python中的字符串

2021年1月15日

11:46

Python的字符串类型是str，在内存中以Unicode表示，一个字符对应若干个字节。如果要在网络上传输，或者保存到磁盘上，就需要把str变为以字节为单位的bytes。

字符编码

2021年1月15日

11:39

计算机是美国人发明的，因此，最早只有127个字符被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为ASCII编码，比如大写字母A的编码是65，小写字母z的编码是122。

但是要处理中文显然一个字节是不够的，至少需要两个字节，而且还不能和ASCII编码冲突，所以，中国制定了GB2312编码，用来把中文编进去。

你可以想得到的是，全世界有上百种语言，日本把日文编到Shift\_JIS里，韩国把韩文编到Euc-kr里，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

因此，Unicode字符集应运而生。Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。

Unicode标准也在不断发展，但最常用的是UCS-16编码，用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要4个字节）。现代操作系统和大多数编程语言都直接支持Unicode。

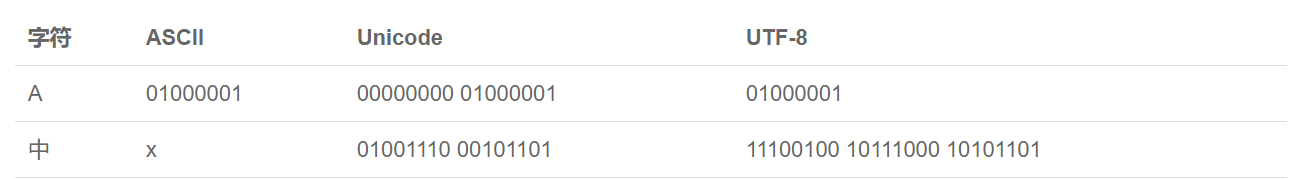
现在，捋一捋ASCII编码和Unicode编码的区别：ASCII编码是1个字节，而Unicode编码通常是2个字节。

UTF-8编码

你可以猜测，如果把ASCII编码的A用Unicode编码，只需要在前面补0就可以，因此，A的Unicode编码是00000000 01000001。

新的问题又出现了：如果统一成Unicode编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用Unicode编码比ASCII编码需要多一倍的存储空间，在存储和传输上就十分不划算。

所以，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：

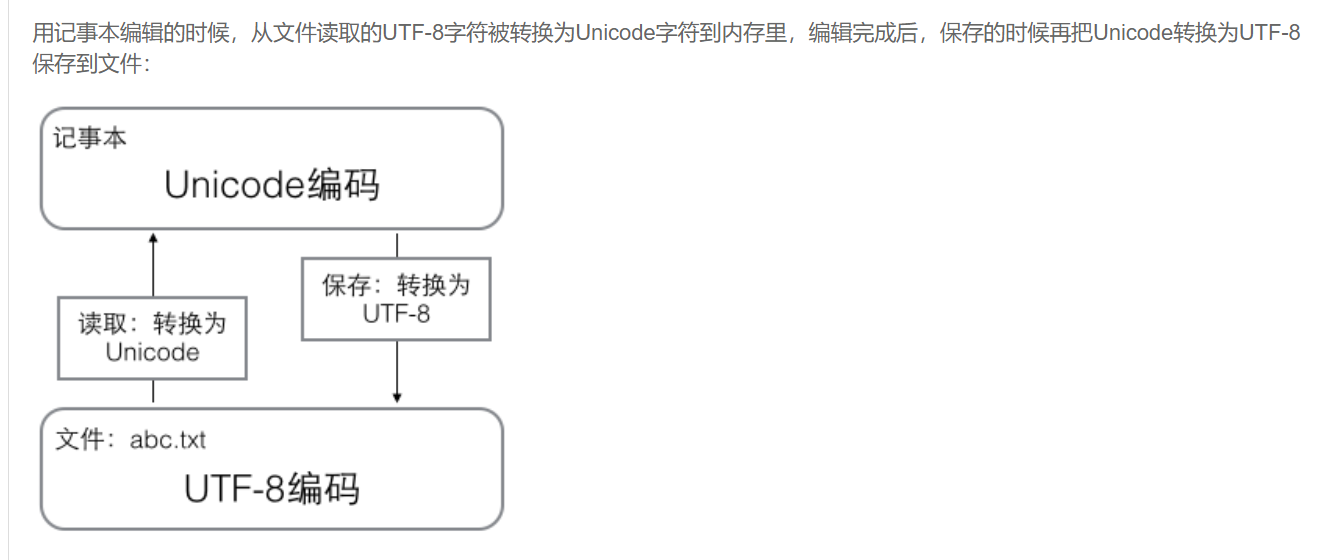


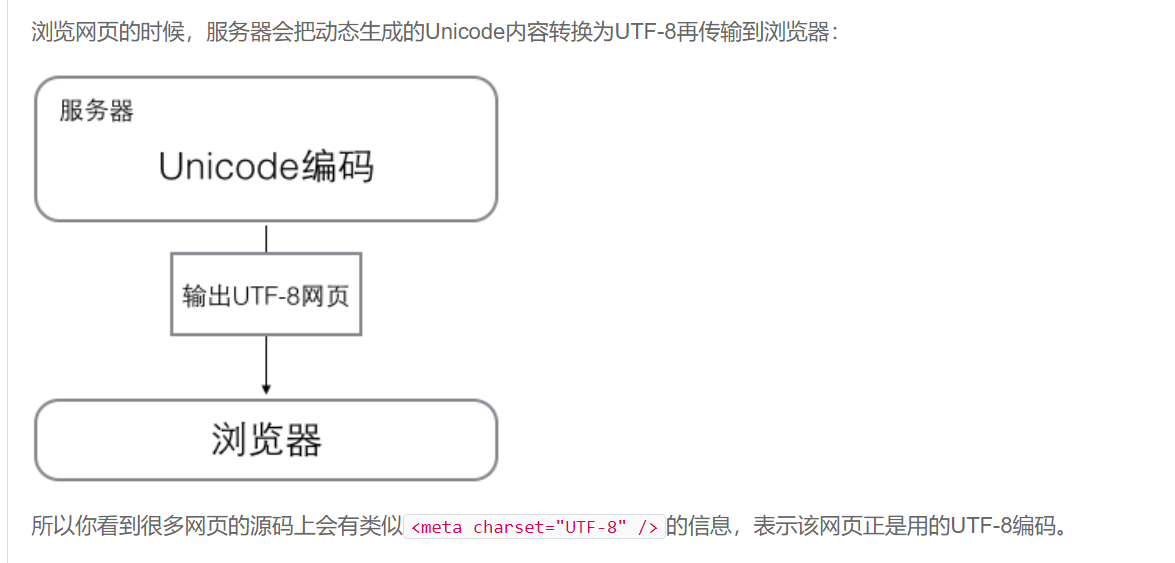
从上面的表格还可以发现，UTF-8编码有一个额外的好处，就是ASCII编码实际上可以被看成是UTF-8编码的一部分，所以，大量只支持ASCII编码的历史遗留软件可以在UTF-8编码下继续工作。

现在计算机系统通用的字符编码工作方式：

在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。

用记事本编辑的时候，从文件读取的UTF-8字符被转换为Unicode字符到内存里，编辑完成后，保存的时候再把Unicode转换为UTF-8保存到文件：





字符串，list列表类型

2020年10月5日 星期一

下午5:33

name="Hello World!"

|  |  |
| --- | --- |
| print(name.title()) | 首字母大写 |
| print(name.upper()) | 全部大写 |
| print(name.lower()) | 全部小写 |

编辑字符串

a=['name','phone\_num','QQ\_num']

print(a)#-1引用最后一个

a.append("Ronchy")#append()添加到末尾

dela[1]

print(a)

popped\_a=a.pop()#pop()弹出最后个元素，即栈顶元素，将其与原栈分离，但仍可使用

print("a=",a)#可指定弹出某一项->a.pop(1)

print("popped\_a=",popped\_a)

#print("a=",a)

#a.remove("name")#remove("")删除指定的元素

#print("a=",a)

a.insert(0,"name!")#插入元素,剩余的元素右移

print("a=",a)

元组tuple

2021年1月15日

12:23

tuple和list类似，但是tuple一旦初始化就不能修改，

定义：

classmates = ('Michael', 'Bob', 'Tracy')

圆括号

dict 字典

2021年1月15日

14:13

Python内置了字典：dict的支持，dict全称dictionary，在其他语言中也称为map，使用键-值（key-value）存储，具有极快的查找速度。

定义一个dict：

Example：

>>> d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}  
>>> d['Michael']  
95

数据放入dict的方法，除了初始化时指定外，还可以通过key放入：

>>> d['Adam'] = 67  
>>> d['Adam']  
67

若对一个key放入多个value，则会覆盖：

>>> d['Jack'] = 90  
>>> d['Jack']  
90  
>>> d['Jack'] = 88  
>>> d['Jack']  
88

key

value



读文件 写文件

注意：使用with等语句时注意缩进，空格缩进！

2021年1月15日

19:55

Python引入了with语句来自动帮我们调用close()方法：

**with** open('/path/to/file', 'r') **as** f:  
 print(f.read())

这和前面的try ... finally是一样的，但是代码更佳简洁，并且不必调用f.close()方法。

读取文本文件：

**> > >with** open('/path/to/file', 'r') **as** f1:  
 print(f.read())

读取二进制文件：

>>>with open('/path/to/file', 'rb') as f2  
 print(f.read())

其他的读操作：

* 要读取非UTF-8编码的文本文件，需要给open()函数传入encoding参数，例如，读取GBK编码的文件：
  + >>> f = open('/Users/michael/gbk.txt', 'r', encoding='gbk')  
    >>> f.read()
* 遇到有些编码不规范的文件，你可能会遇到UnicodeDecodeError，因为在文本文件中可能夹杂了一些非法编码的字符。遇到这种情况，open()函数还接收一个errors参数，表示如果遇到编码错误后如何处理。最简单的方式是直接忽略：
  + >>> f = open('/Users/michael/gbk.txt', 'r', encoding='gbk', errors='ignore')

写文件

写文件和读文件是一样的，唯一区别是调用open()函数时，传入标识符'w'或者'wb'表示写文本文件或写二进制文件：

调用操作：

**>>>with** open('/Users/michael/test.txt', 'w') **as** f:  
 f.write('Hello, world!')