**Python os 模块详解**

**1. 简介**

os就是“operating system”的缩写，顾名思义，os模块提供的就是各种 Python 程序与操作系统进行交互的接口。通过使用os模块，一方面可以方便地与操作系统进行交互，另一方面页可以极大增强代码的可移植性。如果该模块中相关功能出错，会抛出OSError异常或其子类异常。

注意，如果是读写文件的话，建议使用内置函数open()；如果是路径相关的操作，建议使用os的子模块os.path；如果要逐行读取多个文件，建议使用fileinput模块；要创建临时文件或路径，建议使用tempfile模块；要进行更高级的文件和路径操作则应当使用shutil模块。

当然，使用os模块可以写出操作系统无关的代码并不意味着os无法调用一些特定系统的扩展功能，但要切记一点：一旦这样做就会极大**损害代码的可移植性**。

此外，导入os模块时还要小心一点，千万**不要**为了图调用省事儿而将os模块解包导入，即不要使用from os import \*来导入os模块；否则os.open()将会覆盖内置函数open()，从而造成预料之外的错误。

**2. 常用功能**

注意，os模块中大多数接受路径作为参数的函数也可以接受“文件描述符”作为参数。

文件描述符：file descriptor，在 Python 文档中简记为 fd，是一个与某个打开的文件对象绑定的整数，可以理解为该文件在系统中的编号。

**2.1 os.name**

该属性宽泛地指明了当前 Python 运行所在的环境，实际上是导入的操作系统相关模块的名称。这个名称也决定了模块中**哪些功能是可用**的，哪些是没有相应实现的。

目前有效名称为以下三个：posix，nt，java。

其中posix是 Portable Operating System Interface of UNIX（可移植操作系统接口）的缩写。Linux 和 Mac OS 均会返回该值；nt全称应为“Microsoft Windows NT”，大体可以等同于 Windows 操作系统，因此 Windows 环境下会返回该值；java则是 Java 虚拟机环境下的返回值。

因此在我的电脑（win10）上执行下述代码，返回值是nt：

**>>>** **import** os

**>>>** os**.**name

'nt'

而在 WSL（Windows Subsystem Linux，Windows 下的 Linux 子系统）上的结果则是：

**>>>** **import** os

**>>>** os**.**name

'posix'

查看sys模块中的sys.platform属性可以得到关于运行平台更详细的信息，在此不再赘述

**2.2 os.environ**

os.environ属性可以返回环境相关的信息，主要是各类环境变量。返回值是一个映射（类似字典类型），具体的值为第一次导入os模块时的快照；其中的各个键值对，键是环境变量名，值则是环境变量对应的值。在第一次导入os模块之后，除非直接修改os.environ的值，否则该属性的值不再发生变化。

比如其中键为“HOMEPATH”（Windows 下，Linux 下为“HOME”）的项，对应的值就是用户主目录的路径。Windows 下，其值为：

**>>>** os**.**environ["HOMEPATH"]

'd:\\justdopython'

Linux 下，其值为：

**>>>** os**.**environ["HOME"]

'/home/justdopython'

**2.3 os.walk()**

这个函数需要传入一个路径作为top参数，函数的作用是在以top为根节点的目录树中游走，对树中的每个目录生成一个由(dirpath, dirnames, filenames)三项组成的三元组。

其中，dirpath是一个指示这个目录路径的字符串，dirnames是一个dirpath下子目录名（除去“.”和“..”）组成的列表，filenames则是由dirpath下所有非目录的文件名组成的列表。要注意的是，这些名称并不包含所在路径本身，要获取dirpath下某个文件或路径从top目录开始的完整路径，需要使用os.path.join(dirpath, name)。

注意最终返回的结果是一个迭代器，我们可以使用for语句逐个取得迭代器的每一项：

**>>>** **for** item **in** os**.**walk("."):

**...** **print**(item)

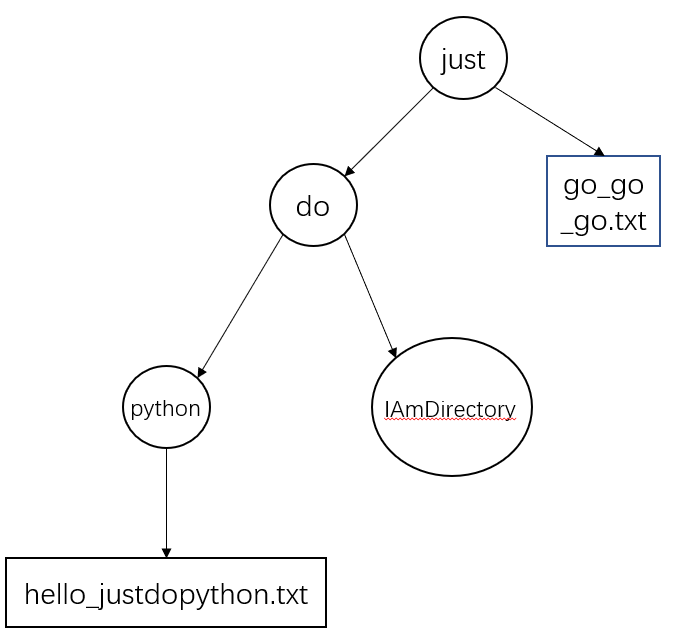
**...**

('.', ['do'], ['go\_go\_go.txt'])

('.\\do', ['IAmDirectory', 'python'], [])

('.\\do\\IAmDirectory', [], [])

('.\\do\\python', [], ['hello\_justdopython.txt'])



**2.4 os.listdir()**

“listdir”即“list directories”，列出（当前）目录下的全部路径（及文件）。该函数存在一个参数，用以指定要列出子目录的路径，默认为“.”，即“当前路径”。

函数返回值是一个列表，其中各元素均为字符串，分别是各路径名和文件名。

通常在需要遍历某个文件夹中文件的场景下极为实用。

比如定义以下函数：

**def** **get\_filelists**(file\_dir**=**'.'):

list\_directory **=** os**.**listdir(file\_dir)

filelists **=** []

**for** directory **in** list\_directory:

*# os.path 模块稍后会讲到*

**If** os**.**path**.**isfile(directory):

filelists**.**append(directory)

**return** filelists

该函数的返回值就是当前目录下所有文件而非文件夹的名称列表。

**2.5 os.mkdir()**

“mkdir”，即“make directory”，用处是“新建一个路径”。需要传入一个类路径参数用以指定新建路径的位置和名称，如果指定路径已存在，则会抛出FileExistsError异常。

该函数只能在已有的路径下新建一级路径，否则（即新建多级路径）会抛出FileNotFoundError异常。

相应地，在需要新建多级路径的场景下，可以使用os.makedirs()来完成任务。函数os.makedirs()执行的是递归创建，若有必要，会分别新建指定路径经过的中间路径，直到最后创建出末端的“叶子路径”。

示例如下：

**>>>** os**.**mkdir("test\_os\_mkdir")

**>>>** os**.**mkdir("test\_os\_mkdir")

Traceback (most recent call last):

File "<stdin>", line 1, **in** **<**module**>**

FileExistsError: [WinError 183] 当文件已存在时，无法创建该文件。: 'test\_os\_mkdir'

**>>>**

**>>>** os**.**mkdir("test\_os\_mkdir/test\_os\_makedirs/just/do/python/hello")

Traceback (most recent call last):

File "<stdin>", line 1, **in** **<**module**>**

FileNotFoundError: [WinError 3] 系统找不到指定的路径。: 'test\_os\_mkdir/test\_os\_makedirs/just/do/python/hello'

**>>>**

**>>>** os**.**makedirs("test\_os\_mkdir/test\_os\_makedirs/just/do/python/hello")

**2.6 os.remove()**

用于删除文件，如果指定路径是目录而非文件的话，就会抛出 IsADirectoryError异常。删除目录应该使用os.rmdir()函数。

同样的，对应于os.makedirs()，删除路径操作os.rmdir()也有一个递归删除的函数os.removedirs()，该函数会尝试从最下级目录开始，逐级删除指定的路径，几乎就是一个os.makedirs()的逆过程；一旦遇到非空目录即停止。

**2.7 os.rename()**

该函数的作用是将文件或路径重命名，一般调用格式为os.rename(src, dst)，即将src指向的文件或路径重命名为dst指定的名称。

注意，如果指定的目标路径在其他目录下，该函数还可实现文件或路径的“剪切并粘贴”功能。但无论直接原地重命名还是“剪切粘贴”，中间路径都必须要存在，否则就会抛出FileNotFoundError异常。如果目标路径已存在，Windows 下会抛出FileExistsError异常；Linux 下，如果目标路径为空且用户权限允许，则会静默覆盖原路径，否则抛出OSError异常，

和上两个函数一样，该函数也有对应的递归版本os.renames()，能够创建缺失的中间路径。

注意，这两种情况下，如果函数执行成功，都会调用os.removedir()函数来递归删除源路径的最下级目录。

**2.8 os.getcwd()**

“getcwd”实际上是“get the current working directory”的简写，顾名思义，也就是说这个函数的作用是“获取当前工作路径”。在程序运行的过程中，无论物理上程序在实际存储空间的什么地方，“当前工作路径”即可认为是程序所在路径；与之相关的“相对路径”、“同目录下模块导入”等相关的操作均以“当前工作路径”为准。

在交互式环境中，返回的就是交互终端打开的位置；而在 Python 文件中，返回的则是文件所在的位置。

在 Windows 下会有如下输出：

**>>>** os**.**getcwd()

'd:\\justdopython\\just\\do\\python'

Linux 下的输出则是：

**>>>** os**.**getcwd()

'/home/justdopython/just/do/python'

**2.9 os.chdir()**

“chdir”其实是“change the directory”的简写，因此os.chdir()的用处实际上是切换当前工作路径为指定路径。其中“指定路径”需要作为参数传入函数os.chdir()，该参数既可以是文本或字节型字符串，也可以是一个文件描述符，还可以是一个广义的类路径（path-like）对象。若指定路径不存在，则会抛出FileNotFoundError异常。

在 Windows 下，调用该函数的效果为：

**>>>** os**.**chdir("d:/justdopython/just/do")

**>>>** os**.**getcwd()

'd:\\justdopython\\just\\do'

在 Linux 下的效果则是：

**>>>** os**.**chdir("/home/justdopython/just/do") *# 也可将参数指定为".."，即可切换到父目录*

**>>>** os**.**getcwd()

'/home/justdopython/just/do'

有了这个函数，跨目录读写文件和调用模块就会变得非常方便了，很多时候也就不必再反复将同一个文件在各个目录之间复制粘贴运行，脚本完全可以坐镇中军，在一个目录下完成对其他目录文件的操作，正所谓“运筹帷幄之中，决胜于千里之外”也。

举例来说，可以通过将“当前工作目录”切换到父目录，从而直接访问父目录的文件内容：

**>>>** os**.**chdir("..")

**>>>** os**.**getcwd()

'D:\\justdopython\\just'

**>>>** **with** open("hello\_justdopython.txt", encoding**=**"utf-8") **as** f:

**...** f**.**read()

**...**

'欢迎访问 justdopython.com，一起学习 Python 技术~'

**>>>** os**.**listdir()

['hello\_justdopython.txt']

**3. os.path 模块**

其实这个模块是os模块根据系统类型从另一个模块导入的，并非直接由os模块实现，比如os.name值为nt，则在os模块中执行import ntpath as path；如果os.name值为posix，则导入posixpath。

使用该模块要注意一个很重要的特性：os.path中的函数基本上是纯粹的字符串操作。换句话说，传入该模块函数的参数甚至不需要是一个有效路径，该模块也不会试图访问这个路径，而仅仅是按照“路径”的通用格式对字符串进行处理。

更进一步地说，os.path模块的功能我们都可以自己使用字符串操作手动实现，该模块的作用是让我们在实现相同功能的时候不必考虑具体的系统，尤其是不需要过多关注文件系统分隔符的问题。

**3.1 os.path.join()**

这是一个十分实用的函数，可以将多个传入路径组合为一个路径。实际上是将传入的几个字符串用系统的分隔符连接起来，组合成一个新的字符串，所以一般的用法是将第一个参数作为父目录，之后每一个参数即使下一级目录，从而组合成一个新的符合逻辑的路径。

但如果传入路径中存在一个“绝对路径”格式的字符串，且这个字符串不是函数的第一个参数，那么其他在这个参数之前的所有参数都会被丢弃，余下的参数再进行组合。更准确地说，只有最后一个“绝对路径”及其之后的参数才会体现在返回结果中。

**>>>** os**.**path**.**join("just", "do", "python", "dot", "com")

'just\\do\\python\\dot\\com'

**>>>**

**>>>** os**.**path**.**join("just", "do", "d:/", "python", "dot", "com")

'd:/python\\dot\\com'

**>>>**

**>>>** os**.**path**.**join("just", "do", "d:/", "python", "dot", "g:/", "com")

'g:/com'

**3.2 os.path.abspath()**

将传入路径规范化，返回一个相应的绝对路径格式的字符串。

也就是说当传入路径符合“绝对路径”的格式时，该函数仅仅将路径分隔符替换为适应当前系统的字符，不做其他任何操作，并将结果返回。所谓“绝对路径的格式”，其实指的就是一个字母加冒号，之后跟分隔符和字符串序列的格式：

**>>>** os**.**path**.**abspath("a:/just/do/python")

'a:\\just\\do\\python'

**>>>** *# 我的系统中并没有 a 盘*

当指定的路径不符合上述格式时，该函数会自动获取当前工作路径，并使用os.path.join()函数将其与传入的参数组合成为一个新的路径字符串。示例如下：

**>>>** os**.**path**.**abspath("ityouknow")

'D:\\justdopython\\ityouknow'

**3.3 os.path.basename()**

该函数返回传入路径的“基名”，即传入路径的最下级目录。

**>>>** os**.**path**.**basename("/ityouknow/justdopython/IAmBasename")

'IAmBasename'

**>>>** *# 我的系统中同样没有这么一个路径。可见 os.path.basename() 页是单纯进行字符串处理*

整这个函数要注意的一点是，返回的“基名”实际上是传入路径最后一个分隔符之后的子字符串，也就是说，如果最下级目录之后还有一个分隔符，得到的就会是一个空字符串：

**>>>** os**.**path**.**basename("/ityouknow/justdopython/IAmBasename/")

''

**3.4 os.path.dirname()**

与上一个函数正好相反，返回的是最后一个分隔符前的整个字符串：

**>>>** os**.**path**.**dirname("/ityouknow/justdopython/IAmBasename")

'/ityouknow/justdopython'

**>>>**

**>>>** os**.**path**.**dirname("/ityouknow/justdopython/IAmBasename/")

'/ityouknow/justdopython/IAmBasename'

**3.5 os.path.split()**

哈哈实际上前两个函数都是弟弟，这个函数才是老大。

函数os.path.split()的功能就是将传入路径以最后一个分隔符为界，分成两个字符串，并打包成元组的形式返回；前两个函数os.path.dirname()和os.path.basename()的返回值分别是函数os.path.split()返回值的第一个、第二个元素。就连二者的具体实现都十分真实：

**def** **basename**(p):

"""Returns the final component of a pathname"""

**return** split(p)[1]

**def** **dirname**(p):

"""Returns the directory component of a pathname"""

**return** split(p)[0]

通过os.path.join()函数又可以把它们组合起来得到原先的路径。

**3.6 os.path.exists()**

这个函数用于判断路径所指向的位置是否存在。若存在则返回True，不存在则返回False：

**>>>** os**.**path**.**exists(".")

True

**>>>** os**.**path**.**exists("./just")

True

**>>>** os**.**path**.**exists("./Inexistence") *# 不存在的路径*

False

一般的用法是在需要持久化保存某些数据的场景，为避免重复创建某个文件，需要在写入前用该函数检测一下相应文件是否存在，若不存在则新建，若存在则在文件内容之后增加新的内容。

**3.7 os.path.isabs()**

该函数判断传入路径是否是绝对路径，若是则返回True，否则返回False。当然，仅仅是检测格式，同样不对其有效性进行任何核验：

**>>>** os**.**path**.**isabs("a:/justdopython")

True

**3.8 os.path.isfile() 和 os.path.isdir()**

这两个函数分别判断传入路径是否是文件或路径，注意，此处会核验路径的有效性，如果是无效路径将会持续返回False。

**>>>** *# 无效路径*

**>>>** os**.**path**.**isfile("a:/justdopython")

False

**>>>**

**>>>** *# 有效路径*

**>>>** os**.**path**.**isfile("./just/plain\_txt")

True

**>>>**

**>>>** *# 无效路径*

**>>>** os**.**path**.**isdir("a:/justdopython/")

False

**>>>** *# 有效路径*

**>>>** os**.**path**.**isdir("./just/")

True

**4. 总结**

本文详细介绍了与操作系统交互的os模块中一些常用的属性和函数，基本可以覆盖初阶的学习和使用。有了这些功能，我们已经可以写出一些比较实用的脚本了。

除了文中介绍的函数外，os模块还有很多更加复杂的功能，但大多是我们暂时用不到的，以后用到会进一步讲解。

示例代码：[python-100-days](https://github.com/JustDoPython/python-100-day)

**5. 参考资料**

[Python3 文档-标准库](https://docs.python.org/3/tutorial/stdlib.html)

[Python3 文档-库-os](https://docs.python.org/3/library/os.html)

## 常见函数列表

**os.sep:取代操作系统特定的路径分隔符**

**os.name:指示你正在使用的工作平台。比如对于Windows，它是'nt'，而对于Linux/Unix用户，它是'posix'。**

**os.getcwd():得到当前工作目录，即当前python脚本工作的目录路径。**

**os.getenv()和os.putenv:分别用来读取和设置环境变量**

**os.listdir():返回指定目录下的所有文件和目录名**

**os.remove(file):删除一个文件**

**os.stat（file）:获得文件属性**

**os.chmod(file):修改文件权限和时间戳**

**os.mkdir(name):创建目录**

**os.rmdir(name):删除目录**

**os.removedirs（r“c：\python”）:删除多个目录**

**os.system():运行shell命令**

**os.exit():终止当前进程**

**os.linesep:给出当前平台的行终止符。例如，Windows使用'\r\n'，Linux使用'\n'而Mac使用'\r'**

**os.path.split():返回一个路径的目录名和文件名**

**os.path.isfile()和os.path.isdir()分别检验给出的路径是一个目录还是文件**

**os.path.existe():检验给出的路径是否真的存在**

**os.listdir(dirname):列出dirname下的目录和文件**

**os.getcwd():获得当前工作目录**

**os.curdir:返回当前目录（'.'）**

**os.chdir(dirname):改变工作目录到dirname**

**os.path.isdir(name):判断name是不是目录，不是目录就返回false**

**os.path.isfile(name):判断name这个文件是否存在，不存在返回false**

**os.path.exists(name):判断是否存在文件或目录name**

**os.path.getsize(name):或得文件大小，如果name是目录返回0L**

**os.path.abspath(name):获得绝对路径**

**os.path.isabs():判断是否为绝对路径**

**os.path.normpath(path):规范path字符串形式**

**os.path.split(name):分割文件名与目录（事实上，如果你完全使用目录，它也会将最后一个目录作为文件名而分离，同时它不会判断文件或目录是否存在）**

**os.path.splitext():分离文件名和扩展名**

**os.path.join(path,name):连接目录与文件名或目录**

**os.path.basename(path):返回文件名**

**os.path.dirname(path):返回文件路径**

**文件操作**

**os.mknod("text.txt")：创建空文件  
fp = open("text.txt",w):直接打开一个文件，如果文件不存在就创建文件**

**关于open的模式**

**w 写方式  
a 追加模式打开（从EOF开始，必要时创建新文件）  
r+ 以读写模式打开  
w+ 以读写模式打开  
a+ 以读写模式打开  
rb 以二进制读模式打开  
wb 以二进制写模式打开 (参见 w )  
ab 以二进制追加模式打开 (参见 a )  
rb+ 以二进制读写模式打开 (参见 r+ )  
wb+ 以二进制读写模式打开 (参见 w+ )  
ab+ 以二进制读写模式打开 (参见 a+ )**

**关于文件的函数**

**fp.read([size])**

**size为读取的长度，以byte为单位**

**fp.readline([size])**

**读一行，如果定义了size，有可能返回的只是一行的一部分**

**fp.readlines([size])**

**把文件每一行作为一个list的一个成员，并返回这个list。其实它的内部是通过循环调用readline()来实现的。如果提供size参数，size是表示读取内容的总长，也就是说可能只读到文件的一部分。**

**fp.write(str)**

**把str写到文件中，write()并不会在str后加上一个换行符**

**fp.writelines(seq)**

**把seq的内容全部写到文件中(多行一次性写入)。这个函数也只是忠实地写入，不会在每行后面加上任何东西。**

**fp.close()**

**关闭文件。python会在一个文件不用后自动关闭文件，不过这一功能没有保证，最好还是养成自己关闭的习惯。 如果一个文件在关闭后还对其进行操作会产生ValueError**

**fp.flush()**

**把缓冲区的内容写入硬盘**

**fp.fileno()**

**返回一个长整型的”文件标签“**

**fp.isatty()**

**文件是否是一个终端设备文件（unix系统中的）**

**fp.tell()**

**返回文件操作标记的当前位置，以文件的开头为原点**

**fp.next()**

**返回下一行，并将文件操作标记位移到下一行。把一个file用于for … in file这样的语句时，就是调用next()函数来实现遍历的。**

**fp.seek(offset[,whence])**

**将文件打操作标记移到offset的位置。这个offset一般是相对于文件的开头来计算的，一般为正数。但如果提供了whence参数就不一定了，whence可以为0表示从头开始计算，1表示以当前位置为原点计算。2表示以文件末尾为原点进行计算。需要注意，如果文件以a或a+的模式打开，每次进行写操作时，文件操作标记会自动返回到文件末尾。**

**fp.truncate([size])**

**把文件裁成规定的大小，默认的是裁到当前文件操作标记的位置。如果size比文件的大小还要大，依据系统的不同可能是不改变文件，也可能是用0把文件补到相应的大小，也可能是以一些随机的内容加上去。**

**目录操作**

**os.mkdir("file")**

**创建目录**

**复制文件:**

**shutil.copyfile("oldfile","newfile")**

**oldfile和newfile都只能是文件**

**shutil.copy("oldfile","newfile")**

**oldfile只能是文件夹，newfile可以是文件，也可以是目标目录**

**shutil.copytree("olddir","newdir")**

**复制文件夹.olddir和newdir都只能是目录，且newdir必须不存在**

**os.rename("oldname","newname")**

**重命名文件（目录）.文件或目录都是使用这条命令**

**shutil.move("oldpos","newpos")**

**移动文件（目录）**

**os.rmdir("dir")**

**只能删除空目录**

**shutil.rmtree("dir")**

**空目录、有内容的目录都可以删**

**os.chdir("path")**

**转换目录，换路径**

**代码演示**

**#!/usr/bin/env python**

**import os**

**print(os.getcwd() ) #获取当前路径**

**print(os.listdir('/mnt') ) #列出/mnt下面的所有目录和文件**

**print(os.mkdir('lala')) #创建目录lala和haha**

**print(os.mkdir('haha'))**

**print(os.rmdir('haha')) #删除haha**

**print(os.rename('example.txt','back.txt')) #修改文件名**

**运行：**

**root@ruanyang-HP-ProDesk-680-G2-MT:/mnt/python# python os.py**

**/mnt/python**

**['proc', 'perl\_bash', 'docker', 'warn', 'blog', 'python', 'hostname', 'test', 'tags', 'perl\_DB', 'shell', 'perl\_test', '.vimrc~', 'club.yml', 'test.c', '.vimrc']**

**None**

**None**

**None**

**None**

**root@ruanyang-HP-ProDesk-680-G2-MT:/mnt/python# ls**

**back.txt exit.py lala modules.py os\_file.py os.py sys.py**

**参考文章：**[**http://blog.csdn.net/b\_h\_l/article/details/17289397**](http://blog.csdn.net/b_h_l/article/details/17289397)[**http://blog.csdn.net/devil\_2009/article/details/7941241**](http://blog.csdn.net/devil_2009/article/details/7941241)