

小白必备文件操作练习

80 文件读操作的案例

文件读、写操作比较常见。

读取文件，要先判断文件是否存在。

- 若文件存在，再读取；
- 不存在，抛出文件不存在异常。

```
In [8]: import os

In [9]: def read_file(filename):
...:     if os.path.exists(filename) is False:
...:         raise FileNotFoundError('%s not exists'%(filename,))
...:     f = open(filename)
...:     content = f.read()
...:     f.close()
...:     return content
```

试着读取一个 D 盘文件：

```
In [13]: read_file('D:/source/python-zhuanlan/update_log.md')
-----
UnicodeDecodeError                                Traceback (most recent call las
t)
<ipython-input-13-92e8f4011b20> in <module>
----> 1 read_file('D:/source/python-zhuanlan/update_log.md')

<ipython-input-12-5591c3ba7b42> in read_file(filename)
      3         raise FileNotFoundError('%s not exists'%(filename,))
      4     f = open(filename)
----> 5     content = f.read()
      6     f.close()
      7     return content

UnicodeDecodeError: 'gbk' codec can't decode byte 0xaa in position 46: il
legal multibyte sequence
```

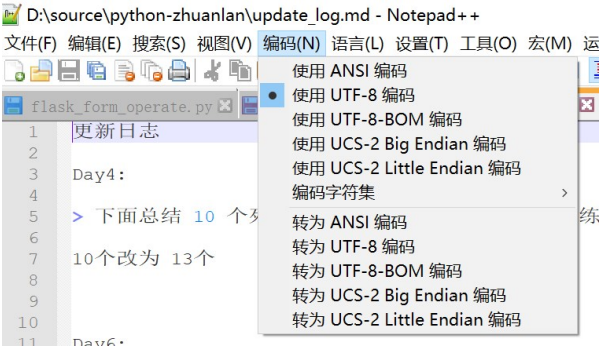
出错！在 `f.read` 这行，有错误提示看，是编码问题。`open` 函数打开文件，默认编码格式与平台系统有关，鉴于此，有必要在 `open` 时为参数 `encoding` 赋值，一般采用 `utf-8`：

```
In [9]: def read_file(filename):
...:     if os.path.exists(filename) is False:
...:         raise FileNotFoundError('%s not exists'%(filename,))
...:     f = open(filename,encoding='utf-8')
...:     content = f.read()
...:     f.close()
...:     return content
```

代码打开文件的编码确认为 `utf-8` 后

还需要确认，磁盘中这个文件编码格式也为 `utf-8`

推荐使用 `notepad++` 查看文件编码格式，查看方式如下图所示：



下面，读入成功：

```
In [22]: read_file('D:/source/python-zhuanlan/update_log.md')
Out[22]: '更新日志\n\nDay4: \n\n> 下面总结 10 个列表和元祖使用的经典例子, 通过练习
, 进一步加深对它们提供的方法的理解。'\n\n10个改为 13个'\n\nDay6: \n\n12个, 13个间
有多余的空行'
```

上面，还提到 `open` 后，务必要 `close`，这种写法有些繁琐，还容易出错。

借助 `with` 语法，同时实现 `open` 和 `close` 功能，这是更常用的方法。

```
In [9]: def read_file(filename):
...:     if os.path.exists(filename) is False:
...:         raise FileNotFoundError('%s not exists'%(filename,))
...:     with open(filename,encoding='utf-8') as f :
...:         content = f.read()
...:     return content
```

81 文件按行读的案例

`read` 函数一次读取整个文件，`readlines` 函数按行一次读取整个文件。读入文件小时，使用它们没有问题。

但是，如果读入文件大，`read` 或 `readlines` 一次读取整个文件，内存就会面临重大挑战。

使用 `readline` 一次读取文件一行，能解决大文件读取内存溢出问题。

文件 `a.txt` 内容如下：

80 文件读操作的案例
81 文件按行读的案例
82 文件写操作的案例
83 如何获取文件名？
84 如何获取后缀名？
85 获取指定后缀名...
86 如何批量修改后...
87 xls 批量转换成 x...
88 批量获取文件修...
89 批量压缩文件的...
90 32 位文件加密

```
Hey, Python

I just love      Python so much,
and want to get the whole  Python stack by this 60-days column
and believe Python !
```

复制

如下，读取文件 `a.txt`，`r+` 表示读写模式。代码块实现：

- 每次读入一行
- 选择正则 `split` 分词，注意观察 `a.txt`，单词间有的一个空格，有的多个。这些情况，实际工作中确实也会遇到。
- 使用 `defaultdict` 统计单词出现频次
- 按照频次从大到小降序

```
In [38]: from collections import defaultdict
...: import re
...:
...: rec = re.compile('\s+')
...: dd = defaultdict(int)
...: with open('a.txt','r+') as f:
...:     for line in f:
...:         clean_line = line.strip()
...:         if clean_line:
...:             words = rec.split(clean_line)
...:             for word in words:
...:                 dd[word] += 1
...: dd = sorted(dd.items(),key=Lambda x: x[1],reverse=True)
...: print('---print stat---')
...: print(dd)
...: print('---words stat done---')
```

复制

程序运行结果：

```
Hey, Python

I just love      Python so much,

and want to get the whole  Python stack by this 60-days column

and believe Python !
---print stat---
[('Python', 3), ('and', 2), ('I', 1), ('just', 1), ('love', 1), ('so', 1),
 ('much', 1), ('want', 1), ('to', 1), ('get', 1), ('the', 1), ('whole',
 1), ('stack', 1), ('by', 1), ('this', 1), ('60-days', 1), ('column', 1),
 ('believe', 1), ('!', 1)]
---words stat done---
```

复制

82 文件写操作的案例

文件写操作时，需要首先判断要写入的文件路径是否存在。

若不存在，通过 `mkdir` 创建出路径

否则，直接写入文件

```
import os

def write_to_file(file_path,file_name):
    if os.path.exists(file_path) is False:
        os.mkdir(file_path)

    whole_path_filename = os.path.join(file_path,file_name)
    to_write_content = '''
        Hey, Python
        I just love Python so much,
        and want to get the whole python stack by this 60
-days column
        and believe!
    '''

    with open(whole_path_filename, mode="w", encoding='utf-8') as f:
        f.write(to_write_content)
    print('-----write done-----')

    print('-----begin reading-----')
    with open(whole_path_filename,encoding='utf-8') as f:
        content = f.read()
        print(content)
        if to_write_content == content:
            print('content is equal by reading and writing')
        else:
            print('----Warning: NO Equal-----')
```

复制

以上这段代码思路：

- 路径不存在，创建路径
- 写文件
- 读取同一文件
- 验证写入到文件的内容是否正确

打印出的结果：

```
-----begin writing-----
-----write done-----
-----begin reading-----

        Hey, Python
        I just love Python so much,
        and want to get the whole python stack by this 60
-days column
        and believe!

content is equal by reading and writing
```

复制

83 如何获取文件名？

有时拿到一个文件名时，名字带有路径。

这时，使用 `os.path`，`split` 方法实现路径和文件的分离。

```
In [11]: import os
...: file_ext = os.path.split('./data/py/test.py')
...: ipath,ifile = file_ext

In [12]: ipath
Out[12]: './data/py'

In [13]: ifile
Out[13]: 'test.py'
```

84 如何获取后缀名？

如何优雅地获取文件后缀名，`os.path` 模块，`splitext` 能够优雅地提取文件后缀。

```
In [1]: import os

In [2]: file_extension = os.path.splitext('./data/py/test.py')

In [3]: file_extension[0]
Out[3]: './data/py/test'

In [4]: file_extension[1]
Out[4]: '.py'
```

85 获取指定后缀名的文件

```
import os

def find_file(work_dir,extension='jpg'):
    lst = []
    for filename in os.listdir(work_dir):
        print(filename)
        splits = os.path.splitext(filename)
        ext = splits[1] # 拿到扩展名
        if ext == '.'+extension:
            lst.append(filename)
    return lst

r = find_file('.', 'md')
print(r) # 返回所有目录下的md文件
```

86 如何批量修改后缀名？

本案例使用 Python `os` 模块和 `argparse` 模块。

将工作目录 `work_dir` 下所有后缀名为 `old_ext` 的文件，修改为 `new_ext`

通过此案例，同时掌握 `argparse` 模块。

首先，导入模块

```
import argparse
import os
```

定义脚本参数

```
def get_parser():
    parser = argparse.ArgumentParser(description='工作目录中文件后缀名修改')
    parser.add_argument('work_dir', metavar='WORK_DIR', type=str, nargs=1
                        ,
                        help='修改后后缀的文件目录')
    parser.add_argument('old_ext', metavar='OLD_EXT',
                        type=str, nargs=1, help='原来的后缀')
    parser.add_argument('new_ext', metavar='NEW_EXT',
                        type=str, nargs=1, help='新的后缀')
    return parser
```

后缀名批量修改，实现思路：

- 遍历目录下的所有文件
- 拿到此文件的后缀名
- 如果后缀名命中为 `old_ext`，`rename` 重命名

```
def batch_rename(work_dir, old_ext, new_ext):
    """
    传递当前目录、原来后缀名、新的后缀名后，批量重命名后缀
    """
    for filename in os.listdir(work_dir):
        # 获取得到文件后缀
        split_file = os.path.splitext(filename)
        file_ext = split_file[1]
        if old_ext == file_ext: # 定位后缀名为old_ext 的文件
            newfile = split_file[0] + new_ext # 修改后文件的完整名称
            # 实现重命名操作
            os.rename(
                os.path.join(work_dir, filename),
                os.path.join(work_dir, newfile)
            )
    print("完成重命名")
    print(os.listdir(work_dir))
```

实现 Main

```
def main():
    # 命令行参数
    parser = get_parser()
    args = vars(parser.parse_args())
    # 从命令行参数中依次解析出参数
    work_dir = args['work_dir'][0]
    old_ext = args['old_ext'][0]

    if old_ext[0] != '.':
        old_ext = '.' + old_ext
    new_ext = args['new_ext'][0]
    if new_ext[0] != '.':
        new_ext = '.' + new_ext

    batch_rename(work_dir, old_ext, new_ext)
```

实现仅对 `xls` 文件后缀修改。

```
import os

def xls_to_xlsx(work_dir):
    old_ext, new_ext = '.xls', '.xlsx'
    for filename in os.listdir(work_dir):
        # 获取得到文件后缀
        split_file = os.path.splitext(filename)
        file_ext = split_file[1]

        # 定位后缀名为old_ext 的文件
        if old_ext == file_ext:
            # 修改后文件的完整名称
            newfile = split_file[0] + new_ext
            # 实现重命名操作
            os.rename(
                os.path.join(work_dir, filename),
                os.path.join(work_dir, newfile)
            )
    print("完成重命名")
    print(os.listdir(work_dir))
```

调用函数 `xls_to_xlsx` :

```
xls_to_xlsx('./data')

# 输出结果:
# ['cut_words.csv', 'email_list.xlsx', 'email_test.docx', 'email_test.jpg',
#  'email_test.xlsx', 'geo_data.png', 'geo_data.xlsx',
#  'iotest.txt', 'pyside2.md', 'PySimpleGUI-4.7.1-py3-none-any.whl', 'test.txt',
#  'test_excel.xlsx', 'ziptest', 'ziptest.zip']
```

88 批量获取文件修改时间

`os.walk` 生成文件树结构

`os.path.getmtime` 返回文件的最后一次修改时间

```
# 获取目录下文件的修改时间
import os
from datetime import datetime

print(f"当前时间:{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")

def get_modify_time(indir):
    for root, _, files in os.walk(indir): # 循环目录和子目录
        for file in files:
            whole_file_name = os.path.join(root, file)
            modify_time = os.path.getmtime(whole_file_name) # 1581164725.991523
            # 991523 , 这种时间格式太不人性化
            nice_show_time = datetime.fromtimestamp(modify_time) # 转化为人性化的时间显示格式:2020-02-08 20:25:25.991523
            print('文件 %s 最后一次修改时间:%s' % (file,nice_show_time))
```

调用函数 `get_modify_time` , 输出当前目录下, 所有文件最后一次修改时间:

```
get_modify_time('.')

# 输出结果:
# 当前时间:2020-02-21 22:58:09
# 文件 a.sql 最后一次修改时间:2020-02-08 20:25:25.991523
# 文件 barchart.py 最后一次修改时间:2020-01-31 09:30:05.954167
# 文件 barstack.py 最后一次修改时间:2020-02-04 11:30:33.678985
# 文件 bar_numpy.py 最后一次修改时间:2020-01-29 16:43:20.228680
# 文件 class_method.py 最后一次修改时间:2020-02-19 19:23:50.218961
# 文件 decorator.py 最后一次修改时间:2020-01-24 15:33:44.498871
# 文件 flask_form_operate.py 最后一次修改时间:2020-02-13 13:36:16.969256
# 文件 flask_hello.py 最后一次修改时间:2020-02-03 21:26:26.416181
```

输出部分结果展示:

```
当前时间:2020-02-21 22:58:09
文件 a.sql 最后一次修改时间:2020-02-08 20:25:25.991523
文件 barchart.py 最后一次修改时间:2020-01-31 09:30:05.954167
文件 barstack.py 最后一次修改时间:2020-02-04 11:30:33.678985
文件 bar_numpy.py 最后一次修改时间:2020-01-29 16:43:20.228680
文件 class_method.py 最后一次修改时间:2020-02-19 19:23:50.218961
文件 decorator.py 最后一次修改时间:2020-01-24 15:33:44.498871
文件 flask_form_operate.py 最后一次修改时间:2020-02-13 13:36:16.969256
文件 flask_hello.py 最后一次修改时间:2020-02-03 21:26:26.416181
```

89 批量压缩文件的方法

首先导入 `zipfile` , 压缩和解压的Python模块。

```
import zipfile # 导入zipfile,这个是用来做:
import os
import time

def batch_zip(start_dir):
    start_dir = start_dir # 要压缩的文件夹路径
    file_news = start_dir + '.zip' # 压缩后文件夹的名字

    z = zipfile.ZipFile(file_news, 'w', zipfile.ZIP_DEFLATED)
    for dir_path, dir_names, file_names in os.walk(start_dir):
        # 这一句很重要,不replace的话,就从根目录开始复制
        f_path = dir_path.replace(start_dir, '')
        f_path = f_path and f_path + os.sep # 实现当前文件夹以及包含的所有文件
        # 的压缩
        for filename in file_names:
            z.write(os.path.join(dir_path, filename), f_path + filename)
    z.close()
    return file_news
```

调用 `batch_zip` , 压缩 `ziptest` 文件夹。

```
batch_zip('./data/ziptest')
```

90 32 位文件加密

`hashlib` 模块支持多种文件的加密策略

本案例使用 `md5` 加密策略

```
import hashlib
# 对字符串s实现32位加密
```

```
def hash_cry32(s):
    m = hashlib.md5()
    m.update((str(s).encode('utf-8'))))
    return m.hexdigest()

print(hash_cry32(1)) # c4ca4238a0b923820dcc509a6f75849b
print(hash_cry32('hello')) # 5d41402abc4b2a76b9719d911017c592
```

91 定制文件不同行的案例

比较两个文件在哪些行内容不同，返回这些行的编号，行号编号从 1 开始。

定义统计文件行数的函数

```
# 统计文件个数
def statlineCnt(statfile):
    print('文件名:'+statfile)
    cnt = 0
    with open(statfile, encoding='utf-8') as f:
        while f.readline():
            cnt += 1
    return cnt
```

复制

统计文件不同之处的子函数：

```
# more表示含有更多行数的文件
def diff(more, cnt, less):
    difflist = []
    with open(less, encoding='utf-8') as l:
        with open(more, encoding='utf-8') as m:
            lines = l.readlines()
            for i, line in enumerate(lines):
                if line.strip() != m.readline().strip():
                    difflist.append(i)

    if cnt - i > 1:
        difflist.extend(range(i + 1, cnt))
    return [no+1 for no in difflist]
```

复制

主函数：

```
# 返回的结果行号从1开始
# list表示fileA和fileB不同的行的编号

def file_diff_line_nos(fileA, fileB):
    try:
        cntA = statlineCnt(fileA)
        cntB = statlineCnt(fileB)
        if cntA > cntB:
            return diff(fileA, cntA, fileB)
        return diff(fileB, cntB, fileA)

    except Exception as e:
        print(e)
```

复制

比较两个文件 A 和 B，拿相对较短的文件去比较，过滤行后的换行符 \n 和空格。

暂未考虑某个文件最后可能有的多行空行等特殊情况

使用 file_diff_line_nos 函数：

```
if __name__ == '__main__':
    import os
    print(os.getcwd())

    '''
    例子:
    fileA = "hello world!!!!\n
            'nice to meet you'\n
            'yes'\n
            'no!\n
            'jack!'
    fileB = "hello world!!!!\n
            'nice to meet you'\n
            'yes' "
    '''
    diff = file_diff_line_nos('./testdir/a.txt', './testdir/b.txt')
    print(diff) # [4, 5]
```

复制

92 文件列表

```
import os
a = [d for d in os.listdir('D:/source/test')]
```

复制

再结合 if，只查找出文件夹，如下找出三个文件夹：

```
In [23]: import os
...: dirs = [d for d in os.listdir('D:/source/test') if os.path.isdir(d)]

In [24]: dirs
Out[24]: ['.vscode', 'templates', '__pycache__']
```

复制

只查找出文件，如下找出 .sql、.py、.db 和 .json 文件。

```
In [20]: import os
...: files = [d for d in os.listdir('D:/source/test') if os.path.isfile(d)]

In [21]: files
Out[21]:
['a.sql',
 'barchart.py',
 'barstack.py',
 'bar_numpy.py',
 'decorator.py',
 'flask_form_operate.py',
 'flask_hello.py',
 'intest.py',
 'pyecharts_server.py',
 'pytorchtest.py',
 'requests_started.py',
 'sqlite3_started.py',
 'test.db',
```

复制

```
'timLibe.json',
'uniform_points.py']
```

93 遍历目录与子目录，抓取 .py 文件

os 模块，walk 方法实现递归遍历所有文件；

os.path.splitext 返回文件的名称和扩展名；

如果扩展名匹配到 ext，则添加到 res 中。


```
import os

def get_files(directory,ext):
    res = []
    for root,dirs,files in os.walk(directory):
        for filename in files:
            name,suf = os.path.splitext(filename)
            if suf == ext:
                res.append(os.path.join(root,filename))
    return res

get_files('D:/source/python-zhuanlan','.py')
```

下一章

互动评论



说点什么

评论

The Scrapper2 个月前

例子，列举的也不错

鼓掌



存

