Simi Documentation / Python

Author: Craig Brown
Version: 1.0.0
Date: Feb 2, 2025
Source: https://github.com/saidake/simi/tree/master/docs/pdf

Python / Concept

基础

脚本语言：　解释器直接运行源代码，无需编译成可执行程序。(效率和执行性能差)

跨平台：　不同平台对应的解释器都可以运行同一 python 代码，python 开源

标识符

由字母 下划线 数字组成　不能数字开头

Command

python -m venv <path>

.\venv\Scripts\activate

deactivate

pip install -r <requirements-file>

pip install <python-module>

Python / Core

变量常量

```
# coding=utf-8      申明执行编码
a= 100   a=0b1000   a=0o1000    a=0x189    a=3+4j   a=3.0+4.0J        数字
a = 1000.0                                                            浮点型
a = "John"   a='John'   a="ab"+"c"*2   --> abcc    a= b'a\x01c',  a=u'sp\xc4m'   a=r'C:\Desktop'  a=f'xx{val}xx'  字符串
//EX："xxabc"[2:5]   取出 [2, 5) 的字符串，2 可省
```

(u 字符串后面以 Unicode 格式 进行编码，r 字符串不识别转义字符，b 字符串表示字符串是 bytes 类型，f 模板字符串)

```
a = True   a=False                                                    布尔
a = {"name":"lala","age":12 }   a= dict(hours=10)       a['b']        字典
a = [[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]                     数组
a= {'a',  'b',  'c'}     a=set( 'abc' )                               集合 set
a= (1,  'spam',  4,  'U')          a= tuple('spam')                   元组 tuple    // EX：  a,b,c,d=a  取出所有元素
val= xxtup[1:5]   取出 1 到 5 的元素      val= xxtup[3]     取出第三个
```

类型转换

```
int( xx )              转换为整数
long( x )              转换为长整数
float(x )              转换为浮点数
complex(real [,imag ])       创建一个复数
    str( x )               转换为字符串
    repr( x )              转换为表达式字符串
    eval( "xxx" )        用来计算在字符串中的有效 Python 表达式,并返回一个对象
tuple( s )         将序列 s 转换为一个元组
list( s )          将序列 s 转换为一个列表
chr( 68 )          将一个整数转换为一个字符
unichr( 2 )        将一个整数转换为 Unicode 字符
ord( 'x' )         将一个字符转换为它的整数值
hex( 99 )          将一个整数转换为一个十六进制字符串
```

oct(88)　　将一个整数转换为一个八进制字符串
bytearray(b'\x01\x02\x03')　　返回新字节数组

type(xxval)　　返回变量类型不会认为子类是一种父类类型，不考虑继承关系。　　//EX:　type(a) == str
isinstance(xxval, xxtype)　判断是某个类型，会认为子类是一种父类类型，考虑继承关系。　　//EX:　isinstance(1,　int)
isinstance(1.0,　float)　　isinstance("xxx",　,str)

语句 运算符

```
def  lala():
    pass                占位，相当于一行代码

for val in xxlist:          遍历获取值（字典为键）
     print ("xx")       遍历
for ind in range(len(xxlist)):
   print( xxlist[ind] )
for ind, val in enumerate( xxsequence ):
  print ind, val
return {c.name: getattr(self, c.name, None) for c in self.__table__.columns}   单行写法

while not (True and False):       false 会继续循环

if X is None:
if not X:                 当 X 为 None，False, "", 0, 空列表[], 空字典{}, 空元组()这些时       not X 为真（即无法分辨出他们之间的不同)
if not X is None:
a if a>b else b          表达式为真：a　表达式为假：b
currentBtn=nextBtn if nextBtn.text == "下一页" else  (nextBtn2 if nextBtn2.text == "下一页" else (nextBtn3 if nextBtn3.text == "下一页" else  None))

9 // 8　　整数除法返回整数
```

Python / Function

函数

```
def func( a, b=9, c=None, *rest):          *rest 长度不固定的列表。
   xxglobalval +=1                         函数内部修改全局变量  会导致解释器识别 globalVal 为局部变量，在之后继续当做全局变量访问会报错
   return a+b
func(1,2,3,4)

def func (a, b, **rest):                   **rest 长度不固定的字典
   return a+b
func (1, 2, name="xx", age=12 )
func (a= 1, b= 2, name="xx", age=12 )
```

类

```
class Person:
    xxcount = 0                        静态成员
   def __init__(self, name, age):      构造函数（创建了这个类的实例时就会调用该方法）
       self.name = name
```

```
        self.age = age
        Person.xxcount += 1
    def func(self):                         普通函数
        print(self.name, self.age)

    @staticmethod                           静态函数
    def xxstafunc( xxx ):                   静态函数不需要 self    // Person.xxstafunc
      print(Person.xxcount)

    @classmethod                    不需要实例化类就可以被类本身调用
    def xxclsfunc(cls):             cls 表示没用被实例化的类本身
        print(cls.xxcount)
        cls().func()


class Son( Person ):                        继承父类
    def __init__(self, name, age):      继承
        Person.__init__(self)
        self.name= name
        self.age = age
per = Person("aa", 22)
```

内置属性：

Person.__dict__　　类的属性（包含一个字典，由类的数据属性组成）

Person.__doc__　　类的文档字符串

Person.__name__　　类名

Person.__module__　　类定义所在的模块　　//EX：类的全名是'__main__.className'，如果类位于一个导入模块 mymod 中，那么 className.__module__ 等于 mymod）

Person.__bases__　　类的所有父类构成元素（包含了一个由所有父类组成的元组）

访问属性：

hasattr(per, 'age')　　如果存在 'age' 属性返回 True。

getattr(per, 'age')　　返回 'age' 属性的值

setattr(per, 'age', 8)　　添加属性 'age' 值为 8

delattr(per, 'age')　　删除属性 'age'

异常

```
try:
  raise Expection("aaa")        抛出异常
except Expection:
  print("aaa")
finally:
    print("final")
```

包

from xxmodule import a, b　　绝对路径导入（首先检查 a 是不是__init__.py 文件的变量。然后检查是不是 subpackage，再检查是不是 module，最后抛出 ImportError）

xxmodule 能使用在__init__.py 文件内，它上层被导入的包

from ...package import *　　相对路径导入（第一个.表示当前目录，后面的每一个 . 表示上一层目录）

Python / Build-in Libraries

全局

input("xxx")
print("xx")
Decimal(1.0) 小数
Fraction(1, 3) 分数
__name__ 通过判断__name__的值，就可以区分 py 文件是直接被运行，还是被引入其他程序中（被导入： “temp2.py” 仅执行： "__main__"）
__file__ 当前运行的 py 文件相对路径（不包含在 sys.path 里面时 返回一个绝对路径） //EX：look.py

print(f'xxx{text}')

list

append() 追加元素
insert(2, "99") 在索引插入值
count() 方法用于统计某个元素在列表中出现的次数（> 0）
extend([1, 2, 3]) 当前列表 追加新列表
index(xxval) 从数组中找出某个值第一个匹配项的索引值

tuple

cmp(xxtup1, xxtup2): 比较两个元组元素 （元组中的元素是不允许被修改的）
len(xxtup): 返回元组中元素的个数
max(xxtup): 返回元组中元素最大的值
min(xxtup) 返回元组中元素最小的值
tuple(xxseq) 将列表转化为元组
index(xxval) 从元组中找出某个值第一个匹配项的索引值
count(xxval) 统计某个元素在元组中出现的次数

str

"a {0} b {1}".format(name ,age) 填充前方括号，返回格式化字符串
"a {name} b {url}".format(**xxDict) 通过字典的键设置参数
"a {0[0]} b {0[1]}".format(xxList) 通过列表索引设置参数（前方的 0 是必须的）
"John %s like %s" %('repstr1', 'replace2') 格式化字符串
"Hey %(name) s, there is a 0x%(errno)x error!" %{ "name": name, "errno": errno } 格式化字符串
result = "world" in str 判断字符串是否包含 【false】

replace("xxold", "xxnew") 不支持正则（不改变原本字符串）
split(self, sep, maxsplit): 分隔符，分隔次数
rstrip("\n") 删除字符串末尾指定字符
'\n'.join(xxlist) 用一个字符连接数组
str.upper()) # 把所有字符中的小写字母转换成大写字母
str.lower()) # 把所有字符中的大写字母转换成小写字母
str.capitalize()) # 把第一个字母转化为大写字母，其余小写
str.title()) # 把每个单词的第一个字母转化为大写，其余小写

dict

dict(dict1, **dict2) 合并字典（键相同时，后方覆盖前方）
__contains__("xxkey") 是否有一个键
get("xxkey") 根据键获取一个值

update({"new_food":0}) 添加键值对

keys( ) 返回对象 key 迭代器，不是数组 // len(xxdict.keys()) 键个数 list(dic.keys()) 字典键列表 list(dic.values()) 字典值列表

file

open("test.txt","w", encoding="utf-8") 直接打开一个文件，如果文件不存在则创建文件

w 以写方式打开（原有内容会被删除）

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（+ 加号表示 如果该文件不存在，创建新文件）

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+)

write("content") 写入文件

close() 关闭文件

read() 每次读取整个文件，它通常用于将文件内容放到一个字符串变量中。如果文件大于可用内存，为了保险起见，可以反复调用 read(size)方法，每次最多读取 size 个字节的内容。

readlines() 之间的差异是后者一次读取整个文件，象 .read() 一样。.readlines() 自动将文件内容分析成一个行的列表，该列表可以由 Python 的 for ... in ... 结构进行处理。

readline() 每次只读取一行，通常比 readlines() 慢得多。仅当没有足够内存可以一次读取整个文件时，才应该使用 readline()。

re（正则）

sub(r'\n|\s|p', "xxreplace" ,"xxtarget") xxreplace 替换 xxtarget 内部正则匹配部分 之后的字符串【没有返回原来的字符串 】

search(r'\n|\s|p', "xxx", flags=0) 匹配整个字符串，直到找到一个匹配【没找到返回 None】

result.group() 返回匹配结果第一个 // <re.Match object; span=(11, 49), match='154792957E1EB672580707A0129CF736.node1'>

result.groups() 返回所有匹配结果

time（代码停顿）

sleep(10) 暂停 10 秒

time() 返回秒级时间戳 //EX：1639025762.660346

tsp 原始时间数据 // 1499825149.257892

int(tsp) 秒级时间戳 // 1499825149 (10 位)

int(round(tsp * 1000)) 毫秒级时间戳 // 1499825149257 (13 位)

int(round(tsp * 1000000)) 微秒级时间戳 // 1499825149257892 (16 位)

ctime(tsp) 格式化秒级时间戳 Tue Feb 17 10:00:18 2013

strftime("%Y-%m-%d %H:%M:%S", time.localtime()) 获取格式化时间

datetime

%y 两位数的年份表示（00-99）
%Y 四位数的年份表示（000-9999）
%m 月份（01-12）
%d 月内中的一天（0-31）
%H 24小时制小时数（0-23）
%I 12小时制小时数（01-12）
%M 分钟数（00=59）
%S 秒（00-59）
%a 本地简化星期名称
%A 本地完整星期名称
%b 本地简化的月份名称
%B 本地完整的月份名称
%c 本地相应的日期表示和时间表示
%j 年内的一天（001-366）
%p 本地A.M.或P.M.的等价符
%U 一年中的星期数（00-53）星期天为星期的开始
%w 星期（0-6），星期天为星期的开始
%W 一年中的星期数（00-53）星期一为星期的开始

(dobj +datetime.timedelta(hours=1)).strftime("%Y-%m-%d %H:%M:%S")　当前时间加 1 小时【datetime 对象能直接比较和加减】

(dobj - relativedelta(years=1)). strftime("%Y-%m-%d %H:%M:%S")　　当前时间减 1 年

time()

datetime(2012, 04, 22).strftime('%w')　年月日转换成时间，获取今天的星期

datetime：

　　now()　　获取当前时间　// 2021-12-09 12:56:02.660346　【datetime 对象可以直接比对大小】

　　strptime('09/19/18 13:55:26', '%m/%d/%y %H:%M:%S')　字符串 转 datetime

os

mknod("text.txt")　　创建空文件

makedirs("/usr")　　创建路径

remove("/usr/test.txt")　　删除文件

removedirs（"/usr/home"）　删除多个目录

　path.exists("test.txt")　　路径是否存在

　path.dirname(r"/usr/test.txt")　　去掉文件名，返回目录

　path.abspath(os.path.dirname(__file__))　　获取当前文件的目录　　//EX：__file__：D:\Desktop\DevProject\loopo_python\sdkAi2.py

　path.realpath(".")　　获取当前文件的绝对路径

　path.join(dir_name, 'pac01', 'demo.txt')　凭借路径

environ['HOME']　　获取环境变量

system("adb shell")　　执行终端命令

chdir("xxpath")　　方法用于改变当前工作目录到指定的路径

listdir()　　方法用于返回指定的文件夹包含的文件或文件夹的名字的列表

json

dumps({'a': 'Runoob', 'b': 7} , sort_keys=True, indent=4, separators=(',', ': '))　　转换为 json 的双字符串格式　　// [{ 'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4, 'e' : 5 }]

loads(xxjsonData)　　j 串转 json

load(xxfileobj)　　读取文件对象

demjson

demjson.encode　　将 Python 对象编码成 JSON 字符串

demjson.decode　　将已编码的 JSON 字符串解码为 Python 对象

threating

```
from threading import  Thread      子线程   (主线程会等待所有的子线程结束后才结束)
from threading import  Lock        线程锁
from threading import Timer        计时器

thread1 = Thread (target=func, args=(1, ) )      创建新线程（执行函数，参数）
thread1.setDaemon(True)                          如果主线程结束了，也随之结束
thread1.start()                                   开启线程

lock = threading.Lock()
lock.acquire()        获取锁  (获取之后 多个线程只能有一个调用下方方法)
lock.release()        释放锁

thread1=Timer(10,test1 ,() )      延迟多长时间执行任务(单位: 秒)      要执行的任务, 即函数      调用函数的参数(tuple)
thread2=Timer(10, test2 ,() )

thread1.start()
thread2.start()
thread1.join()          主线程一直等待全部的子线程结束之后，才继续执行
```

base64

```
#image 转 base64
import base64
with open("C:\\Users\\wonai\\Desktop\\1.jpg","rb") as f:#转为二进制格式
    base64_data = base64.b64encode(f.read())#使用 base64 进行加密     //  
    print(base64_data)
    file=open('1.txt','wt')#写成文本格式
    file.write(base64_data)
    file.close()
```

urllib

```
from urllib import parse
        urlencode(dict1)          编码 url     //将字典{k1:v1,k2:v2}  转化为  k1=v1&k2=v2
        unquote(url_data)       #解码 url     //将 k1=v1&k2=v2   转化为 字典{k1:v1,k2:v2}

        quote(str1)              #quote()将字符串进行编码
        unquote(url_data)    #解码 url
```

pathlib

```
import pathlib

xxdir=pathlib.Path( "xxpath" )       返回路径对象
    xxdir.glob('train/*/*.jpg')       返回找到的图片结果   list(  xxdir.glob("xxx") )
```

random

```
random.choice( xxlist  )  随机选择一个元素
```

math

```
全局：abs(-45)：45
```

logging

```
日志一共分成 5 个等级，从低到高分别是：DEBUG INFO WARNING ERROR CRITICAL。
```

```
logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)-8s: %(message)s')
        filename='./log/log.txt',       #输出文件
        filemode='w',
)

logging.info('this is a loggging info message')
logging.debug('this is a loggging debug message')
logging.warning('this is loggging a warning message')
logging.error('this is an loggging error message')
logging.critical('this is a loggging critical message')
```

%(levelno)s:	打印日志级别的数值
%(levelname)s:	打印日志级别名称
%(pathname)s:	打印当前执行程序的路径，其实就是 sys.argv[0]
%(filename)s:	打印当前执行程序名
%(funcName)s:	打印日志的当前函数
%(lineno)d:	打印日志的当前行号
%(asctime)s:	打印日志的时间
%(thread)d:	打印线程 ID
%(threadName)s:	打印线程名称
%(process)d:	打印进程 ID
%(message)s:	打印日志信息

traceback

traceback.print_exc() 打印异常信息

random

一.Python 自带的 random 库

1.参生 n--m 范围内的一个随机数: random.randint(n,m)

2.产生 0 到 1 之间的浮点数: random.random()

3.产生 n---m 之间的浮点数: random.uniform(1.1,5.4)

4.产生从 n---m 间隔为 k 的整数: random.randrange(n,m,k)

5.从序列中随机选取一个元素: random.choice([1, 2, 3, 4, 5, 6, 7, 8, 9, 0])

Python / Third-party Package

核心包

requests

post(url="http://www.baidu.com", json={"a":"b"}, headers = {'user-agent': 'my-app/0.0.1''}, cookies = {'key':'value'} , proxies=proxies) 发送 post 请求 【res】

proxies={'http':"http://"+proxy, "https":"https://"+proxy} 可以不带 https

get(url='http://www.baidu.com', params={'a': 'b'}, headers = {'user-agent': 'my-app/0.0.1''}, cookies = {'key':'value'} , proxies=proxies, timeout=120) 带参数的 get 请求，秒数超时

encoding 获取当前的编码

encoding = 'utf-8'　　设置编码
text　　以 encoding 解析返回内容（字符串方式的响应体，会自动根据响应头部的字符编码进行解码）
content　　以字节形式，返回二进制数据（字节方式的响应体，会自动为你解码 gzip 和 deflate 压缩）
headers　　以字典对象存储服务器响应头（这个字典比较特殊，字典键不区分大小写，若键不存在则返回None）
status_code　　响应状态码
raw　　返回原始响应体，也就是 urllib 的 response 对象，使用 r.raw.read()
ok　　是否 200 状态码　【True】

json()　　Requests 中内置的 JSON 解码器，以 json 形式返回（不是 json 数据解析出错会抛异常）
raise_for_status()　　失败请求时(非 200 响应)，抛出异常
session()　　获取 session
get('https://xxx ')　发送请求
cookies　　获取 session 饼干
get_dict()　　获取饼干字典

标准化 class 写法

selenium

from selenium import webdriver　　用于打开网站（Helium 更高级的功能）
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException

WebDriverWait：显示等待，同样也是 webdirver 提供的方法。在设置时间内，默认每隔一段时间检测一次当前页面元素是否存在，如果超过设置时间检测不到则抛出异常。
默认检测频率为 0.5s，默认抛出异常为：NoSuchElementException
driver:　　传入 WebDriver 实例，即我们上例中的 driver
timeout:　　超时时间，等待的最长时间（同时要考虑隐性等待时间）
poll_frequency:　　调用 until 或 until_not 中的方法的间隔时间，默认是 0.5 秒
ignored_exceptions:　　忽略的异常，如果在调用 until 或 until_not 的过程中抛出这个元组中的异常，则不中断代码，继续等待，
如果抛出的是这个元组外的异常，则中断代码，抛出异常。默认只有 NoSuchElementException。
EC.visibility_of_element_located(By.XPATH, "//div")　　判断某个 locator 元素是否可见。可见代表非隐藏、可显示，并且元素的宽和高都大于 0
EC.element_to_be_clickable(By.XPATH, "//div")　　判断某个 locator 元素是否可点击
WebDriverWait(driver, 20).until(EC.element_to_be_clickable(NEXTBUTTON)).click()

until　　在等待期间，每隔一段时间（__init__中的 poll_frequency）调用这个传入的方法，直到返回值不是 False
message: 如果超时，抛出 TimeoutException，将 message 传入异常
until_not　　与 until 相反，until 是当某元素出现或什么条件成立则继续执行， until_not 是当某元素消失或什么条件不成立则继续执行，参数也相同，不再赘述。

初始化浏览器

option = webdriver.ChromeOptions()　返回驱动选项
add_argument(('--proxy-server=' + "'199.2.2.1:4455"))　　自动加上 http://　可以使用 socks5://

```
        add_argument('--disable-gpu')                   谷歌文档提到需要加上这个属性来规避 bug
        add_argument('--hide-scrollbars')               隐藏滚动条, 应对一些特殊页面
        add_argument('--start-maximized')               启动就最大化
        add_argument('--headless')                      浏览器不提供可视化页面. linux 下如果系统不支持可视化不加这条会启动失败
        add_argument('--user-agent=xxxxxxxx')           修改 HTTP 请求头部的 Agent 字符串
        add_argument('--lang=zh-CN')                    设置语言为简体中文
        add_argument("--user-data-dir="+r"C:/Users/Administrator/AppData/Local/Google/Chrome/User Data/")  添加个人插件到浏览器中
        add_extension('assets/Tampermonkey.crx' )               Add extension.（无头浏览器不可用）
        add_experimental_option('excludeSwitches', ['enable-automation'])       以键值对的形式加入参数（zim 验证）
        add_experimental_option("debuggerAddress", "127.0.0.1:9999")            连接到已经打开的浏览器
desired_capabilities = option.to_capabilities()    webdriver.DesiredCapabilities.CHROME.copy()    配置代理    //
print(self.driver.page_source)  查看代理生效
desired_capabilities['proxy'] = {
    "httpProxy":"199.2.2.1:4455",
    "noProxy":None,
    "proxyType":"MANUAL",
    "class":"org.openqa.selenium.Proxy",
    "autodetect":False
}
desired_capabilities["userAgent"] = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
driver= webdriver.Chrome(executable_path="C://chromedriver", options=option,  desired_capabilities=desired_capabilities)
返回 driver
        execute_cdp_cmd('Page.addScriptToEvaluateOnNewDocument', {              去掉 navigator 验证
            'source': 'Object.defineProperty(navigator, "webdriver", {get: () => undefined})'  # zim 验证
          })
        set_window_rect(900,50,1000,800)   窗口位置和宽高   x, y,   width height
driver = webdriver.Remote("command_executor=self.sessionUrl")    控制远程已有浏览器
```

使用——

```
from selenium.webdriver.common.by import By
from selenium.webdriver.common.proxy import Proxy
from selenium.webdriver.common.proxy import ProxyType
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
from selenium.webdriver import ActionChains

driver = webdriver.Remote("command_executor=self.sessionUrl", desired_capabilities)
    current_window_handle       跳转前 获取当前窗口句柄
    command_executor._url       会话地址（回随机切换端口）
    session_id                  会话 id
    get("http://www.baidu.com")     打开网站
    refresh()                       刷新页面
    implicitly_wait(40)             5 秒钟内找到元素就往下执行，否则抛出异常；（全局性）
    switch_to. default_content()    回到主页面
    set_window_size(1440, 900)      设置窗口大小
```

set_window_rect(22,33, 1440, 900)　　设置窗口 xy 坐标和高 宽

set_page_load_timeout(3)　　页面打开超时时间

switch_to.frame(frameElement)　　定位到 iframe 元素上

switch_to.alert()　　获取弹出对话框

save_screenshot('capture.png')　#全屏截图

execute_script("return var a=arguments[0]",999)　　执行 js 代码，通过 return 获取返回值

text()　　获取对话框文本值

accept()　　相当于点击“确认”

dismiss()　　相当于点击“取消”

send_keys()　　输入值（alert 和 confirm 没有输入对话框，所以就不用能用了，只能使用在 prompt 里）

find_elements(By.TAG_NAME, "input")

find_element_by_css_selector('iframe')[1]　　通过 css 选择器选择出元素（选出的元素可以继续调用 选择器方法）

find_elements_by_class_name　　选出 多个元素

find_elements_by_tag_name

find_elements_by_id

find_elements_by_link_text

find_elements_by_partial_link_text

find_elements_by_xpath('//*[@id="recaptcha-anchor"]/div[1]')　　或者 .//*[@href and @lmv]

find_elements_by_xpath ('//div[@id="content" and @id="ul"]/ul[@id="ul"]/li/text()')　　虚拟路径　使用“@标签属性”获取 a 便签的 href 属性值

//table[2]　　当前目录第二个 table

//div[contains(@style,"xxx")][@type!="submit"]')　　属性 style 包含 xxx　type 不等于 submit

//a[last()-1]　　倒数第二个

./preceding-sibling::td[2]（当前节点之前的节点）或者 following-sibling（当前节点之后的节点）

//div[@class='el-tab-pane' and not(contains(@style,'none'))]//button[./span[text()='确定']]　　不包含

//tr[not(@id) and not(@class)]　　不包含属性

./ div[@class='el-tab-pane' and not(contains(@style,'none'))]

click()　　点击元素

screenshot('ele.png')　#元素截图

send_keys("123")　　用于在一个输入框内输入 XX 内容

clear()　　清空输入框

get_attribute("src");　获取属性值

send_keys(Keys.CONTROL, "a")

send_keys(Keys.DELETE)

actions =ActionChains(driver);

moveToElement(element).click().double_click().perform();　　执行链条

action.key_down(Keys.CONTROL).send_keys('a').key_up(Keys.CONTROL).perform() # ctrl+a

actions.move_to_element(originInputEle).key_down(Keys.CONTROL).send_keys('a').key_up(Keys.CONTROL).send_keys(self.sourceReadData[self.currentDataIndex][0]).perform()

move_to_element(to_element) ——鼠标移动到某个元素

其他——

跳转到元素视区 方法 2

JavascriptExecutor jse = (JavascriptExecutor)driver;

jse.executeScript("arguments[0].scrollIntoView()", Webelement);

跳转后获取新句柄

```
all_window=driver.window_handles
for window in all_window:
    if window != current_window:
        driver.switch_to.window(window)
current_window = firefox_login.current_window_handle     # 获取当前窗口 handle name
browser.close()      # 关闭当前窗口 B
driver.quit()          退出 driver
```

使用上一个会话，再次 get 不会重新打开浏览器

```
driver2 =  webdriver.Remote(command_executor=executor_url, desired_capabilities={})
driver2.session_id = session_id
self.driver=driver2
self.execTimes=self.execTimes+1
```

修改 textarea 内容（通过 js）

```
js = 'var ucode = document.getElementById("txarea_serial"); ucode.value=arguments[0]'
driver.execute_script(js,'123\t456\n789')
```

tensorflow

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os
import numpy as np
import matplotlib.pyplot as plt
```

1. 添加路径

```
train_dir = os.path.join(PATH, 'train')                    训练路径    //EX：C:\train
validation_dir = os.path.join(PATH, 'validation')      验证路径    //EX：C:\validation

train_cats_dir = os.path.join(train_dir, 'cats')            猫训练  路径    //EX：C:\validation
train_dogs_dir = os.path.join(train_dir, 'dogs')           狗训练 路径    //EX：C:\validation
validation_cats_dir = os.path.join(validation_dir, 'cats')         猫验证  路径    //EX：C:\validation
validation_dogs_dir = os.path.join(validation_dir, 'dogs')     狗验证  路径    //EX：C:\validation
```

2. 查看图片数量

```
num_cats_tr = len(os.listdir(train_cats_dir))            猫训练 图片个数
num_dogs_tr = len(os.listdir(train_dogs_dir))        狗训练  图片个数

num_cats_val = len(os.listdir(validation_cats_dir))      猫验证 图片个数
num_dogs_val = len(os.listdir(validation_dogs_dir))    狗验证  图片个数

total_train = num_cats_tr + num_dogs_tr            总训练张数
total_val = num_cats_val + num_dogs_val          总验证张数
```

```
print('total training cat images:', num_cats_tr)
print('total training dog images:', num_dogs_tr)

print('total validation cat images:', num_cats_val)
print('total validation dog images:', num_dogs_val)
print("--")
print("Total training images:", total_train)
print("Total validation images:", total_val)

3. 使用 ImageDataGenerator 处理数据
batch_size = 128
epochs = 15
IMG_HEIGHT = 125
IMG_WIDTH = 125
train_image_generator = ImageDataGenerator(rescale=1./255)
```

matplotlib
numpy

hstack()　　平铺合并水平方向的数组
vstack()　　在竖直方向上堆叠

apscheduler

```
from apscheduler.schedulers.blocking import BlockingScheduler
from apscheduler.schedulers.background import BackgroundScheduler
from apscheduler.jobstores.sqlalchemy import SQLAlchemyJobStore
import pymysql
pymysql.install_as_MySQLdb()
```

BlockingScheduler　　调用 start 函数后会阻塞当前线程。当调度器是你应用中唯一要运行的东西时(如上例)使用。
BackgroundScheduler(timezone='Asia/Shanghai')　　调用 start 后主线程不会阻塞。当你不运行任何其他框架时使用，并希望调度器在你应用的后台执行。

执行器

执行器的选择取决于应用场景。通常默认的 ThreadPoolExecutor 已经在大部分情况下是可以满足我们需求的。
如果我们的任务涉及到一些 CPU 密集计算的操作。那么应该考虑 ProcessPoolExecutor。然后针对每种程序， apscheduler 也设置了不同的 executor：

- ThreadPoolExecutor：线程池执行器。
- ProcessPoolExecutor：进程池执行器。
- GeventExecutor： Gevent 程序执行器。
- TornadoExecutor： Tornado 程序执行器。
- TwistedExecutor： Twisted 程序执行器。
- AsyncIOExecutor： asyncio 程序执行器。

任务存储

任务存储器的选择有两种。一是内存，也是默认的配置。二是数据库。
使用内存的方式是简单高效，但是不好的是，一旦程序出现问题，重新运行的话，会把之前已经执行了的任务重新执行一遍。
数据库则可以在程序崩溃后，重新运行可以从之前中断的地方恢复正常运行。有以下几种选择：

MemoryJobStore：没有序列化，任务存储在内存中，增删改查都是在内存中完成。
SQLAlchemyJobStore：使用 SQLAlchemy 这个 ORM 框架作为存储方式。
MongoDBJobStore：使用 mongodb 作为存储器。
RedisJobStore：使用 redis 作为存储器。

redis：

```
second_redis_jobstore = RedisJobStore(
    db=2,
    jobs_key="apschedulers.second_jobs",
    run_times_key="apschedulers.second_run_times",
    host="127.0.0.1",
    port=6379,
    password="test"
)

scheduler.add_jobstore(second_redis_jobstore, 'second')
```

mysql：

```
url="mysql+pymysql://user:passwd@host/dbname?charset=utf8"
job.scheduler.add_jobstore(jobstore="sqlalchemy",url=url,tablename='api_job')
```

sqlite：

```
jobstores = {
    'mongo': MongoDBJobStore(),
    'default': SQLAlchemyJobStore(url='sqlite:///jobs.sqlite')
}
executors = {
    'default': ThreadPoolExecutor(20),
    'processpool': ProcessPoolExecutor(5)
}
job_defaults = {
    'coalesce': False,
    'max_instances': 3
}
scheduler = BackgroundScheduler(jobstores=jobstores, executors=executors, job_defaults=job_defaults,
timezone=utc)
```

任务启动

```
scheduler.add_job(
    func=xxfunc,                    执行的函数地址
    trigger="interval"
    name="ROUTEPRICE"               线程名
    id='xxjobid'                    任务名

    seconds=20                      20s 执行一次（可以和其他参数叠加 minutes 等）
    minutes = 19                    19m 执行一次
    hours = 17                      17h 执行一次
    days = 3d                       3d 执行一次

    misfire_grace_time = 20         超过用户设定的时间范围外 20s 时，该任务依旧执行，超出这个时间不执行(单位时间 s)。
```

```
        coalesce = True                 进程挂掉时，导致任务多次没有调用，则前几次的累计任务的任务是否执行的策略。
        max_instances=3                 同一个任务在线程池中最多跑的线程实例数（3 个线程 同时定时执行同一任务）
        next_run_time=datetime.datetime.now()       立刻执行
    __getstate__  获取 job 状态
scheduler.start()
scheduler.remove_job('xxjobid')     移除任务
scheduler.pause_job(job_id,jobstore=None)。         暂停任务
scheduler.resume_job(job_id,jobstore=None)。        恢复任务：
scheduler.modify_job(job_id,jobstore=None,**changes)。                  修改某个任务属性信息
scheduler.reschedule_job(job_id,jobstore=None,trigger=None,**trigger_args)  修改单个作业的触发器并更新下次运行时间：
scheduler.print_jobs(jobstore=None,out=sys.stdout)                      输出作业信息
scheduler.get_job('xxjobid')  打印 job 信息，没有返回 None
scheduler.get_jobs()            获取所有 job
scheduler.print_jobs()          打印所有的 job 信息
scheduler.add_listener(SDK.taskListener,EVENT_JOB_EXECUTED | EVENT_JOB_ERROR | EVENT_JOB_MISSED)   监听执行和执行失败事件
```

数据库

pymysql

```
import pymysql
import re

connection = pymysql.connect(           连接数据库
    host="localhost",
    port=3306,
    database="medicine",
    user="root",
    password="root",
    charset="utf8"
)
datamodel = connection.cursor()             获取数据库模型

sql = "INSERT INTO homework(id, title ) VALUES( %d, '%s' );" \
      % (1, pymysql.escape_string( " test " ) )     插入字符串需要提前处理
datamodel.execute(sql)                          执行 sql 语句
datamodel.fetchall()            返回结果数据
connection.commit()             提交修改
datamodel.close()               关闭数据库模型
connection.close()

for (row,) in rows:
    print(row)
        curs.execute("insert into user (name, age) values (%s, %s)", ("Marsen", '26'))
                last_id = curs.lastrowid
db.insert_id()
```

只有在 user 表的主键是自增 id 的时候，而且在执行的 INSERT sql 语句中不能去自己去指定 id, 才能使用 curs.lastrowid 来获取新插入数据的 id。否则获取到的 id 都为 0。

SQLalchemy

数据库 ORM(Object Relational Mapping) 对象-关系映射

pip install PyMySQL

```
import pymysql
pymysql.install_as_MySQLdb()                使用 pythone3 的 PyMsql
from sqlalchemy import create_engine
```

连接数据库

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from urllib import parse
password = parse.quote_plus(password)    解决密码包含特殊字符

engine = create_engine("mysql://user:password@hostname:3306/dbname?charset=utf8",
                       echo=True,              #当设置为 True 时会将 orm 语句转化为 sql 语句打印，一般 debug 的时候可用
                       pool_size=8,            # 连接池的大小，默认为 5 个，设置为 0 时表示连接无限制
                       pool_recycle=60*30    #设置时间以限制数据库多久没连接自动断开
                       )
dbengine.dispose()  关闭
创建 session:
    DbSession = sessionmaker(bind=engine)
    session = DbSession()
```

创建实体

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.types import CHAR, Integer, String
from sqlalchemy import Column
Base = declarative_base()

class Person(Base):
   __tablename__ = "users"
   id = Column(Integer, primary_key=True)
   name = Column(String(64), unique=True)
   email = Column(String(64))

   def __init__(self, name, email):
      self.name = name
      self.email = email
   def drop_db():
      BaseModel.metadata.drop_all(engine)      #   删除数据表
   def create_db():
      BaseModel.metadata.create_all(engine)    #    创建数据库表
   def to_dict(self):
      return {c.name: getattr(self, c.name, None) for c in self.__table__.columns}
```

外键——

```
class Parent(Base):
    __tablename__ = 'parent'
    id = Column(Integer, primary_key=True)
    children = relationship("Child", back_populates="parent")        Parent.children 是指的一个 Child 实例列表。

class Child(Base):
    __tablename__ = 'child'
    id = Column(Integer, primary_key=True)
    parent_id = Column(Integer, ForeignKey('parent.id'))                当前外键关联到表'parent
    parent = relationship("Parent", back_populates="children")       Child.parent 是指一个 Parent
```

使用——

查询：

```
    users = session.query(Users).filter_by(id=1).all()
    for item in users:
        print(item.name)
    q = session.query(User).filter(User.name.like('e%'))
    sesseion.query(User).filter(or_(User.name == 'jack', User.name == 'ed')).all()
```

新增：

```
    add_user = Users("test", "test123@qq.com")
    session.add(add_user)         # 把 Model 加入当前 session 维护的持久空间(可以从 session.dirty 看到)中，直到 commit 时提交到数据库
    session.flush()                  # 这样便可在 session 中 get 到对象的属性 // user_id=add_user.id
    session.commit()
```

删除：

```
    session.query(Users).filter(Users.name == "test").delete()
    session.commit()
```

更新：

```
    session.query(Users).filter_by(id=1).update({'name': "Jack"})

    user = session.query(Users).filter_by(name="Jack").first()
    user.name = "test"
    session.add(user)
```

工具包

request_html

```
r.html.absolute_links   获取链接
r.html.links
   jobs.text              获取文本
   jobs.full_text
attrs = jobs.attrs     获取属性
attrs.get("key")
r.html.find('div#menu', first=True).text     查找 css 返回结果数组   【find，search 返回的都是封装 Element 元素，只有 html 是标签元素】
```

selector ，要用的 CSS 选择器；
clean， 布尔值，如果为真会忽略 HTML 中 style 和 script 标签造成的影响（原文是 sanitize，大概这么理解）；
containing，如果设置该属性，会返回包含该属性文本的标签；
first， 布尔值，如果为真会返回第一个元素，否则会返回满足条件的元素列表；
_encoding，编码格式。
attrs={"class": 'post_summary'}
r.html.search('把{}夹')[0] # 获取从 "把" 到 "夹" 字的所有内容

pipenv

包管理工具

parse

parse("The {} who {} {}", "The knights who say Ni!") 匹配内容 <Result ('knights', 'say', 'Ni!') {}>

pillow

from PIL import Image 用于打开图片和对图片处理
import io

Image.open(r"D:\a.png") 返回图片对象
size 获取图片大小 //EX：w, h =img.size
getpixel((x,y)) 得到某个位置的像素，对应从左上角开始的宽高 x，y //EX：r,g,b = img.getpixel((x,y))
convert("L") 【1 为二值图像，非黑即白。但是它每个像素用 8 个 bit 表示，0 表示黑，255 表示白】
【 L 为灰色图像，它的每个像素用 8 个 bit 表示，0 表示黑，255 表示白，其他数字表示不同的灰度。】
在 PIL 中，从模式“RGB”转换为“L”模式是按照下面的公式转换的： L = R * 299/1000 + G * 587/1000+ B * 114/1000】
【P，RGB，RGBA，CMYK，YCbCr,I，F】
show() 显示图片
thumbnail((width/10, height/10))
resize((100,100) , Image.ANTIALIAS) 重新缩放大小，返回新图片
save(xxximgByteArr | xxpath, format='JPEG', quality=95) Image 格式转为 bytes 字节流格式，或保存到路径
【quality 参数： 保存图像的质量，值的范围从 1 到 95。 默认值为 75，使用中应尽量避免高于 95 的值; 100 会禁用部分 JPEG 压缩算法，并导致大文件图像质量几乎没有任何增益。】
// imgByteArr = io.BytesIO() save(imgByteArr)
imgByteArr = imgByteArr.getvalue()

opencv-python

import cv2 as cv 打开图片和图像处理

pytesseract

图片转文字（tesseract 添加环境变量 TESSDATA_PREFIX）
pytesseract.pytesseract.tesseract_cmd = tesseractPath # 设置 pyteseract 路径

urllib3

from urillib3 import request
request.urlopen()
read() readline() ,readlines() , fileno() , close() 对 HTTPResponse 类型数据进行操作
info() 返回 HTTPMessage 对象，表示远程服务器返回的头信息
getcode() 返回 Http 状态码。如果是 http 请求，200 请求成功完成;404 网址未找到
geturl() 返回请求的 url

demjson

demjson.encode(self, obj, nest_level=0) 对象转 j 串

demjson.encode(data)　　json　转 j 串

demjson.decode(self, txt)　　j 串　　转 json

numpy

import numpy as np　读取二进制图片

nparr = np.asarray(bytearray(image_bytes), dtype="uint8")　　数组转换为 np 数组

image = nparr.reshape((960, 540)) # (height, width)

im = Image.fromarray(image, mode="L")

opencv-python

pred_img = cv2.resize(nparr,(28,28))

xlsxwriter

worksheet.set_row()

cell_format = workbook.add_format({'bold': True})

set_row(row, height, ceel_format, options)

xlrd 1.2.0

```
rbXlsx=xlrd.open_workbook(writeFile)
wbXlsx=copy(rbXlsx)
wbXlsx.write(1,3,"some text")
os.remove(writeFile)
wbXlsx.save(writeFile)
        cell = sheetFile.cell(rowInd, newNameInd)
        targetStr=cell.value
        if cell.ctype == 2 and cell.value % 1 == 0:
            targetStr = int(cell.value)
            print('[EXCEL UTIL] float data',targetStr)
```

xlwt

xlutils

xlutils.copy(xlsxObj)　返回一个打开的 xlsx　　// xlrd.open_workbook("path")

openpyxl

openpyxl：对 excel 文件的打开、读写、编辑、保存相关

pandas

```
    data = pandas.read_csv(numPath,encoding='gb18030')
     # 必须添加 header=None，否则默认把第一行数据处理成列名导致缺失
     list = data.values.tolist()
```

pytz

pytz：常用于时区的转换

baidu-aip

aip：百度 ocr 识别文字

poplib

```
from email.parser import Parser
from email.header import decode_header
from email.utils import parseaddr
import poplib

email = 'saidake@qq.com'　　# 输入邮件地址, 口令和 POP3 服务器地址:
password = 'quatvbcmlzymcabi'　# 这个密码不是邮箱登录密码，是 pop3 服务密码
pop3_server = 'pop.qq.com'
```

```
def guess_charset(msg):
    charset = msg.get_charset()
    if charset is None:
        content_type = msg.get('Content-Type', '').lower()
        pos = content_type.find('charset=')
        if pos >= 0:
            charset = content_type[pos + 8:].strip()
    return charset

def decode_str(s):
    value, charset = decode_header(s)[0]
    if charset:
        value = value.decode(charset)
    return value

def print_info(msg, indent=0):
    if indent == 0:
        for header in ['From', 'To', 'Subject']:
            value = msg.get(header, '')
            if value:
                if header=='Subject':
                    value = decode_str(value)
                else:
                    hdr, addr = parseaddr(value)
                    name = decode_str(hdr)
                    value = u'%s <%s>' % (name, addr)
            print('%s%s: %s' % ('  ' * indent, header, value))
    if (msg.is_multipart()):
        parts = msg.get_payload()
        for n, part in enumerate(parts):
            print('%spart %s' % ('  ' * indent, n))
            print('%s--------------------' % ('  ' * indent))
            print_info(part, indent + 1)
    else:
        content_type = msg.get_content_type()
        if content_type=='text/plain' or content_type=='text/html':
            content = msg.get_payload(decode=True)
            charset = guess_charset(msg)
            if charset:
                content = content.decode(charset)
            print('%sText: %s' % ('  ' * indent, content + '...'))
        else:
            print('%sAttachment: %s' % ('  ' * indent, content_type))

# 连接到 POP3 服务器:
server = poplib.POP3_SSL(pop3_server, 995)
# 可以打开或关闭调试信息:
```

```
server.set_debuglevel(1)
# 可选:打印 POP3 服务器的欢迎文字:
print(server.getwelcome().decode('utf-8'))
# 身份认证:
server.user(email)
server.pass_(password)
# stat()返回邮件数量和占用空间:
print('Messages: %s. Size: %s' % server.stat())
# list()返回所有邮件的编号:
resp, mails, octets = server.list()
# 可以查看返回的列表类似[b'1 82923', b'2 2184', ...]
print(mails)
# 获取最新一封邮件, 注意索引号从 1 开始:
index = len(mails)
print('未读邮件的数量',index)
resp, lines, octets = server.retr(index)
# lines 存储了邮件的原始文本的每一行,
# 可以获得整个邮件的原始文本:
msg_content = b'\r\n'.join(lines).decode('utf-8')
# 稍后解析出邮件:
msg = Parser().parsestr(msg_content)
msg.get_payload(decode=True)        获取邮件体
print_info(msg)
# 可以根据邮件索引号直接从服务器删除邮件:
# server.dele(2)
# 关闭连接:
server.quit()
```

scrapy

命令

```
scrapy startproject  mypro                          创建项目
scrapy genspider myspider www.baidu.com    创建爬虫
scrapy list       查看项目有几个爬虫
```

settings.py

```
LOG_LEVEL = "DEBUG"  # 输出级别
LOG_STDOUT = true  # 是否标准输出
            CRITICAL -- 关键错误
            ERROR -- 一般级别的错误
            WARNING -- 警告信息
            INFO -- 信息消息的日志（建议生产模式使用）
            DEBUG -- 调试消息的日志（建议开发模式）
ITEM_PIPELINES = {
    'heartsong.pipelines.HeartsongPipeline': 300,          注册管道并定义优先级
}
```

one.py

```
import scrapy
from ..items import OneStatusItem
```

```
class OneSpider(scrapy.Spider):
    name = 'one'
    allowed_domains = ['www.baidu.com']
    start_urls = ['http://www.baidu.com/']
    def start_requests(self):
        request = scrapy.Request( url, method='POST',
                    body=json.dumps(my_data),
                  headers={'Content-Type':'application/json'} )
    def parse(self, response):          处理相应
            response.url
            status
            headers
            body
            request
            meta
            flags
            urljoin(url)
            text
            encoding
            selector
            xpath
            css
            body_as_unicode
```

main.py

```
from scrapy.crawler import CrawlerProcess
from scrapy.utils.project import get_project_settings

# 根据项目配置获取 CrawlerProcess 实例
process = CrawlerProcess(get_project_settings())

# 添加需要执行的爬虫
process.crawl('one')
# process.crawl('dining')
# process.crawl('experience')

# 执行
process.start()
```

python main.py

mitmdump

http://mitm.it/ 检测请求是否通过了 mitmproxy

rich

Python / adb

```
adb devices                          查看连接设备
adb shell pm list packages           显示所有包名
adb shell dumpsys activity activities                显示活动程序
```

adb shell input tap 10 10 点击

adb shell input swipe x_start y_start x_end y_end 滑动

adb shell input text xxx 输入文字信息

adb shell input keyevent X 3 对应的是 HOME 键 24 对应的是音量+ 25 对应的是音量- 66 对应的是确认键

adb shell getevent 监听手机事件 0003 0035 xx 0003 0036 yy

adb shell screencap -p /sdcard/autolottery.png ADB 截取屏幕

adb pull /sdcard/autolottery.png ./img 第一个路径是手机中文件的路径和文件名，后一个路径是存放在电脑中的路径（./img 表示存在当前 py 文件目录下的 img 文件夹里）