关于作者

python免费公开课
编程学习网
授课模式：在线直播+课后视频，从零基础到中高级开发工程师
查看详情

文件读写

读写文件是最常见的IO操作。Python内置了读写文件的函数，用法和C是兼容的。

读写文件前，我们先必须了解一下，在磁盘上读写文件的功能都是由操作系统提供的，现代操作系统不允许普通的程序直接操作磁盘，所以，读写文件就是请求操作系统打开一个文件对象（通常称为文件描述符），然后，通过操作系统提供的接口从这个文件对象中读取数据（读文件），或者把数据写入这个文件对象（写文件）。

读文件

要以读文件的模式打开一个文件对象，使用Python内置的 `open()` 函数，传入文件名和标示符：

```
>>> f = open('/Users/michael/test.txt', 'r')
```

标示符'r'表示读，这样，我们就成功地打开了一个文件。

如果文件不存在，`open()` 函数就会抛出一个 `IOError` 的错误，并且给出错误码和详细的信息告诉你文件不存在：

```
>>> f=open('/Users/michael/notfound.txt', 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: '/Users/michael/notfound.txt'
```

如果文件打开成功，接下来，调用 `read()` 方法可以一次读取文件的全部内容，Python把内容读到内存，用一个 `str` 对象表示：

```
>>> f.read()
'Hello, world!'
```

最后一步是调用 `close()` 方法关闭文件。文件使用完毕后必须关闭，因为文件对象会占用操作系统的资源，并且操作系统同一时间能打开的文件数量也是有限的：

```
>>> f.close()
```

由于文件读写时都有可能产生 `IOError`，一旦出错，后面的 `f.close()` 就不会调用。所以，为了保证无论是否出错都能正确地关闭文件，我们可以使用 `try ... finally` 来实现：

```
try:
    f = open('/path/to/file', 'r')
    print(f.read())
finally:
    if f:
        f.close()
```

但是每次都这么写实在太繁琐，所以，Python引入了 `with` 语句来自动帮我们调用 `close()` 方法：

```
with open('/path/to/file', 'r') as f:
    print(f.read())
```

这和前面的 `try ... finally` 是一样的，但是代码更佳简洁，并且不必调用 `f.close()` 方法。

调用 `read()` 会一次性读取文件的全部内容，如果文件有10G，内存就爆了，所以，要保险起见，可以反复调用 `read(size)` 方法，每次最多读取size个字节的内容。另外，调用 `readline()` 可以每次读取一行内容，调用 `readlines()` 一次读取所有内容并按行返回 `list`。因此，要根据需要决定怎么调用。

如果文件很小，`read()` 一次性读取最方便；如果不能确定文件大小，反复调用 `read(size)` 比较保险；如果是配置文件，调用 `readlines()` 最方便：

```
for line in f.readlines():
    print(line.strip()) # 把末尾的'\n'删掉
```

file-like Object

像 `open()` 函数返回的这种有个 `read()` 方法的对象，在Python中统称为file-like Object。除了file外，还可以是内存的字节流，网络流，自定义流等等。file-like Object不要求从特定类继承，只要写个 `read()` 方法就行。

`StringIO` 就是在内存中创建的file-like Object，常用作临时缓冲。

二进制文件

前面讲的默认都是读取文本文件，并且是UTF-8编码的文本文件。要读取二进制文件，比如图片、视频等等，用 `'rb'` 模式打开文件即可：

```
>>> f = open('/Users/michael/test.jpg', 'rb')
>>> f.read()
b'\xff\xd8\xff\xe1\x00\x18Exif\x00\x00...' # 十六进制表示的字节
```

字符编码

要读取非UTF-8编码的文本文件，需要给 `open()` 函数传入 `encoding` 参数，例如，读取GBK编码的文件：

```
>>> f = open('/Users/michael/gbk.txt', 'r', encoding='gbk')
>>> f.read()
'测试'
```

遇到有些编码不规范的文件，你可能会遇到 `UnicodeDecodeError`，因为在文本文件中可能夹杂了一些非法编码的字符。遇到这种情况，`open()` 函数还接收一个 `errors` 参数，表示如果遇到编码错误后如何处理。最简单的方式是直接忽略：

```
>>> f = open('/Users/michael/gbk.txt', 'r', encoding='gbk', errors='ignore')
```

写文件

写文件和读文件是一样的，唯一区别是调用 `open()` 函数时，传入标识符 `'w'` 或者 `'wb'` 表示写文本文件或写二进制文件：

```
>>> f = open('/Users/michael/test.txt', 'w')
>>> f.write('Hello, world!')
>>> f.close()
```

你可以反复调用 `write()` 来写入文件，但是务必要调用 `f.close()` 来关闭文件。当我们写文件时，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。所以，还是用 `with` 语句来得保险：

```
with open('/Users/michael/test.txt', 'w') as f:
    f.write('Hello, world!')
```

要写入特定编码的文本文件，请给 `open()` 函数传入 `encoding` 参数，将字符串自动转换成指定编码。

细心的童鞋会发现，以 `'w'` 模式写入文件时，如果文件已存在，会直接覆盖（相当于删掉后新写入一个文件）。如果我们希望追加到文件末尾怎么办？可以传入 `'a'` 以追加（append）模式写入。

所有模式的定义及含义可以参考Python的官方文档。

练习

请将本地一个文本文件读为一个str并打印出来：

```
# -*- coding: utf-8 -*-
fpath = r'C:\Windows\system.ini'

with open(fpath, 'r') as f:
    s = f.read()
    print(s)
# 运行代码观察结果
```

▶ Run

小结

在Python中，文件读写是通过 `open()` 函数打开的文件对象完成的。使用 `with` 语句操作文件IO是个好习惯。

参考源码

with_file.py

读后有收获可以请作者喝咖啡，读后有疑问请加群讨论：

还可以分享给朋友：

分享到微博

上一页 下一页

评论

实现一个简单的with上下文管理对象

ywjco_567 created at 2 days ago, Last updated at 2 days ago

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# FileName: with_example.py

class Sample(object):
        def __init__(self, name):
                self.name = name

        def __enter__(self):
                print(f'with语句 "{self.name}" 的__enter__被触发...')
                return self
```