Yuan先生

随笔 - 1 文章 - 141 评论 - 143

Py 编码的真相

今天让我们一起彻底揭开py编码的真相，包括py2和py3。有同学可能问：以后py3是大势所趋，还有必要了解py2那令人头疼的编码吗？答案是太有必要啦。py2在生产中还是中流砥柱。

什么是编码？

基本概念很简单。首先，我们从一段信息即消息说起，消息以人类可以理解、易懂的表示存在。我打算将这种表示称为“明文”（plain text）。对于说英语的人，纸张上打印的或屏幕上显示的英文单词都算作明文。

其次，我们需要能将明文表示的消息转成另外某种表示，我们还需要能将编码文本转回成明文。从明文到编码文本的转换称为“编码”，从编码文本又转回成明文则为“解码”。

...

py2编码

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。对UTF-8编码的str'苑'使用len()函数时，结果是3，因为utf8编码的'苑' == '\xe8\x8b\x91'。

而unicode是一个字符串，str是unicode这个字符串经过编码（utf8,gbk等）后的字节组成的序列。如上面utf8编码的字符串'汉'。

unicode才是真正意义上的字符串，对字节串str使用正确的字符编码进行解码后获得，并且len(u'苑') == 1。

在Py2里，str = bytes。

py2编码的最大特点是Python 2 将会自动的将bytes数据解码成 unicode 字符串

所以在2里我们可以将字节与字符串拼接。

```
#coding:utf8

print '苑昊' #  苑昊
print repr('苑昊')#'\xe8\x8b\x91\xe6\x98\x8a'

print (u"hello"+"yuan")

#print (u'苑昊'+'最帅')   #UnicodeDecodeError: 'ascii' codec can't decode byte 0xe6
                         # in position 0: ordinal not in range(128)
```

两个问题：

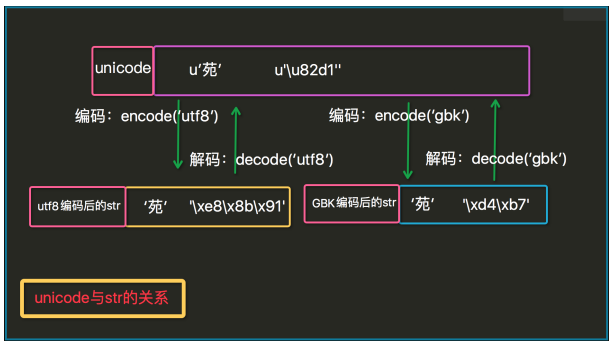
1 print '苑昊' ： 本来存的是'\xe8\x8b\x91\xe6\x98\x8a',为什么显示了 苑昊 的明文?

2 字节串和字符串可以拼接?

这就是那些可恶的 UnicodeError 。你的代码中包含了 unicode 和 byte 字符串，只要数据全部是 ASCII 的话，所有的转换都是正确的，一旦一个非 ASCII 字符偷偷进入你的程序，那么默认的解码将会失效，从而造成 UnicodeDecodeError 的错误。

Python 2 悄悄掩盖掉了 byte 到 unicode 的转换，让程序在处理 ASCII 的时候更加简单。你复出的代价就是在处理非 ASCII 的时候将会失败。

再来看看encode()和decode()两个basestring的实例方法，理解了str和unicode的区别后，这两个方法就不会再混淆了：




```
#coding:utf8

u = u'苑'
print repr(u)  # u'\u82d1'
# print str(u)   #UnicodeEncodeError

s = u.encode('utf8')
print repr(s) #'\xe8\x8b\x91'
print str(s)  #  苑
u2 = s.decode('utf8')
print repr(u2) # u'\u82d1'
```

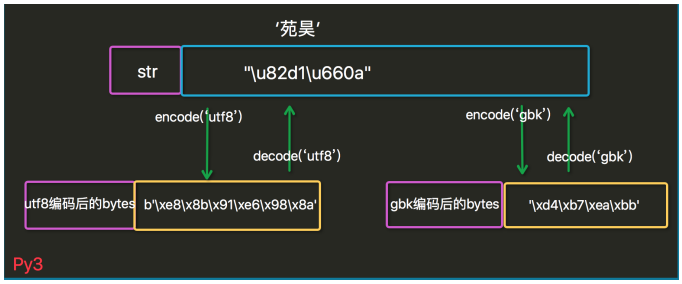
py3编码

python3 renamed the unicode type to str ,the old str type has been replaced by bytes.

跟 Python 2 类似，Python 3 也有两种类型，一个是 Unicode,一个是 byte 码。但是他们有不同的命名。

现在你从普通文本转换成 “str” 类型后存储的是一个 unicode，“bytes” 类型存储的是 byte 串。你也可以通过一个 b 前缀来制造 byte 串。

Python 3最重要的新特性大概要算是对文本和二进制数据作了更为清晰的区分。文本总是Unicode，由str类型表示，二进制数据则由bytes类型表示。Python 3不会以任意隐式的方式混用str和bytes，正是这使得两者的区分特别清晰。你不能拼接字符串和字节包，也无法在字节包里搜索字符串（反之亦然），也不能将字符串传入参数为字节包的函数（反之亦然）。这是件好事。



Python 3 中对 Unicode 支持的最大变化就是将会没有对 byte 字节串的自动解码。如果你想要用一个 byte 字节串和一个 unicode 相链接的话，你将会得到一个错误，不管你包含的内容是什么。

所有这些在 Python 2 中都将会有隐式的处理，而在 Python 3 中你将会得到一个错误。

```
#print('alvin'+u'yuan')#字节串和unicode连接 py2:alvinyuan
print(b'alvin'+'yuan')#字节串和unicode连接 py3:报错 can't concat bytes to str
```

转换：

```
import json

s='苑昊'
print(json.dumps(s))    #"\u82d1\u660a"
b1=s.encode('utf8')
```

```
print(b1,type(b1))      #b'\xe8\x8b\x91\xe6\x98\x8a' <class 'bytes'>

print(b1.decode('utf8'))#苑昊
# print(b1.decode('gbk'))# 鏂戞棴

b2=s.encode('gbk')
print(b2,type(b2))  #'\xd4\xb7\xea\xbb' <class 'bytes'>
print(b2.decode('gbk')) #苑昊
```

注意：无论py2，还是py3,与明文直接对应的就是unicode数据，打印unicode数据就会显示相应的明文(包括英文和中文)

编码实现

说到编码，我们需要在全局掌握这个工作过程，比如我们在pycharm上编写一个.py文件，从保存到运行数据到底是怎么转换的呢?

在解决这个问题之前，我们需要解决一个问题：默认编码

默认编码

什么是默认编码？其实就是你的解释器解释代码时默认的编码方式，在py2里默认的编码方式是ASCII，在py3里则是utf8(sys.getdefaultencoding()查看)。

```
1 #-*- coding: UTF-8 -*-
```

这个声明是做什么的？我们在最开始只知道在py2里如果不加上这么一句话，程序一旦出现中文就会报错，其实就是因为py2默认的ASCII码，对于中文这些特殊字符无法编码；

声明这句话就是告诉python2.7解释器 (默认ACSII编码方式)解释hello.py文件声明下面的内容按utf8编码，对，就是编码(编码成字节串最后转成0101的形式让机器去执行)

大家注意hello.py文件保存时有自己特定的编码方式，比如utf8,比如gbk。

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。所以，保存的编码样式取决于你的编辑器默认的样式（可调）。

文件保存和执行过程

我们讲过，字符串在内存中是以unicode的数据形式保存的，可什么时候我们数据是在内存呢？让我们一起解析这个过程

比如我们在pycharm上(py3.5)创建一个hello.py文件:

```
1 print('hello 苑昊')
```

这个时候我们的数据在内存吗？NO，它已经被pycharm以默认的文件保存编码方式存到了硬盘(二进制数据)，所以一定注意，你点击运行的时候，其实首先需要打开这个文件，然后将所有的数据转移到内存，字符串此时就以unicode的数据格式存到内存的某块地址上（为什么要这样处理一会讲到），其它内容还是utf8的编码方式，然后解释器就可以按着默认的utf8的编码方式逐行解释了。

所以，一旦你的文件保存时的编码与解释器解释的编码不一致时就会出现错误。

print 都做了什么?

在Python 2中，print是一个语句（statement）；而在Python 3中变成了函数（function）。

print语句

在Python 2中，print语句最简单的使用形式就是`print A`，这相当于执行了：

```
1 sys.stdout.write(str(A) + '\n')
```

如果你以逗号为分隔符，传递额外的参数（argument），这些参数会被传递至`str()`函数，最终打印时每个参数之间会空一格。

```
# print A, B, C相当于sys.stdout.write(' '.join(map(str, [A, B, C])) + '\n')。如果print语句的最后再加上

# print A 相当于sys.stdout.write(str(A))
```

print函数

函数定义

从上面的示例代码中我们就可以看出，使用print函数有明显的好处：与使用print语句相比，我们现在能够指定其他的分隔符（separator）和结束符（end string）。

因为我们的目标是编码，所以print函数的好处我们在这就不提了。

所以，无论2或3，对于print我们需要明晰一个方法：str()

py2:str()

```
# class str(object='')
# Return a string containing a nicely printable representation of an object. For
# strings, this returns the string itself. The difference with repr(object) is that
#     str(object) does not always attempt to return a string that is acceptable to
#     eval(); its goal is to return a printable string. If no argument is given,
#     returns the empty string, ''.

# For more information on strings see Sequence Types — str, unicode, list, tuple,
# bytearray, buffer, xrange which describes sequence functionality (strings are
# sequences), and also the string-specific methods described in the String Methods
# section. To output formatted strings use template strings or the % operator described
# in the String Formatting Operations section. In addition see the String Services
# section. See also unicode().
```

py3:str()

```
# class str(object=b'', encoding='utf-8', errors='strict')
#     Return a string version of object. If object is not provided, returns the empty string. Oth
#     whether encoding or errors is given, as follows.
#
#     If neither encoding nor errors is given, str(object) returns object.__str__(), which is the
#     representation of object. For string objects, this is the string itself. If object does not
#     back to returning repr(object).
#
#     If at least one of encoding or errors is given, object should be a bytes-like object (e.g.
#     is a bytes (or bytearray) object, then str(bytes, encoding, errors) is equivalent to bytes.
#      object underlying the buffer object is obtained before calling bytes.decode(). See Binary
#     and Buffer Protocol for information on buffer objects.
#
#     Passing a bytes object to str() without the encoding or errors arguments falls under the fi
#     representation (see also the -b command-line option to Python). For example:
```

常见编码错误

1 cmd下的乱码问题

hello.py

```
#coding:utf8

print ('苑昊')
```

文件保存时的编码也为utf8。

思考：为什么在IDE下用2或3执行都没问题，在cmd.exe下3正确，2乱码呢?

我们在win下的终端即cmd.exe去执行，大家注意，cmd.exe本身就是一个软件；当我们python2 hello.py时，python2解释器(默认ASCII编码)去按声明的utf8编码文件，而文件又是utf8保存的，所以没问题；问题出在当我们print'苑昊'时，解释器这边正常执行，也不会报错，只是print的内容会传递给cmd.exe显示，而在py2里这个内容就是utf8编码的字节数据，而这个软件默认的编码解码方式是GBK，所以cmd.exe用GBK的解码方式去解码utf8自然会乱码。

py3正确的原因是传递给cmd的是unicode数据，符合ISO统一标准的，所以没问题。

```
1 print (u'苑昊')
```

改成这样后，cmd下用2也不会有问题了。

2 print问题

在py2里

```
#coding:utf8
print ('苑昊') #苑昊
print (['苑昊','yuan']) #['\xe8\x8b\x91\xe6\x98\x8a', 'yuan']
```

在py3里

```
print ('苑昊') #苑昊
print (['苑昊','yuan']) #['苑昊', 'yuan']
```

参考文章

http://pycoders-weekly-chinese.readthedocs.io/en/latest/issue5/unipain.html

http://codingpy.com/article/why-print-became-a-function-in-python-3/

Yuan先生
关注 - 1
粉丝 - 3951
我在关注他 取消关注

相关博文:
· Py3编码解码
· py编码终极版
· py编码终极版
· 字符编码py2,py3操作，SecureCRT的会话编码的设置
· 历史的真相
» 更多推荐...

最新 IT 新闻:
· 黎巴嫩首都爆炸能量有多大？物理学家看视频计算：300吨TNT!
· 马斯克到底从特斯拉赚了多少钱？他是最富的穷光蛋
· 特朗普封禁微信，张小龙至少有两张牌可打
· “抖音点赞员”月入1万+，靠谱吗?
· 减持500亿元、死磕马斯克，贝索斯在下一盘大棋?
» 更多新闻...