UTF-8最多是使用3个字节来表示一个字符。但理论上来说，UTF-8最多需要用6字节表示一个字符。

在UTF-8里，英文字符仍然跟ASCII编码一样，因此原先的函数库可以继续使用。而中文的编码范围是在0080-07FF之间，因此是2个字节表示（但这两个字节和GB编码的两个字节是不同的），用专门的Unicode处理类可以对UTF编码进行处理。

下面说说中文的问题。

由于历史的原因，在Unicode之前，一共存在过3套中文编码标准。

GB2312-80，是中国大陆使用的国家标准，其中一共编码了6763个常用简体汉字。
Big5，是台湾使用的编码标准，编码了台湾使用的繁体汉字，大概有8千多个。
HKSCS，是中国香港使用的编码标准，字体也是繁体，但跟Big5有所不同。

这3套编码标准都采用了两个扩展ASCII的方法，因此，几套编码互不兼容，而且编码区间也各有不同

因为其不兼容性，在同一个系统中同时显示GB和Big5基本上是不可能的。当时的南极星、RichWin等等软件，在自动识别中文编码、自动显示正确编码方面都做了很多努力。

他们用了怎样的技术我就不得而知了，我知道好像南极星曾经以同屏显示繁简中文为卖点。

后来，由于各方面的原因，国际上又制定了针对中文的统一字符集GBK和GB18030，其中GBK已经在Windows、Linux等多种操作系统中被实现。

GBK兼容GB2312，并增加了大量不常用汉字，还加入了几乎所有的Big5中的繁体汉字。但是GBK中的繁体汉字和Big5中的几乎不兼容。

GB18030相当于是GBK的超集，比GBK包含的字符更多。据我所知目前还没有操作系统直接支持GB18030。

谈谈Unicode编码，简要解释UCS、UTF、BMP、BOM等名词

这是一篇程序员写给程序员的趣味读物。所谓趣味是指可以比较轻松地了解一些原来不清楚的概念，增进知识，类似于打RPG游戏的升级。整理这篇文章的动机是两个问题：

问题一：

使用Windows记事本的“另存为”，可以在GBK、Unicode、Unicodebigendian和UTF-8这几种编码方式间相互转换。同样是txt文件，Windows是怎样识别编码方式的呢？

我很早前就发现Unicode、Unicodebigendian和UTF-8编码的txt文件的开头会多出几个字节，分别是FF、FE（Unicode）,FE、FF（Unicodebigendian）,EF、BB、BF（UTF-8）。但这些标记是基于什么标准呢？

问题二：

最近在网上看到一个ConvertUTF.c，实现了UTF-32、UTF-16和UTF-8这三种编码方式的相互转换。对于Unicode(UCS2)、GBK、UTF-8这些编码方式，我原来就了解。但这个程序让我有些糊涂，想不起来UTF-16和UCS2有什么关系。

查了查相关资料，总算将这些问题弄清楚了，顺带也了解了一些Unicode的细节。写成一篇文章，送给有过类似疑问的朋友。本文在写作时尽量做到通俗易懂，但要求读者知道什么是字节，什么是十六进制。

0、bigendian和littleendian

bigendian和littleendian是CPU处理多字节数的不同方式。例如“汉”字的Unicode编码是6C49。那么写到文件里时，究竟是将6C写在前面，还是将49写在前面？如果将6C写在前面，就是bigendian。还是将49写在前面，就是littleendian。

“endian”这个词出自《格列佛游记》。小人国的内战就源于吃鸡蛋时是究竟从大头(Big-Endian)敲开还是从小头(Little-Endian)敲开，由此曾发生过六次叛乱，其中一个皇帝送了命，另一个丢了王位。

我们一般将endian翻译成“字节序”，将bigendian和littleendian称作“大尾”和“小尾”。

1、字符编码、内码，顺带介绍汉字编码

字符必须编码后才能被计算机处理。计算机使用的缺省编码方式就是计算机的内码。早期的计算机使用7位的ASCII编码，为了处理汉字，程序员设计了用于简体中文的GB2312和用于繁体中文的big5。

GB2312(1980年)一共收录了7445个字符，包括6763个汉字和682个其它符号。汉字区的内码范围高字节从B0-F7，低字节从A1-FE，占用的码位是72*94=6768。其中有5个空位是D7FA-D7FE。

GB2312支持的汉字太少。1995年的汉字扩展规范GBK1.0收录了21886个符号，它分为汉字区和图形符号区。汉字区包括21003个字符。2000年的GB18030是取代GBK1.0的正式国家标准。该标准收录了27484个汉字，同时还收录了藏文、蒙文、维吾尔文等主要的少数民族文字。现在的PC平台必须支持GB18030，对嵌入式产品暂不作要求。所以手机、MP3一般只支持GB2312。

从ASCII、GB2312、GBK到GB18030，这些编码方法是向下兼容的，即同一个字符在这些方案中总是有相同的编码，后面的标准支持更多的字符。在这些编码中，英文和中文可以统一地处理。区分中文编码的方法是高字节的最高位不为0。按照程序员的称呼，GB2312、GBK到GB18030都属于双字节字符集(DBCS)。

有的中文Windows的缺省内码还是GBK，可以通过GB18030升级包升级到GB18030。不过GB18030相对GBK增加的字符，普通人是很难用到的，通常我们还是用GBK指代中文Windows内码。

这里还有一些细节：

GB2312的原文还是区位码，从区位码到内码，需要在高字节和低字节上分别加上A0。

在DBCS中，GB内码的存储格式始终是bigendian，即高位在前。

GB2312的两个字节的最高位都是1。但符合这个条件的码位只有128*128=16384个。所以GBK和GB18030的低字节最高位都可能不是1。不过这不影响DBCS字符流的解析：在读取DBCS字符流时，只要遇到高位为1的字节，就可以将下两个字节作为一个双字节编码，而不用管低字节的高位是什么。

2、Unicode、UCS和UTF

前面提到从ASCII、GB2312、GBK到GB18030的编码方法是向下兼容的。而Unicode只与ASCII兼容（更准确地说，是与ISO-8859-1兼容），与GB码不兼容。例如“汉”字的Unicode编码是6C49，而GB码是BABA。

Unicode也是一种字符编码方法，不过它是由国际组织设计，可以容纳全世界所有语言文字的编码方案。Unicode的学名是UniversalMultiple-OctetCodedCharacterSet，简称为UCS。UCS可以看作是UnicodeCharacterSet的缩写。

根据维基百科全书的记载：历史上存在两个试图独立设计Unicode的组织，即国际标准化组织（ISO）和一个软件制造商的协会（unicode.org）。ISO开发了ISO10646项目，Unicode协会开发了Unicode项目。

在1991年前后，双方都认识到世界不需要两个不兼容的字符集。于是它们开始合并双方的工作成果，并为创立一个单一编码表而协同工作。从Unicode2.0开始，Unicode项目采用了与ISO10646-1相同的字库和字码。

目前两个项目仍都存在，并独立地公布各自的标准。Unicode协会现在的最新版本是2005年的Unicode4.1.0。ISO的最新标准是10646-3:2003。

UCS规定了怎么用多个字节表示各种文字。怎样传输这些编码，是由UTF(UCSTransformationFormat)规范规定的，常见的UTF规范包括UTF-8、UTF-7、UTF-16。

IETF的RFC2781和RFC3629以RFC的一贯风格，清晰、明快又不失严谨地描述了UTF-16和UTF-8的编码方法。我总是记不得IETF是InternetEngineeringTaskForce的缩写。但IETF负责维护的RFC是Internet上一切规范的基础。

3、UCS-2、UCS-4、BMP

UCS有两种格式：UCS-2和UCS-4。顾名思义，UCS-2就是用两个字节编码，UCS-4就是用4个字节（实际上只用了31位，最高位必须为0）编码。下面让我们做一些简单的数学游戏：

UCS-2有2^16=65536个码位，UCS-4有2^31=2147483648个码位。

UCS-4根据最高位为0的最高字节分成2^7=128个group。每个group再根据次高字节分为256个plane。每个plane根据第3个字节分为256行(rows)，每行包含256个cells。当然同一行的cells只是最后一个字节不同，其余都相同。

group0的plane0被称作BasicMultilingualPlane,即BMP。或者说UCS-4中，高两个字节为0的码位被称作BMP。

将UCS-4的BMP去掉前面的两个零字节就得到了UCS-2。在UCS-2的两个字节前加上两个零字节，就得到了UCS-4的BMP。而目前的UCS-4规范中还没有任何字符被分配在BMP之外。

4、UTF编码

UTF-8就是以8位为单元对UCS进行编码。从UCS-2到UTF-8的编码方式如下：

UCS-2编码(16进制)UTF-8字节流(二进制)

0000-007F0xxxxxxx

0080-07FF110xxxxx10xxxxxx

0800-FFFF1110xxxx10xxxxxx10xxxxxx

例如“汉”字的Unicode编码是6C49。6C49在0800-FFFF之间，所以肯定要用3字节模板了：1110xxxx10xxxxxx10xxxxxx。将6C49写成二进制是：0110110001001001，用这个比特流依次代替模板中的x，得到：111001101011000110001001，即E6B189。

读者可以用记事本测试一下我们的编码是否正确。

UTF-16以16位为单元对UCS进行编码。对于小于0x10000的UCS码，UTF-16编码就等于UCS码对应的16位无符号整数。对于不小于0x10000的UCS码，定义了一个算法。不过由于实际使用的UCS2，或者UCS4的BMP必然小于0x10000，所以就目前而言，可以认为UTF-16和UCS-2基本相同。但UCS-2只是一个编码方案，UTF-16却要用于实际的传输，所以就不得不考虑字节序的问题。

5、UTF的字节序和BOM

UTF-8以字节为编码单元，没有字节序的问题。UTF-16以两个字节为编码单元，在解释一个UTF-16文本前，首先要弄清楚每个编码单元的字节序。例如收到一个“奎”的Unicode编码是594E，“乙”的Unicode编码是4E59。如果我们收到UTF-16字节流“594E”，那么这是“奎”还是“乙”？

Unicode规范中推荐的标记字节顺序的方法是BOM。BOM不是“BillOfMaterial”的BOM表，而是ByteOrderMark。BOM是一个有点小聪明的想法：

在UCS编码中有一个叫做ZEROWIDTHNO-BREAKSPACE的字符，它的编码是FEFF。而FFFE在UCS中是不存在的字符，所以不应该出现在实际传输中。UCS规范建议我们在传输字节流前，先传输字符ZEROWIDTHNO-BREAKSPACE。

这样如果接收者收到FEFF，就表明这个字节流是Big-Endian的；如果收到FFFE，就表明这个字节流是Little-Endian的。因此字符ZEROWIDTHNO-BREAKSPACE又被称作BOM。

UTF-8不需要BOM来表明字节顺序，但可以用BOM来表明编码方式。字符ZEROWIDTHNO-BREAKSPACE的UTF-8编码是EFBBBF（读者可以用我们前面介绍的编码方法验证一下）。所以如果接收者收到以EFBBBF开头的字节流，就知道这是UTF-8编码了。

Windows就是使用BOM来标记文本文件的编码方式的。

6、进一步的参考资料

本文主要参考的资料是ShortoverviewofISO-IEC10646andUnicode

关键词： 编码方法 (http://information.paoku.com.cn/tagbianmafangfa/) 国家标准 (http://information.paoku.com.cn/tagguojiabiaozhun/)

责任编辑：QL0009

为你推荐