python中文编码详解

标签： python character transformation windows 数据库 文本编辑

2011-08-10 16:50 8105人阅读 评论(4) 收藏 举报

分类：

Python

部分转自：**http://www.cnblogs.com/huxi/archive/2010/12/05/1897271.html**

编码的历史

1. ASCII

ASCII(American Standard Code forInformation Interchange)，是一种单字节的编码。计算机世界里一开始只有英文，而单字节可以表示256个不同的字符，可以表示所有的英文字符和许多的控制符号。不过ASCII只用到了其中的一半（\x80以下），这也是MBCS得以实现的基础。

Python输出中文时，各种\x开头的编码，如\xe4。完全摸不着头脑

2. MBCS

然而计算机世界里很快就有了其他语言，单字节的ASCII已无法满足需求。后来每个语言就制定了一套自己的编码，由于单字节能表示的字符太少，而且同时也需要与ASCII编码保持兼容，所以这些编码纷纷使用了多字节来表示字符，如GBxxx、BIGxxx等等，他们的规则是，如果第一个字节是\x80以下，则仍然表示ASCII字符；而如果是\x80以上，则跟下一个字节一起（共两个字节）表示一个字符，然后跳过下一个字节，继续往下判断。

这里，IBM发明了一个叫Code Page的概念，将这些编码都收入囊中并分配页码，GBK是第932页，也就是CP932。所以，也可以使用CP932表示GBK。

MBCS(Multi-Byte Character Set)是这些编码的统称。目前为止大家都是用了双字节，所以有时候也叫做DBCS(Double-Byte Character Set)。必须明确的是，MBCS并不是某一种特定的编码，Windows里根据你设定的区域不同，MBCS指代不同的编码，而Linux里无法使用MBCS作为编码。在Windows中你看不到MBCS这几个字符，因为微软为了更加洋气，使用了ANSI来吓唬人，记事本的另存为对话框里编码ANSI就是MBCS。同时，在简体中文Windows默认的区域设定里，指代GBK。

3. Unicode

后来，有人开始觉得太多编码导致世界变得过于复杂了，让人脑袋疼，于是大家坐在一起拍脑袋想出来一个方法：所有语言的字符都用同一种字符集来表示，这就是Unicode。

最初的Unicode标准UCS-2使用两个字节表示一个字符，所以你常常可以听到Unicode使用两个字节表示一个字符的说法。但过了不久有人觉得256*256太少了，还是不够用，于是出现了UCS-4标准，它使用4个字节表示一个字符，不过我们用的最多的仍然是UCS-2。

UCS(Unicode Character Set)还仅仅是字符对应码位的一张表而已，比如"汉"这个字的码位是6C49。字符具体如

何传输和储存则是由UTF(UCS Transformation Format)来负责。

一开始这事很简单，直接使用UCS的码位来保存，这就是UTF-16，比如，"汉"直接使用\x6C\x49保存(UTF-16-BE)，或是倒过来使用\x49\x6C保存(UTF-16-LE)。但用着用着美国人觉得自己吃了大亏，以前英文字母只需要一个字节就能保存了，现在大锅饭一吃变成了两个字节，空间消耗大了一倍……于是UTF-8横空出世。

UTF-8是一种很别扭的编码，具体表现在他是变长的，并且兼容ASCII，ASCII字符使用1字节表示。然而这里省了的必定是从别的地方抠出来的，你肯定也听说过UTF-8里中文字符使用3个字节来保存吧？4个字节保存的字符更是在泪奔……（具体UCS-2是怎么变成UTF-8的请自行搜索）

另外值得一提的是BOM(Byte Order Mark)。我们在储存文件时，文件使用的编码并没有保存，打开时则需要我们记住原先保存时使用的编码并使用这个编码打开，这样一来就产生了许多麻烦。（你可能想说记事本打开文件时并没有让选编码？不妨先打开记事本再使用文件-> 打开看看）而UTF则引入了BOM来表示自身编码，如果一开始读入的几个字节是其中之一，则代表接下来要读取的文字使用的编码是相应的编码：

BOM_UTF8 '\xef\xbb\xbf'

BOM_UTF16_LE '\xff\xfe'

BOM_UTF16_BE '\xfe\xff'

并不是所有的编辑器都会写入BOM，但即使没有BOM，Unicode还是可以读取的，只是像MBCS的编码一样，需要另行指定具体的编码，否则解码将会失败。

你可能听说过UTF-8不需要BOM，这种说法是不对的，只是绝大多数编辑器在没有BOM时都是以UTF-8作为默认编码读取。即使是保存时默认使用ANSI(MBCS)的记事本，在读取文件时也是先使用UTF-8测试编码，如果可以成功解码，则使用UTF-8解码。记事本这个别扭的做法造成了一个BUG：如果你新建文本文件并输入"姹塭"然后使用ANSI(MBCS)保存，再打开就会变成"汉a"，你不妨试试

python中的编码

1. python中的str和unicode

unicode不算字符串？

python中的字符串有两种：str和unicode。

str和unicode都是basestring的子类。严格意义上说，str其实是字节串，它是unicode经过编码后的字节组成的序列。它们都有两个方法：encode和decode：

那么encode('utf-8')就是指的将unicode转换成utf-8格式了

encode是指将**unicode**转换成其他格式的编码

那是否可以理解成：python中，一切字符的本源是unicode，然后encode()作为“包装”程序把unicode包装成好看的编码；decode()程序则把外包装全脱掉变回本源？

decode是指将其他格式的编码转换成**unicode**

decode和encode示例代码：

```
[python]
```

```
#coding=utf-8

u1 = u'你好'
print repr(u1)
s = u1.encode('utf-8')
print repr(s)
u2 = s.decode('utf-8')
print repr(u2)
```

输出：

```
[plain]
u'\u4f60\u597d'
'\xe4\xbd\xa0\xe5\xa5\xbd'
u'\u4f60\u597d'
```

需要注意的是，对str调用encode()方法是错误的，虽然实际上Python不会抛出异常，而是返回另外一个相同内容但不同id的str；对 unicode调用decode()方法也是这样。

再来看一下str和unicode长度的区别：

```
[python]
#coding=utf-8

s1 = '你好'
s2 = u'你好'
print len(s1)
print len(s2)
```

输出6和2

这是因为，s1其实是经过unicode编码后的字节序列，而UTF-8对"你好"这两个字，每个字都用了三个字节来表示，因而导致它的长度为6。

不对啊，经过试验，我这是2个字节来表示。（我用的python 2.7）

而s2是unicode，"你好"中的每一个字都对应一个unicode字符，所以长度为2。

2. python字符编码声明

不管是str还是unicode，字符串在python内部的表示都是unicode编码，它相当于一种统一的中间编码。所以经常会需要从其他编码成unicode，或者从unicode解码到其他编码。这就意味着，对于代码文件，或者其他内容，不管这些内容的编码是ASCII还是UTF-8，python都会把这些内容转换成它所接受的编码—即unicode。

现在试着把下面代码写到文件中，enc.py：

```
[python]
s1='你好'
s2=u'你好'
print repr(s1)
print repr(s2)
```

执行脚本：python enc.py，出错了：

```
[plain]
File "enc.py", line 1
yntaxError: Non-
ASCII character '\xc4' infile enc.py on line 1, but no encoding declared; seehttp://www.python.org
0263.html for details
```

这是因为python解析器读取代码文件本身时，会试图把它转成unicode。但是上面的代码文件中出现了中文，并且文件头上没有任何的编码声明，python就会试图采用默认的ASCII编码来把中文转成unicode，但是显然，ASCII无法将中文转成unicode，所以它报错了。

我们在文件的第一行加上声明：

```
[python]
#coding=utf-8
```

（注：实际上Python只检查#、coding和编码字符串，其他的字符都是为了美观加上的。另外，Python中可用的字符编码有很多，并且还有许多别名，还不区分大小写，比如UTF-8可以写成u8。参见 http://docs.python.org/library/codecs.html#standard-encodings）

输出：

```
[plain]
'\xe4\xbd\xa0\xe5\xa5\xbd'
u'\u4f60\u597d'
```

另外需要注意的是声明的编码必须与文件实际保存时用的编码一致，否则很大几率会出现代码解析异常。现在的IDE一般会自动处理这种情况，改变声明后同时换成声明的编码保存，但文本编辑器控们需要小心。

OK，再改一下代码：

```
[python]
#coding=utf-8

s1 = '你好'
s2 = u'你好'
print s1
print s2
```

再执行一下，输出了：

```
[plain]
浣犲ソ
你好
```

这里第一个“你好”变成乱码了。这是因为，python解析器看到了UTF-8的编码声明，于是把s1转换从UTF-8转换成了unicode。而调用print的时候，它又试图将由UTF-8解码成unicode的字符串编码成gbk（下面解释为什么会转成gbk）的字符串，这时就出现乱码了。

而s2本来就已经是unicode了，所以不会有影响。

为什么print的时候要转成gbk呢，这里需要解释一下python中print的工作原理：

print只是把字节串传递给操作系统，而由操作系统决定以何种编码输出。对于普通的ASCII字节串，当然是直接输出。而对于unicode，就依赖于stdout的输出编码了。

控制台里默认的编码是gbk（可以通过查看cmd.exe窗口属性看到），所以它会试图把unicode编码成gbk。

（注意不同的编辑器，如vim，IDLE，Eclipse使用的输出编码都是不一致的。所以，在一个地方能正常输出中文，不代码在所有地方都能正常地输出中文）

这里我们再改一下，直接把下面的代码在python的交互式shell中执行：

```python
s1 = '你好'
s2 = u'你好'
print s1
print s2
```

为什么这时又都正确地输出了中文呢？

因为现在是在控制台，默认编码就是gbk。所以对于s1，它会从gbk解码到unicode。输出时再编码回gbk，这时就能正常地输出了。

3. python读写文件

内置的open()方法打开文件时，read()读取的是str，读取后需要使用正确的编码格式进行decode()。write()写入时，如果参数是 unicode，则需要使用你希望写入的编码进行encode()，如果是其他编码格式的str，则需要先用该str的编码进行decode()，转成 unicode后再使用写入的编码进行encode()。如果直接将unicode作为参数传入write()方法，Python将先使用源代码文件声明的字符编码进行编码然后写入。

```python
#coding=utf-8

f = open('in.txt')
line = f.read()
f.close()
print type(line) #输出type 'str'
u1 = line.decode('gbk')
f = open('out.txt', 'w')
line2 = u1.encode('utf-8')
f.write(line2)
f.close()
```

另外，模块codecs提供了一个open()方法，可以指定一个编码打开文件，使用这个方法打开的文件读取返回的将是unicode。写入时，如果参数是unicode，则使用open()时指定的编码进行编码后写入；如果是str，则先根据源代码文件声明的字符编码，解码成unicode后再进行前述操作。相对内置的open()来说，这个方法比较不容易在编码上出现问题。

4. 数据库中使用中文

a. 设置mysql中的charset是UTF-8的，然后在python代码文件中设置#coding=utf-8。

b. 使用MySQLdb连接数据库时，要加上charset='utf-8'的选项，不然它会以默认的编码去读取数据库的内容。

确保了1和2之后，就能正常地读取中文数据了，示例代码见上一篇。