昵称：Z贺
园龄：1年7个月
粉丝：11
关注：6
+加关注

< 2019年1月 >

日	一	二	三	四	五	六
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

搜索

找找看

谷歌搜索

常用链接

我的随笔

Python（字符编码）

一 了解字符编码的知识储备

1. 文本编辑器存取文件的原理（nodepad++，pycharm，word）

打开编辑器就打开了启动了一个进程，是在内存中的，所以在编辑器编写的内容也都是存放与内存中的，断电后数据丢失

因而需要保存到硬盘上，点击保存按钮，就从内存中把数据刷到了硬盘上。

在这一点上，我们编写一个py文件（没有执行），跟编写其他文件没有任何区别，都只是在编写一堆字符而已。

2. python解释器执行py文件的原理 ，例如python test.py

第一阶段：python解释器启动，此时就相当于启动了一个文本编辑器

第二阶段：python解释器相当于文本编辑器，去打开test.py文件，从硬盘上将test.py的文件内容读入到内存中

第三阶段：python解释器解释执行刚刚加载到内存中test.py的代码

总结：

1. python解释器是解释执行文件内容的，因而python解释器具备读py文件的功能，这一点与文本编辑器一样

2. 与文本编辑器不一样的地方在于，python解释器不仅可以读文件内容，还可以执行文件内容

我的评论

我的参与

最新评论

我的标签

我的标签

Linux(3)

Linux命令(1)

vim(1)

处理器(1)

存储器(1)

计算机基础(1)

计算机启动流程(1)

命令(1)

权限(1)

网络基础(1)

更多

二 什么是字符编码

计算机要想工作必须通电,也就是说‘电’驱使计算机干活,而‘电’的特性，就是高低电平(高低平即二进制数1,低电平即二进制数0),也就是说计算机只认识数字

编程的目的是让计算机干活，而编程的结果说白了只是一堆字符，也就是说我们编程最终要实现的是：一堆字符驱动计算机干活

所以必须经过一个过程：

字符--------（翻译过程）------->数字

这个过程实际就是一个字符如何对应一个特定数字的标准，这个标准称之为字符编码

三 字符编码的发展史

阶段一：现代计算机起源于美国，最早诞生也是基于英文考虑的ASCII

ASCII:一个Bytes代表一个字符（英文字符/键盘上的所有其他字符），1Bytes=8bit，8bit可以表示0-2**8-1种变化，即可以表示256个字符

ASCII最初只用了后七位，127个数字，已经完全能够代表键盘上所有的字符了（英文字符/键盘的所有其他字符）

后来为了将拉丁文也编码进了ASCII表，将最高位也占用了

阶段二:为了满足中文，中国人定制了GBK

GBK:2Bytes代表一个字符

为了满足其他国家，各个国家纷纷定制了自己的编码

日本把日文编到Shift_JIS里，韩国把韩文编到Euc- kr里

阶段三：各国有各国的标准，就会不可避免地出现冲突，结果就是，在多语言混合的文本中，显示出来会有乱码。

于是产生了unicode， 统一用2Bytes代表一个字符， 2**16-1=65535，可代表6万多个字符，因而兼容万国语言

但对于通篇都是英文的文本来说，这种编码方式无疑是多了一倍的存储空间（二进制最终都是以电或者磁的方式存储到存储介质中的）

随笔分类

Django(1)

git

Mysql(2)

Python(28)

汇总(2)

爬虫

前端(5)

玄学

周边知识(3)

随笔档案

2018年2月 (1)

2018年1月 (3)

2017年8月 (5)

2017年7月 (14)

2017年6月 (23)

于是产生了UTF-8，对英文字符只用1Bytes表示，对中文字符用3Bytes

需要强调的一点是：

unicode：简单粗暴，所有字符都是2Bytes，优点是字符->数字的转换速度快，缺点是占用空间大

utf-8：精准，对不同的字符用不同的长度表示，优点是节省空间，缺点是：字符->数字的转换速度慢，因为每次都需要计算出字符需要多长的Bytes才能够准确表示

1. **内存中使用的编码是unicode，用空间换时间（程序都需要加载到内存才能运行，因而内存应该是尽可能的保证快）**
2. **硬盘中或者网络传输用utf-8，网络I/O延迟或磁盘I/O延迟要远大与utf-8的转换延迟，而且I/O应该是尽可能地节省带宽，保证数据传输的稳定性。**

四.字符编码分类

计算机由美国人发明，最早的字符编码为ASCII，只规定了英文字母数字和一些特殊字符与数字的对应关系。

ascii用1个字节（8位二进制）代表一个字符

unicode常用2个字节（16位二进制）代表一个字符，生僻字需要用4个字节

如果我们的文档通篇都是英文，你用unicode会比ascii耗费多一倍的空间，在存储和传输上十分的低效

本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的UTF-8编码。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。如果你要传输的文本包含大量英文字符，用UTF-8编码就能节省空间：

字符	ASCII	Unicode	UTF-8
A	01000001	00000000 01000001	01000001
中	x	01001110 00101101	11100100 10111000 10101101

从上面的表格还可以发现，UTF-8编码有一个额外的好处，就是ASCII编码实际上可以被看成是UTF-8编码的一部分，所以，大量只支持ASCII编码的历史遗留软件可以在UTF-8编码下继续工作。

2017年5月 (12)

最新评论

1. Re:Python（面向对象编程——2 继承、派生、组合、抽象类）

难得的好文章，解释得非常清楚易懂！

--多味夏天

2. Re:Python（面向对象编程——2 继承、派生、组合、抽象类）

写的不赖

--DiggingDeeply

阅读排行榜

1. Python（调用函数、定义函数）(35461)

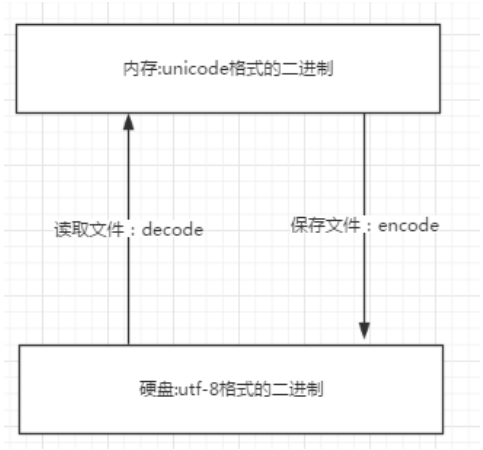
2. Python（输入、输出；简单运算符；流程控制;转译）(12649)

3. Python（字符编码）(11372)

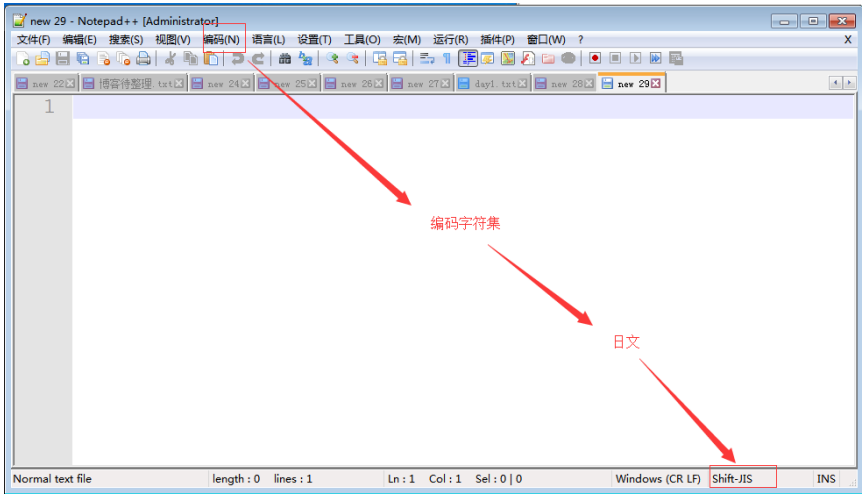
4. SHELL —— grep命令+正则表达式 (9873)

五 字符编码的使用

5.1 文本编辑器一锅端



5.1.2 文本编辑器nodpad++



总结：

5. Python（可变/不可变类型，list，tuple，dict，set）(3569)

无论是何种编辑器，要防止文件出现乱码（请一定注意，存放一段代码的文件也仅仅只是一个普通文件而已，此处指的是文件没有执行前，我们打开文件时出现的乱码）

核心法则就是，文件以什么编码保存的，就以什么编码方式打开

而文件编码保存时候使用的编码方式是右下角的编码方式，而解码的时候是使用文档开头申明的编码方式，两种编码不同的时候很容易出现乱码的情况。

5.2 程序的执行

python test.py　（我再强调一遍，执行test.py的第一步，一定是先将文件内容读入到内存中）

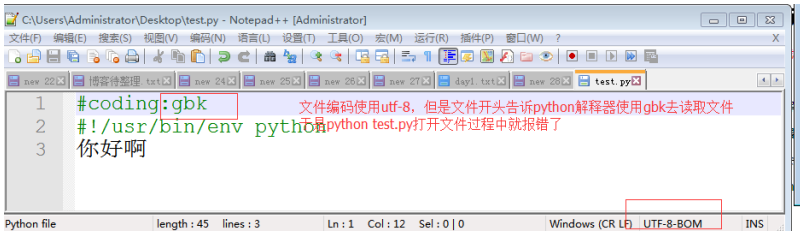
阶段一：启动python解释器

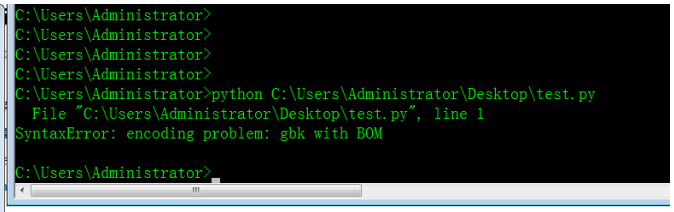
阶段二：python解释器此时就是一个文本编辑器，负责打开文件test.py,即从硬盘中读取test.py的内容到内存中

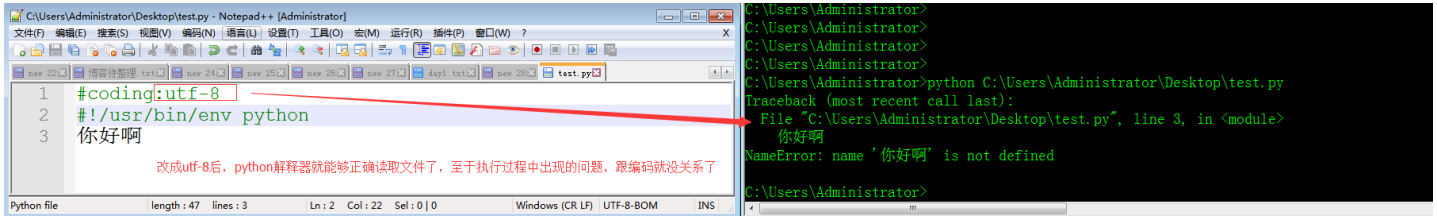
此时，python解释器会读取test.py的第一行内容，#coding:utf-8，来决定以什么编码格式来读入内存，这一行就是来设定python解释器这个软件的编码使用的编码格式这个编码，

可以用sys.getdefaultencoding()查看，如果不在python文件指定头信息 #-*-coding:utf-8-*-,那就使用默认的

python2中默认使用ascii，python3中默认使用utf-8







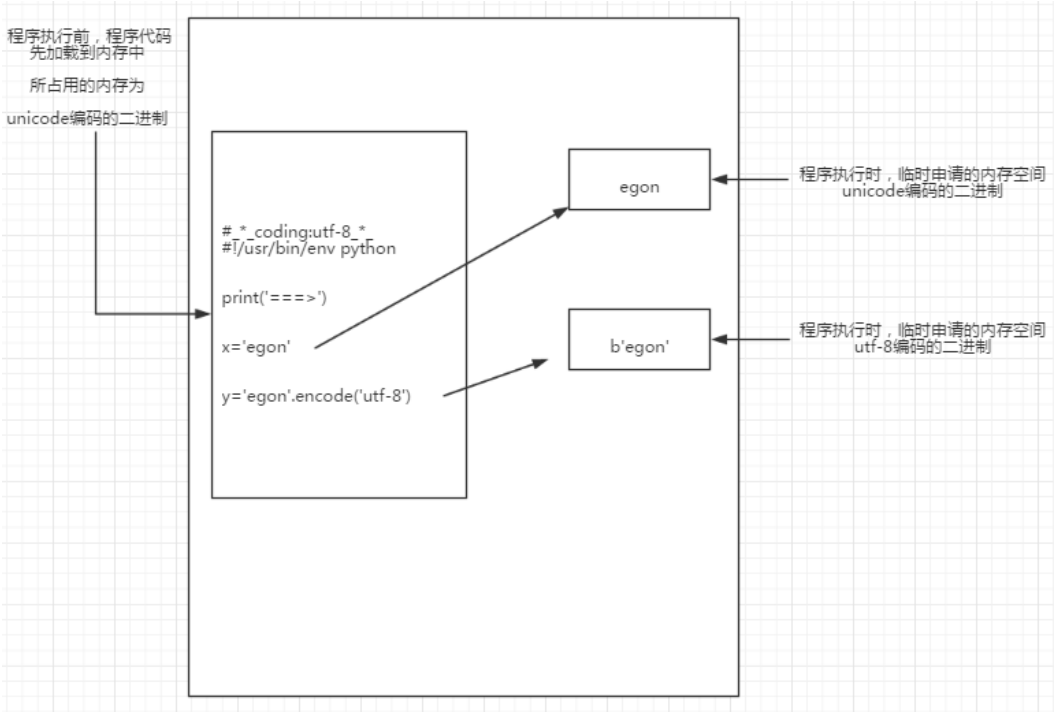
阶段三：读取已经加载到内存的代码（unicode编码的二进制），然后执行，执行过程中可能会开辟新的内存空间，比如x="egon"

内存的编码使用unicode，不代表内存中全都是unicode编码的二进制，

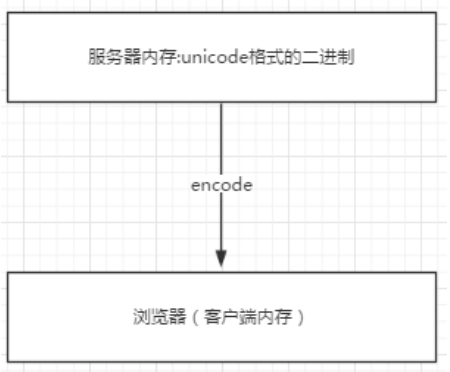
在程序执行之前，内存中确实都是unicode编码的二进制,比如从文件中读取了一行x="egon",其中的x，等号，引号，地位都一样，都是普通字符而已，都是以unicode编码的二进制形式存放与内存中的

但是程序在执行过程中，会申请内存（与程序代码所存在的内存是俩个空间），可以存放任意编码格式的数据，比如x="egon",会被python解释器识别为字符串，会申请内存空间来存放"hello"，然后让x指向该内存地址，此时新申请的该内存地址保存也是unicode编码的egon,如果代码换成x="egon".encode('utf-8'),那么新申请的内存空间里存放的就是utf-8编码的字符串egon了

针对python3如下图



浏览网页的时候，服务器会把动态生成的Unicode内容转换为UTF-8再传输到浏览器



如果服务端encode的编码格式是utf-8， 客户端内存中收到的也是utf-8编码的二进制。

5.3 python2与python3的区别

5.3.1 在python2中有两种字符串类型str和unicode

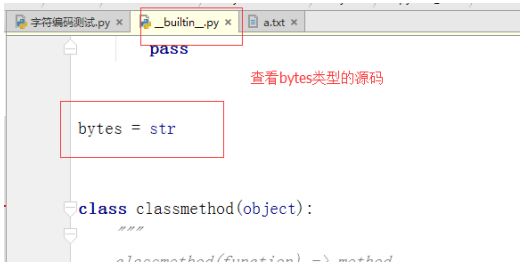
str类型

当python解释器执行到产生字符串的代码时（例如s='林'），会申请新的内存地址，然后将'林'encode成文件开头指定的编码格式，这已经是encode之后的结果了，所以s只能decode

```
1 #_*_coding:gbk_*_
2 #!/usr/bin/env python
3
4 x='林'
5 # print x.encode('gbk') #报错
6 print x.decode('gbk') #结果：林
```

所以很重要的一点是：

在python2中，str就是编码后的结果bytes，str=bytes,所以在python2中，unicode字符编码的结果是str/bytes




```
#coding:utf-8
s='林' #在执行时,'林'会被以conding:utf-8的形式保存到新的内存空间中

print repr(s) #'\xe6\x9e\x97' 三个Bytes,证明确实是utf-8
print type(s) #<type 'str'>
```

```
s.decode('utf-8')
# s.encode('utf-8') #报错，s为编码后的结果bytes，所以只能decode
```

unicode类型

当python解释器执行到产生字符串的代码时（例如s=u'林'），会申请新的内存地址，然后将'林'以unicode的格式存放到新的内存空间中，所以s只能encode，不能decode

```
s=u'林'
print repr(s) #u'\u6797'
print type(s) #<type 'unicode'>


# s.decode('utf-8') #报错，s为unicode，所以只能encode
s.encode('utf-8')
```

打印到终端

对于print需要特别说明的是：

当程序执行时，比如

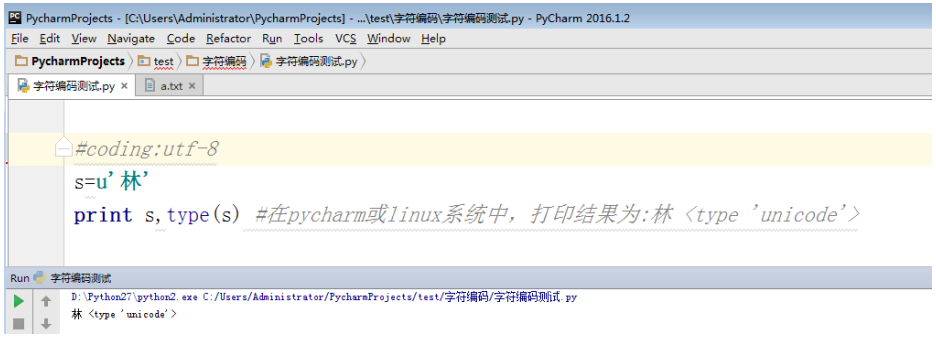
x='林'

print(x) #这一步是将x指向的那块新的内存空间（非代码所在的内存空间）中的内存，打印到终端，而终端仍然是运行于内存中的，所以这打印可以理解为从内存打印到内存，即内存->内存，unicode->unicode

对于unicode格式的数据来说，无论怎么打印，都不会乱码

python3中的字符串与python2中的u'字符串'，都是unicode，所以无论如何打印都不会乱码

在pycharm中

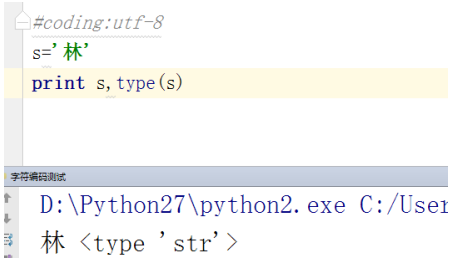


在windows终端

```
C:\Users\Administrator>python2 C:\Users\Administrator\PycharmProjects\test\字符编码\字符编码测试.py
林 <type 'unicode'>
```

但是在python2中存在另外一种非unicode的字符串，此时，print x，会按照终端的编码执行x.decode('终端编码')，变成unicode后，再打印，此时终端编码若与文件开头指定的编码不一致，乱码就产生了

在pycharm中（终端编码为utf-8，文件编码为utf-8，不会乱码）



在windows终端（终端编码为gbk，文件编码为utf-8，乱码产生）

```
C:\Users\Administrator>python2 C:\Users\Administrator\PycharmProjects\test\字符编码\字符编码测试.py
鏄?<type 'str'>
```

思考题：

分别验证在pycharm中和cmd中下述的打印结果

```
#coding:utf-8
s=u'林' #当程序执行时，'林'会被以unicode形式保存新的内存空间中


#s指向的是unicode，因而可以编码成任意格式，都不会报encode错误
s1=s.encode('utf-8')
s2=s.encode('gbk')
print s1 #打印正常否？
print s2 #打印正常否


print repr(s) #u'\u6797'
print repr(s1) #'\xe6\x9e\x97' 编码一个汉字utf-8用3Bytes
print repr(s2) #'\xc1\xd6' 编码一个汉字gbk用2Bytes

print type(s) #<type 'unicode'>
print type(s1) #<type 'str'>
print type(s2) #<type 'str'>
```

5.3.2 在python3中也有两种字符串类型str和bytes

str是unicode

```
#coding:utf-8
s='林' #当程序执行时，无需加u，'林'也会被以unicode形式保存新的内存空间中,

#s可以直接encode成任意编码格式
s.encode('utf-8')
s.encode('gbk')

print(type(s)) #<class 'str'>
```

bytes是bytes

```
#coding:utf-8
s='林' #当程序执行时，无需加u，'林'也会被以unicode形式保存新的内存空间中,

#s可以直接encode成任意编码格式
s1=s.encode('utf-8')
s2=s.encode('gbk')



print(s) #林
print(s1) #b'\xe6\x9e\x97' 在python3中，是什么就打印什么
print(s2) #b'\xc1\xd6' 同上

print(type(s)) #<class 'str'>
print(type(s1)) #<class 'bytes'>
print(type(s2)) #<class 'bytes'>
```

分类: Python

« 上一篇：Python（可变/不可变类型，list，tuple，dict，set）
» 下一篇：Python（函数的参数）

相关博文：

· python 字符编码
· Python 字符编码
· Python字符编码
· python -字符编码
· python 字符编码问题