本文实例讲述了python中os操作文件及文件路径的方法。分享给大家供大家参考。具体分析如下：

python获取文件上一级目录：取文件所在目录的上一级目录

复制代码代码如下:

os.path.abspath(os.path.join(os.path.dirname('settings.py'),os.path.pardir))

os.path.pardir是父目录，os.path.abspath是绝对路径  
举例具体看一下输出：

复制代码代码如下:

print os.path.dirname(os.path.abspath("\_\_file\_\_"))  
print os.path.pardir  
print os.path.join(os.path.dirname("\_\_file\_\_"),os.path.pardir)  
print os.path.abspath(os.path.join(os.path.dirname("\_\_file\_\_"),os.path.pardir))

输出结果：

  G:\work\python  
  ..                      #这是两个点“..”，也就是上级目录的表示方法  
  ..  
  G:\work  
   
获取文件当前路径：

复制代码代码如下:

os.path.dirname(os.path.abspath('\_\_file\_\_'))

   
python中对文件、文件夹（文件操作函数）的操作需要涉及到os模块和shutil模块。  
得到当前工作目录，即当前Python脚本工作的目录路径: os.getcwd()  
返回指定目录下的所有文件和目录名:os.listdir()  
函数用来删除一个文件:os.remove()  
删除多个目录：os.removedirs（r“c：\python”）  
检验给出的路径是否是一个文件：os.path.isfile()  
检验给出的路径是否是一个目录：os.path.isdir()  
判断是否是绝对路径：os.path.isabs()  
检验给出的路径是否真地存:os.path.exists()  
返回一个路径的目录名和文件名:os.path.split() eg os.path.split('/home/swaroop/byte/code/poem.txt') 结果：('/home/swaroop/byte/code', 'poem.txt')  
分离扩展名：os.path.splitext()  
获取路径名：os.path.dirname()  
获取文件名：os.path.basename()  
运行shell命令: os.system()  
读取和设置环境变量:os.getenv() 与os.putenv()  
给出当前平台使用的行终止符:os.linesep Windows使用'\r\n'，Linux使用'\n'而Mac使用'\r'  
指示你正在使用的平台：os.name 对于Windows，它是'nt'，而对于Linux/Unix用户，它是'posix'  
重命名：os.rename（old， new）  
创建多级目录：os.makedirs（r“c：\python\test”）  
创建单个目录：os.mkdir（“test”）  
获取文件属性：os.stat（file）  
修改文件权限与时间戳：os.chmod（file）  
终止当前进程：os.exit（）  
获取文件大小：os.path.getsize（filename）  
   
   
**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  
复制文件夹：  
shutil.copytree("olddir","newdir") olddir和newdir都只能是目录，且newdir必须不存在  
重命名文件（目录）  
os.rename("oldname","newname") 文件或目录都是使用这条命令  
移动文件（目录）  
shutil.move("oldpos","newpos")   
删除文件  
os.remove("file")  
删除目录  
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  
转换目录  
os.chdir("path") 换路径

**pyhton文件操作函数：**

os.mknod("test.txt") 创建空文件  
fp = open("test.txt",w) 直接打开一个文件，如果文件不存在则创建文件  
关于open 模式：  
w 以写方式打开，  
a 以追加模式打开 (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 (参见 w )  
a+ 以读写模式打开 (参见 a )  
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+ )  
   
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。

os目录操作

* os.getcwd()：得到当前Python脚本工作的目录路径
* os.chdir(“path”)：改变Python当前目录
* os.mkdir(“dir”)：创建目录
* os.makedirs（r“path”）：创建多级目录
* os.rmdir(“dir”)：删除目录，只能是空目录
* os.removedirs(r”path”)：由子到父删除多级目录，若最下级非空，抛异常。如：file1有1和2两个子文件夹，os.removedirs（r”file1\1”）删除1，保留file1和2。若file1下只有1这个文件夹，则os.removedirs（r”file1\1”）删除1和file1两层。
* os.remove(“file”)：删除文件，不能删除目录
* os.rename(“oldname”,”newname”) ：重命名文件或目录
* os.listdir(“path”)：返回一个列表，包含制定目录下所有文件和目录名

os.path模块

* os.path.isfile(“path”)：检验给出路径是否是文件
* os.path.isdir(”path“)：检验给出路径是否是目录
* os.path.isabs(”path“)：判断是否是绝对路径
* os.path.exists(”path“)：检验给出的路径是否真的存在
* os.path.split()：拆分路径和文件名，返回两个值的列表（一个为路径，一个为文件）
* os.path.splitext()：分离扩展名
* os.path.dirname()：获取路径名
* os.path.basename（）：获取文件名
* os.path.getsize（filename）：获取文件大小

os文件操作

* os.mknod(“test.txt”)：创建空文件
* fp = open(“test.txt”,w) ：直接打开一个文件，如果文件不存在则创建文件

打开模式：   
\* w 以写方式打开，   
\* a 以追加模式打开 (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 (参见 w )   
\* a+ 以读写模式打开 (参见 a )   
\* 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+ )

* 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](http://lib.csdn.net/base/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。

   
**相关例子**

1 将文件夹下所有图片名称加上'\_fc'  
python代码:

复制代码代码如下:

# -\*- coding:utf-8 -\*-  
import re  
import os  
import time  
#str.split(string)分割字符串  
#'连接符'.join(list) 将列表组成字符串  
def change\_name(path):  
global i  
if not os.path.isdir(path) and not os.path.isfile(path):  
return False  
if os.path.isfile(path):  
file\_path = os.path.split(path) #分割出目录与文件  
lists = file\_path[1].split('.') #分割出文件与文件扩展名  
file\_ext = lists[-1] #取出后缀名(列表切片操作)  
img\_ext = ['bmp','jpeg','gif','psd','png','jpg']  
if file\_ext in img\_ext:  
os.rename(path,file\_path[0]+'/'+lists[0]+'\_fc.'+file\_ext)  
i+=1 #注意这里的i是一个陷阱  
#或者  
#img\_ext = 'bmp|jpeg|gif|psd|png|jpg'  
#if file\_ext in img\_ext:  
# print('ok---'+file\_ext)  
elif os.path.isdir(path):  
for x in os.listdir(path):  
change\_name(os.path.join(path,x)) #os.path.join()在路径处理上很有用

img\_dir = 'D:\\xx\\xx\\images'  
img\_dir = img\_dir.replace('\\','/')  
start = time.time()  
i = 0  
change\_name(img\_dir)  
c = time.time() - start  
print('程序运行耗时:%0.2f'%(c))  
print('总共处理了 %s 张图片'%(i))

输出结果：  
程序运行耗时:0.11  
总共处理了 109 张图片

#!/usr/bin/env python

#-\*- encoding:UTF-8 -\*-

import os,time,stat

fileStats = os.stat ( 'test.txt' ) #获取文件/目录的状态

fileInfo = {

'Size':fileStats [ stat.ST\_SIZE ], #获取文件大小

'LastModified':time.ctime( fileStats [ stat.ST\_MTIME ] ), #获取文件最后修改时间

'LastAccessed':time.ctime( fileStats [ stat.ST\_ATIME ] ), #获取文件最后访问时间

'CreationTime':time.ctime( fileStats [ stat.ST\_CTIME ] ), #获取文件创建时间

'Mode':fileStats [ stat.ST\_MODE ] #获取文件的模式

}

#print fileInfo

for field in fileInfo: #显示对象内容

print '%s:%s' % (field,fileInfo[field])

for infoField,infoValue in fileInfo:

print '%s:%s' % (infoField,infoValue)

if stat.S\_ISDIR ( fileStats [ stat.ST\_MODE ] ): #判断是否路径

print 'Directory. '

else:

print 'Non-directory.'

if stat.S\_ISREG( fileStats [ stat.ST\_MODE ] ): #判断是否一般文件

print 'Regular file.'

elif stat.S\_ISLNK ( fileStats [ stat.ST\_MODE ] ): #判断是否链接文件

print 'Shortcut.'

elif stat.S\_ISSOCK ( fileStats [ stat.ST\_MODE ] ): #判断是否套接字文件

print 'Socket.'

elif stat.S\_ISFIFO ( fileStats [ stat.ST\_MODE ] ): #判断是否命名管道

print 'Named pipe.'

elif stat.S\_ISBLK ( fileStats [ stat.ST\_MODE ] ): #判断是否块设备

print 'Block special device.'

elif stat.S\_ISCHR ( fileStats [ stat.ST\_MODE ] ): #判断是否字符设置

print 'Character special device.'

stat模块描述了os.stat(filename)返回的文件属性列表中各值的意义.我们可方便地根据stat模块存取os.stat()中的值.

os.stat(path)执行一个stat()系统调用在给定的path上，返回一个类元组对象(stat\_result对象,包含10个元素),属性与stat结构成员相关:st\_mode(权限模式),st\_ino(inode number),st\_dev(device),st\_nlink(number of hard links),st\_uid(所有用户的user id),st\_gid(所有用户的group id),st\_size(文件大小，以位为单位)，st\_atime(最近访问的时间),st\_mtime(最近修改的时间),st\_ctime(创建的时间)

>>> import os

>>> print os.stat("/root/python/zip.py")

(33188, 2033080, 26626L, 1, 0, 0, 864, 1297653596, 1275528102, 1292892895)

>>> print os.stat("/root/python/zip.py").st\_mode #权限模式

33188

>>> print os.stat("/root/python/zip.py").st\_ino #inode number

2033080

>>> print os.stat("/root/python/zip.py").st\_dev #device

26626

>>> print os.stat("/root/python/zip.py").st\_nlink #number of hard links

1

>>> print os.stat("/root/python/zip.py").st\_uid #所有用户的user id

0

>>> print os.stat("/root/python/zip.py").st\_gid #所有用户的group id

0

>>> print os.stat("/root/python/zip.py").st\_size #文件的大小，以位为单位

864

>>> print os.stat("/root/python/zip.py").st\_atime #文件最后访问时间

1297653596

>>> print os.stat("/root/python/zip.py").st\_mtime #文件最后修改时间

1275528102

>>> print os.stat("/root/python/zip.py").st\_ctime #文件创建时间

1292892895