与计算机打交道，字符编码问题是永远需要面对的，我决定花点时间把各种字符集、编码搞清楚。

> 如果你是一个生活在2003年的程序员，却不了解字符、字符集、编码和Unicode这些基础知识。那你可要小心了，要是被我抓到你，我会让你在潜水艇里剥六个月洋葱来惩罚你。

这个邪恶的恐吓是 Joel Spolsky 在十几年前发出的。不幸的是，现在仍有许多程序员不能完全理解Unicode，以及Unicode，UTF-8，UTF-16之间的区别，包括我自己，写完这篇文章，我也感慨编码这件事没有那么简单，但是希望先明确一些概念，捞一些重点。

在说明URL编码之前，先回顾编码和字符集的相关基础知识，再引出为什么URL中需要做编码，以及如何编码的。

一、编码和解码

众所周知，**程序中的所有信息都是以二进制的形式存储在计算机的底层**，也就是说我们在代码中定义的一个 char 字符或者一个 int 整数都会被转换成二进制码储存起来，这个过程可以被称为编码，而将计算机底层的二进制码转换成屏幕上有意义的字符（如 hello world ），这个过程就称为解码。

想要解码，自然需要知道规则。

二、字符集

在计算机中字符的编解码就涉及到字符集（ Character Set ）这个概念，他就相当于能够将一个字符与一个整数一一对应的一个映射表，常见的字符集有 ASCII、Unicode 等。

有了这个字符集，我们就可以知道编码和解码的规则。

三、ASCII编码

历史中的很长一段时间里，计算机仅仅应用在欧洲的一些发达国家，因此在他们的程序中只存在他们所理解的拉丁字母（如a、b、c、d等）和阿拉伯数字，他们在编码解码时也只需要考虑这一种情况，就是如何将这些字符转换成计算机所能理解的二进制数，此时 ASCII 字符集应运而生，他们在编码时只需要对照着 ASCII 字符集，每当在程序中遇到字符 a 时，就会相应的找到其中 a 对应的 ASCII 值 97 然后以二进制形式存起来即可。

ASCII 字符集支持 128 种字符，仅使用 7 个 bit 位，也就是一个字节的后 7 位就可以将它们全部表示出来，而最前面的一位统一规定为 0 即可（如 0110 0001 表示 a），对应表如下：

ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符
0	NUT	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	/	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

因此ASCII 字符集包含的范围很小，仅包括了控制字符（回车键、退格、换行键等）和可显示字符（英文大小写字符、阿拉伯数字和西文符号）。

后来，为了能够表示更多的欧洲国家的常用字符如法语中带符号的字符 `é`，又制定了 ASCII 额外扩展的版本 EASCII，这里就可以使用一个完整子节的 8 个 bit 位表示共 256 个字符，其中就又包括了一些衍生的拉丁字母。

显然，在ASCII 编码中，一个字母只需要占用一个字节。

四、GB2312字符集

ASCII 字符集支持范围太小了，不能满足需求，我们国家也为此做了扩展，最具代表性的就是国内的 GB 类的汉字编码模式，**这种模式规定：ASCII 值小于 127 的字符的意义与原来 ASCII 集中的字符相同，**但当两个 ASCII 值大于 127 的字符连在一起时，就表示一个简体中文的汉字**，前面的一个字节（高字节）从 0xA1 拓展到 0xF7，后面一个字节（低字节）从 0xA1 到 0xFE，这样就可以组合出了大约 7000 多个简体汉字了。

为了在解码时操作的统一，GB 类编码表中还也加入了数学符号、罗马希腊的字母、日文的假名等，连在 ASCII 里本来就有的数字、标点、字母都统一重新表示为了两个字节长的编码，这就是我们常说的 “全角” 字符，而原来在 127 号以下的那些就叫 “半角” 字符了，这种编码规则就是后来的 GB2312。

因此我们可以知道，**在GB2312字符集中，一个汉字要占两个字节。**

五、GBK编码的由来

不过中国的汉字又实在是太多了，**人们很快就发现 GB2312 字符集只能够显示那点汉字明显不够**（如中国前总理朱镕基的 “镕” 字并不在 GB2312 字符集中），于是专家们又继续把 GB2312 没有用到的码位使用到其他没有被收录的汉字中。

后来还是不够用，**于是干脆规定只要第一个字节是大于 127 就固定表示这是一个汉字的开始**，不管后面跟的是不是扩展字符集里的内容。结果扩展之后的编码方案被称为 GBK 标准，**GBK 包括了 GB2312 的所有内容**，同时又增加了近 20000 个新的汉字（包括繁体字）和符号。

显然，**GBK是咱们国家基于GB2312字符集的扩展，一个汉字仍然占用两个字节。**

六、GBK的新问题

当时的各个国家都像中国这样制定出了一套自己的编码标准，之后当我们需要使用计算机与国际接轨时，问题出现了！国家与国家之间谁也不懂谁的编码，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，在俄语编码中又会代表另一个符号。但是所有这些编码方式中，0–127 表示的符号依然都是一样的，因为他们都兼容 ASCII 码，这一点，如今也是一样。

自然而然地，就需要有一个世界能统一使用的字符集才行。

七、Unicode字符集

正如上面所说，世界上各国都有不同的编码方式，同一个二进制数字可以被解码成不同的符号。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。

为了解决这个问题，**最终的集大成者 Unicode 字符集出现了**，它将世界上所有的符号都纳入其中，目前，Unicode 字符集中已经收录超过 13 万个字符（第十万个字符在2005年获采纳）。值得关注的是，Unicode 依然兼容 ASCII，即 0～127 意义依然不变。

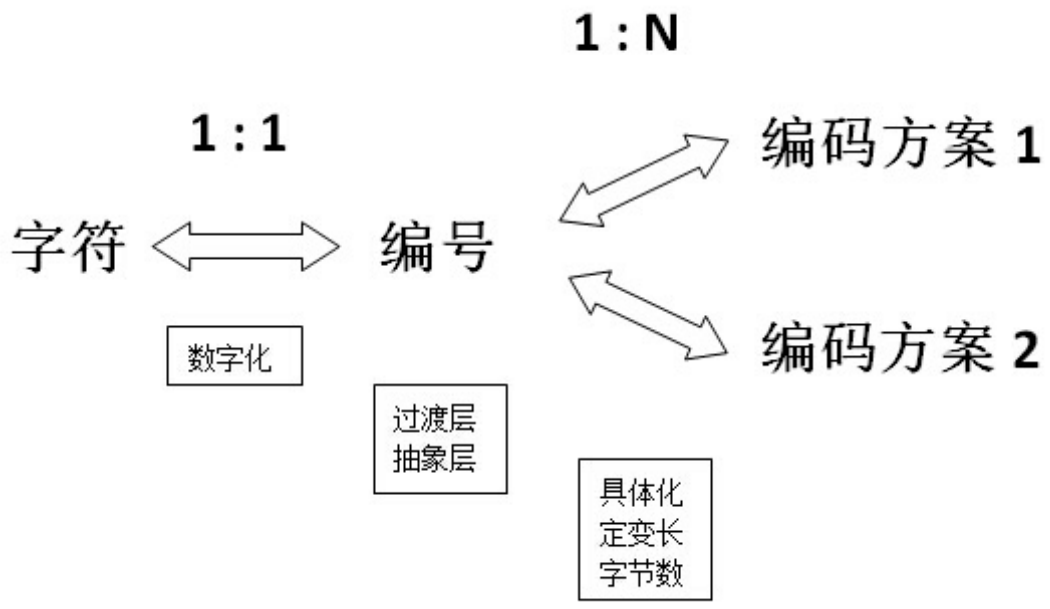
ASCII 码中有个编号，比如 a 对应的 ASCII 值 97。同样地，Unicode 字符集中也有唯一表示某个字符的标识，叫做**码点**（Code Point），如码点 U+0061，这里的 61 就是 97 的十六进制表示，它就表示 Unicode 字符集中的字符 'a'。

码点的表示的形式为 U+[XX]XXXX，X 代表一个十六制数字，一般可以有 4-6 位，不足 4 位前补 0 补足 4 位，超过则按是几位就是几位，具体范围是 U+0000~U+10FFFF，**大概是 111 万**。按 Unicode 官方的说法，码点范围就这样了，以后也不扩充了，一百多万足够用了，目前也只定义了 11 万多个字符左右。

码点从0开始，为每个符号指定一个编号，这叫做”码点”（code point）。比如，码点 0 的符号就是 null（表示所有二进制位都是 0）。

```
U+0000 = null
```

整个编码过程中码点就作为了一个中间的过渡层，可用下面这张图来表示：



从这张图可以看出，整个解码可分为两个过程。首先，**将程序中的字符根据字符集中的编号数字化为某个特定的数值，然后根据编号以特定的方式存储到计算机中**。

显然，这时候我们就可以发现编号并不是最终存储在计算机中的结果。按照之前的理解，编码即把一个字符编码为一个二进制数字存储起来，然而这种表述并不准确，真正的编码不止这么简单，这其中还涉及了每个数字用几个字节表示，是用定长还是变长表示等具体细节。

举个例子，字符 a 的码点为 U+0061（十进制为 97），那么这个 U+0061 该如何存储，单纯的表示 U+0061 可以直接使用 7 位的二进制数 110 0001 表示，但在 GB 类的编码模式中就需要以两个字节存储即 0000 0000 0110 0001（空位用 0 填充）。

八、字符集不等于编码

Unicode 字符集衍生出来的编码方案有三种，分别是 UTF-32、UTF-16 和 UTF-8，这使他与之前的编码模式不同，因为 ASCII、GBK 等类编码模式的字符集和编码方式都是一一对应的，**而 Unicode 的编码实现却有三种**，这就是我们需要区分字符集与编码的原因之一，因为此时 Unicode 并不特指 UTF-8 或者 UTF-32。

九、UTF-8是如何编码和解码的

下面，我们来看下面这张示意图，探究各种编码模式下，码点是如何具体转换成各种编码的：

	H	æ	你	𝄞	字符层面
码点 (十进制)	U+0048 (72)	U+00E6 (230)	U+4F60 (20320)	U+1D11E (119070)	抽象编码层面
UTF-8	48	C3 A6	E4 BD A0	F0 9D 84 9E	具体编码层面
UTF-16	00 48	00 E6	4F 60	D8 34 DD 1E	
UTF-32	00 00 00 48	00 00 00 E6	00 00 4F 60	00 01 D1 1E	

这其中又涉及到编码过程中定长与变长两种实现方式，这里的 `UTF-32` 就属于定长编码，即永远用 4 字节存储码点，而 `UTF-8` 、`UTF-16` 就属于变长存储，`UTF-8` 根据不同的情况使用 1-4 字节，而 `UTF-16` 使用 2 或 4 字节来存储码点。

由于UTF-32是定长，固定4个字节一组，这样解码的时候会很简单，不过最大缺点是占用空间太大，因为不管都大的码点都需要四个字节来存储，非常的占空间。

UTF-8 属于变长的编码方式，它可以由 1，2，3，4 四种字节组合，必然需要有个规则来确定到底是哪种长度才能正确解码，它使用的是高位保留的方式来区别不同变长，具体方式如下：

Unicode 码点范围（十六进制）	UTF-8 编码方式（二进制）	字节数
0000 0000 ~ 0000 007F	0xxxxxxx	一个字节
0000 0080 ~ 0000 07FF	110xxxxx 10xxxxxx	二个字节
0000 0800 ~ 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx	三个字节
0001 0000 ~ 0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	四个字节

在 UTF-8 编码方式下，编码字符时就简单了，我们可以先拿到某个汉字的码点，然后根据码点判断所在的范围，确认需要几个字节的编码形式来存储。下面以汉字“汉”为例，具体说明如何进行 UTF-8 编码和解码。

“汉”的 Unicode 码点是 0x6c49（110 1100 0100 1001），通过上面的对照表可以发现，0x0000 6c49 位于第三行的范围，那么得出其格式为 1110xxxx 10xxxxxx 10xxxxxx。接着，从“汉”的二进制数最后一位开始，从后向前依次填充对应格式中的 x，多出的 x 用 0 补上。这样，就得到了“汉”的 UTF-8 编码为 11100110 10110001 10001001，转换成十六进制就是 0xE6 0xB7 0x89。

解码 UTF-8 编码也很简单了，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。

关于 UTF-16 ，它使用的是一种变长为 2 或 4 字节编码模式。它的编码和方式也略有不同，利用了代理的思想达到跟 UTF-8 中高位保留的目的，实际上我到现在也没有接触过这个 UTF-16 和 UTF-32 ，因此不加以深入研究了。

总而言之，在 Unicode 字符集中，就是根据码点和选择的编码方案可以方便地进行编码和解码。

十、URL编码

复习了一波字符集、编码、解码的问题，是因为我在看URL编码和解码，想到了上面的一些概念，索性就一次性都搞定。

在HTTP中，如果某个信息需要编码，说明它不适合直接传输，原因多种多样，如Size过大，包含隐私数据，对于Url来说，之所以要进行编码，可能是因为Url中有些字符会引起歧义。

一般来说，URL只能使用英文字母、阿拉伯数字和某些标点符号，不能使用其他文字和符号。比如，世界上有英文字母的网址 `http://www.abc.com` ，但是没有希腊字母的网址 `http://www.aβγ.com` 。这是因为网络标准 RFC 1738 做了硬性规定：

> "...Only alphanumerics [0-9a-zA-Z], the special characters "$-_.+!*'()," [not including the quotes - ed], and reserved characters used for their reserved purposes may be used unencoded within a URL."
>
> "只有字母和数字[0-9a-zA-Z]、一些特殊符号"$-_.+!*'(),"[不包括双引号]、以及某些保留字，才可以不经过编码直接用于URL。"

我们也知道，Http协议中参数的传输是 `key=value` 这种键值对形式的，如果要传多个参数就需要用 `&` 符号对键值对进行分割。

如 `?name1=value1&name2=value2` ，这样在服务端在收到这种字符串的时候，会用 `&` 分割出每一个参数，然后再 `=` 来分割出参数值。我们说下实际解析过程。上述字符串在计算机中用ASCII码表示为：

我们对比 ASCII 编码可以知道，`=` 是61，对应16进制就是3D， `&` 是38，对应16进制是26，当代码一个字节一个字节吃的时候，吃到3D这字节后，服务端就知道前面吃的字节表示一个key，再向后吃，如果遇到26，说明从刚才吃的3D到26字节之间的是上一个key的value，以此类推就可以解析出客户端传过来的参数。

明白了逻辑后，问题就来了，如果我的参数值中就包含=或&这种特殊字符的时候该怎么办?

比如说 `name1=value1` ,其中 `value1` 的值是 `va&lu=e1` 字符串，那么实际在传输过程中就会变成这样 `name1=va&lu=e1` 。我们的本意是就只有一个键值对，但是服务端会解析成两个键值对，这样就产生了歧义。

这个就属于上面说的不适合直接传输的一个情形，可以通过URL编码来解决。

Url编码通常也被称为百分号编码。为什么这么叫呢？我们来看下它如何编码的，就清楚了。

对于 ASCII 字符，比如 `a` ，对应编号是61，那么URL编码之后的结果是 `%61` ，假设我们在地址栏上输入的是 `http://g.cn/search?q=abc` ，那么等同于在 google 中搜索 abc 了。那么上面的问题， `name1=va&lu=e1` 就会被编码为 `name1=va%26lu%3D` 。我们应该知道为什么叫他为百分号编码了。

并且经过学习，我们知道 ASCII 字符是不支持中文的。我们先在浏览器上输入中文看看是什么情况，比如我在谷歌浏览器上输入：

> http://zh.wikipedia.org/wiki/春节

看了下请求头：

"春"和"节"的utf-8编码分别是 E6 98 A5 和 E8 8A 82 ，因此，%E6%98%A5%E8%8A%82 就是按照顺序，在每个字节前加上%而得到的。可以看出来，这里使用utf-8编码。

再试下 ？ 这种带参数的访问形式，拿 http://www.oursnail.cn:8080/fossi-shop/ 再来测试下：

可以看到，**在我的 win10 的谷歌浏览器上实验，对于中文默认使用的是 UTF-8 编码**。对于 Unicode 字符， RFC 文档建议使用 utf-8 对其进行编码得到相应的字节，然后对每个字节执行百分号编码。如"中文"使用 UTF-8 字符集得到的字节为 0xE4 0xB8 0xAD 0xE6 0x96 0x87 ，经过Url编码之后得到 %E4%B8%AD%E6%96%87。

好了，关于URL编码就说到这里。