# Day09回顾

## settings.py常用变量

```
【1】settings.py中常用变量
   2.1) 设置日志级别
        LOG_LEVEL = ''
   2.2) 保存到日志文件(不在终端输出)
        LOG_FILE = ''
   2.3) 设置数据导出编码(主要针对于json文件)
        FEED_EXPORT_ENCODING = 'utf-8'
   2.4) 设置User-Agent
        USER_AGENT = ''
   2.5) 设置最大并发数(默认为16)
        CONCURRENT_REQUESTS = 32
   2.6) 下载延迟时间(每隔多长时间请求一个网页)
        DOWNLOAD_DELAY = 1
   2.7) 请求头
        DEFAULT_REQUEST_HEADERS = {'User-Agent':'Mozilla/'}
   2.8) 添加项目管道
        ITEM_PIPELINES = {'项目目录名.pipelines.类名':优先级}
   2.9) cookie(默认禁用,取消注释-True|False都为开启)
        COOKIES_ENABLED = False
   2.10) 非结构化数据存储路径
        IMAGES_STORE = '/home/tarena/images/'
        FILES_STORE = '/home/tarena/files/'
   2.11) 添加下载器中间件
       DOWNLOADER_MIDDLEWARES = {'项目名.middlewares.类名':200}
```

## 非结构化数据抓取

```
【1】spider
   yield item['链接']

【2】pipelines.py
   from scrapy.pipelines.images import ImagesPipeline
   import scrapy
   class TestPipeline(ImagesPipeline):
      def get_media_requests(self,item,info):
           yield scrapy.Request(url=item['url'],meta={'name':item['name']})

      def file_path(self,request,response=None,info=None):
           name = request.meta['name']
```

```
            filename = name
            return filename

【3】settings.py
    IMAGES_STORE = 'D:/Spider/images'
```

# Post请求

- **方法**

```
scrapy.FormRequest(url=url,formdata=formdata,callback=self.xxx)
```

- **使用cookie**

```
【1】方法1
    COOKIES_ENABLED = False
    DEFAULT_REQUEST_HEADERS = {'Cookie':'xxxx'}

【2】方法2
    COOKIES_ENABLED = True
    yield scrapy.Request(url=url,cookies={},callback=self.xxxx)
    yield scrapy.FormRequest(url=url,formdata={},cookies={},callback=self.xxxx)

【3】方法3
    COOKIES_ENBALED = True
    class XxxCookieDownloaderMiddleware(object):
      def process_request(self,request,spider):
        request.cookies = {}
```

- **对象属性**

```
【1】请求对象request属性
    1.1) request.url     : 请求URL地址
    1.2) request.headers : 请求头 - 字典
    1.3) request.meta    : item数据传递、定义代理
    1.4) request.cookies : Cookie

【2】响应对象response属性
    2.1) response.url      : 返回实际数据的URL地址
    2.2) response.text     : 响应内容 - 字符串
    2.3) response.body     : 响应内容 - 字节串
    2.4) response.encoding : 响应字符编码
    2.5) response.status   : HTTP响应码
```

# Day10笔记

# 中间件

## 设置中间件(随机User-Agent)

- **少量UA设置 - 不使用中间件**

```
【1】方法一 : settings.py
   1.1) USER_AGENT = ''
   1.2) DEFAULT_REQUEST_HEADERS = {}

【2】方法二 : 爬虫文件
   yield scrapy.Request(url,callback=函数名,headers={})
```

- **大量UA设置 - 使用middlewares.py中间件**

```
【1】获取User-Agent方式
   1.1) 方法1 : 新建useragents.py,存放大量User-Agent, random模块随机切换
   1.2) 方法2 : 使用fake_useragent模块
        from fake_useragent import UserAgent
        agent = UserAgent().random

【2】middlewares.py新建中间件类
   class RandomUseragentMiddleware(object):
        def process_request(self,reuqest,spider):
            agent = UserAgent().random
            request.headers['User-Agent'] = agent

【3】settings.py添加此下载器中间件
   DOWNLOADER_MIDDLEWARES = {'' : 优先级}
```

## 设置中间件(随机代理)

- **代理IP中间件**

```
class RandomProxyDownloaderMiddleware(object):
    def process_request(self,request,spider):
        request.meta['proxy'] = xxx

    # 捕获异常的方法,一旦代理不能用,会被此方法捕获,并重新包装请求再次发送
    def process_exception(self,request,exception,spider):
        return request
```

## 设置中间件(Cookie)

- **Cookie中间件**

```python
class BaiduCookieDownloaderMiddleware(object):
    def process_request(self,request,spider):
        cookies = self.get_cookies()
        print('middleware3', cookies)
        # 利用请求对象request的cookies属性
        request.cookies = cookies


    def get_cookies(self):
        costr = ''
        cookies = {}
        for kv in costr.split('; '):
            cookies[kv.split('=')[0]] =kv.split('=')[1]

        return cookies
```

- **练习**

```
将有道翻译案例的cookie使用中间件的方式来实现
```

# 分布式爬虫

- **分布式爬虫介绍**

```
【1】原理
    多台主机共享1个爬取队列

【2】实现
    2.1) 重写scrapy调度器(scrapy_redis模块)
    2.2) sudo pip3 install scrapy_redis
```

- **为什么使用redis**

```
【1】Redis基于内存,速度快
【2】Redis非关系型数据库,Redis中集合,存储每个request的指纹
```

# scrapy_redis详解

- **GitHub地址**

```
https://github.com/rmax/scrapy-redis
```

- **settings.py说明**

```
# 重新指定调度器: 启用Redis调度存储请求队列
SCHEDULER = "scrapy_redis.scheduler.Scheduler"

# 重新指定去重机制: 确保所有的爬虫通过Redis去重
```

```
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

# 不清除Redis队列: 暂停/恢复/断点续爬(默认清除为False,设置为True不清除)
SCHEDULER_PERSIST = True

# 优先级队列 （默认）
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.PriorityQueue'
#可选用的其它队列
# 先进先出
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.FifoQueue'
# 后进先出
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.LifoQueue'

# redis管道
ITEM_PIPELINES = {
    'scrapy_redis.pipelines.RedisPipeline': 300
}

#指定连接到redis时使用的端口和地址
REDIS_HOST = 'localhost'
REDIS_PORT = 6379
```

# 腾讯招聘分布式改写

- **分布式爬虫完成步骤**

```
【1】首先完成非分布式scrapy爬虫 ： 正常scrapy爬虫项目抓取
【2】设置,部署成为分布式爬虫
```

- **分布式环境说明**

```
【1】分布式爬虫服务器数量: 2 (其中1台Windows,1台Ubuntu虚拟机)
【2】服务器分工:
   2.1) Windows ： 负责数据抓取
   2.2) Ubuntu  ： 负责URL地址统一管理,同时负责数据抓取
```

- **腾讯招聘分布式爬虫 - 数据同时存入1个Redis数据库**

```
【1】完成正常scrapy项目数据抓取（非分布式 - 拷贝之前的Tencent）

【2】设置settings.py，完成分布式设置
   2.1-必须) 使用scrapy_redis的调度器
        SCHEDULER = "scrapy_redis.scheduler.Scheduler"

   2.2-必须) 使用scrapy_redis的去重机制
        DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

   2.3-必须) 定义redis主机地址和端口号
        REDIS_HOST = '192.168.1.107'
        REDIS_PORT = 6379

   2.4-非必须) 是否清除请求指纹,True:不清除 False:清除（默认）
```

```
        SCHEDULER_PERSIST = True

    2.5-非必须) 在ITEM_PIPELINES中添加redis管道,数据将会存入redis数据库
        'scrapy_redis.pipelines.RedisPipeline': 200

【3】把代码原封不动的拷贝到分布式中的其他爬虫服务器,同时开始运行爬虫

【结果】: 多台机器同时抓取,数据会统一存到Ubuntu的redis中, 而且所抓数据不重复
```

- **腾讯招聘分布式爬虫 - 数据存入MySQL数据库**

```
"""和数据存入redis步骤基本一样,只是变更一下管道和MySQL数据库服务器的IP地址"""
【1】settings.py
    1.1) SCHEDULER = 'scrapy_redis.scheduler.Scheduler'
    1.2) DUPEFILTER_CLASS = 'scrapy_redis.dupefilter.RFPDupeFilter'
    1.3) SCHEDULER_PERSIST = True
    1.4) REDIS_HOST = '192.168.1.105'
    1.5) REDIS_POST = 6379
    1.6) ITEM_PIPELINES = {'Tencent.pipelines.TencentMysqlPipeline' : 300}
    1.7) MYSQL_HOST = '192.168.1.105'

【2】将代码拷贝到分布式中所有爬虫服务器

【3】多台爬虫服务器同时运行scrapy爬虫

# 赠送腾讯MySQL数据库建库建表语句
"""
create database tencentdb charset utf8;
use tencentdb;
create table tencenttab(
job_name varchar(1000),
job_type varchar(200),
job_duty varchar(5000),
job_require varchar(5000),
job_address varchar(200),
job_time varchar(200)
)charset=utf8;
"""
```

# 新的篇章

# 腾讯招聘分布式改写之方法二

- **使用redis_key改写（同时存入MySQL数据库）**

```
【1】settings.py和方法一中写法一致
    1.1) SCHEDULER = 'scrapy_redis.scheduler.Scheduler'
    1.2) DUPEFILTER_CLASS = 'scrapy_redis.dupefilter.RFPDupeFilter'
    1.3) SCHEDULER_PERSIST = True
    1.4) REDIS_HOST = '192.168.1.107'
```

```
    1.5) REDIS_PORT = 6379
    1.6) ITEM_PIPELINES = {'Tencent.pipelines.TencentMysqlPipeline' : 300}
    1.7) MYSQL_HOST = '192.168.1.107'

【2】爬虫文件:tencent.py (必须基于start_urls)
    from scrapy_redis.spiders import RedisSpider
    class TencentSpider(RedisSpider):
        # 1. 去掉start_urls
        # 2. 定义redis_key
        redis_key = 'tencent:spider'
        def parse(self,response):
            pass

【3】把代码复制到所有爬虫服务器，并启动项目

【4】到redis命令行，执行LPUSH命令压入第一个要爬取的URL地址
    >LPUSH tencent:spider 第1页的URL地址

【注意】：项目爬取结束后无法退出，如何退出?
setting.py
CLOSESPIDER_TIMEOUT = 3600
# 到指定时间(3600秒)时,会自动结束并退出
```

# 机器视觉与tesseract

- **概述**

```
【1】作用
    处理图形验证码

【2】三个重要概念 - OCR、tesseract-ocr、pytesseract
    2.1) OCR
        光学字符识别(Optical Character Recognition),通过扫描等光学输入方式将各种票据、报刊、书籍、文稿及其它印刷品的文字转化为图像信息，再利用文字识别技术将图像信息转化为电子文本

    2.2) tesseract-ocr
        OCR的一个底层识别库（不是模块，不能导入），由Google维护的开源OCR识别库

    2.3) pytesseract
        Python模块,可调用底层识别库，是对tesseract-ocr做的一层Python API封装
```

- **安装tesseract-ocr**

```
【1】Ubuntu安装
    sudo apt-get install tesseract-ocr

【2】Windows安装
    2.1) 下载安装包
    2.2) 添加到环境变量(Path)

【3】测试 (终端 | cmd命令行)
    tesseract xxx.jpg 文件名
```

- **安装pytesseract**

```
【1】安装
    sudo pip3 install pytesseract

【2】使用示例
    import pytesseract
    # Python图片处理库
    from PIL import Image

    # 创建图片对象
    img = Image.open('test1.jpg')
    # 图片转字符串
    result = pytesseract.image_to_string(img)
    print(result)
```

# 在线打码平台

- **为什么使用在线打码**

```
tesseract-ocr识别率很低,文字变形、干扰，导致无法识别验证码
```

- **云打码平台使用步骤**

```
【1】下载并查看接口文档
【2】调整接口文档，调整代码并接入程序测试
【3】真正接入程序，在线识别后获取结果并使用
```

## *破解云打码网站验证码*

- **1 - 下载并调整接口文档，封装成函数，打码获取结果**

```
import http.client, mimetypes, urllib, json, time, requests

######################################################################

class YDMHttp:

    apiurl = 'http://api.yundama.com/api.php'
    username = ''
    password = ''
    appid = ''
    appkey = ''

    def __init__(self, username, password, appid, appkey):
        self.username = username
        self.password = password
        self.appid = str(appid)
```

```
        self.appkey = appkey

    def request(self, fields, files=[]):
        response = self.post_url(self.apiurl, fields, files)
        response = json.loads(response)
        return response

    def balance(self):
        data = {'method': 'balance', 'username': self.username, 'password': self.password,
'appid': self.appid, 'appkey': self.appkey}
        response = self.request(data)
        if (response):
            if (response['ret'] and response['ret'] < 0):
                return response['ret']
            else:
                return response['balance']
        else:
            return -9001

    def login(self):
        data = {'method': 'login', 'username': self.username, 'password': self.password,
'appid': self.appid, 'appkey': self.appkey}
        response = self.request(data)
        if (response):
            if (response['ret'] and response['ret'] < 0):
                return response['ret']
            else:
                return response['uid']
        else:
            return -9001

    def upload(self, filename, codetype, timeout):
        data = {'method': 'upload', 'username': self.username, 'password': self.password,
'appid': self.appid, 'appkey': self.appkey, 'codetype': str(codetype), 'timeout':
str(timeout)}
        file = {'file': filename}
        response = self.request(data, file)
        if (response):
            if (response['ret'] and response['ret'] < 0):
                return response['ret']
            else:
                return response['cid']
        else:
            return -9001

    def result(self, cid):
        data = {'method': 'result', 'username': self.username, 'password': self.password,
'appid': self.appid, 'appkey': self.appkey, 'cid': str(cid)}
        response = self.request(data)
        return response and response['text'] or ''

    def decode(self, filename, codetype, timeout):
        cid = self.upload(filename, codetype, timeout)
        if (cid > 0):
            for i in range(0, timeout):
                result = self.result(cid)
                if (result != ''):
```

```
                    return cid, result
                else:
                    time.sleep(1)
            return -3003, ''
        else:
            return cid, ''

    def report(self, cid):
        data = {'method': 'report', 'username': self.username, 'password': self.password, 'appid': self.appid, 'appkey': self.appkey, 'cid': str(cid), 'flag': '0'}
        response = self.request(data)
        if (response):
            return response['ret']
        else:
            return -9001

    def post_url(self, url, fields, files=[]):
        for key in files:
            files[key] = open(files[key], 'rb');
        res = requests.post(url, files=files, data=fields)
        return res.text

######################################################################
def get_result(filename):
    # 用户名
    username    = 'yibeizi001'
    # 密码
    password    = 'zhishouzhetian001'
    # 软件ＩＤ，开发者分成必要参数。登录开发者后台【我的软件】获得!
    appid       = 1
    # 软件密钥，开发者分成必要参数。登录开发者后台【我的软件】获得!
    appkey      = '22cc5376925e9387a23cf797cb9ba745'
    # 验证码类型，# 例：1004表示4位字母数字，不同类型收费不同。请准确填写，否则影响识别率。在此查询所有类型 http://www.yundama.com/price.html
    codetype    = 5000
    # 超时时间，秒
    timeout     = 60
    # 初始化
    yundama = YDMHttp(username, password, appid, appkey)
    # 登陆云打码
    uid = yundama.login()
    # 查询余额
    balance = yundama.balance()
    # 开始识别，图片路径，验证码类型ID，超时时间（秒），识别结果
    cid, result = yundama.decode(filename, codetype, timeout)

    return result
######################################################################
```

- **2 - 访问云打码网站，获取验证码并在线识别**

```
'''识别云打码官网的验证码'''
from selenium import webdriver
from ydmapi import *
from PIL import Image

```

```python
class YdmSpider(object):
    def __init__(self):
        self.browser = webdriver.Chrome()
        self.browser.maximize_window()
        self.browser.get('http://www.yundama.com/')

    # 获取验证码图片截取出来
    def get_image(self):
        # 1.获取页面截图
        self.browser.save_screenshot('index.png')
        # 2.获取验证码节点坐标,把图片截取出来
        # location: 获取节点左上角的坐标(x y)
        location = self.browser.find_element_by_xpath('//*[@id="verifyImg"]').location
        # size: 获取节点的大小(宽度和高度)
        size = self.browser.find_element_by_xpath('//*[@id="verifyImg"]').size
        # 四个坐标
        left_x = location['x']
        left_y = location['y']
        right_x = left_x + size['width']
        right_y = left_y + size['height']
        # 从index.png中截图图片,注意crop()方法参数为元组
        img = Image.open('index.png').crop((left_x,left_y,right_x,right_y))
        img.save('verify.png')

    # 获取识别结果
    def get_result(self):
        result = get_result('verify.png')
        print('识别结果:',result)

    # 入口函数
    def run(self):
        self.get_image()
        self.get_result()
        self.browser.close()

if __name__ == '__main__':
    spider = YdmSpider()
    spider.run()
```

# Fiddler抓包工具

- **配置Fiddler**

```
【1】Tools -> Options -> HTTPS
   1.1) 添加证书信任:  勾选 Decrypt Https Traffic 后弹出窗口，一路确认
   1.2) 设置之抓浏览器的包:  ...from browsers only

【2】Tools -> Options -> Connections
   2.1) 设置监听端口（默认为8888）

【3】配置完成后重启Fiddler（重要）
   3.1) 关闭Fiddler,再打开Fiddler
```

- **配置浏览器代理**

```
【1】安装Proxy SwitchyOmega谷歌浏览器插件

【2】配置代理
    2.1) 点击浏览器右上角插件SwitchyOmega -> 选项 -> 新建情景模式 -> myproxy(名字) -> 创建
    2.2) 输入  HTTP://  127.0.0.1  8888
    2.3) 点击 : 应用选项

【3】点击右上角SwitchyOmega可切换代理

【注意】: 一旦切换了自己创建的代理,则必须要打开Fiddler才可以上网
```

- **Fiddler常用菜单**

```
【1】Inspector : 查看数据包详细内容
    1.1) 整体分为请求和响应两部分

【2】Inspector常用菜单
    2.1) Headers : 请求头信息
    2.2) WebForms: POST请求Form表单数据 : <body>
                   GET请求查询参数: <QueryString>
    2.3) Raw : 将整个请求显示为纯文本
```

# 移动端app数据抓取

- **方法1 - 手机 + Fiddler**

```
设置方法见文件夹 - 移动端抓包配置
```

- **方法2 - F12浏览器工具**

## *有道翻译手机版破解案例*

```python
import requests
from lxml import etree

word = input('请输入要翻译的单词:')

post_url = 'http://m.youdao.com/translate'
post_data = {
  'inputtext':word,
  'type':'AUTO'
}

html = requests.post(url=post_url,data=post_data).text
parse_html = etree.HTML(html)
xpath_bds = '//ul[@id="translateResult"]/li/text()'
```

```
result = parse_html.xpath(xpath_bds)[0]

print(result)
```

# 补充 - 滑块缺口验证码案例

## *豆瓣网登录*

- **案例说明**

```
【1】URL地址：https://www.douban.com/
【2】先输入几次错误的密码，让登录出现滑块缺口验证，以便于我们破解
【3】模拟人的行为
    3.1) 先快速滑动
    3.2) 到离重点位置不远的地方开始减速
【4】详细看代码注释
```

- **代码实现**

```
"""
说明：先输入几次错误的密码，出现滑块缺口验证码
"""
from selenium import webdriver
# 导入鼠标事件类
from selenium.webdriver import ActionChains
import time

# 加速度函数
def get_tracks(distance):
    """
    拿到移动轨迹，模仿人的滑动行为，先匀加速后匀减速
    匀变速运动基本公式：
    ①v=v0+at
    ②s=v0t+½at²
    """
    # 初速度
    v = 0
    # 单位时间为0.3s来统计轨迹，轨迹即0.3内的位移
    t = 0.3
    # 位置/轨迹列表,列表内的一个元素代表0.3s的位移
    tracks = []
    # 当前的位移
    current = 0
    # 到达mid值开始减速
    mid = distance*4/5
    while current < distance:
        if current < mid:
            # 加速度越小,单位时间内的位移越小,模拟的轨迹就越多越详细
            a = 2
        else:
            a = -3

```

```
        # 初速度
        v0 = v
        # 0.3秒内的位移
        s = v0*t+0.5*a*(t**2)
        # 当前的位置
        current += s
        # 添加到轨迹列表
        tracks.append(round(s))
        # 速度已经达到v，该速度作为下次的初速度
        v = v0 + a*t
    return tracks
    # tracks: [第一个0.3秒的移动距离,第二个0.3秒的移动距离,...]


# 1、打开豆瓣官网 - 并将窗口最大化
options = webdriver.ChromeOptions()
options.add_argument('--start-maximized')
browser = webdriver.Chrome(options=options)
browser.get('https://www.douban.com/')

# 2、切换到iframe子页面
login_frame = browser.find_element_by_xpath('//*[@id="anony-reg-new"]/div/div[1]/iframe')
browser.switch_to.frame(login_frame)

# 3、密码登录 + 用户名 + 密码 + 登录豆瓣
browser.find_element_by_xpath('/html/body/div[1]/div[1]/ul[1]/li[2]').click()
browser.find_element_by_xpath('//*[@id="username"]').send_keys('自己的用户名')
browser.find_element_by_xpath('//*[@id="password"]').send_keys('自己的密码')
browser.find_element_by_xpath('/html/body/div[1]/div[2]/div[1]/div[5]/a').click()
time.sleep(4)

# 4、切换到新的iframe子页面 - 滑块验证
auth_frame = browser.find_element_by_xpath('//*[@id="TCaptcha"]/iframe')
browser.switch_to.frame(auth_frame)

# 5、按住开始滑动位置按钮 - 先移动180个像素
element = browser.find_element_by_xpath('//*[@id="tcaptcha_drag_button"]')
# click_and_hold(): 按住某个节点并保持
ActionChains(browser).click_and_hold(on_element=element).perform()
# move_to_element_with_offset(): 移动到距离某个元素(左上角坐标)多少距离的位置
ActionChains(browser).move_to_element_with_offset(to_element=element,xoffset=180,yoffset=0)
.perform()

# 6、使用加速度函数移动剩下的距离
tracks = get_tracks(28)
for track in tracks:
    # move_by_offset() : 鼠标从当前位置移动到某个坐标
    ActionChains(browser).move_by_offset(xoffset=track,yoffset=0).perform()

# 7、延迟释放鼠标: release()
time.sleep(0.5)
ActionChains(browser).release().perform()
```

# 爬虫总结

```
# 1、什么是爬虫
  爬虫是请求网站并提取数据的自动化程序

# 2、robots协议是什么
  爬虫协议或机器人协议,网站通过robots协议告诉搜索引擎哪些页面可以抓取，哪些页面不能抓取

# 3、爬虫的基本流程
  1、请求得到响应
  2、解析
  3、保存数据

# 4、请求
  1、urllib
  2、requests
  3、scrapy

# 5、解析
  1、re正则表达式
  2、lxml+xpath解析
  3、json解析模块

# 6、selenium+browser

# 7、常见反爬策略
  1、Headers ： 最基本的反爬手段，一般被关注的变量是UserAgent和Referer，可以考虑使用浏览器中
  2、UA ： 建立User-Agent池,每次访问页面随机切换
  3、拉黑高频访问IP
     数据量大用代理IP池伪装成多个访问者,也可控制爬取速度
  4、Cookies
     建立有效的cookie池，每次访问随机切换
  5、验证码
     验证码数量较少可人工填写
     图形验证码可使用tesseract识别
     其他情况只能在线打码、人工打码和训练机器学习模型
  6、动态生成
     一般由js动态生成的数据都是向特定的地址发get请求得到的，返回的一般是json
  7、签名及js加密
     一般为本地JS加密,查找本地JS文件,分析,或者使用execjs模块执行JS
  8、js调整页面结构
  9、js在响应中指向新的地址

# 8、scrapy框架的运行机制

# 9、分布式爬虫的原理
  多台主机共享一个爬取队列
```