Python的中文编码问题

WuXianglong 104 2014年12月08日 发布

字符串是Python中最常用的数据类型，而且很多时候你会用到一些不属于标准ASCII字符集的字符，这时候代码就很可能抛出UnicodeDecodeError: 'ascii' codec can't decode byte 0xc4 in position 10: ordinal not in range(128)异常。这种异常在Python中很容易遇到，尤其是在Python2.x中，是一个很让初学者费解头疼的问题。不过，如果你理解了Python的Unicode，并在编码中遵循一定的原则，这种编码问题还是比较容易理解和解决的。

字符串在Python内部的表示是unicode编码，因此，在做编码转换时，通常需要以unicode作为中间编码，即先将其他编码的字符串解码（decode）成unicode，再从unicode编码（encode）成另一种编码。但是，Python 2.x的默认编码格式是ASCII，就是说，在没有指定Python源码编码格式的情况下，源码中的所有字符都会被默认为ASCII码。也因为这个根本原因，在Python 2.x中经常会遇到UnicodeDecodeError或者UnicodeEncodeError的异常。

关于Unicode

Unicode是一种字符集，它为每一种现代或古代使用的文字系统中出现的每一个字符都提供了统一的序列号，规定了符号的二进制代码，但没有规定这个二进制代码应该如何存储。也就是说：Unicode的编码方式是固定的，但是实现方式根据不同的需要有跟多种，常见的有UTF-8、UTF-16和UTF-32等。

为了能够处理Unicode数据，同时兼容Python某些内部模块，Python 2.x中提供了Unicode这种数据类型，通过decode和encode方法可以将其它编码和Unicode编码相互转化，但同时也引入了UnicodeDecodeError和UnicodeEncodeError异常。。

常见的几种编码异常

Python中常见的几种编码异常有SyntaxError: Non-ASCII character、UnicodeDecodeError和UnicodeEncodeError等。下面依次举例说明一下：

1、SyntaxError: Non-ASCII character

这种异常最不容易出现，也最容易处理，主要原因是Python源码文件中有非

ASCII字符，而且同时没有声明源码编码格式，例如：

```
s = '中文'
print s      # 抛出异常
```

2、UnicodeDecodeError

这个异常有时候会在调用decode方法时出现，原因是Python打算将其他编码的字符转化为Unicode编码，但是字符本身的编码格式和decode方法传入的编码格式不一致，例如：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
s = '中文'
s.decode('gb2312') # UnicodeDecodeError: 'gb2312' codec can't decode
print s
```

上面这段代码中字符串s的编码格式是utf-8，但是在使用decode方法转化为Unicode编码时传入的参数是‘gb2312’，因此在转化的时候抛出UnicodeDecodeError异常。还有一种情况是在encode的时候：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
s = '中文'
s.encode('gb2312') # UnicodeDecodeError: 'ascii' codec can't decode
print s
```

3、UnicodeEncodeError

错误的使用decode和encode方法会出现这种异常，比如：使用decode方法将Unicode字符串转化的时候：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
s = u'中文'
s.decode('utf-8') # UnicodeEncodeError: 'ascii' codec can't encode c
print s
```

当然，除了上面列出的几种出现异常的情况之外还有很多可能出现异常的例子，这里就不在一一说明了。

解决方法

对于以上的几个异常，有以下几个处理的方法和原则。

1、遵循PEP0263原则，声明编码格式

在PEP 0263 -- Defining Python Source Code Encodings中提出了对Python编码问题的最基本的解决方法：在Python源码文件中声明编码格式，最常见的声明方式如下：

```
#!/usr/bin/python
# -*- coding: <encoding name> -*-
```

其中是代码所需要的编码格式，它可以是任意一种Python支持的格式，一般都会使用utf-8的编码格式。

2、使用 `u'中文'` 替代 `'中文'`

```
str1 = '中文编码'
str2 = u'中文编码'
```

Python中有以上两种声明字符串变量的方式，它们的主要区别是编码格式的不同，其中，str1的编码格式和Python文件声明的编码格式一致，而str2的编码格式则是Unicode。如果你要声明的字符串变量中存在非ASCII的字符，那么最好使用str2的声明格式，这样你就可以不需要执行decode，直接对字符串进行操作，可以避免一些出现异常的情况。

3、Reset默认编码

Python中出现这么多编码问题的根本原因是Python 2.x的默认编码格式是ASCII，所以你也可以通过以下的方式修改默认的编码格式：

```
import sys
sys.setdefaultencoding('utf-8')
```

这种方法是可以解决部分编码问题，但是同时也会引入很多其他问题，得不偿失，不建议使用这种方式。

4、终极原则：decode early, unicode everywhere, encode late

最后分享一个终极原则：decode early, unicode everywhere, encode late，即：在输入或者声明字符串的时候，尽早地使用decode方法将字符串转化成unicode编码格式；然后在程序内使用字符串的时候统一使用unicode格式进行处理，比如字符串拼接、字符串替换、获取字符串的长度等操作；最后，在输出字符串的时候（控制台/网页/文件），通过encode方法将字符串转化为你所想要的编码格式，比如utf-8等。

按照这个原则处理Python的字符串，基本上可以解决所有的编码问题（只要你的代码和Python环境没有问题）。。。

5、升级Python 2.x到3.x

额，最后一个方法，升级Python 2.x，使用Python 3.x版本。。这样说主要是为了吐槽Python 2.x的编码设计问题。当然，升级到Python 3.x肯定可以解决大部分因为编码产生的异常问题。毕竟Python 3.x版本对字符串这部分还是做了相当大的改进的，具体的下面会说。。。。

Python 3.x中的Unicode

在Python 3.0之后的版本中，所有的字符串都是使用Unicode编码的字符串序列，同时还有以下几个改进：

1. 默认编码格式改为unicode
2. 所有的Python内置模块都支持unicode
3. 不再支持u'中文'的语法格式

所以，对于Python 3.x来说，编码问题已经不再是个大的问题，基本上很少遇到上述的几个异常。关于Python 2.x str&unicode和Python 3.x str&bytes的更多说明和对比，大家可以看一下：Python中字符编码的总结和对比

PS: 该文章转自我的博客：Python的中文编码问题

Over!