第十章　常用标准库

本章导读

库丰富是Python语言的重要特征，已经有一系列功能强大的标准库(Python Standard Library)。Python语言的核心除语言本身外，只包含了数字、字符串、列表、Tuple、字典、集合、文件等常见数据类型和函数，而由Python标准库提供了系统管理、网络通信、文本处理、数据库接口、图形系统、XML处理等额外的功能。Python标准库命名接口清晰、文档良好，很容易学习和使用。

学习目标：

1. 掌握常见标准库的用法；

本章目录

第一节　pickle库：数据持久

例程10-1

```
第1行  import pickle
第2行
第3行  class UD():
第4行      def __init__(self,u,d):
第5行          self.Up=u
第6行          self.Down=d
第7行      def __str__(self):
第8行          return str(self.Up)+"/"+str(self.Down)
第9行  listUD=[UD(2,3),UD(1,2)]
第10行 outFile=open(r"D:\UD.pkl","wb")
第11行 pickle.dump(listUD,outFile)
第12行 outFile.close()
第13行
第14行 inFile=open(r"D:\UD.pkl","rb")
第15行 udList=pickle.load(inFile)
第16行 for i in udList:
第17行     print(i,end="\t")
第18行
第19行 #eof
```

第二节　collections库

1、namedtuple子库：命名元组

例程10-2

```
from collections import namedtuple
 #定义一个数据类型friend，其名称为FRIEND
friend=namedtuple("朋友信息",['Name','Age','Email'])

f1=friend('张三',33,'zhangsan@999.net')#顺序与定义保持一致
print(f1)#输出：朋友信息(Name='张三', Age=33, Email='zhangsan@999.net')
print(f1.Name,f1.Age,f1.Email)#输出：张三 33 zhangsan@999.net

#顺序与定义顺序并不一致，对应名称赋值，更加直观
f2=friend(Name='李四',Email='lisi@yyy.com',Age=30)
print(f2)#输出：朋友信息(Name='李四', Age=30, Email='lisi@yyy.com')

name,age,email=f2
print(name,age,email)#输出：李四 30 lisi@yyy.com

poetList=[("李白","望庐山瀑布","唐朝"),("王维","使至塞上","唐朝"),
("苏东坡","念奴娇·赤壁怀古","宋朝"),("元好问","摸鱼儿·雁丘词","金代")]
Poet = namedtuple('诗词名家', ['Name', 'Work', 'Dynasty'])

for p in poetList:
    print(Poet._make(p))
#输出如下：
#诗词名家(Name='李白', Work='望庐山瀑布', Dynasty='唐朝')
#诗词名家(Name='王维', Work='使至塞上', Dynasty='唐朝')
#诗词名家(Name='苏东坡', Work='念奴娇·赤壁怀古', Dynasty='宋朝')
#诗词名家(Name='元好问', Work='摸鱼儿·雁丘词', Dynasty='金代')

for p in poetList:
    data=Poet._make(p)
    print(data.Name,data.Dynasty,data.Work)
#输出形如：李白 唐朝 望庐山瀑布

#eof
```

2、counter子库：计数

例程10-3

```
from collections import Counter

listA=[12,123,12,23,123,345]
print(Counter(listA).most_common(2))

words="""中国是以华夏文明为源泉、中华文化为基础并以汉族为主体民族
```

```
的多民族国家，通用汉语。中国人常以龙的传人、炎黄子孙自居。中国是
世界四大文明古国之一，有着悠久的历史，距今约5000年前，以中原地区
为中心开始出现聚落组织进而形成国家，后历经多次民族交融和朝代更迭，
直至形成多民族国家的大一统局面。20世纪初辛亥革命后，君主政体退出
历史舞台，共和政体建立。1949年中华人民共和国成立后，在中国大陆建
立了人民代表大会制度的政体。"""

for i in Counter(words).most_common(10):
    print(i[0],"出现次数=",i[1])

wordCount=Counter()
for word in words:
    if word not in ["、","，","，","。","\n","0","1","2","3","4","5","6","7","8","9"]:
        wordCount[word]+=1

print(wordCount.most_common(10))
#上行输出：[('国', 9), ('中', 8), ('民', 6), ('族', 5), ('的', 5), ('为', 4), ('体', 4), ('人', 4), ('以', 4), ('大', 4)]

cnt=Counter("中国国家家中中心心中中华华夏夏天天国国家家国")
print(cnt)#输出：[('中', 5), ('国', 5), ('家', 4), ('心', 2), ('华', 2), ('天', 2), ('夏', 2)]
print(cnt["中"],cnt["美"])#输出：5 0("美"没有出现，故为0)
print(list(cnt.elements()))
#输出：['国', '国', '国', '国', '国', '中', '中', '中', '中', '中', '夏', '夏', '天', '天', '心', '心', '华', '华', '家', '家', '家', '家']

#eof
```

3、deque子库：双端队列

使用list存储数据时，按索引访问元素很快，但是插入和删除元素就很慢了，因为list是线性连续存储，数据量大的时候，插入和删除效率很低。deque是为了高效实现插入和删除操作的双向列表，既可以如list一样在尾部高效插入和删除，也可以在头部高效插入和删除，非常适合队列和栈的实现。例程10-4是deque的应用示例。

例程10-4

```
from collections import deque

listA=["李白","杜甫","王维","孟浩然"]
dequeA=deque(listA)#从list中创建deque
dequeA.append("贺知章")
#append()用于在尾部追加元素，pop()用于删除尾部元素
print(dequeA)#输出：deque(['李白', '杜甫', '王维', '孟浩然', '贺知章'])
for i in dequeA:
    print(i,end="$")#输出：李白$杜甫$王维$孟浩然$贺知章$

dequeA.appendleft("骆宾王")#popleft()用于删除头部元素
for i in dequeA:print(i,end="$")
#上行输出：骆宾王$李白$杜甫$王维$孟浩然$贺知章$

```

```
#下行实现尾部追加多个元素，多元素必须为可迭代序列
dequeA.extend(["柳宗元","孟郊","韩愈","白居易","卢纶","李贺","李益","刘禹锡"])
for i in dequeA:print(i,end="$")
#上行输出：骆宾王$李白$杜 甫$王维$……$韩愈$白居易$卢纶$李贺$李益$刘禹锡$
dequeA.extendleft({"王勃","卢照邻"})
for i in dequeA:print(i,end="$")
#上行输出：王勃$卢照邻$骆宾王$李白……李贺$李益$刘禹锡$

dequeA.insert(5,"王之涣")
dequeA.remove("李益")#删除元素"李益"，如果没有改元素，则返回ValueError
dequeA.reverse()#反序
print()
for i in dequeA:print(i,end="$")

dequeA.rotate(4)#将右边4个元素旋转到左侧
for i in dequeA:print(i,end="$")

dequeA.rotate(-4)#将左边4个元素旋转到右侧
for i in dequeA:print(i,end="$")

#maxlen是deque唯一的只读属性，用于限制最大长度
#当在尾部追加时，删除头部，反之亦成立。
dequeB=deque([1,2,3,4,5],maxlen=8)
dequeB.extend(['A','B','C','D',"E"])
for i in dequeB:print(i,end="$")#输出：3$4$5$A$B$C$D$E$

dequeB.appendleft("Z")
for i in dequeB:print(i,end="$")#输出：3$4$5$A$B$C$D$E$

#eof
```

4、OrderedDict子库：有序字典

例程10-5

```
from collections import OrderedDict
items=(('A', 1),('B', 2),('C', 3))
regularDict = dict(items)
print(regularDict)

orderDict=OrderedDict(items)
print(orderDict)#注：输出顺序不稳定

for x in orderDict:
    print(x,orderDict[x])
#输出顺序稳定为：A、B、C及其对应值

```

```
    for k,v in orderDict.items():
        print(k,v)
#输出效果与上面相同，另外一种实现方式

#eof
```

5、defaultdict子库：默认值字典

使用dict时，如果引用的Key不存在，就会抛出KeyError。如果希望key不存在时，返回一个默认值，就可以用defaultdict。例程10-6的应用示例。

例程10-6

```
from collections import defaultdict
dd = defaultdict(lambda: '查无此人')
dd["李白"]="青莲居士"
dd["杜甫"]="少陵野老"
print(dd["李白"])#输出：青莲居士
print(dd["王维"])#输出：查无此人

#eof
```

第三节　stuct库

在用Python变成时，有时需要用处理二进制数据，比如图片文件、网络socket等，这时可用Python的struct库来完成，形如C语言的struct。struct模块最重要的三个函数分别是pack()、 unpack()、calcsize()。pack(fmt, v1, v2, ...)，按照给定的格式(fmt)，将数据封装成字符串类似C/C++的字节流；unpack(fmt, string)按照给定的格式(fmt)解析字节流string，返回结果为tuple；calcsize(fmt)计算给定的格式(fmt)占用多少字节的内存。例程10-7是struct的应用示例。

例程10-7

```
import struct

a=12.34
bytes=struct.pack('f',a)
print(len(bytes),bytes,sep="___")#输出：4___b'\xa4pEA'
#bytes为string字符串，相当于将a的内容简单按字节方式存储

#逆向反操作

#现有二进制数据bytes，（其实就是字符串），将它反过来转换成python的数据类型：
a,=struct.unpack('f',bytes)#注意,a后的逗号不能省略，稍后会有解释
print(a)#输出：12.34000015258789
#注意，unpack返回的是tuple，如果仅有一个变量，需要写成a,或者(a,)

#eof
```

例程10-8

```
import struct

a=(12,'abc'.encode("utf-8"),24.45)
structCompiled = struct.Struct('I3sf')
```

```
第5行  packed_data = structCompiled.pack(*a)
第6行  c= (structCompiled.unpack(packed_data))
第7行  print(c)#输出：(12, b'abc', 24.450000762939453)
第8行  print(c[1])#输出：b'abc'
第9行
第10行 #eof
```

表10-1：struct字节序及其对其方式

Character	Byte order	Size	Alignment
@	native	native	native
=	native	standard	none
<	little-endian	standard	none
>	big-endian	standard	none
!	network (= big-endian)	standard	none

表10-2：struct格式字符

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	bytes of length 1	1	
b	signed char	integer	1	(1),(3)
B	unsigned char	integer	1	(3)
?	_Bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
I	unsigned int	integer	4	(3)
l	long	integer	4	(3)
L	unsigned long	integer	4	(3)
q	long long	integer	8	(2), (3)
Q	unsigned long long	integer	8	(2), (3)
n	ssize_t	integer		(4)
N	size_t	integer		(4)
f	float	float	4	(5)
d	double	float	8	(5)
s	char[]	bytes		
p	char[]	bytes		
P	void*	integer		(6)

第四节　array库

array类型只能存入相同数据类型的数据，这是其他序列类型最大的不同，使用array将显著提高效率。

例程10-9

```
第1行  import array
```

```
第2行 arrA=array.array('l', [1, 2, 3, 4, 5])#只能是统一类型的数据
第3行 arrB=array.array('d', [1.0, 2.0, 3.14])
第4行 for i in arrB:print(i,end="\t")
第5行
第6行 arrA.append(123)#在尾部追加数据
第7行 print(arrA.buffer_info())#输出tuple型数据，是地址与元素数量的tuple，(2008877762448, 6)
第8行
第9行 print(arrA.count(4))#查找4在arrA中出现次数
第10行
第11行 #eof
```

表10-3：array库的数据类型

Type code	C Type	Python Type	Minimum size in bytes	Notes
'b'	signed char	int	1	
'B'	unsigned char	int	1	
'u'	Py_UNICODE	Unicode character	2	(1)
'h'	signed short	int	2	
'H'	unsigned short	int	2	
'i'	signed int	int	2	
'I'	unsigned int	int	2	
'l'	signed long	int	4	
'L'	unsigned long	int	4	
'q'	signed long long	int	8	(2)
'Q'	unsigned long long	int	8	(2)
'f'	float	float	4	
'd'	double	float	8	

第五节　小结