一文透彻掌握 Python 编码问题

一、当我说字符时，我在说什么？

当我们提起字符串时，每个程序员都能理解到，我们说的是一个字符序列。但是，当我们说字符时，很多人就困惑了。

写在纸上的字符很容易辨识，但是为了将不同的字符在计算机中标识出来，人类发明了unicode字符。简单讲，unicode可以看成是一个标准的函数，它将一个具体的字符映射成0-1114111之间的一个数字，这个数字叫做码位。通常，码位用十六进制表示，并且前面会加上“U+”的字样。例如，字母A的码位是U+0041。

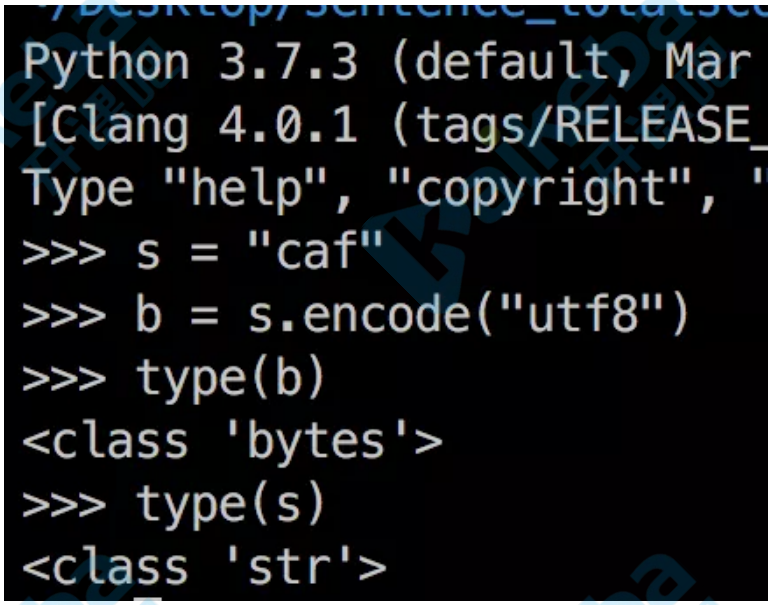
按道理说，我们在计算机中，用unicode的码位来代表字符就很完美了。实际上，python3中的str对象和python2中的unicode对象在内存中就是用码位来表示字符的。但是，由于全世界的字符比较多，导致表示码位的数字也要用long或者int这样的数据类型表示，每个字符都要占固定的几个字节。在存储到磁盘或者通过网络进行传输时，比较浪费空间。于是，聪明的人类又搞了一个函数，这个函数将一个码位映射成字节序列。映射的目的是减少占用的空间。这个函数就是编码。也就是说，编码是在码位和字节序列之间转换时使用的算法。

比如大写字母A（U+0041)，使用UTF-8编码后是\x41，这里\x表示一个字节，字节的值是41。我们看到，如果用U+0041 这个整数代表大写字母A，需要4个字节，因为一个整数就是用4个字节表示的，经过编码后，只占用了一个字节，达到了减少减少空间的目的。

这里提示一下，下文中，当我们再说到码位时，可以将其简单想象成一个int。

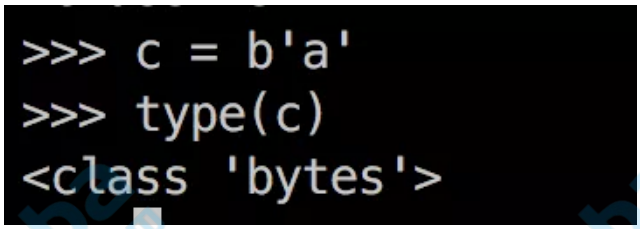
二、python3中码位和编码是如何表示的

在python3的代码中，str类型的对象就是用码位表示的字符串， 编码后的字节序列可以用bytes类型的对象表示。如下所示：

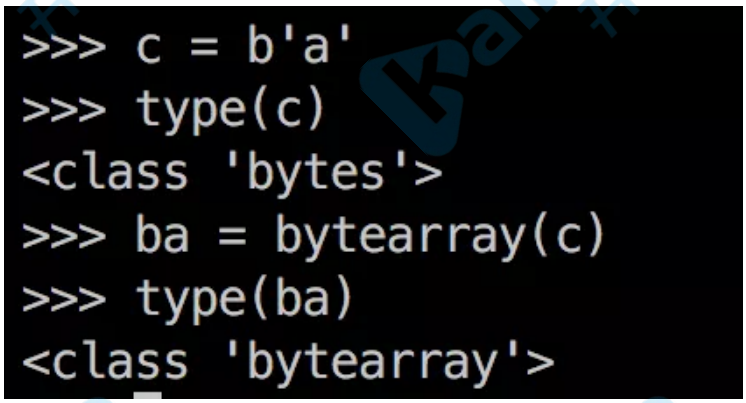


可以将bytes类型的对象看成一个数组，切片啥的都不在话下，里面的元素是介于0-255（含）之间的整数。从python2.6起，新增一个类似的类型，bytearray。它和bytes很像，不同之处有以下两点：

1、没有字面量句法，看图：



上图是bytes对象的字面量创建方法。bytearray没有类似的构造方法，它只能这样获得：



2、bytes不可变，bytearray可变

当我们print一个bytes对象时，常常会看到这种情况：b'caf\xc3\t'看起来有点乱，让我们来观察一下：

b表示你print的是bytes对象

- caf表示三个字节，这几个字节中的值就是caf三个字符的ascii码值，这里直接用caf三个字符表示了。
- \xc3表示这个字节中的值是十六进制的c3，无法用ascii码值表示，所以这里用了两个字节的十六进制数表示。
- \t表示，这个字节的值是tab字符，这里就用转义字符来表示了。

三、python中的编解码器

python有100多种编解码器！第一次知道这个消息，我很震惊，人类真是喜欢折腾啊。

下面，让我们一起来欣赏一下几个常用的编解码器对一些字符的编码：

图 4-1 展示了不同编解码器对“A”和高音谱号等字符编码后得到的字节序列。注意，后3种是可变长度的多字节编码。

字符	码位	ascii	latin1	cp1252	cp437	gb2312	utf-8	utf-16le
A	U+0041	41	41	41	41	41	41	41 00
¿	U+00BF	*	BF	BF	A8	*	C2 BF	BF 00
Ã	U+00C3	*	C3	C3	*	*	C3 83	C3 00
á	U+00E1	*	E1	E1	A0	A8 A2	C3 A1	E1 00
Ω	U+03A9	*	*	*	EA	A6 B8	CE A9	A9 03
ڿ	U+06BF	*	*	*	*	*	DA BF	BF 06
'	U+201C	*	*	93	*	A1 B0	E2 80 9C	1C 20
€	U+20AC	*	*	80	*	*	E2 82 AC	AC 20
┌	U+250C	*	*	*	DA	A9 B0	E2 94 8C	0C 25
气	U+6C14	*	*	*	*	C6 F8	E6 B0 94	14 6C
氣	U+6C23	*	*	*	*	*	E6 B0 A3	23 6C
𝄞	U+1D11E	*	*	*	*	*	F0 9D 84 9E	34 D8 1E DD

图 4-1：12 个字符，它们的码位及不同编码的字节表述（十六进制，星号表明该编码不支持表示该字符）

注：这些编解码器通常用在open(),str.encode(),bytes.decode()等函数中。最常见的编解码器肯定是utf-8。它还有几个别名，即 utf_8, utf8, U8。最好还是熟悉下这几个别名。

四、处理常见的编解码错误

在用python进行编解码时，经常发生各种错误。很多人的办法就是各种google各种试，搞定之后就不再管了。我自己之前就是这样。但更系统的办法就是理解常见的错误类型，在遇到时可以按步骤地去解决问题。下面我们就来看看常见的三类错误。

- **UnicodeEncoderError**

当你用了某个编码器将unicode字符进行编码输出时，如果这个编码器中没有包含某些要编码的unicode字符，就会发生UnicodeEncoderError。这种情况下，通常需要去改变所用的编码器。简单讲就是在将unicode进行encode时发生了error

- **UnicodeDecodeError**

在将一个字节序列用指定的解码器解码成unicode时，如果这个字节序列不符合解码器的要求，就会发生UnicodeDecoderError。这里的不符合要求有两种情况，一种是字节序列错误的，一种就是用的解码器不合适。

- **SyntaxError**

python3默认使用UTF-8编码源码，python2则默认使用ASCII。如果加载的.py文件中包含UTF-8之外的数据，而且没有声明编码，就会发生SyntaxError。
处理编解码的最佳实践是，明确指定encoding字段，显式声明所用的编解码器。

五、几种编码默认值的区别

- **locale.getpreferredencoding()**

这个设置是打开文本文件时，默认使用的解码器。如果open()文件时没有指定解码器，并且发生了错误，就要检查一下这个值。如下是在我的电脑上测试的结果

```
>>> import locale
>>> locale.getpreferredencoding()
'cp936'
```

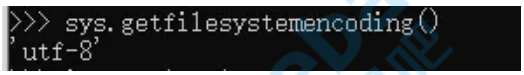
赶紧看看自己的电脑是什么编码吧。

- **sys.getdefaultencoding()**

当在python程序内，在字节序列和字符串之间转换时，默认使用这个编码。python默认的是UTF-8。

- **sys.getfilesystemencoding()**

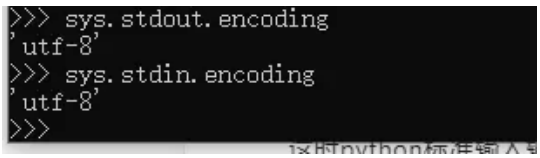
这个是文件名默认的编解码器，注意：不是文件内容，只是文件名称。open()里面传入文件名给python，这时的文件名是unicode字符串，python是用这个编码器对名字进行编码，转成字节序列后再去文件系统中查找的。如下所示，是我电脑上的结果：



注意和上面的locale.getpreferredencoding()进行一下对比哈。

- **sys.stdout.encoding和sys.stdin.coding**

这时python标准输入输出使用的默认编码，在我的电脑上是这样的：



我们经常发现中文输出乱码时，原因要从两头找，一头就是python默认输出时使用的编码器，一头就是显示的控制台使用的解码器，理论上，只要二者一致，就不会发生错误。

六、无总结、不进步

上面所叙述的关于编解码的知识，如果真正掌握，足够应付工作需要了。真正掌握这些知识，还要在实际中遇到问题后，主动用这些知识来帮助查找问题，这样可以很快加深理解。