

Day08回顾

scrapy框架

■ 五大组件+工作流程+常用命令

```
1  【1】五大组件
2      1.1) 引擎 (Engine)
3      1.2) 爬虫程序 (Spider)
4      1.3) 调度器 (Scheduler)
5      1.4) 下载器 (Downloader)
6      1.5) 管道文件 (Pipeline)
7      1.6) 下载器中间件 (Downloader Middlewares)
8      1.7) 蜘蛛中间件 (Spider Middlewares)
9
10 【2】工作流程
11     2.1) Engine向Spider索要URL,交给Scheduler入队列
12     2.2) Scheduler处理后出队列,通过Downloader Middlewares交给Downloader去下载
13     2.3) Downloader得到响应后,通过Spider Middlewares交给Spider
14     2.4) Spider数据提取:
15         a) 数据交给Pipeline处理
16         b) 需要跟进URL,继续交给Scheduler入队列,依次循环
17
18 【3】常用命令
19     3.1) scrapy startproject 项目名
20     3.2) scrapy genspider 爬虫名 域名
21     3.3) scrapy crawl 爬虫名
```

完成scrapy项目完整流程

■ 完整流程

```
1  【1】crapy startproject Tencent
2  【2】cd Tencent
3  【3】scrapy genspider tencent tencent.com
4  【4】items.py(定义爬取数据结构)
5      import scrapy
6      class TencentItem(scrapy.Item):
7          name = scrapy.Field()
8          address = scrapy.Field()
9
10 【5】tencent.py (写爬虫文件)
11     import scrapy
```

```

12     from ..items import TencentItem
13
14     class TencentSpider(scrapy.Spider):
15         name = 'tencent'
16         allowed_domains = ['tencent.com']
17         start_urls = ['http://tencent.com/']
18         def parse(self, response):
19             item = TencentItem()
20             xxx
21             yield item
22
23     【6】 pipelines.py(数据处理)
24     class TencentPipeline(object):
25         def process_item(self, item, spider):
26             return item
27
28     【7】 settings.py(全局配置)
29     LOG_LEVEL = '' # DEBUG < INFO < WARNING < ERROR < CRITICAL
30     LOG_FILE = ''
31     FEED_EXPORT_ENCODING = ''
32
33     【8】 run.py
34     from scrapy import cmdline
35     cmdline.execute('scrapy crawl tencnet'.split())

```

我们必须记住

■ 熟练记住

```

1     【1】 响应对象response属性及方法
2         1.1) response.text : 获取响应内容 - 字符串
3         1.2) response.body : 获取bytes数据类型
4         1.3) response.xpath('')
5         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
6         1.5) response.xpath('').extract_first() : 序列化提取列表中第1个文本内容
7         1.6) response.xpath('').get() : 提取列表中第1个文本内容(等同于extract_first())
8
9     【2】 settings.py中常用变量
10        2.1) 设置日志级别
11            LOG_LEVEL = ''
12        2.2) 保存到日志文件(不在终端输出)
13            LOG_FILE = ''
14        2.3) 设置数据导出编码(主要针对于json文件)
15            FEED_EXPORT_ENCODING = 'utf-8'
16        2.4) 设置User-Agent
17            USER_AGENT = ''
18        2.5) 设置最大并发数(默认为16)
19            CONCURRENT_REQUESTS = 32
20        2.6) 下载延迟时间(每隔多长时间请求一个网页)
21            DOWNLOAD_DELAY = 1
22        2.7) 请求头
23            DEFAULT_REQUEST_HEADERS = {'User-Agent': 'Mozilla/'}
24        2.8) 添加项目管道

```

```

25     ITEM_PIPELINES = {'项目目录名.pipelines.类名': 优先级}
26 2.9) cookie(默认禁用,取消注释-True|False都为开启)
27     COOKIES_ENABLED = False
28 2.10) 非结构化数据存储路径
29     IMAGES_STORE = '/home/tarena/images/'
30     FILES_STORE = '/home/tarena/files/'
31 2.11) 添加下载器中间件
32     DOWNLOADER_MIDDLEWARES = {'项目名.middlewares.类名': 200}
33
34 【3】 日志级别
35     DEBUG < INFO < WARNING < ERROR < CRITICAL

```

爬虫项目启动方式

启动方式

```

1  【1】 方式一:基于start_urls
2     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,
3     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
4
5  【2】 方式二
6     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
7     2.1) 去掉start_urls变量
8     2.2) def start_requests(self):
9         # 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

```

1  *****存入MySQL、MongoDB*****
2
3  【1】 在setting.py中定义相关变量
4  【2】 pipelines.py中新建管道类,并导入settings模块
5     def open_spider(self, spider):
6         # 爬虫开始执行1次,用于数据库连接
7
8     def process_item(self, item, spider):
9         # 用于处理抓取的item数据
10        return item
11
12    def close_spider(self, spider):
13        # 爬虫结束时执行1次,用于断开数据库连接
14
15  【3】 settings.py中添加此管道
16     ITEM_PIPELINES = {'': 200}
17
18  【注意】 process_item() 函数中一定要 return item
19
20 *****存入JSON、CSV文件*****

```

```
21 scrapy crawl maoyan -o maoyan.csv
22 scrapy crawl maoyan -o maoyan.json
23 【注意】
24 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

```
1  【场景1】只抓取一级页面的情况
2  """
3  一级页面: 名称(name)、爱好(likes)
4  """
5  import scrapy
6  from ..items import OneItem
7  class OneSpider(scrapy.Spider):
8      name = 'One'
9      allowed_domains = ['www.one.com']
10     start_urls = ['http://www.one.com']
11     def parse(self, response):
12         dd_list = response.xpath('//dd')
13         for dd in dd_list:
14             # 创建item对象
15             item = OneItem()
16             item['name'] = dd.xpath('./text()').get()
17             item['likes'] = dd.xpath('./text()').get()
18
19             yield item
20
21
22  【场景2】二级页面数据抓取
23  """
24  一级页面: 名称(name)、详情页链接(url)-需要继续跟进
25  二级页面: 详情页内容(content)
26  """
27  import scrapy
28  from ..items import TwoItem
29
30  class TwoSpider(scrapy.Spider):
31      name = 'two'
32      allowed_domains = ['www.two.com']
33      start_urls = ['http://www.two.com/']
34      def parse(self, response):
35          """一级页面解析函数,提取 name 和 url(详情页链接,需要继续请求)"""
36          dd_list = response.xpath('//dd')
37          for dd in dd_list:
38              # 有继续交给调度器入队列的请求,就要创建item对象
39              item = TwoItem()
40              item['name'] = dd.xpath('./text()').get()
41              item['url'] = dd.xpath('./@href').get()
42
43              yield scrapy.Request(
44                  url=item['url'], meta={'item': item}, callback=self.detail_page)
```

```

45
46     def detail_page(self, response):
47         item = response.meta['item']
48         item['content'] = response.xpath('//text()').get()
49
50         yield item
51
52
53     【场景3】三级页面抓取
54     """
55     一级页面：名称(one_name)、详情页链接(one_url)-需要继续跟进
56     二级页面：名称(two_name)、下载页链接(two_url)-需要继续跟进
57     三级页面：具体所需内容(content)
58     """
59     import scrapy
60     from ..items import ThreeItem
61
62     class ThreeSpider(scrapy.Spider):
63         name = 'three'
64         allowed_domains = ['www.three.com']
65         start_urls = ['http://www.three.com/']
66
67         def parse(self, response):
68             """一级页面解析函数 - one_name、one_url"""
69             dd_list = response.xpath('//dd')
70             for dd in dd_list:
71                 # 有继续发往调度器的请求,创建item对象的时刻到啦!!
72                 item = ThreeItem()
73                 item['one_name'] = dd.xpath('./text()').get()
74                 item['one_url'] = dd.xpath('./@href').get()
75                 yield scrapy.Request(
76                     url=item['one_url'], meta={'meta_1': item}, callback=self.parse_two)
77
78         def parse_two(self, response):
79             """二级页面解析函数: two_name、two_url"""
80             meta1_item = response.meta['meta_1']
81             li_list = response.xpath('//li')
82             for li in li_list:
83                 # 有继续交给调度器入队列的请求啦,所以创建item对象的时刻来临了!!!
84                 item = ThreeItem()
85                 item['two_name'] = li.xpath('./text()').get()
86                 item['two_url'] = li.xpath('./@href').get()
87                 item['one_name'] = meta1_item['one_name']
88                 item['one_url'] = meta1_item['one_url']
89                 # 交给调度器入队列
90                 yield scrapy.Request(
91                     url=item['two_url'], meta={'meta_2': item}, callback=self.detail_page)
92
93         def detail_page(self, response):
94             """三级页面解析: 具体内容content"""
95             item = response.meta['meta_2']
96             # 太好了! 提具体内容了,没有继续交给调度器的请求了! 所以,我不再去创建item对象啦
97             item['content'] = response.xpath('//text()').get()
98
99             # 交给管道文件处理
100             yield item

```

Day09笔记

新浪新闻全站抓取

■ 目标

- 1 【1】 抓取新浪新闻下的所有分类的所有新闻，保存到本地
- 2 【2】 URL：新浪官网 - 更多 - 导航
- 3 <http://news.sina.com.cn/guide/>
- 4 【3】 要求
- 5 将信息保存到scrapy项目目录的 data 文件夹中,并按照分类名称创建子文件夹

实现步骤

■ 步骤1 - 创建项目和爬虫文件

```
1 scrapy startproject Sina
2 cd Sina
3 scrapy genspider sina news.sina.com.cn
```

■ 步骤2 - 定义要抓取的数据结构 (items.py)

```
1 class SinaItem(scrapy.Item):
2     # define the fields for your item here like:
3     # 一级页面：大类名称、大类URL、小类名称、小类URL
4     parent_name = scrapy.Field()
5     parent_url = scrapy.Field()
6     son_name = scrapy.Field()
7     son_url = scrapy.Field()
8     # 二级页面：新闻链接
9     news_url = scrapy.Field()
10    # 三级页面：新闻标题、新闻内容
11    news_head = scrapy.Field()
12    news_content = scrapy.Field()
13    # 路径：./data/体育/NBA/
14    son_directory = scrapy.Field()
```

■ 步骤3 - 爬虫文件进行数据解析提取(sina.py)

```
1 # -*- coding: utf-8 -*-
2 import scrapy
3 from ..items import SinaItem
4 import os
5
6 class SinaSpider(scrapy.Spider):
7     name = 'sina'
8     allowed_domains = ['sina.com.cn']
```

```

9     start_urls = ['http://news.sina.com.cn/guide/']
10
11     def parse(self, response):
12         # 基准xpath: 提取所有大分类的节点对象列表
13         div_list = response.xpath('//div[@id="tab01"]/div')
14         for div in div_list:
15             # 大分类名称+URL
16             # xpath表达式或 | , 匹配地方站(./h3/span/text())
17             parent_name = div.xpath('./h3/a/text() | ./h3/span/text()').get()
18             parent_url = div.xpath('./h3/a/@href').get()
19             # 地方站情况特殊,没有大链接
20             if not parent_url:
21                 parent_url = 'http://'
22
23             # 小分类的 li 节点对象列表
24             li_list = div.xpath('./ul/li')
25             for li in li_list:
26                 # 创建item对象: 继续交给调度器入队列的请求对象
27                 item = SinaItem()
28                 item['son_name'] = li.xpath('./a/text()').get()
29                 item['son_url'] = li.xpath('./a/@href').get()
30                 item['parent_name'] = parent_name
31                 item['parent_url'] = parent_url
32                 # son_directory: ./data/体育/NBA/
33                 son_directory =
34                 './data/{}/{}'.format(item['parent_name'], item['son_name'])
35                 item['son_directory'] = son_directory
36                 # 创建对应的目录结构
37                 if not os.path.exists(son_directory):
38                     os.makedirs(son_directory)
39
40                 yield scrapy.Request(url=item['son_url'], meta=
41                 {'meta1': item}, callback=self.parse_son_url)
42
43     def parse_son_url(self, response):
44         """解析1个小分类的函数 - 提取新闻链接"""
45         meta1_item = response.meta['meta1']
46         # 通过观察URL地址规律,新闻链接基本上都是以 大分类URL开头,且以.shtml结尾
47         news_url_list = response.xpath('//a/@href').extract()
48         for news_url in news_url_list:
49             if news_url.startswith(meta1_item['parent_url']) and
50             news_url.endswith('.shtml'):
51                 # 只要你想把URL地址交给调度器入队列了,说明你创建item对象的时刻到了
52                 item = SinaItem()
53                 item['news_url'] = news_url
54                 item['parent_name'] = meta1_item['parent_name']
55                 item['parent_url'] = meta1_item['parent_url']
56                 item['son_name'] = meta1_item['son_name']
57                 item['son_url'] = meta1_item['son_url']
58                 item['son_directory'] = meta1_item['son_directory']
59
60                 yield scrapy.Request(url=item['news_url'], meta=
61                 {'meta2': item}, callback=self.get_content)
62
63     def get_content(self, response):
64         """提取具体新闻内容的函数"""
65         item = response.meta['meta2']

```

```

62         # 此处增加了几个类别的xpath匹配,因为有些类别的新闻 xpath 不一样
63         item['news_head'] = response.xpath('//h1[@class="main-title"]/text() |
//span[@class="location"]/h1/text()).get()
64         item['news_content'] = '\n'.join(response.xpath('//div[@class="article"]/p/text() |
//div[@id="artibody"]/p/text()).extract())
65
66         yield item

```

■ 步骤4 - 数据处理(pipelines.py)

```

1  class SinaPipeline(object):
2      def process_item(self, item, spider):
3          # url: http://dl.sina.com.cn/zl/auto/sinadlcyh/index.shtml
4          # 文件名使用url地址的中间(即去掉协议和后缀.shtml)
5          filename = item['news_url'][7:-6].replace('/', '-')
6          filename_ = item['son_directory'] + filename + '.txt'
7          # 写入本地文件
8          with open(filename_, 'w', encoding='utf-8') as f:
9              f.write(item['news_content'])
10
11         return item
12

```

■ 步骤5 - 全局配置(settings.py)

```

1  """settings.py"""
2  ROBOTSTXT_OBEY = False
3  DOWNLOAD_DELAY = 1
4  DEFAULT_REQUEST_HEADERS = {
5      'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
6      'Accept-Language': 'en',
7      'User-Agent': 'Mozilla/5.0',
8  }
9  ITEM_PIPELINES = {
10     'Sina.pipelines.SinaPipeline': 300,
11 }

```

■ 步骤6 - 运行爬虫(run.py)

```

1  from scrapy import cmdline
2  cmdline.execute('scrapy crawl sina'.split())

```

图片管道(360图片抓取案例)

■ 目标


```
1 【1】URL地址
2   1.1) www.so.com -> 图片 -> 美女
3   1.2) 即: https://image.so.com/z?ch=beauty
4
5 【2】图片保存路径
6   ./images/xxx.jpg
```

■ 抓取网络数据包

```
1 【1】通过分析, 该网站为Ajax动态加载
2 【2】F12抓包, 抓取到json地址 和 查询参数(QueryString)
3   2.1) url = 'https://image.so.com/zjl?ch=beauty&sn={}&listtype=new&temp=1'
4   2.2) 查询参数
5         ch: beauty
6         sn: 0 # 发现sn的值在变, 0 30 60 90 120 ... ...
7         listtype: new
8         temp: 1
```

项目实施

■ 1、创建爬虫项目和爬虫文件

```
1 scrapy startproject So
2 cd So
3 scrapy genspider so image.so.com
```

■ 2、定义要爬取的数据结构(items.py)

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

■ 3、爬虫文件实现图片链接+名字抓取

```
1 import scrapy
2 import json
3 from ..items import SoItem
4
5 class SoSpider(scrapy.Spider):
6     name = 'so'
7     allowed_domains = ['image.so.com']
8     # 重写start_requests()方法
9     url = 'https://image.so.com/zjl?ch=beauty&sn={}&listtype=new&temp=1'
10
11     def start_requests(self):
12         for sn in range(0, 91, 30):
13             full_url = self.url.format(sn)
14             # 扔给调度器入队列
15             yield scrapy.Request(url=full_url, callback=self.parse_image)
16
17     def parse_image(self, response):
```

```

18         html = json.loads(response.text)
19         item = SoItem()
20         for img_dict in html['list']:
21             item['img_url'] = img_dict['qhimg_url']
22             item['img_title'] = img_dict['title']
23
24         yield item

```

■ 4、管道文件 (pipelines.py)

```

1  from scrapy.pipelines.images import ImagesPipeline
2  import scrapy
3
4  class SoPipeline(ImagesPipeline):
5      # 重写get_media_requests()方法
6      def get_media_requests(self, item, info):
7          yield scrapy.Request(url=item['img_url'], meta={'name': item['img_title']})
8
9      # 重写file_path()方法, 自定义文件名
10     def file_path(self, request, response=None, info=None):
11         img_link = request.url
12         # request.meta属性
13         filename = request.meta['name'] + '.' + img_link.split('.')[ -1]
14         return filename

```

■ 5、全局配置(settings.py)

```

1  ROBOTSTXT_OBEY = False
2  DOWNLOAD_DELAY = 0.1
3  DEFAULT_REQUEST_HEADERS = {
4      'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
5      'Accept-Language': 'en',
6      'User-Agent': 'Mozilla/5.0',
7  }
8  ITEM_PIPELINES = {
9      'So.pipelines.SoPipeline': 300,
10 }
11 IMAGES_STORE = 'D:/AID1910/spider_day09_code/So/images/'

```

■ 6、运行爬虫(run.py)

```

1  from scrapy import cmdline
2
3  cmdline.execute('scrapy crawl so'.split())

```

图片管道使用方法总结

```

1  【1】爬虫文件：将图片链接yield到管道
2  【2】管道文件：
3      from scrapy.pipelines.images import ImagesPipeline
4      class XxxPipeline(ImagesPipeline):
5          def get_media_requests(self,xxx):
6              pass
7
8          def file_path(self,xxx):
9              pass
10
11  【3】 settings.py中：
12      IMAGES_STORE = '绝对路径'

```

文件管道使用方法总结

```

1  【1】爬虫文件：将文件链接yield到管道
2  【2】管道文件：
3      from scrapy.pipelines.files import FilesPipeline
4      class XxxPipeline(FilesPipeline):
5          def get_media_requests(self,xxx):
6              pass
7
8          def file_path(self,xxx):
9              return filename
10
11  【3】 settings.py中：
12      FILES_STORE = '绝对路径'

```

scrapy - post请求

▪ 方法+参数

```

1  scrapy.FormRequest(
2      url=posturl,
3      formdata=formdata,
4      callback=self.parse
5  )

```

抓取全国所有城市肯德基门店信息

▪ 目标说明

```
1  【1】 主页URL地址
2      http://www.kfc.com.cn/kfccda/storelist/index.aspx
3
4  【2】 抓取所有城市的肯德基门店信息
5      2.1) 门店编号
6      2.2) 门店名称
7      2.3) 门店地址
8      2.4) 所属城市
9
10 【3】 将所抓数据存储在MySQL数据库中
```

■ 步骤1 - 创建项目+爬虫文件

```
1 scrapy startproject Kfc
2 cd Kfc
3 scrapy genspider kfc www.kfc.com.cn
```

■ 步骤2 - 定义要抓取的数据结构(items.py)

```
1 import scrapy
2
3 class KfcItem(scrapy.Item):
4     # 门店编号 + 门店名称 + 门店地址 + 所属城市
5     row_num = scrapy.Field()
6     store_name = scrapy.Field()
7     address_detail = scrapy.Field()
8     city_name = scrapy.Field()
```

■ 步骤3 - 写爬虫程序(kfc.py)

```
1  # -*- coding: utf-8 -*-
2  import scrapy
3  import requests
4  import json
5  import re
6  from ..items import KfcItem
7
8
9  class KfcSpider(scrapy.Spider):
10     name = 'kfc'
11     allowed_domains = ['www.kfc.com.cn']
12     index_url = 'http://www.kfc.com.cn/kfccda/storelist/index.aspx'
13     post_url = 'http://www.kfc.com.cn/kfccda/ashx/GetStoreList.ashx?op=cname'
14     headers = {'User-Agent': 'Mozilla/5.0'}
15
16     # 经过分析为POST请求,故使用start_requests()方法
17     def start_requests(self):
18         """拼接多页地址,进行数据抓取"""
19         # 获取所有的城市
20         all_city = self.get_all_city()
21         for city in all_city:
22             # 获取每个城市的门店页数
23             total = self.get_total_page(city)
24             for i in range(1,total+1):
```

```

25         # 此为抓包抓到的Form表单数据
26         formdata = {
27             "cname": city,
28             "pid": "",
29             "pageIndex": str(i),
30             "pageSize": "10"
31         }
32         yield
33     scrapy.FormRequest(url=self.post_url, formdata=formdata, callback=self.parse)
34
35     def get_all_city(self):
36         """获取所有的城市列表"""
37         html = requests.get(url=self.index_url, headers=self.headers).text
38         pattern = re.compile('<a href=".*?rel="(.*?)">', re.S)
39         all_city = pattern.findall(html)
40
41         return all_city
42
43     def get_total_page(self, city):
44         """获取某个城市的肯德基总数 - 向第1页发请求即可获取"""
45         data = {
46             "cname": city,
47             "pid": "",
48             "pageIndex": "1",
49             "pageSize": "10"
50         }
51         html = requests.post(url=self.post_url, data=data, headers=self.headers).json()
52         kfc_count = html['Table'][0]['rowcount']
53         total = kfc_count//10 if kfc_count%10==0 else kfc_count//10 + 1
54
55         return total
56
57     def parse(self, response):
58         html = json.loads(response.text)
59         kfc_shop_list = html['Table1']
60         for kfc_shop in kfc_shop_list:
61             item = KfcItem()
62             item['row_num'] = kfc_shop['rownum']
63             item['store_name'] = kfc_shop['storeName']
64             item['address_detail'] = kfc_shop['addressDetail']
65             item['city_name'] = kfc_shop['cityName']
66
67         yield item

```

■ 步骤4 - 管道文件实现(pipelines.py)

```

1  # 存入MySQL管道
2  """
3  create database kfcdb charset utf8;
4  use kfcdb;
5  create table kfctab(
6  row_num int,
7  store_name varchar(100),
8  address_detail varchar(200),
9  city_name varchar(100)
10 )charset=utf8;

```

```

11 """
12 import pymysql
13 from .settings import *
14
15 class KfcMysqlPipeline(object):
16     def open_spider(self, spider):
17         self.db = pymysql.connect(MYSQL_HOST, MYSQL_USER, MYSQL_PWD, MYSQL_DB, charset=CHARSET)
18         self.cursor = self.db.cursor()
19         self.ins = 'insert into kfctab values(%s,%s,%s,%s)'
20
21     def process_item(self, item, spider):
22         shop_li = [
23             item['row_num'],
24             item['store_name'],
25             item['address_detail'],
26             item['city_name']
27         ]
28         self.cursor.execute(self.ins, shop_li)
29         self.db.commit()
30
31         return item
32
33     def close_spider(self, spider):
34         self.cursor.close()
35         self.db.close()

```

■ 步骤5 - 全局配置(settings.py)

```

1  【1】 ROBOTSTXT_OBEY = False
2  【2】 DOWNLOAD_DELAY = 0.1
3  【3】 DEFAULT_REQUEST_HEADERS = {
4      'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
5      'Accept-Language': 'en',
6      'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like
7      Gecko) Chrome/80.0.3987.149 Safari/537.36',
8  }
9  【4】 ITEM_PIPELINES = {
10     'Kfc.pipelines.KfcMysqlPipeline': 300,
11 }
12 【5】 MYSQL_HOST = 'localhost'
13 【6】 MYSQL_USER = 'root'
14 【7】 MYSQL_PWD = '123456'
15 【8】 MYSQL_DB = 'kfcdb'
16 【9】 CHARSET = 'utf8'

```

■ 步骤6 - 运行爬虫(run.py)

```

1  from scrapy import cmdline
2
3  cmdline.execute('scrapy crawl kfc'.split())

```

■ 练习

```

1  请使用scrapy框架实现有道翻译案例

```

有道翻译案例实现

■ 步骤1 - 创建项目+爬虫文件

```
1 scrapy startproject Youdao
2 cd Youdao
3 scrapy genspider youdao fanyi.youdao.com
```

■ 步骤2 - items.py

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

■ 步骤3 - youdao.py

```
1 # -*- coding: utf-8 -*-
2 import scrapy
3 import time
4 import random
5 from hashlib import md5
6 import json
7 from ..items import YoudaoItem
8
9 class YoudaoSpider(scrapy.Spider):
10     name = 'youdao'
11     allowed_domains = ['fanyi.youdao.com']
12     word = input('请输入要翻译的单词:')
13
14     def start_requests(self):
15         post_url = 'http://fanyi.youdao.com/translate_o?smartresult=dict&smartresult=rule'
16         salt, sign, ts = self.get_salt_sign_ts(self.word)
17         formdata = {
18             'i': self.word,
19             'from': 'AUTO',
20             'to': 'AUTO',
21             'smartresult': 'dict',
22             'client': 'fanyideskweb',
23             'salt': salt,
24             'sign': sign,
25             'ts': ts,
26             'bv': 'cf156b581152bd0b259b90070b1120e6',
27             'doctype': 'json',
28             'version': '2.1',
29             'keyfrom': 'fanyi.web',
30             'action': 'FY_BY_REALTIME'
31         }
32         # 发送post请求的方法
33         yield scrapy.FormRequest(url=post_url, formdata=formdata)
34
35     def get_salt_sign_ts(self, word):
36         # salt
37         salt = str(int(time.time() * 1000)) + str(random.randint(0, 9))
38         # sign
39         string = "fanyideskweb" + word + salt + "\n%A-rKaT5fb[Gy?;N5@Tj"
40         s = md5()
```

```

41         s.update(string.encode())
42         sign = s.hexdigest()
43         # ts
44         ts = str(int(time.time() * 1000))
45         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

```

■ 步骤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

```

■ 步骤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=aaa9i7p1XP1KaJH_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 }

```

■ 步骤6 - run.py

```

1 from scrapy import cmdline
2
3 cmdline.execute('scrapy crawl youdao'.split())

```

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             yield scrapy.FormRequest(url=url, formdata=formdata, cookies={}, callback=xxx)
10
11 【3】 DownloadMiddleware设置中间件
12     3.1) settings.py: COOKIES_ENABLED = TRUE # 找Request()方法中cookies参数
13     3.2) middlewares.py
14         def process_request(self, request, spider):
15             request.cookies={}

```

scrapy shell的使用

▪ 定义

```

1 【1】 调试蜘蛛的工具
2 【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

```

1 # scrapy shell URL地址
2 *1、 request.url      : 请求URL地址
3 *2、 request.headers  : 请求头(字典)
4 *3、 request.meta     : item数据传递, 定义代理(字典)
5
6 4、 response.text     : 字符串
7 5、 response.body     : bytes
8 6、 response.xpath('')
9 7、 response.status   : HTTP响应码
10
11 # 可用方法
12 shelp() : 帮助
13 fetch(request) : 从给定的请求中获取新的响应, 并更新所有相关对象
14 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 1、 url
2 2、 callback
3 3、 headers
4 4、 meta : 传递数据,定义代理
5 5、 dont_filter : 是否忽略域组限制
6     默认False,检查allowed_domains['']
7 6、 cookies

```

设置中间件(随机User-Agent)

■ 少量User-Agent切换

```
1  【1】方法一：settings.py
2      1.1) USER_AGENT = ''
3      1.2) DEFAULT_REQUEST_HEADERS = {}
4
5  【2】方法二：爬虫文件
6      yield scrapy.Request(url,callback=函数名,headers={})
```

■ 大量User-Agent切换 (middlewares.py设置中间件)

```
1  【1】获取User-Agent方式
2      1.1) 方法1：新建useragents.py,存放大量User-Agent, random模块随机切换
3      1.2) 方法2：安装fake_useragent模块(sudo pip3 install fake_useragent)
4          from fake_useragent import UserAgent
5          agent = UserAgent().random
6
7  【2】middlewares.py新建中间件类
8      class RandomUseragentMiddleware(object):
9          def process_request(self,request,spider):
10             agent = UserAgent().random
11             request.headers['User-Agent'] = agent
12
13  【3】settings.py添加此下载器中间件
14      DOWNLOADER_MIDDLEWARES = {'': 优先级}
```

设置中间件(随机代理)

```
1  class RandomProxyDownloaderMiddleware(object):
2      def process_request(self,request,spider):
3          request.meta['proxy'] = xxx
4
5      def process_exception(self,request,exception,spider):
6          return request
```

■ 练习

```
1  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  【1】URL地址: http://www.1ppt.com/xiazai/
2  【2】目标:
3      2.1) 爬取所有栏目分类下的,所有页的PPT
4      2.2) 数据存放: /home/tarena/ppts/工作总结PPT/xxx
5                  /home/tarena/ppts/个人简历PPT/xxx
6  【提示】: 使用 from scrapy.pipelines.files import FilesPipeline 管道,并重写方法
```

