OS 系统操作

在日常使用计算机时，经常需要列出一个文件夹或者目录的内容，创建和删除文件，以及做其他一些比较无聊但是不得不做的“家务活”。在 Python 程序中可以做到同样的事，甚至能做更多的事。这些功能是否能减少你的工作量呢？我们拭目以待。

Python 在模块 `os`（操作系统，operating system）中提供了许多系统函数，本章的所有程序都需要导入这个模块。

Python 的 os 模块封装了常见的文件和目录操作，本文只列出部分常用的方法，更多的方法可以查看官方文档。

os 目录操作

方法	说明
os.mkdir	创建目录
os.rmdir	删除目录
os.rename	重命名
os.remove	删除文件
os.getcwd	获取当前工作路径
os.chdir	修改当前工作目录

os.mkdir

创建文件目录。

```
In [4]: os.mkdir(os.path.join(os.path.abspath('.'), 'testdir'))
```

os.rmdir

删除文件目录。

```
In [6]: os.rmdir(os.path.join(os.path.abspath('.'), 'testdir'))
```

os.rename

重命名文件。如果文件不存在则报错

```
In [10]: os.rename('test.txt', 'test.py')
```

os.remove

删除文件，文件不存在时报错

```
In [13]: os.remove('test.py')
```

os.getcwd

显示当前程序的工作目录

```
In [2]: os.getcwd()
Out[2]: 'C:\\Users\\Administrator'
```

os.chdir

改变程序的工作目录

```
In [2]: os.getcwd()
Out[2]: 'C:\\Users\\Administrator'
In [3]: os.chdir('C:\\')
In [4]: os.getcwd()
Out[4]: 'C:\\'
```

案例一:

显示当前目录中所有的文件

os.path 路径操作

`os.path` 模块是跨平台的，即使不打算在平台之间移植自己的程序也应该用 `os.path`，好处多多。

操作路径

后文的例子以下面的目录结构为参考，工作目录为 `'C:\\Users\\Administrator'`。

方法	说明
os.path.join	连接目录与文件名
os.path.split	分割文件名与目录
os.path.abspath	获取绝对路径
os.path.dirname	获取路径
os.path.basename	获取文件名或文件夹名
os.path.splitext	分离文件名与扩展名
os.path.isfile	判断给出的路径是否是一个文件
os.path.isdir	判断给出的路径是否是一个目录
os.path.exists	检查文件是否存在

```
In [4]: os.getcwd()
Out[4]: 'C:\\Users\\Administrator'

In [7]: os.path.abspath('hello.py')
Out[7]: 'C:\\Users\\Administrator\\hello.py'

In [8]: os.path.abspath('.')
Out[8]: 'C:\\Users\\Administrator'
```

- os.path.split：分离目录与文件名

- `split()` 函数将路径分成两个独立的部分，并返回一个`tuple`结果。第二个元素是路径的最后一个元素，第一个元素是它之前的所有元素。
- `join()` 函数能将两个或者多个独立部分,拼接为一个路径.

```
In [9]: os.path.split(os.path.abspath('.'))
Out[9]: ('C:\\Users', 'Administrator')

In [11]: os.path.join('C:\\Users', 'Administrator')
Out[11]: 'C:\\Users\\Administrator'
```

- os.path.splitext：分离文件名与扩展名

  `splitext()`类似于`split()`，但在扩展分隔符上划分路径，而不是目录分隔符。

```
In [17]: os.path.splitext('filename.txt')
Out[17]: ('filename', '.txt')
```

案例二:

在测试文件夹下, 给所有的 `txt` 文件前加上前缀 `[前缀]` 两个字

检查路径

- os.path.dirname：获取文件或文件夹的路径

```
In [7]: os.path.dirname('C:\\Users\\Administrator\\hello.py')
Out[7]:'C:\\Users\\Administrator'

In [8]: os.path.dirname('C:\\Users\\Administrator\\python')
Out[8]: 'C:\\Users\\Administrator'

In [9]: os.path.dirname('C:\\Users\\Administrator')
Out[9]: 'C:\\Users'
```

- os.path.basename：获取文件名或文件夹名

```
In [10]: os.path.basename('C:\\Users\\Administrator\\hello.py')
Out[10]: 'hello.py'

In [11]: os.path.basename('/Users/ethan/coding/python')
Out[11]: 'python'
```

- os.path.isfile/os.path.isdir

```
In [17]: os.path.isfile('C:\\Users\\Administrator\\hello.py')
Out[17]: True

In [18]: os.path.isfile('C:\\Users\\Administrator\\hello1.py')
Out[18]: False

In [20]: os.path.isdir('C:\\Users\\Administrator\\hello1.py')
Out[20]: False

In [21]: os.path.isdir('C:\\Users\\Administrator')
Out[21]: True
```

案例二:

显示当前目录中所有的目录

```
In [12]: [x for x in os.listdir('.') if os.path.isdir(x)]
```

sys 模块

> 系统自身的一些操作

sys模块包括了一组非常实用的服务，内含很多函数方法和变量，用来处理Python运行时配置以及资源，从而可以与前当程序之外的系统环境交互，如：Python解释器。与os模块相比，其更多的在于针对系统环境的交互，而os则操作文件目录。

导入sys模块

首先，打开终端模拟器进入Python解释器或者打开IDE编辑器创建一个新的.py后缀名的Python程序文件。
下面，以解释器中的操作举例：

```
>>> import sys # 导入sys模块
>>> dir(sys)     #dir()方法查看模块中可用的方法
```

sys 常见函数

> 了解

- `sys.argv`: 命令行参数，包括脚本名称，实现从程序外部向程序传递参数
- `sys.exit([arg])`: 程序中间的退出，arg=0为正常退出
- `sys.path`: 查找模块所在目录的目录名列表
- `sys.modules`：映射模块名字到载入模块的字典
- `sys.platform`: 获取当前系统平台
- `sys.stdin`：标准输入流-一个类文件（file-like）对象
- `sys.stdout`：标准输出流-一个类文件对象
- `sys.stderr`：标准错误流-一个类文件对象
- `sys.getdefaultencoding()`: 获取系统当前编码，一般默认为ascii。
- `sys.setdefaultencoding()`: 设置系统默认编码，需要reload(sys)才能查询看到
- `sys.getfilesystemencoding()`: 获取文件系统使用编码方式，Windows下返回'mbcs'，mac下返回'utf-8'

sys.argv

`sys.argv`是命令行参数，包括脚本名称，它的功能可以实现从程序外部向程序传递参数。那么这个过程是如何实现的呢？假设有一个名为`test.py`的文件，则可以通过`import sys`和`print sys.argv[number]`两句来实现。number=0为这个脚本的名字，1,2,...则为命令行下传递的参数。

如test.py脚本内容如下：

```
import sys

print(sys.argv[0])
print(sys.argv[1])
print(sys.argv[2])
print(sys.argv[3])
```

那么要实现参数传递即可采用`>>>python test.py arg1 arg2 arg3`实现。下文会结合`sys.path`给出一个示例。上述命令的输出结果为：

```
test.py
arg1
arg2
arg3
```

`sys.exit([arg])`

执行至主程序的末尾时，解释器会自动退出。但如果需要中途退出程序，可以调用`sys.exit()`函数来实现。它带有一个可选的整数参数返回给调用它的程序。这意味着你可以在主程序中捕获对`sys.exit()`的调用。（0为正常退出，其他为不正常，可抛异常事件供捕获）

`sys.exit()`函数运行示例如下：

```
import sys

sys.exit(1) # 直接退出程序

print('hello')
```

sys.path

`sys.path`的功能是获取指定模块搜索路径的字符串集合，可以将写好的模块放在得到的某个路径下，就可以在程序中import时正确找到。

示例：

```
In [1]: import sys

In [2]: sys.path
Out[2]:
['',
 'C:\\paisen\\Anaconda3\\Scripts',
 'C:\\paisen\\Anaconda3\\python36.zip',
 'C:\\paisen\\Anaconda3\\DLLs',
 'C:\\paisen\\Anaconda3\\lib',
 'C:\\paisen\\Anaconda3',
 'C:\\Users\\Administrator\\AppData\\Roaming\\Python\\Python36\\site-packages',
 'C:\\paisen\\Anaconda3\\lib\\site-packages',
 'C:\\paisen\\Anaconda3\\lib\\site-packages\\setuptools-39.1.0-py3.6.egg',
 'C:\\paisen\\Anaconda3\\lib\\site-packages\\pip-10.0.1-py3.6.egg',
 'C:\\paisen\\Anaconda3\\lib\\site-packages\\win32',
 'C:\\paisen\\Anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\paisen\\Anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\paisen\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
```

```
 'C:\\Users\\Administrator\\.ipython']
```

可以采用`sys.path.append("自定义模块路径")`增加模块搜索路径（也可以用`sys.path.insert`进行列表插入）。以上列出的模块目录，在python执行`import module_name`时，会依次从上述列出的路径来搜索对应的模块。如有需要手动添加模块，可放入对应的路径中，如无必要建议不要手动添加路径。`.`代表当前文件目录。

上问提到的`sys.argv`参数传递示例如下：

输入：

```
import sys


print('当前 Python PATH 路径有以下内容:')
for path in sys.path:
    print(path)
```

输出：

```
当前 Python PATH 路径有以下内容:
F:\进阶部分
F:\进阶部分
C:\paisen\Anaconda3\python36.zip
C:\paisen\Anaconda3\DLLs
C:\paisen\Anaconda3\lib
C:\paisen\Anaconda3
C:\Users\Administrator\AppData\Roaming\Python\Python36\site-packages
C:\paisen\Anaconda3\lib\site-packages
C:\paisen\Anaconda3\lib\site-packages\setuptools-39.1.0-py3.6.egg
C:\paisen\Anaconda3\lib\site-packages\pip-10.0.1-py3.6.egg
C:\paisen\Anaconda3\lib\site-packages\win32
C:\paisen\Anaconda3\lib\site-packages\win32\lib
C:\paisen\Anaconda3\lib\site-packages\Pythonwin
C:\Program Files\JetBrains\PyCharm
2018.2.5\helpers\pycharm_matplotlib_backend
```

sys.modules

> 下面部分内容均为演示

`sys.modules`是一个全局字典，该字典是python启动后就加载在内存中。每当导入新的模块时，`sys.modules`将自动记录该模块。当第二次再导入该模块时，python会直接到字典中查找，从而加快了程序运行的速度。它拥有字典所拥有的一切方法。

示例：

```
import sys

print(sys.modules.keys())  # 列出所有的模块名
print(sys.modules['os'])  # 列出指定模块如os模块所在路径
print(sys.modules.values())  # 列出所有模块所在路径
```

sys.platform

`sys.platform`语句用于查看当前平台，如win32、linux2等。

```
In [28]: sys.platform
Out[28]: 'win32'
```