一、ASCII 码

我们知道，计算机内部，所有信息最终都是一个二进制值。每一个二进制位（bit）有`0`和`1`两种状态，因此八个二进制位就可以组合出256种状态，这被称为一个字节（byte）。也就是说，一个字节一共可以用来表示256种不同的状态，每一个状态对应一个符号，就是256个符号，从`00000000`到`11111111`。

上个世纪60年代，美国制定了一套字符编码，对英语字符与二进制位之间的关系，做了统一规定。这被称为 ASCII 码，一直沿用至今。

ASCII 码一共规定了128个字符的编码，比如空格`SPACE`是32（二进制`00100000`），大写的字母`A`是65（二进制`01000001`）。这128个符号（包括32个不能打印出来的控制符号），只占用了一个字节的后面7位，最前面的一位统一规定为`0`。

二、非 ASCII 编码

英语用128个符号编码就够了，但是用来表示其他语言，128个符号是不够的。比如，在法语中，字母上方有注音符号，它就无法用 ASCII 码表示。于是，一些欧洲国家就决定，利用字节中闲置的最高位编入新的符号。比如，法语中的`é`的编码为130（二进制`10000010`）。这样一来，这些欧洲国家使用的编码体系，可以表示最多256个符号。

但是，这里又出现了新的问题。不同的国家有不同的字母，因此，哪怕它们都使用256个符号的编码方式，代表的字母却不一样。比如，130在法语编码中代表了`é`，在希伯来语编码中却代表了字母`Gimel`(`ג`)，在俄语编码中又会代表另一个符号。但是不管怎样，所有这些编码方式中，0--127表示的符号是一样的，不一样的只是128--255的这一段。

至于亚洲国家的文字，使用的符号就更多了，汉字就多达10万左右。一个字节只能表示256种符号，肯定是不够的，就必须使用多个字节表达一个符号。比如，简体中文常见的编码方式是 GB2312，使用两个字节表示一个汉字，所以理论上最多可以表示 256 x 256 = 65536 个符号。

中文编码的问题需要专文讨论，这篇笔记不涉及。这里只指出，虽然都是用多个字节表示一个符号，但是GB类的汉字编码与后文的 Unicode 和 UTF-8 是毫无关系的。

三. Unicode

正如上一节所说，世界上存在着多种编码方式，同一个二进制数字可以被解释成不同的符号。因此，要想打开一个文本文件，就必须知道它的编码方式，否则用错误的编码方式解读，就会出现乱码。为什么电子邮件常常出现乱码？就是因为发信人和收信人使用的编码方式不一样。

可以想象，如果有一种编码，将世界上所有的符号都纳入其中。每一个符号都给予一个独一无二的编码，那么乱码问题就会消失。这就是 Unicode，就像它的名字都表示的，这是一种所有符号的编码。

Unicode 当然是一个很大的集合，现在的规模可以容纳100多万个符号。每个符号的编码都不一样，比如，`U+0639`表示阿拉伯字母`Ain`，`U+0041`表示英语的大写字母`A`，`U+4E25`表示汉字`严`。具体的符号对应表，可以查询 unicode.org。

四、Unicode 的问题

需要注意的是，Unicode 只是一个符号集，它只规定了符号的二进制代码，却没有规定这个二进制代码应该如何存储。

比如，汉字`严`的 Unicode 是十六进制数`4E25`，转换成二进制数足足有15位（`100111000100101`），也就是说，这个符号的表示至少需要2个字节。表示其他更大的符号，可能需要3个字节或者4个字节，甚至更多。

这里就有两个严重的问题，第一个问题是，如何才能区别 Unicode 和 ASCII ？计算机怎么知道三个字节表示一个符号，而不是分别表示三个符号呢？第二个问题是，我们已经知道，英文字母只用一个字节表示就够了，如果 Unicode 统一规定，每个符号用三个或四个字节表示，那么每个英文字母前都必然有二到三个字节是`0`，这对于存储来说是极大的浪费，文本文件的大小会因此大出二三倍，这是无法接受的。

它们造成的结果是：1）出现了 Unicode 的多种存储方式，也就是说有许多种不同的二进制格式，可以用来表示 Unicode。2）Unicode 在很长一段时间内无法推广，直到互联网的出现。

五、UTF-8

互联网的普及，强烈要求出现一种统一的编码方式。UTF-8 就是在互联网上使用最广的一种 Unicode 的实现方式。其他实现方式还包括 UTF-16（字符用两个字节或四个字节表示）和 UTF-32（字符用四个字节表示），不过在互联网上基本不用。**重复一遍，这里的关系是，UTF-8 是 Unicode 的实现方式之一。**

UTF-8 最大的一个特点，就是它是一种变长的编码方式。它可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度。

UTF-8 的编码规则很简单，只有二条：

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。

下表总结了编码规则，字母`x`表示可用编码的位。

```
Unicode符号范围     |        UTF-8编码方式
(十六进制)          |              （二进制）
----------------------+---------------------------------------------
0000 0000-0000 007F | 0xxxxxxx
0000 0080-0000 07FF | 110xxxxx 10xxxxxx
0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx
0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

跟据上表，解读 UTF-8 编码非常简单。如果一个字节的第一位是`0`，则这个字节单独就是一个字符；如果第一位是`1`，则连续有多少个`1`，就表示当前字符占用多少个字节。

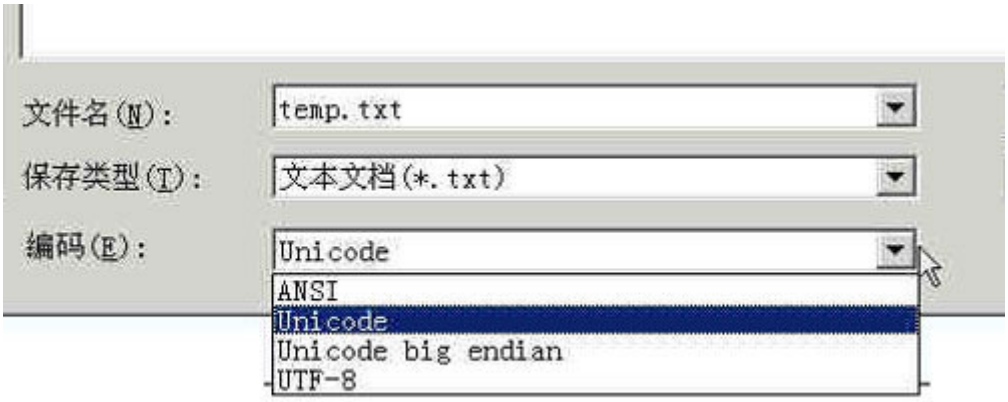
下面，还是以汉字`严`为例，演示如何实现 UTF-8 编码。

`严`的 Unicode 是`4E25`（`100111000100101`），根据上表，可以发现`4E25`处在第三行的范围内（`0000 0800 - 0000 FFFF`），因此`严`的 UTF-8 编码需要三个字节，即格式是`1110xxxx 10xxxxxx 10xxxxxx`。然后，从`严`的最后一个二进制位开始，依次从后向前填入格式中的`x`，多出的位补`0`。这样就得到了，`严`的 UTF-8 编码是`11100100 10111000 10100101`，转换成十六进制就是`E4B8A5`。

六、Unicode 与 UTF-8 之间的转换

通过上一节的例子，可以看到`严`的 Unicode码 是`4E25`，UTF-8 编码是`E4B8A5`，两者是不一样的。它们之间的转换可以通过程序实现。

Windows平台，有一个最简单的转化方法，就是使用内置的记事本小程序`notepad.exe`。打开文件后，点击`文件`菜单中的`另存为`命令，会跳出一个对话框，在最底部有一个`编码`的下拉条。



里面有四个选项：`ANSI`，`Unicode`，`Unicode big endian`和`UTF-8`。

1. `ANSI`是默认的编码方式。对于英文文件是`ASCII`编码，对于简体中文文件是`GB2312`编码（只针对 Windows 简体中文版，如果是繁体中文版会采用 Big5 码）。
2. `Unicode`编码这里指的是`notepad.exe`使用的 UCS-2 编码方式，即直接用两个字节存入字符的 Unicode 码，这个选项用的 little endian 格式。
3. `Unicode big endian`编码与上一个选项相对应。我在下一节会解释 little endian 和 big endian 的涵义。
4. `UTF-8`编码，也就是上一节谈到的编码方法。

选择完"编码方式"后，点击"保存"按钮，文件的编码方式就立刻转换好了。

七、Little endian 和 Big endian

上一节已经提到，UCS-2 格式可以存储 Unicode 码（码点不超过`0xFFFF`）。以汉字`严`为例，Unicode 码是`4E25`，需要用两个字节存储，一个字节是`4E`，另一个字节是`25`。存储的时候，`4E`在前，`25`在后，这就是 Big endian 方式；`25`在前，`4E`在后，这是 Little endian 方式。

这两个古怪的名称来自英国作家斯威夫特的《格列佛游记》。在该书中，小人国里爆发了内战，战争起因是人们争论，吃鸡蛋时究竟是从大头(Big-endian)敲开还是从小头(Little-endian)敲开。为了这件事情，前后爆发了六次战争，一个皇帝送了命，另一个皇帝丢了王位。

第一个字节在前，就是"大头方式"（Big endian），第二个字节在前就是"小头方式"（Little endian）。

那么很自然的，就会出现一个问题：计算机怎么知道某一个文件到底采用哪一种方式编码？

Unicode 规范定义，每一个文件的最前面分别加入一个表示编码顺序的字符，这个字符的名字叫做"零宽度非换行空格"（zero width no-break space），用`FEFF`表示。这正好是两个字节，而且`FF`比`FE`大`1`。

如果一个文本文件的头两个字节是`FE FF`，就表示该文件采用大头方式；如果头两个字节是`FF FE`，就表示该文件采用小头方式。

八、实例

下面，举一个实例。

打开"记事本"程序`notepad.exe`，新建一个文本文件，内容就是一个`严`字，依次采用`ANSI`，`Unicode`，`Unicode big endian`和`UTF-8`编码方式保存。

然后，用文本编辑软件 UltraEdit 中 的"十六进制功能"，观察该文件的内部编码方式。

1. ANSI：文件的编码就是两个字节`D1 CF`，这正是`严`的 GB2312 编码，这也暗示 GB2312 是采用大头方式存储的。
2. Unicode：编码是四个字节`FF FE 25 4E`，其中`FF FE`表明是小头方式存储，真正的编码是`4E25`。
3. Unicode big endian：编码是四个字节`FE FF 4E 25`，其中`FE FF`表明是大头方式存储。
4. UTF-8：编码是六个字节`EF BB BF E4 B8 A5`，前三个字节`EF BB BF`表示这是UTF-8编码，后三个`E4B8A5`就是`严`的具体编码，它的存储顺序与编码顺序是一致的。