ASCII码对照表，ASCII码一览表（非常详细）

ASCII（American Standard Code for Information Interchange，美国信息互换标准代码）是一套基于拉丁字母的字符编码，共收录了 128 个字符，用一个字节就可以存储，它等同于国际标准 ISO/IEC 646。

ASCII 编码于 1967 年第一次发布，最后一次更新是在 1986 年，迄今为止共收录了 128 个字符，包含了基本的拉丁字母（英文字母）、阿拉伯数字（也就是 1234567890）、标点符号（,.!等）、特殊符号（@#$%^&等）以及一些具有控制功能的字符（往往不会显示出来）。

以上说的是标准 ASCII 编码，是学习编程语言必须了解的。标准 ASCII 编码用一个字节中的 7 位就能存储，为了让第 8 位（最高位）也参与编码，就形成了扩展 ASCII 编码。扩展 ASCII 主要包含了一些特殊符号、外来语字母和图形符号。

针对扩展的 ASCII 编码，不同的国家有不同的字符集，所以它并不是国际标准，本文也不会展示，感兴趣的读者请转到：扩展 ASCII 码对照表

标准 ASCII 码对照表（淡黄色背景为控制字符，白色背景为可显示字符）

二进制	八进制	十进制	十六进制	字符/缩写	解释
00000000	000	0	00	NUL (NULL)	空字符
00000001	001	1	01	SOH (Start Of Headling)	标题开始
00000010	002	2	02	STX (Start Of Text)	正文开始
00000011	003	3	03	ETX (End Of Text)	正文结束
00000100	004	4	04	EOT (End Of Transmission)	传输结束
00000101	005	5	05	ENQ (Enquiry)	请求
00000110	006	6	06	ACK (Acknowledge)	回应/响应/收到通知
00000111	007	7	07	BEL (Bell)	响铃
00001000	010	8	08	BS (Backspace)	退格
00001001	011	9	09	HT (Horizontal Tab)	水平制表符

00001010	012	10	0A	LF/NL(Line Feed/New Line)	换行键
00001011	013	11	0B	VT (Vertical Tab)	垂直制表符
00001100	014	12	0C	FF/NP (Form Feed/New Page)	换页键
00001101	015	13	0D	CR (Carriage Return)	回车键
00001110	016	14	0E	SO (Shift Out)	不用切换
00001111	017	15	0F	SI (Shift In)	启用切换
00010000	020	16	10	DLE (Data Link Escape)	数据链路转义
00010001	021	17	11	DC1/XON (Device Control 1/Transmission On)	设备控制1/传输开始
00010010	022	18	12	DC2 (Device Control 2)	设备控制2
00010011	023	19	13	DC3/XOFF (Device Control 3/Transmission Off)	设备控制3/传输中断
00010100	024	20	14	DC4 (Device Control 4)	设备控制4
00010101	025	21	15	NAK (Negative Acknowledge)	无响应/非正常响应/拒绝接收
00010110	026	22	16	SYN (Synchronous Idle)	同步空闲
00010111	027	23	17	ETB (End of Transmission Block)	传输块结束/块传输终止
00011000	030	24	18	CAN (Cancel)	取消
00011001	031	25	19	EM (End of Medium)	已到介质末端/介质存储已满/介质中断
00011010	032	26	1A	SUB (Substitute)	替补/替换
00011011	033	27	1B	ESC (Escape)	逃离/取消
00011100	034	28	1C	FS (File Separator)	文件分割符
00011101	035	29	1D	GS (Group Separator)	组分隔符/分组符
00011110	036	30	1E	RS (Record Separator)	记录分离符
00011111	037	31	1F	US (Unit Separator)	单元分隔符
00100000	040	32	20	(Space)	空格

00100001	041	33	21	!	
00100010	042	34	22	"	
00100011	043	35	23	#	
00100100	044	36	24	$	
00100101	045	37	25	%	
00100110	046	38	26	&	
00100111	047	39	27	'	
00101000	050	40	28	(	
00101001	051	41	29	)	
00101010	052	42	2A	*	
00101011	053	43	2B	+	
00101100	054	44	2C	,	
00101101	055	45	2D	-	
00101110	056	46	2E	.	
00101111	057	47	2F	/	
00110000	060	48	30	0	
00110001	061	49	31	1	
00110010	062	50	32	2	
00110011	063	51	33	3	
00110100	064	52	34	4	
00110101	065	53	35	5	
00110110	066	54	36	6	
00110111	067	55	37	7	
00111000	070	56	38	8	
00111001	071	57	39	9	
00111010	072	58	3A	:	
00111011	073	59	3B	;	
00111100	074	60	3C	<	

00111101	075	61	3D	=	
00111110	076	62	3E	>	
00111111	077	63	3F	?	
01000000	100	64	40	@	
01000001	101	65	41	A	
01000010	102	66	42	B	
01000011	103	67	43	C	
01000100	104	68	44	D	
01000101	105	69	45	E	
01000110	106	70	46	F	
01000111	107	71	47	G	
01001000	110	72	48	H	
01001001	111	73	49	I	
01001010	112	74	4A	J	
01001011	113	75	4B	K	
01001100	114	76	4C	L	
01001101	115	77	4D	M	
01001110	116	78	4E	N	
01001111	117	79	4F	O	
01010000	120	80	50	P	
01010001	121	81	51	Q	
01010010	122	82	52	R	
01010011	123	83	53	S	
01010100	124	84	54	T	
01010101	125	85	55	U	
01010110	126	86	56	V	
01010111	127	87	57	W	
01011000	130	88	58	X	

01011001	131	89	59	Y	
01011010	132	90	5A	Z	
01011011	133	91	5B	[	
01011100	134	92	5C	\	
01011101	135	93	5D	]	
01011110	136	94	5E	^	
01011111	137	95	5F	_	
01100000	140	96	60	`	
01100001	141	97	61	a	
01100010	142	98	62	b	
01100011	143	99	63	c	
01100100	144	100	64	d	
01100101	145	101	65	e	
01100110	146	102	66	f	
01100111	147	103	67	g	
01101000	150	104	68	h	
01101001	151	105	69	i	
01101010	152	106	6A	j	
01101011	153	107	6B	k	
01101100	154	108	6C	l	
01101101	155	109	6D	m	
01101110	156	110	6E	n	
01101111	157	111	6F	o	
01110000	160	112	70	p	
01110001	161	113	71	q	
01110010	162	114	72	r	
01110011	163	115	73	s	
01110100	164	116	74	t	

01110101	165	117	75	u	
01110110	166	118	76	v	
01110111	167	119	77	w	
01111000	170	120	78	x	
01111001	171	121	79	y	
01111010	172	122	7A	z	
01111011	173	123	7B	{	
01111100	174	124	7C	|	
01111101	175	125	7D	}	
01111110	176	126	7E	~	
01111111	177	127	7F	DEL (Delete)	删除

对控制字符的解释

ASCII 编码中第 0~31 个字符（开头的 32 个字符）以及第 127 个字符（最后一个字符）都是不可见的（无法显示），但是它们都具有一些特殊功能，所以称为控制字符（Control Character）或者功能码（Function Code）。

这 33 个控制字符大都与通信、数据存储以及老式设备有关，有些在现代电脑中的含义已经改变了。

> 有些控制符需要一定的计算机功底才能理解，初学者可以跳过，选择容易的理解即可。

下面列出了部分控制字符的具体功能：

- **NUL (0)**

  NULL，空字符。空字符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。

  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 被用于C语言中，表示字符串的结束，当一个字符串中间出现 NUL 时，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。

- **SOH (1)**

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。

- **STX (2) 和 ETX (3)**

  STX 表示 Start Of Text，意思是“文本开始”；ETX 表示 End Of Text，意思是“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。

  而中间具体传输的数据内容，ASCII 并没有去定义，它和你所用的传输协议有关。

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

- **BEL (7)**

  BELl，响铃。在 ASCII 编码中，BEL 是个比较有意思的东西。BEL 用一个可以听得见的声音来吸引人们的注意，既可以用于计算机，也可以用于周边设备（比如打印机）。

  注意，BEL 不是声卡或者喇叭发出的声音，而是蜂鸣器发出的声音，主要用于报警，比如硬件出现故障时就会听到这个声音，有的计算机操作系统正常启动也会听到这个声音。蜂鸣器没有直接安装到主板上，而是需要连接到主板上的一种外设，现代很多计算机都不安装蜂鸣器了，即使输出 BEL 也听不到声音，这个时候 BEL 就没有任何作用了。

- **BS (8)**

  BackSpace，退格键。退格键的功能，随着时间变化，意义也变得不同了。

  退格键起初的意思是，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。

- **HT (9)**

  Horizontal Tab，水平制表符，相当于 Table/Tab 键。

  水平制表符的作用是用于布局，它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，只不过在多数设备上制表符 Tab 都预定义为 4 个空格的宽度。

  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 4 个空格。

- **LF (10)**

  Line Feed，直译为“给打印机等喂一行”，也就是“换行”的意思。LF 是 ASCII 编码中常被误用的字符之一。

  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C语言和 Unix 操作系统将 LF 的含义重新定义为“新行”，即 LF 和 CR 的组合效果，也就是回车且换行的意思。

  从程序的角度出发，C语言和 Unix 对 LF 的定义显得更加自然，而 MS-DOS 的实现更接近于 LF 的本意。

  现在人们常将 LF 用做“新行（newline）”的功能，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。

- **VT (11)**

  Vertical Tab，垂直制表符。它类似于水平制表符 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 控制符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 VT，因为一般在换行的时候都是用 LF 代替 VT 了。

- **FF (12)**

  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符所表现的行为各不同，有些会清除屏幕，有些只是显示 `^L` 字符，有些只是新换一行而已。例如，Unix/Linux 下的 Bash Shell 和 Tcsh 就把 FF 看做是一个清屏幕的命令。

- **CR (13)**

  Carriage return，回车，表示机器的滑动部分（或者底座）返回。

  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的意思。随着时间的流逝，后来人们把 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，人们按下 Enter 的同时，也希望把光标移动到下一行，因此C语言和 Unix 重新定义了 CR 的含义，将其表示为移动到下一行。当输入 CR 时，系统也常常隐式地将其转换为LF。

- **SO (14) 和 SI (15)**

  SO，Shift Out，不用切换；SI，Shift In，启用切换。

  早在 1960s 年代，设计 ASCII 编码的美国人就已经想到了，ASCII 编码不仅仅能用于英文，也要能用于外文字符集，这很重要，定义 Shift In 和 Shift Out 正是考虑到了这点。

  最开始，其意为在西里尔语和拉丁语之间切换。西里尔语 ASCII（也即 KOI-7 编码）将 Shift 作为一个普通字符，而拉丁语 ASCII（也就是我们通常所说的 ASCII）用 Shift 去改变打印机的字体，它们完全是两种含义。

  在拉丁语 ASCII 中，SO 用于产生双倍宽度的字符（类似于全角），而用 SI 打印压缩的字体（类似于半角）。

- **DLE (16)**

  Data Link Escape，数据链路转义。

  有时候我们需要在通信过程中发送一些控制字符，但是总有一些情况下，这些控制字符被看成了普通的数据流，而没有起到对应的控制效果，ASCII 编码引入 DLE 来解决这类问题。

  如果数据流中检测到了 DLE，数据接收端会对数据流中接下来的字符另作处理。但是具体如何处理，ASCII 规范中并没有定义，只是弄了个 DLE 去打断正常的数据流，告诉接下来的数据要特殊对待。

- **DC1 (17)**

  Device Control 1，或者 XON – Transmission on。

  这个 ASCII 控制符尽管原先定义为 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，在通信被控制符 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 控制符，它可以将

些由于终端或者主机方面，由于偶尔出现的错误的 XOFF 控制符而中断的通信解锁，使其正常通信。

- **DC3 (19)**

  Device Control 3，或者 XOFF（Transmission off，传输中断）。

  EM (25)

  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。

- **FS(28)**

  File Separator，文件分隔符。FS 是个很有意思的控制字符，它可以让我们看到 1960s 年代的计算机是如何组织的。

  我们现在习惯于随机访问一些存储介质，比如 RAM、磁盘等，但是在设计 ASCII 编码的那个年代，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，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。

- **GS(29)**

  Group Separator，分组符。

  ASCII 定义控制字符的原因之一就是考虑到了数据存储。

  大部分情况下，数据库的建立都和表有关，表包含了多条记录。同一个表中的所有记录属于同一类型，不同的表中的记录属于不同的类型。

  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 Excel 表格，ASCII 时代的人把它叫做组。

- **RS(30)**

  Record Separator，记录分隔符，用于分隔一个组或表中的多条记录。

- **US(31)**

  Unit Separator，单元分隔符。

  在 ASCII 定义中，数据库中所存储的最小的数据项叫做单元（Unit）。而现在我们称其字段（Field）。单元分隔符 US 用于分割串行数据存储环境下的不同单元。

现在的数据库实现都要求大部分类型都拥有固定的长度，尽管有时候可能用不到，但是对于每一个字段，却都要分配足够大的空间，用于存放最大可能的数据。

这种做法的弊端就是占用了大量的存储空间，而 US 控制符允许字段具有可变的长度。在 1960s 年代，数据存储空间很有限，用 US 将不同单元分隔开，能节省很多空间。

- **DEL (127)**

  Delete，删除。

  有人也许会问，为何 ASCII 编码中其它控制字符的值都很小（即 0~31），而 DEL 的值却很大呢（为 127）？

  这是由于这个特殊的字符是为纸带而定义的。在那个年代，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 1111（所有 7 个比特位都是1），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除掉了，就起到了删除的作用。

想了解扩展 ASCII 编码的读者请转到：扩展 ASCII 码对照表