字符编码详解

版本：v2.2

Crifan Li

摘要

本文主要介绍了字符编码的基础知识，以及常见的字符编码类型，比如ASCII，Unicode，UTF-8，ISO 8859等，以及各种编码之间的关系，同时专门解释了中文字符相关的编码标准，包括GB2312，GBK，GB18030，也专门解释了Windows系统中的Code Page，以及相关的BOM等内容

本文提供多种格式供：

在线阅读	HTML [1]	HTMLs [2]	PDF [3]	CHM [4]	TXT [5]	RTF [6]	WEBHELP [7]
下载（7zip压缩包）	HTML [8]	HTMLs [9]	PDF [10]	CHM [11]	TXT [12]	RTF [13]	WEBHELP [14]

HTML版本的在线地址为：

http://www.crifan.com/files/doc/docbook/char_encoding/release/html/char_encoding.html

有任何意见，建议，提交bug等，都欢迎去讨论组发帖讨论：

http://www.crifan.com/bbs/categories/char_encoding/

修订历史		
修订 1.0	2011-11-02	crl
1. 添加了编码相关背景知识介绍 2. 添加了ASCII和EASCII编码介绍 3. 添加了ISO/IEC 8859相关的编码和各种单字节编码的关系 4. 添加了Unicode和ISO 10646的解释 5. 添加了UTF-8和Unicode的区别和联系		
修订 2.2	2012-08-09	crl
1. 通过Docbook发布		

[1] http://www.crifan.com/files/doc/docbook/char_encoding/release/html/char_encoding.html
[2] http://www.crifan.com/files/doc/docbook/char_encoding/release/htmls/index.html
[3] http://www.crifan.com/files/doc/docbook/char_encoding/release/pdf/char_encoding.pdf
[4] http://www.crifan.com/files/doc/docbook/char_encoding/release/chm/char_encoding.chm
[5] http://www.crifan.com/files/doc/docbook/char_encoding/release/txt/char_encoding.txt
[6] http://www.crifan.com/files/doc/docbook/char_encoding/release/rtf/char_encoding.rtf
[7] http://www.crifan.com/files/doc/docbook/char_encoding/release/webhelp/index.html
[8] http://www.crifan.com/files/doc/docbook/char_encoding/release/html/char_encoding.html.7z
[9] http://www.crifan.com/files/doc/docbook/char_encoding/release/htmls/index.html.7z
[10] http://www.crifan.com/files/doc/docbook/char_encoding/release/pdf/char_encoding.pdf.7z
[11] http://www.crifan.com/files/doc/docbook/char_encoding/release/chm/char_encoding.chm.7z
[12] http://www.crifan.com/files/doc/docbook/char_encoding/release/txt/char_encoding.txt.7z
[13] http://www.crifan.com/files/doc/docbook/char_encoding/release/rtf/char_encoding.rtf.7z
[14] http://www.crifan.com/files/doc/docbook/char_encoding/release/webhelp/char_encoding.webhelp.7z

2. 合并了原先在zhcn_charset中的内容
3. 详细解释了Code Page
4. 详细解释了ANSI编码
5. 解释BOM

字符编码详解:

Crifan Li

版本：v2.2

出版日期 2012-08-09

[15] http://www.crifan.com/files/doc/docbook/soft_dev_basic/release/html/soft_dev_basic.html#cc_by_nc

目录

插图清单

表格清单

缩略词

ASCII (ASCII)	American Standard Code for Information Interchange 美国信息交换标准代码
BMP (BMP)	Basic Multilingual Plane 基本多文种平面
EBCDIC (EBCDIC)	Extended Binary Coded Decimal Interchange Code 扩展二进制编码十进制交换码
IANA (IANA)	Internet Assigned Numbers Authority 互联网号码分配局
ISO/IEC (ISO/IEC)	International Organization for Standardization / International Electrotechnical Commission 国际标准化组织和国际电工委员会
UCS (UCS)	Universal Character Set 通用字符集
UTF (UTF)	Unicode Transformation Format Unicode转换格式

正文之前

1. 目的

本文旨在讲清楚字符编码的概念和来龙去脉，和常见标准之间的关系和区别。

2. 本文内容

个人对于字符编码的理解，最开始主要是看了阮一峰的这篇文章：
【转】字符编码笔记：ASCII，Unicode和UTF-8 [1]

然后自己花了更多的时间，搜集整理了和字符编码的更详细的知识，整理出来，以供大家参考。

其中也摘录了该贴的部分内容。

3. 声明

任何问题，意见，建议等，都欢迎一起探讨：admin (at) crifan.com。

[1] http://www.crifan.com/switch_character_encoding_notes_ascii_unicode_and_utf-8/

第 1 章 字符编码相关的背景知识

1.1. 拉丁字母

1.1.1. 我们的目标

在介绍计算机的字符编码知识前，先来说说这个拉丁字母，估计也会有人和我一样，对于拉丁字母和英文字母以及汉语拼音中的字母的关系，不是很清楚。

拉丁字母，也叫罗马字母，是当今世界上使用最广的字母系统。

拉丁字母，或者说基本的拉丁字母，就是你所常见的到的ABCD等26个英文字母。

原先是欧洲那边使用的，后来由于欧洲殖民主义，导致后来的美洲等地，也是用的这套字母体系。

而其他有些地方，比如越南等，本来有自己的文字语言的，结果受西方文化的影响和由于基督教的传播，也用拉丁字母了。

所以总的说，现在欧洲多数国家，美洲，澳洲，非洲的多数国家，都是用的拉丁字母，即你所常见的英文字母，也是拉丁字母。而中国的汉语拼音，也是用的这个拉丁字母。

其中，欧洲很多国家，是对已有的26个基本的拉丁字母，加上连字，变音字符，弄出个衍生拉丁字母，但是还是属于拉丁字母。

说了这么多，就是要让你知道，后面内容所提到的英文字母，其来源于拉丁字母，而且我们汉语的汉语拼音，也是拉丁字母。

即：

- 基本的拉丁字母 = 26个英文字母 = 汉语中的汉语拼音
- 衍生的拉丁字母 = 从基本的26个英文字母，加上连字，变音等字符而衍生出来的拉丁字母 = 很多西欧国家的字母（每个国家都不太一样）

1.2. 什么是字符编码

计算机中存放的都是0和1的二进制值。8个位对应一个字节，常用16进制来表示。

而我们普通用户所希望看到的是，计算机把其所存储的对应的16进制的数值，转化为对应的字符，包括英文和中文等其他语言的字符，然后输出到屏幕上。

而所谓编码，就是，定义了一套规则，去指定，哪些数值，对应着哪些字符。

举个最简单的例子，常见65=0x41对应的是大写字母A，97=0x61对应的是小写字母a，而这套数值和字母之间的映射关系，说白了，就是一套规则，就叫做字符编码，即我们常说的ASCII编码。

那有人会问了，如果我定义了一套规则，假如叫张三编码，然后故意去把ASCII中的映射关系改变，比如97=0x61对应的是大写字母A，65=0x41对应的是小写字母a，等等，可不可以？答案是，完全可以，不过这套规则，首先没有得到所有计算机业界的一致认同，所以，除了你自己用，其他人不原意使用，那么也就是没了存在的价值了。

换句话说，当初ASCII之所以这么定义这套规则，就是这么定义了而已，然后大家都接受这个标准，然后就都用这个定义了。

即，如果当初定义为0x41代表的是小写字母a，而不是大写字母A，那么现在你所看到的，就是小写字母a就是对应着计算机中存储的0x41，而不是之前的0x61了。

所以，简单的说就是：

所谓字符编码，就是定义了一套规则，指定了计算机中存放的这么多值中的哪个值，对应了电脑屏幕显示出来的哪个字母。

第 2 章 字符编码标准

2.1. 只支持基本的拉丁字符的字符编码：ASCII

2.1.1. ASCII的由来

计算机刚出现的时候，虽然是美国人发明的，但是也要面对一个问题，即如何将对应的计算机中的数值，转化为对应的字母，而显示出来，即采用什么样的规则，而当时，各个厂家或公司都有自己的做法，也就是说，编码规则没有统一。

但是相对来说，得到大家认可的有，IBM的EBCDIC和此处要谈的ASCII。

其中EBCDIC现在基本没人再用，而大家统一都用ASCII了。

ASCII，即American Standard Code for Information Interchange，美国信息交换标准代码。

单独看名字，就是知道，这字符编码是设计给美国人用的。

那是因为，计算机是美国人所发明和使用的，所以计算机的早期，所设计编码标准，自然需要先为英文字符来设计和考虑，所以此最早的字符编码ASCII可以显示常见的英文字符，也可以这么说，也只能显示基本的英文字符。

由于ASCII编码中，不包括其他欧洲的很多国家的衍生的拉丁字母的那些字符，更不包含亚洲，比如中国的中文字符，因此才会有后面所提到的各种其他字符集，为的就是可以让计算机显示出自己国家的字符。

2.1.2. ASCII编码规则

ASCII的编码规则，由于最初只是为英文字母所考虑的，而英文只有26个字母，以及加上其他大小写字母，常见的字符，常见数字等，所有的加起来，也就几十个，而一个字节8位中前7位的理论上可以表示2^7=128个字符，所以对于设计出来的编码规则来说，只需要用一个字节来表示，就足够了。

即ASCII编码规则中规定，用单个字节共8位来表示字符，其中最高位为0，其他7位所对于的每一个值，映射到某个特定的字符，这样就形成了ASCII编码。

ASCII共包含了2^7=128个字符。

其中包括33个不可显示的字符和95个可显示的字符。

而对于ASCII编码规则，简单说就是：

7位的字符编码，即每个字节的最高位第8位为0，其余7位的某个值对应着某个字符。

ASCII字符集共2^7=128个字符 = 33个控制字符 + 95个可见字符。

>ASCII中的可显示的字符和不可显示字符

ASCII中可显示的字符，也叫可打印printable字符；

而ASCII中的值为0 – 31的那些字符，叫做不可显示的字符，也叫不可见字符，不可打印（non-printable）字符，由于其字符的作用是起一定的控制作用，所以常称为控制字符（control character），即不同的字符实现不同的功能，因此又称为功能字符（function code，function character）。

即ASCII字符集中：

不可见字符

=不可打印（non-printable）字符

=控制字符（control character）

=功能字符（function code，function character）

对于ASCII中的控制字符，都包括哪些，以及每个字符的详细含义，第 2.1.2.1 节 “ASCII字符集中的功能/控制字符” 中会有详细介绍。

2.1.2.1. ASCII字符集中的功能/控制字符

2.1.2.1.1. 什么是Function Code功能码或 Function Character功能字符

ASCII字符集，大家都知道吧，最基本的包含了128个字符。其中前32个，0-31，即0x00-0x1F，都是不可见字符。这些字符，就叫做控制字符。

这些字符没法打印出来，但是每个字符，都对应着一个特殊的控制功能的字符，简称功能字符或功能码Function Code。

简言之：ASCII中前32个字符，统称为Function Code功能字符。

此外，由于ASCII中的127对应的是Delete，也是不可见的，所以，此处根据笔者的理解，也可以归为Function Code。

此类字符，对应不同的“功能”，起到一定的“控制作用”，所以，称为控制字符。

关于每个控制字符的控制功能缩写，参见表 2.1 “ASCII中的控制字符”

表 2.1. ASCII中的控制字符

十进制	十六进制	控制字符	转义字符❶	说明	Ctrl + 下列字母❷
0	00	NUL	\0	Null character(空字符)	@❸
1	01	SOH		Start of Header(标题开始)	A
2	02	STX		Start of Text(正文开始)	B
3	03	ETX		End of Text(正文结束)	C
4	04	EOT		End of Transmission(传输结束)	D
5	05	ENQ		Enquiry(请求)	E
6	06	ACK		Acknowledgment(收到通知/响应)	F
7	07	BEL	\a	Bell(响铃)	G
8	08	BS	\b	Backspace(退格)	H
9	09	HT	\t	Horizontal Tab(水平制表符)	I
10	0A	LF	\n	Line feed(换行键)	J
11	0B	VT	\v	Vertical Tab(垂直制表符)	K
12	0C	FF	\f	Form feed(换页键)	L
13	0D	CR	\r	Carriage return(回车键)	M
14	0E	SO		Shift Out(不用切换)	N
15	0F	SI		Shift In(启用切换)	O
16	10	DLE		Data Link Escape(数据链路转义)	P
17	11	DC1		Device Control 1(设备控制1) / XON(Transmit On)	Q
18	12	DC2		Device Control 2(设备控制2)	R
19	13	DC3		Device Control 3(设备控制3) / XOFF(Transmit Off)	S

十进制	十六进制	控制字符	转义字符❶	说明	Ctrl + 下列字母❷
20	14	DC4		Device Control 4(设备控制4)	T
21	15	NAK		Negative Acknowledgement(拒绝接收/无响应)	U
22	16	SYN		Synchronous Idle(同步空闲)	V
23	17	ETB		End of Trans the Block(传输块结束)	W
24	18	CAN		Cancel(取消)	X
25	19	EM		End of Medium(已到介质末端/介质存储已满)	Y
26	1A	SUB		Substitute(替补/替换)	Z
27	1B	ESC	\e	Escape(溢出/逃离/取消)	[
28	1C	FS		File Separator(文件分割符)	\
29	1D	GS		Group Separator(分组符)	]
30	1E	RS		Record Separator(记录分隔符)	^❸
31	1F	US		Unit Separator(单元分隔符)	_❸
32	20	SP		White space	[Space]❹
127	7F	DEL		Delete(删除)	?❺

❶ 即在C语言中或其他地方如何表示。
❷ 可以通过 “Ctrl+对应字母/按键” 实现上述控制字符的输入

下面列举一些你可能遇到的情况：

- 用Ctrl+V输入[SYNC]
- 用Ctrl+M输入[Enter]
  当然也可以直接用Enter键，但是在Windows下面，其会发送两个字符：CR和LF

  关于CR，LF，详情参考：【详解】回车 换行 0x0D 0x0A CR LF r n的来龙去脉[1]
- 用Ctrl+Q输入XON
- 用Ctrl+S输入XOFF

❸ 注意此处想要在键盘上输入这三个字符的话，是需要通过Shift加上对应字符才能输入的：

- @：用Shift + 2输入
- ^：用Shift + 6输入
- _：用Shift + -输入

❹ 32=0x20，对应的是空格（Blank Space）键。不需要加Ctrl键，即可直接通过键盘上的空格键输入。
❺ 127=0x7F=删除（Delete）键;，除了可以用键盘上的删除键输入，也可以用'Ctrl+?'输入。

2.1.2.1.2. ASCII中的Function/Control Code功能字符的详细含义

2.1.2.1.2.1. 0 – NUL – NULl 字符/空字符

ASCII字符集中的空字符，NULL，起初本意可以看作为NOP（中文意为空操作，就是啥都不做的意思），此位置可以忽略一个字符。

[1] http://www.crifan.com/detailed_carriage_return_0x0d_0x0a_cr_lf__r__n_the_context/

之所以有这个空字符，主要是用于计算机早期的记录信息的纸带，此处留个NUL字符，意思是先占这个位置，以待后用，比如你哪天想起来了，在这个位置在放一个别的啥字符之类的。

后来呢，NUL字符被用于C语言中，字符串的终结符，当一个字符串中间出现NUL / NULL，代码里面表现为\0，的时候，就意味着这个是一个字符串的结尾了。这样就方便按照自己需求去定义字符串，多长都行，当然只要你内存放得下，然后最后加一个\0, 即空字符，意思是当前字符串到此结束。

2.1.2.1.2.2. 1 – SOH – Start Of Heading 标题开始

如果信息沟通交流主要以命令和消息的形式的话，SOH就可以用于标记每个消息的开始。

1963年，最开始ASCII标准中，把此字符定义为Start of Message，后来又改为现在的Start Of Heading。

现在，这个SOH常见于主从（master-slave）模式的RS232的通信中，一个主设备，以SOH开头，和从设备进行通信。这样方便从设备在数据传输出现错误的时候，在下一次通信之前，去实现重新同步（resynchronize）。如果没有一个清晰的类似于SOH这样的标记，去标记每个命令的起始或开头的话，那么重新同步，就很难实现了。

2.1.2.1.2.3. 2 – STX，3 – ETX

2 – STX – Start Of Text 文本开始

3 – ETX – End Of Text 文本结束

通过某种通讯协议去传输的一个数据（包），称为一帧的话，常会包含一个帧头，包含了寻址信息，即你是要发给谁，要发送到目的地是哪里，其后跟着真正要发送的数据内容。

而STX，就用于标记这个数据内容的开始。接下来是要传输的数据，最后是ETX，表明数据的结束。

其中，中间具体传输的数据内容，ASCII规范并没有去定义，其和你所用的传输协议，具体自己要传什么数据有关。

帧头		数据或文本内容		
SOH（表明帧头开始）	……（帧头信息，比如包含了目的地址，表明你发送给谁等等）	STX（表明数据开始）	……（真正要传输的数据）	ETX（表明数据结束

不过其中有趣的是，1963年，ASCII标准最初版本的时候，把现在的STX叫做EOA（End Of Address），ETX叫做（End Of Message）。

这是因为，最早的时候，一个消息中，总是包含一个开始符和一个终止符。现在的新的定义，使得可以去发送一个固定长度的命令，而只用一个SOH表明帧头开始即可，而不需要再加上一个命令终止符或帧头结束符。

总结一下：

一般发送一个消息，包含了一个帧头和后面真正要传的数据。

而对于帧头，属于控制类的信息，这部分之前属于命令，后面的真实要传的数据属于数据。即消息=帧头+数据。

而之前的命令都要有个开始符和结束符，这样就是：

消息

= 帧头 + 要传的数据

= 帧头开始+帧头信息+帧头结束 + 要传的数据

而现在新的定义，使得只需要：

消息

= 帧头 +要传的数据

= SOH（表明帧头开始）+帧头信息+ 要传的数据

= SOH（表明帧头开始）+帧头信息 + STX + 数据内容+ETX

就可以少用一个帧头结束符。

而如今，在很多协议中，也常见到，一个固定长度的帧头，后面紧接着就是数据了，而没有所谓的帧头结束符之类的东西去区分帧头和数据。

2.1.2.1.2.4. 4 – EOT – End Of Transmission 传输结束

2.1.2.1.2.5. 5 – ENQ – ENQuiry 请求

2.1.2.1.2.6. 6 – ACK – ACKnowledgment 回应/响应

2.1.2.1.2.7. 7 – BEL – [audible] BELl

在ASCII字符集中，BEL，是个比较有意思的东东。

因为其原先本意不是用来数据编码的，于此相反，ASCII中的其他字符，都是用于字符编码（即用什么字符，代表什么含义）或者起到控制设备的作用。

BEL用一个可以听得见的声音，来吸引人们的注意，其原打算即用于计算机也用于一些设备，比如打印机等。

C语言里面也支持此BEL，用a来实现这个响铃。

2.1.2.1.2.8. 8 – BS – BackSpace 退格键

退格键的功能，随着时间变化，意义也变得不同了。

起初，意思是，在打印机和电传打字机上，往回移动一格光标，以起到强调该字符的作用。

比如你想要打印一个a，然后加上退格键后，就成了aBS^。在机械类打字机上，此方法能够起到实际的强调字符的作用，但是对于后来的CTR下时期来说，就无法起到对应效果了。

而现代所用的退格键，不仅仅表示光标往回移动了一格，同时也删除了移动后该位置的字符。在C语言中，退格键可以用b表示。

2.1.2.1.2.9. 9 – HT – Horizontal Tab 水平制表符

ASCII中的HT控制符的作用是用于布局的。

其控制输出设备前进到下一个表格去处理。

而制表符Table/Tab的宽度也是灵活不固定的，只不过，多数设备上，制表符Tab的宽度都预定义为8。

水平制表符HT不仅能减少数据输入者的工作量，对于格式化好的文字来说，还能够减少存储空间，因为一个Tab键，就代替了8个空格，所以说省空间。

对于省空间的优点，我们现在来看，可能会觉得可笑，因为现在存储空间已足够大，一般来说根本不会需要去省那么点可怜的存储空间。

但是，实际上在计算机刚发明的时候，存储空间（主要指的是内存）极其有限也极其昂贵，而且像ZIP等压缩方法也还没发明呢，所以对于当时来说，对于存储空间，那是能够省一点是一点，省任何一点，都是好的，也都是不容易的，省空间就是省钱啊。

C语言中，用t表示制表符。

2.1.2.1.2.10. 10 – LF – Line Feed 换行

LF，直译为（给打印机等）喂一行，意思就是所说的，换行。

换行字符，是ASCII字符集中，被误用的字符中的其中一个。

LF的最原始的含义是，移动打印机的头到下一行。而另外一个ASCII字符，CR（Carriage Return）才是将打印机的头，移到最左边即一行的开始，行首。很多串口协议和MS-DOS及Windows操作系统，也都是这么实现的。

而于此不同，对于C语言和Unix操作系统，其重新定义了LF字符的含义为新行，即LF和CR的组合才能表达出的，回车且换行的意思。

虽然你可以争论哪种用法是错的，但是，不可否认，是从程序的角度出发，C语言和Unix对此LF的含义实现显得就很自然，而MS-DOS的实现更接近于LF的本意。

如果最开始ASCII标准中，及定义 CF也定义newline，那样意思会清楚，会更好理理解：

LF表示物理上的，设备控制方面的移动到下一行（并没有移动到行首）；

新行（newline）表示逻辑上文本分隔符，即回车换行。

不过呢，现在人们常将LF用做newline新行的功能，而大多数文本编辑软件也都可以处理单个LF或者CR/LF的组合了。

LF在C语言中，用n表示。

2.1.2.1.2.11. 11 – VT – Vertical Tab 垂直制表符

垂直制表符，类似于水平制表符Tab，目的是为了减少布局中的工作，同时也减少了格式化字符时所需要存储字符的空间。VT控制码用于跳到下一个标记行。

说实话，还真没看到有些地方需要用这个VT呢，因为一般在换行的时候，都是用LF代替VT了。

2.1.2.1.2.12. 12 – FF – Form Feed 换页

设计换页键，是用来控制打印机行为的。

当打印机收到此键码的时候，打印机移动到下一页。

不同的设备的终端对此控制码所表现的行为各不同。有些会去清除屏幕，而其他有的只是显示^L字符或者是只是新换一行而已。

Shell脚本程序Bash和Tcsh的实现方式是，把FF看作是一个清除屏幕的命令。C语言程序中用f表示FF（换页）。

2.1.2.1.2.13. 13 – CR – Carriage return 机器的滑动部分/底座 返回 -> 回车

CR回车的原意是让打印头回到左边界，并没有移动到下一行。

随着时间流逝，后来人把CR的意思弄成了Enter键，用于示意输入完毕。

在数据以屏幕显示的情况下，人们在Enter的同时，也希望把光标移动到下一行。

因此C语言和Unix操作系统，重新定义了LF的意思，使其表示为移动到下一行。当输入CR去存储数据的时候，软件也常常隐式地将其转换为LF。

2.1.2.1.2.14. 14 – SO，15 – SI

14 – SO – Shift Out 不用切换

15 – SI – Shift In 启用切换

早在1960s年代，定义ASCII字符集的人，就已经懂得了，设计字符集不单单可以用于英文字符集，也要能应用于外文字符集，是很重要的。

定义Shift In 和Shift Out的含义，即考虑到了此点。

最开始，其意为在西里尔语和拉丁语之间切换。

西里尔ASCII定义中，KOI-7用到了Shift字符。拉丁语用Shift去改变打印机的字体。

在此种用途中，SO用于产生双倍宽度的字符，而用SI打印压缩的字体。

2.1.2.1.2.15. 16 – DLE – Data Link Escape 数据链路转义

有时候，我们需要在正在进行的通信过程中去发送一些控制字符。但是，总有一些情况下，这些控制字符却被看成了普通的数据流，而没有起到对应的控制效果。而ASCII标准中，定义DLE来解决这类问题。

如果数据流中检测到了DLE，数据接收端则对其后面接下来的数据流中的字符，另作处理。

而关于具体如何处理这些字符，ASCII规范中则没有具体定义，而只是弄了个DLE去打断正常数据的处理，告诉接下来的数据，要特殊对待。

根据Modem中的Hayes通信协议DLE定义为“无声+++无声”。

以我的观点，这样可能会更好：如果Hayes协议没有把DLE处理为嵌入通讯的无声状态，那样就符合现存的标准了。

然而Hayes的开发者却觉得+++用的频率要远高于原始的DLE，所以才这么定义了。

2.1.2.1.2.16. 17 – DC1 – Device Control 1 / XON – Transmission on

这个ASCII控制字符尽管原先定义为DC1， 但是现在常表示为XON，用于串行通信中的软件流控制。

其主要作用为，在通信被控制码XOFF中断之后，重新开始信息传输。

用过串行终端的人应该还记得，当有时候数据出错了，按Ctrl+Q（等价于XON）有时候可以起到重新传输的效果。

这是因为，此Ctrl+Q键盘序列实际上就是产生XON控制码，其可以将那些由于终端或者主机方面，由于偶尔出现的错误的XOFF控制码而中断的通信解锁，使其正常通信。

2.1.2.1.2.17. 18 – DC2 – Device Control 2

2.1.2.1.2.18. 19 – DC3 – Device Control 3 / XOFF – Transmission off 传输中断

2.1.2.1.2.19. 20 – DC4 – Device Control 4

2.1.2.1.2.20. 21 – NAK – Negative AcKnowledgment 负面响应-> 无响应, 非正常响应

2.1.2.1.2.21. 22 – SYN – SYNchronous idle

2.1.2.1.2.22. 23 – ETB – End of Transmission Block 块传输中止

2.1.2.1.2.23. 24 – CAN – CANcel 取消

2.1.2.1.2.24. 25 – EM – End of Medium 已到介质末端，介质存储已满

EM用于，当数据存储到达串行存储介质末尾的时候，就像磁带或磁头滚动到介质末尾一样。其用于表述数据的逻辑终点，即不必非要是物理上的达到数据载体的末尾。

2.1.2.1.2.25. 26 – SUB – SUBstitute character替补/替换

2.1.2.1.2.26. 27 – ESC – ESCape 逃离/取消

字符Escape，是ASCII标准的首创的，由Bob Bemer提议的。用于开始一段控制码的扩展字符。如此，即可以不必将所有可能想得到的字符都放到ASCII标准中了。

因为，新的技术可能需要新的控制命令，而ESC可以用作这些字符命令的起始标志。

ESC广泛用于打印机和终端，去控制设备设置，比如字体，字符位置和颜色等等。

如果最开始的ASCII标准中，没有定义ESC，估计ASCII标准早就被其他标准所替代了，因为其没有包含这些新出现的字符，所以肯定会有其他新的标准出现，用于表示这些字符的。

即，ESC给开发者提供了，可以根据需要而定义新含义的字符的可能。

2.1.2.1.2.27. 28 – FS – File Separator 文件分隔符

文件分隔符是个很有意思的控制字符，因为其可以让我们看到1960s年代的时候，计算机技术是如何组织的。

我们现在，习惯于随即访问一些存储介质，比如RAM，磁盘，但是在定义ASCII标准的那个年代，大部分数据还是顺序的，串行的，而不是随机访问的。此处所说的串行的，不仅仅指的是串行通信，还指的是顺序存储介质，比如穿孔卡片，纸带，磁带等。

在串行通信的时代，设计这么一个用于表示文件分隔符的控制字符，用于分割两个单独的文件，是一件很明智的事情。而FS的原因就在于此。

2.1.2.1.2.28. 29 – GS – Group Separator分组符

ASCII定义控制字符的原因中，其中一条就是考虑到了数据存储方面的情况。

大部分情况下，数据库的建立，都和表有关，包含了对应的记录。同一个表中的所有的记录，属于同一类型。不同的表中的记录，属于对应的不同的类型。

而分组符GS就是用来分隔串行数据存储系统中的不同的组。值得注意的是，当时还没有使用word的表格，当时ASCII时代的人，把他叫做组。

2.1.2.1.2.29. 30 – RS – Record Separator记录分隔符

记录分隔符RS用于分隔在一个组或表内的多个记录。

2.1.2.1.2.30. 31 – US – Unit Separator 单元分隔符

在ASCII定义中，在数据库中所存储的，最小的数据项，叫做Unit单元。而现在我们称其field域。单元分隔符US用于分割串行数据存储环境下的不同的域。

现在大部分的数据库实现，要求大部分类型都拥有固定的长度。

尽管大部分时候可能用不到，但是对于每一个域，却都要分配足够大的空间，用于存放最大可能的成员变量。

这样的做法，占用了大量的存储空间，而US控制码允许域具有可变的长度。在1960s年代，数据存储空间很有限，用US这个单元分隔符，将不同单元分隔开，这样就可以实现更高效地存储那些宝贵的数据。

另一方面，串行存储的存储效率，远低于RAM和磁盘中所实现的表格存储。我个人无法想象，如果现在的数据，还是存储在自带或者带滚轮的磁带上，会是何种景象。

2.1.2.1.2.31. 32 – SP – White SPace 空格键

也许你会争论说，空格键是否真的能算是一个控制字符？因为现在在普通文字中使用空格键是如此常见。

但是，既然水平制表符和退格键在ASCII中，都被叫做控制字符了，那么我觉得也很自然地，可以把空格键（向前的空格）也叫做控制字符，毕竟，其本身并不代表一个真正的可见的字符，而仅仅只是很常用于输出设备，用于处理位置前向移动一格，清除当前位置的内容而已。

在很多程序中，比如字符处理程序，白空格同样可能从导致行尾转到下一行行首，而网络浏览器将多个空格组合成单个空格输出。

所以，这更加坚定了我的想法，觉得完全可以把空格看成是一个控制字符，而不仅仅是一个很独特的普通字符。

2.1.2.1.2.32. 127 – DEL – DELete 删除

有人也许会问，为何ASCII字符集中的控制字符的值都是很小的，即0-32，而DEL控制字符的值却很大，是127。

这是由于这个特殊的字符是为纸带而定义的。而在那个时候，绝大多数的纸带，都是用7个孔洞去编码数据的。

而127这个值所对应的二进制值为111 1111b，表示所有7个比特位都是高，所以，将DEL用在现存的纸带上时，所有的洞就都被穿孔了，就把已经存在的数据都擦出掉了，就起到了对应的删除的作用了。

2.1.2.1.3. 各种字符的标准的读法/叫法

常见ASCII字符，以及其他非常见的字符，Unicode中的字符，其他特殊字符等等，这些字符的英文叫法，可以去Unicode官方找到：

http://www.unicode.org/charts/#symbols

比如：

ASCII字符/字母的叫法/读法 如何读 ：

1. C0 Control and Basic Latin Range：0000-007F [2]
2. Alphabetic Presentation Forms Range：FB00-FB4F [3]
3. CJK Compatibility Forms [4]
4. Fullwidth ASCII Punctuation [5]

[2] http://www.unicode.org/charts/PDF/U0000.pdf
[3] http://www.unicode.org/charts/PDF/UFB00.pdf
[4] http://www.unicode.org/charts/PDF/UFE30.pdf
[5] http://www.unicode.org/charts/PDF/UFF00.pdf

2.1.3. ISO 646

ASCII的字符编码是美国自己定义的标准，而其对应的国际标准叫做ISO/IEC 646。

ISO/IEC是参考了多个国家的字符编码标准，其中主要是美国ASCII标准，然后制定出来的7位的国际字符编码标准。

所以，此处，可以简单看成美国的国家标准ASCII和国际标准ISO/IEC 646，两者是是等价的，即：

美国的国家的字符编码标准ASCII

=国际的字符编码标准ISO/IEC 646

2.2. 支持多种衍生拉丁字母的字符编码：EASCII和ISO 8859

计算机出现之后，从美国发展到欧洲，而由于欧洲很多国家中所用到的字符中，除了基本的美国也用的那些拉丁字母之外，还有很多衍生的拉丁字母，而且是不同的国家用到的衍生字符都不太相同，所以欧洲人也遇到类似的问题，即如何将自己国家的那些字符，在计算机上显示出来，这就需要设计一个合理的字符编码，把所有这些字符都囊括其中。

即设计一个新编码标准，即兼容旧的ASCII的编码，又支持欧洲多个国家的那些衍生拉丁字母。

这样的标准有两个，一个是EASCII编码标准，一个是国际标准ISO 8859字符编码标准。

2.2.1. EASCII

将ASCII中的第八位也用上，那么就是8位的字符编码了，然后将EASCII中0xA0-0xFF这部分比ASCII码扩充出来的编码，用来表示表格符号、计算符号、希腊字母和特殊的拉丁符号等，这样就可以实现支持那么多欧洲的衍生拉丁字母了，也就是这个EASCII字符编码了。但是EASCII虽然解决了这些西欧语言的字符显示问题，但是对于其他语言显示，比如中文等，还是无法处理显示。

目前，很少使用EASCII，常用的是下面要介绍的第 2.2.2 节 “ISO 8859”编码标准。

2.2.2. ISO 8859

2.2.2.1. ISO/IEC 8859出现的背景

前面已经提到了，正是因为ASCII等字符编码中，没有包括欧洲很多国家所用到的一些扩展的拉丁字母，比如一些重音字母，带音标的等等，所以，才设计出新的这个ISO/IEC 8859来支持这些字符。

最终设计出来的ISO/IEC 8859的字符集，支持很多欧洲的语言，包括丹麦语、荷兰语、德语、意大利语、拉丁语、挪威语、葡萄牙语、西班牙语，瑞典语等。

2.2.2.2. ISO/IEC 8859的编码规则

我们已经知道了，ASCII是7位的单字节编码，其中0x20-0x7E的可见字符。

而ISO/IEC 8859，是在ASCII中的普通的可见字符(0x20-0x7E)的基础上，利用了ASCII的7位编码所没有用到的第8位，这样就编码范围就从原先ASCII的0x00-0x7F多扩展出了0x80-0xFF，其中的0xA0-0xFF部分，被ISO/IEC 8859编码所用到。

有别于ASCII的单个独立的编码规则，ISO/IEC 8859是一组编码规则的总称，其下包含了共15个字符集，即ISO/IEC 8859-n,其中n=1,...,11,13,...,16。

关于这15种字符集是如何分类的，可以参考：表 A.2 “ISO/IEC 8859编码标准中的15种字符集”

这15个字符集，每一个字符集，编码取值都是0xA0-0xFF，但是对于同一个值，不同字符集所对应的字符，都不太一样。

此处截取那15个字符集的其中一部分，以便更加直观的了解不同字符集的区别：

图 2.1. ISO/IEC 8859的15个字符集的部分比较

ISO/IEC 8859十五个字符集的比较

Bin	Oct	Dec	Hex	1	2	3	4	5	6	7	8	9	10	11	13	14	15	16
10100000	240	160	A0	NBSP														
10100001	241	161	A1	¡	Ą	Ħ	Ą	Ë		‘		¡	Ą	ก	”	Ḃ	¡	Ą
10100010	242	162	A2	¢	˘	˘	ĸ	Ъ		’	¢	¢	Ē	ข	¢	ḃ	¢	ą
10100011	243	163	A3	£	Ł	£	Ŗ	Ґ		£	£	£	Ģ	ฃ	£	£	£	Ł
10100100	244	164	A4	¤	¤	¤	¤	Є	¤	€	¤	¤	Ī	ค	¤	Ċ	€	€
10100101	245	165	A5	¥	Ľ		Ĩ	S		₯	¥	¥	Ĩ	ฅ	„	ċ	¥	„
10100110	246	166	A6	¦	Ś	Ĥ	Ļ	I		¦	¦	¦	Ķ	ฃ	¦	Ḋ	Š	Š
10100111	247	167	A7	§	§	§	§	Ї		§	§	§	§	ง	§	§	§	§
10101000	250	168	A8	¨	¨	¨	¨	J		¨	¨	¨	Ļ	จ	Ø	Ẁ	š	š

完整的字符表，请参见表 A.2 “ISO/IEC 8859编码标准中的15种字符集”

ASCII编码有时候也会写成ISO/IEC 8859-1编码

另外，需要注意的是，对于原先的美国的英文字母，即普通的英语，其虽然没有重音，音标等字母，但是由于其本身也还是包含在ASCII中，而ISO/IEC 8859-1包括了ASCII,所以，很多时候，对于英文字母来说，也仍会标明为ISO/IEC 8859-1编码。

2.2.2.3. ISO/IEC 8859的特点

对于ISO/IEC 8859所包含的全部字符，我们可以看到，对于基本的拉丁字母，那都是和ASCII一样的，因为其就是借用了ASCII中的0x20-0x7F这段的编码，对应的是那些常见的可显示的字符，而对于0xA0-0xFF这段空间，则是对于同一个值，不同的字符集中，对应着不同的符号。

对于ISO/IEC 8859的编码方式是设计了多个字符集，我们不难看出，其之所以这么编码，而不是像ASCII中每个编码值，都对应唯一的一个字符，那是因为，欧洲的全部所用的字符数很多，如果是对于全部的欧洲用的字符都用一个对应的值来表示，那么这剩下的0xA0-0xFF，甚至是0x80-0xFF，也都不够用的，因为0x80-0xFF128个值，当然不够表示欧洲那几百上千的不同国家的不同字符。

所以，才会去设计出这么15个字符集，然后对于同一个值，你用了ISO/IEC 8859-n,就表示对应的字符集中的那个特定的字符。

而上述做法的好处是，可以避免去用多个字节，比如两个字节（8×2=16位，可以表示最多2^16=65536个字符）去表示一个单独的字符，即节省了存放数据的空间。

但是缺点是，比如你写一篇文章，中间出现了多个不同语系的不同的字符，那么此文章如果用ISO/IEC 8859来编码的话，那么就无法单独存成某一种对应的字符集，即包含多个欧洲国家不同语系的特殊字

符的数据，无法用ISO/IEC 8859的某一个单独的字符集来表示出来，即无法在同一个文章中支持显示不同语系的不同的字符。

当然，相对于亚洲字符，即中文，日文，韩文等字符来说，另外一个如果算的上是缺点的话，那就是没有把咱亚洲字符考虑进去。

正因此，字符编码，才会继续演化出更加通用的，包含了世界上所有的字符的字符编码标准：Unicode。

关于Unicode的详细解释请去看：第 2.4 节 “支持世界上几乎所有字符的字符编码：Unicode”

此处先来说说，其他几个和ISO/IEC 8859相关的内容。

2.2.2.4. ISO/IEC 6429

可以看到，对应的ASCII编码取值范围是0x0-0x7F，而ISO/IEC 8859虽然是8位的单字节的编码，但是只是除了ASCII的0x20-0x7E之外，只是指定了0xA0-0xFF，而中间还有一段的值，即0x80-0x9F，却没有定义。

对此部分，是对应的ISO/IEC 6429编码标准所定义的，此标准还定义了0x0-0x1F，即ASCII中的控制字符。

即，ISO/IEC 6429是专门定义对应的控制字符的，其中，0x0-0x1F部分称为C0控制（C0 control）字符，0x80-0x9F部分称为C1控制（C1 control）字符。

对应的C0 control部分，和ASCII编码重复定义了，但是两者含义都是一样的，所以编码规则并不冲突。

可以算是，在控制字符领域，ISO/IEC 6429在ASCII的C0 control的基础上，对于由ASCII的7位所扩展出的8位编码中的0x80-0x9F这部分，也做出了对应的定义。

简言之：

ISO/IEC 6429

= 0x0-0x1F + 0x80-0x9F

=7位编码ASCII中的0x0-0x1F + 扩展8位编码中的0x80-0x9F

=C0 control + C1 control

2.2.2.5. ISO 8859和ISO-8859的区别和联系

ISO 8859和ISO-8859，不是同一个东西。

注意，后者是ISO和8859中间带了一个小横短线的。

前者，ISO 8859是ISO/IEC 8859标准集合的简称，对应包含了 ISO/IEC 8859-n,其中n为除去2之外的1到16，这共15种字符集合。

ISO-8859，是ISO-8859-n的简称，是IANA根据ISO/IEC 8859-n的标准，加上对应的前面提到的普通的ASCII字符，和ISO/IEC 6429所定义的的控制字符，所制定的标准。

其中，由于ASCII中也包含了0x00-0x1F的控制字符，所以和ISO/IEC 6429中的C0控制字符重复了，但是两者定义都是一样的，所以从字符编码上来说，不会产生任何冲突。

因此，ISO-8859-n所以可以表示：

ISO-8859

= ISO-8859-n的简称

= ISO 8859-n + ASCII + ISO/IEC 6429

其中，n=1,...,11,13,...,16，共15种编码集合。

对应的，ISO 8859-1和ISO-8859-1两者当前也是不一样的，其区别也是：

ISO-8859-1

= ISO 8859-1 + ASCII + ISO/IEC 6429

= ISO/IEC 8859-1 + ASCII + ISO/IEC 6429

2.2.2.5.1. 原先的ISO 8859-1和我们常说的ISO 8859-1

原先的ISO 8859-1即ISO/IEC 8859-1，其编码前面已经介绍过了，此处只是给出对应的字符表：

图 2.2. ISO/IEC 8859-1字符集表

ISO/IEC 8859-1															
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x																
1x																
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8x																
9x																
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY	®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

在上表中，0x20是空格、0xA0是不换行空格、0xAD是选择性连接号。

0x00-0x1F、0x7F、0x80-0x9F在此字符集中未有定义。（控制字符是由ISO/IEC 6429定义）。

其中绿色的部分，就是原先ISO 8859-1中未定义的部分。而这部分，之前已经解释了，是在另外的一个标准ISO/IEC 6429中定义的。

此处，需要注意的是，目前大家最常见的，提到最多的ISO 8859-1，实际多数都是指的是ISO-8859-1，即，是那个，既整合了0x20-0x1F这部分的普通的可以显示的ASCII字母（基本的拉丁字母），又包含了对应的控制字符（ C0 control和C1 control），同时也包含了欧洲多数国家所用到那些字符（扩展的拉丁字母，即ISO/IEC 8859-1中所定义的那些字符）。

总结起来就是：

常说的ISO 8859-1

= 实际上是ISO-8859-1

= ASCII + ISO/IEC 6429 + ISO 8859-1

= ASCII + ISO/IEC 6429 + ISO/IEC 8859-1

= (0x20-0x1F) + (0x0-0x1F + 0x80-0x9F) + (0xA0-0xFF)

= ASCII中的可见字符 + C0和C1的控制字符 + 欧洲多国所用的扩展的拉丁字符

2.3. 各种单字节编码标准的关系

不论是ASCII的7位的编码，还是后期演化出来的ISO/IEC 8859的8位的编码，都还是用单个字节就可以表示一个字符，叫做单字节编码。

各种单字节编码之间的关系，可以用下面图表来解释：

表 2.2. 各种单字节编码标准之间的关系

单字节编码		单个字节=8位=2^8=256个字符				
编码标准	注释	用到了前7位=2^7=128个字符			用到了第8位	
		0x0-0x1F	0x20-0x7E	0x7F	0x80-0x9F	0xA0-0xFF
ASCII	=ISO/IEC 646	yes	yes	yes		
ISO/ IEC 6429	=C0 control + C1 control	yes			yes	
ISO 8859	=ISO/IEC 8859-n = ISO/IEC 8859-1 ..... ISO/IEC 8859-11 ISO/IEC 8859-13 ..... ISO/IEC 8859-16		yes			yes yes yes yes yes yes yes yes yes yes yes yes yes yes yes
ISO-8859	=ISO-8859-n = ISO-8859-1 ..... ISO-8859-11 ISO-8859-13	yes	yes	yes	yes	yes yes yes yes

单字节编码		单个字节=8位=2^8=256个字符				
编码标准	注释	用到了前7位=2^7=128个字符			用到了第8位	
		0x0-0x1F	0x20❻-0x7E	0x7F	0x80-0x9F	0xA0-0xFF
…… ISO-8859-16						yes
						yes
						yes
						yes
						yes
						yes
						yes
						yes
						yes
						yes
						yes

❻ 0x20是空格Space，常缩写为SP。

此空格字符，严格意义上说，属于不可显示字符，因为显示或打印出来，也看不见

2.4. 支持世界上几乎所有字符的字符编码：Unicode

好了，介绍完了ISO/IEC 8859的种种，这下可以开始介绍Unicode了。

前面已经提到了，由于随着计算机的发展，自然会发展到亚洲各国和其他一些地方，然后这些国家也遇到同样问题，即如何把自己的国家的字符，显示到对应的屏幕上。

2.4.1. Unicode和ISO 10646的关系

Unicode这个词的中文翻译，有译为万国码，单一码，标准万国码，但是最常见的翻译还是统一码。

2.4.1.1. ISO 10646=UCS

国际标准组织ISO，定义了对应的编码标准ISO/IEC 10646，简称为ISO 10646，此标准所定义的字符集，称作为通用字符集（Universal Character Set，UCS）。

并不是所有的系统都需要支持像組合字符这样的的先进机制。

因此ISO 10646指定了如下三种实现级别:

1. 级别1：不支持组合字符和諺文字母字符。
2. 级别2：类似于级别1，但在某些文字中，允许一列固定的组合字符，因为如果没有最起码的几个组合字符，UCS就不能完整地表达这些语言。
3. 级别3：支持所有的通用字符集字符，如，可以在任意一个字符上加上一个箭头或一个鼻音化符號

即，对于多数的实际使用中，并不一样要求你实现包括世界上所有的字符，那就不一定非的要实现对应的第三级别，很多时候只需要实现第一级别就足够涵盖平常所用到的大部分的字符了。

我们平时会看到UCS-2，UCS-4，就是对应的ISO 10646标准中所定义的，对应的用2个字节或4个字节去表示同一个字符。

2.4.1.2. Unicode 和ISO 10646的联系

历史上存在两个独立的尝试创立单一字符集的组织，即国际标准化组织（ISO）和多语言软件制造商组成的统一码联盟。

前者开发的 ISO/IEC 10646 项目，后者开发的Unicode项目。

因此最初制定了不同的标准。

1991年前后，两个项目的参与者都认识到，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Unicode 2.0开始，Unicode采用了与ISO 10646-1相同的字库和字码；ISO也承诺，ISO 10646将不会超出U+10FFFF的UCS-4编码赋值，以使得两者保持一致。

两个项目仍都存在，并独立地公布各自的标准，但统一码联盟和ISO/IEC JTC1/SC2都同意保持两者标准的码表兼容，并紧密配合以保证之后的扩展也一致。

其各自的标准之间的对应关系如下：

表 2.3. ISO/IEC 10646与Unicode的版本对应关系

ISO/IEC 10646版本	Unicode版本
ISO/IEC 10646-1:1993	Unicode 1.1
ISO/IEC 10646-1:2000	Unicode 3.0
ISO/IEC 10646-2:2001	Unicode 3.2
ISO/IEC 10646:2003	Unicode 4.0
ISO/IEC 10646:2003 plus Amendment 1	Unicode 4.1
ISO/IEC 10646:2003 plus Amendment 1, Amendment 2, and part of Amendment 3	Unicode 5.0
ISO/IEC 10646:2003 plus Amendments 1 to 4	Unicode 5.1
ISO/IEC 10646:2003 plus Amendments 1 to 6	Unicode 5.2
ISO/IEC 10646:2011	Unicode 6.0

2.4.1.3. Unicode和ISO 10646的区别

统一码联盟公布的Unicode标准包含了ISO/IEC 10646-1实现级别3的基本多文种平面BMP。在两个标准里，所有的字符都在相同的位置并且有相同的名字。

ISO/IEC 10646标准，就像ISO/IEC 8859标准一样，只不过是一个简单的字符集表。它定义了一些编码的别名，指定了一些与标准有关的术语，并包括了规范说明，指定了怎样使用UCS连接其他ISO标准的实现，比如ISO/IEC 6429和ISO/IEC 2022。还有一些与ISO紧密相关的，比如ISO/IEC 14651是关于UCS字符串排序的。

Unicode标准，额外定义了许多与字符有关的语义符号学。Unicode详细说明了绘制某些语言（如阿拉伯语）表达形式的算法，处理双向文字（比如拉丁文和希伯来文的混合文字）的算法，排序与字符串比较所需的算法，等等。

所以，可以理解为，ISO 10646中定义了编码规则，定义了哪些值对应了哪些字符，而Unicode不仅定义了这些编码规则，还定义了其他一些关于文字处理的细节算法等内容。

即：

Unicode

= ISO 10646的编码规则 + 某些语言的细节处理算法

对于一般人来说，Unicode 和 ISO 10646，虽然两者有些细节的区别，但是我们多数不用去关系这点细节内容，而对于字符编码规则方面，此处可以简单的理解为：

Unicode

= ISO 10646编码标准

= 标准所制定的UCS字符集

2.4.2. Unicode编码规则

为了将世界上几乎所有的字符都涵盖了，那么就要了解世界上，有哪些字符。

除了之前ASCII的拉丁字母，ISO 8859所包含的欧洲多国用的字符之外，亚洲一些国家，包括中文，日文，韩文等，尤其是中文，包含的字符数，大概有几万个。

因此，Unicode的编码就要设计的把这么多的字符都包含在内。

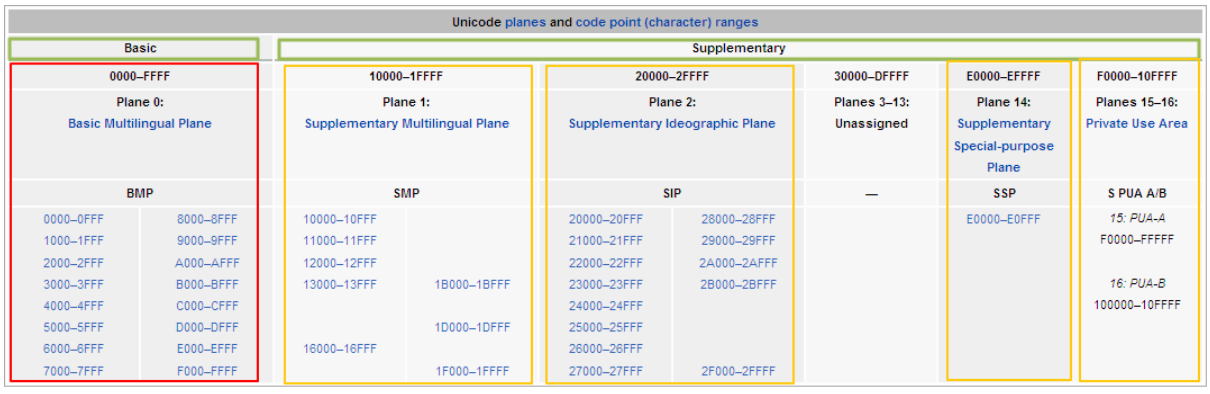
Unicode的编码方式与上面提到的ISO 10646的UCS概念相对应，目前实际应用的Unicode版本对应于UCS-2，即2字节的UCS字符集，使用16位的编码空间。每个字符占用2个字节，这样理论上一共最多可以表示2^16=65536个字符。基本满足各种语言的使用。实际上目前版本的Unicode尚未填充满这16位编码，保留了大量空间作为特殊使用或将来扩展。

上述16位Unicode字符构成基本多文种平面（Basic Multilingual Plane，简称BMP）。最新（但未实际广泛使用）的Unicode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2^31=2147483648=21亿左右个字符，完全可以涵盖一切语言所用的符号。

具体的取值范围和所对应的平面空间划分，参见图 2.3 “Unicode中的各种平面划分”

图 2.3. Unicode中的各种平面划分



Unicode中的0-0xFFF的BMP中的任何一个编码的值，称为码点（Code Point），对应用U+hhhh来表示，其中每个h 代表一个十六进制数位。

与UCS-2编码完全相同。对应的4字节UCS-4编码后两个字节一致，前两个字节的所有位均为0。

2.4.3. Unicode字符编码所对应的存储和交换标准：UTF-8, UTF-16, UTF-32

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Unicode中的字符，而不是分别表示三个ASCII的 字符呢？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

2.4.3.1. 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和UTF-32，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：

- 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

- 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。

表 2.4. Unicode与UTF-8之间的编码映射关系

Unicode符号范围(十六进制)	UTF-8编码方式（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

下面，还是以汉字“严”为例，演示如何实现UTF-8编码。

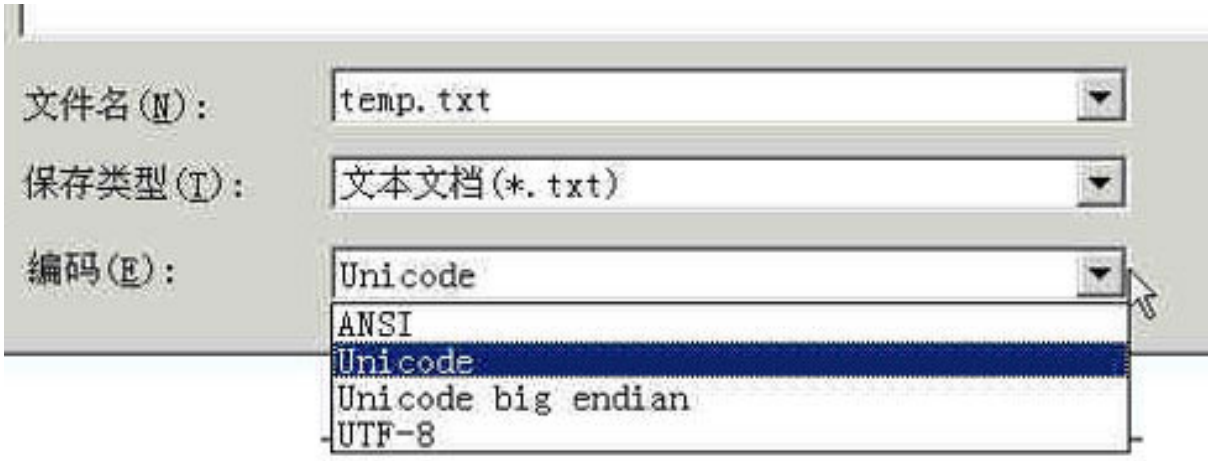
已知“严”的Unicode是4E25（100111000100101），根据上表，可以发现4E25处在第三行的范围内（0000 0800-0000 FFFF），因此“严”的UTF-8编码需要三个字节，即格式是“1110xxxx 10xxxxxx 10xxxxxx”。然后，从“严”的最后一个二进制位开始，依次从后向前填入格式中的x，多出的位补0。这样就得到了，“严”的UTF-8编码是“11100100 10111000 10100101”，转换成十六进制就是E4B8A5。

2.4.3.2. 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。

在Windows平台下，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"→"另存为"会跳出一个对话框，在最底部有一个"编码"的下拉条:

图 2.4. Notepad中的各种编码



里面有四个选项：ANSI，Unicode，Unicode big Endian 和 UTF-8:

- 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）。
- Unicode编码指的是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
- Unicode Big Endian编码与上一个选项相对应
  关于Little Endian和Big Endian，可以参考大端(Big Endian)与小端(Little Endian)详解[6]
- 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。

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的”十六进制功能“，观察该文件的内部编码方式。

- 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

- 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。

- 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。

- 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

[6] http://www.crifan.com/big_endian_big_endian_and_small_end_little_endian_detailed/

2.4.3.2.1. 关于UTF-8的BOM：“EF BB BF”

对于UTF-8的BOM（Byte Order Mark），即“EF BB BF”，是对于UTF-8编码，微软自己添加的，由此，会导致和其他很多软件等不兼容。而Unicode标准中，也不推荐此给UTF-8添加“EF BB BF”的BOM。

刚去测试了一下，在Window XP中，将中文汉字“严”在记事本中另存为UTF-8之后，用Notepad++去查看其十六进制的值，的确是“EF BB BF E4 B8 A5”，然后手动删除了“EF BB BF”的BOM，保存后，再去用记事本打开，发现没了BOM的UTF-8，记事本也是可以正确显示出“严”字的。所以，结论是：

1. 给UTF-8加“EF BB BF”的BOM，是微软自己的做法，即微软发现编码是UTF-8的话，会给文件最开始加上“EF BB BF”
2. Unicode的官方标准，不推荐这种做法，即不推荐给UTF-8加“EF BB BF”的BOM
3. 所以，其他人写软件处理文字编码的话，最好不要给UTF-8加BOM。当然，如果你非得要兼容微软的做法，那么去解析不同编码的文件的话，针对UTF-8编码，就要考虑这个特殊的的BOM了

更多关于BOM的解释，参见第 2.7 节 “BOM”

2.5. 代码页Code Page

2.5.1. 什么是代码页（Code Page）

Code Page，是字符编码的另一种说法。

Code Page包含了一个表，表中的值，用于表示针对某种语言所用的字符集。

更简单点说，就是Code Page中，用一个数字编号，表示了所要采用何种字符编码，去编解码相应的值，用于正确显示出相应的字符。

Code Page这一概念，源于IBM，后被其他常见广泛采用，包括Microsoft，SAP，Oracle等。

这些常见，各自定义了一套自己的Code Page，给每一个code page号，指定一个字符编码。

比如，对于众所周知的UTF-8编码，在IBM的Code Page中编号是1208，在微软中是65001，在SAP中是4110。

接下来，主要介绍大家最常见的Windows的Code Page

2.5.2. Windows中的Code Page

如前所述，Windows中也有自己的一套Code Page的定义。

用对应的某个数字，Code Page Number，即Code Page中的标识符（Identifier），表示相应的字符编码。

而一般Code Page也常缩写为CP

比如，CP936表示GBK中文编码，CP65001表示UTF-8编码，CP54936表示GB18030编码，CP950表示BIG5繁体中文等等。

C#中使用当前系统默认编码处理字符

对于C#来说，处理字符时涉及可能在不同环境中使用的话，那么最好用系统默认编码：

```
StreamReader reader = new StreamReader(path, System.Text.Encoding.Default);
```

2.5.2.1. Windows中的Code Page分类：ANSI和OEM

Windows中的Code Page，按照引用领域来划分，可以分为两类：ANSI Code Page和 OEM Code Page

2.5.2.1.1. Windows的ANSI Code Page表

ANSI Code Page的官网正式叫法其实是Windows Code Page。但是由于ANSI Code Page被误用的太广泛了，索性微软也就接受了此叫法，然后就叫做ANSI Code Page了。

类似地，ANSI Code Page=ANSI Windows Code Page

Windows的Code Page为何被误称为ANSI Code Page

Windows的Code Page中用的最广泛的是Windows 1252，其用于英语和西欧语言字符。

Windows 1252是基于ANSI草案（ANSI draft）而设计的。

结果，本来叫做Windows Code Page，就被很多不熟悉的人误读为ANSI Code Page

而实际上，（作为标准制定者的）ANSI和ISO，都从来没有去标准化Windows的Code Page，即没有为Windows Code Page指定过任何标准。

但是呢，由于ANSI Code Page被误用的太普遍了，导致微软官方也都承认此叫法了。

总之，记住一点，ANSI Code Page，就是Windows Code Page，就行了。

ANSI Code Page主要是用于Windows系统中，本地编码是非Unicode的，图形用户界面（GUI）程序。

ANSI的Code Page相关的表格，参见第 A.3.1.1 节 “ANSI Code Page表”

2.5.2.1.2. Windows的OEM Code Page表

OEM Code Page主要是用于Windows系统中的命令行界面（console）程序，虚拟DOS。

OEM Code Page可以视为是DOS和IBM PC时代的（过渡）剩余产品。

除了ANSI Code Page之外，之所以又设计出一个OEM Code Page，是因为：

1. 兼容性
   因为作为新的图形用户界面系统的Windows，也要兼容旧的命令行程序，即向后兼容性。

2. 字体和硬件的要求
   字体和旧的VGA硬件建议，文字图形界面所用的编码，最好和Code Page 437兼容。

   多数的OEM的Code Page，和（非ASCII的）后半部分的CP437，都是公用同一套代码点(code point)的。

   一般的OEM Code Page的后半段编码，和ANSI Code Page，完全不同。不过，对于部分双字节编码的，定长的Code Page（如泰语的847，越南语的1258）和多字节CJK编码的Code Page（如932,936,949,950）来说，ANSI和OEM的Code Page，都用的同一套编码。

和OEM Code Page相关的表格，参见第 A.3.1.2 节 “OEM Code Page表”

2.5.2.1.3. 一些常见的Code Page表

其中，ANSI和OEM共有的一些Code Page，可参见第 A.3.1.3 节 “ANSI和OEM共有的Code Page表”

而其他一些常见的Code Page，可参见第 A.3.1.4 节 “其他一些常见的Code Page表”

2.5.2.2. 所有的Code Page表

除了ANSI和OEM，以及ANSI，OEM共有的Code Page之外，其他还有很多Code Page定义。

关于所有的Windows中的Code Page的定义，可在微软官网[24]中找到。

此处已收录至第 A.3.2 节 “所有的Code Page相关的表格”，以方便查阅。

2.6. ANSI字符编码

2.6.1. ANSI是啥

ANSI，本身是American National Standards Institute的缩写，中文翻译为美国国家标准学会

ANSI是个非营利组织，其负责制定美国国家标准。

2.6.2. ANSI编码规则

ANSI字符编码的规则，或者是其所包含的字符的由来，主要是：

1. 0-127 (0x00-0x7F)
   完全和7位编码的ASCII字符集(ASA X3.4-1963)相同

2. 128-159 (0x80-0x9F)
   一些可打印字符

   这部分的编码，与国际编码ISO 8859-1的做法不同，ISO 8859-1是将此部分编码用于控制字符

3. 160-255 (0xA0-0FF)
   参考了ISO 8859-1中的字符

由此可以看出，ANSI中很多字符，和ISO-8859中的字符，看起来非常相似。

这就导致了很多人误以为，ANSI和ISO-8859是一回事呢。

总结：

ANSI

= Windows Code Page 1252

= Windows Codepage 1252

= Windows 1252

= CP 1252

= 共256个字符

= 0-127的ASCII + 128-159的可打印字符 + 160-255的和ISO 8859-1中类似的字符

2.6.3. ANSI (Windows 1252)编码表

关于ANSI（Windows 1252）编码表格，可以参考：

[20]

[35]

2.6.4. ANSI编码与ANSI的关系

既然ANSI负责制定美国的国标，而在计算机方面，由于计算机最早是从美国最开始发展的，相应的所用到的字符编码方面，ANSI也制定了对应的标准，所以就叫做ANSI字符编码/ANSI字符集，英文为ANSI Code/ANSI Encoding/ANSI set/ANSI charset

2.6.5. ANSI字符编码和Windows 1252

Windows为了支持英语和西欧字符，自己设计了一个编码，对应的在Code Page号是1252，被称为Windows 1252。

Windows 1252的设计，是参考了ANSI草案(ANSI Draft)。

而ANSI draft后来发展成为正式的国际标准：ISO 8859-1

即，Windows 1252是在其成为正式标准ISO 8859-1之前而设计的，因此很容易理解，Windows 1252和ISO 8859-1不是完全等同的。

下面就来简要说说两者的区别。

2.6.5.1. Windows 1252和ISO 8859-1之间的区别

Windows 1252和ISO 8859-1基本等同

有点不同的是，在128-159(0x80-0x9F)的范围的值，ISO 8859-1编码为控制字符，而微软编码为可打印字符。

提示

1. 类似Windows的Code Page为何被误称为ANSI Code Page，Windows 1252也被误称为ANSI编码,所以此处也可以说是ANSI编码和ISO 8859-1之间的区别。
2. 而由于ISO 8859-1对应的Latin-1的西欧语言，所以此处也可以称为ANSI编码和ISO Latin-1之间的区别，比如[19]
3. 微软的此种变体，有各种叫法：ANSI/Windows-1252/Windows Latin-1

   甚至有些微软的程序将其叫做Western European (Windows)。

   更有甚至，由于不清楚，而错称其为ASCII

注意

因此，如果你把包含了128-159范围内的ISO Latin-1编码的文件，用Windows的记事本Notepad去另存为为ANSI的话，则会导致文件内容被错误处理了。

因为本身的那些128-159的字符，是控制字符，但是却被Notepad识别为可打印的字符了。

总之，对于Windows 1252，目前的各种叫法，可以理解为：

ANSI = Windows 1252 = CP 1252 = Windows code page 1252 = Windows Latin-1

2.6.6. 为何"ANSI编码"（在Windows中）被称为"本地编码"

先说一下本地编码，所谓本地编码，即当前Windows中的二进制的值，用何种编码去解析，然后显示出对应的该编码中的字符。

即，当然系统使用什么类型的编码。

而ANSI编码，根据前面内容得知，只是一个普通的对应于Windows 1252的一个编码而已。并不是其他某些编码合集的总称。

但是有时候，却又看到有人把ANSI编码解释为“本地编码”，比如[22]

其意思，就是[30]中所说的，Windows code pages有时又被称为"active code pages"，"system active code pages"。

而作为微软用A表示ANSI版本的函数，W表示Wide，Unicode版本的函数，此时所有的A版本的函数，就都用的是当前有效的Code Page,即"本地编码"了

其中,Windows系统中，当前有且只有一个active Windows code page。

也就意味着，此处所谓的ANSI编码，就相当于之前所说的Code Page了，即当前系统采用何种编码去解析字符

也就是你当前系统中设置的本地编码为何种编码，然后系统中，遇到需要解析的字符，就按照你所设置的本地编码去解析了。

比如，本身对于中文GBK编码的字符，如果你本地编码设置为UTF-8，那么按照UTF-8编码去解析出来的GBK字符，当前就是乱码了。

而只有正确设置为GBK，才能正确解析原本就是GBK编码后的字符，才能正确显示出中文。

同理，用GBK编码去解析原本用UTF-8编码后的字符，也会导致乱码。

提示

这种乱码问题，常常会在和编码打交道的事情中遇到

比如Python中在命令行cmd中打印输出字符串，如果本身字符串是GBK编码的，那么你的cmd中的本地编码，就要设置为是936 (ANSI/OEM - Simplified Chinese GBK)，这样中文字符才能正确显示。

当然，如果你本身输出的字符中，即包含UTF-8编码的字符，又包含GBK编码的字符，那么则是无论如何设置，都是无法同时正常显示的。除非你转换为Unicode编码，然后让Python输出函数自动处理，才可以正确显示。

2.7. BOM

2.7.1. BOM是什么

BOM是一个Unicode字符。

BOM用于指示文件/字符流的大小端（字节序）。

不同编码所对应的BOM不同。

2.7.2. 为何需要BOM

即使对于字符进行了某种编码，可以正确读取/写入，可以正常显示字符了。

但是遇到将一个文件/字符流，从一个地方传输到另一个地方的话，就会遇到一些问题：

比如，本地是一个用UTF-16 LE编码的文件，传入到网络的另一端，接受者怎么知道你用的是UTF-16 LE的编码，而不是UTF-16 BE呢？

如果不知道，用了错误的UTF-16 BE去解码，且不是会导致乱码问题了。

所以，就需要一个说明和解释，目前的做法就是，在文件头（字符流的开始），添加一个BOM，Byte Order Mark，字节序的标记，用于表示此编码是LE还是BE。

2.7.3. BOM表

相关的不同编码所用的BOM，参见摘录自[36]的表 A.8 “不同编码所用的BOM”

关于UTF-8的BOM: EF BB BF,可参考第 2.4.3.2.1 节 “关于UTF-8的BOM： “EF BB BF” ”

2.8. 中文字符编码标准

2.8.1. GB2312，CP936，GBK，GB18030，GB13000

2.8.1.1. GB2312

1980年，中国制定了GB2312-80，一共收录了 7445 个字符，包括 6763 个汉字和 682 个其它符号。

GB2312-80，简称为GB2312。

在 Windows 中的代码页（Code Page）是 CP936。

2.8.1.2. GB13000

1993年，国际标准Unicode 1.1版本推出，收录中国大陆、台湾、日本及韩国通用字符集的汉字，总共有20,902个。

中国大陆订定了等同于Unicode 1.1版本的“GB 13000.1-93”，简称为GB13000。

GB13000，显然包含的GB2312已有的文字和其他很多为包含的文字，如GB 2312-80推出以后才简化的汉字（如“啰”），部分人名用字（如中国前总理朱镕基的“镕”字），台湾及香港使用的繁体字，日语及朝鲜语汉字等。

2.8.1.3. GBK

微软，对GB2312-80的扩展，即利用GB 2312-80未使用的编码空间，收录所有的GB 13000.1-93和Unicode 1.1之中的汉字全部字符，制定了GBK编码。

GBK 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。

GBK 作为对 GB2312 的扩展，在现在的 Windows 系统中仍然使用代码页 CP936 表示，但是同样的 936 的代码页跟一开始的 936 的代码页只支持 GB2312 编码不同，现在的 936 代码页支持 GBK 的编码，GBK 同时也向下兼容GB2312 编码。

所以，技术编码上，GBK兼容旧的GB2312，但是编码方式和GB13000不同，不兼容GB13000，但是所包含文字上，算是和GB13000相同。

2.8.1.4. GB18030

GBK自身并非国家标准，只是曾由国家技术监督局标准化司、电子工业部科技与质量监督司公布为“技术规范指导性文件”。

原始GB13000一直未被业界采用，2000年，国家出了标准GB18030-2000，简称GB18030，技术上兼容GBK而非GB13000，取代了 GBK1.0，成了正式的国家标准。

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。

GB18030 在 Windows 中的代码页是 CP54936。

这么多汉字编码标准的关系，总结起来就是第 2.8.2 节 “各种中文字符编码标准的关系”中所介绍的。

2.8.2. 各种中文字符编码标准的关系

（ 中国大陆的标准）GB 13000.1-93

=（国际标准）Unicode 1.1

（中国大陆标准）GB2312-80

= 简称GB2312

= Windows系统中的原先的CP936

（微软制定的）GBK

=（微软在编码方面）对 GB2312 的扩展

=（微软在所包含字符方面上包含了）GB 13000.1-93 + 其他部分汉字+ 台湾和香港的繁体 + 日语 + 朝鲜汉字

= Unicode 1.1 + 其他部分汉字+ 台湾和香港的繁体 + 日语 + 朝鲜汉字

对于GBK：

- 在编码方面：向下兼容GB2312，但是和GB 13000不同
- 在内容方面：等价于GB13000

微软中现在的新的CP936

= GBK

=兼容旧的GB2312

在技术编码方面上，演化顺序为：

ASCII ⇒ GB2312 ⇒ GBK ⇒ GB18030

后者对之前的，都是支持之前的编码，即向下兼容，即同一个字符，在这些编码中，都是同样的值，后面的标准，支持更多的字符。

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)

表 2.5. 中文字符相关编码标准

编码标准	别名	标准所属	包含字符
ASCII		国际通用	
GB2312	微软Windows中以前的CP936	中国大陆	6763 个汉字和 682 个其它符号
Unicode 1.1		国际通用	20,902个字符
GB13000		中国大陆	20,902个字符

编码标准	别名	标准所属	包含字符
GBK	微软Windows中现在的CP936	微软	21886 个符号
GB18030	微软Windows中的CP54936	中国大陆	27484 个汉字+其他少数民族字符

2.9. 字符存储（交换）标准

下表列出了常见的字符编码的标准，及其对应的交换存储时候所用的标准：

表 2.6. 字符（存储）交换标准

字符编码标准		存储(交换/传输)标准
包含字符	字符编码领域的叫法	
英文	ASCII ==ISO/IEC 646	ASCII
欧洲多国的字符	ISO 8859	
通用(的任何)字符	Unicode	UTF❶-8
		UTF❶-16
		UTF❶-32
简体中文❷	GB2312 ==GB2312-80 ==GB ==GB0❸	EUC❹-CN
	GBK	
	GB18030	
繁体中文	BIG5 ==大五码 ==五大码	CCCII
		CNS-11643
		EUC❹-TW
日文	JIS X 0208 == JIS C 6226和JIS X 0212	Shift JIS
		ISO-2022-JP
		EUC❹-JP
	JIS X 0213	EUC-JISX0213
韩文	KS X 1001 == KS C 5601	EUC❹-KR

❶ UTF==Unicode Transformation Format==Unicode转换格式
❷ 关于中文字符编码的各种标准的演化，可以参考中文字符编码标准+Unicode+Code Page[7]
❸ 关于GB0，另外还有其他的GBn，也是关于中文的国家标准：

1. GB1==GB/T 12345 – 90==《信息交换用汉字编码字符集 第一辅助集》
2. GB2==GB/T 7589 – 87==《信息交换用汉字编码字符集 第二辅助集》
3. GB3==GB 13131 – 91==《信息交换用汉字编码字符集 第三辅助集》
4. GB4==GB/T 7590 – 87==《信息交换用汉字编码字符集 第四辅助集》
5. GB5==GB 13132 – 91==《信息交换用汉字编码字符集 第五辅助集》

关于GB的含义，即国标的首字母。其中，强制标准冠以“GB”，推荐标准冠以“GB/T”。

[7] http://www.crifan.com/files/doc/docbook/zhcn_charset/release/html/zhcn_charset.html

❹ EUC==Extended Unix Code

2.10. 字形和你所看到的字符的关系

对于某个字符，其字形是固定的，是对应的字符编码标准，即编码集中所定义好了的。比如，入门的“入”，这一撇一捺，是连在一起的，你不能写错了，写成左右分开的，那错写成了八个的“八”了。而这样的字符的形状，简称字形，是编码中定义好的，你不能随便乱写。

但是同一个字符，具体的字体，大小等，则是按照自己喜好去设置的，是留给其他软件来处理的，比如宋体的“宋”这个汉字，不同的字体，会显示出来不同的效果：

图 2.5. 汉字“宋”的不同字体

汉字“宋”的不同字体

	微软雅黑	宋体	华文楷体	黑体	隶书
汉字“宋”	宋	宋	宋	宋	宋

这样的对于同一个字符的后期处理，即想要用什么样的字体，什么样的大小来显示等，都是后期软件，比如浏览器，微软的word等文本编辑器中去设置的。

参考书目

[1] 【转】字符编码笔记：ASCII，Unicode和UTF-8[1]

[2] 拉丁字母[2]

[3] 衍生拉丁字母[3]

[4] ASCII[4]

[5] ISO/IEC 6429[5]

[6] ASCII字符集中的功能 控制字符[6]

[7] ISO/IEC 646[7]

[8] ISO/IEC 8859[8]

[9] 国家标准代码[9]

[10] EUC[10]

[11] EBCDIC[11]

[12] ISO/IEC 8859-1[12]

[13] 通用字符集[13]

[14] 大端(Big Endian)与小端(Little Endian)详解[14]

[15] 中文字符编码标准+Unicode+Code Page[15]

[16] UTF-8[16]

[17] ANSI[17]

[18] Windows 1252[18]

[19] Cast of Characters- ASCII, ANSI, UTF-8 and all that[19]

[20] ANSI character set and equivalent Unicode and HTML characters[20]

[1] http://www.crifan.com/switch_character_encoding_notes_ascii_unicode_and_utf-8/
[2] http://zh.wikipedia.org/wiki/%E6%8B%89%E4%B8%81%E5%AD%97%E6%AF%8D
[3] http://zh.wikipedia.org/wiki/%E8%A1%8D%E7%94%9F%E6%8B%89%E4%B8%81%E5%AD%97%E6%AF%8D
[4] http://baike.baidu.com/view/15482.htm
[5] http://webstore.iec.ch/preview/info_isoiec6429%7Bed3.0%7Den.pdf
[6] http://www.crifan.com/the_ascii_character_set_function__control_characters_function__control_code__character_in_ascii/
[7] http://zh.wikipedia.org/wiki/ISO/IEC_646
[8] http://zh.wikipedia.org/wiki/ISO/IEC_8859
[9] http://zh.wikipedia.org/wiki/%E5%9B%BD%E6%A0%87%E7%A0%81
[10] http://zh.wikipedia.org/wiki/EUC
[11] http://zh.wikipedia.org/wiki/EBCDIC
[12] http://zh.wikipedia.org/wiki/ISO/IEC_8859-1
[13] http://zh.wikipedia.org/wiki/%E9%80%9A%E7%94%A8%E5%AD%97%E7%AC%A6%E9%9B%86
[14] http://www.crifan.com/big_endian_big_endian_and_small_end_little_endian_detailed/
[15] http://www.crifan.com/files/doc/docbook/zhcn_charset/release/html/zhcn_charset.html
[16] http://en.wikipedia.org/wiki/UTF-8
[17] http://en.wikipedia.org/wiki/American_National_Standards_Institute
[18] http://en.wikipedia.org/wiki/Windows-1252
[19] http://vlaurie.com/computers2/Articles/characters.htm
[20] http://www.alanwood.net/demos/ansi.html

[21] ASCII vs ANSI Encoding[21]

[22] 字符，字节和编码[22]

[23] http://www.sttmedia.com/unicode-ansiASCII and ANSI

[24] Code Page Identifiers [23]

[25] 什么是Unicode？什么是UTF-8？[24]

[26] Code Page[25]

[27] 什么是字符集、编码、代码页（code page）[26]

[28] GBK[27]

[29] 中文编码 GB2312||GBK||GB18030||GB13000 详解[28]

[30] Code Pages[29]

[31] Code Pages Supported by Windows -- Windows Code Pages [30]

[32] Code Pages Supported by Windows -- OEM Code Pages [31]

[33] Code Pages [32]

[34] Windows code page[33]

[35] Microsoft Windows Codepage 1252 (ANSI)[34]

[36] Byte Order Mark[35]

[37] Byte Order Mark (BOM)[36]

[21] http://weblogs.asp.net/ahoffman/archive/2004/01/19/60094.aspx
[22] http://www.regexlab.com/zh/encoding.htm
[23] http://msdn.microsoft.com/en-us/library/windows/desktop/dd317756(v=vs.85).aspx
[24] http://www.bianceng.cn/web/Html/200710/4839.htm
[25] http://en.wikipedia.org/wiki/Code_page
[26] http://zhidao.baidu.com/question/43462968
[27] http://zh.wikipedia.org/wiki/GBK
[28] http://www.satsuns.com/ci/1373.html
[29] http://msdn.microsoft.com/en-us/library/windows/desktop/dd317752(v=vs.85).aspx
[30] http://msdn.microsoft.com/en-us/goglobal/bb964654
[31] http://msdn.microsoft.com/en-us/goglobal/bb964655
[32] http://msdn.microsoft.com/en-us/goglobal/bb964653
[33] http://en.wikipedia.org/wiki/Windows_code_page
[34] http://www.kostis.net/charsets/cp1252.htm
[35] http://en.wikipedia.org/wiki/Byte_order_mark
[36] http://www.sttmedia.com/unicode-byteordermark

附录 A. 编码相关的表格

A.1. ASCII编码表(0-127)

表 A.1. ASCII编码表(0-127)

十六进制	十进制	字符	十六进制	十进制	字符	十六进制	十进制	字符	十六进制	十进制	字符
0	0	NUL	20	32	[space]	40	64	@	60	96	'
1	1	SOH	21	33	!	41	65	A	61	97	a
2	2	STX	22	34	"	42	66	B	62	98	b
3	3	ETX	23	35	#	43	67	C	63	99	c
4	4	EOT	24	36	$	44	68	D	64	100	d
5	5	ENQ	25	37	%	45	69	E	65	101	e
6	6	ACK	26	38	&	46	70	F	66	102	f
7	7	BEL	27	39	`	47	71	G	67	103	g
8	8	BS	28	40	(	48	72	H	68	104	h
9	9	HT	29	41	)	49	73	I	69	105	i
0a	10	LF	2a	42	*	4a	74	J	6a	106	j
0b	11	VT	2b	43	+	4b	75	K	6b	107	k
0c	12	FF	2c	44	,	4c	76	L	6c	108	l
0d	13	CR	2d	45	-	4d	77	M	6d	109	m
0e	14	SO	2e	46	.	4e	78	N	6e	110	n
0f	15	SI	2f	47	/	4f	79	O	6f	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1a	26	SUB	3a	58	:	5a	90	Z	7a	122	z
1b	27	ESC	3b	59	;	5b	91	[	7b	123	{
1c	28	FS	3c	60	<	5c	92	\	7c	124	|
1d	29	GS	3d	61	=	5d	93	]	7d	125	}
1e	30	RS	3e	62	>	5e	94	^	7e	126	~
1f	31	US	3f	63	?	5f	95	_	7f	127	[delete]

A.2. ISO/IEC 8859编码标准中的15种字符集

表 A.2. ISO/IEC 8859编码标准中的15种字符集

ISO/IEC 8859-n	英文别名	中文解释
ISO/IEC 8859 -1	Latin-1	西欧语言
ISO/IEC 8859 -2	Latin-2	中欧语言
ISO/IEC 8859 -3	Latin-3	南欧语言。世界语也可用此字符集显示。
ISO/IEC 8859 -4	Latin-4	北欧语言
ISO/IEC 8859 -5	Cyrillic	斯拉夫语言
ISO/IEC 8859 -6	Arabic	阿拉伯语
ISO/IEC 8859 -7	Greek	希腊语
ISO/IEC 8859 -8	Hebrew	希伯来语（视觉顺序）；ISO 8859-8-I是 希伯来语（逻辑顺序）
ISO/IEC 8859 -9	Latin-5 或 Turkish	它把Latin-1的冰岛语字母换走，加入土耳其语字母
ISO/IEC 8859 -10	Latin-6 或 Nordic	北日耳曼语支，用来代替Latin-4
ISO/IEC 8859 -11	Thai	从泰国的 TIS620 标准字集演化而来
ISO/IEC 8859 -13	Latin-7 或 Baltic Rim	波罗的语族
ISO/IEC 8859 -14	Latin-8 或 Celtic	凯尔特语族
ISO/IEC 8859 -15	Latin-9	西欧语言，加入Latin-1欠缺的芬兰语字母和大写法语重音字母，以及欧元（€）符号。
ISO/IEC 8859 -16	Latin-10	东南欧语言。主要供罗马尼亚语使用，并加入欧元符号。

A.3. Code Page表格

A.3.1. 常见的ANSI和OEM的Code Page的表格

A.3.1.1. ANSI Code Page表

ANSI Code Page中，字符编码都是用的8位单字节，所以也被称为SBCS (Single Byte Character Set) Codepages，即单字节字符集的代码页

表 A.3. ANSI的SBCS Code Page

代码页Code Page	对应的字符集Character Set
Code Page 1250	Central and East European Latin
Code Page 1251	Cyrillic
Code Page 1252	West European Latin
Code Page 1253	Greek
Code Page 1254	Turkish
Code Page 1255	Hebrew
Code Page 1256	Arabic
Code Page 1257	Baltic
Code Page 1258	Vietnamese
Code Page 874	Thai

A.3.1.2. OEM Code Page表

OEM的Code Page如下：

表 A.4. OEM的Code Page

代码页Code Page	对应的字符集Character Set
Code Page 437	original IBM PC Code Page==USA
Code Page 720	Arabic
Code Page 737	Greek
Code Page 775	Estonian, Lithuanian and Latvian
Code Page 850	"Multilingual (Latin-1)" (Western European languages)
Code Page 852	"Slavic (Latin-2)" (Central and Eastern European languages)
Code Page 855	Cyrillic
Code Page 857	Turkish
Code Page 858	"Multilingual" with euro symbol
Code Page 860	Portuguese
Code Page 861	Icelandic
Code Page 862	Hebrew
Code Page 863	French (Quebec French)
Code Page 865	Danish/Norwegian Differs from 437
Code Page 869	Greek
Code Page 874	Thai

A.3.1.3. ANSI和OEM共有的Code Page表

ANSI和OEM共有Code Page，是一些DBCS (Double Byte Character Set) Codepages，即双字节字符集的代码页。

表 A.5. ANSI和OEM共有的DBCS Code Page

代码页Code Page	对应的字符集Character Set
Code Page 932	Japanese
Code Page 936	GBK - Simplified Chinese
Code Page 949	Korean
Code Page 950	BIG5 - Traditional Chinese

A.3.1.4. 其他一些常见的Code Page表

其他一些常见的Code Page如下：

表 A.6. 一些常见的Code Page

代码页Code Page	对应的字符集Character Set
Code Page 1200	UTF-16LE Unicode little-endian
Code Page 1201	UTF-16BE Unicode big-endian
Code Page 65000	UTF-7 Unicode
Code Page 65001	UTF-8 Unicode

代码页Code Page	对应的字符集Character Set
Code Page 10000	Macintosh Roman encoding (followed by several other Mac character sets)
Code Page 10007	Macintosh Cyrillic encoding
Code Page 10029	Macintosh Central European encoding
Code Page 20127	US-ASCII The classic US 7 bit character set with no char larger than 127
Code Page 28591	ISO-8859-1 (followed by ISO-8859-2 to ISO-8859-15)

A.3.2. 所有的Code Page相关的表格

表 A.7. 微软的代码页标识符(Code Page Identifiers)

Identifie	.NET Name	Additional information
037	IBM037	IBM EBCDIC US-Canada
437	IBM437	OEM United States
500	IBM500	IBM EBCDIC International
708	ASMO-708	Arabic (ASMO 708)
709		Arabic (ASMO-449+, BCON V4)
710		Arabic - Transparent Arabic
720	DOS-720	Arabic (Transparent ASMO); Arabic (DOS)
737	ibm737	OEM Greek (formerly 437G); Greek (DOS)
775	ibm775	OEM Baltic; Baltic (DOS)
850	ibm850	OEM Multilingual Latin 1; Western European (DOS)
852	ibm852	OEM Latin 2; Central European (DOS)
855	IBM855	OEM Cyrillic (primarily Russian)
857	ibm857	OEM Turkish; Turkish (DOS)
858	IBM00858	OEM Multilingual Latin 1 + Euro symbol
860	IBM860	OEM Portuguese; Portuguese (DOS)
861	ibm861	OEM Icelandic; Icelandic (DOS)
862	DOS-862	OEM Hebrew; Hebrew (DOS)
863	IBM863	OEM French Canadian; French Canadian (DOS)
864	IBM864	OEM Arabic; Arabic (864)
865	IBM865	OEM Nordic; Nordic (DOS)
866	cp866	OEM Russian; Cyrillic (DOS)
869	ibm869	OEM Modern Greek; Greek, Modern (DOS)
870	IBM870	IBM EBCDIC Multilingual/ROECE (Latin 2); IBM EBCDIC Multilingual Latin 2
874	windows-874	ANSI/OEM Thai (same as 28605, ISO 8859-15); Thai (Windows)
875	cp875	IBM EBCDIC Greek Modern
932	shift_jis	ANSI/OEM Japanese; Japanese (Shift-JIS)
936	gb2312	ANSI/OEM Simplified Chinese (PRC, Singapore); Chinese Simplified (GB2312)
949	ks_c_5601-1987	ANSI/OEM Korean (Unified Hangul Code)

Identifie	.NET Name	Additional information
950	big5	ANSI/OEM Traditional Chinese (Taiwan; Hong Kong SAR, PRC); Chinese Traditional (Big5)
1026	IBM1026	IBM EBCDIC Turkish (Latin 5)
1047	IBM01047	IBM EBCDIC Latin 1/Open System
1140	IBM01140	IBM EBCDIC US-Canada (037 + Euro symbol); IBM EBCDIC (US-Canada-Euro)
1141	IBM01141	IBM EBCDIC Germany (20273 + Euro symbol); IBM EBCDIC (Germany-Euro)
1142	IBM01142	IBM EBCDIC Denmark-Norway (20277 + Euro symbol); IBM EBCDIC (Denmark-Norway-Euro)
1143	IBM01143	IBM EBCDIC Finland-Sweden (20278 + Euro symbol); IBM EBCDIC (Finland-Sweden-Euro)
1144	IBM01144	IBM EBCDIC Italy (20280 + Euro symbol); IBM EBCDIC (Italy-Euro)
1145	IBM01145	IBM EBCDIC Latin America-Spain (20284 + Euro symbol); IBM EBCDIC (Spain-Euro)
1146	IBM01146	IBM EBCDIC United Kingdom (20285 + Euro symbol); IBM EBCDIC (UK-Euro)
1147	IBM01147	IBM EBCDIC France (20297 + Euro symbol); IBM EBCDIC (France-Euro)
1148	IBM01148	IBM EBCDIC International (500 + Euro symbol); IBM EBCDIC (International-Euro)
1149	IBM01149	IBM EBCDIC Icelandic (20871 + Euro symbol); IBM EBCDIC (Icelandic-Euro)
1200	utf-16	Unicode UTF-16, little endian byte order (BMP of ISO 10646); available only to managed applications
1201	unicodeFFFE	Unicode UTF-16, big endian byte order; available only to managed applications
1250	windows-1250	ANSI Central European; Central European (Windows)
1251	windows-1251	ANSI Cyrillic; Cyrillic (Windows)
1252	windows-1252	ANSI Latin 1; Western European (Windows)
1253	windows-1253	ANSI Greek; Greek (Windows)
1254	windows-1254	ANSI Turkish; Turkish (Windows)
1255	windows-1255	ANSI Hebrew; Hebrew (Windows)
1256	windows-1256	ANSI Arabic; Arabic (Windows)
1257	windows-1257	ANSI Baltic; Baltic (Windows)
1258	windows-1258	ANSI/OEM Vietnamese; Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	MAC Roman; Western European (Mac)
10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	MAC Traditional Chinese (Big5); Chinese Traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x-mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)

Identifie	.NET Name	Additional information
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	MAC Simplified Chinese (GB 2312); Chinese Simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	MAC Latin 2; Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
12000	utf-32	Unicode UTF-32, little endian byte order; available only to managed applications
12001	utf-32BE	Unicode UTF-32, big endian byte order; available only to managed applications
20000	x-Chinese_CNS	CNS Taiwan; Chinese Traditional (CNS)
20001	x-cp20001	TCA Taiwan
20002	x_Chinese-Eten	Eten Taiwan; Chinese Traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan
20105	x-IA5	IA5 (IRV International Alphabet No. 5, 7-bit); Western European (IA5)
20106	x-IA5-German	IA5 German (7-bit)
20107	x-IA5-Swedish	IA5 Swedish (7-bit)
20108	x-IA5-Norwegian	IA5 Norwegian (7-bit)
20127	us-ascii	US-ASCII (7-bit)
20261	x-cp20261	T.61
20269	x-cp20269	ISO 6937 Non-Spacing Accent
20273	IBM273	IBM EBCDIC Germany
20277	IBM277	IBM EBCDIC Denmark-Norway
20278	IBM278	IBM EBCDIC Finland-Sweden
20280	IBM280	IBM EBCDIC Italy
20284	IBM284	IBM EBCDIC Latin America-Spain
20285	IBM285	IBM EBCDIC United Kingdom
20290	IBM290	IBM EBCDIC Japanese Katakana Extended
20297	IBM297	IBM EBCDIC France
20420	IBM420	IBM EBCDIC Arabic
20423	IBM423	IBM EBCDIC Greek
20424	IBM424	IBM EBCDIC Hebrew
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC Korean Extended

Identifie	.NET Name	Additional information
20838	IBM-Thai	IBM EBCDIC Thai
20866	koi8-r	Russian (KOI8-R); Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC Icelandic
20880	IBM880	IBM EBCDIC Cyrillic Russian
20905	IBM905	IBM EBCDIC Turkish
20924	IBM00924	IBM EBCDIC Latin 1/Open System (1047 + Euro symbol)
20932	EUC-JP	Japanese (JIS 0208-1990 and 0121-1990)
20936	x-cp20936	Simplified Chinese (GB2312); Chinese Simplified (GB2312-80)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC Cyrillic Serbian-Bulgarian
21027		(deprecated)
21866	koi8-u	Ukrainian (KOI8-U); Cyrillic (KOI8-U)
28591	iso-8859-1	ISO 8859-1 Latin 1; Western European (ISO)
28592	iso-8859-2	ISO 8859-2 Central European; Central European (ISO)
28593	iso-8859-3	ISO 8859-3 Latin 3
28594	iso-8859-4	ISO 8859-4 Baltic
28595	iso-8859-5	ISO 8859-5 Cyrillic
28596	iso-8859-6	ISO 8859-6 Arabic
28597	iso-8859-7	ISO 8859-7 Greek
28598	iso-8859-8	ISO 8859-8 Hebrew; Hebrew (ISO-Visual)
28599	iso-8859-9	ISO 8859-9 Turkish
28603	iso-8859-13	ISO 8859-13 Estonian
28605	iso-8859-15	ISO 8859-15 Latin 9
29001	x-Europa	Europa 3
38598	iso-8859-8-i	ISO 8859-8 Hebrew; Hebrew (ISO-Logical)
50220	iso-2022-jp	ISO 2022 Japanese with no halfwidth Katakana; Japanese (JIS)
50221	csISO2022JP	ISO 2022 Japanese with halfwidth Katakana; Japanese (JIS-Allow 1 byte Kana)
50222	iso-2022-jp	ISO 2022 Japanese JIS X 0201-1989; Japanese (JIS-Allow 1 byte Kana - SO/SI)
50225	iso-2022-kr	ISO 2022 Korean
50227	x-cp50227	ISO 2022 Simplified Chinese; Chinese Simplified (ISO 2022)
50229		ISO 2022 Traditional Chinese
50930		EBCDIC Japanese (Katakana) Extended
50931		EBCDIC US-Canada and Japanese
50933		EBCDIC Korean Extended and Korean
50935		EBCDIC Simplified Chinese Extended and Simplified Chinese
50936		EBCDIC Simplified Chinese
50937		EBCDIC US-Canada and Traditional Chinese
50939		EBCDIC Japanese (Latin) Extended and Japanese
51932	euc-jp	EUC Japanese

Identifie	.NET Name	Additional information
51936	EUC-CN	EUC Simplified Chinese; Chinese Simplified (EUC)
51949	euc-kr	EUC Korean
51950		EUC Traditional Chinese
52936	hz-gb-2312	HZ-GB2312 Simplified Chinese; Chinese Simplified (HZ)
54936	GB18030	Windows XP and later: GB18030 Simplified Chinese (4 byte); Chinese Simplified (GB18030)
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali
57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii-as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)
65001	utf-8	Unicode (UTF-8)

A.4. 不同编码所用的BOM

表 A.8. 不同编码所用的BOM

编码类型	BOM值(16进制)	BOM值(10进制)
UTF-8	EF BB BF	239 187 191
UTF-16 (BE)	FE FF	254 255
UTF-16 (LE)	FF FE	255 254
UTF-32 (BE)	00 00 FE FF	0 0 254 255
UTF-32 (LE)	FF FE 00 00	255 254 0 0
UTF-7	2B 2F 76 38	43 47 118 56
	2B 2F 76 39	43 47 118 57
	2B 2F 76 2B	43 47 118 43
	2B 2F 76 2F	43 47 118 47
UTF-1	F7 64 4C	247 100 76
UTF-EBCDIC	DD 73 66 73	221 115 102 115
SCSU	0E FE FF	14 254 255
BOCU-1	FB EE 28	251 238 40
GB-18030	84 31 95 33	132 49 149 51