

文章编号：1000-5013(2009)01-0038-04

# GB 18030与Unicode编码转换算法

程小刚<sup>1</sup>, 郭韧<sup>2</sup>

(1. 华侨大学 计算机科学与技术学院; 2. 华侨大学 工商管理学院, 福建泉州 362021)

**摘要：**在分析 GB 18030 和 Unicode 标准编码空间和对应关系的基础上, 提出一种基于计算和查表相结合的转换算法。它利用这两种编码标准之间的“范围对应”关系, 显著减少对应表的大小, 因而可用于对资源要求十分严格的嵌入式设备与系统中。所提出的算法经简单变换也可用于不同的 Unicode 编码方式, 如 UTF8, UTF16 等与 GB 18030 之间的转换。

**关键词：**GB 18030; Unicode; 汉字编码; 编码转换

**中图分类号：**TP 335; TP 391.11

**文献标识码：**A

2000年3月17日, 国家质量技术监督局发布了一项国家标准——《GB 18030 - 2000 信息技术信息交换用汉字编码字符集——基本集的扩充》(简称 GB 18030)<sup>[1]</sup>。它不但代表着有关中文信息处理的标准化走上一个新阶段, 使中文处理在质和量的方面更适应网络时代的要求, 而且对发展我国的中文信息处理有着重大的指导作用和深刻的影响。GB 18030 是强制性标准, 凡在 2001 年 8 月 31 日后正式发布或出厂的个人计算机操作系统产品, 必须符合 GB 18030 相关要求<sup>[2-3]</sup>。Unicode 是一种国际字符集标准, 它给每个字符提供了一个唯一的数字, 是实现 ISO/ IEC 10646 的正规方式。Unicode 标准的出现和支持它的存在, 是近来全球软件技术最重要的发展趋势。本文分析了这两种编码标准的编码空间和相互之间的对应关系, 并探讨两种重要字符集标准之间的转换算法。

## 1 编码对应关系

GB 18030 中包括 3 种长度的中文字符编码: 单字节、双字节和四字节<sup>[1,4]</sup>, 码位空间分配如表 1 所示。Unicode 的编码范围是 U + 0000 至 U + 10FFFF, 分为 17 个平面(Plane), 每个平面有 65 536 个码位, 其中基本平面 Plane 0(U + 0000 至 U + FFFF) 包含了所有常用的中外文字符。另在 Plane 1(U + 10000 至 U + 1FFFF) 和 Plane 2(U + 20000 至 U + 2FFFF) 中也定义有中文字符。探讨 UTF32 与 GB 18030 的转换算法, 而其他常用的 Unicode 编码格式如 UTF8 和 UTF16, 可先转换为 UTF32 后再应用本文的算法<sup>[5-6]</sup>。

表 1 GB 18030 编码空间

Tab. 1 GB 18030 code space

字节长度	码位空间		码位数目
单字节	0x00 ~ 0x80		129 个码位
双字节	第 1 字节 0x81 ~ 0xFE	第 2 字节 0x40 ~ 0x7E, 0x80 ~ 0xFE	23 940 个码位
四字节	第 1 字节 0x81 ~ 0xFE	第 2 字节 0x30 ~ 0x39	第 3 字节 0x81 ~ 0xFE
			第 4 字节 0x30 ~ 0x39
			1 587 600 个码位

对于单字节的 GB 18030 字符, 除 0x80 表示欧元符号与 Unicode 不同外, 其余单字节编码直接对应

收稿日期: 2007-11-23

作者简介: 程小刚(1973-), 男, 助教, 硕士, 主要从事网络与信息系统的研究。E-mail:cxg@hqu.edu.cn.

基金项目: 华侨大学科研基金资助项目(06HSK11)

于 Unicode 相应字符码位,无需转换。对于双字节的 GB 18030 字符,则要一张查询表来进行转换。对于四字节的 GB 18030 字符,可采用查表与计算相结合的方法来转换,因为一部分字符之间是“范围对应”的关系。如 U + 0452 至 U + 200F 这部分 Unicode 字符就一一对应于 0x8130D330 至 0x8136A531 的 GB 18030 字符。对此部分字符则可采取计算的方法来转换,这样可使查询表较小。其他的“范围对应”关系如下面定义的数组所示:

```
unsigned int
GB_18030Ranges[13][4] = {
    {0x0452, 0x200F, LINEAR(0x8130D330), LINEAR(0x8136A531)},
    {0x2643, 0x2E80, LINEAR(0x8137A839), LINEAR(0x8138FD38)},
    {0x361B, 0x3917, LINEAR(0x8230A633), LINEAR(0x8230F237)},
    {0x3CE1, 0x4055, LINEAR(0x8231D438), LINEAR(0x8232AF32)},
    {0x4160, 0x4336, LINEAR(0x8232C937), LINEAR(0x8232F837)},
    {0x44D7, 0x464B, LINEAR(0x8233A339), LINEAR(0x8233C931)},
    {0x478E, 0x4946, LINEAR(0x8233E838), LINEAR(0x82349638)},
    {0x49B8, 0x4C76, LINEAR(0x8234A131), LINEAR(0x8234E733)},
    {0x9FA6, 0xD7FF, LINEAR(0x82358F33), LINEAR(0x8336C738)},
    {0xE865, 0xF92B, LINEAR(0x8336D030), LINEAR(0x84308534)},
    {0xFA2A, 0xFE2F, LINEAR(0x84309C38), LINEAR(0x84318537)},
    {0xFFE6, 0xFFFF, LINEAR(0x8431A234), LINEAR(0x8431A439)},
    {0x10000, 0x10FFFF, LINEAR(0x90308130), LINEAR(0xE3329A35)}
};
```

其中,LINEAR 为计算 GB 18030 四字节字符的索引所定义的宏:

```
# define LINEAR_18030(a,b,c,d) (((a) * 10 + (b)) * 126L + (c)) * 10L + (d))
# define LINEAR_18030_BASE LINEAR_18030(0x81,0x30,0x81,0x30)
# define LINEAR(x) LINEAR_18030(x > 24, (x > 16) & 0xff, (x > 8) & 0xff, x & 0xff)
```

$10 = 0x39 - 0x30 + 1$ ,  $126 = 0xFE - 0x81 + 1$ , 而 LINEAR\_18030\_BASE 为第 1 个四字节的 GB 18030 字符所对应的索引值。

## 2 转换算法及程序代码

### 2.1 GB 18030 转换为 Unicode

(1) 单字节的 GB 18030 代码(0x00 ~ 0x80)。如上所述,对 0x80 要转换为欧元符号,即 Unicode 的 U + 20AC,其余的无需转换,直接对应于 Unicode 代码。因为这两种标准都考虑到了与 ASCII 码的兼容性,所以这部分编码是一致的。

(2) 双字节的 GB 18030 代码。记第 1,2 字节为 byte 1 和 byte 2,如果 byte 1 = 0xFE,byte 2 = 0x40,byte 2 = 0xFE 且 byte 2 = 0x7F,则此两字节为 1 个合法的双字节 GB 18030 字符。因此,可按如下方法计算索引:

```
index = (byte 1 - 0x81) * 191 + (byte 2 - 0x40).
```

其中, $191 = 0x FE - 0x40 + 1$ 。得到此索引后可在相应的查询表中查到相应的 Unicode 代码值:lookup\_GB 18030 to UNI2[index],查询表的生成参见第 3 节。

(3) 四字节的 GB 18030 代码。对 1 个合法的四字节 GB 18030 代码有

```
0x81 byte1 0xFE,0x30 byte2 0x39,
0x81 byte3 0xFE,0x30 byte4 0x39.
```

先计算索引值  $index = LINEAR_18030(byte 1, byte 2, byte 3, byte 4)$ ,然后,根据 index 是否位于上述“范围对应”关系来做不同处理。如果位于“范围对应”关系内,可进行计算得到相应 Unicode 值;否则,要查表获得相应 Unicode 值。其代码:

```

for(int i = 0; i < 13; i + +) {
    if(index >= GB_18030Ranges[i][2] && index <= GB_18030Ranges[i][3])
    {
        // range map , compute the corresponding unicode
        return index - GB_18030Ranges[i][2] + GB_18030Ranges[i][0];
    }
    if(index > GB_18030Ranges[i][3] && index < GB_18030Ranges[i + 1][2])
    {
        // lookup table
        for(int j = 0; j <= i; j + +)
            index - = (GB_18030Ranges[j][1] - GB_18030Ranges[j][0] + 1);
        index - = LINEAR_18030_BASE;
        return lookup_GBTOUNI4[index];
    }
}

```

查询表 lookup\_GBTOUNI4[ ]的生成参见第3节.

## 2.2 Unicode转换为GB 18030

用1个32位无符号整数 u32 来表示1个Unicode字符.

- (1) 如果 u32 > 0x10FFFF, 则错误, 没有这么大的 Unicode 编码.
- (2) 如果 u32 < GB\_18030Ranges[0][0], 则 u32 是位于低码位的 Unicode, 可直接以 u32 为索引, 从查询表中得到相应 GB\_18030 编码值:lookup\_UNItogb18030[u32], 查询表的生成见第3节.

(3) 对于其他 u32 值, 分是否处于“范围对应”关系内进行计算或查表的不同处理, 有

```

for(int i = 0; i < 13; i + +) {
    if(u32 >= GB_18030Ranges[i][0] && u32 <= GB_18030Ranges[i][1])
    {
        // range map , compute the corresponding GB 18030
        unsigned int gb = u32 - GB_18030Ranges[i][0] + GB_18030Ranges[i][2];
        gb - = LINEAR_18030_BASE;
        char bytes[4];
        bytes[3] = (char)(0x30 + gb % 10); gb / = 10;
        bytes[2] = (char)(0x81 + gb % 126); gb / = 126;
        bytes[1] = (char)(0x30 + gb % 10); gb / = 10;
        bytes[0] = (char)(0x81 + gb);
        gb = (bytes[0] & 0xff) < < 24 | (bytes[1] & 0xff) < < 16 | (bytes[2] & 0xff) < < 8 |
        (bytes[3] & 0xff);
        return gb;
    }
    if(u32 > GB_18030Ranges[i][1] && u32 < GB_18030Ranges[i + 1][0])
    {
        // lookup table
        for(int j = 0; j <= i; j + +)
            u32 - = (GB_18030Ranges[j][1] - GB_18030Ranges[j][0] + 1);
        return lookup_UNItogb18030[u32];
    }
}

```

### 3 查询表的生成

上述转换算法中用到了3张查询表:lookup\_GB 18030toUNI2[],lookup\_GB 18030toUNI4[]和lookup\_UNItoGB 18030[].它们分别是双字节GB 18030至Unicode转换表,四字节GB 18030至Unicode转换表和Unicode至GB 18030转换表。

这3张查询表皆为一维数组,可利用微软公司Windows的GB 18030支持库中转换函数MultiByte To WideChar(GB 18030至Unicode)/WideChar ToMultiByte(Unicode至GB 18030)生成得到。对于lookup\_GB 18030toUNI2[],可直接逐一对每一双字节GB 18030码调用MultiByte To WideChar来得到相应的Unicode,按一定格式写入文件即可得到查询表。而对lookup\_GB 18030toUNI4[]和lookup\_UNItoGB 18030[]需要注意的是“范围对应”关系,即在生成查询时要跳过那些成“范围对应”关系的字符。因为这些字符在转换时是通过计算得到相应码值,而不是查表得到的。

### 4 结束语

分析国家标准GB 18030和Unicode标准编码空间和对应关系,提出一种基于计算和查表相结合的转换算法。利用这两种编码标准之间的范围对应关系,显著减少对应表的大小,可用于对资源要求十分严格的嵌入式设备与系统中。本文所提出的算法经简单变换也可用于不同的Unicode编码方式,如UTF8,UTF16等与GB 18030之间的转换。

#### 参考文献:

- [1] 国家质量技术监督局. GB 18030 - 2000 信息交换用汉字编码字符集——基本集的扩充[S]. 北京:中国标准出版社,2000:3-17.
- [2] 孟庆余. 汉字编码字符集的新标准——GB 18030 - 2000[J]. 微型机与应用,2000, 19(12): 4-6.
- [3] 王立建,陈 壮,王 欣,等. 中文信息处理标准化[J]. 中国标准化,2002, 45(6): 20-22.
- [4] 闫凡蕾,林仲湘,李 龙. 古籍电子化中生僻汉字的处理[J]. 华侨大学学报:自然科学版,2003, 24(3): 331-334.
- [5] 何 华,卜佳俊. 数据库管理系统的多字符集支持[J]. 计算机应用研究,2005, 22(12): 79-81.
- [6] 鹿文鹏,薛若娟. Unicode与UTF-8编码转换方法研究[J]. 计算机时代,2005, 23(9): 44-45.

## Research on Code Transformation Algorithm between GB 18030 and Unicode

CHEN G Xiao-gang<sup>1</sup>, GUO Ren<sup>2</sup>

(1. College of Computer Science and Technology, Huaqiao University;  
2. College of Business Administration, Huaqiao University, Quanzhou 362021, China)

**Abstract:** Based on the analysis of the relationship between GB 18030 and Unicode standards code space, put forward a transformation algorithm combined computation and table lookup. The algorithm exploits the range mapping relationship between these two coding standards, thus can significantly reduce the mapping table size, which is crucial for embedded devices with limited resources. With little change, the algorithm can also be used for transformation between GB 18030 and other Unicode coding formats, such as UTF8 and UTF16.

**Keywords:** GB 18030; unicode; chinese character coding; code transformation

(责任编辑:钱筠 英文审校:吴逢铁)