字符编码详解

版本: **1.0**

作者: crifan

邮箱: green-waste（ at ）163.com

关键字

字符编码，ASCII，ISO 8859，ISO 10646，UCS，Unicode，UTF-8

版本

版本	日期	内容更新
1.0	2011-11-02	添加了编码相关背景知识介绍 添加了 ASCII 和 EASCII 编码介绍 添加了 ISO/IEC 8859 相关的编码和各种单字节编码的关系 添加了 Unicode 和 ISO 10646 的解释 添加了 UTF-8 和 Unicode 的区别和联系

目录

1 正文之前 5
1.1 目的 5
1.2 本文内容 5
1.3 声明 5
2 字符编码相关的背景知识 6
2.1 拉丁字母 6
2.2 什么是字符编码 6
3 字符编码标准 8
3.1 只支持基本的拉丁字符的字符编码：ASCII 8
3.1.1 ASCII 的由来 8
3.1.2 ASCII 编码规则 8
3.1.3 ISO 646 9
3.1.4 ASCII 码表/编码字符表 9
3.2 支持多种衍生拉丁字母的字符编码：EASCII 和 ISO 8859 10
3.2.1 EASCII 11
3.2.2 ISO 8859 11
3.2.2.1 ISO/IEC 8859 出现的背景 11
3.2.2.2 ISO/IEC 8859 的编码规则 11
3.2.2.3 ISO/IEC 8859 的特点 13
3.2.2.4 ISO/IEC 6429 14
3.2.2.5 ISO 8859 和 ISO-8859 的区别和联系 14
3.2.2.5.1 原先的 ISO 8859-1 和我们常说的 ISO 8859-1 15
3.3 各种单字节编码标准的关系 17
3.4 支持世界上几乎所有字符的字符编码：Unicode 17
3.4.1 Unicode 和 ISO 10646 的关系 18
3.4.1.1 ISO 10646=UCS 18
3.4.1.2 Unicode 和 ISO 10646 的联系 18
3.4.1.3 Unicode 和 ISO 10646 的区别 19
3.4.2 Unicode 编码规则 20
3.4.3 Unicode 字符编码所对应的存储和交换标准：UTF-8, UTF-16, UTF-32 21

3.4.3.1 UTF-8 21
3.4.3.2 Unicode 与 UTF-8 之间的转换 22
3.4.3.2.1 关于 UTF-8 的 BOM：“EF BB BF” 23
3.5 字符存储（交换）标准 24
3.6 字形和你所看到的字符的关系 25
4 引用 26

图表

图表 1 ASCII 编码表 10
图表 2 ISO/IEC 8859 编码标准中的 15 种字符集 12
图表 3 ISO/IEC 8859 的 15 个字符集的部分比较 13
图表 4 ISO/IEC 8859-1 字符集表 16
图表 5 各种单字节编码标准之间的关系 17
图表 6 ISO/IEC 10646 与 Unicode 的版本对应关系 19
图表 7 Unicode 中的各种平面划分 20
图表 8 Unicode 与 UTF-8 之间的编码映射关系 22
图表 9 Notepad 中的各种编码 22
图表 10 字符（存储）交换标准 24
图表 11 汉字“宋”的不同字体 25

缩写

缩写	全称	含义
ASCII	American Standard Code for Information Interchange	美国信息交换标准代码
BMP	Basic Multilingual Plane	基本多文种平面
EBCDIC	Extended Binary Coded Decimal Interchange Code	扩展二进制编码十进制交换码
IANA	Internet Assigned Numbers Authority	互联网号码分配局
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	Universal Character Set	通用字符集
UTF	Unicode Transformation Format	Unicode 转换格式

1 正文之前

1.1 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。

1.2 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

http://againinput4.blog.163.com/blog/static/172799491201192410540799/

才搞懂基本概念的，在此感谢一下这位仁兄。

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。

1.3 声明

任何问题，意见，建议等，都欢迎一起探讨：green-waste (at) 163.com

2 字符编码相关的背景知识

2.1 拉丁字母

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）

2.2 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。

3 字符编码标准

3.1 只支持基本的拉丁字符的字符编码：ASCII

3.1.1 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。

3.1.2 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。

注：

（1）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）

关于控制字符的详细解释，请参考这里：

ASCII 字符集中的功能 控制字符

http://againinput4.blog.163.com/blog/static/172799491201111510423599 2/

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，具体对应关系，请查询 ASCII 码表。

ASCII 字符集共 2^7=128 个字符

= 33 个控制字符 + 95 个可见字符。

3.1.3 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

3.1.4 ASCII 码表/编码字符表

对应的哪个值对应那个字符，即对应的映射表，常说的 ASCII 码表，如下：

图表 1 ASCII 编码表

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]

3.2 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。

3.2.1 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 ISO 8859 编码标准。

3.2.2 ISO 8859

3.2.2.1ISO/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。

3.2.2.2ISO/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。

图表 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。

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：

图表 3 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	Ђ		’	¢	¢	Ē	ข	¢	ḃ	¢	ą
10100011	243	163	A3	£	Ł	£	Ŗ	Ѓ		£	£	£	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	І		¦	¦	¦	Ķ	ฆ	¦	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，请参见这里：
ISO/IEC 8859
http://zh.wikipedia.org/wiki/ISO/IEC_8859

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。

3.2.2.3ISO/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，后文会详细解释。
此处先来说说，其他几个和 ISO/IEC 8859 相关的内容。

3.2.2.4ISO/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

3.2.2.5ISO 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

3.2.2.5.1 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：

图表 4 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）。

上表中的绿色的部分，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

3.3 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：

图表 5 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（注 1）	0x7F	0x80-0x9F	0xA0-0xFF
ASCII	=ISO/IEC 646	√	√	√		
ISO/IEC 6429	=C0 control + C1 control	√			√	
ISO 8859	=ISO/IEC 8859-n = **ISO/IEC 8859-1** ... ISO/IEC 8859-11 ISO/IEC 8859-13 ... ISO/IEC 8859-16		√			
ISO-8859	=ISO-8859-n = **ISO-8859-1** ... ISO-8859-11 ISO-8859-13 ... ISO-8859-16	√	√	√	√	

注：

（1）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。

3.4 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。

3.4.1 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。

3.4.1.1ISO 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:
级别 1：不支持组合字符和諺文字母字符。
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。
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。

3.4.1.2Unicode 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：

图表 6 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

3.4.1.3Unicode 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

3.4.2 Unicode 编码规则

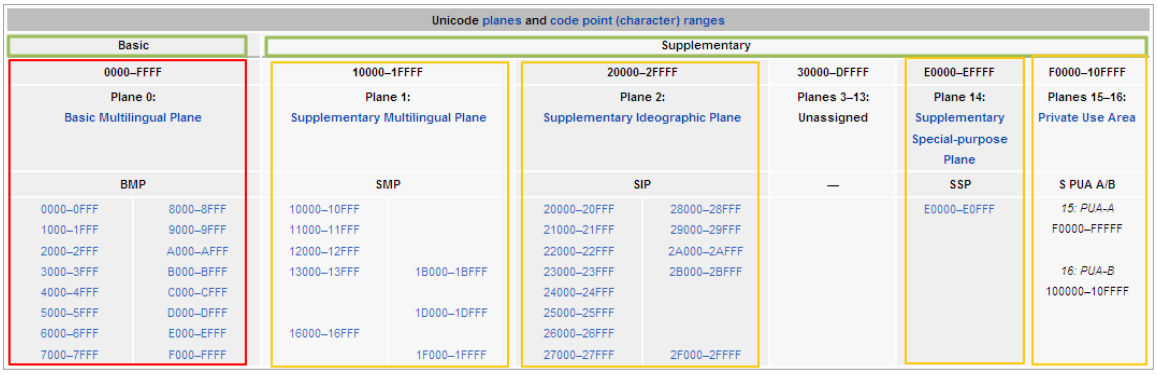
为了将世界上几乎所有的字符都涵盖了，那么就要了解世界上，有哪些字符。
除了之前 ASCII 的拉丁字母，ISO 8859 所包含的欧洲多国用的字符之外，亚洲一些国家，包括中文，日文，韩文等，尤其是中文，包含的字符数，大概有几万个。
因此，Unicode 的编码就要设计的把这么多的字符都包含在内。

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，参见下图：

图表 7 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。

3.4.3 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。

3.4.3.1UTF-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：

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。

图表 8 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。

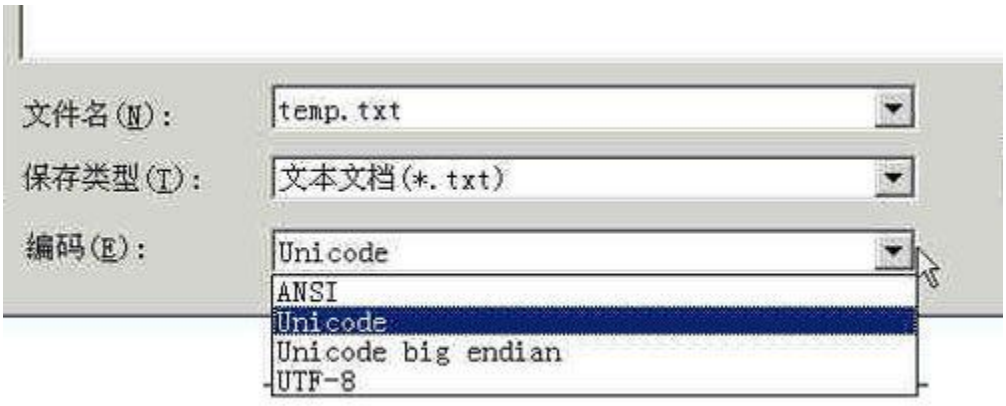
已知“严”的 Unicode 是 4E25（100111000100101），根据上表，可以发现 4E25 处在第三行的范围内（0000 0800-0000 FFFF），因此“严”的 UTF-8 编码需要三个字节，即格式是“1110xxxx 10xxxxxx 10xxxxxx”。然后，从“严”的最后一个二进制位开始，依次从后向前填入格式中的 x，多出的位补 0。这样就得到了，“严”的 UTF-8 编码是“11100100 10111000 10100101”，转换成十六进制就是 E4B8A5。

3.4.3.2Unicode 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”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。

图表 9 Notepad 中的各种编码



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。

3）Unicode Big Endian 编码与上一个选项相对应。

关于 Little Endian 和 Big Endian，可以参考这里：

大端(Big Endian)与小端(Little Endian)详解

http://hi.baidu.com/serial_story/blog/item/7e110587c3ed8e29c75cc3c7.html

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。

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。

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。

3.4.3.2.1 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。

3.5 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：

图表 10 字符（存储）交换标准

<table>
<tr><th colspan="2">字符编码标准</th><th>存储(交换/传输)标准</th></tr>
<tr><th>包含字符</th><th>字符编码领域的叫法</th><th></th></tr>
<tr><td>英文</td><td>ASCII
= ISO/IEC 646</td><td>ASCII</td></tr>
<tr><td>欧洲多国的字符</td><td>ISO 8859</td><td></td></tr>
<tr><td rowspan="3">通用(的任何)字符</td><td rowspan="3">Unicode</td><td>UTF-8</td></tr>
<tr><td>UTF-16</td></tr>
<tr><td>UTF-32</td></tr>
<tr><td rowspan="3">简体中文</td><td>GB2312
==GB2312-80
==GB
==GB0</td><td rowspan="3">EUC-CN</td></tr>
<tr><td>GBK</td></tr>
<tr><td>GB18030</td></tr>
<tr><td rowspan="3">繁体中文</td><td rowspan="3">BIG5
==大五码
==五大码</td><td>CCCII</td></tr>
<tr><td>CNS-11643</td></tr>
<tr><td>EUC-TW</td></tr>
<tr><td rowspan="3">日文</td><td rowspan="2">JIS X 0208
== JIS C 6226
和
JIS X 0212</td><td>Shift JIS</td></tr>
<tr><td>ISO-2022-JP</td></tr>
<tr><td>JIS X 0213</td><td>EUC-JP</td></tr>
<tr><td>韩文</td><td>KS X 1001
== KS C 5601</td><td>EUC-KR</td></tr>
</table>

注：

1. 关于 GB0，另外还有其他的 GBn，也是关于中文的国家标准：

GB1==GB/T 12345 – 90==《信息交换用汉字编码字符集 第一辅助集》

GB2==GB/T 7589 – 87==《信息交换用汉字编码字符集 第二辅助集》

GB3==GB 13131 – 91==《信息交换用汉字编码字符集 第三辅助集》

GB4==GB/T 7590 – 87==《信息交换用汉字编码字符集 第四辅助集》

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”。

2. EUC==Extended Unix Code

3. UTF==Unicode Transformation Format== Unicode 转换格式

4.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

http://bbs.chinaunix.net/thread-3610023-1-1.html

3.6 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：

图表 11 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。

4 引用

【转】字符编码笔记：ASCII，Unicode 和 UTF-8
http://againinput4.blog.163.com/blog/static/172799491201192410540799/

拉丁字母
http://zh.wikipedia.org/wiki/%E6%8B%89%E4%B8%81%E5%AD%97%E6%AF%8D

衍生拉丁字母
http://zh.wikipedia.org/wiki/%E8%A1%8D%E7%94%9F%E6%8B%89%E4%B8%81%E5%AD%97%E6%AF%8D

ASCII
http://baike.baidu.com/view/15482.htm

ISO/IEC 6429
http://webstore.iec.ch/preview/info_isoiec6429%7Bed3.0%7Den.pdf

ASCII 字符集中的功能 控制字符
http://againinput4.blog.163.com/blog/static/172799491201111510423599 2/

ISO/IEC 646
http://zh.wikipedia.org/wiki/ISO/IEC_646

ISO/IEC 8859
http://zh.wikipedia.org/wiki/ISO/IEC_8859

国家标准代码
http://zh.wikipedia.org/wiki/%E5%9B%BD%E6%A0%87%E7%A0%81

EUC
http://zh.wikipedia.org/wiki/EUC

EBCDIC
http://zh.wikipedia.org/wiki/EBCDIC

ISO/IEC 8859-1
http://zh.wikipedia.org/wiki/ISO/IEC_8859-1

通用字符集
http://zh.wikipedia.org/wiki/%E9%80%9A%E7%94%A8%E5%AD%97%E7%AC%A6%E9%9B%86

大端(Big Endian)与小端(Little Endian)详解
http://hi.baidu.com/serial_story/blog/item/7e110587c3ed8e29c75cc3c7.html

中文字符编码标准+Unicode+Code Page
http://bbs.chinaunix.net/thread-3610023-1-1.html

UTF-8
http://en.wikipedia.org/wiki/UTF-8