文件读写

1714次阅读

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>
IOError: [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ASCII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()
'\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ASCII编码的文本文件，就必须以二进制模式打开，再解码。比如GBK编码的文件：

```
>>> f = open('/Users/michael/gbk.txt', 'rb')
>>> u = f.read().decode('gbk')
>>> u
u'\u6d4b\u8bd5'
>>> print u
测试
```

如果每次都这么手动转换编码嫌麻烦（写程序怕麻烦是好事，不怕麻烦就会写出又长又难懂又没法维护的代码），Python还提供了一个codecs模块帮我们在读文件时自动转换编码，直接读出unicode：

```
import codecs
with codecs.open('/Users/michael/gbk.txt', 'r', 'gbk') as f:
    f.read() # u'\u6d4b\u8bd5'
```

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，请效仿codecs的示例，写入unicode，由codecs自动转换成指定编码。

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。
