# 第9章 文件内容操作

# 第9章 文件内容操作

- 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。
- 文件操作在各类应用软件的开发中均占有重要的地位：
  - ✓ 管理信息系统是使用数据库来存储数据的，而数据库最终还是要以文件的形式存储到硬盘或其他存储介质上。
  - ✓ 应用程序的配置信息往往也是使用文件来存储的，图形、图像、音频、视频、可执行文件等等也都是以文件的形式存储在磁盘上的。

# 第9章 文件内容操作

- 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。

✓ **文本文件**：文本文件存储的是常规字符串，由若干文本行组成，通常每行以换行符'\n'结尾。常规字符串是指记事本或其他文本编辑器能正常显示、编辑并且人类能够直接阅读和理解的字符串，如英文字母、汉字、数字字符串。文本文件可以使用字处理软件如gedit、记事本进行编辑。

✓ **二进制文件**：二进制文件把对象内容以字节串(bytes)进行存储，无法用记事本或其他普通字处理软件直接进行编辑，通常也无法被人类直接阅读和理解，需要使用专门的软件进行解码后读取、显示、修改或执行。常见的如图形图像文件、音视频文件、可执行文件、资源文件、各种数据库文件、各类office文档等都属于二进制文件。

# 9.1 文件操作基本知识

- 无论是文本文件还是二进制文件，其操作流程基本都是一致的
- 首先打开文件并创建文件对象
- 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。

# 9.1.1 内置函数open()

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,
     newline=None, closefd=True, opener=None)
```

- ✓ file参数指定了被打开的文件名称。
- ✓ mode参数指定了打开文件后的处理方式。
- ✓ buffering参数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。
- ✓ encoding参数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。

# 9.1.1 内置函数open()

- 文件打开模式

| 模式 | 说明 |
| --- | --- |
| r | 读模式（**默认模式**，可省略），如果文件不存在则抛出异常 |
| w | 写模式，如果文件已存在，先清空原有内容 |
| x | 写模式，创建新文件，如果文件已存在则抛出异常 |
| a | 追加模式，不覆盖文件中原有内容 |
| b | 二进制模式（可与其他模式组合使用） |
| t | 文本模式（**默认模式**，可省略） |
| + | 读、写模式（可与其他模式组合使用） |

# 9.1.1 内置函数open()

- 如果执行正常，open()函数返回1个文件对象，通过该文件对象可以对文件进行读写操作。如果指定文件不存在、访问权限不够、磁盘空间不足或其他原因导致创建文件对象失败则抛出异常。

```
f1 = open( 'file1.txt', 'r' )      # 以读模式打开文件
f2 = open( 'file2.txt', 'w')       # 以写模式打开文件
```

- 当对文件内容操作完以后，一定要关闭文件对象，这样才能保证所做的任何修改都确实被保存到文件中。

```
f1.close()
```

# 9.1.2 文件对象属性与常用方法

| 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 |
| 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 |
| 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 |
| seek(offset[, whence]) | 把文件指针移动到新的字节位置，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 |
| tell() | 返回文件指针的当前位置 |
| write(s) | 把s的内容写入文件 |
| writelines(s) | 把字符串列表写入文本文件，不添加换行符 |

# 9.1.3 上下文管理语句with

- 在实际开发中，读写文件应优先考虑使用上下文管理语句with，关键字with可以自动管理资源，不论因为什么原因（哪怕是代码引发了异常）跳出with块，总能保证文件被正确关闭，并且可以在代码块执行完毕后自动还原进入该代码块时的上下文，常用于文件操作、数据库连接、网络连接、多线程与多进程同步时的锁对象管理等场合。

```
with open(filename, mode, encoding) as fp:
    #这里写通过文件对象fp读写文件内容的语句
```

- 上下文管理语句with还支持下面的用法：

```
with open('test.txt', 'r') as src, open('test_new.txt', 'w') as dst:
    dst.write(src.read())
```

# 9.2 文本文件内容操作案例精选

- **示例9-1** 向文本文件中写入内容，然后再读出。

```
s = 'Hello world\n文本文件的读取方法\n文本文件的写入方法\n'

with open('sample.txt', 'w') as fp:
    fp.write(s)

with open('sample.txt') as fp:
    print(fp.read())
```

# 9.2 文本文件内容操作案例精选

- **示例9-2** 将一个CP936编码格式的文本文件中的内容全部复制到另一个使用UTF8编码的文本文件中。

```
def fileCopy(src, dst, srcEncoding, dstEncoding):
    with open(src, 'r', encoding=srcEncoding) as srcfp:
        with open(dst, 'w', encoding=dstEncoding) as dstfp:
            dstfp.write(srcfp.read())

fileCopy('sample.txt', 'sample_new.txt', 'cp936', 'utf8')
```

# 9.2 文本文件内容操作案例精选

- **示例9-3** 遍历并输出文本文件的所有行内容。

```
with open('sample.txt') as fp:          #假设文件采用CP936编码
    for line in fp:                     #文件对象可以直接迭代
        print(line)
```

# 9.2 文本文件内容操作案例精选

- **示例9-4** 假设已有一个文本文件sample.txt，将其中第13、14两个字符修改为测试。

```
with open('sample.txt', 'r+') as fp:
    fp.seek(13)
    fp.write('测试')
```

# 9.2 文本文件内容操作案例精选

- **示例9-5** 假设文件data.txt中有若干整数，所有整数之间使用英文逗号分隔，编写程序读取所有整数，将其按升序排序后再写入文本文件data_asc.txt中。

```
with open('data.txt', 'r') as fp:
    data = fp.readlines()                          #读取所有行
data = [line.strip() for line in data]             #删除每行两侧的空白字符
data = ','.join(data)                              #合并所有行
data = data.split(',')                             #分隔得到所有数字字符串
data = [int(item) for item in data]                #转换为数字
data.sort()                                        #升序排序
data = ','.join(map(str,data))                     #将结果转换为字符串
with open('data_asc.txt', 'w') as fp:              #将结果写入文件
    fp.write(data)
```

# 9.2 文本文件内容操作案例精选

- **示例9-6** 统计文本文件中最长行的长度和该行的内容。

```
with open('sample.txt') as fp:
    result = [0, '']
    for line in fp:
        t = len(line)
        if t > result[0]:
            result = [t, line]
print(result)
```

# 9.2 文本文件内容操作案例精选

- **示例9-7** 使用标准库json进行数据交换。

```
>>> import json
>>> with open('test.txt', 'w') as fp:
    json.dump({'a':1, 'b':2, 'c':3}, fp) #写入文件

>>> with open('test.txt', 'r') as fp:
    print(json.load(fp))                  #从文件中读取

{'a': 1, 'b': 2, 'c': 3}
```

# 9.2 文本文件内容操作案例精选

- **示例9-8** 使用csv模块读写文件内容。

```
>>> import csv
>>> with open('test.csv', 'w', newline='') as fp:
    test_writer = csv.writer(fp, delimiter=' ', quotechar='"')
    test_writer.writerow(['red', 'blue', 'green'])  #写入一行内容
    test_writer.writerow(['test_string']*5)

>>> with open('test.csv', newline='') as fp:
    test_reader = csv.reader(fp, delimiter=' ', quotechar='"')
    for row in test_reader:                          #遍历所有行
        print(row)                                   #每行作为一个列表返回
['red', 'blue', 'green']
['test_string', 'test_string', 'test_string', 'test_string', 'test_string']
```