第四章 文件对象-文件的读写

目录

读写操作

新建文件

- **open()函数**　　语法：file object=open(“文件路径”“文件名”，mode)
  - 对文件操作之前需要用 open() 函数打开文件，打开一个已经存在的文件或者创建一个新文件。
  - 打开文件的模式(mode)：一般常用模式：r(只读)、w(只写)、a(追加)、b(二进制)
  - 打开之后将返回一个文件对象（file object），后续对文件内数据的操作都是基于这个文件对象的方法（method）来实现的

- 举例：
  - 直接打开一个文件，若文件不存在则创建新文件

```
import os#文件和路径进行操作需要的模块
f=open("D:\\"'helloworld.txt',mode='r')
```

  - 如不给出路径则在当前工作路径下

文件路径

- **绝对路径：** 从根文件夹开始，Window 系统中以盘符（C：、D：）作为根文件夹
  'd:\\a.txt'，转义的方式
  r'd:\a.txt'，声明字符串不需要转义
- **相对路径：** 相对于当前工作目录所在的位置。例如，当前工作目录为 "C:\Windows\System32"，若文件 demo.txt 就位于这个 System32 文件夹下，则 demo.txt 的相对路径表示为 ".\demo.txt"（其中 .\ 就表示当前所在目录）

利用 os.getcwd() 函数可以取得当前工作路径的字符串。

```
os.getcwd()
```

```
'C:\\Users\\lenovo'
```

路径写法

Windows 中，路径书写使用反斜杠 "\" 作为文件夹之间的分隔符，可用 os.path.join() 函数来做这件事。如将单个文件和路径上的文件夹名称的字符串传递给它，os.path.join() 就会返回一个文件路径的字符串，包含正确的路径分隔符。

```
import os#路径操作需要导入的模块
os.path.join('demo', 'exercise')
'demo\\exercise'
```

创建带有文件名称的文件存储路径，os.path.join() 函数同样很有用。如：将一个文件名列表中的名称，添加到文件夹名称的末尾

```
import os
myFiles = ['accounts.txt', 'details.csv', 'invite.docx']
for filename in myFiles:
    print(os.path.join('C:\\demo\\exercise', filename))
```

```
C:\demo\exercise\accounts.txt
C:\demo\exercise\details.csv
C:\demo\exercise\invite.docx
```

Python处理绝对路径和相对路径

Python os.path 模块提供了一些函数，实现绝对路径和相对路径之间的转换，检查给定的路径是否为绝对路径:

•os.path.abspath(path) 将返回 path 参数的绝对路径的字符串，这是将相对路径转换为绝对路径的简便方法。

•os.path.isabs(path)，如果参数是一个绝对路径，就返回 True，如果参数是一个相对路径，就返回 False。

•os.path.relpath(path, start) 将返回从 start 路径到 path 的相对路径的字符串。如果没有提供 start，就使用当前工作目录作为开始路径。

•os.path.dirname(path) 将返回一个字符串，它包含 path 参数中最后一个斜杠之前的所有内容;

•os.path.basename(path) 将返回一个字符串，它包含 path 参数中最后一个斜杠之后的所有内容。

```
os.getcwd()#获取当前工作路径，相对路径
```

```
'C:\\Users\\lenovo'
```

```
os.path.abspath('.')#获取当前工作路径的绝对路径
```

```
'C:\\Users\\lenovo'
```

```
os.path.abspath('.\\fancy')
```

```
'C:\\Users\\lenovo\\fancy'
```

```
os.path.isabs('.')
```

```
False
```

```
os.path.isabs(os.path.abspath('.'))
```

```
True
```

```
path = 'C:\\Windows\\System32\\calc.exe'
os.path.dirname(path)
```

```
'C:\\Windows\\System32'
```

```
os.path.relpath('C:\\Windows', 'C:\\')
```

```
'Windows'
```

```
os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')
```

```
'..\\..\\Windows'
```

```
path = 'C:\\Windows\\System32\\calc.exe'
```

```
os.path.basename(path)
```

```
'calc.exe'
```

```
os.path.dirname(path)
```

```
'C:\\Windows\\System32'
```

Python路径操作

如果同时需要一个按照路径将文件名和路径分割开，调用 os.path.split() 获得这两个字符串的元组

```
path = 'C:\\Windows\\System32\\calc.exe'
os.path.split(path)
```

```
('C:\\Windows\\System32', 'calc.exe')
```

路径上文件(夹)存在性检查

如提供的路径不存在，许多 Python 函数就会崩溃并报错，os.path 模块提供了以下函数检测给定路径是否存在，以及它是文件还是文件夹：

- 如path 参数所指的文件或文件夹存在，调用 os.path.exists(path) 将返回 True，否则返回 False。
- 如 path 参数存在，并且是一个文件，调用 os.path.isfile(path) 将返回 True，否则返回 False。
- 如path 参数存在，并且是一个文件夹，调用 os.path.isdir(path) 将返回 True，否则返回 False。

在交互式环境中尝试这些函数的结果

```
os.path.exists('C:\\Windows')
True
os.path.exists('C:\\some_made_up_folder')
False
 os.path.isdir('C:\\Windows\\System32')
True
os.path.isfile('C:\\Windows\\System32')
False
 os.path.isdir('C:\\Windows\\System32\\calc.exe')
False
 os.path.isfile('C:\\Windows\\System32\\calc.exe')
True
```

文件操作

```
f=open("D:\\"'helloworld.txt',mode='r',)
content = f.read()
print(content)
f.close()
print(f.closed)
# 输出访问模式
print(f.mode)
#输出编码格式
print(f.encoding)
# 输出文件名
print(f.name)
print(f)
```

```
Hello, world!
True
 r
 cp936
D:\helloworld.txt
 <_io.TextIOWrapper name='D:\\helloworld.txt' mode='r' encoding='cp936'>
```

添加了编码方式UTF-8后

```
f=open("D:\\"'helloworld.txt',mode='r',encoding='UTF-8')
content = f.read()
print(content)
f.close()
print(f.closed)# 输出文件是否已经关闭
print(f.mode)# 输出访问模式
print(f.encoding)#输出编码格式
print(f.name)# 输出文件名
print(f)#输出了文件对象的相关信息，包括打开文件的名称、打开模式、打开文件时所使用的编码格式。
```

```
Hello, world!
True
r
UTF-8
D:\helloworld.txt
<_io.TextIOWrapper name='D:\\helloworld.txt' mode='r' encoding='UTF-8'>
```

思考：helloworld.txt中的内容是中文时以上操作的影响。

输出了文件对象的相关信息，包括打开文件的名称、打开模式、打开文件时所使用的编码格式

CP936：就是GBK，IBM在发明Code Page的时候将GBK放在第936页，所以叫CP936， GBK全称《汉字内码扩展规范》

读文件

- **read()函数**
  - 读取文件数据， 返回文件中的所有内容
  - read()读取文件的全部内容，如果文件大于可用内存，为了保险起见，可以反复调用read(size)方法，每次最多读取size个字节的内容
- **readline()函数**
  - 每次只读取一行，通常比readlines() 慢得多
  - 仅当没有足够内存可以一次读取整个文件时才使用
- **readlines()函数**
  - 一次性读入文件所有数据，读取后得到的是每行数据组成的列表
  - 一行样本数据全部存储为一个字符串，并且数据读入后并没有将换行符去掉

```
['i am fine\n', 'how are you?\n', "what's your name?"]
```

读文件

- 使用read()文件读取文件
- 使用 **print** 命令打印文件内容，显示 Hello, world!
  - 每次操作完文件后，都要关闭文件 **f.close()**，因为只有在关闭文件的同时才会将内存中的内容更新到文件里边。
  - 否则，文件会一直被Python占用，不能被其他进程使用

```
f = open("D:\\" 'helloworld.txt',mode='r')
content = f.read()
print(content)
f.close()
```

```
hello , world!
```

- 也可以使用 **with open() as f:** 在操作后自动关闭文件

```
with open("D:\\" 'helloworld.txt') as f:
    content = f.read()
    print(content)
```

```
hello , world!
```

读文件

- 在 **read()** 中加入数字，可指定读取的字符数

```
f = open("D:\\"'helloworld.txt','r')
content = f.read(5)
print(content)
f.close()
```

- 输出结果

```
Hello
```

```
f = open("D:\\"'helloworld.txt','r')
content = f.read(8)
print(content)
f.close()
```

- 输出结果

```
Hello, w
```

open() 完整方法

完整的语法格式为：

open(“path” ‘file_name’, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

参数说明(注意默认项):

path： 文件路径（相对或者绝对路径）

file: 文件名

mode: 可选， 文件打开模式

buffering: 设置缓冲

encoding: 一般使用utf8

errors: 报错级别

newline: 区分换行符

closefd: 传入的file参数类型

opener: 设置自定义开启器， 开启器的返回值必须是一个打开的文件描述符。

简单读写操作

- 创建(读写)文件

序号	模式	说明
1	r	以读的方式打开文件。文件指针在文件的开始。这是文件的默认打开模式。
2	w	以写的方式打开文件。如果文件已经存在，则覆盖原文件，即返回文件对象前 清空该文件；否则新建文件。
3	a	以写的方式打开文件。如果文件已经存在，则指针在文件的最后，可以实现向文件中追加新内容；否则，新建文件，并能实现读写操作。
4	b	以二进制模式打开文件，但不单独使用，配合r/w/a等模式使用。rb(只读)为输入打开一个二进制文件；wb（只写）为输出建立一个新的二进制文件；ab（追加）向二进制文件尾增加数据。
5	+	同时实现读写操作，但是不单独使用，配合r/w/a等模式使用。w+表示可以对文件进行读写双重操作。r+(读/写)为读/写打开一个文本文件；w+(读/写)为读/写建立一个新的文本文件；a+(读/写)为读/写建立一个新的文本文件；rb+(读/写)为读/写打开一个二进制文件；wb+(读、写)为读/写建立一个新的二进制文件；ab+(读/写)为读/写打开一个二进制文件。
6	x	创建文件，但是如果文件已经存在，则无法创建。

读写特定类型文件

- CSV文档
- **JSON文档**
- Word文档
- Excel文档

读写csv文件

- 利用Python内置的 csv 模块读取数据

•csv（comma Seperated Values）逗号分隔值文件。
•文件的格式非常简单，类似一个文本文档，每一行保存一条数据，同一行中的各个数据通常采用逗号（或tab）分隔。
•python自带了csv模块，专门用于处理csv文件的读取和存档。

```
import csv #导入CSV模块
file_name= 'Pokemon.csv'#文件名
f = open(file_name ,'r',encoding='UTF-8') #打开CSV文件，path?
reader = csv.reader(f) #读取CSV文件数据
content=[]
for con in reader:
    content.append(con)
f.close #关闭文件
print(content[0])
print(content[1])
```

```
['#', 'Name', 'Type 1', 'Type 2', 'Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed', 'Generation', 'Legendary']
['1', 'Bulbasaur', 'Grass', 'Poison', '318', '45', '49', '49', '65', '65', '45', '1', 'False']
```

读出了表头和第一行具体数据

读写csv文件

- 利用Python内置的 csv 模块写入数据

```
import csv
path = 'Pokemon_write.csv'
#写入Pokemon前两行数据
content = [['#', 'Name', 'Type 1', 'Type 2', 'Total', 'HP', 'Attack',
'Defense', 'Sp. Atk', 'Sp. Def', 'Speed', 'Generation',
'Legendary'],['1', 'Bulbasaur', 'Grass', 'Poison', '318', '45', '49',
'49', '65', '65', '45', '1', 'False']]
f = open (path,‘w’) #打开文件
writer = csv.writer(f)
for con in content:
    writer.writerow(con) #写入数据
f.close()
```

读写Json文件

- 如果要在不同的编程语言之间传递对象，就必须把对象序列化为标准格式，比如XML，(Extensible Markup Language,可扩展标记语言)但更好的方法是序列化为JSON，因为JSON表示出来就是一个字符串，可以被所有语言读取，也可以方便地存储到磁盘或者通过网络传输。JSON不仅是标准格式，并且比XML更快，而且可以直接在Web页面中读取，非常方便。
- JSON(JavaScript Object Notation, JS 对象简谱) 是一种轻量级的数据交换格式。

  最初是为JavaScript开发的，但随后成了一种常用的数据格式文件，被包括Python的众多语言所采用

  JavaScript（简称"JS"） 是一种具有函数优先的轻量级，解释型或即时编译型的编程语言。虽然它是作为开发Web页面的脚本语言而出名，但是它也被用到了很多非浏览器环境中。

Json与python

JSON中的数据格式和Python中的数据格式转化关系如下

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

json的主要函数

- 由于JSON标准规定JSON编码是UTF-8，所以我们总是能正确地在Python的str与JSON的字符串之间转换。
- dumps():返回一个字符串，内容是标准json形式；
- dump():将对象序列化成json文件
- loads():将json形式字符串反序列化为python对象
- load():将json文件反序列化

json的主要函数

- .json.dumps() 函数是将一个 Python 数据类型列表（可以理解为字典）进行json格式的编码（转换成字符串，用于传播）

```
dict = {"age": "12"}
json_str = json.dumps(dict)
```

- json.loads() 函数是将 json 格式数据（字符串）转换为字典（方便取出里面的数据），和 json.dumps() 正好相反

```
json_str = '{"age": "12"}‘
dict = json.loads(json_str)
age = dict['age']
```

- json.dump() 函数用于将 json 信息（字符串）写进文件

```
json_str = "{'age': '12'}”
file = open('1.json', 'w', encoding='utf-8‘)
json.dump(json_str, file)
```

- json.load() 函数用于读取 json 信息（文件）转成字符串，和 json.dump() 相反

```
file = open('1.json', 'r', encoding='utf-8‘)
info = json.load(file)
```

https://blog.csdn.net/whjkm/article/details/81159888

json的主要函数

json.dumps()用于将dict类型的数据转成str，因为如果直接将dict类型的数据写入json文件中会发生报错，因此在将数据写入时需要用到该函数。

```
import json
name_emb = {'a':'1111','b':'2222','c':'3333','d':'4444'}

jsObj = json.dumps(name_emb)

print(name_emb)
print(jsObj)

print(type(name_emb))
print(type(jsObj))
```

运行结果如下：

```
{'a': '1111', 'c': '3333', 'b': '2222', 'd': '4444'}
{"a": "1111", "c": "3333", "b": "2222", "d": "4444"}
<type 'dict'>
<type 'str'>
```

若在数据写入json文件时，未先进行转换，报错如下

```
In [1]: import json

        name_emb = {'a':'1111','b':'2222','c':'3333','d':'4444'}

        emb_filename = ('/home/cqh/faceData/emb_json.json')

        # jsObj = json.dumps(name_emb)

        with open(emb_filename, "w") as f:
            f.write(name_emb)
            f.close()
```



```
TypeError                                 Traceback (most recent call last)
<ipython-input-1-9a96b25e7fc8> in <module>()
      8
      9 with open(emb_filename, "w") as f:
---> 10     f.write(name_emb)
     11     f.close()

TypeError: expected a string or other character buffer object
```

转换后再写入，则不报错

```
import json

name_emb = {'a':'1111','b':'2222','c':'3333','d':'4444'}

emb_filename = ('/home/cqh/faceData/emb_json.json')

jsObj = json.dumps(name_emb)

with open(emb_filename, "w") as f:
    f.write(jsObj)
    f.close()
```

json的主要函数

- json.loads()用于将str类型的数据转成dict

```
import json
name_emb = {'a':'1111','b':'2222','c':'3333','d':'4444'}
 jsDumps = json.dumps(name_emb)
 jsLoads = json.loads(jsDumps)
 print(name_emb)
print(jsDumps)
print(jsLoads)
 print(type(name_emb))
print(type(jsDumps))
print(type(jsLoads))
```

运行结果如下:

```
{'a': '1111', 'c': '3333', 'b': '2222', 'd': '4444'}
{"a": "1111", "c": "3333", "b": "2222", "d": "4444"}
{u'a': u'1111', u'c': u'3333', u'b': u'2222', u'd': u'4444'}
<type 'dict'>
<type 'str'>
<type 'dict'>
```

a'变成了u'a'是因为发生了类型转换，str会转换成unicode

json的主要函数

- json.dump()用于将dict类型的数据转成str，并写入到json文件中。下面两种方法都可以将数据写入json文件

```
import json
name_emb = {'a':'1111','b':'2222','c':'3333','d':'4444'}
emb_filename = ('/home/cqh/faceData/emb_json.json')
# solution 1
jsObj = json.dumps(name_emb)
with open(emb_filename, "w") as f:
f.write(jsObj)
f.close()
# solution 2
json.dump(name_emb, open(emb_filename, "w"))
```

```python
import json

name_emb = {'a':'1111','b':'2222','c':'3333','d':'4444'}

emb_filename = ('/home/cqh/faceData/emb_json.json')

# solution 1
jsObj = json.dumps(name_emb)
with open(emb_filename, "w") as f:
    f.write(jsObj)
    f.close()

# solution 2
json.dump(name_emb, open(emb_filename, "w"))
```

json的主要函数

- json.load()用于从json文件中读取数据。

```
import json
emb_filename = ('/home/cqh/faceData/emb_json.json')
jsObj = json.load(open(emb_filename))
print(jsObj)
print(type(jsObj))
for key in jsObj.keys():
        print('key: %s  value: %s' % (key,jsObj.get(key)))
```

运行结果如下：

```
{u'a': u'1111', u'c': u'3333', u'b': u'2222', u'd': u'4444'}
    <type 'dict'>
    key: a value: 1111
    key: c value: 3333
    key: b value: 2222
    key: d value: 4444
```

总结：

json.dumps : dict转成str　　json.dump是将python数据保存成json

json.loads:str转成dict　　json.load是读取json数据

json.dump() 函数参数

json.dump(obj, fp, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)

•obj: 表示是要序列化的对象。

•fp: 文件描述符

•ensure_ascii: 默认值为True,能将所有传入的非ASCII字符转义输出。如果ensure_ascii为False,则这些字符将按原样输出。

```
import json
a=[1,2,'自动化']
with open('f.json','w',encoding='utf-8') as f:
    print(json.dump(a,f,ensure_ascii=False))
    #注意ensure_ascii这个参数的默认值为Ture，它会将对象按照ascii码序列化，也就是说，当打开json文件时中文会以乱码的形式呈现出来
with open('f.json','r',encoding='utf-8') as f:
    temp=json.load(f)
    print(temp)
```

```
>>>
[1,2,'自动化']
```

json.dumps() json.load() json.loads() 参数

- json.dumps(obj, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)

  dumps函数不需要传文件描述符，其他的参数和dump函数的一样

- json.load(fp, *, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw)

- json.loads(s, *, encoding=None, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw)

  s: 将s（包含JSON文档的str，bytes或bytearray实例）反序列化为Python对象。
encoding: 指定一个编码的格式。

读写Word文件

- 简介
  - Python可以利用python-docx模块处理Word文档（安装 pip3 install python-docx）
  - python-docx模块会把Word文档，文档中的**段落、文本、字体**等都看做对象，对对象进行处理就是对Word文档的内容处理

- **python-docx模块相关概念**
  - Document对象，表示一个Word文档
  - Paragraph对象，表示Word文档中的一个段落
  - Paragraph对象的**text**属性，表示段落中的文本内容

python-docx模块安装

•由于python-docx 已经提交给PyPI仓库，所以可以使用pip安装，如下：pip install python-docx

•如果同时安装了python2和python3那么pip可能不能用，可以使用pip3来安装，如下： pip3 install python-docx

•python-docx 也可以使用easy_install来安装，如下：easy_install python-docx

•如果不能使用 pip 和 easy_install ，可以在PyPI下载包、解压、运行 setup.py ，如下：tar xvzf python-docx-{version}.tar.gz cd python-docx-{version} python setup.py installpython-docx 还依赖 lxml 包 ，使用前2种方法会自动安装所需依赖包，第三种方法需要自己手动安装。

PyPI 仓库： https://pypi.org/

Find, install and publish Python packages with the Python Package Index

The Python Package Index (PyPI) is a repository of software for the Python programming language.

读写Word文件

- 读取Word文件

```
import docx#导入docx包
from docx.shared import Inches

file = docx.Document(“Python编程实践.docx")#打开文件
print("段落数:"+str(len(file.paragraphs)))#打印段落数
for para in file.paragraphs:#查看段落内容
    print(para.text)
```

```
段落数:1
Python编程实践
```

- 写入Word文件

```
#写入数据,并可通过style设置格式
file.add_paragraph( ‘Python机器学习',style = '9')
file.save(“Python编程实践.docx")#保存文件
```

读写Excel文件

- 简介
  - Python使用**openpyxl**库读写**Excel**文件
  - 这是一个第三方库，可以处理xlsx格式的Excel文件，不能处理xls格式文件

- **Excel文件三个对象**
  - **workbook**： 工作簿，一个**Excel**文件包含多个**sheet**
  - **sheet**：工作表，一个**workbook**有多个，如“sheet1”,“sheet2”等
  - **cell**： 单元格，存储数据对象

读写Excel文件

- 读取Excel

```
from openpyxl import load_workbook“#导入所需包
file_excel = load_workbook("数据.xlsx")#读文件openpyxl.reader.excel
sheetnames = file_excel.get_sheet_names()#获取读文件中所有的sheet
ws = file_excel.get_sheet_by_name(sheetnames[0])#获取第一个sheet内容
rows_max = ws.max_row#获取sheet的最大行数
cols_max = ws.max_column#获取sheet的最大列数
print('行数',str(rows_max))
print('列数',str(cols_max))
for r in range(1,rows_max+1):
    for c in range(1,cols_max+1):
        print(ws.cell(r,c).value)
    if r==5: #显示前五行
        break
saveExcel = "数据.xlsx"
file_excel.save(saveExcel)  # 保存
```

读写Excel文件

- 写入Excel
  - 使用append()插入
  - 也可直接插入，例如sheet1['A3'] = '小明'

```
from openpyxl import Workbook

test1 = Workbook()  # 打开一个将写的文件
sheet1 = test1.create_sheet(index=0)  # 在将写的文件创建
sheet
row1 = ['姓名','数学','语文']
row2 = ['tom',78,89]
sheet1.append(row1)#使用append插入数据
sheet1.append(row2)
sheet1['A3'] = '小明'#直接插入数据
sheet1['B3'] = 80
sheet1['C3'] = 84

saveExcel = "test1.xlsx"
test1.save(saveExcel)  # 保存
```

	A	B	C
1	姓名	数学	语文
2	tom	78	89
3	小明	80	84
4			

保存数据

使用pickle、shelve模块

存入SQLite数据库

保存数据的简单方式

对象持久化保存方法

- 使用pickle模块
- 使用shelve模块

将数据写入pickle

- **pickle模块简介（序列化库）**
  - Python内置模块。python程序运行中得到了一些字符串，列表，字典等数据，想要长久的保存下来，方便以后使用，而不是简单的放入内存中关机断电就丢失数据。pickle提供简单的持久化功能，可以将对象以文件的形式存放在磁盘上。
  - pickle模块只能在python中使用
  - pickle序列化后的数据，可读性差，人一般无法识别。
- **pickle可以存储的数据类型**
  - 所有Python支持的原生类型：布尔值，整数，浮点数，复数，字符串，字节，None
  - 由任何原生类型组成的列表、元组、字典和集合
  - 函数、类、类的实例

将数据写入pickle

- 将字典对象存储到文件（序列化）

```
#将字典对象存储到文件中
import pickle
f1 = open('pickle.txt','wb')
d = {‘class‘:’数据结构','evaluate':'good'}
pickle.dump(d,f1)  #将数据通过特殊的形式转换为只有python语言认识的字符串，并写入文件
f1.close()
```

- 从数据文件中读取数据（反序列化）

```
#从数据文件中读取数据
import pickle
read_file = open('pickle.txt','rb')
data = pickle.load(read_file)#从数据文件中读取数据，并转换
为python的数据结构
print(data)
read_file.close()
```

```
{'class': ‘数据结构', 'evaluate': 'good'}
```

将数据写入shelve

- shelve模块简介
  - shelve是Python的自带模块，可以直接通过import shelve来引用
  - shelve类似于一个存储持久化对象的持久化字典，即字典文件
  - 使用方法类似于字典
- Shelve模块特点
  - shelve模块只有一个open()函数；
  - shelve模块是一个简单的将内存数据通过文件持久化保存的模块；
  - shelve模块可以持久化任何pickle可支持的python数据格式。
- Shelve中的open（）函数
  - 格式为：shelve.open(filename)#打开文件

将数据写入shelve

- 保存对象至shelve文件中

```
import shelve
firm1 = dict(zip(['class','function'],[‘数据分析','数据科学']))
firm2 = dict(zip([‘class’,‘function’],[‘教学','视频']))
db = shelve.open('shelveDict')#打开一个文件
db['firm1']=firm1#向文件中添加内容，添加方式和字典的添加键值对相同
db['firm2']=firm2
db.close#关闭文件
```

- 从文件中读取对象

```
db = shelve.open('shelveDict')#打开文件
print(db['firm1'])#像从字典中获取键的方式一样读取内容
print(db['firm2'])
db.close()
```

```
{'class': ‘数据分析', 'function': '数据科学'}
{'class': ‘教学', 'function': '视频'}
```

将数据写入shelve

- 更新文件中的数据

```
db = shelve.open('shelveDict')#打开文件
firm2 = db['firm2']#从文件中读取之前存储的对象
firm2['class'] = 'education '#直接对对象进行修改,
db['firm2'] = firm2#重新存储至字典文件对象中
print(db['firm2'])
db.close()
```

```
{'class': 'education', 'function': '视频'}
```

SQLite数据库

- SQLite简介
  - SQLite是内嵌在Python中的轻量级、开源的、嵌入式的、关系型数据库。不需要安装和配置服务
  - 2000年由D. Richard Hipp发布，可以支持Java、Net、PHP、Ruby、Python、Perl、C等几乎所有的现代编程语言，支持Windows、Linux、Unix、Mac OS、Android、IOS等几乎所有的主流操作系统平台。
  - 支持使用SQL(Structured Query Language，结构化查询语言)语句来访问数据库
  - 一个数据库就是一个文件，通过直接复制数据库文件就可以实现数据库的备份
  - 访问和操作SQLite数据时，首先导入sqlite3模块，然后创建一个与数据库关联的Connection对象

建立连接对象

- 将数据写入SQLite数据库

```
#导入模块
import sqlite3
#首先要连接数据库，如果不存在就自动创建一个，如果存在的话，就打开那个数据库
语法格式：Sqlite3.connect(database)
conn = sqlite3.connect('example1.db')  #返回一个链接对象
#创建一个Cusor对象，并调用Cursor对象的execute方法来执行SQL语句
c = conn.cursor()   #获取游标，通过游标可以执行SQL语句
```

cursor提供的方法来进行工作.
这些方法包括两大类:1.执行命令,2.接收返回值

使用SQL语言

- 创建表格并插入数据

```
#创建表
c.execute('''CREATE TABLE stocks(date real,event text,place text)''') #向表中插入数据
c.execute('''insert into stocks values('2008','奥运会','北京')''')
conn.commit()#提交当前事务，保存数据
conn.close()#关闭数据库连接
```

- 查询插入的数据

```
#由于刚才已经关闭了数据库连接，需要重新创建
Connection对象和Cursor对象
conn = sqlite3.connect('example1.db')
c = conn.execute('''select * from stocks''')
print(c)
print(list(c))#数据成功提取出来了
```

```
<sqlite3.Cursor object at
0x00000000088F5810>
[(2008.0, '奥运会', '北京')]
```

总结

- 读写操作
- 读写特定类型文件
- 保存数据