第5章

文件与文件夹操作

程序中使用的数据都是暂时的，当程序执行终止时它们就会丢失，除非这些数据被保存起来。为了能永久地保存程序中创建的数据，需要将它们存储到磁盘或光盘上的文件中。计算机文件是以计算机硬盘为载体存储在计算机上的信息集合。文件的主要属性如下。

(1) 文件类型，即从不同的角度来对文件进行分类。

(2) 文件长度，可以用字节、字或块表示。

(3) 文件的位置，指示文件保存在哪个存储介质上以及在介质上的具体位置。

(4) 文件的存取控制，指文件的存取权限，包括读、写和执行。

(5) 文件的建立时间，指文件最近的修改时间。

从文件编码的方式来看，文件可分为文本文件和二进制文件两种。文本文件用于存储编码的字符串，二进制文件直接存储字节码。

5.1 文本文件

文本文件

5.1.1 文本文件的字符编码

文本文件是基于字符编码的文件，常见的编码有 ASCII 编码、Unicode 编码、UTF-8 编码等。在 Windows 平台中，扩展名为 txt、log、ini 的文件都属于文本文件，可以使用字处理软件(如 gedit、记事本)进行编辑。

由于计算机只能处理数字，若要处理文本，就必须先把文本转换为数字才能处理。最早的计算机采用 8 位(bit)作为 1 字节(byte)，1 字节能表示的最大的整数就是 255，如果要表示更大的整数，就必须用更多的字节。例如，2 字节可以表示的最大整数是 65 535，4 字节可以表示的最大整数是 4 294 967 295。

计算机最早使用 ASCII 编码将 127 个字母编码到计算机里。ASCII 编码是 1 字节，字节的最高位作为奇偶校验位，ASCII 编码实际使用 1 字节中的 7 位来表示字符，第一个 00000000 表示空字符，因此，ASCII 编码实际上只包括了字母、标点符号、特殊符号等共 127 个字符。

随着计算机的发展，非英语国家的人要处理他们的语言，但 ASCII 编码用上了浑身解数，把 8 位都用上也不够用。因此，后来出现了统一的、囊括多国语言的 Unicode 编码。Unicode 编码通常由 2 字节组成，一共可表示 256×256 个字符，某些偏僻字还会用到 4 字节。

在 Unicode 中，原本 ASCII 中的 127 个字符只需在前面补一个全零的字节即可，例如字符 a：01100001，在 Unicode 中变成了 00000000 01100001。这样原本只需 1 字节就能传输的英文现在变成 2 字节，非常浪费存储空间和传输速度。

针对空间浪费问题，于是出现了 UTF-8 编码。UTF-8 编码是可变长短的，从英文字母的 1 字节，到中文的通常的 3 字节，再到某些生僻字的 6 字节。UTF-8 编码还兼容了 ASCII 编码。

注意：除了英文字母相同，汉字在 Unicode 编码和 UTF-8 编码中通常是不同的。例如汉字的“中”，在 Unicode 编码中是 01001110 00101101，而在 UTF-8 编码中是 11100100 10111000 10101101。

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时，就转换为 UTF-8 编码。用记事本编辑时，从文件读取的 UTF-8 字符被转换为 Unicode 字符存储到内存里，编辑完成后，保存时再把 Unicode 字符转换为 UTF-8 字符保存到文件。浏览网页时，服务器会把动态生成的 Unicode 内容转换为 UTF-8 再传输到浏览器。

Python3 中的默认编码是 UTF-8，可以通过以下代码查看 Python3 的默认编码：

```
>>>import sys
>>>sys.getdefaultencoding()   #查看 Python3 的默认编码
'utf-8'
```

对于单个字符的编码，Python 提供了 ord()函数获取字符的整数表示，chr()函数把编码转换为对应的字符。

```
>>>ord('A')
65
>>>ord('中')
20013
>>>chr(20013)
'中'
```

由于 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。Python 对 bytes 类型的数据用带 b 前缀的单引号或双引号表示。例如：

```
x=b'ABC'
```

注意：'ABC'和 b'ABC'之间的区别，前者是 str，后者虽然内容看起来与前者一样，但 bytes 的每个字符只占用一字节。以 Unicode 表示的 str 通过 encode()方法可以编码为指定的 bytes。

```
>>>'ABC'.encode('ascii')          #编码成 ASCII 字节的形式
b'ABC'
>>>'中国'.encode('utf-8')         #编码成 UTF-8 字节的形式
b'\xe4\xb8\xad\xe5\x9b\xbd'
```

1 个中文字符经过 UTF-8 编码后通常会占用 3B,而 1 个英文字符只占用 1B。

注意:纯英文的 str 可以用 ASCII 编码为 bytes,内容是一样的,含有中文的 str 可以用 UTF-8 编码为 bytes。但含有中文的 str 无法用 ASCII 编码,因为中文编码的范围超过了 ASCII 编码的范围,Python 会报错。

要把 bytes 变为 str,就需要用 decode()方法:

```
>>>b'ABC'.decode('ascii')
'ABC'
>>>b'\xe4\xb8\xad\xe5\x9b\xbd'.decode('utf-8')
'中国'
```

在操作字符串时,人们经常遇到 str 与 bytes 的互相转换。为了避免乱码问题,应当始终坚持使用 UTF-8 编码对 str 和 bytes 进行转换。Python 源代码也是一个文本文件,所以,当源代码中包含中文时,在保存源代码时,就需要务必指定保存为 UTF-8 编码。当 Python 解释器读取源代码时,为了让它按 UTF-8 编码读取,通常在文件开头写上下面一行语句:

```
#-*-coding: utf-8-*-
```

该语句告诉 Python 解释器,按照 UTF-8 编码读取源代码,否则,在源代码中包含的中文输出时可能会有乱码。

5.1.2 文本文件的打开

向(从)一个文件写(读)数据之前,需要先创建一个与物理文件相关的文件对象,然后通过该文件对象对文件内容进行读取、写入、删除、修改等操作,最后关闭并保存文件内容。Python 内置的 open()函数可以按指定的模式打开指定的文件并创建文件对象。

```
file_object=open(file, mode='r', buffering=-1)
```

open()函数打开文件 file,返回一个指向文件 file 的文件对象 file_object。

各个参数说明如下。

file: file 是一个包含文件所在路径及文件名称的字符串值,如'c:\\User\\test. txt'。

mode: mode 指定打开文件的模式,如只读、写入、追加等,默认文件访问模式为只读'r'。

buffering: 表示是否需要缓冲,设置为 0 时,表示不使用缓冲区,直接读写,仅在二进制模式下有效。设置为 1 时,表示在文本模式下使用行缓冲区方式。设置为大于 1 时,表示缓冲区的设置大小。默认值为-1,表示使用系统默认的缓冲区大小。

文件打开的不同模式如表 5-1 所示。

表 5-1　文件打开的不同模式

模式	描　　述
r	以只读方式打开文件，文件的指针放在文件的开头。这是默认模式，可省略
rb	以只读二进制格式打开一个文件，文件的指针放在文件的开头
r+	以读写格式打开一个文件，文件指针放在文件的开头
rb+	以读写二进制格式打开一个文件，文件指针放在文件的开头
w	以写入格式打开一个文件，如果该文件已存在，则将其覆盖；如果该文件不存在，则创建新文件
wb	以二进制格式打开一个文件只用于写入，如果该文件已存在则将其覆盖；如果该文件不存在，则创建新文件
w+	以读写格式打开一个文件，如果该文件已存在则将其覆盖；如果该文件不存在，则创建新文件
wb+	以读写二进制格式打开一个文件，如果该文件已存在则将其覆盖；如果该文件不存在，则创建新文件
a	以追加格式打开一个文件，如果该文件已存在，文件指针将会放在文件的结尾，也就是说，新的内容将会被写入到已有内容之后；如果该文件不存在，创建新文件进行写入
ab	以追加二进制格式打开一个文件，如果该文件已存在，文件指针将会放在文件的结尾，也就是说，新的内容将会被写入到已有内容之后；如果该文件不存在，创建新文件进行写入
a+	以读写格式打开一个文件，如果该文件已存在，文件指针将会放在文件的结尾；如果该文件不存在，创建新文件用于读写
ab+	以读写二进制格式打开一个文件，如果该文件已存在，文件指针将会放在文件的结尾；如果该文件不存在，创建新文件用于读写

“+”表示可以同时读写某个文件。

r+：读写，即可读、可写，可理解为先读后写，不擦除原文件内容，指针在 0。

w+：写读，即可读、可写，可理解为先写后读，擦除原文件内容，指针在 0。

a+：写读，即可读、可写，不擦除原文件内容，指针指向文件的结尾，要读取原内容需先重置文件指针。

不同模式下打开文件的异同点如表 5-2 所示。

表 5-2　不同模式下打开文件的异同点

模式	可做操作	若文件不存在	是否覆盖	指针位置
r	只能读	报错	否	0
r+	可读、可写	报错	否	0
w	只能写	创建	是	0
w+	可写、可读	创建	是	0
a	只能写	创建	否，追加写	最后
a+	可读、可写	创建	否，追加写	最后

下面的语句以读的模式打开当前目录下一个名为 scores.txt 的文件。

```
file_object1=open('scores.txt', 'r')
```

也可以使用绝对路径文件名来打开文件，如下所示。

```
file_object=open(r'D:\Python\scores.txt', 'r')
```

上述语句以读的模式打开 D:\Python 目录下的 scores.txt 文件。绝对路径文件名前的 r 前缀可使 Python 解释器将文件名中的反斜杠理解为字面意义上的反斜杠。如果没有 r 前缀，需要使用反斜杠字符\转义\，使之成为字面意义上的反斜杠。

```
file_object=open('D:\\Python\\scores.txt', 'r')
```

一个文件被打开后，返回一个文件对象 file_object，通过文件对象 file_object 可以得到有关该文件的各种信息，文件对象的常用属性如表 5-3 所示。

表 5-3　文件对象的常用属性

属　性	描　述
closed	判断文件是否关闭，如果文件已被关闭，返回 True，否则返回 False
mode	返回被打开文件的访问模式
name	返回文件的名称

```
>>>file_object=open('D:\\Python\\scores.txt', 'r')
>>>print('文件名: ', file_object.name)
文件名:  D:\Python\scores.txt
>>>print('是否已关闭: ',file_object.closed)
是否已关闭: False
>>>print('访问模式: ', file_object.mode)
访问模式: r
```

文件对象是使用 open()函数来创建的，文件对象的常用方法如表 5-4 所示。文件读写操作相关的方法都会自动改变文件指针的位置。例如，以读模式打开一个文本文件，读取 10 个字符，会自动把文件指针移到第 11 个字符，再次读取字符的时候总是从文件指针的当前位置开始读取。写文件操作的方法也具有相同的特点。

表 5-4　文件对象的常用方法

方　法	功 能 说 明
close()	刷新缓冲区里还没写入的信息，并关闭该文件
flush()	刷新文件内部缓冲区，把内部缓冲区的数据立刻写入文件，但不关闭文件
next()	返回文件下一行
read([size])	从文件的开始位置读取指定的 size 个字符数，如果未给定则读取所有
readline()	读取整行，包括"\n"字符

续表

方　法	功 能 说 明
readlines()	把文本文件中的每行文本作为一个字符串存入列表中并返回该列表
seek(offset[,whence])	用于移动文件读取指针到指定位置,offset 为需要移动的字节数;whence 指定从哪个位置开始移动,默认值为 0,0 代表从文件开头开始,1 代表从当前位置开始,2 代表从文件末尾开始
tell()	返回文件的当前位置,即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 个字符,其余的删除
write(str)	把字符串 str 的内容写入文件中,没有返回值。由于缓冲,字符串内容可能没有加入到实际的文件中,直到调用 flush() 或 close()方法被调用
writelines([str])	用于向文件中写入字符串序列
writable()	测试当前文件是否可写
readable()	测试当前文件是否可读

5.1.3　文本文件的写入

当一个文件以“写”的方式打开后,可以使用 write()方法和 writelines()方法,将字符串写入文本文件。

file_object.write(str):把字符串 str 写入到文件 file_object 中,write()并不会在 str 后自动加上一个换行符。

file_object.writelines(seq):它接收一个字符串列表 seq 作为参数,把字符串列表 seq 写入到文件 file_object,这个方法也只是忠实地写入,不会在每行后面加上换行符。

```
>>>file_object=open('test.txt', 'w')  #以写的方式打开文件 test.txt
>>>file_object.write('Hello, world!') #将"Hello, world!"写入到文件 test.txt
13                                    #成功写入的字符数量
>>>file_object.close()
```

注意:可以反复调用 file_object.write()来写入文件,写完之后一定要调用 file_object.close()来关闭文件。这是因为当写文件时,操作系统往往不会立刻把数据写入磁盘,而是放到内存缓存起来,空闲的时候再慢慢写入。只有调用 close()方法时,操作系统才保证把没有写入的数据全部写入磁盘。忘记调用 close()的后果是数据可能只写了一部分到磁盘,剩下的丢失了。Python 中提供了 with 语句,可以防止上述事情的发生,当 with 代码块执行完毕时,会自动关闭文件释放内存资源,不用特意加上 file_object.close()。上面的语句可改写为如下 with 语句:

```
with open('test.txt', 'w') as file_object:
    file_object.write('Hello, world!')            #with 语句块
```

这里使用了 with 语句,不管在处理文件过程中是否发生异常,都能保证 with 语句块

执行完之后自动关闭打开的文件 test.txt。with 语句可以对多个文件同时操作。

```
>>>f=open('test.txt', 'w')
#把字符串列表["hello"," ","Python"]写入到文件 f
>>>f.writelines(["hello"," ","Python"])
>>>f.close()
>>>f=open("test.txt","r")
>>>f.read()
'hello Python'
>>>fo=open("test.txt", "w")                    #打开文件
>>>seq=["君子赠人以言\n", "庶人赠人以财"]
>>>fo.writelines(seq)                          #向文件中写入字符串序列
>>>fo.close()                                  #关闭文件
>>>fo=open("test.txt", "r")
>>>print(fo.read())
君子赠人以言
庶人赠人以财
```

【例 5-1】 创建一个新文件,内容是 0～9 的整数,每个数字占一行。

```
f=open('file1.txt','w')
for i in range(0,10):
    f.write(str(i)+'\n')
f.close()
```

5.1.4 文本文件的读取

当一个文件被打开后,可使用三种方式从文件中读取数据:read()、readline()、readlines()。

read([size]):从文件读取指定的 size 个字符数,如果未给定则读取所有。

readline():该方法每次读出一行内容,返回一个字符串对象。

readlines():把文本文件中的每行文本作为一个字符串存入列表中并返回该列表。

这里假设在当前目录下有一个文件名为 test.txt 的文本文件,里面的数据如下:

```
白日不到处
青春恰自来
苔花如米小
也学牡丹开
```

1. 读取整个文件

人们经常需要从一个文件中读取全部数据,这里有两种方法可以完成这个任务。

(1) 使用 read()方法从文件读取所有数据,然后将它作为一个字符串返回。

(2) 使用 readlines()方法从文件中读取每行文本,然后将它们作为一个字符串列表返回。

方法 1：

```
with open('test.txt') as f:                    #默认模式为'r',只读模式
    contents=f.read()                          #读取文件全部内容
    print(contents)
```

上述代码在 IDLE 中运行的结果如下：

```
白日不到处
青春恰自来
苔花如米小
也学牡丹开
```

方法 2：

```
with open('test.txt') as f:                    #默认模式为'r',只读模式
    contents1=f.readlines()                    #读取文件全部内容
    print(contents1)
```

上述代码在 IDLE 中运行的结果如下：

```
['白日不到处\n', '青春恰自来\n', '苔花如米小\n', '也学牡丹开\n']
```

2. 逐行读取

使用 read()方法和 readlines()方法从一个文件中读取全部数据，对于小文件来说是简单而且有效的，但是如果文件大到它的内容无法全部读到内存时该怎么办？这时可以编写循环，每次读取文件的一行，并且持续读取下一行直到文件末端。

方法 1：

```
with open('test.txt') as f:
    for line in f:
        print(line, end='')
```

上述代码在 IDLE 中运行的结果如下：

```
白日不到处
青春恰自来
苔花如米小
也学牡丹开
```

方法 2：

```
f=open("test.txt")
line=f.readline()
print(type(line))          #输出 line 的数据类型
while line:
    print(line, end='')
    line=f.readline()
```

```
f.close()
```

上述代码在 IDLE 中运行的结果如下：

```
<class 'str'>
白日不到处
青春恰自来
苔花如米小
也学牡丹开
```

5.1.5 文本文件指针的定位

文件对象的 tell()方法返回文件的当前位置，即文件指针当前位置。使用文件对象的 read()方法读取文件之后，文件指针到达文件的末尾，如果再来一次 read()将会发现读取的是空内容，如果想再次读取全部内容，或读取文件中的某行字符，必须将文件指针移动到文件开始或某行开始，这可通过文件对象的 seek()方法来实现，其语法格式如下：

```
seek(offset[,whence])
```

作用：用于移动文件读取指针到指定位置，offset 为需要移动的字节数；whence 指定从哪个位置开始移动，默认值为 0，0 代表从文件开头开始，1 代表从当前位置开始，2 代表从文件末尾开始。

注意：Python3 不允许非二进制打开的文件，相对于文件末尾的定位。

```
>>>f=open('file2.txt', 'a+')
>>>f.write('123456789abcdef')
15
>>>f.seek(3)                          #移动文件指针，并返回移动后的文件指针当前位置
3
>>>f.read(1)
'4'
>>>f.seek(-3,2)                       #报错
Traceback(most recent call last):
  File "<pyshell#5>", line 1, in <module>
    f.seek(-3,2)
io.UnsupportedOperation: can't do nonzero end-relative seeks
>>>f.close()
>>>f=open('file2.txt', 'rb+')         #以二进制模式读写文件
>>>f.seek(-3,2)                       #移动文件指针，并返回移动后的文件指针当前位置
12                                     #没有报错
>>>f.tell()                           #返回文件指针当前位置
12
>>>f.read(1)
b'd'
```

【例 5-2】 修改模式下打开文件，然后输出，观察指针区别。

其中，file2.txt 的内容如下：

```
123456789abcdef
```

程序代码：

```
f=open(r'D:\Python\file2.txt','r+')
print('文件指针在:',f.tell())
if f.writable():
    f.write('Python\n')
else:
    print("此模式不可写")
print('文件指针在:',f.tell())
f.seek(0)
print("最后的文件内容: ")
print(f.read())
f.close()
```

程序代码在 IDLE 中运行的结果如下：

```
文件指针在: 0
文件指针在: 8
最后的文件内容:
Python
9abcdef
```

5.2　二进制文件

二进制文件是基于值编码的文件，二进制文件直接存储字节码，可以根据具体应用，指定某个值是什么意思(这样一个过程，可以看作是自定义编码)。二进制文件可看成是变长编码的，多少个比特代表一个值，完全由用户决定。二进制文件编码是变长的，存储利用率高，但译码难(不同的二进制文件格式，有不同的译码方式)。常见的图形图像文件、音频和视频文件、可执行文件、资源文件、各种数据库文件等均属于二进制文件。

5.2.1　二进制文件的写入

二进制文件的写入一般包括三个步骤：打开文件、写入数据和关闭文件。

通过内置函数 open()函数可以创建或打开二进制文件，返回一个指向文件的文件对象。

```
>>>f1=open('data1', 'rb')        #以只读二进制格式打开一个文件
>>>f2=open('data2', 'wb')        #以二进制格式创建或打开一个文件只用于写入
```

以二进制的方式打开二进制文件后，可以使用文件对象的 write()方法将二进制数据写入文件。可以使用文件对象的 flush()方法强制把缓冲的数据更新到文件中。