

# 系统与命令

BBCLOUD 林凡

[https://github.com/lvancer/course\\_python](https://github.com/lvancer/course_python)

lin029011@163.com

# 大纲

---

系统模块

交互式

命令行

打包

配置

练习



# 系统模块

Python的系统模块主要包括 `os` 和 `sys` 模块，这两个是很常用的模块，基本都会引入。

模块导入：

```
import os
import sys
```

系统常量：获取当前系统相关的信息。

1、当前系统：`sys.platform`。

```
print(sys.platform)
```

 输出：`win32`

Python是跨平台的语言，就是说在windows上开发的代码，在linux、macos上也可以运行。

在其他平台开发的语言也能在windows上运行。

但是实际在编写跨平台程序的时候，必然会遇到不同平台不同的解决方式。

这时候就需要获得当前系统来分离出不同平台的代码。

# 系统模块

2、Python环境变量：`sys.version`、`sys.executable`、`sys.path`。

```
print(sys.version.split()[0])    # 获得Python版本
print(sys.executable)            # 获得Python的执行文件
print(sys.path)                  # 获得Python代码的所有目录
```

输出：

```
3.6.8
C:\Users\lin02\Documents\教学\Python\course_python\code\venv\Scripts\python.exe
['C:\\Users\\lin02\\Documents\\教学\\Python\\course_python\\code\\07', 'C:\\Users\\
```

获得Python的**版本号**时，一般只要取第一块内容就可以了。可以用于不同版本的处理。

Python的**执行文件**就是使用的python，结果也验证了我们在使用**venv**的虚拟环境。

Path目录则是所有的**Python代码所在的目录**，在运行时会在这些目录下来找到**模块**。

这个列表中包括了我们的**开发目录**、**Python自带包目录**、以及**第三方包目录**。

lin029011@163.com



# 系统模块

3、跨平台符号：`os.linesep`、`os.sep`、`os.pathsep`。

```
print(os.linesep)      # 换行符, Unix为 \n , Win32为 \r\n
print(os.sep)          # 文件路径分隔符, Unix为 /, Win32为 \
print(os.pathsep)      # 多个文件路径间的分隔符, Unix为 :, Win32为 ;
```

在遇到需要编写运行不同平台的代码时，这些符号最好使用这些常量。

4、其他不重要的：`sys.maxsize`、`sys.winver`、`os.curdir`、`os.pardir`、`os.copyright`等。

```
print(sys.maxsize)     # 整数最大值 9223372036854775807
print(sys.winver)      # 主版本号 3.6
print(os.curdir)       # 当前目录 .
print(os.pardir)       # 上一级目录 ..
print(sys.copyright)   # 版权信息
```

# 系统模块

os模块的一个重要的功能就是对文件目录进行操作。我们先来学习文件路径的操作。

文件路径操作：`os.path`【os的子模块】，包含了各种处理文件和文件名的方法。

1、几个常量：`os.path.sep`、`os.path.pathsep`、`os.path.extsep`。

```
print(os.path.sep)      # 路径分隔符 (Unix为 /, Win为 \\  
print(os.path.pathsep)  # 多个路径间的分隔符, 多用于环境变量 (Unix为 :, Win为 ;)  
print(os.path.extsep)   # 后缀名符号 一般为 .
```

与os模块差不多。

2、路径组合：`os.path.join`。

```
path = os.path.join('newdir', 'python.txt')
```

 输出：`newdir\python.txt`

`join`的参数是一个不定参数，可以填入无限多个参数，`print`方法也有同样的方式。

`join`会将所有的参数组合成一个路径，使用该系统的分隔符。

# 系统模块

3、路径分割：`os.path.split`、`os.path.dirname`、`os.path.basename`、`os.path.splitext`。

```
_dir, _file = os.path.split(path)    # 分割成目录和文件
print(_dir)                          # 目录
print(_file)                         # 文件
print(os.path.dirname(path))         # 与_dir一样
print(os.path.basename(path))       # 与_file一样
```

输出：

```
newdir
python.txt
```

`split`方法将路径分解为目录和文件两个。这里文件也可以是最后一级目录。

或者也可以使用`dirname`和`basename`方法分别获得目录和文件。

```
_filename, _ext = os.path.splitext(_file)    # 文件名和扩展名
print(_filename, _ext)
```

```
python .txt
```

`splitext`用于分解出扩展名，也是很常用的一个方法。

# 系统模块

4、路径变换：`os.path.relpath`、`os.path.abspath`、`os.path.commonprefix`。

```
print(os.path.relpath(path))    # 获得相对于当前路径的路径
print(os.path.abspath(path))    # 绝对路径
```

输出：

```
newdir\python.txt
C:\Users\lin02\Documents\教学\Python\course_python\code\07\newdir\python.txt
```

`relpath`将路径转化为基于当前目录的**相对路径**。`abspace`将路径转化为**绝对路径**。

```
f1 = 'newdir/py/file.txt'
f2 = 'newdir/py/l1/file.txt'
f3 = 'newdir/py/l2/file.txt'
print(os.path.commonprefix([f1, f2, f3]))
```

输出：

```
newdir/py/
```

`commonprefix`可以获得的几个路径【列表传入】的**共同前缀**。



# 系统模块

5、判断路径存在：`os.path.exists`、`os.path.isdir`、`os.path.isfile`。

```
print(os.path.exists(path))      # 判断是否存在
print(os.path.isdir(path))       # 判断是否文件夹
print(os.path.isfile(path))      # 判断是否文件
```

`exists`存在，`isdir`存在且是文件夹，`isfile`存在且是文件。

6、文件属性：`os.path.getatime`、`os.path.getmtime`、`os.path.getctime`、`os.path.getsize`。

```
print(os.path.getatime('./os1.py')) # 访问时间
print(os.path.getmtime('./os1.py')) # 修改时间
print(os.path.getctime('./os1.py')) # 创建时间
print(os.path.getsize('./os1.py'))  # 文件大小
```

文件**必须存在**，否则报错。

时间返回的是float类型**时间戳**，文件大小以**kb**为单位。

# 系统模块

文件操作：创建、修改、删除、查看。

1、创建文件夹：`os.mkdir`、`os.makedirs`。

```
os.mkdir('./newdir')    # 创建目录
```

`mkdir`只能创建一级目录，必须保证前一级目录`os.path.dirname(path)`是存在的。

如果不存在，必须使用`makedirs`。

```
os.mkdir('./newdir2/abc')    # 错误，newdir2文件夹并不存在  
os.makedirs('./newdir2/abc') # 正确，创建多级目录
```

如果目录已经存在，两个方法都会报错，一般使用前会先判断是否存在。

# 系统模块

2、删除目录：os.rmdir、os.removedirs。

```
os.rmdir('./newdir')      # 删除目录  
os.removedirs('./newdir2/abc') # 删除多级目录
```

**rmdir**要删除的目录下必须是空的。

**removedirs**会从最底层目录开始向上删除，每级目录也都要保证是空的。

3、删除文件：**os.remove**。

```
open('./abc.txt', 'w').close() # 创建一个文件  
os.remove('./abc.txt') # 删除文件
```

**remove**会删除一个文件，文件**必须存在**。

# 系统模块

## 4、文件移动：`os.rename`。

```
os.rename('./abc.txt', './ccc.txt')
```

第一个参数为要移动的文件，第二个参数为目标地址。

要保证要移动文件存在；目标地址要保证目录存在，且该地址不存在。

## 5、文件列表：`os.listdir`。

```
print(os.listdir('./'))
```

参数写入一个文件目录，返回该目录下所有文件和目录列表。

```
['.idea', '01', '02', '03', '04', '05', '06', '07', 'venv']
```

另一种获取文件列表的方式是通过`glob`模块，使用`*`作为通配符，找到格式一致的文件。

```
import glob  
print(glob.glob('./*/*.py'))
```

`*`匹配任意字符，可以实现比`listdir`更强大的功能。

lin029011@163.com



# 系统模块

## 6、遍历文件夹：`os.walk`。

使用`listdir`来实现文件夹遍历，只能实现在有规律的文件夹上。

`walk`可以实现某个文件夹下所有文件的遍历。

```
for root, dirs, files in os.walk('../06/'):
    for f in files:
        filename = os.path.join(root, f)
        print(filename)
```

`root`是当前遍历到的子目录，`walk`循环会逐层将所有子目录都遍历一遍。

`dirs`是该子目录下的所有文件夹，比较少使用。

`files`是该子目录下的所有文件，可以实现文件遍历搜索各种功能。

# 交互式

Python有两种运行方式：一种是**脚本式**，就是我们所开发的python文件。

另一种是**交互式**，主要用于简单的python运行或调试。

```
C:\Users\lin02>python
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018,
Type "help", "copyright", "credits" or "license" f
>>> import os
>>> print('Hello')
Hello
>>>
```

在**cmd**命令行界面中输入**python**，进入了交互式界面。输入一块代码，马上会显示结果。

交互式虽然比较简单，很难实现复杂的功能，

但也发展出了像**Jupyter notebook**这样的在**数据分析**、**机器学习**方面常用的工具。

关于**Jupyter**未来有相关课再做介绍。

# 命令行

注意这里使用的是系统安装的python，而不是虚拟环境

脚本运行的方式是**命令行**：

```
C:\Users\lin02\Documents\教学\Python\course_python\code\07>python main.py  
Hello World
```

在cmd界面通过python命令运行程序。

我们如果观察Pycharm执行Run时打印的内容

```
C:\Users\lin02\Documents\教学\Python\course_python\code\venv\Scripts\python.exe C:/Users/I  
Hello World|
```

其实也是执行了venv下的python命令，后面跟上要执行的文件。

命令行实际就是使用了一个python命令，它的第一个参数必须是文件。

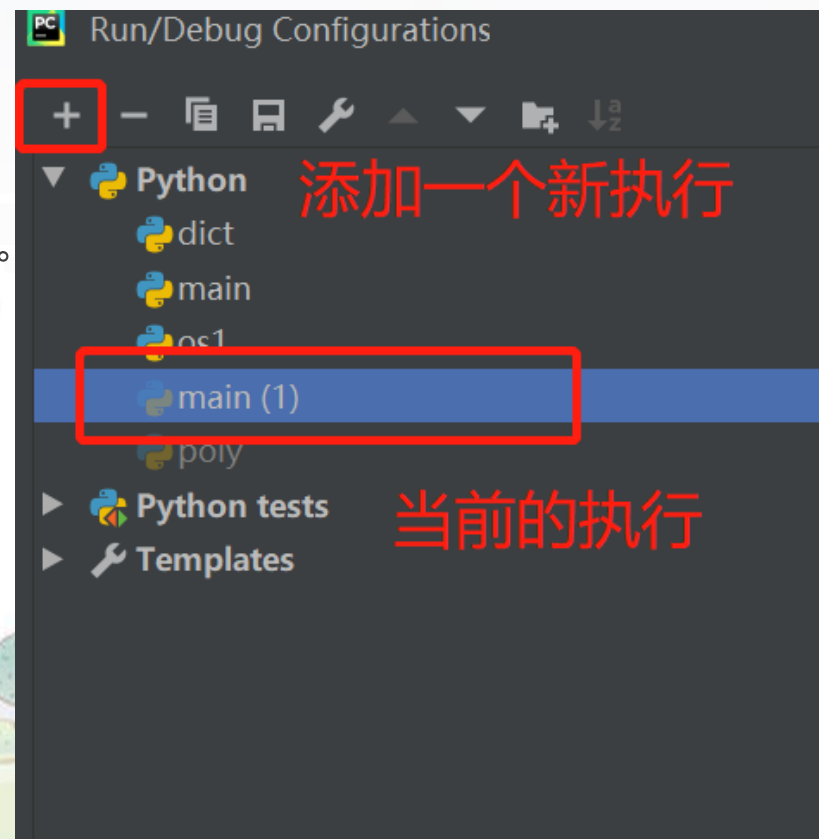
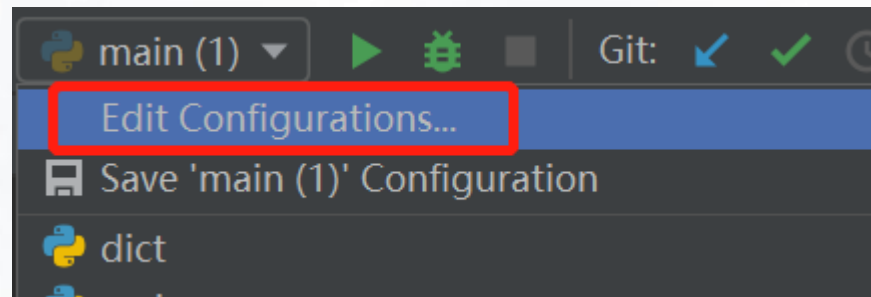
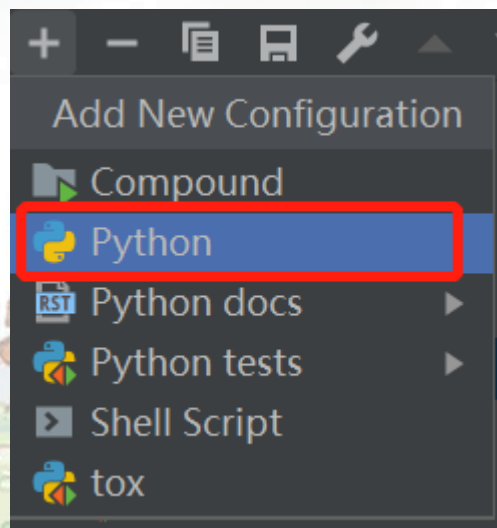
# 命令行

**命令参数**：python命令后面除了第一个文件参数，还可以跟上任意参数来方便使用。

**Pycharm**中添加参数：

右上角点击下拉框，选择**Edit Configurations**。

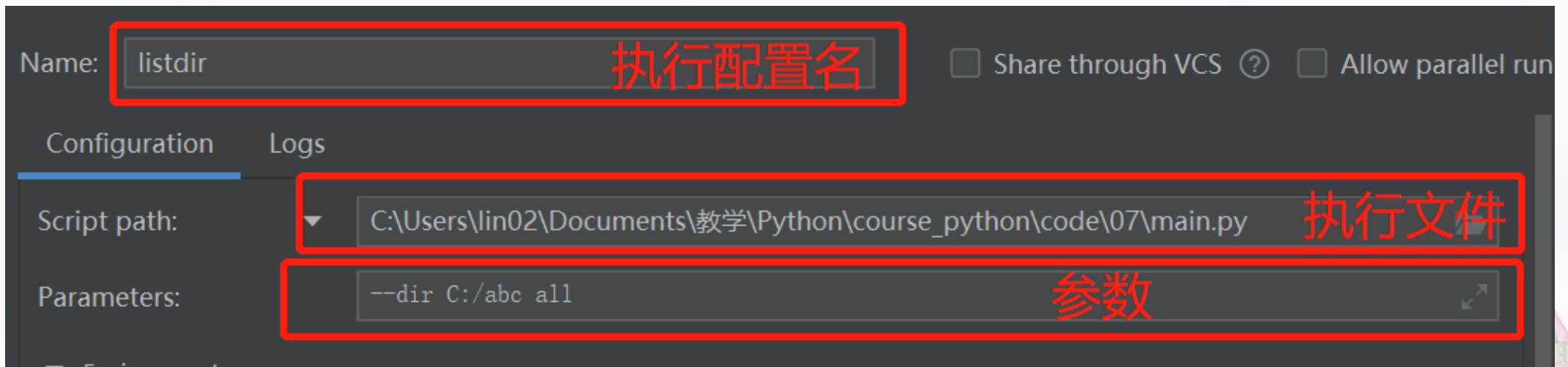
左边是所有的**执行配置**，我们可以自己添加一个新的。





# 命令行

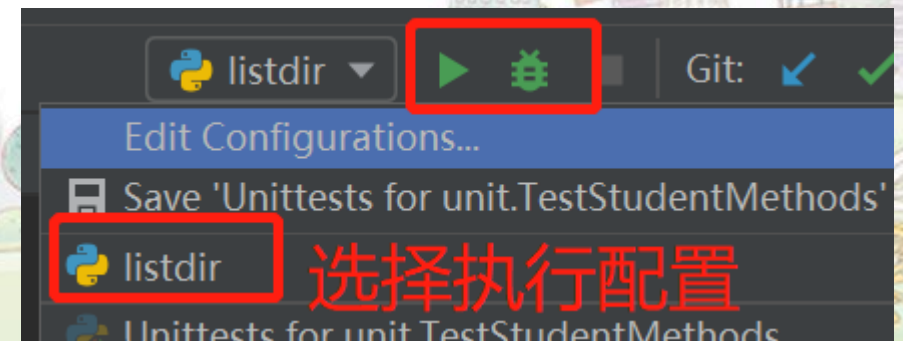
输入执行配置名，选择执行文件，最后在Parameters里写入参数。右下角OK保存。



要执行时，在右上角选择要运行的执行配置，  
使用旁边的两个按钮进行运行和调试。

这里的配置相当于执行了

`python main.py --dir C:/abc -all`



lin029011@163.com

# 命令行

获取参数：`sys.argv`。

```
import sys
print(sys.argv)      # 获取所有参数
```

返回了一个参数列表，第一个是执行文件，后面的就是我们刚才写入的参数。

下面是通过获取参数来展示输入目录下的文件列表，且在-all时同时输出文件夹。

```
def listdir(path, with_dir=False):
    result = []
    for f in os.listdir(path):
        if with_dir or os.path.isfile(os.path.join(path, f)):
            result.append(f)
    return result
if __name__ == '__main__':
    if sys.argv[1] == '--dir':
        path = sys.argv[2]
        with_dir = len(sys.argv) > 3 and sys.argv[3] == 'all'
        print(listdir(path, with_dir))
```

# 命令行

更优雅的方式：`argparse`模块。

```
import argparse
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--dir', default='.')
    parser.add_argument('all', nargs='?', default=False)
    args = parser.parse_args()
    print(listdir(args.dir, args.all))
```

# 开启参数解析  
# 必填位置参数  
# 可选参数，默认为False

先创建一个参数解析器`parser`，通过`add_argument`添加参数定义，最后解析到`args`中。

添加`--dir` 位置参数，横杠打头的参数都是位置参数，后面必须跟上参数值。

添加 `all` 可选参数，`nargs='?'`定义了可选，默认不存在时为False，存在则解析为all。

最后通过`parse_args`将参数解析出来到变量中，就可以直接使用成员变量的方式调用了。

lin029011@163.com

# 打包

学习完命令行参数，我们就可以让用户根据设定好的参数进行程序的使用。

但是，这还需要用户安装环境。让用户可以**零基础**使用我们的程序，就要进行**打包**。

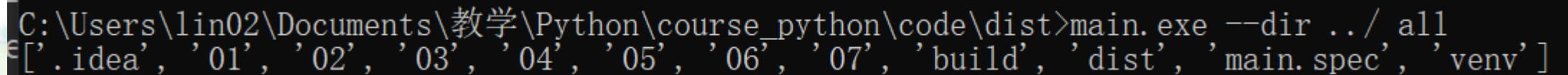
**打包模块**：**pyinstaller**模块。如果出现安装失败的情况，请先安装**wheel**模块。

**打包命令**：先在底部开启命令行模式**Terminal**。

输入**`.\venv\Scripts\pyinstaller.exe -F .\07\main.py`**

第一个是pyinstall命令，-F 表示打包所有，最后是**启动文件**。

然后就可以在当前项目下生成**dist文件夹**，里面就是打包完成的**exe文件**。



```
C:\Users\lin02\Documents\教学\Python\course_python\code\dist>main.exe --dir ../ all  
['.idea', '01', '02', '03', '04', '05', '06', '07', 'build', 'dist', 'main.spec', 'venv']
```

lin029011@163.com



# 配置

除了可以通过命令行参数来传递参数，还可以通过配置文件来进行。

配置文件：**ini**文件。创建一个**config.ini**配置文件。

[]里面是**section**，下面的是具体的**option**。

读取配置文件：**configparser**模块。

```
conf = configparser.ConfigParser() # 创建对象
conf.read('config.ini')           # 读取文件
print(conf['DEFAULT']['dir'])      # 获取配置
print(int(conf['INFO']['x']))
print(conf['INFO']['save'] == 'True')
```

```
[DEFAULT]
dir = C:/Users
```

```
[INFO]
x = 1
save = True
```

输出：

```
C:/Users
1
True
```

ConfigParser对象，读取配置文件后，就可以直接获取里面的配置。

通过**类似字典**的方式获取配置[**section**][**option**]，所有配置获取出来都是**字符串**。

lin029011@163.com

# 配置

其他获取方式：`get`、`getint`、`getboolean`、`getfloat`。

```
conf.get('DEFAULT', 'dir')      # 获取配置  
conf.getint('INFO', 'x')        # int  
conf.getboolean('INFO', 'save') # boolean
```

与直接获取不同的是，`get`方法可以进行基础转换。

判断方法：`has_section`、`has_option`。

```
conf.has_section('DEFAULT')     # 是否存在section  
conf.has_option('INFO', 'dir')  # 是否存在option
```

我们发现`[INFO][dir]`的option是存在的，即第二行代码返回的是`True`。

这是因为`DEFAULT`是配置文件的父类，其他所有的section都继承自`DEFAULT`。

与类一样，同样也可以进行复写。

# 配置

配置遍历：`sections`、`options`、`items`。

```
print(conf.sections())      # 返回所有section
print(conf.options('INFO')) # 返回section下所有options
```

输出：

```
['INFO']
['x', 'save', 'dir']
```

`sections`并没有返回DEFAULT，因为他是父类。

`options`里也返回了从父类继承的dir。

```
for group, section in conf.items():
    for key, value in section.items():
        print('conf[{}][{}] = {}'.format(group, key, value))
```

输出：

```
conf[DEFAULT][dir] = C:/Users
conf[INFO][x] = 1
conf[INFO][save] = True
conf[INFO][dir] = C:/Users
```

循环中的`group`、`section`分别是名称和section对象。

section对象也有items方法，返回options的内容。

上面的代码就打印了所有的配置。

# 配置

配置修改：`add_section`、`set`、`remove_option`、`remove_section`。

```
conf.add_section('DEBUG')           # 添加新section
conf.set('DEBUG', 'y', '1')          # 添加新option
conf.set('INFO', 'dir', 'C:/Users/lin02') # 修改option
with open('config.ini', 'w') as f:
    conf.write(f)                    # 写入文件
```

使用`add_section`、`set`进行添加修改操作，然后`write`写入文件，一般我们会覆盖原文件。

`remove_option`、`remove_section`用于删除。

如果没有进行读取【`read`】操作，上面的方法则可以直接创建一个配置文件。



# 配置

命令行中添加一个默认值，从配置文件读取。

```
import argparse
import configparser
if __name__ == '__main__':
    conf = configparser.ConfigParser()
    conf.read('config.ini')
    default_dir = conf.get('INFO', 'dir') # 获取默认值
    parser = argparse.ArgumentParser()
    parser.add_argument('--dir', default=default_dir)
    parser.add_argument('all', nargs='?', default=False)
    args = parser.parse_args()
    print(listdir(args.dir, args.all))
```

# 配置

重新打包后，将`config.ini`放在`exe`文件同级目录，就可以正常使用了。



因为我们使用的是相对路径，而程序的相对路径就是`exe`所在的路径。

有了配置文件，我们的程序就更完善了。

# 练习

---

- 1、编写一个简化版的dir命令。
- 2、编写一个批量修改文件名的工具，并打包成exe。
  - 1、自己创建一个文件夹，里面任意一推的文件。
  - 2、给下面的文件批量加一个前缀。
  - 3、根据文件创建时间排序，在扩展名前加序号，序号递增。

main.exe --dir C:\abc --prefix hh

xxx.txt => hhxxx1.txt