初探python编码

分类： Python（37）

背景：在实际数据处理中，我们或多或少会接触到中文，如两个dc pack包的diff。使用python对中文数据 处理难免会遇到编码问题。

python里面主要考虑三种编码：

1、源文件编码：

如果我们在源文件中使用中文注释或中文docstring或中文字符串，如不明确指定应使用哪个中文字符集，解释器将无法处理我们的程序。这是因为解释器默认程序使用的是ASCII或ISO-8859-1(即LATIN-1)编码。

解决方法是在文件头部使用coding声明(往往紧跟在#!注释行后面)：

```
#coding: gbk
或# coding=gbk
或# -*- coding: gbk -*-
```

2、内部编码： python内部表示一个字符串有两种方式：一种是普通的str对象(基于字节的字符表示)，另一种是unicode字符 串。它们之间可相互转换：

unicode转其他编码形式的str对象（encode）：

```
>>> unicodestring = u"我爱你"
>>> unicodestring
u'\xce\xd2\xb0\xae\xc4\xe3'
>>> unicodestring.__class__  #另外使用isinstance(unicodestring, unicode)也可以查看是否是unicode字符串

>>> utf8string = unicodestring.encode("utf-8")
>>> utf8string
'\xc3\x8e\xc3\x92\xc2\xb0\xc2\xae\xc3\x84\xc3\xa3'
>>> utf8string.__class__

>>> isostring = unicodestring.encode("ISO-8859-1")
>>> isostring
'\xce\xd2\xb0\xae\xc4\xe3'
>>> isostring.__class__

>>> utf16string = unicodestring.encode("utf-16")
>>> utf16string
'\xff\xfe\xce\x00\xd2\x00\xb0\x00\xae\x00\xc4\x00\xe3\x00'
>>> utf16string.__class__
```

str对象转unicode（decode）：

```
>>> unistring1 = unicode(utf8string, "utf-8")
```

```
>>> unistring1
u'\xce\xd2\xb0\xae\xc4\xe3'
>>> unistring1.__class__

>>> unistring2 = unicode(isostring, "ISO-8859-1")
>>> unistring2
u'\xce\xd2\xb0\xae\xc4\xe3'
>>> unistring2.__class__

>>> unistring3 = unicode(utf16string, "utf-16")
>>> unistring3
u'\xce\xd2\xb0\xae\xc4\xe3'
>>> unistring3.__class__
```

s.decode方法和u.encode方法是最常用的，
简单说来就是，python内部表示字符串用unicode（其实python内部的表示和真实的unicode是有点差别的，对我们几乎透明，可不考虑），和人交互的时候用str对象。
s.decode -------->将s解码成unicode，参数指定的是s本来的编码方式。这个和unicode(s,encodename)是一样的。
u.encode -------->将unicode编码成str对象，参数指定使用的编码方式。
助记：decode to unicode from parameter
encode to parameter from unicode

只有decode方法和unicode构造函数可以得到unicode对象。
上述最常见的用途是比如这样的场景，我们在python源文件中指定使用编码cp936，
coding=cp936或#-*- coding:cp936 -*-或#coding:cp936的方式（不写默认是ascii编码）
这样在源文件中的str对象就是cp936编码的，我们要把这个字符串传给一个需要保存成其他编码的地方（比如xml的utf-8,excel需要的utf-16）
通常这么写：
strobj.decode("cp936").encode("utf-16")

3、外部编码： 一般不必要为字符串内在的表示担忧; 只有当尝试把Unicode传递给给一些基于字节的函数的时候，Unicode字符的表示 变成一个议题, 比如文件的write方法或网络套接字的send 方法。那时，你必须要选择该如何表示这些(Unicode) 字符为字节。从Unicode码到字节串的转换被叫做编码。同样地，当你从文件，套接字或其他的基于字节的对象 中装入一个Unicode字符串的时候,你需要把字节串解码为(Unicode)字符。

示例：

```
>>> f = open("/home/spider/atdc/data/alias/case1/pack", "r")
>>> data = f.read()
>>> data.__class__

>>> msg = u""
>>> msg += data
Traceback (most recent call last):
  File "", line 1, in
UnicodeDecodeError: 'ascii' codec can't decode byte 0xd5 in position 835: ordinal not in range(128)
```

msg被初始化为unicode字符串，而data是read函数返回的文件内容，是str对象，msg要粘结data就需要编

码转换了：

```
>>> msg += unicode(data, "ISO-8859-1")
```

另外跟标准输出打交道时也需要考虑编码转换：

```
>>> print msg
Traceback (most recent call last):
  File "", line 1, in
UnicodeEncodeError: 'ascii' codec can't encode characters in position 835-844: ordinal not in
range(128)
```

有人建议重设sys的defaultencoding，是否能奏效呢？

```
>>> import sys
>>> reload(sys)

>>> sys.setdefaultencoding('utf8')
>>> print msg
Traceback (most recent call last):
  File "", line 1, in
UnicodeEncodeError: 'ascii' codec can't encode characters in position 835-844: ordinal not in
range(128)
```

哦，跟之前一样的错误。那setdefaultencoding起了什么样的作用呢？继续看下面的示例：

```
>>> import sys
>>> sys.getdefaultencoding()
'ascii'
>>> u = u"我爱你"
>>> u
u'\xce\xd2\xb0\xae\xc4\xe3'
>>> u.decode("utf8")        #unicode不能用utf8解码？！
Traceback (most recent call last):
  File "", line 1, in
  File "/home/spider/bin/lib/python2.6/encodings/utf_8.py", line 16, in decode
    return codecs.utf_8_decode(input, errors, True)
UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-5: ordinal not in range(128)
>>> reload(sys)

>>> sys.setdefaultencoding('utf8')
>>> u.decode("utf8")
u'\xce\xd2\xb0\xae\xc4\xe3'
```

其实unicode已经是python支持的最底层的字符串格式了，再对其解码的话就使用sys的defaultencoding对结果进行编码。 而reload之前sys的defaultencoding是ascii，ascii只支持0-127的英文字符串的编码格式，所以无法编码中文。而reload 之后重设只是将解码后的unicode再用unicode编码，所以结果跟解码前的一样。

另外一些环境变量也会影响python的编码：

```
export LANG=zh_CN.gbk  #之前LAGN=C
```

```
>>> u = u"我爱你"
>>> u
u'\u6211\u7231\u4f60'
>>> print u      #这个时候就可以直接print了
我爱你
>>> s = "我爱你"
>>> s
'\xce\xd2\xb0\xae\xc4\xe3'
>>> print s
我爱你
>>> s.decode("gbk")  #跟变量u的内容一样，说明此时的unicode是用gbk编码的
u'\u6211\u7231\u4f60'
>>> import sys
>>> sys.getdefaultencoding()
'ascii'
>>> sys.stdout.encoding
'GBK'
>>> sys.getfilesystemencoding()
'GBK'
```

外部编码更进一步的例子可以参考：http://blog.csdn.net/jiyucn/archive/2008/02/16/2100006.aspx

4、其他话题： (1)插入非法字符：

一个字符串中不一定都是统一编码的，如读取一个网页，head部分是utf8编码，而body部分却可能是gb编码，或者由于 截断的缘故一些字符串已经不完整了，这个时候再做编码操作就可能出错。如：

```
>>> s = "我爱你"
>>> s
'\xce\xd2\xb0\xae\xc4\xe3'
>>> serr = '\xce\xd2\xb0\xae\xc4\xe3\xa1'
>>> s.decode("gbk")
u'\u6211\u7231\u4f60'
>>> serr.decode("gbk")
Traceback (most recent call last):
  File "", line 1, in
UnicodeDecodeError: 'gbk' codec can't decode byte 0xa1 in position 6: incomplete multibyte
sequence
```

gbk无法编码第6个字符0xa1。这个时候我们可以指定decode的第2个参数为"ignore"：

```
>>> serr.decode("gbk", "ignore")
u'\u6211\u7231\u4f60'
```

使用ignore可以忽视那些错误字符，可选项还有strict(缺省)，replace(会替换成一个合适的字符，往往是编码集外 的字符)，如：

```
export LANG=zh_CN.gbk
>>> serr = '\xce\xd2\xb0\xae\xc4\xe3\xa1'
>>> serr.decode("gbk", "replace")
u'\u6211\u7231\u4f60\ufffd'
>>> print serr.decode("gbk", "replace")
Traceback (most recent call last):
```

File "", line 1, in
UnicodeEncodeError: 'gbk' codec can't encode character u'\ufffd' in position 3: illegal multibyte sequence
对多数编码集来说字符u'\ufffd'都是非法字符。

(2)编码判断：

A、利用chardet识别编码：

```
#从标准输入读取字符串,重新编码,输出到标准输出

import sys
import chardet

data=sys.stdin.read()

incodec=chardet.detect(data)['encoding']

data=data.decode(incodec)
data=data.encode('utf-8')
sys.stdout.write(data)
```

唯一麻烦的是chardet不是python的标准库，需要自己手动安装。

B、利用系统信息，但会有例外：

```
import sys
reload(sys)
sys.setdefaultencoding('utf8')

class OutputWrapper:
  """进行控制台输出编码转换的封装方案"""

  input_enc = sys.getdefaultencoding()
  output_enc = sys.getfilesystemencoding()

  def __init__(self, target):
      self.target = target
      self.buffer = ''

  def flush(self):
      self.target.flush()

  def write(self, message):
      lines = message.split('\n')
      lines[0] = self.buffer + lines[0]
      self.buffer = lines.pop()
      for line in lines:
          try:
              line = line.decode(self.input_enc).encode(self.output_enc) + '\n'
          except:
```

```
            line = line + '\n'
        self.target.write(line)


sys.stdout = OutputWrapper(sys.__stdout__)
sys.stderr = OutputWrapper(sys.__stderr__)
```

局限性：

1、这个方案只是简单的假设至少在字符串的每一行中，采用的是相同的编码，或者是 sys.getdefaultencoding()， 或者是 sys.getfilesystemencoding()。 2、这个方案假设终端输出采用的编码和本地文件系统采用的编码是一样的(通常如此但总有例外)。

C、自己动手丰衣足食：

```
def zh2unicode(stri):
    """Auto converter encodings to unicode

    It will test utf8,gbk,big5,jp,kr to converter"""
    for c in ('utf-8', 'gbk', 'big5', 'jp','euc_kr','utf16','utf32'):
        try:
            return stri.decode(c)
        except:
            pass
    return stri
```

该函数将含中文字符的str对象转换为unicode，但是不知道哪种编码合适，就遍历地decode。局限性就在于遍历集不 一定全，而且遍历集过大的话会影响性能。