1. 写出在网络爬取过程中,遇到防爬问题的解决办法.
2. 如何提高爬虫的效率?
3. 你的爬虫 爬取的数据量有多少?
4. 列举您使用过的python网络爬虫所用到的模块.
5. 简述requests模块的作用及基本使用?
6. 简述beautifulsoup模块的作用及基本使用?
7. 简述selenium模块的作用及基本使用?
8. 简述scrapy框架中各组件的工作流程?
9. 在scrapy框架中如何设置代理(两种方法)?
10. scrapy框架中如何实现大文件的下载?
11. scrapy中如何实现限速?
12. scrapy中如何实现暂定爬虫?
13. scrapy中如何进行自定制命令?
14. scrapy中如何实现的记录爬虫的深度?
15. scrapy中的pipelines工作原理?
16. scrapy的pipelines如何丢弃一个item对象?
17. 简述scrapy中爬虫中间件和下载中间件的作用?
18. scrapy-redis组件的作用?
19. scrapy-redis组件中如何实现的任务的去重?
20. scrapy-redis的调度器如何实现任务的深度优先和 广度优先?
21. 如何提升scrapy爬取数据的效率
22. scrapy编码流程
23.增量式爬虫
24.数据解析之lxml
复习题相关
1.掌握哪些基于爬虫的模块?
2.常见的数据解析方式
3.列举在爬虫过程中遇到的哪些比较难的反爬机制
4.概述如何抓取动态加载数据
5.移动端数据抓取
6.抓取过哪些类型的数据，量级多少?
7.了解哪些爬虫框架？（pyspider）
8.谈谈对scrapy的了解
9.如何解析出携带标签的局部页面数据
10.scrapy核心组件
11.scrapy中间件的应用
12.如何实现全站数据爬取
13.如何检测网站数据更新?
14.分布式实现原理
15.如何提升爬取数据的效率（异步爬虫）
16.列举你接触的反爬机制
17.什么是深度优先和广度优先（优劣）
18.scrapy如何实现持久化存储
19.谈谈对crawlspider的理解，如何使用其进行深度爬取
20.如何实现数据清洗
21.了解过机器学习吗

1. 写出在网络爬取过程中,遇到防爬问题的解决办法.

```
1.UA检测: 设置身份标识,UA伪装 headers={'User-Agent' : 'xxxx'}
2.检测ip: 代理IP proxies={'http':'60.190.250.120:8080'}
3.cookie:
    - 手动处理: 手动添加cookie,不建议,会有时效性
    - 自动处理: 会话对象session,同requests模块一样可以进行网络请求的发送,并且可以自动携带和                处理cookie.
4.限制访问频率:设置时间间隔/selenium/分布式爬虫
5:验证码: 打码平台(超级鹰, 云打码等)
6:动态加载的数据: selenium
```

2. 如何提高爬虫的效率?

```
-- 线程池
- scrapy配置文件相关配置（禁用cookie，禁止重试，减小下载超时，增加并发，日志等级）
- 分布式
- 异步爬虫
    - 多进程多线程(不建议)
    - 进程池或者线程池(适当)
    - 单线程+异步协程(推荐):
        event_loop:事件循环,将特殊的函数(协程对象)注册到事件循环中,异步执行
        coroutine:协程对象,用async关键字修饰一个普通函数,就得到一个协程对象.
        task:任务,进一步封装协程对象,包含了任务的状态
        future:和task没有本质区别,创建方法不一样而已.
        async/await 关键字:是从 Python 3.5 才出现的。其中，async 定义一个协程，await 用来挂起阻塞方法的执行。

注意: 定义协程对象时,不能出现非异步模块的代码,否则会让asyncio失去异步效果


import aiohttp  #支持异步的网络请求模块
import asyncio
#回调函数：解析响应数据
def callback(task):
    print('this is callback()')
    #获取响应数据
    page_text = task.result()
    print('在回调函数中，实现数据解析')

async def get_page(url):
    async with aiohttp.ClientSession() as session:
        async with await session.get(url=url) as response:
            page_text = await response.text() #read()  json()
            return page_text
start = time.time()
urls = [
    'http://127.0.0.1:5000/jack',
    'http://127.0.0.1:5000/jay',
    'http://127.0.0.1:5000/tom',
]
tasks = []
loop = asyncio.get_event_loop()
```

```
for url in urls:
    c = get_page(url)
    task = asyncio.ensure_future(c)
    #给任务对象绑定回调函数用于解析响应数据
    task.add_done_callback(callback)
    tasks.append(task)
loop.run_until_complete(asyncio.wait(tasks))
print('总耗时：',time.time()-start)

https://www.cnblogs.com/yaraning/p/10821186.html
```

3. 你的爬虫 爬取的数据量有多少?

```

```

4. 列举您使用过的python网络爬虫所用到的模块.

```
requests
urllib: request.urlretrieve(url=url,filename='./qiubai.jpg')
session    #作为requests中的一个方法
beautifulsoup
lxml
selenium
asyncio
aiohttp
scrapy
```

5. 简述requests模块的作用及基本使用?

```
作用：模拟浏览器发送请求
使用：
    1.指定url
    2.发送请求
        requests.get(url,params)
        requests.post(url,data)
    3.获取响应数据：text/content/json()
    4.持久化存储
```

6. 简述beautifulsoup模块的作用及基本使用?

```
作用:从HTML或XML文件中提取数据
使用：
    1.实例化一个BeautifulSoup对象,把即将被解析的页面源码加载到该对象中;
    2.调用该对象中的相关属性或方法进行标签定位和内容提取

#安装:
    pip install bs4
    pip install lxml
```

```
#实例化:
    from bs4 import BeautifulSoup
    - 本地加载: soup = BeautifulSoup(fp,'lxml')
    - 网络加载: soup = BeautifulSoup(page_text,'lxml')

#相关的属性和方法:
    - soup.tagName:定位标签(只可以定位到第一个出现的标签),返回的是一个单数
      #eg: soup.div
    - soup.find(tagName,attrName='value'):基于属性定位实现的标签定位,返回单数
      #eg: soup.find('div',class_='song') #class是关键字,所以加'_'
    - soup.find_all(): 返回一个列表
      #eg: soup.find_all('div')
    - 取文本:
      -string:取得标签中直系的文本内容
      -text/get_text():取得标签下面所有的文本内容
          #eg: soup.p.text
             # soup.p.string
             # soup.p.get_text()
    - 取属性: tagName['attrName']
      #eg: soup.a['href']
    - select: 使用选择器定位标签.返回列表
      -标签,类,id选择器:select('选择器')
          #eg: soup.select('.song')
      -层级选择器:
          -单层级: '.tang > ul > li'
              #eg: soup.select('.tang > ul > li')
          -多层级: '.tang li'
              #eg: soup.select('.tang li')
```

7. 简述selenium模块的作用及基本使用?

作用:用来完成浏览器自动化相关的操作.通过代码定制一些浏览器自动化操作,当代码执行后,浏览器就会自动触发相关的事件,爬虫中使用它主要是用来解决requests无法直接执行JavaScript代码的问题

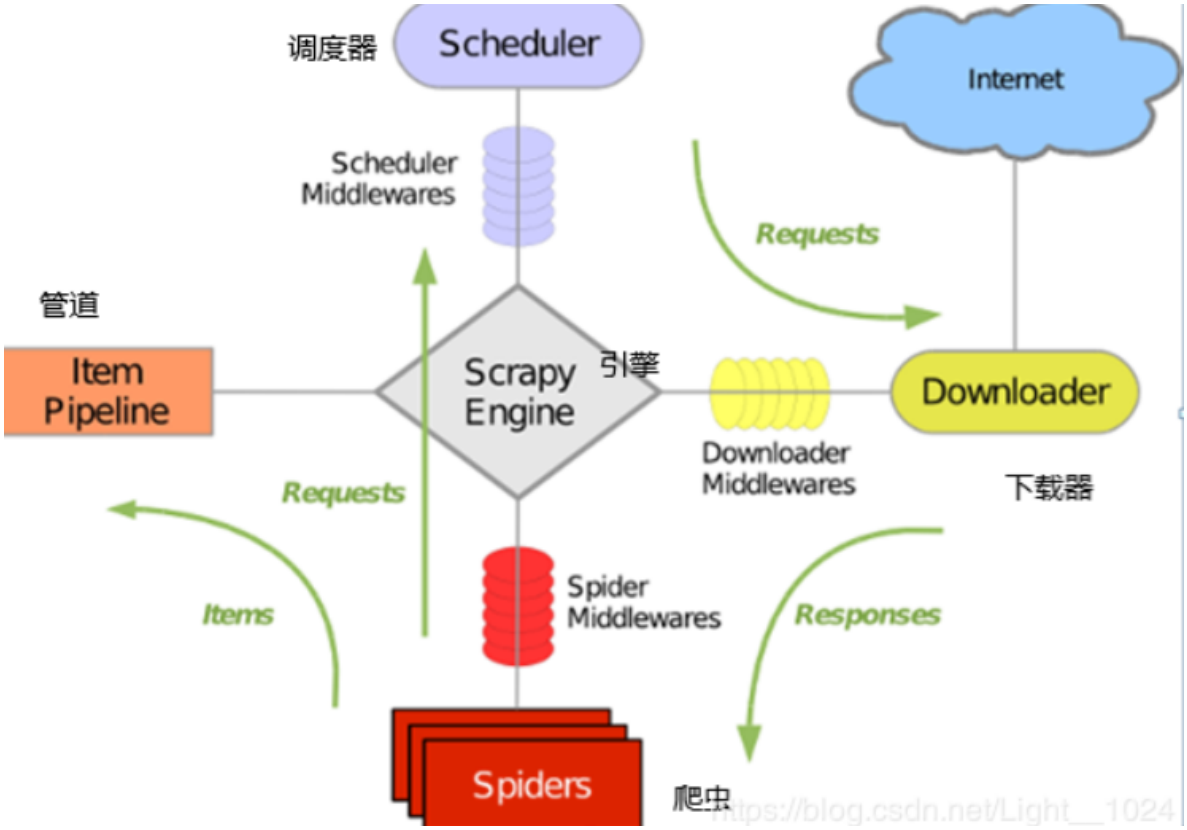
使用:

1.导包: from selenium import webdriver
2.实例化某一款浏览器对象
3.制定相关的行为动作

```
from selenium import webdriver
from time import sleep
bro = webdriver.Chrome(executable_path='./chromedriver.exe')
bro.get('https://www.baidu.com')
sleep(2)
#标签定位
tag_input = bro.find_element_by_id('kw')
tag_input.send_keys('人民币')
sleep(2)
```

```
btn = bro.find_element_by_id('su')
btn.click()
sleep(2)

bro.quit()

#元素定位:
    find_element_by_id()
    find_element_by_name()
    find_element_by_class_name()
    find_element_by_tag_name()
    find_element_by_link_text()
    find_element_by_partial_link_text()
    find_element_by_xpath()
    find_element_by_css_selector()

#谷歌无头浏览器
from selenium import webdriver
from time import sleep
from selenium.webdriver.chrome.options import Options
# 创建一个参数对象，用来控制chrome以无界面模式打开
chrome_options = Options()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--disable-gpu')
bro=webdriver.Chrome(executable_path='./chromedriver.exe',options=chrome_o
ptions)

#如果定位的标签存在于iframe标签之中，则必须经过switch_to操作在进行标签定位
bro.switch_to.frame('iframeResult')

#执行js实现滚轮向下滑动
js = 'window.scrollTo(0,document.body.scrollHeight)'
bro.execute_script(js)

#获取当前浏览器页面数据(动态)
print(bro.page_source)

https://www.cnblogs.com/bobo-zhang/p/9685362.html
```

8. 简述scrapy框架中各组件的工作流程?



#引擎(Scrapy)
用来处理整个系统的数据流处理，触发事务(框架核心)
#调度器(Scheduler)
用来接受引擎发过来的请求，压入队列中，并在引擎再次请求的时候返回. 可以想像成一个URL（抓取网页的网址或者说是链接）的优先队列，由它来决定下一个要抓取的网址是什么，同时去除重复的网址
#下载器(Downloader)
用于下载网页内容，并将网页内容返回给蜘蛛(Scrapy下载器是建立在twisted这个高效的异步模型上的)
#爬虫(Spiders)
爬虫是主要干活的，用于从特定的网页中提取自己需要的信息，即所谓的实体(Item)。用户也可以从中提取出链接,让Scrapy继续抓取下一个页面
#项目管道(Pipeline)
负责处理爬虫从网页中抽取的实体，主要的功能是持久化实体、验证实体的有效性、清除不需要的信息。当页面被爬虫解析后，将被发送到项目管道，并经过几个特定的次序处理数据。

持久化存储的实现流程（基于管道）：
- 数据解析
- 封装item类
- 实例化item类型的对象
- 将解析到的数据依次存储封装到item类型的对象中
- 将item提交给管道
- 在管道中实现io操作
- 开启管道

9. 在scrapy框架中如何设置代理(两种方法)?

```
#方式一: 在下载中间件中配置
class Proxy(object):
```

```
    def process_request(self, request, spider):
        #对拦截到请求的url进行判断（协议头到底是http还是https）
        #request.url返回值：http://www.xxx.com
        h = request.url.split(':')[0]  #请求的协议头
        if h == 'https':
            ip = random.choice(PROXY_https)
            request.meta['proxy'] = 'https://'+'153.180.102.104:80'
        else:
            ip = random.choice(PROXY_http)
            request.meta['proxy'] = 'http://' + '120.83.49.90:9000'
#方式二:直接在爬虫程序中设置proxy字段
yield scrapy.Request(url=url, callback=self.parse, meta={'proxy':
'http://proxy.yourproxy:8001'})
```

10. scrapy框架中如何实现大文件的下载?

利用scrapy下载大量大尺寸图片及视频时有时会报错，显示放弃重试，用户连接时间超过180s导致失去连接.
这是由于scrapy并发请求过多，默认情况下会同时下载16个文件，而连接时间默认超过三分钟就会丢失。就是说如果三分钟之内你的网速没法支持你同时下载完16个文件的话就会造成这个问题。

解决方法就是在settings.py中将默认并发连接数调小或者将timeout时间调大
CONCURRENT_REQUESTS = 2
DOWNLOAD_TIMEOUT=1800

此时并发请求被调成2， 等待时间被1800秒，一般的小视频和图片是没有问题了。

11. scrapy中如何实现限速?

爬虫爬取速度过快，会对网站服务器造成很大的压力，因此很容易会被判断为爬虫，自动限速可以限制爬虫的速度，对网站服务器更友好，并且不容易被反爬。

自动限速设定：

在setting.py开启相关扩展：
AUTOTHROTTLE_ENABLED = True
设定爬取速度：
DOWNLOAD_DELAY = 1 #单位为秒

12. scrapy中如何实现暂定爬虫?

1、首先cd进入到scrapy项目里
2、在scrapy项目里创建保存记录信息的文件夹
3、执行命令：

```
scrapy crawl 爬虫名称 -s JOBDIR=保存记录信息的路径
```

如：scrapy crawl cnblogs -s JOBDIR=zant/001

执行命令会启动指定爬虫，并且记录状态到指定目录

爬虫已经启动，我们可以按键盘上的ctrl+c停止爬虫，停止后我们看一下记录文件夹，会多出3个文件，其中的requests.queue文件夹里的p0文件就是URL记录文件，这个文件存在就说明还有未完成的URL，当所有URL完成后会自动删除此文件

当我们重新执行命令：scrapy crawl cnblogs -s JOBDIR=zant/001 时爬虫会根据p0文件从停止的地方开始继续爬取。

13. scrapy中如何进行自定制命令?

https://www.cnblogs.com/yanlin-10/p/9842729.html

14. scrapy中如何实现的记录爬虫的深度?

方法一：基于Scrapy框架中的Spider的递归爬取进行实现（Request模块递归回调parse方法）。

```
-----------------------------------------------------------------------------------
#爬取多页
pageNum = 1 #起始页码
url = 'https://www.qiushibaike.com/text/page/%s/' #每页的url

def parse(self, response):
    ...
    #爬取所有页码数据
    if self.pageNum <= 13: #一共爬取13页（共13页）
        self.pageNum += 1
        url = format(self.url % self.pageNum)

        #递归爬取数据：callback参数的值为回调函数（将url请求后，得到的相应数据继续进行parse解析），递归调用parse函数
        yield scrapy.Request(url=url,callback=self.parse)
-----------------------------------------------------------------------------------
```

方法二：基于CrawlSpider的自动爬取进行实现（更加简洁和高效）。

```
-----------------------------------------------------------------------------------
#创建工程
scrapy startproject 工程名
cd 工程名
scrapy genspider -t crawl 爬虫文件名 www.xxx.com

#爬虫文件:
class ProSpider(CrawlSpider):
    name = 'pro'
    allowed_domains = ['www.xxx.com']
```

```
    start_urls = ['http://www.xxx.com/']

    #follow:将链接提取器,继续作用到,链接提取器提取的链接,所对应的页面源码中
    rules = (
        Rule(LinkExtractor(allow=r'Items/'), callback='parse_item',
follow=True),
        #可以有多个Rule()
    )

    def parse_item(self, response):
        item = {}
        #item['domain_id'] =
response.xpath('//input[@id="sid"]/@value').get()
        #item['name'] = response.xpath('//div[@id="name"]').get()
        #item['description'] =
response.xpath('//div[@id="description"]').get()
        return item
--------------------------------------------------------------------------
-------
LinkExtractor(
    allow=r'Items/', # 满足括号中“正则表达式”的值会被提取，如果为空，则全部匹配。
    deny=xxx,  # 满足正则表达式的则不会被提取。
    restrict_xpaths=xxx, # 满足xpath表达式的值会被提取
    restrict_css=xxx, # 满足css表达式的值会被提取
    deny_domains=xxx, # 不会被提取的链接的domains。
)
```

15. scrapy中的pipelines工作原理?

```
items.py：数据结构模板文件。定义数据属性。
pipelines.py：管道文件。接收数据（items），进行持久化操作。

持久化流程：
1.爬虫文件爬取到数据后，需要将数据封装到items对象中。
2.使用yield关键字将items对象提交给pipelines管道进行持久化操作。
3.在管道文件中的process_item方法中接收爬虫文件提交过来的item对象，然后编写持久化存储的
代码将item对象中存储的数据进行持久化存储
4.settings.py配置文件中开启管道

#爬虫文件：
def parse(self, response):
    div_list=response.xpath('//*[@id="content-left"]/div')
    for div in div_list:

 content=div.xpath('.//div[@class="content"]/span/text()').extract_first()
        content=content.strip('\n')
        item=QiushibaikeItem()
        item['content']=content

        yield item #提交item到管道进行持久化
```

```
#items.py中定义如下：
class SecondbloodItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    content = scrapy.Field() #存储段子内容


#pipelines.py中代码如下：
class SecondbloodPipeline(object):
    #构造方法
    def __init__(self):
        self.fp = None  #定义一个文件描述符属性
   #下列都是在重写父类的方法：
    #开始爬虫时，执行一次
    def open_spider(self,spider):
        print('爬虫开始')
        self.fp = open('./data.txt', 'w')

    #因为该方法会被执行调用多次，所以文件的开启和关闭操作写在了另外两个只会各自执行一次的方法中。
    def process_item(self, item, spider):
        #将爬虫程序提交的item进行持久化存储
        self.fp.write(item['author'] + ':' + item['content'] + '\n')
        return item

    #结束爬虫时，执行一次
    def close_spider(self,spider):
        self.fp.close()
        print('爬虫结束')
```

16. scrapy的pipelines如何丢弃一个item对象?

```
通过raise DropItem()方法

from scrapy.exceptions import DropItem

class DuplicatesPipeline(object):

    def __init__(self):
        self.ids_seen = set()

    def process_item(self, item, spider):
        if item['id'] in self.ids_seen:
            raise DropItem("Duplicate item found: %s" % item)
        else:
            self.ids_seen.add(item['id'])
            return item
```

17. 简述scrapy中爬虫中间件和下载中间件的作用?

爬虫中间件:爬虫中间件使用方法和下载中间件相同，且功能重复，常使用下载中间件
下载中间件:处理请求和响应
（1）引擎将请求传递给下载器过程中， 下载中间件可以对请求进行一系列处理。比如设置请求的 User-Agent，设置代理等

（2）在下载器完成将Response传递给引擎中，下载中间件可以对响应进行一系列处理。比如进行gzip解压,篡改响应数据等。

18. scrapy-redis组件的作用?

可以给原生的scrapy提供可以被共享的管道和调度器,从而实现分布式爬虫

19. scrapy-redis组件中如何实现的任务的去重?

```
# 增加了一个去重容器类的配置，作用使用Redis的set集合来存储请求的指纹数据，从而实现请求去重的持久化
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
# 使用scrapy-redis组件自己的调度器
SCHEDULER = "scrapy_redis.scheduler.Scheduler"
# 配置调度器是否要持久化，也就是当爬虫结束了，要不要清空Redis中请求队列和去重指纹的set。如果是True，就表示要持久化存储，就不清空数据，否则清空数据
SCHEDULER_PERSIST = True
```

20. scrapy-redis的调度器如何实现任务的深度优先和 广度优先?

```
DEPTH_PRIORITY = 1 #广度优先
DEPTH_PRIORITY = -1 # 深度优先
# 广度优先
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.FifoQueue'
# 深度优先
SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.LifoQueue'
```

21. 如何提升scrapy爬取数据的效率

#增加并发：
默认scrapy开启的并发线程为32个，可以适当进行增加。在settings配置文件中修改CONCURRENT_REQUESTS = 100值为100,并发设置成了为100。

#降低日志级别：
在运行scrapy时，会有大量日志信息的输出，为了减少CPU的使用率。可以设置log输出信息为INFO或者ERROR即可。在配置文件中编写：LOG_LEVEL = ‘INFO’

#禁止cookie：
如果不是真的需要cookie，则在scrapy爬取数据时可以禁止cookie从而减少CPU的使用率，提升爬取效率。在配置文件中编写：COOKIES_ENABLED = False

#禁止重试：
对失败的HTTP进行重新请求（重试）会减慢爬取速度，因此可以禁止重试。在配置文件中编写：RETRY_ENABLED = False

#减少下载超时：

如果对一个非常慢的链接进行爬取，减少下载超时可以能让卡住的链接快速被放弃，从而提升效率。在配置文件中进行编写：DOWNLOAD_TIMEOUT = 10 超时时间为10s

22. scrapy编码流程

- 搭建流程：
  - 创建工程
  - 爬虫文件
  - 修改爬虫文件：
    - 导包：from scrapy_redis.spiders import RedisCrawlSpider

      '''将爬虫类的父类修改成基于RedisSpider或者RedisCrawlSpider。注意：如果原始爬虫文件是基于Spider的，则应该将父类修改成RedisSpider，如果原始爬虫文件是基于CrawlSpider的，则应该将其父类修改成RedisCrawlSpider'''
    - 将当前爬虫类的父类进行修改RedisCrawlSpider
    - allowed_domains，start_url删除，添加一个新属性redis_key(调度器队列的名称)
    - 数据解析，将解析的数据封装到item中然后向管道提交
  - 配置文件的编写：
    - 指定管道：

      ```
      ITEM_PIPELINES = {
          'scrapy_redis.pipelines.RedisPipeline': 400
      }
      ```
    - 指定调度器：

      ```
      # 增加了一个去重容器类的配置，作用使用Redis的set集合来存储请求的指纹数据，从而实现请求去重的持久化
      DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
      # 使用scrapy-redis组件自己的调度器
      SCHEDULER = "scrapy_redis.scheduler.Scheduler"
      # 配置调度器是否要持久化，也就是当爬虫结束了，要不要清空Redis中请求队列和去重指纹的set。如果是True，就表示要持久化存储，就不清空数据，否则清空数据
      SCHEDULER_PERSIST = True
      ```
    - 指定具体的redis：

      ```
      REDIS_HOST = 'redis服务的ip地址'
      REDIS_PORT = 6379
      REDIS_ENCODING = ‘utf-8’
      REDIS_PARAMS = {‘password’:’xxx’}
      ```
    - 对redis的配置文件进行适当的配置：
      - #bind 127.0.0.1
      - protected-mode no
    - 开启redis服务(携带redis的配置文件：redis-server ./redis.windows.conf)
  - 启动程序：scrapy runspider xxx.py
  - 向调度器队列中扔入一个起始的url（redis的客户端）：lpush xxx www.xxx.com
    - xxx表示的就是redis_key的属性值

23.增量式爬虫

- 概念：用来《检测》网站数据更新的情况。只会爬取网站中更新出来的新数据。

- 核心：去重
    -url去重
    -数据去重

```
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule
from redis import Redis
from moviePro.items import MovieproItem
class MovieSpider(CrawlSpider):
    conn = Redis(host='127.0.0.1',port=6379)
    name = 'movie'
    # allowed_domains = ['www.xxx.com']
    start_urls = ['https://www.4567tv.tv/frim/index1.html']

    rules = (
        Rule(LinkExtractor(allow=r'/frim/index1-\d+\.html'),
callback='parse_item', follow=True),
    )

    def parse_item(self, response):
        #解析出当前页码对应页面中电影详情页的url
        li_list = response.xpath('//div[@class="stui-pannel_bd"]/ul/li')
        for li in li_list:
            #解析详情页的url
            detail_url =
'https://www.4567tv.tv'+li.xpath('./div/a/@href').extract_first()
  ----------------------------------------重点-------------------------
  ----------------------------------------重点-------------------------
  ----------------------------------------重点-------------------------
            #ex == 1:该url没有被请求过  ex == 0:该url已经被请求过了
            ex = self.conn.sadd('movie_detail_urls',detail_url)
            if ex == 1:
                print('有新数据可爬取......')
                yield
scrapy.Request(url=detail_url,callback=self.parse_detail)
            else:
                print('暂无新数据可爬取！')
```

24.数据解析之lxml

- 通用性比较强
- 环境的安装：pip install lxml
- 解析原理：
    - 1.实例化一个etree对象，且将解析的页面源码加载到该对象中
    - 2.使用该对象中的xpath方法结合着xpath表达式进行标签定位和数据解析提取
- etree对象的实例化：
    - 本地加载：
        tree = etree.parse('filePath')
    - 网络加载：
        tree = etree.HTML(page_text)

常用的xpath表达式:基于标签的层级实现定位. 返回的永远是一个列表
- /:从根标签开始实现层级定位
- //:从任意位置实现标签的定位
- 属性定位：tag[@attrName="attrValue"]
- 索引定位：//div[@class="tang"]/ul/li[5] 索引值是从1开始
- 取文本：
    -取得直系文本内容：/text()
    -取得所有的文本内容：//text()
- 取属性：/@attrName

复习题相关

1.掌握哪些基于爬虫的模块?

```
-urllib
-requests模块重点：
    -发起请求的参数
        -url
        -headers
        -data/prame
        -proxies={"http://":"ip:port"}
    - get请求，post请求，ajax的get，ajax的post.
    - 获取响应数据：
        -text
        -content
        -json()
        -encoding
    -requests处理cookie:
        -手动处理
        -自动处理 session = requests.Session()
        #参照雪球网    https://www.cnblogs.com/Bottle-
cap/articles/10817312.html
    -如何提升requests爬取数据的效率
        -多线程
        -多进程
    - 单线程+异步协程
        - 事件循环：#将协程对象注册到事件循环中，然后启动事件循环对象
        - 协程对象：#async修饰的函数的定义，函数调用后会返回一个协程对象
        - 任务对象：#就协程进行进一步的封装，封装到了task对象中
task=loop.create_task(c)
        - await,async ：#挂起某一个协程，async修饰的函数的定义
```

2.常见的数据解析方式

```
##数据解析的原理
实现标签定位
将标签中存储的文本内容或者相关的属性值进行提取

-正则
-xpath
-bs4
-pyquery
```

3.列举在爬虫过程中遇到的哪些比较难的反爬机制

```
  -robots.txt
  -UA检测
  -验证码
  -cookie
   -检测IP(代理)
  -动态参数（token）    ##考试中有一个案例用到这个，和session
   -动态加载的数据
   -图片懒加载
   -数据加密
```

4.概述如何抓取动态加载数据

```
-ajax动态请求的
-js动态生成的（selenium）
```

5.移动端数据抓取

```
-fidller
-青花瓷（过时）
-mitproxy
#如果抓取https页面的数据，需要进行证书的配置
```

6.抓取过哪些类型的数据，量级多少?

```
-新闻资讯（体育）
-财经数据（股票，金融产品）    ##去找去练习
-设备参数（医疗设备，硬件设备的参数）
-购物平台的某指定类型的商品信息
-机票

量级多少？
-requests单线程爬取200w条数据 => 耗时1个小时     一天10万条左右（数据清洗之后）
```

7.了解哪些爬虫框架？（pyspider）

```
-scrapy
-pyspider（具有可视化界面，可以看到爬到的数据，功能没有scrapy多）
```

8.谈谈对scrapy的了解

-高效的网络请求（下载）
-高性能的持久化存储，数据解析
-中间件（下载中间件）拦截请求和响应

9.如何解析出携带标签的局部页面数据

-使用bs4

10.scrapy核心组件

-引擎(Scrapy)
用来处理整个系统的数据流处理，触发事务(框架核心)

-调度器(Scheduler)
用来接受引擎发过来的请求，压入队列中，并在引擎再次请求的时候返回．可以想像成一个URL（抓取网页的网址或者说是链接）的优先队列，由它来决定下一个要抓取的网址是什么，同时去除重复的网址　　（队列，过滤器）

-下载器(Downloader)
用于下载网页内容，并将网页内容返回给蜘蛛(Scrapy下载器是建立在twisted这个高效的异步模型上的)

-爬虫(Spiders)
爬虫是主要干活的，用于从特定的网页中提取自己需要的信息，即所谓的实体(Item)。用户也可以从中提取出链接,让Scrapy继续抓取下一个页面

-项目管道(Pipeline)
负责处理爬虫从网页中抽取的实体，主要的功能是持久化实体、验证实体的有效性、清除不需要的信息。当页面被爬虫解析后，将被发送到项目管道，并经过几个特定的次序处理数据。

11.scrapy中间件的应用

IP代理，UA，响应数据满足不了我们的需求

12.如何实现全站数据爬取

-Splider.Request
-CrowlSplider

13.如何检测网站数据更新?

-增量式爬虫（数据，URL去重）

14.分布式实现原理

scrapy-redis组件

15.如何提升爬取数据的效率（异步爬虫）

```
##增加并发：
默认scrapy开启的并发线程为32个，可以适当进行增加。在settings配置文件中修改
CONCURRENT_REQUESTS = 100值为100,并发设置成了为100。

##降低日志级别：
    在运行scrapy时，会有大量日志信息的输出，为了减少CPU的使用率。可以设置log输出信息为
INFO或者ERROR即可。在配置文件中编写：LOG_LEVEL = ‘INFO’

##禁止cookie：
    如果不是真的需要cookie，则在scrapy爬取数据时可以禁止cookie从而减少CPU的使用率，
提升爬取效率。在配置文件中编写：COOKIES_ENABLED = False

##禁止重试：
    对失败的HTTP进行重新请求（重试）会减慢爬取速度，因此可以禁止重试。在配置文件中编写：
RETRY_ENABLED = False

##减少下载超时：
    如果对一个非常慢的链接进行爬取，减少下载超时可以能让卡住的链接快速被放弃，从而提升效
率。在配置文件中进行编写：DOWNLOAD_TIMEOUT = 10 超时时间为10s

##DOWNLOAF_DELAY = 3(下载延迟，调度器调度url时，每隔三秒发送一次)
```

16.列举你接触的反爬机制

```
 -robots.txt
-UA检测
-验证码
-cookie
 -检测IP(代理)
-动态参数（token）    ##考试中有一个案例用到这个，和session
 -动态加载的数据
 -图片懒加载
 -数据加密
```

17.什么是深度优先和广度优先（优劣）

18.scrapy如何实现持久化存储

```
-基于终端指令的持久化存储
    scrapy crawl 爬虫名称 -o xxx.json
    scrapy crawl 爬虫名称 -o xxx.xml
    scrapy crawl 爬虫名称 -o xxx.csv
-基于管道的持久化存储
    持久化流程：
    1.数据解析
    2.封装item类
```

3.实例化item类型的对象
4.将解析到的数据依次存储封装到item类型的对象中
5.使用yield关键字将items对象提交给pipelines管道进行持久化操作。
6.在管道文件中的process_item方法中接收爬虫文件提交过来的item对象，然后编写持久化存储的代码将item对象中 存储的数据进行持久化存储
7.settings.py配置文件中开启管道

19.谈谈对crawlspider的理解，如何使用其进行深度爬取

20.如何实现数据清洗

-清洗空值
    -dropna
    -fillna
-清洗重复值
    -drop_duplicates(keep=)
-清洗异常值
    -指定一个判断异常值的条件

21.了解过机器学习吗

-sklearn,仅限于应用层