第7章 文件与文件夹操作

7.1 文件的概念与分类

为了长期保存数据以便重复使用、修改和共享，必须将数据以文件的形式存储到外部存储介质(如磁盘、U盘、光盘或云盘、网盘、快盘等)中。

按文件中数据的组织形式把文件分为文本文件和二进制文件两类。

7.2 文件操作的基本知识

文件内容操作三部曲：打开、读写、关闭

文件对象名=open(file, mode='r', buffering=-1, encoding=None, errors=None,newline=None, closefd=True, opener=None)

文件名指定了被打开的文件名称。

打开模式指定了打开文件后的处理方式。

缓冲区指定了读写文件的缓存模式。0表示不缓存，1表示缓存，如大于1则表示缓冲区的大小。默认值是缓存模式。

参数encoding指定对文本进行编码和解码的方式，只适用于文本模式，可以使用Python支持的任何格式，如GBK、utf8、CP936等等。

基本操作示例

f1 = open( 'file1.txt', 'r' )

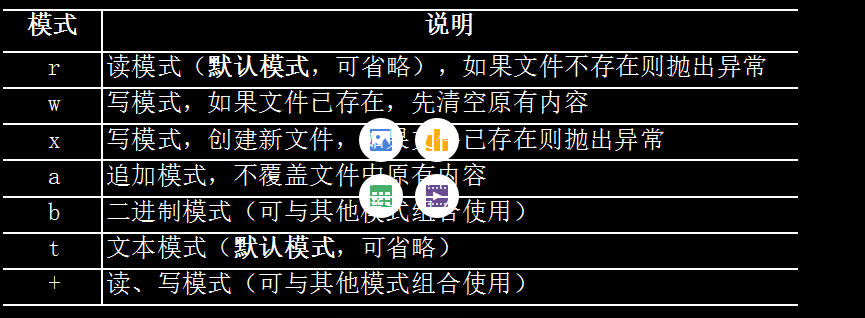
f2 = open( 'file2.txt', 'w')

……

f1.close()

f2.close()

文件打开方式



文件对象常用属性

|  |  |
| --- | --- |
| **属性** | **说明** |
| buffer | 返回当前文件的缓冲区对象 |
| closed | 判断文件是否关闭，若文件已关闭则返回True |
| fileno | 文件号，一般不需要太关心这个数字 |
| mode | 返回文件的打开模式 |
| name | 返回文件的名称 |

文件对象常用方法

|  |  |
| --- | --- |
| **方法** | 功能说明 |
| **close()** | **把缓冲区的内容写入文件，同时关闭文件，并释放文件对象** |
| **detach()** | 分离并返回底层的缓冲，底层缓冲被分离后，文件对象不再可用，不允许做任何操作 |
| **flush()** | **把缓冲区的内容写入文件，但不关闭文件** |
| **read([size])** | **从文本文件中读取size个字符（Python 3.x）的内容作为结果返回，或从二进制文件中读取指定数量的字节并返回，如果省略size则表示读取所有内容** |
| **readable()** | 测试当前文件是否可读 |
| **readline()** | 从文本文件中读取一行内容作为结果返回 |
| **readlines()** | 把文本文件中的每行文本作为一个字符串存入列表中，返回该列表，对于大文件会占用较多内存，不建议使用 |
| **seek(offset[, whence])** | **把文件指针移动到新的位置，offset表示相对于whence的位置。whence为0表示从文件头开始计算，1表示从当前位置开始计算，2表示从文件尾开始计算，默认为0** |
| **seekable()** | 测试当前文件是否支持随机访问，如果文件不支持随机访问，则调用方法seek()、tell()和truncate()时会抛出异常 |
| **tell()** | 返回文件指针的当前位置 |
| **truncate([size])** | 删除从当前指针位置到文件末尾的内容。如果指定了size，则不论指针在什么位置都只留下前size个字节，其余的一律删除 |
| **write(s)** | **把s的内容写入文件** |
| **writable()** | 测试当前文件是否可写 |

用with open 读写文件内容

用于文件内容读写时，with语句的用法如下：

with open(filename, mode, encoding) as fp:

#这里通过文件对象fp读写文件内容的语句

另外，上下文管理语句with还支持下面的用法，进一步简化了代码的编写。

with open('test.txt', 'r') as src, open('test\_new.txt', 'w') as dst:

dst.write(src.read())

示例：

例7-1 读取test.txt中的内容，删除多余空白行，写入到test1.txt中。

with open('test.txt','r') as fin,open('test1.txt','w')as fout:

for each\_line in fin:

if each\_line!='\n': #each\_line.split()

fout.write(each\_line)

else:

fout.write("")

二进制文件操作

数据库文件、图像文件、可执行文件、音视频文件、Office文档等等均属于二进制文件。

对于二进制文件，不能使用记事本或其他文本编辑软件进行正常读写，也无法通过Python的文件对象直接读取和理解二进制文件的内容。必须正确理解二进制文件结构和序列化规则，才能准确地理解二进制文件内容并且设计正确的反序列化规则。

所谓序列化，简单地说就是把内存中的数据在不丢失其类型信息的情况下转成对象的二进制形式的过程，对象序列化后的形式经过正确的反序列化过程应该能够准确无误地恢复为原来的对象。

Python中常用的序列化模块有struct、pickle、marshal和shelve。

使用pickle模块

示例：

将一个列表数据转换成二进制存取

使用pickle模块

例7-2将一个列表数据转换成二进制存取

>>> import pickle

>>> my\_list=[1,2,3.14,'绵阳',[1,2,3]]

>>> with open('test2.pkl','wb') as pickle\_file:

pickle.dump(my\_list,pickle\_file)

>>> with open('test2.pkl','rb') as pickle\_file:

my\_list2=pickle.load(pickle\_file)

>>> print(my\_list2)

7.3文件夹操作

如果需要处理文件路径，可以使用os.path模块中的对象和方法；

如果需要使用命令行读取文件内容可以使用fileinput模块；

创建临时文件和文件夹可以使用tempfile模块；

另外，Python 3.4之后版本的pathlib模块提供了大量用于表示和处理文件系统路径的类。

**7.3.1 目录操作**

os模块常用的目录操作函数

|  |  |
| --- | --- |
| 函数名称 | 使用说明 |
| mkdir(path[, mode=0o777]) | 创建目录，要求上级目录必须存在 |
| makedirs(path1/path2…, mode=511) | 创建多级目录，会根据需要自动创建中间缺失的目录 |
| rmdir(path) | 删除目录，要求该文件夹中不能有文件或子文件夹 |
| removedirs(path1/path2…) | 删除多级目录 |
| listdir(path) | 返回指定目录下所有文件信息 |
| getcwd() | 返回当前工作目录 |
| chdir(path) | 把path设为当前工作目录 |
| walk(top, topdown=True, onerror=None) | 遍历目录树，该方法返回一个元组，包括3个元素：所有路径名、所有目录列表与文件列表 |

示例：

>>> import os

>>> os.getcwd() #返回当前工作目录

'C:\\Python35'

>>> os.mkdir(os.getcwd()+'\\temp') #创建目录

>>> os.chdir(os.getcwd()+'\\temp') #改变当前工作目录

>>> os.getcwd()

'C:\\Python35\\temp'

>>> os.mkdir(os.getcwd()+'\\test')

>>> os.listdir('.')

['test']

>>> os.chdir('..')

>>> os.rmdir('test') #删除目录

>>> os.listdir('.')

[]

>>> os.listdir('..')

['7-1.py', '7-2.py'……]

**7.3.2 os与os.path模块**

os模块常用的文件操作函数

|  |  |
| --- | --- |
| **方法** | **功能说明** |
| **access(path, mode)** | **测试是否可以按照mode指定的权限访问文件** |
| **chdir(path)** | **把path设为当前工作目录** |
| **chmod(path, mode, \*, dir\_fd=None, follow\_symlinks=True)** | **改变文件的访问权限** |
| **curdir** | **当前文件夹** |
| **environ** | **包含系统环境变量和值的字典** |
| **extsep** | **当前操作系统所使用的文件扩展名分隔符** |
| **get\_exec\_path()** | **返回可执行文件的搜索路径** |
| **getcwd()** | **返回当前工作目录** |
| **listdir(path)** | **返回path目录下的文件和目录列表** |
| **open(path, flags, mode=0o777, \*, dir\_fd=None)** | **按照mode指定的权限打开文件，默认权限为可读、可写、可执行** |
| **popen(cmd, mode='r', buffering=-1)** | **创建进程，启动外部程序** |
| **方法** | **功能说明** | |
| **remove(path)** | **删除指定的文件，要求用户拥有删除文件的权限，并且文件没有只读或其他特殊属性** | |
| **rename(src, dst)** | **重命名文件或目录，可以实现文件的移动，若目标文件已存在则抛出异常，不能跨越磁盘或分区** | |
| **replace(old, new)** | **重命名文件或目录，若目标文件已存在则直接覆盖，不能跨越磁盘或分区** | |
| **scandir(path='.')** | **返回包含指定文件夹中所有DirEntry对象的迭代对象，遍历文件夹时比listdir()更加高效** | |
| **sep** | **当前操作系统所使用的路径分隔符** | |
| **startfile(filepath [, operation])** | **使用关联的应用程序打开指定文件或启动指定应用程序** | |
| **stat(path)** | **返回文件的所有属性** | |
| **system()** | **启动外部程序** | |
| **truncate(path, length)** | **将文件截断，只保留指定长度的内容** | |
| **write(fd, data)** | **将bytes对象data写入文件fd** |

os.path常用的文件操作函数

|  |  |
| --- | --- |
| **方法** | **功能说明** |
| **abspath(path)** | **返回给定路径的绝对路径** |
| **basename(path)** | **返回指定路径的最后一个组成部分** |
| **commonpath(paths)** | **返回给定的多个路径的最长公共路径** |
| **commonprefix(paths)** | **返回给定的多个路径的最长公共前缀** |
| **dirname(p)** | **返回给定路径的文件夹部分** |
| **exists(path)** | **判断文件是否存在** |
| **getatime(filename)** | **返回文件的最后访问时间** |
| **getctime(filename)** | **返回文件的创建时间** |
| **getmtime(filename)** | **返回文件的最后修改时间** |
| **getsize(filename)** | **返回文件的大小** |
| **isabs(path)** | **判断path是否为绝对路径** |
| **isdir(path)** | **判断path是否为文件夹** |
| **isfile(path)** | **判断path是否为文件** |
| **join(path, \*paths)** | **连接两个或多个path** |
| **realpath(path)** | **返回给定路径的绝对路径** |
| **relpath(path)** | **返回给定路径的相对路径，不能跨越磁盘驱动器或分区** |
| **samefile(f1, f2)** | **测试f1和f2这两个路径是否引用的同一个文件** |
| **split(path)** | **以路径中的最后一个斜线为分隔符把路径分隔成两部分，以元组形式返回** |
| **splitext(path)** | **从路径中分隔文件的扩展名** |
| **splitdrive(path)** | **从路径中分隔驱动器的名称** |

示例：

>>> import os

>>> import os.path

>>> os.path.exists('test1.txt')

False

>>> os.rename(‘c:\\test1.txt’,‘c:\\test2.txt’) # 此时‘c:\\test1.txt’不存在

出错信息

>>> os.rename(‘c:\\dfg.txt’,‘c:\\test2.txt’) # 可以实现文件的改名和移动

>>> os.path.exists('c:\\dfg.txt')

False

>>> os.path.exists('c:\\test2.txt')

True

小练:

统计不同类型的文件数量

dict1={}

for each\_file in os.listdir('.'):

if os.path.isdir(each\_file):

dict1.setdefault('文件夹',0)

dict1['文件夹']+=1

else:

filetype=os.path.splitext(each\_file)[1]

dict1.setdefault(filetype,0)

dict1[filetype]+=1

for i in dict1.keys():

print(i,dict1[i])

7.4编程挑战