**一、编码原理及区别**

·第一阶段：编码的由来：大家都知道计算机的母语是英语，英语是由26个字母组成的。所以最初的计算机字符编码是通过ASCII来编码的，也是现今最通用的单字节编码系统，使用7位二进制数来表示所有的字母、数字、标点符号及一些特殊控制字符，作为美国编码标准来使用。

·第二阶段：由于计算机的不断普及，计算机进入了不同的国家和地区。很快表现出了众多的水土不服的症状。比如：中国人就表示不服，就不能让计算机说中文吗？但是ASCII编码用上浑身解数256个字符也不够中文用啊。   
因此后来出现了Unicode编码。Unicode编码通常由两个字节组成，共表示256\*256个字符，即所谓的UCS-2。某些生僻的字还会用到四个字节，即UCS-4。Unicode向下兼容ASCII。

·第三个阶段：在Unicode中，很快，美国人不高兴了，原本用一个字节就够了，现在却要用两个字节来表示，非常浪费存储空间和传输速度。人们再发挥聪明才智，于是出现了UTF-8编码。主要针对空间浪费的问题。UTF-8从英文字母的一个字节，到中文的通常的三个字节，再到某些生僻字的六个字节。解决了空间浪费的问题，并且兼容老大哥ASCII编码。这样一些老古董软件在UTF-8编码中可以继续工作了。   
另外需要注意的是汉字在Unicode编码和UTF-8编码中通常是不同的。

·第四个阶段：同样我们祖国也制定了自己的一套标准。那就是GB2312和GBK。但是大多数还是直接用的UTF-8。UTF-8/UTF-16等是对Unicode进行了编码，是其一种实现方式。   
另外还有ISO-8859-1，ISO-8859-1是单字节编码，向下兼容ASCII，是许多欧洲国家使用的编码标准。   
Unicode是国际通用的编码标准，可以表示全世界的字符，但其字符集也是最复杂、占用空间最大的。开发者可以根据需要进行选择编码方式。

·我们的Python3默认就是Unicode编码

1. **编码与解码**

·Python关于编码和解码对应的函数分别是encode()和decode()，对应编码和解码

·Python中有str和bytes之分

·str to unicode，使用decode()；unicode to str，使用encode()

用图片来说明

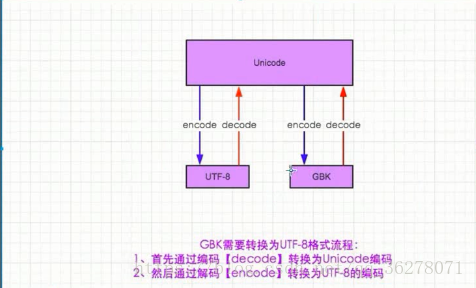


图1 编码流程

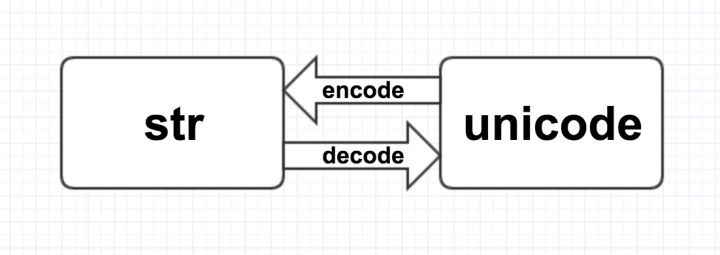


图2 Python2 字符序列类型

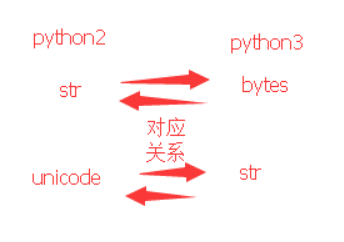
1. **Python中的Print大法**

·它不管你塞过来的是什么格式什么编码，字符串数组对象什么的的都一口气全打印出来。

·Python3有两种表示字符序列的类型：bytes和str。前者的实例包含原始的8位值；后者的实例包含Unicode字符。

·Python2中也有两种表示字符序列的类型，分别叫做str和unicode。与Python3不同的是，str的实例包含原始的8位值，而unicode的实例，则包含Unicode字符。

·[Python2中的unicode和str以及Python3中的str和bytes](https://www.cnblogs.com/yangmingxianshen/p/7990102.html)，对应关系如下：



·在Python2中，作为两种类型的字符序列，str与unicode需要转换，它们是这样转换的：

str——decode方法——》unicode——encode方法——》str

在Python3中可以这样对应这转换，配合上面的图，也许会好理解一点。

bytes——decode（解码）方法——》str——>encode（编码）方法——》bytes

1. **本地文件和网络资源编码**

·本地文件编码测试：

Python2中读取的是str类型，如果要把str转化为unicode，就要解码，也就是decoding.

·网络资源编码：

Python2环境下requests库使用



1. **常见错误**

其一：UnicodeEncodeError: xxx codec can't encode character xxx in position xxx

·在Python3中字符就是unicode字符，字符串就是unicode字符数组

·str转bytes叫encode，bytes转str叫decode

·print()函数自身有限制，不能完全打印所有的unicode字符

解决方法其一：

print()函数的局限就是Python默认编码的局限，因为系统是windows的，python的默认编码不是'utf-8',改一下python的默认编码成'utf-8'就行了

1. **import** io
2. **import** sys
3. **import** urllib.request
4. sys.stdout = io.TextIOWrapper(sys.stdout.buffer,encoding='utf8') #改变标准输出的默认编码
5. res=urllib.request.urlopen('http://www.baidu.com')
6. htmlBytes=res.read()
7. **print**(htmlBytes.decode('utf-8'))

运行后不报错了，但是居然有好多乱码（英文显示正常，中文则显示乱码）！！又一阵折腾后发现是控制台的问题，具体来说就是我在cmd下运行该脚本会有乱码，而在IDLE下运行却很正常。

由此我推测是cmd不能很好地兼容utf8，而IDLE就可以，甚至在IDLE下运行，连“改变标准输出的默认编码”都不用，因为它默认就是utf8。如果一定要在cmd下运行，那就改一下编码，比如我换成“gb18030”，就能正常显示了：

1. sys.stdout = io.TextIOWrapper(sys.stdout.buffer,encoding='gb18030')         #改变标准输出的默认编码

编码对照表：



解决方法其二：

· #encoding='XXX' 这里(也就是python文件第一行的内容)的编码是指该python脚本文件本身的编码，无关紧要。只要XXX和文件本身的编码相同就行了。 比如notepad++ "格式"菜单里面里可以设置各种编码，这时需要保证该菜单里设置的编码和encoding XXX相同就行了，不同的话会报错

·网络数据流的编码 比如获取网页，那么网络数据流的编码就是网页的编码。需要使用decode解码成unicode编码。

·目标文件的编码 要将网络数据流的编码写入到新文件，那么我么需要指定新文件的编码。写文件代码如：

1. f.write(string)

那么string是一个字符串，它是通过decode解码过的字符串。关键点就要来了：目标文件的编码是导致标题所指问题的罪魁祸首。如果我们打开一个文件：

1. f = open("out.html","w")

在windows下面，新文件的默认编码是gbk，这样的话，python解释器会用gbk编码去解析我们的网络数据流txt，然而txt此时已经是decode过的unicode编码，这样的话就会导致解析不了，出现上述问题。 解决的办法就是，改变目标文件的编码：

1. f = open("out.html","w",encoding='utf-8')

看个例子：

1. f = urllib.request.urlopen('http://www.scut.edu.cn/new')
2. html = f.read().decode('utf-8')
3. with open('html.txt','w',encoding='utf-8') as file:
4. file.write(html)
5. [httPs://www.cnblogs.com/yangmingxianshen/P/7990102.html](https://www.cnblogs.com/yangmingxianshen/p/7990102.html)
6. [httPs://www.zhihu.com/question/27062410](https://www.zhihu.com/question/27062410)
7. [httPs://blog.csdn.net/qq\_36278071/article/details/79660196](https://blog.csdn.net/qq_36278071/article/details/79660196)
8. [httPs://blog.csdn.net/jim7424994/article/details/22675759](https://blog.csdn.net/jim7424994/article/details/22675759)
9. [httP://www.jb51.net/article/64816.htm](http://www.jb51.net/article/64816.htm)