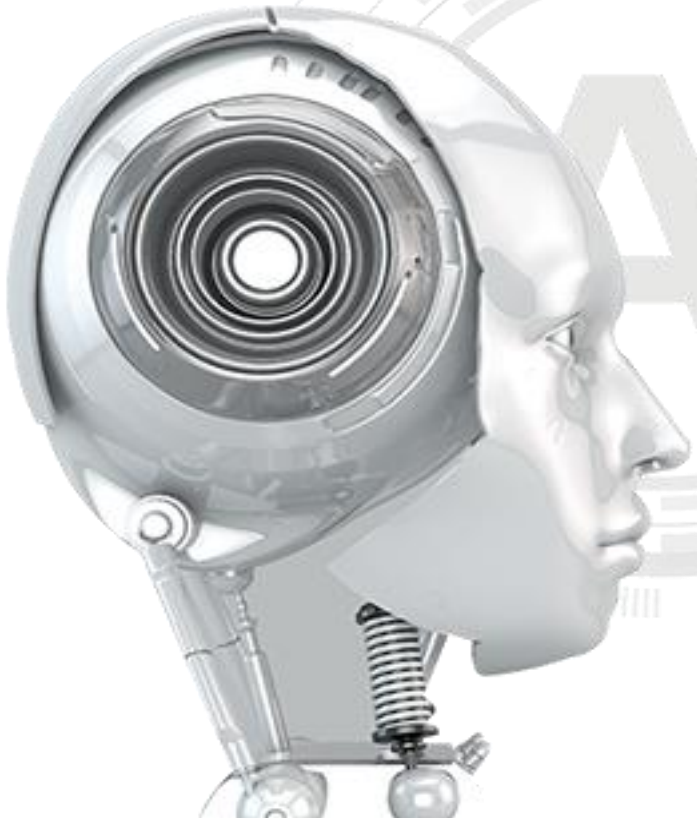
Python高级-爬虫



目录

1、基础知识

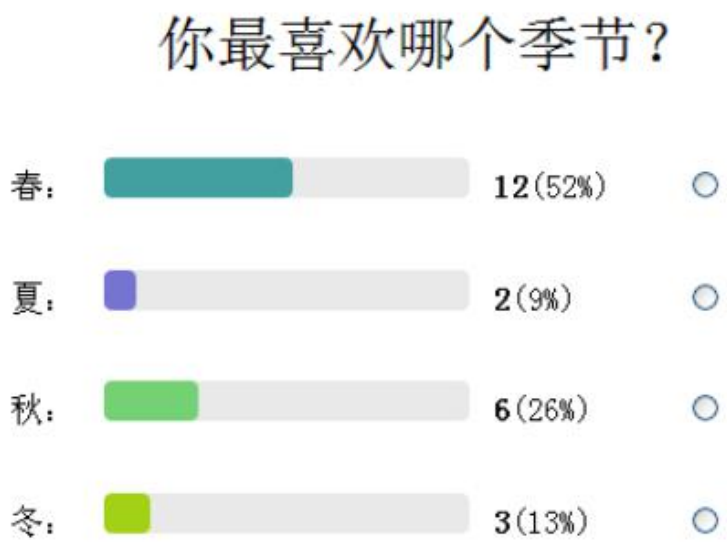
为什么要学习爬虫&爬虫的定义

- 网络爬虫又称网络蜘蛛、网络机器人等，可以自动化浏览网络中的信息，当然浏览信息的时候需要按照所制定的相应规则进行，即网络爬虫算法。
- 原因很简单，我们可以利用爬虫技术，自动地从互联网中获取我们感兴趣的内容，并将这些数据内容爬取回来，作为我们的数据源，从而进行更深层次的数据分析，并获得更多有价值的信息。
- 在大数据时代，这一技能是必不可少的。
- 只要是浏览器能做的事情，原则上，爬虫都能够做。

爬虫的应用场景

12306抢票

网站上的投票



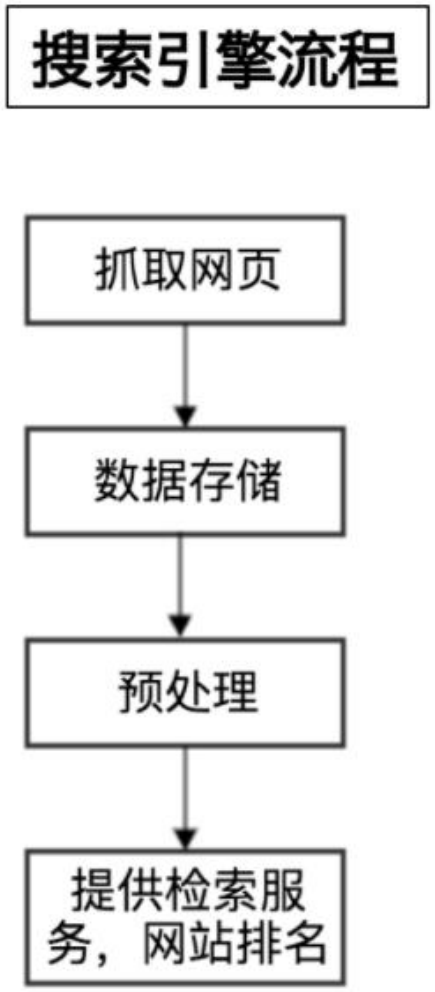
爬虫的思路

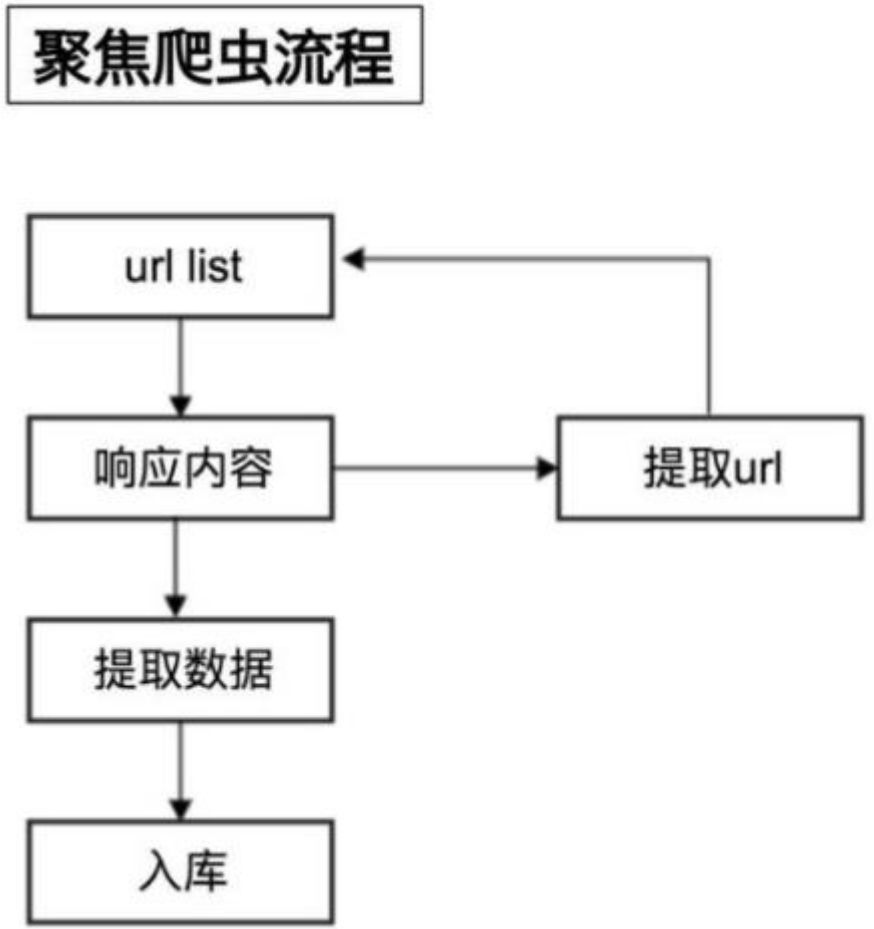
每一个网页都是一份 HTML文档，全称叫 hypertext markup language，是一种文本标记语言，他长的就像这样：

```
html
    head
        title    title
     head
    body
        h1        h1
        img src "xxx"
     body
 html
```

这是一份很有规则的文档写法
打开一个网页，即是通过了 HTTP协议，对一个资源进行了请求，返还你一份 HTML文档，然后浏览器进行文档的渲染，这样，就形成了一个网页
所以，我们只需要模拟浏览器，发送一份请求，获得这份文档，再抽取出我们需要的内容就好

通用爬虫和聚焦爬虫工作流程





通用搜索引擎的局限性

- 通用搜索引擎所返回的网页里90%的内容无用。
- 图片、音频、视频多媒体的内容通用搜索引擎无能为力。
- 不同用户搜索的目的不全相同，但是返回内容相同

ROBOTS协议

例如： https://www.taobao.com/robots.txt

Robots协议是Web站点和搜索引擎爬虫交互的一种方式，Robots.txt是存放在站点根目录下的一个纯文本文件。该文件可以指定搜索引擎爬虫只抓取指定的内容，或者是禁止搜索引擎爬虫抓取网站的部分或全部内容。当一个搜索引擎爬虫访问一个站点时，它会首先检查该站点根目录下是否存在robots.txt，如果存在，搜索引擎爬虫就会按照该文件中的内容来确定访问的范围；如果该文件不存在，那么搜索引擎爬虫就沿着链接抓取。

另外，robots.txt必须放置在一个站点的根目录下，而且文件名必须全部小写。如果搜索引擎爬虫要访问的网站地址是http://www.w3.org/，那么robots.txt文件必须能够通过http://www.w3.org/robots.txt打开并看到里面的内容。

ROBOTS协议

具体使用格式如下:

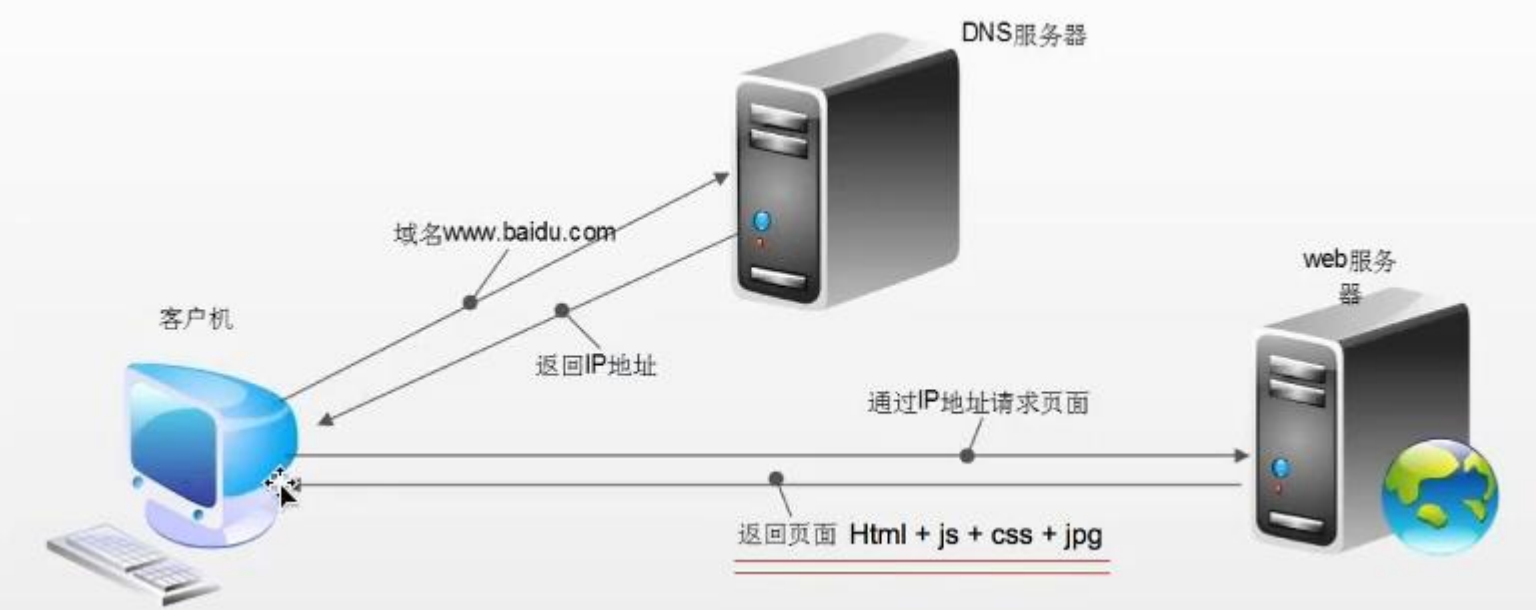
(1)User.agent:用于描述搜索引擎爬虫的名字。在Robots.txt文件中，如果有多条User-agent记录，说明有多个搜索引擎爬虫会受到该协议的限制，对该文件来说，至少要有一条User-agent记录。如果该项的值设为空，则该协议对任何搜索引擎爬虫均有效，在Robots.txt文件中，“User-agent:*这样的记录只能有一条。

(2)Disallow:用于描述不希望被访问到的一个URL。这个URL可以是一条完整的路径，也可以是部分路径，任何以Disallow开头的URL均不会被Robot访问到。

搜索引擎爬虫必须要遵守Robots协议并执行Web站点的要求。因此搜索引擎爬虫需要有一个分析Robots协议的模块，并严格按照Robots协议的规定抓取Web主机允许访问的目录和网页。

当然，Robots.txt只是一个协议，如果搜索引擎爬虫的设计者不遵循这个协议，网站管理员也无法阻止搜索引擎爬虫对于某些页面的访问，但一般的搜索引擎爬虫都会遵循这些协议，而且网站管理员还可以通过其他方式来拒绝网络蜘蛛对某些网页的抓取。

浏览器发送HTTP请求的过程



浏览器渲染出来的页面和爬虫请求的页面并不一样

2、requests

为什么要学习requests，而不是urllib?

1.requests的底层实现就是urllib。

2.requests在python2和python3中通用，方法完全一样。

3.requests简单易用。

4.requests能够自动帮我们解压（gzip压缩等）网页内容。

requests的作用

作用：发送网络请求，返回响应数据

中文文档API：

https://requests-docs-cn.readthedocs.io/zh_CN/latest/_modules/requests/api.html

requests的用法

- requests是python实现的简单易用的HTTP库，使用起来比urllib简洁很多

  因为是第三方库，所以使用前需要cmd安装　　pip install requests

- 基本用法：requests.get()用于请求目标网站，类型是一个HTTPresponse类型

```
import requests

response = requests.get('http://www.baidu.com')
print(response.status_code)  # 打印状态码
#常见状态码https://www.runoob.com/http/http-status-codes.html
print(response.url)          # 打印请求url
print(response.headers)      # 打印头信息
print(response.cookies)      # 打印cookie信息
print(response.text)  #以文本形式打印网页源码
print(response.content) #以字节流形式打印
```

各种请求方式：

```
import requests

requests.get('http://httpbin.org/get')
requests.post('http://httpbin.org/post')
requests.put('http://httpbin.org/put')
requests.delete('http://httpbin.org/delete')
requests.head('http://httpbin.org/get')
requests.options('http://httpbin.org/get')
```

基本的get请求

```
import requests

response = requests.get('http://httpbin.org/get')
print(response.text)
```

带参数的GET请求：

- **1.直接将参数放在url内：**

```
import requests

response = requests.get('http://httpbin.org/get?name=gemey&age=22')
print(response.text)
```

- **2.先将参数填写在dict中，发起请求时params参数指定为dict**

```
import requests

data = { 'name': 'tom', 'age': 20}

response = requests.get('http://httpbin.org/get', params=data)
print(response.text)
```

解析json

```
import requests

response = requests.get('http://httpbin.org/get')
print(response.text)
print(type(response.text))
print(response.json())  #response.json()方法同json.loads(response.text)
print(type(response.json()))
```

简单保存一个二进制文件

二进制内容为response.content

```
import requests

response = requests.get('http://img.ivsky.com/img/tupian/pre/201708/30/kekeersitao-002.jpg')
b = response.content
with open('fengjing.jpg','wb') as f:
    f.write(b)
```

response.text和response.content的区别

- **response.text**

  类型：str

  解码类型：根据HTTP头部对响应的编码做出有根据的推测，推测的文本编码

  如何修改编码方式：response.encoding="gbk"

- **response.content**

  类型：bytes

  解码类型：没有指定

  如何修改编码方式：response.content.decode("utf8")

为你的请求添加头信息

```
import requests
heads = {}
heads['User-Agent'] = 'Mozilla/5.0 ' \
                      '(Macintosh; U; Intel Mac OS X 10_6_8; en-us) AppleWebKit/534.50 ' \
                      '(KHTML, like Gecko) Version/5.1 Safari/534.50'
 response = requests.get('http://www.baidu.com',headers=headers)
```

使用代理

同添加headers方法，代理参数也要是一个dict,这里使用requests库爬取了IP代理网站的IP与端口和类型因为是免费的，使用的代理地址很快就失效了

```
import requests
import re
def get_html(url):
    proxy = { 'http': '120.25.253.234:812',
        'https': '163.125.222.244:8123'}
    heads = {}
    heads['User-Agent'] = 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.221 Safari/537.36 SE 2.X MetaSr 1.0'
    req = requests.get(url, headers=heads,proxies=proxy)
    html = req.text
    return html
def get_ipport(html):
    regex = r'<td data-title="IP">(.+)</td>'
    iplist = re.findall(regex, html)
    regex2 = '<td data-title="PORT">(.+)</td>'
    portlist = re.findall(regex2, html)
    regex3 = r'<td data-title="类型">(.+)</td>'
    typelist = re.findall(regex3, html)
    sumray = []
    for i in iplist:
        for p in portlist:
            for t in typelist:
                pass
            pass
        a = t+','+i + ':' + p
        sumray.append(a)
    print('高匿代理')
    print(sumray)
if __name__ == '__main__':
    url = 'http://www.kuaidaili.com/free/'
    get_ipport(get_html(url))
```

基本POST请求：

```
import requests

data = {'name':'tom','age':'22'}

response = requests.post('http://httpbin.org/post', data=data)
```

获取cookie

```
#获取cookie
import requests

response = requests.get('http://www.baidu.com')
print(response.cookies)
print(type(response.cookies))
for k,v in response.cookies.items():
    print(k+':'+v)
```

结果:

```
<RequestsCookieJar[<Cookie BDORZ=27315 for .baidu.com/>]>
<class 'requests.cookies.RequestsCookieJar'>
BDORZ:27315
```

cookie和session的区别

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗
考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能，
考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

处理cookie和session

1.带上cookie和session的好处：

能够请求到登陆后的页面

2.带上cookie和session的弊端：

一套cookie和session往往对应一个用户，请求太快

请求次数太多，容易被识别为爬虫

不需要cookie的时候尽量不去使用cookie

但是有时为了获取登陆的页面，必须发送带有cookie的请求

处理cookie和session

- **requests提供了一个session类，来实现客户端和服务器端的会话保持**
- **使用的方法**:

  1.实例化一个session对象
  2.让session来发送get或post请求

  ```
  session=requests.session()
  response=session.get(url,headers)
  ```

会话维持

```
import requests

session = requests.Session()
session.get('http://httpbin.org/cookies/set/number/12345')
response = session.get('http://httpbin.org/cookies')
print(response.text)
```

结果:


```
{
  "cookies": {
    "number": "12345"
  }
}
```

证书验证设置

```
import requests
from requests.packages import urllib3

urllib3.disable_warnings()  #从urllib3中消除警告
response = requests.get('https://www.12306.cn',verify=False)  #证书验证设为FALSE
print(response.status_code)
```

打印结果：200

超时异常捕获

```
import requests
from requests.exceptions import ReadTimeout

try:
    res = requests.get('http://httpbin.org', timeout=0.1)
    print(res.status_code)
except ReadTimeout:
    print(timeout)
```

异常处理

在你不确定会发生什么错误时，尽量使用try...except来捕获异常

所有的requests exception：

```
import requests
from requests.exceptions import ReadTimeout,HTTPError,RequestException

try:
    response = requests.get('http://www.baidu.com',timeout=0.5)
    print(response.status_code)
except ReadTimeout:
    print('timeout')
except HTTPError:
    print('httperror')
except RequestException:
    print('reqerror')
```

3、数据提取

什么是数据提取？

什么是数据提取？

简单的来说，数据提取就是从响应中获取我们想要的数据的过程。

数据分类

非结构化的数据：html等

处理方法：正则表达式、xpath

结构化数据：json，xml等

处理方法：转化为python数据类型

数据提取之JSON

- **1.为什么要学习json?**

  由于把json数据转化为python内建数据类型很简单，所以爬虫中，如果我们能够找到返回json数据的URL，就会尽量使用这种URL，而很多地方也都会返回json

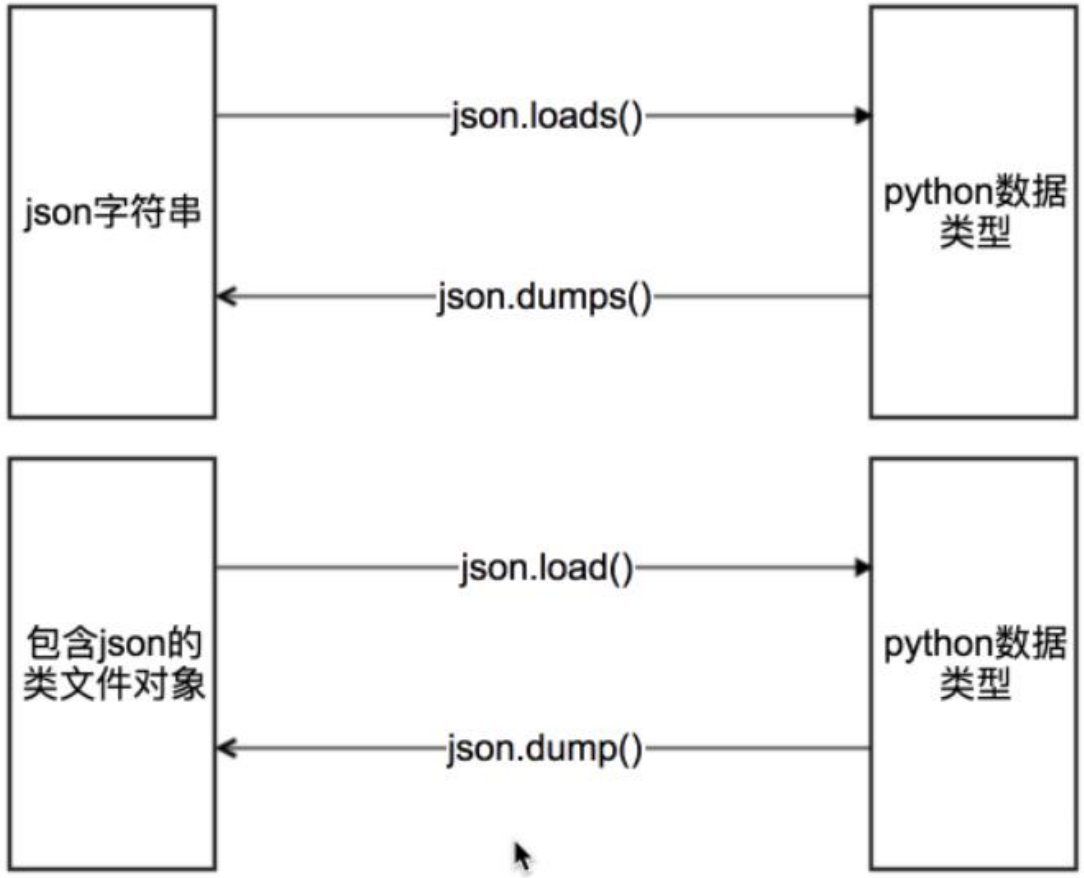
- **2. 什么是json?**

  JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，它使得人们很容易的进行阅读和编写。同时也方便了机器进行解析和生成。适用于进行数据交互的场景，比如网站前台与后台之间的数据交互。

  3.哪里能找到返回json的url?

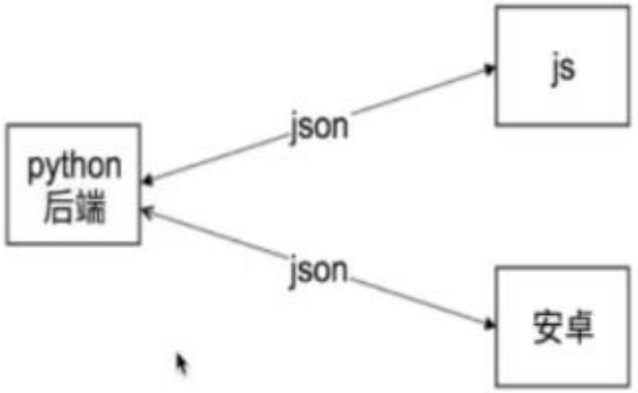
  1.使用chrome切换到手机页面；
  2.抓包手机app的软件

数据提取之JSON



具有read()或者write()方法的对象就是类文件对象
f = open(“a.txt”,”r”)
f就是类文件对象

数据提取之JSON



JSON	Python
object	dict
array	list
string	unicode
number (int)	int, long
number (real)	float
true	True
false	False
null	None

json在数据交换中起到了一个载体的作用，承载着相互传递的数据

json使用注意点

json中的字符串都是用 双引号 引起来的，且必须是双引号

如果不是双引号

eval:能实现简单的字符串和python类型的转化

replace：把单引号替换为双引号

往一个文件中写入多个json串，不再是一个json串，不能直接读取

这时我们可以一行写一个json串，按照行来读取

4、动态网站数据抓取

什么是AJAX：

AJAX（Asynchronouse JavaScript And XML）异步JavaScript和XML。过在后台与服务器进行少量数据交换，Ajax 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。传统的网页（不使用Ajax）如果需要更新内容，必须重载整个网页页面。因为传统的在传输数据格式方面，使用的是XML语法。因此叫做AJAX，其实现在数据交互基本上都是使用JSON。使用AJAX加载的数据，即使使用了JS，将数据渲染到了浏览器中，在右键->查看网页源代码还是不能看到通过ajax加载的数据，只能看到使用这个url加载的html代码。

获取ajax数据的方式：

- 直接分析ajax调用的接口。然后通过代码请求这个接口。
- 使用Selenium+chromedriver模拟浏览器行为获取数据。

方式	优点	缺点
分析接口	直接可以请求到数据。不需要做一些解析工作。代码量少，性能高。	分析接口比较复杂，特别是一些通过js混淆的接口，要有一定的js功底。容易被发现是爬虫。
selenium	直接模拟浏览器的行为。浏览器能请求到的，使用selenium也能请求到。爬虫更稳定。	代码量多。性能低。

Selenium+chromedriver获取动态数据：

Selenium相当于是一个机器人。可以模拟人类在浏览器上的一些行为，自动处理浏览器上的一些行为，比如点击，填充数据，删除cookie等。chromedriver是一个驱动Chrome浏览器的驱动程序，使用他才可以驱动浏览器。当然针对不同的浏览器有不同的driver。以下列出了不同浏览器及其对应的driver：

Chrome：https://sites.google.com/a/chromium.org/chromedriver/downloads

Firefox：https://github.com/mozilla/geckodriver/releases

Edge：https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

Safari：https://webkit.org/blog/6900/webdriver-support-in-safari-10/

安装Selenium和chromedriver：

1.安装**Selenium**：Selenium有很多语言的版本，有java、ruby、python等。我们下载python版本的就可以了。

```
pip install selenium
```

2.安装**chromedriver**：下载完成后，放到不需要权限的纯英文目录下就可以了。

Selenium和chromedriver：

以一个简单的获取百度首页的例子来讲下Selenium和chromedriver

```
from selenium import webdriver
# chromedriver的绝对路径
driver_path = r'D:\ProgramApp\chromedriver\chromedriver.exe'
# 初始化一个driver，并且指定chromedriver的路径
driver = webdriver.Chrome(executable_path=driver_path)
# 请求网页
driver.get("https://www.baidu.com/")
# 通过page_source获取网页源代码
print(driver.page_source)
```

Notes:

如果只是想要解析网页中的数据，那么推荐将网页源代码扔给lxml来解析。因为lxml底层使用的是C语言，所以解析效率会更高。

如果是想要对元素进行一些操作，比如给一个文本框输入值，或者是点击某个按钮，那么就必须使用selenium给我们提供的查找元素的方法。

Notes:

如果只是想要解析网页中的数据，那么推荐将网页源代码扔给lxml来解析。因为lxml底层使用的是C语言，所以解析效率会更高。

如果是想要对元素进行一些操作，比如给一个文本框输入值，或者是点击某个按钮，那么就必须使用selenium给我们提供的查找元素的方法。

5、Beautiful Soup

Beautiful Soup是什么?

- Beautiful Soup（简称BS4）提供一些简单的、python式的函数用来处理导航、搜索、修改分析树等功能。它
- Beautiful Soup自动将输入文档转换为Unicode编码，输出文档转换为utf-8编码。
- BeautifulSoup已成为和lxml、html6lib一样出色的python解释器,为用户灵活地提供不同的解析策略或强劲的速度。
- Beautiful Soup不需要考虑编码方式，除非文档没有指定，重新原始编码方式即可。

如何遍历文档树？

- **直接子节点**

  .contents ： 标签的　 .content属性可以将tag的子节点以列表的方式输出

  .children：返回一个可迭代对象

- **所有子孙节点**

  .descendants属性可以对所有tag的子孙节点进行递归循环，和children类似，我们也需要遍历获取其中内容。

- **节点内容：**

  .string：返回标签里面的内容

  .text：返回标签的文本

如何搜索文档树?

- find_all(name,attrs,recursive,text,**kwargs)
- 1.传入True：找到所有的Tag

```
#直接找元素
soup.find_all('b')
```

2.传入正则表达式

```
#通过正则找
import re
for tag in soup.find_all(re.compile("^b")):
print(tag.name)
```

3.传入列表

```
#找a 和 b标签
soup.find_all(["a", "b"])
```

如何搜索文档树?

- **4.keyword参数（name,attrs）**

  soup.find_all(id='link2')

  如果传入 href 参数,Beautiful Soup会搜索每个tag的"href"属性:

  soup.find_all(href=re.compile("elsie"))

- **5.通过text参数可以搜索文档中的字符串内容**

  soup.find_all(text="Elsie")

  soup.find_all(text=['Tillie','Elsie','Lacie'])

- **6.限定查找个数-limit参数**

  soup.find_all("a",limit=2)

- **7.recursive参数**

  调用tag的 find_all方法，BeautifulSoup会检查当前tag的所有子孙节点，如果只想搜索tag的直接子节点，可是使用参数recursive =False

  soup.find_all('title',recursive=False)

如何搜索文档树?

- **find_all(name,attrs,recursive,text,**kwargs)**

  结合节点操作

  print(soup.find(id='head'.div.div.next_sibling.next_sibling)

- **find_parents() find_parent()**

  ind_all() 和 find() 只搜索当前节点的所有子节点,孙子节点等.

  find_parents() 和 find_parent() 用来搜索当前节点的父辈节点,搜索方法与普通tag的搜索方法相同,搜索文档搜索文档包含的内容

- **find_next_siblings() find_next_sibling()**

  这2个方法通过 .next_siblings 属性对当 tag 的所有后面解析的兄弟 tag 节点进行迭代,

  find_next_siblings() 方法返回所有符合条件的后面的兄弟节点

  find_next_sibling() 只返回符合条件的后面的第一个tag节点

如何搜索文档树?

- **find_previous_siblings()　　find_previous_sibling()**

  这2个方法通过 .previous_siblings 属性对当前 tag 的前面解析的兄弟 tag 节点进行迭代,

  find_previous_siblings() 方法返回所有符合条件的前面的兄弟节点,

  find_previous_sibling() 方法返回第一个符合条件的前面的兄弟节点

- **find_all_next()　　find_next()**

  这2个方法通过 .next_elements 属性对当前 tag 的之后的 tag 和字符串进行迭代,

  find_all_next() 方法返回所有符合条件的节点

  find_next() 方法返回第一个符合条件的节点

- **find_all_previous() 和 find_previous()**

  这2个方法通过 .previous_elements 属性对当前节点前面的 tag 和字符串进行迭代,

  find_all_previous() 方法返回所有符合条件的节点,

  find_previous()方法返回第一个符合条件的节点

CSS选择器?

- 这就是另一种与 find_all 方法有异曲同工之妙的查找方法.
  写 CSS 时，标签名不加任何修饰，类名前加.，id名前加#
  在这里我们也可以利用类似的方法来筛选元素，用到的方法是 soup.select()，返回类型是 list

  (1) 通过标签名查找 soup.select('a')

  (2) 通过类名查找 soup.select('.sister')

  (3) 通过 id 名查找 soup.select('#link1')

  (4) 组合查找 soup.select('p #link1')

  (5) 属性查找 soup.select('a[class="sister"]')

  (6) 获取内容 get_text()

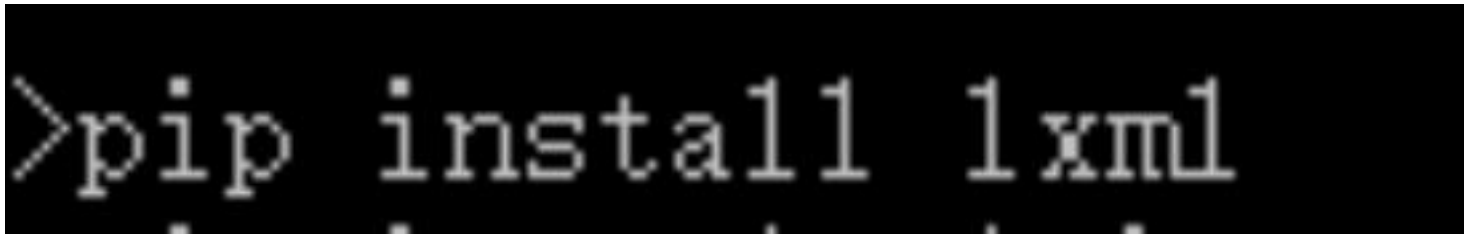
6、IXML

如何安装lxml

- xml的详细介绍，官网链接：http://lxml.de/,是一种使用Python编写的库，可以迅速、灵活地处理xml

- **安装：**

  pip install lxml



lxml库

lxml 是 一个HTML/XML的解析器，主要的功能是如何解析和提取 HTML/XML 数据。

lxml和正则一样，也是用 C 实现的，是一款高性能的 Python HTML/XML 解析器，我们可以利用之前学习的XPath语法，来快速的定位特定元素以及节点信息。

lxml python 官方文档：http://lxml.de/index.html

需要安装C语言库，可使用 pip 安装：pip install lxml
（或通过wheel方式安装）

创建lxml对象

```
# 导入lxml
from lxml import etree

# html字符串
html_str = """
<html>
<head>
<title>demo</title>
</head>
<body>
<p>1111111</p>
</body>
</html>
"""

# 利用html_str创建一个节点树对象
html = etree.HTML(html_str)
type(html) # 输出结果为：lxml.etree._Element
```

节点与属性

- 1.Element类是lxml的一个基础类，大部分XML都是通过Element存储的。可以通过Element方法创建：

```
>>> from lxml import etree
>>> root=etree.Element('root');
>>> print root.tag
root
```

- 2.为root节点添加子节点：

```
>>> child1=etree.SubElement(root,'child1')
>>> print root
<Element root at 0x2246760>
>>> print etree.tostring(root)
<root><child1/></root>
```

- 3.XML Element的属性格式为Python的dict。可以通过get/set方法进行设置或获取操作：

```
>>> root.set('id','123')
>>> id=root.get('id')
>>> id
'123'
```

- 4.遍历全部属性：

```
>>> for value,name in root.items():
...     print value,'\t',name
...
id      123
```

7、Xpath

Xpath是什么?

- XPath(XML Path Lauguage)是一门在XML文档中查找信息的语言，可用来在XML文档中对元素和属性进行遍历。

(1) 节点关系

(2) 选取节点

(3) 谓语

(4) 选取未知节点

(5) Xpath运算符

选取节点

表达式	描述
nodename	选取此节点的所有子节点。
/	从根节点选取。
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。

谓语

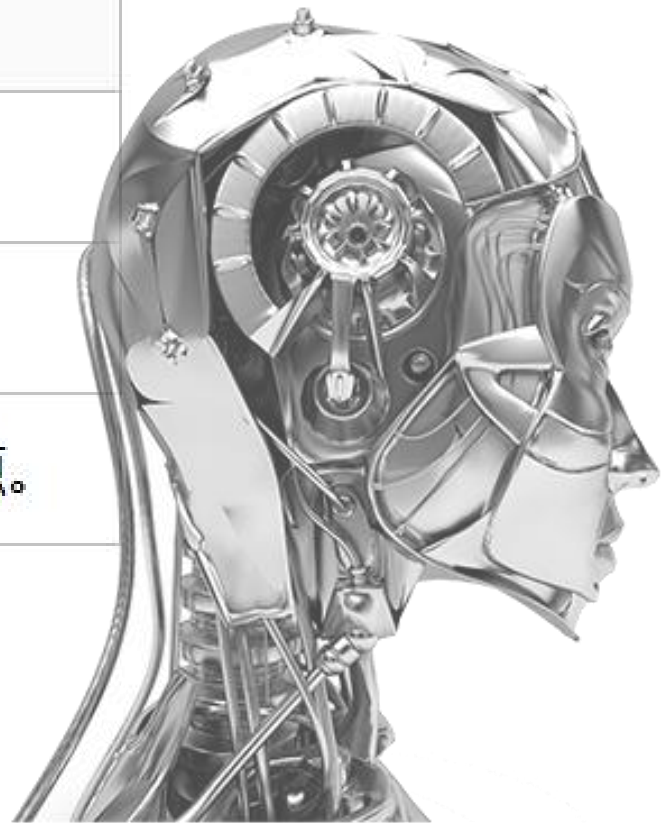
- 谓语用来查找某个特定的节点或者包含某个指定的值的节点，被嵌在方括号中。
  在下面的表格中，我们列出了带有谓语的一些路径表达式，以及表达式的结果：

路径表达式	结果
/bookstore/book[1]	选取属于 bookstore 子元素的第一个 book 元素。
/bookstore/book[last()]	选取属于 bookstore 子元素的最后一个 book 元素。
/bookstore/book[last()-1]	选取属于 bookstore 子元素的倒数第二个 book 元素。
/bookstore/book[position()<3]	选取最前面的两个属于 bookstore 元素的子元素的 book 元素。
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素。
//title[@lang=‘eng’]	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
/bookstore/book[price>35.00]	选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。
/bookstore/book[price>35.00]/title	选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。
//div[contains(@class,"f1")]	选择div属性包含"f1"的元素

选取未知节点

- XPath 通配符可用来选取未知的 XML 元素

通配符	描述
*	匹配任何元素节点。
@*	匹配任何属性节点。
node()	匹配任何类型的节点。



Xpath运算符

运算符	描述	实例	返回值
|	计算两个节点集	//book | //cd	返回所有拥有 book 和 cd 元素的节点集
+	加法	6 + 4	10
–	减法	6 – 4	2
*	乘法	6 * 4	24
div	除法	8 div 4	2
=	等于	price=9.80	如果 price 是 9.80，则返回 true。如果 price 是 9.90，则返回 false。
!=	不等于	price!=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
<	小于	price<9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。
<=	小于或等于	price<=9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。
>	大于	price>9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
>=	大于或等于	price>=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.70，则返回 false。
or	或	price=9.80 or price=9.70	如果 price 是 9.80，则返回 true。如果 price 是 9.50，则返回 false。
and	与	price>9.00 and price<9.90	如果 price 是 9.80，则返回 true。如果 price 是 8.50，则返回 false。
mod	计算除法的余数	5 mod 2	1

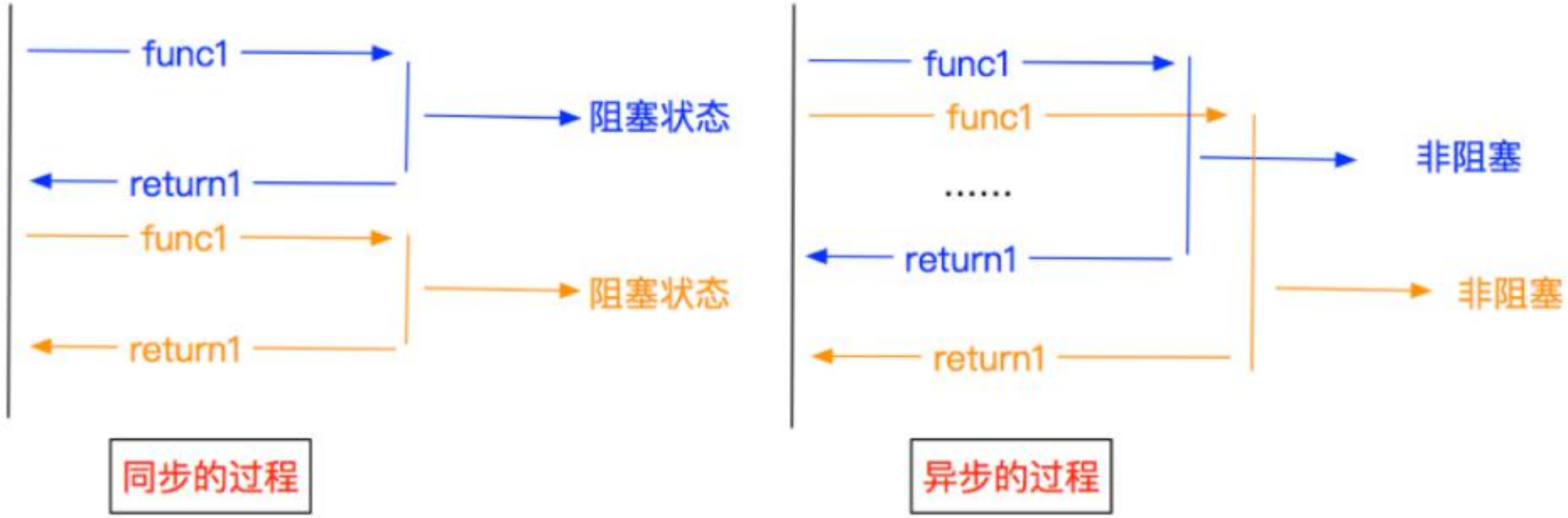
8、Scrapy

什么是Scrapy?

- Scrapy 是一个为了爬取网站数据，提取结构性数据而编写的应用框架，我们只需要实现少量代码，就能够快速的抓取到数据内容。

- Scrapy 使用了 Twisted['twɪstɪd](其主要对手是Tornado)异步网络框架来处理网络通讯，可以加快我们的下载速度，不用自己去实现异步框架，并且包含了各种中间件接口，可以灵活的完成各种需求。

异步与非阻塞的区别



异步：调用在发出之后，这个调用就直接返回，不管有无结果

非阻塞：关注的是程序在等待调用结果（消息，返回值）时的状态，指在不能立刻得到结果之前，该调用不会阻塞当前线程。

Scrapy 框架安装和使用方法（Windows 操作系统）

- **安装（anaconda）**

  使用anaconda直接 **conda install scrapy**

  过后会停留在**Proceed <[y]/n>**? 这一行

  输入 **y** 确认 继续安装

  待安装完成后，输入 **scrapy**

  若能显示出scrapy的一些命令则为安装成功


Twisted, an event-driven networking engine.
Twisted-19.2.1-cp27-cp27m-win32.whl
Twisted-19.2.1-cp27-cp27m-win_amd64.whl
Twisted-19.2.1-cp35-cp35m-win32.whl
Twisted-19.2.1-cp35-cp35m-win_amd64.whl
Twisted-19.2.1-cp36-cp36m-win32.whl
Twisted-19.2.1-cp36-cp36m-win_amd64.whl
Twisted-19.2.1-cp37-cp37m-win32.whl
Twisted-19.2.1-cp37-cp37m-win_amd64.whl
Twisted-19.2.1-cp38-cp38m-win32.whl
Twisted-19.2.1-cp38-cp38m-win_amd64.whl

- **安装（python 3.7）**

  scrapy框架的安装需要许多的依赖库，其中 Twisted 用 pip 方法安装会不成功

  因此，先去下载 Twisted 的离线安装包,使用**CTRL + F** 查找**Twisted**，再下载相应的 whl 文件

  win32 即 32 位系统，amd64 即 64 位系统

  在 cmd 终端 通过 cd 命令，打开下载的文件所在的文件夹

  输入**pip install 文件名.whl**安装

  安装完成后通过 **pip install scrapy** 安装 scrapy 框架

Scrapy的使用

- **使用**

  导入：

  import scrapy

- **命令：**

  · 测试 xpath/css 表达式：

  response.xpath(“表达式”) [[0], .extract(), .extract_first()]

  response.css(‘表达式’) [[0], .extract(), .extract_first()]

  · 创建：

```
----------常用命令----------        -----作用-----
scrapy                          | 弹出scrapy命令列表
scrapy startproject 项目名        | 创建一个项目
scrapy genspider 爬虫名 url       | 创建一个爬虫
scrapy genspider -t 模板 name url  | 一般用的有basic（默认）和crawl
cd 项目名                         | 进入项目文件夹（Windows命令）
cd ..                           | 退出项目（Windows命令）
scrapy check                      | 检查代码是否存在错误，并且指出错误
scrapy list                     | 返回项目里所有的爬虫名称
scrapy edit 爬虫名                 | 在命令行进行编辑（不方便，一般不用）
                  scrapy settings                 | 查看设置
                  scrapy settings -h 配置名         | 获取配置信息
                  scrapy runspider name.py        | 运行单独一个爬虫文件
```

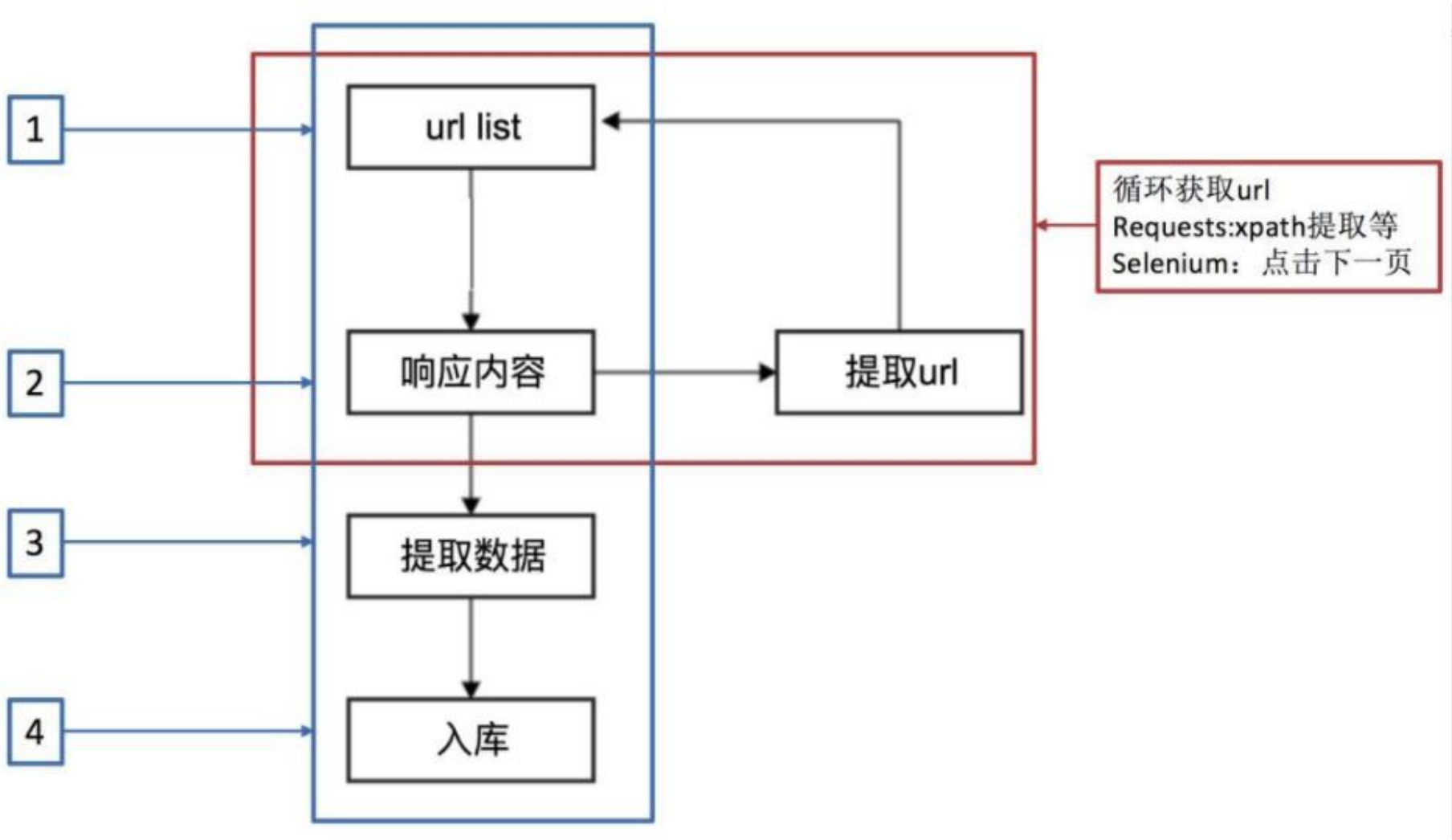
Scrapy 项目文件构成

- 当创建好一个项目后，会生成几个项目文件

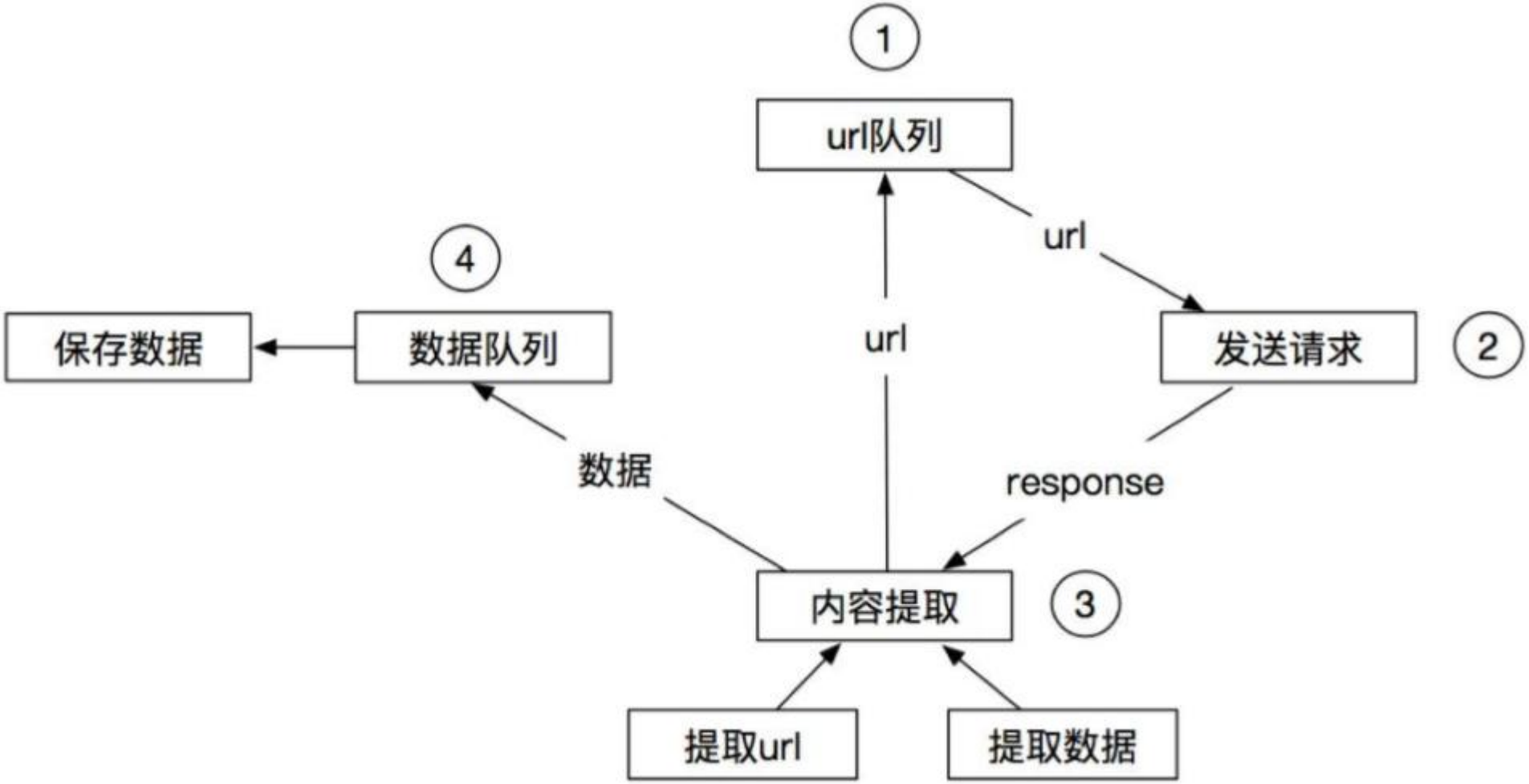
- 具体介绍如下：

```
|-- spider. py          ----  爬虫文件夹
    |---- _init_. py    ----  默认的爬虫代码文件
    |---- myspider. py  ----  具体爬虫代码文件
|-- _init_. py          ----  爬虫项目的初始化文件，用来对项目做初始化工作。
|-- items. py           ----  爬虫项目的数据容器文件，用来定义要获取的数据。
|-- pipelines. py       ----  爬虫项目的管道文件，用来对items中的数据进行进一步的加工处理
|-- middlewares.py      ----  爬虫中间件文件，处理request和reponse等相关配置
|-- setting. py         ----  爬虫项目的设置文件，包含了爬虫项目的设置信息
|-- scrapy. cfg         ----  爬虫项目的配置文件，定义了项目的配置文件路径、部署相关信息等内容
```

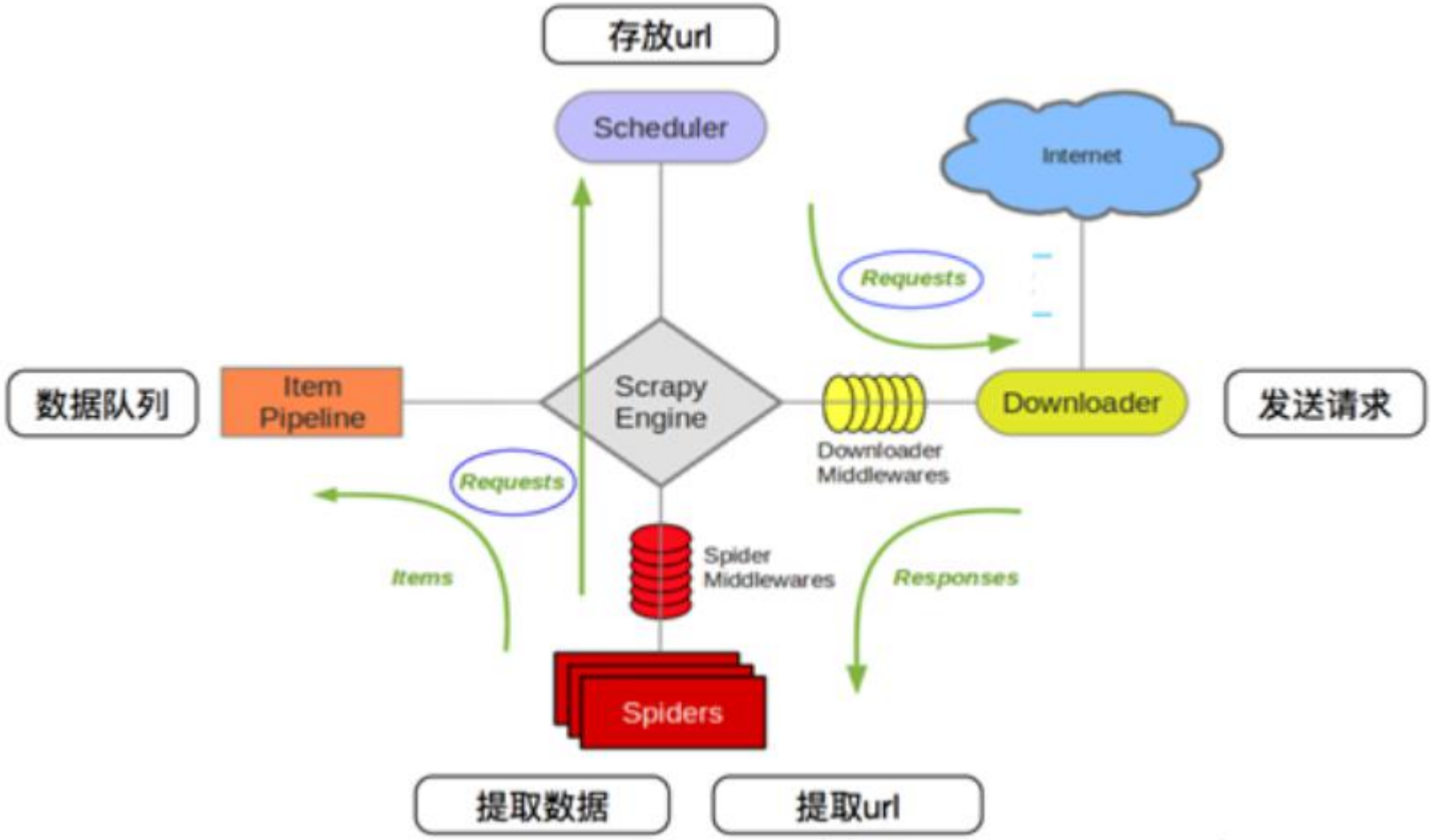
回顾前面的爬虫流程


1
2
3
4
url list
响应内容
提取数据
入库
提取url
循环获取url
Requests:xpath提取等
Selenium：点击下一页

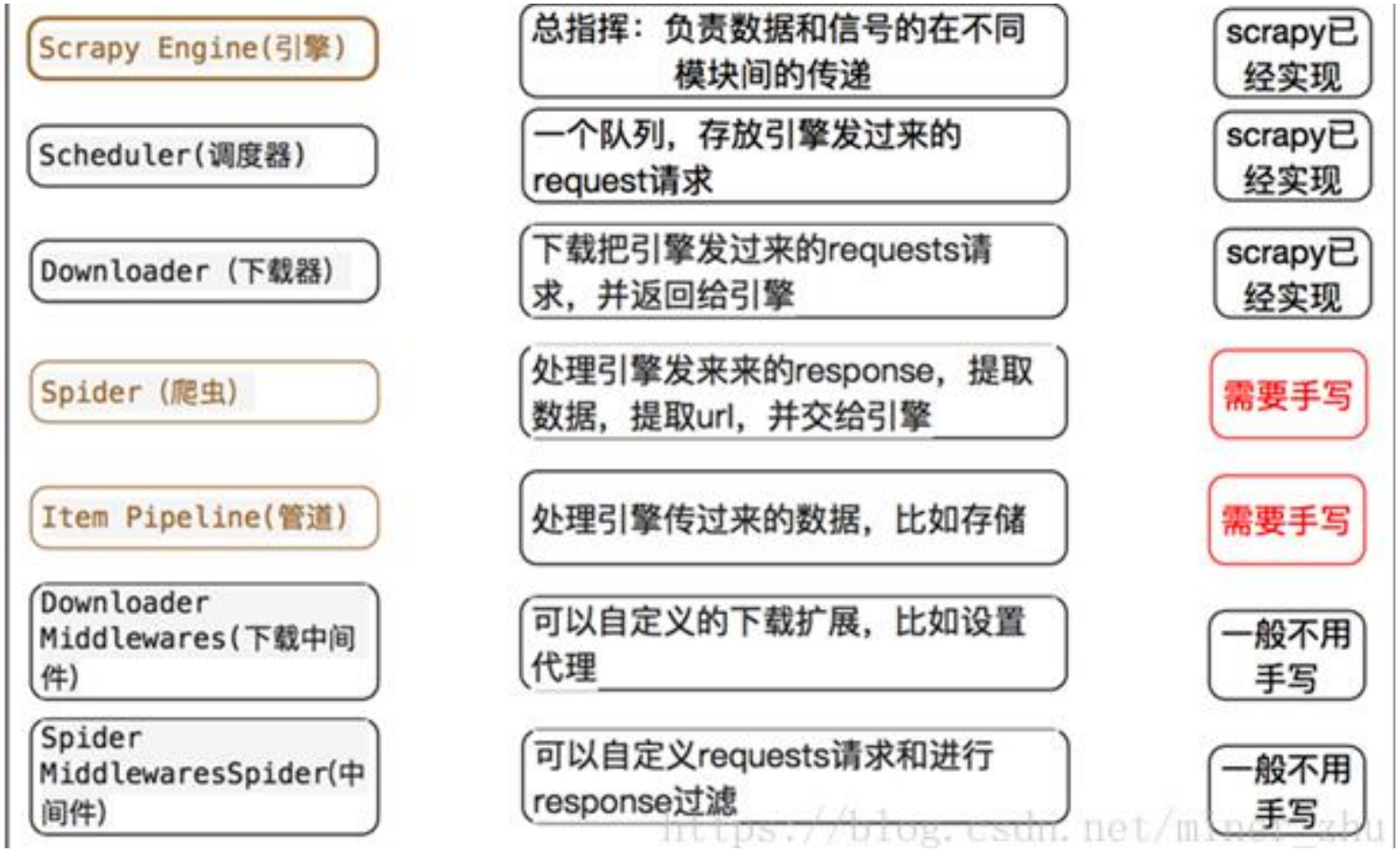
另一种形式爬虫流程


1
url队列
url
4
保存数据
数据队列
url
发送请求
2
数据
response
内容提取
3
提取url
提取数据

Scrapy的爬虫流程



Scrapy的爬虫流程

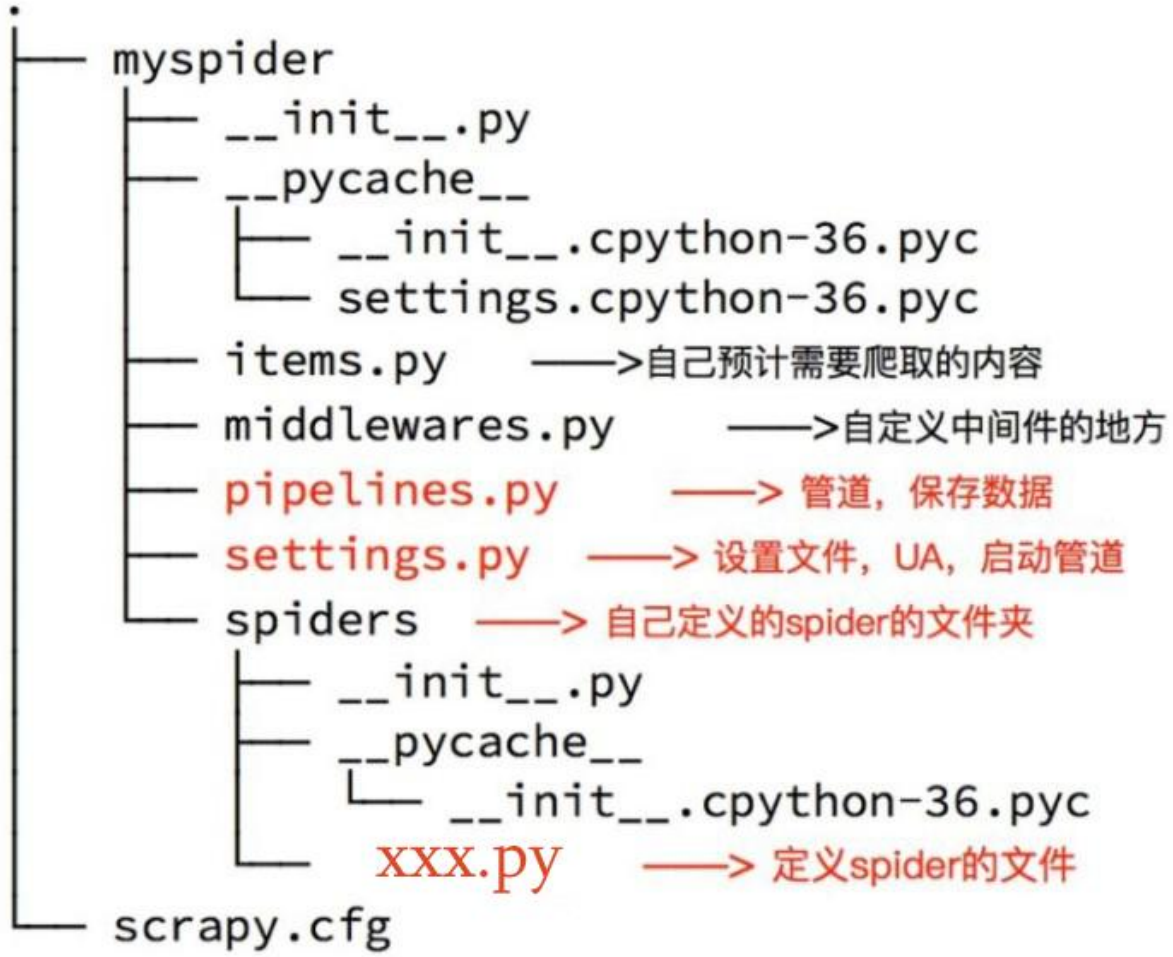


Scrapy的爬虫流程

1.创建一个scrapy项目
    scrapy startproject mySpider
2.生成一个爬虫
    scrapy genspider xxx “xxx.com”
3.提取数据
    完善spider，使用xpath等方法
4.保存数据
    pipeline中保存数据

创建一个爬虫

命令：scrapy genspider　+<爬虫名字> + <允许爬虫的域名>
scrapy genspider xxx “xxx.com”



完善spider

完善spider即通过方法进行数据的提取等操作：

```
class shenSpider(scapy.Spider):      #自定义spider类，继承自scrapy.spider
    name='xxx'      #爬虫名字
    allowed_domains = ['xxx.com']    #允许爬取的范围，防止爬虫爬到了别的网站
    start_urls = ['xxx.com']     #开始爬取的地址

    def parse(self,response):      #数据提取方法，接收下载中间件传过来的response
        names = response.xpath('xxxxxxx')
        print (names)      #返回包含选择器的列表
```

注意：

- response.xpath方法的返回结果是一个类似list的类型，其中包含的是selector对象，操作和列表一样，但是有一些额外的方法
- extract() 返回一个包含有字符串的列表
- extract_first() 返回列表中的第一个字符串，列表为空没有返回None
- spider中的parse方法必须有
- 需要抓取的url地址必须属于allowed_domains,但是start_urls中的url地址没有这个限制
- 启动爬虫的时候注意启动的位置，是在项目路径下启动

spider的数据传到pipeline

```
def parse(self,response):
    products = response.xpath('xxx.com')    #xpath分组提取

    for i in products:
        name = i.xpath('xxx.com')
        price = i.xpath('xxx.com')
        introduction = i.xpath('xxx.com')
        item = (name = name,
                price = price,
                introduction = introduction
                )
        # yield item   → yield 就可以了
```

为什么要使用yield?

- 让整个函数变成一个生成器，有什么好处呢?
- 遍历这个函数的返回值的时候，挨个把数据读到内存，不会造成内存的瞬间占用过高
- python3中的range和python2中的xrange同理

注意:

yield能够传递的对象只能是：BaseItem,Request,dict,None

使用pipeline

```
import json
class MyspiderPipeline(object):
    def process_item(self,item,spider):          #→实现存储方法
        with open('temp.txt','a') as f:
            json.dump(item,f,ensure_ascii=False,indent=2)
```

完成pipeline代码后，需要在setting中设置开启

```
ITEM_PIPELINES = {
    'myspider.pipelines.MyspiderPipeline': 300,
}
```

pipeline的位置　　权重

使用pipeline

pipeline在settings中为什么需要开启多个?

- 不同的pipeline可以处理不同爬虫的数据
- 不同的pipeline能够进行不同的数据处理的操作，比如一个进行数据清洗，一个进行数据的保存

pipeline使用注意点

- 使用之前需要在settings中开启
- pipeline在setting中键表示位置(即pipeline在项目中的位置可以自定义)，值表示距离引擎的远近，越近数据会越先经过
- 有多个pipeline的时候，process_item的方法必须return item,否则后一个pipeline取到的数据为None值
- pipeline中process_item的方法必须有，否则item没有办法接受和处理
- process_item方法接受item和spider，其中spider表示当前传递item过来的spider

设置log

为了让我们自己希望输出到终端的内容能容易看一些，我们可以在setting中设置log级别
在setting中添加一行（**全部大写**）：LOG_LEVEL = “WARNING”
默认终端显示的是debug级别的log信息

1. 使用 WARNING 打印出信息

```
import logging  # 导入模块

'''可以把当前的py文件所在的位置显示出来'''
logger = logging.getLogger(__name__)

'''使用logger打印出信息'''
logger.warning()
```

设置log

创建 名为scrapyTest的爬虫，域名为： quotes.toscrape.com　　scrapy genspider scrapyTest quotes.toscrape.com

在scrapyTest,py里面加上log信息，打印循环次数

```
import scrapy
import logging

'''可以把当前的py文件所在的位置显示出来'''
logger = logging.getLogger(__name__)


class ScrapytestSpider(scrapy.Spider):
    name = 'scrapyTest'
    allowed_domains = ['quotes.toscrape.com']
    start_urls = ['http://quotes.toscrape.com/']

    def parse(self, response):
        for i in range(1, 11):
            item = {"循环次数": str(i)}
            logger.warning(item)
            yield item
```

设置log

还可以在pipelines.py加上log信息，要加的话就要到settings.py里面开启pipeline：

```
'''开启pipeline，只需到settings.py里面把下面的注释去掉就行'''
ITEM_PIPELINES = {
   'PipelineTest.pipelines.PipelinetestPipeline': 300,
}
```

pipelines.py

```
import logging
logger = logging.getLogger(__name__)
class PipelinetestPipeline(object):
    def process_item(self, item, spider):
        logging.warning("*-*pipeline的warning信息*-*")
        return item
```

设置log

然后运行，查看结果 scrapy crawl scrapyTest

打印的结果 里面包括 日期、时间、还有当前py文件的路径和名称，以及设置的打印信息

```
E:\pycharmcode\Python爬虫\框架爬虫\PipelineTest>scrapy crawl scrapyTest
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '1'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '2'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '3'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '4'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '5'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '6'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '7'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '8'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '9'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
2019-01-10 15:22:04 [PipelineTest.spiders.scrapyTest] WARNING: {'循环次数': '10'}
2019-01-10 15:22:04 [PipelineTest.pipelines] WARNING: *-*pipeline的warning信息*-*
```

设置log

2. 把log的信息保存到本地

在settings.py里面添加上

```
LOG_FILE = "./log.log"
```

然后运行程序，没有打印信息，但是log的信息全部保存到了当前爬虫项目下的log.log，打开文件会发现和打印的信息是一样的

设置log

3. logging也可以用在普通的项目中使用

import logging

logging.basicConfig(...) #设置日志输出的样式，格式

实例化一个`logger=logging.getLogger(__name__)`

在任何py文件中调用logger即可

requests请求翻页

思路
找到下一页url地址
调用 requests.get(url) —— 传递给调度器

scrapy 中操作
组装称一个requests对象再传给引擎
scrapy.Request 能构建一个requests，同时指定提取数据的 callback 函数

```
# 找到下一页的 url 地址
next_url = response.xpath("xpath_expression").extract_first()
if next_url != "结束判断条件":
        next_url = "基础url" + next_url  #获取下一页url
        # 构造一个请求，请求->引擎->调度器->引擎->下载器->响应 下一页爬取
        yield scrapy.Request(
                next_url,
                # 若获取下一页内容采用同一函数（parse），则继续调用 自身 获取
                # 若获取下一页内容采用不同函数（parse1），则直接调用 parse1 获取
                callback = self.parse()

        )
```

scrapy.Request() 可选参数

scrapy.Request(url [, callback, method='GET', headers, body, cookies,meta, dont_filter = False])

callback : 指定传入的 url 交给哪个解析函数去处理

meta : 实现再不同的解析函数中传递数据，meta默认会携带部分信息，比如下载延迟，请求深度等

dont_filter : 让scrapy的去重不会过滤当前url（scrapy 默认会有url去重功能）

Scrapy中的CrawlSpider

简介：

回头看：

之前的代码中，我们有很大一部分时间在寻找下一页的url地址或者是内容的url地址上面，这个过程能更简单一些么？

思路：

1、从response中提取所有的a标签对应的url地址

2、自动的构造自己requests请求，发送给引擎

上面的功能可以做的更好：

满足某个条件的url地址，我们才发送给引擎，同时能够指定callback函数

Scrapy中的CrawlSpider

生成 crawl spider 命令：

scrapy genspider -t crawl demo baidu.com

生成文件代码解析：

```
# -*- coding: utf-8 -*-
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule


class DemoSpider(CrawlSpider):  # 继承了CrawlSpider父类
    name = 'demo'
    allowed_domains = ['baidu.com']
    start_urls = ['http://baidu.com/']

    # 定义提取url地址规则
    rules = (
        # rules   可定义多个
        # LinkExtractor   连接提取器----提取url地址
        # callback   提取出来的url地址的response会交给callback处理
```

注意点：

- 即使在 allow 中 url 不完整，CrawlSpider 也会自动的补充完整再请求
- **不能再有以 parse 为名字的数据提取方法，**这个方法已被 CrawlSpider 使用
- 一个 Rule 对象接收很多参数，首先第一个是报假案url规则的 LinkExtractor 对象，常用的还有callback（制定满足规则的 url 的解析函数的字符串）和 follow（ response 提取的链接是否需要跟进）
- 不指定 callback 函数的请求下，如果 follow 为 True ， 满足该入了的俩还会继续被请求
- 如果多个 Rule 都满足某一个 url，会从 rules 中选择第一个满足的进行操作
- **LinkExtractor更多常见参数:**

  allow: 满足括号中"正则表达式"的URL会被提取,如果为空,则全部匹配

  deny: 满足括号中“正则表达式"的URL一定不提取(优先级高于allow)

  allow_domains: **会**被提取的链接的 domains.

  deny_domains: **一定不会**被提取链接的 domains.

  restrict_xpaths: 使用 **xpath表达式,**和 allow 共同作用过滤链接,级 xpath 满足范围内的 url 地址会被提取
- **spiders. Rule常见参数:**

  link_extractor: 是一个 Link Extractor 对象,用于定义需要提取的链接。

  callback: 从 link_extractor 中每获取到链接时,参数所指定的值作为回调函数

  follow: 是一个布尔(boolean)值,指定了根据该规则从 response 提取的链接是否需要跟进

  如果 callback 为 None , follow 默认设置为 True , 否则默认为 False

  process_links: 指定该 spider 中哪个的函数将会被调用,从 link_extractor 中获取到链接列表时将会调用该函数,**该方法主要用来过滤 url**

  process_request: 指定该 spider 中哪个的函数将会被调用,该规则提取到每个 request 时都会调用该函致,**用来过滤 request**

Scrapy模拟登录之携带cookie登录:

- **应用场景**

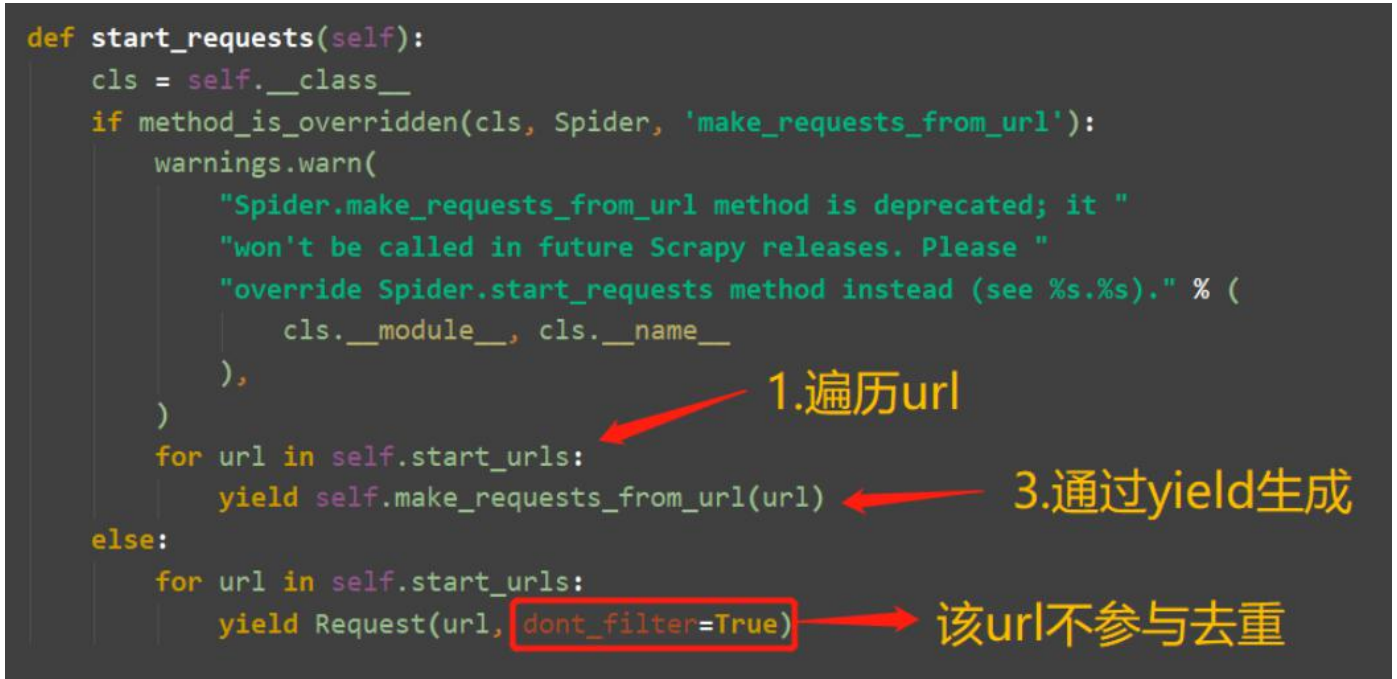
  cookie过期时间很长,常见于一些不规范的网站

  能在cookie过期之前把搜有的数据拿到

  配合其他程序使用,比如其使用selenium把登陆之后的cookie获取到保存到本地, scrapy发送请求之前先读取本地cookie

- **如何实现**

  我们定义在spider下的start_urls = [] 都是默认交给 start_requests 处理的，所以如果有必要，可以重写 start_requests 方法



Scrapy模拟登录之携带cookie登录:

```
def start_requests(self):
    cls = self.__class__
    if method_is_overridden(cls, Spider, 'make_requests_from_url'):
        warnings.warn(
            "Spider.make_requests_from_url method is deprecated; it "
            "won't be called in future Scrapy releases. Please "
            "override Spider.start_requests method instead (see %s.%s)." % (
                cls.__module__, cls.__name__
            ),
        )
        for url in self.start_urls:
            yield self.make_requests_from_url(url)
    else:
        for url in self.start_urls:
            yield Request(url, dont_filter=True)

def make_requests_from_url(self, url):
    """ This method is deprecated. """
    return Request(url, dont_filter=True)
```

1.遍历url

3.通过yield生成

该url不参与去重

2.构造一个request请求

Scrapy模拟登录之携带cookie登录:

1.cookie 字符串转化为字典:

```
cookie = {i.split("=")[0]:i.split("=")[1] for i in cookies.split("; ")}
```

2.使用

```
def start_requests(self):
cookies = "..."
cookies = {i.split("=")[0]:i.split("=")[1] for i in cookies.split("; ")}
yield scrapy.Request(
    self.start_urls[0],
    callback=self.parse,
    cookies=cookies
)
```

3.若在 settings 文件中关闭了 COOKIES_ENABLED，后续的页面请求也会带上传递进去的cookies

4.也可以在 settings 文件中添加 COKIES_DEBUG = True 以查看 cookies 去向

5.cookies不能放在 headers 中去请求，否则无效；直接放在 scrapy.Request 中就能获取到cookie

Scrapy模拟登录之发送post请求

1.scrapy.Request发送的是get请求；scrapy.FormRequest发送post请求（同时使用formdata来携带post数据）

2.通过发送cookie中的 form_data 数据进行提交（根据网站的Form Data来填写）

```
def parse(self, response):
  authenticity_token = response.xpath("xpath_expression").extract_first()
  utf8 = response.xpath("xpath_expression").extract_first()
  commit = response.xpath("xpath_expression").extract_first()
  post_data = dict(
      login="login_name",
      password="login_password",
      authenticity_token=authenticity_token,
      utf8=utf8,
      commit=commit
  )
  yield scrapy.FormRequest(
      "html_url",
      formdata=post_data,
      callback=self.after_login
  )

  类似于:
  yield scrapy.Request(
                        url,
                        method="POST",
                        body = {   }
  )
```

Scrapy模拟登录之发送post请求

3.利用Scrapy中自动找到action中的url和form表单的功能进行账号和密码的填写进行登录操作

```
def parse(self, response):
    yield scrapy.FormRequest.from_response(
        response, #自动从response中寻找form表单
        formdata={"login":"login_name","password":"your_password"},
        callback=self.after_login
    )
```

TIPS:
如果有多个form表单，可通过加入formname，formid，formnumber，formxpath来选择不同的form表单

9、scrapy redis

scrapy_redis 相关

1.分布式爬虫（多台机器协作）的关键是：共享爬取队列，爬取队列在主机维护，各从机调度器统一从共享队列获取Request，即主机维护爬取队列，从机负责数据抓取、数据处理、数据存储；

2.Redis 是**非关系型数据库**（Key – Value形式存储），结构灵活；是**内存中数据结构存储系统**，处理速度快，性能好；提供队列、集合等多重存储结构，方便队列维护

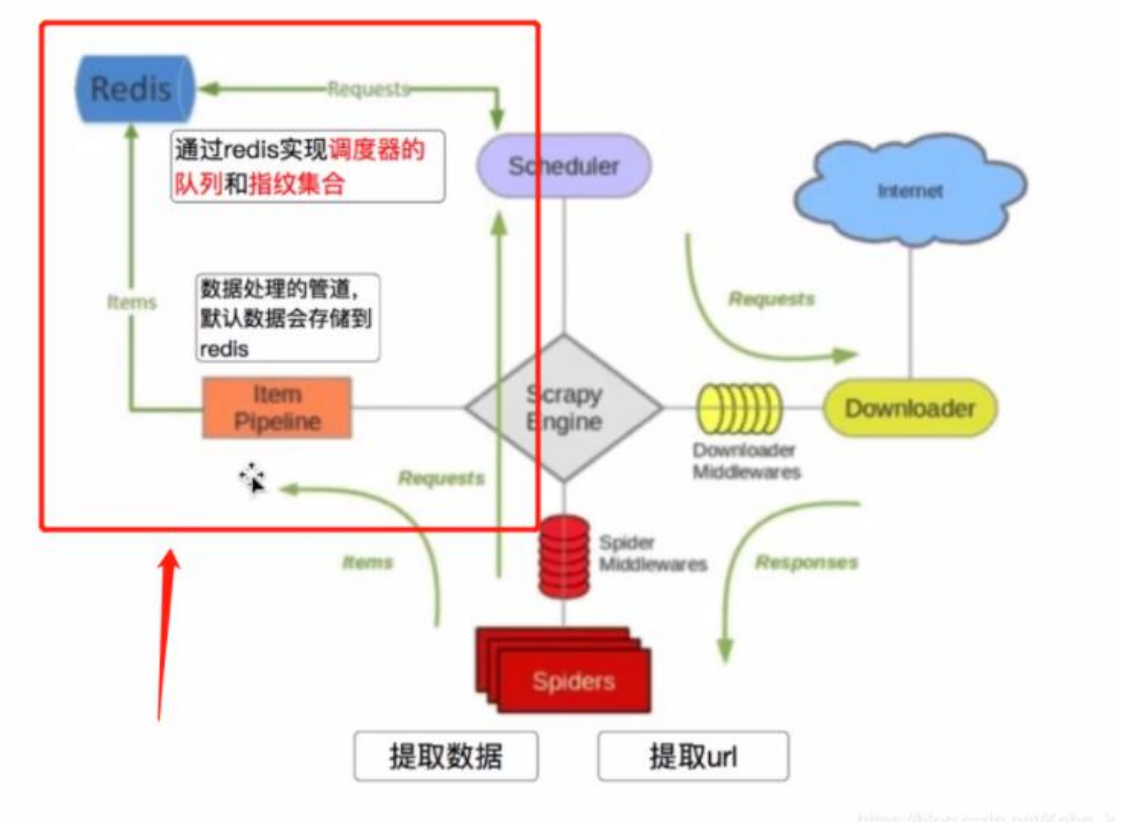
3.scrapy_redis 在 scrapy 基础上实现了更多、更强大的功能，具体体现在：**request去重，爬虫持久化，轻松实现分布式**

4.**如何实现去重：**借助Redis集合，在集合中存储每个Request的指纹。在向Request队列中加入Request之前先验证这个Request的指纹是否已经在集合中：
（1）已存在则不添加到队列
（2）不存在则添加入队列并将指纹加入集合

5.**如何防止中断（实现持久化）**：在每台从机Scrapy启动时都会首先判断Redis Request队列是否为空：
（1）不为空则从队列中取得下一个Request执行爬取
（2）为空则重新开始爬取，第一台从机执行爬取向队列中天下Request

scrapy_redis的爬虫流程：

7.scrapy_redis的爬虫流程



8.Redis中存在爬过的（用来去重）以及待爬的（可实现断点续爬）指纹集合

THANKS!