**读写文件内容**

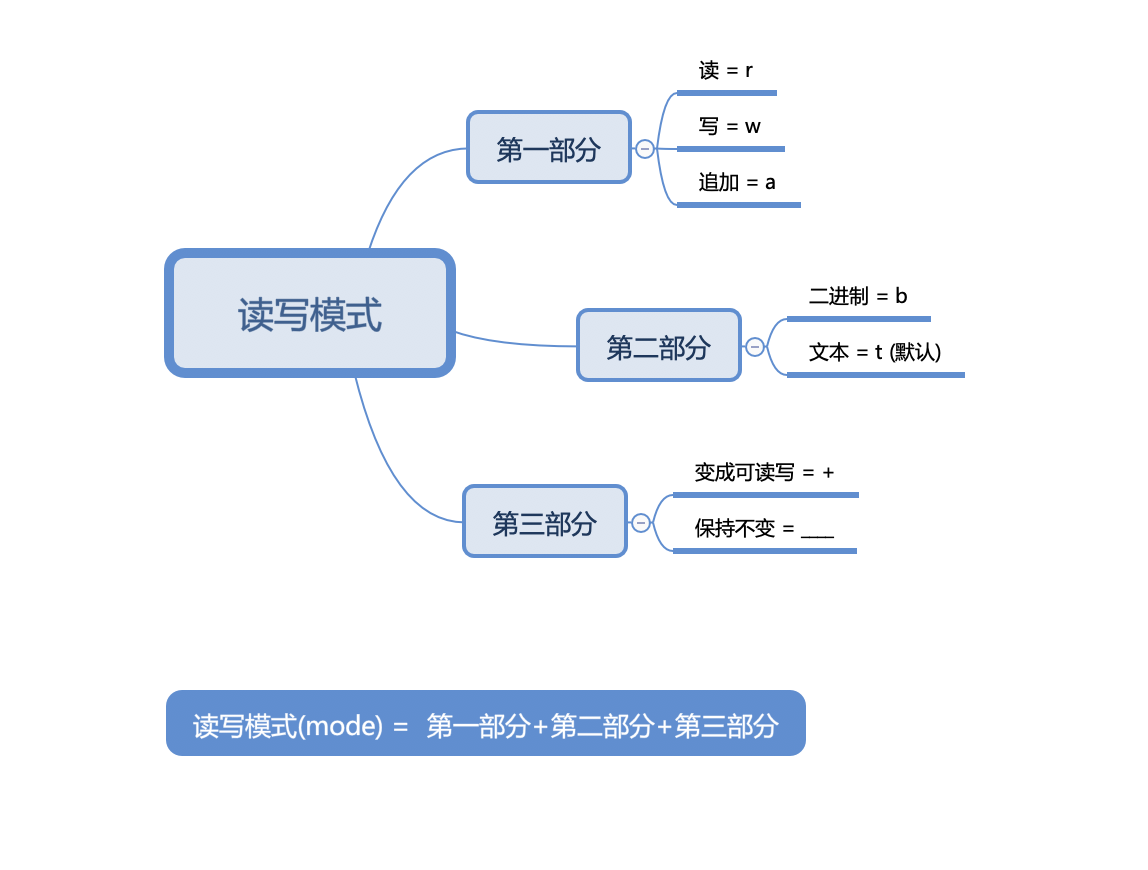
|  |  |
| --- | --- |
| 课题 | 读写文件内容 |
| 课程内容 | 如何从文件读取信息，如何将信息写入文件，并且编程实现文件的复制。 |
| 前置知识 | 基础语法，异常捕获 |
| 课程重点 | 读写文件的模式  read()\readline()\readlines()函数的区别  seek() 函数，控制读写位置  with open语句  read()\readline()\readlines()函数的区别 |
| 本课单词 | seek 寻找 、append 追加 |
| 课注意事项 |  |

## Python 文件读写模式

所谓 “读”, 就是读取文件中的数据到 Python 程序中.

所谓 “写”, 就是将Python程序中的数据输出到某个文件.

在 Python 想要打开一个文件进行读写, 需要指定一个**模式**, 而一个模式就由下图的三个部分组成, 每一个部分都需要选择一个选项, 最后进行拼接.



1. 第一部分:

r 从头开始读文件

w 从头开始写文件

a 从末尾开始写文件，追加内容

1. 第二部分:

b 以二进制方式操作文件

t 以文本文件方式操作文件, 如果第二个部分不填写,则默认为 “t”

1. 第三部分:

+ 将操作方式变为读写模式,不管第一部分是 r 还是 w

什么都不写表示不需要第三部分

下图做了一份总结：



比如:

“r” = 文本形式读取文件,

"rb" = 以二进制的形式读取文件

“r+” = 以读的模式打开文件，并以文本形式处理（指针在开头，会逐字覆盖后面的内容），但也可以写入信息.

"w+" = 以写的模式打开文件，并以文本形式处理（指针在开头，会清空原来的内容），但也可以读取信息

注意: 所谓 “覆盖后面” 即表示, 以该模式打开文件后, 会先将文件内的内容全部清空.

w+与r+ 模式的区别：

“w+” 读写文本文件,但是 “w+” 会将原文件中所有内容全部清空后再进行写入。

“r+” 读写文本文件, 而 “r+” 并不会清空原文件.

## “r”模式，读取文件内容

在桌面上创建一个文本文本, 命名为 “test.txt”，文件中输入 “Hello Python”。并保存。

使用如下代码进行“读”操作, 并将读入的结果打印出来。

# 以“只读模式”获取指定文件的操作信息放入 file 变量, 之后将可以通过

# file 对指定的文件进行操作.

f = open('C:/Users/Lema/Desktop/test.txt', 'r')

# 将 file 对应的文件的全部内容读入 file\_content

file\_content = f.read()

# 将文件关闭, 否则文件将处于被引用状态,若该文件位于 u 盘,则 u 盘是无法

# 正常弹出的

f.close()

print(file\_content)

用open()函数打开文件时，如果文件不存在，会报IOError错误。

注意：读取中文字符时，要注意文件的编码格式，如果文件是Unicode格式保存的文件，@×那么就需要UTF-8来解码。如果文件是以ANSI格式保存，那么在中国ANSI编码就是指GBK编码，所以解码时可以用GBK解码。

（不了解ANSI的学员，可参考-《字符编码》一节相关内容）

f=open('C:/Users/Lema/Desktop/test.txt', 'r'，encoding='UTF-8')

## “w”模式，创建文件并写入内容

直接使用如下代码进行“写”操作. Python 将在你指定的位置创建一个 not\_exist.txt 文件, 然后在此文件中写入 “Hello 空文件”.

# w模式，如果not\_exist.txt文件不存在，就会自动创建一个文件

f = open("c:/User/Lema/Desktop/not\_exist.txt",'w')

# 让 file 将指定信息写入文件, file 一般不会马上写入.

f.write("Hello 空文件")

# 写入文件的信息,一般先保存在内存之中,关闭之后,会将内存中的数据写入文件

f.close()

如果用"w"模式时，想要读取文件内容，会导致程序出错。

f = open("c:/User/Lema/Desktop/not\_exist.txt",'w')

f.write("Hello 空文件")

# 尝试进行读取, 因为是只写模式, 故 Python 会报出异常

file\_content = file.read()

f.close()



## “w”模式，写入已经存在的文本文件

在桌面上创建一个文本文本, 命名为 “test.txt”，写入 “Hello Python.”

使用如下代码进行“写”操作

# “w” 模式在写入已经存在的文本文件时会将其中的内容全部清空,再写入

# 指定内容

file = open("c:\\User\\Lema\\Desktop\\test.txt",'w')

file.write("123") # 擦除文件所有内容，再写入

file.close()

运行程序之后,打开 test.txt 会发现, 原本的内容“Hello Python”变成了 “123”。文件内容全部被抹掉，重写了。

## “r+”模式，读写

‘r+’ 模式打开文件,既可以读取文件,也可以写入文件。写入文件的方式

是从文件头部向后逐字覆盖。

在桌面上创建一个文本文本, 命名为 “test.txt”，写入 “Hello Python.”

使用如下代码进行“读写”操作

file = open("c:/User/Lema/Desktop/test.txt",'r+')

file.write("123")

# 将光标后面的内容读取出来

file\_content = file.read()

file.close()

print(file\_content)

程序输出: “lo Python”，并没有输出“123lo Python”。因为写入123之后，指针位于字符 "l" 的位置。

运行程序后，“test.txt” 文本文件中的内容将变为: 123lo Python

注意：前3个字母，hel 被改成了123。并没有把整个文件的所有内容覆盖。

## “w+”模式，写读模式

模式打开文件,既可以读取文件,也可以写入文件。写入文件的方式，是从文件头部向后全文覆盖。

在桌面上创建一个文本文本, 命名为 “test.txt”，写入 “Hello Python”。

使用如下代码进行“写”操作

# ‘w+’ 模式打开文件,既可以读取文件,也可以写入文件,不过写入文件的方式

# 是先将文件中的内容全部清空,然后把程序指定的数据写

file = open("c:\\User\\Lema\\Desktop\\test.txt",'w+')

file.write("123")

file\_content = file.read()

file.close()

print(file\_content)

程序输出: 123

运行程序后，“test.txt” 文本文件中的内容将变为: 123

注意：原文中的“Hello Python“ 整个内容都被擦除了，而不是像“r+”模式，只是逐字覆盖了前几个字符。

## “a”模式，尾部追加

在桌面上创建一个文本文本, 命名为 “test.txt”

在 “test.txt” 中写入 “Hello Python.”

使用如下代码进行“写”操作

# 追加模式打开文件,并不会同 “w“ 模式一样清空其中内容.

# 追加写入文件的内容，会被放到文件的末尾

file = open("C:\\Users\\Lema\\Desktop\\test.txt","a")

file.write("123")

file.close()

运行程序后，“test.txt” 文本文件中的内容将变为: Hello Python123

## readline()每次读取一行

在桌面上创建一个文本文本, 命名为 “test.txt”

在 “test.txt” 中写入如下文字:

我是第一行

我是第二行

我是第三行

我是最后一行

使用如下代码读取文件中每一行:

file = open("C:\\Users\\Lema\\Desktop\\test.txt","r")

line = file.readline()

# 我们一般不知道一个文件中有多少行.

# readline 在读取到文件末尾的时候会返回一个空字符串,

# 我们可以以此来判断是否已经读取了所有行.

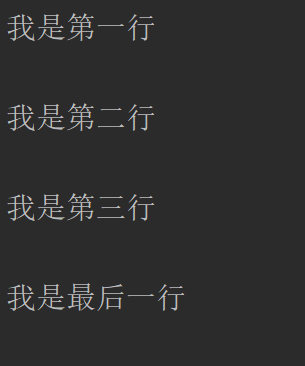
while line != "":

print(line)

line = file.readline()

file.close()

输出结果:



注意: 由于 print 会自动在每一次输出完毕后加入一个换行符, 故程序输出结果较原文本文件每行中多了一个空行。可以使用 print(line, end="")

## readlines()读取所有行,返回列表

在桌面上创建一个文本文本, 命名为 “test.txt”

在 “test.txt” 中写入如下文字:

我是第一行

我是第二行

我是第三行

我是最后一行

使用如下代码一次读取文件内容，并按行返回一个列表:

file = open("C:\\Users\\Lema\\Desktop\\test.txt","r")

# all\_lines 是一个列表,列表中每一项都是文件中的一行

all\_lines = file.readlines()

print(all\_lines)

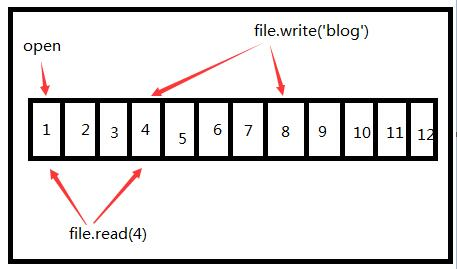
输出结果： file.readlines() 返回的是一个列表：



注意：readlines() 与read()函数都是一次性把文件内容都放入内存，如果文件太大，会导致内存崩溃。

## seek指定开始读写的位置

对文件进行读写操作的时候，本质上是有一个指针，指向当前要操作的位置。刚打开文件时，指针指向开头的位置。



file.seek(offset, whence=0)函数能够控制指针的移动。

offset : 距离 whence 的偏移量

whence : 指定偏移量的起始位置。可填0或1或2.

* 0 : 整个文件开头位置
* 1 : 当前位置 ( offset 必须为 0, 否则报错)
* 2 : 文件末尾 ( offset 必须为 0, 否则报错)

例: seek(0,2) 定位到文件末尾(‘abcd’的末尾不是 b, 而是一个空字符 )

例: seek(3,0) 定位到文件开始第 4 个字节 (每一个字节从 0 开始编号)

下面继续来看一个示例：

在桌面上创建一个文本文本, 命名为 “test.txt”，写入“ABCD”

使用如下代码读取第2个字符:

file = open("C:\\Users\\Lema\\Desktop\\test.txt","r")

file.seek(1,0)# 若文件内容为 ABCD, 则当前读写位置在 B.

c = file.read(1) # c得到 'B'

print(c)

file.close()

# file.read(1) 读取当前位置的一个字符

# file.read() 读取当前位置往后的所有字符

使用如下代码读取第2,3个字符:

file = open("C:\\Users\\Lema\\Desktop\\test.txt","r")

file.seek(1,0)

c = file.read(2) # 读取2个字符，c得到 'BC'

print(c)

file.close()

## with open 语句

使用 open函数() 打开文件后, 可能会忘记在最后使用 close 关闭文件。

由于文件读写时都有可能产生IOError，一旦出错，后面的 f1.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 语句，来捕获异常，并最终保证关闭文件：

try:

f = open('文件路径', 'r')

content = r.read()

print(content)

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 open语句，文件使用完后，会自动关闭文件，即使出现异常也可以正常关闭。

注意：with open内部的代码要缩进一层。

# 代码A

with open("文件路径", "r") as f:

content = f.read()

print(content)

## 通过文件读写实现二进制文件复制

从网上任意找一张图片,将其命名为 “图片A.jpg”

注意: 你寻找到的图片可能后缀为 “.png”, 此时命名为 “图片A.png” 即可。

使用如下代码将“图片A”复制一份,命名为“图片B”:

# 二进制模式打开图片A,读出所有二进制数据

with open("C:\\Users\\Lema\\Desktop\\图片A.jpg",'rb') as f1:

c = f1.read()

# 二进制模式打开图片B(不存在则创建)\后,写入图片A的二进制数据

with open("C:\\Users\\Lema\\Desktop\\图片B.jpg",'wb') as f2

f2.write(c)