Day08回顾

scrapy框架

- **五大组件+工作流程+常用命令**

```
【1】五大组件
    1.1) 引擎（Engine）
    1.2) 爬虫程序（Spider）
    1.3) 调度器（Scheduler）
    1.4) 下载器（Downloader）
    1.5) 管道文件（Pipeline）
    1.6) 下载器中间件（Downloader Middlewares）
    1.7) 蜘蛛中间件（Spider Middlewares）

【2】工作流程
    2.1) Engine向Spider索要URL,交给Scheduler入队列
    2.2) Scheduler处理后出队列,通过Downloader Middlewares交给Downloader去下载
    2.3) Downloader得到响应后,通过Spider Middlewares交给Spider
    2.4) Spider数据提取：
        a) 数据交给Pipeline处理
        b) 需要跟进URL,继续交给Scheduler入队列，依次循环

【3】常用命令
    3.1) scrapy startproject 项目名
    3.2) scrapy genspider 爬虫名 域名
    3.3) scrapy crawl 爬虫名
```

完成scrapy项目完整流程

- **完整流程**

```
【1】crapy startproject Tencent
【2】cd Tencent
【3】scrapy genspider tencent tencent.com
【4】items.py(定义爬取数据结构)
    import scrapy
    class TencentItem(scrapy.Item):
        job_name = scrapy.Field()

【5】tencent.py（写爬虫文件）
    import scrapy
    class TencentSpider(scarpy.Spider):
```

```
        name = 'tencent'
        allowed_domains = ['tencent.com']
        start_urls = ['http://tencent.com/']
        def parse(self,response):
            xxx
            yield item
【6】pipelines.py(数据处理)
    class TencentPipeline(object):
        def process_item(self,item,spider):
            return item
【7】settings.py(全局配置)
    LOG_LEVEL = ''
    LOG_FILE = ''
    FEED_EXPORT_ENCODING = ''
【8】run.py
    scrapy crawl tencent
```

我们必须记住

- **熟练记住**

```
【1】响应对象response属性及方法
    1.1) response.text ：获取响应内容 - 字符串
    1.2) response.body ：获取bytes数据类型
    1.3) response.xpath('')
    1.4) response.xpath('').extract() ：提取文本内容,将列表中所有元素序列化为Unicode字符串
    1.5) response.xpath('').extract_first() ：序列化提取列表中第1个文本内容
    1.6) response.xpath('').get() ：  提取列表中第1个文本内容(等同于extract_first())

【2】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 = 0.5
    2.7) 请求头
        DEFAULT_REQUEST_HEADERS = {}
    2.8) 添加项目管道
        ITEM_PIPELINES = {'项目名.pipelines.类名':200}
    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 = ''
        FILES_STORE = ''
    2.11) 添加下载器中间件
        DOWNLOADER_MIDDLEWARES = {'项目名.middlewares.类名':200}
```

```
【3】日志级别
    DEBUG < INFO < WARNING < ERROR < CRITICAL
```

爬虫项目启动方式

- **启动方式**

```
【1】方式一
    1.1) 从爬虫文件(spider)的start_urls变量中遍历URL地址交给调度器入队列，
    1.2) 把下载器返回的响应对象（response）交给爬虫文件的parse(self,response)函数处理

【2】方式二
    重写start_requests()方法，从此方法中获取URL，交给指定的callback解析函数处理
    2.1) 去掉start_urls变量
    2.2) def start_requests(self):
            # 生成要爬取的URL地址，利用scrapy.Request()方法交给调度器
```

数据持久化存储

- **MySQL-MongoDB-Json-csv**

```
**************************存入MySQL、MongoDB*****************************

【1】在setting.py中定义相关变量
【2】pipelines.py中新建管道类，并导入settings模块
    def open_spider(self,spider):
        # 爬虫开始执行1次,用于数据库连接

    def process_item(self,item,spider):
        # 用于处理抓取的item数据
        return item

    def close_spider(self,spider):
        # 爬虫结束时执行1次,用于断开数据库连接

【3】settings.py中添加此管道
    ITEM_PIPELINES = {'':200}

【注意】 process_item() 函数中一定要 return item

*******************************存入JSON、CSV文件***********************
scrapy crawl maoyan -o maoyan.csv
scrapy crawl maoyan -o maoyan.json
【注意】
    存入json文件时候需要添加变量(settings.py) : FEED_EXPORT_ENCODING = 'utf-8'
```

多级页面抓取之爬虫文件

- **多级页面攻略**

```
【场景1】只抓取一级页面的情况
"""
一级页面：名称(name)、爱好(likes)
"""
import scrapy
from ..items import OneItem

class OneSpider(scrapy.Spider):
    name = 'one'
    allowed_domains = ['www.one.com']
    start_urls = ['http://www.one.com/']
    def parse(self,response):
        # 抓取1条数据,将item对象传给管道1次,可以只创建1次
        item = OneItem()
        dd_list = response.xpath('//dd')
        for dd in dd_list:
            item['name'] = dd.xpath('./a/text()').get()
            item['likes'] = dd.xpath('./a/text()').get()

            yield item

【场景2】二级页面数据抓取
"""
一级页面：名称(name)、详情页链接(url)-需要继续跟进
二级页面：详情页内容(content)
"""
import scrapy
from ..items import TwoItem

class TwoSpider(scrapy.Spider):
    name = 'two'
    allowed_domains = ['www.two.com']
    start_urls = ['http://www.two.com/']
    def parse(self,response):
        """一级页面解析函数,提取 name 和 url(详情页链接,需要继续请求)"""
        dd_list = response.xpath('//dd')
        for dd in dd_list:
            item = TwoItem()
            item['name'] = dd.xpath('./text()').get()
            item['url'] = dd.xpath('./@href').get()
            # 生成1个需要跟进的URL地址,将此item对象交给调度器入队列
            yield scrapy.Request(
                url=item['url'],meta={'item':item},callback=self.parse_two_page)

    def parse_two_page(self,response):
        """二级页面解析函数,提取内容(content)"""
        item = response.meta['item']
        item['content'] = response.xpath('//content/text()').get()

        # 所有字段提取完成,yield给管道文件
        yield item

```

```
【场景3】三级页面抓取
"""
一级页面：名称(one_name)、详情页链接(one_url)-需要继续跟进
二级页面：名称(two_name)、下载页链接(two_url)-需要继续跟进
三级页面：具体所需内容(content)
"""
import scrapy
from ..items import ThreeItem

class ThreeSpider(scrapy.Spider):
    name = 'three'
    allowed_domains = ['www.three.com']
    start_urls = ['http://www.three.com/']

    def parse(self,response):
        """一级页面解析函数 - one_name、one_url"""
        dd_list = response.xpath('//dd')
        for dd in dd_list:
            # 此item需要交给调度器入队列了
            item = ThreeItem()
            item['one_name'] = dd.xpath('./text()').get()
            item['one_url'] = dd.xpath('./@href').get()
            # 交给调度器入队列
            yield scrapy.Request(
                url=item['one_url'],meta={'one_item':item},callback=self.parse_two)

    def parse_two(self,response):
        """二级页面解析函数 - two_name、two_url(需要跟进的链接有多个)"""
        one_item = response.meta['item']
        li_list = response.xpath('//li')
        for li in li_list:
            # 此处提取的链接需要继续跟进了,所以要创建item对象
            item = ThreeItem()
            item['two_name'] = li.xpath('./text()').get()
            item['two_url'] = li.xpath('./@href').get()
            # 此时item对象中只有 two_name、two_url，并没有one_name、one_url
            item['one_name'] = one_item['one_name']
            item['one_url'] = one_item['one_url']
            # 交给调度器入队列
            yield scrapy.Request(
                url=item['two_url'],meta={'two_item':item},callback=self.parse_three)

    def parse_three(self,response):
        """三级页面解析函数 - content"""
        item = response.meta['two_item']
        item['content'] = response.xpath('//content/text()').get()

        # 至此,1条完整的item数据提取完成,交给调度器入队列
        yield item
```

Day09笔记

腾讯招聘职位信息抓取

- 1、创建项目+爬虫文件

```
scrapy startproject Tencent
cd Tencent
scrapy genspider tencent careers.tencent.com

# 一级页面(postId):
https://careers.tencent.com/tencentcareer/api/post/Query?timestamp=1566266592644&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attrId=&keyword={}&pageIndex={}&pageSize=10&language=zh-cn&area=cn

# 二级页面(名称+类别+职责+要求+地址+时间)
https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1566266695175&postId={}&language=zh-cn
```

- 2、定义爬取的数据结构

```
import scrapy

class TencentItem(scrapy.Item):
    # 名称+类别+职责+要求+地址+时间
    job_name = scrapy.Field()
    job_type = scrapy.Field()
    job_duty = scrapy.Field()
    job_require = scrapy.Field()
    job_address = scrapy.Field()
    job_time = scrapy.Field()
    # 具体职位链接
    job_url = scrapy.Field()
    post_id = scrapy.Field()
```

- 3、爬虫文件

```
# -*- coding: utf-8 -*-
import scrapy
from urllib import parse
import requests
import json
from ..items import TencentItem


class TencentSpider(scrapy.Spider):
    name = 'tencent'
    allowed_domains = ['careers.tencent.com']
    # 定义常用变量
```

```
    one_url = 'https://careers.tencent.com/tencentcareer/api/post/Query?timestamp=1566266592644&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attrId=&keyword={}&pageIndex={}&pageSize=10&language=zh-cn&area=cn'
    two_url = 'https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1566266695175&postId={}&language=zh-cn'
    headers = {'User-Agent': 'Mozilla/5.0'}
    keyword = input('请输入职位类别:')
    keyword = parse.quote(keyword)

    # 重写start_requests()方法
    def start_requests(self):
        total = self.get_total()
        # 生成一级页面所有页的URL地址,交给调度器
        for index in range(1,total+1):
            url = self.one_url.format(self.keyword,index)
            yield scrapy.Request(url=url,callback=self.parse_one_page)

    # 获取总页数
    def get_total(self):
        url = self.one_url.format(self.keyword, 1)
        html = requests.get(url=url, headers=self.headers).json()
        count = html['Data']['Count']
        total = count//10 if count%10==0 else count//10 + 1

        return total

    def parse_one_page(self, response):
        html = json.loads(response.text)
        for one in html['Data']['Posts']:
            # 此处是不是有URL需要交给调度器去入队列了?  - 创建item对象!
            item = TencentItem()
            item['post_id'] = one['PostId']
            item['job_url'] = self.two_url.format(item['post_id'])
            # 创建1个item对象,请将其交给调度器入队列
            yield scrapy.Request(url=item['job_url'],meta={'item':item},callback=self.detail_page)

    def detail_page(self,response):
        """二级页面: 详情页数据解析"""
        item = response.meta['item']
        # 将响应内容转为python数据类型
        html = json.loads(response.text)
        # 名称+类别+职责+要求+地址+时间
        item['job_name'] = html['Data']['RecruitPostName']
        item['job_type'] = html['Data']['CategoryName']
        item['job_duty'] = html['Data']['Responsibility']
        item['job_require'] = html['Data']['Requirement']
        item['job_address'] = html['Data']['LocationName']
        item['job_time'] = html['Data']['LastUpdateTime']

        # 至此: 1条完整数据提取完成,没有继续送往调度器的请求了,交给管道文件
        yield item
```

- **4、提前建库建表 - MySQL**

```
create database tencentdb charset utf8;
use tencentdb;
create table tencenttab(
job_name varchar(500),
job_type varchar(200),
job_duty varchar(5000),
job_require varchar(5000),
job_address varchar(100),
job_time varchar(100)
)charset=utf8;
```

- **5、管道文件**

```
class TencentPipeline(object):
    def process_item(self, item, spider):
        return item

import pymysql
from .settings import *

class TencentMysqlPipeline(object):
    def open_spider(self,spider):
        """爬虫项目启动时,连接数据库1次"""
        self.db = pymysql.connect(MYSQL_HOST,MYSQL_USER,MYSQL_PWD,MYSQL_DB,charset=CHARSET)
        self.cursor = self.db.cursor()

    def process_item(self,item,spider):
        ins='insert into tencenttab values(%s,%s,%s,%s,%s,%s)'
        job_li = [
            item['job_name'],
            item['job_type'],
            item['job_duty'],
            item['job_require'],
            item['job_address'],
            item['job_time']
        ]
        self.cursor.execute(ins,job_li)
        self.db.commit()

        return item

    def close_spider(self,spider):
        """爬虫项目结束时,断开数据库1次"""
        self.cursor.close()
        self.db.close()
```

- **6、settings.py**

```
ROBOTS_TXT = False
DOWNLOAD_DELAY = 0.5
DEFAULT_REQUEST_HEADERS = {
  'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
  'Accept-Language': 'en',
  'User-Agent': 'Mozilla/5.0',
}
```

```
ITEM_PIPELINES = {
    'Tencent.pipelines.TencentPipeline': 300,
    'Tencent.pipelines.TencentMysqlPipeline': 500,
}
# MySQL相关变量
MYSQL_HOST = 'localhost'
MYSQL_USER = 'root'
MYSQL_PWD = '123456'
MYSQL_DB = 'tencentdb'
CHARSET = 'utf8'
```

盗墓笔记小说抓取

- **目标**

```
【1】URL地址 ： http://www.daomubiji.com/
【2】要求 ： 抓取目标网站中盗墓笔记所有章节的所有小说的具体内容，保存到本地文件
    ./data/novel/盗墓笔记1:七星鲁王宫/七星鲁王_第一章_血尸.txt
    ./data/novel/盗墓笔记1:七星鲁王宫/七星鲁王_第二章_五十年后.txt
```

- **准备工作xpath**

```
【1】一级页面 - 大章节标题、链接:
    1.1) 基准xpath匹配a节点对象列表:  '//li[contains(@id,"menu-item-20")]/a'
    1.2) 大章节标题: './text()'
    1.3) 大章节链接: './@href'

【2】二级页面 - 小章节标题、链接
    2.1) 基准xpath匹配article节点对象列表: '//article'
    2.2) 小章节标题: './a/text()'
    2.3) 小章节链接: './a/@href'

【3】三级页面 - 小说内容
    3.1) p节点列表: '//article[@class="article-content"]/p/text()'
    3.2) 利用join()进行拼接: ' '.join(['p1','p2','p3',''])
```

项目实现

- **1、创建项目及爬虫文件**

```
scrapy startproject Daomu
cd Daomu
scrapy genspider daomu www.daomubiji.com
```

- **2、定义要爬取的数据结构 - itemspy**

```python
import scrapy

class DaomuItem(scrapy.Item):
    # 1. 一级页面标题+链接
    parent_title = scrapy.Field()
    parent_url = scrapy.Field()
    # 2. 二级页面标题+链接
    son_title = scrapy.Field()
    son_url = scrapy.Field()
    # 3. 目录
    directory = scrapy.Field()
    # 4. 小说内容
    content = scrapy.Field()
```

- **3、爬虫文件实现数据抓取 - daomu.py**

```python
# -*- coding: utf-8 -*-
import scrapy
from ..items import DaomuItem
import os

class DaomuSpider(scrapy.Spider):
    name = 'daomu'
    allowed_domains = ['www.daomubiji.com']
    start_urls = ['http://www.daomubiji.com/']

    def parse(self, response):
        """一级页面解析函数"""
        # 基准xpath
        a_list = response.xpath('//li[contains(@id,"menu-item-20")]/a')
        for a in a_list:
            # 此处是不是有需要继续交给调度器的请求了？ - 创建item对象！！！
            item = DaomuItem()
            item['parent_title'] = a.xpath('./text()').get()
            item['parent_url'] = a.xpath('./@href').get()
            item['directory'] = './novel/{}/'.format(item['parent_title'])
            if not os.path.exists(item['directory']):
                os.makedirs(item['directory'])

            # 交给调度器入队列
            yield scrapy.Request(url=item['parent_url'], meta={'meta_1': item},
callback=self.detail_page)

    def detail_page(self,response):
        """二级页面解析：提取小章节名称、链接"""
        meta_1 = response.meta['meta_1']
        # 基准xpath,获取所有章节的节点对象列表
        article_list = response.xpath('//article')
        for article in article_list:
            # 此处是不是有继续交给调度器的请求了？ - 创建item对象！！！
            item = DaomuItem()
            item['son_title'] = article.xpath('./a/text()').get()
            item['son_url'] = article.xpath('./a/@href').get()
            item['parent_title'] = meta_1['parent_title']
            item['parent_url'] = meta_1['parent_url']
```

```
            item['directory'] = meta_1['directory']

            # 交给调度器入队列,创建1个,交1个
            yield scrapy.Request(url=item['son_url'],meta=
{'meta_2':item},callback=self.get_content)

    def get_content(self,response):
        """三级页面解析函数：获取小说内容"""
        item = response.meta['meta_2']
        # p_list: ['段落1','段落2','段落3']
        p_list = response.xpath('//article[@class="article-content"]//p/text()').extract()
        content = '\n'.join(p_list)
        item['content'] = content

        # 没有继续交给调度器的请求了,所以不用创建item对象,直接交给管道文件处理
        yield item
```

- **4、管道文件实现数据处理 - pipelines.py**

```
class DaomuPipeline(object):
    def process_item(self, item, spider):
        filename = item['directory'] + item['son_title'].replace(' ','_') + '.txt'
        with open(filename,'w') as f:
            f.write(item['content'])

        return item
```

- **5、全局配置 - setting.py**

```
ROBOTSTXT_OBEY = False
DOWNLOAD_DELAY = 0.5
DEFAULT_REQUEST_HEADERS = {
  'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
  'Accept-Language': 'en',
  'User-Agent': 'Mozilla/5.0'
}
ITEM_PIPELINES = {
   'Daomu.pipelines.DaomuPipeline': 300,
}
```

图片管道(360图片抓取案例)

- **目标**

```
【1】URL地址
   1.1) www.so.com -> 图片 -> 美女
   1.2) 即: https://image.so.com/z?ch=beauty

【2】图片保存路径
   ./images/xxx.jpg
```

- **抓取网络数据包**

```
【1】通过分析，该网站为Ajax动态加载
【2】F12抓包，抓取到json地址 和 查询参数(QueryString)
   2.1) url = 'https://image.so.com/zjl?ch=beauty&sn={}&listtype=new&temp=1'
   2.2) 查询参数
        ch: beauty
        sn: 0 # 发现sn的值在变,0 30 60 90 120 ... ...
        listtype: new
        temp: 1
```

项目实现

- **1、创建爬虫项目和爬虫文件**

```
scrapy startproject So
cd So
scrapy genspider so image.so.com
```

- **2、定义要爬取的数据结构(items.py)**

```
img_url = scrapy.Field()
img_title = scrapy.Field()
```

- **3、爬虫文件实现图片链接+名字抓取**

```
import scrapy
import json
from ..items import SoItem

class SoSpider(scrapy.Spider):
    name = 'so'
    allowed_domains = ['image.so.com']
    # 重写start_requests()方法
    url = 'https://image.so.com/zjl?ch=beauty&sn={}&listtype=new&temp=1'

    def start_requests(self):
        for sn in range(0,91,30):
            full_url = self.url.format(sn)
            # 扔给调度器入队列
            yield scrapy.Request(url=full_url,callback=self.parse_image)

    def parse_image(self,response):
        html = json.loads(response.text)
        item = SoItem()
        for img_dict in html['list']:
            item['img_url'] = img_dict['qhimg_url']
            item['img_title'] = img_dict['title']

            yield item
```

- **4、管道文件（pipelines.py）**

```python
from scrapy.pipelines.images import ImagesPipeline
import scrapy

class SoPipeline(ImagesPipeline):
    # 重写get_media_requests()方法
    def get_media_requests(self, item, info):
        yield scrapy.Request(url=item['img_url'],meta={'name':item['img_title']})

    # 重写file_path()方法,自定义文件名
    def file_path(self, request, response=None, info=None):
        img_link = request.url
        # request.meta属性
        filename = request.meta['name'] + '.' + img_link.split('.')[-1]
        return filename
```

- **5、全局配置(settings.py)**

```python
ROBOTSTXT_OBEY = False
DOWNLOAD_DELAY = 0.1
DEFAULT_REQUEST_HEADERS = {
  'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
  'Accept-Language': 'en',
  'User-Agent': 'Mozilla/5.0',
}
ITEM_PIPELINES = {
   'So.pipelines.SoPipeline': 300,
}
IMAGES_STORE = 'D:/AID1910/spider_day09_code/So/images/'
```

- **6、运行爬虫(run.py)**

```python
from scrapy import cmdline

cmdline.execute('scrapy crawl so'.split())
```

图片管道使用方法总结

```
【1】爬虫文件：将图片链接yield到管道
【2】管道文件：
  from scrapy.pipelines.images import ImagesPipeline
  class XxxPipeline(ImagesPipeline):
       def get_media_requests(self,xxx):
           pass

       def file_path(self,xxx):
           pass

【3】settings.py中：
  IMAGES_STORE = '绝对路径'
```

文件管道使用方法总结

```
【1】爬虫文件：将文件链接yield到管道
【2】管道文件：
  from scrapy.pipelines.images import FilesPipeline
  class XxxPipeline(FilesPipeline):
       def get_media_requests(self,xxx):
           pass

       def file_path(self,xxx):
           return filename

【3】settings.py中：
  FILES_STORE = '绝对路径'
```

scrapy - post请求

- **方法+参数**

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

有道翻译案例实现

- **1、创建项目+爬虫文件**

```
scrapy startproject Youdao
cd Youdao
scrapy genspider youdao fanyi.youdao.com
```

- **2、items.py**

```
result = scrapy.Field()
```

- **3、youdao.py**

```
# -*- coding: utf-8 -*-
import scrapy
import time
import random
from hashlib import md5
import json
from ..items import YoudaoItem

class YoudaoSpider(scrapy.Spider):
    name = 'youdao'
    allowed_domains = ['fanyi.youdao.com']
    word = input('请输入要翻译的单词:')

    def start_requests(self):
        post_url = 'http://fanyi.youdao.com/translate_o?smartresult=dict&smartresult=rule'
        salt, sign, ts = self.get_salt_sign_ts(self.word)
        formdata = {
                  'i': self.word,
                  'from': 'AUTO',
                  'to': 'AUTO',
                  'smartresult': 'dict',
                  'client': 'fanyideskweb',
                  'salt': salt,
                  'sign': sign,
                  'ts': ts,
                  'bv': 'cf156b581152bd0b259b90070b1120e6',
                  'doctype': 'json',
                  'version': '2.1',
                  'keyfrom': 'fanyi.web',
                  'action': 'FY_BY_REALTlME'
              }
       # 发送post请求的方法
        yield scrapy.FormRequest(url=post_url,formdata=formdata)

    def get_salt_sign_ts(self, word):
        # salt
        salt = str(int(time.time() * 1000)) + str(random.randint(0, 9))
        # sign
        string = "fanyideskweb" + word + salt + "n%A-rKaT5fb[Gy?;N5@Tj"
        s = md5()
        s.update(string.encode())
        sign = s.hexdigest()
        # ts
        ts = str(int(time.time() * 1000))
        return salt, sign, ts
```

```
46
47      def parse(self, response):
48          item = YoudaoItem()
49          html = json.loads(response.text)
50          item['result'] = html['translateResult'][0][0]['tgt']
51
52          yield item
```

- **4、pipelines.py**

```
1  class YoudaoPipeline(object):
2      def process_item(self, item, spider):
3          print('翻译结果:',item['result'])
4          return item
```

- **5、settings.py**

```
1  ROBOTSTXT_OBEY = False
2  LOG_LEVEL = 'WARNING'
3  COOKIES_ENABLED = False
4  DEFAULT_REQUEST_HEADERS = {
5        "Cookie": "OUTFOX_SEARCH_USER_ID=970246104@10.169.0.83;
   OUTFOX_SEARCH_USER_ID_NCOO=570559528.1224236;
   _ntes_nnid=96bc13a2f5ce64962adfd6a278467214,1551873108952;
   JSESSIONID=aaae9i7plXPlKaJH_gkYw; td_cookie=18446744072941336803;
   SESSION_FROM_COOKIE=unknown; ___rl__test__cookies=1565689460872",
6        "Referer": "http://fanyi.youdao.com/",
7        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
   like Gecko) Chrome/76.0.3809.100 Safari/537.36",
8  }
9  ITEM_PIPELINES = {
10    'Youdao.pipelines.YoudaoPipeline': 300,
11 }
```

scrapy添加cookie的三种方式

```
1  【1】修改 settings.py 文件
2     1.1) COOKIES_ENABLED = False  -> 取消注释,开启cookie,检查headers中的cookie
3     1.2) DEFAULT_REQUEST_HEADERS = {}   添加Cookie
4
5  【2】利用cookies参数
6     1.1) settings.py: COOKIES_ENABLED = True # 修改为TRUE后, 检查 Request()方法中cookies
7     1.2) def start_requests(self):
8              yield scrapy.Request(url=url,cookies={},callback=xxx)
9
10 【3】DownloadMiddleware设置中间件
11    3.1) settings.py: COOKIES_ENABLED = TRUE  # 找Request()方法中cookies参数
12    3.2) middlewares.py
13         def process_request(self,request,spider):
14             request.cookies={}
```

scrapy shell的使用

- **定义**

```
【1】调试蜘蛛的工具
【2】交互式shell，可在不运行spider的前提下,快速调试 scrapy 代码(主要测试xpath表达式)
```

- **基本使用**

```
# scrapy shell URL地址
*1、request.url     : 请求URL地址
*2、request.headers : 请求头(字典)
*3、request.meta    : item数据传递，定义代理(字典)

4、response.text    : 字符串
5、response.body    : bytes
6、response.xpath('')
7、response.status  : HTTP响应码

# 可用方法
shelp() : 帮助
fetch(request) : 从给定的请求中获取新的响应，并更新所有相关对象
view(response) : 在本地Web浏览器中打开给定的响应以进行检查
```

- **scrapy.Request()参数**

```
1、url
2、callback
3、headers
4、meta : 传递数据,定义代理
5、dont_filter : 是否忽略域组限制
   默认False,检查allowed_domains['']
6、cookies
```

设置中间件(随机User-Agent)

- **少量User-Agent切换**

```
【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User-Agent切换（middlewares.py设置中间件）**

```
【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 : 安装fake_useragent模块(sudo pip3 install fack_useragent)
```

```
        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)

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

    def process_exception(self,request,exception,spider):
        return request
```

- **练习**

```
有道翻译,将cookie以中间件的方式添加的scrapy项目中
```

今日作业

```
【1】URL地址: http://www.1ppt.com/xiazai/
【2】目标:
    2.1) 爬取所有栏目分类下的,所有页的PPT
    2.2) 数据存放: /home/tarena/ppts/工作总结PPT/xxx
                   /home/tarena/ppts/个人简历PPT/xxx
【提示】: 使用FilesPipeline,并重写方法
```