路径操作

路径操作模块

3.4版本之前

os.path模块

```
from os import path

p = path.join('/etc', 'sysconfig', 'network') # 拼接
print(type(p), p)
print(path.exists(p)) # 存在

print(path.split(p))  # 分割
print(path.dirname(p), path.basename(p)) # 路径和基名

print(path.abspath(''), path.abspath('.')) #绝对路径

print(path.splitdrive('o:/temp/test')) # windows方法

p1 = path.abspath(__file__)
print(p1)
while p1 != path.dirname(p1):
    dirname = path.dirname(p1)
    print(dirname)
    p1 = dirname
```

3.4版本开始

建议使用pathlib模块，提供Path对象来操作。包括目录和文件。

pathlib模块

```
from pathlib import Path
```

目录操作

初始化

```
p = Path() # 当前目录， Path()、Path('.')、Path('')
p = Path('a','b','c/d') # 当前目录下的a/b/c/d
p = Path('/etc') # 根下的etc目录
```

路径拼接和分解

操作符/
Path对象 / Path对象
Path对象 / 字符串 或者 字符串 / Path对象

分解
parts属性，可以返回路径中的每一个部分

joinpath
joinpath(*other) 连接多个字符串到Path对象中

```
p = Path()
p = p / 'a'
p1 = 'b' / p
p2 = Path('c')
p3 = p2 / p1
print(p3.parts)
p3.joinpath('etc','init.d',Path('httpd'))
```

获取路径

str 获取路径字符串
bytes 获取路径字符串的bytes

```
p = Path('/etc')
print(str(p), bytes(p))
```

父目录

parent 目录的逻辑父目录
parents 父目录**序列**，可迭代对象，索引0是直接的父

```
p = Path('/a/b/c/d')
print(p.parent.parent)
for x in p.parents:
    print(x)
```

目录组成部分

name、stem、suffix、suffixes、with_suffix(suffix)、with_name(name)
name 目录的最后一个部分
suffix 目录中最后一个部分的扩展名
stem 目录最后一个部分，没有后缀
name = stem + suffix

suffixes 返回多个扩展名列表
with_suffix(suffix) 有扩展名则替换，无则补充扩展名
with_name(name) 替换目录最后一个部分并返回一个新的路径

```
p = Path('/magedu/mysqlinstall/mysql.tar.gz')
print(p.name)
print(p.suffix)
print(p.suffixes)
print(p.stem)
print(p.with_name('mysql-5.tgz'))
print(p.with_suffix('.png'))
p = Path('README')
print(p.with_suffix('.txt'))
```

全局方法

cwd() 返回当前工作目录
home() 返回当前家目录

判断方法

exists() 目录或文件是否存在
is_dir() 是否是目录，目录存在返回True
is_file() 是否是普通文件，文件存在返回True
is_symlink() 是否是软链接
is_socket() 是否是socket文件
is_block_device() 是否是块设备
is_char_device() 是否是字符设备
is_absolute() 是否是绝对路径

绝对路径

resolve() 非Windows，返回一个新的路径，这个新路径就是当前Path对象的绝对路径，如果是软链接则直接被解析。
absolute() 获取绝对路径。

rmdir() 删除空目录。没有提供判断目录为空的方法
touch(mode=0o666, exist_ok=True) 创建一个文件
as_uri() 将路径返回成URI，例如'file:///etc/passwd'

mkdir(mode=0o777, parents=False, exist_ok=False)
parents，是否创建父目录，True等同于mkdir -p。False时，父目录不存在，则抛出FileNotFoundError
exist_ok参数，在3.5版本加入。False时，路径存在，抛出FileExistsError；True时，FileExistsError被忽略

iterdir()
迭代当前目录，不递归

```
p = Path()
p /= 'a/b/c/d'
p.exists() # True

# 创建目录
p.mkdir() # FileNotFoundError
p.mkdir(parents=True)
p.exists() # True
```

```
p.mkdir(parents=True)
p.mkdir(parents=True,exist_ok=True)
p /= 'readme.txt'
p.parent.rmdir() #
p.parent.exists() # False '/a/b/c'
p.mkdir() # FileNotFoundError
p.mkdir(parents=True) # 成功

# 遍历，并判断文件类型，如果是目录是否可以判断其是否为空目录?
for x in p.parents[len(p.parents)-1].iterdir():
    print(x, end='\t')
    if x.is_dir():
        flag = False
        for _ in x.iterdir():
            flag = True
            break

        print('dir','Not Empty' if flag else 'Empyt',sep='\t')
    elif x.is_file():
        print('file')
    else:
        print('other file')
```

通配符

glob(pattern) 通配给定的模式
rglob(pattern) 通配给定的模式，递归目录
都返回一个生成器

? 代表一个字符
* 表示任意个字符
[abc]或[a-z] 表示一个字符

```
list(p.glob('test*')) # 返回当前目录对象下的test开头的文件
list(p.glob('**/*.py')) # 递归所有目录，等同rglob
list(p.glob('**/*'))

g = p.rglob('*.py') # 生成器，递归
next(g)
list(p.rglob('*.???')) # 匹配扩展名为3个字符的文件
list(p1.rglob('[a-z]*.???')) # 匹配字母开头的且扩展名是3个字符的文件
```

匹配

match(pattern)
模式匹配，成功返回True

```
Path('a/b.py').match('*.py') # True
Path('/a/b/c.py').match('b/*.py') # True
Path('/a/b/c.py').match('a/*.py') # False
Path('/a/b/c.py').match('a/*/*.py') # True
Path('/a/b/c.py').match('a/**/*.py') # True
Path('/a/b/c.py').match('**/*.py') # True
```

stat() 相当于stat命令
lstat() 同stat()，但如果是符号链接，则显示符号链接本身的文件信息

```
# ln -s test t

from pathlib import Path
p = Path('test')
p.stat()
p1 = Path('t')
p1.stat()
p1.lstat()
```

文件操作

`Path.open(mode='r', buffering=-1, encoding=None, errors=None, newline=None)`
使用方法类似内建函数open。返回一个文件对象

3.5增加的新函数
`Path.read_bytes()`
以'rb'读取路径对应文件，并返回二进制流。看源码

`Path.read_text(encoding=None, errors=None)`
以'rt'方式读取路径对应文件，返回文本。

`Path.write_bytes(data)`
以'wb'方式写入数据到路径对应文件。

`Path.write_text(data, encoding=None, errors=None)`
以'wt'方式写入字符串到路径对应文件。

```
from pathlib import Path

p = Path('my_binary_file')
p.write_bytes(b'Binary file contents')
p.read_bytes() # b'Binary file contents'

p = Path('my_text_file')
p.write_text('Text file contents')
p.read_text() # 'Text file contents'

with p.open() as f:
    print(f.read(5))
```

os模块

操作系统平台

属性或方法	结果
os.name	windows是nt，linux是posix
os.uname()	*nix支持
sys.platform	windows显示win32，linux显示linux

os.listdir('o:/temp')
返回指定目录内容列表，不递归。

os也有open、read、write等方法，但是太底层，建议使用内建函数open、read、write，使用方式相似。

```
 建立一个软链接
$ ln -s test t1
```

os.stat(path, *, dir_fd=None, follow_symlinks=True)
本质上调用Linux系统的stat。
path：路径的string或者bytes，或者fd文件描述符
follow_symlinks True返回文件本身信息，False且如果是软链接则显示软链接本身。对于软链接本身，可以使用os.lstat方法。

```
os.stat('o:/test.txt')
os.stat_result(st_mode=33206, st_ino=281474976710698, st_dev=389224164, st_nlink=1, st_uid=0,
st_gid=0, st_size=0, st_atime=1508808249, st_mtime=1508808249, st_ctime=1508808249)
st_mode=33206 => 100666
os.stat('test')
os.stat_result(st_mode=33204, st_ino=3407875, st_dev=64768, st_nlink=1, st_uid=500, st_gid=500,
st_size=3, st_atime=1508690220, st_mtime=1508690177, st_ctime=1508690177)
```

os.chmod(path, mode, *, dir_fd=None, follow_symlinks=True)
os.chmod('test',0o777)

os.chown(path, uid, gid)
改变文件的属主、属组，但需要足够的权限