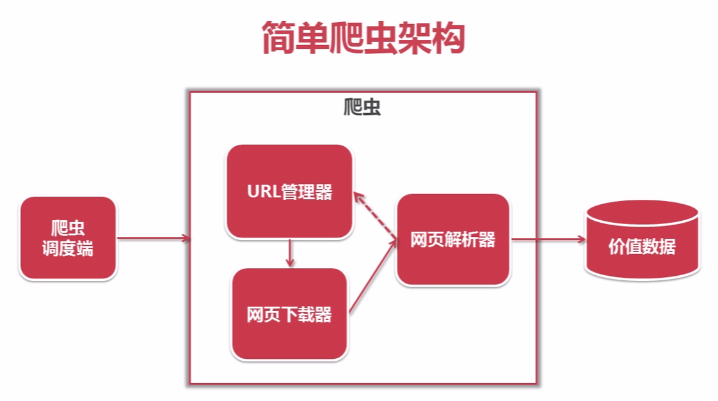
### 爬虫框架

爬虫调度端：启动爬虫，停止爬虫，监视爬虫运行情况

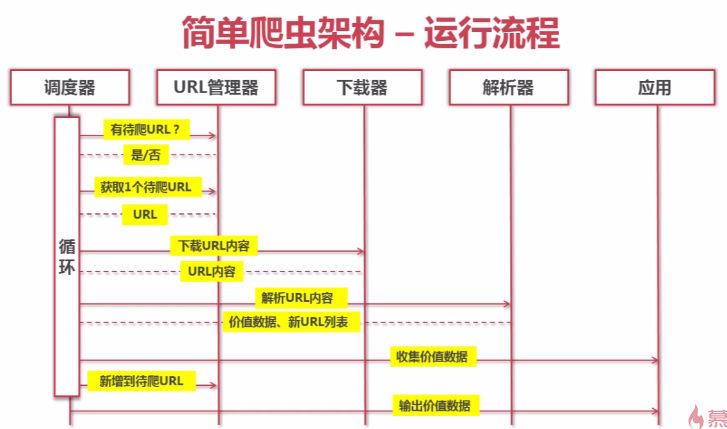
URL管理器：对将要爬取的和已经爬取过的URL进行管理；可取出待爬取的URL，将其传送给“网页下载器”

网页下载器：将URL指定的网页下载，存储成一个字符串，在传送给“网页解析器”

网页解析器：解析网页可解析出①有价值的数据②另一方面，每个网页都包含有指向其他网页的URL，解析出来后可补充进“URL管理器”



### 运行流程

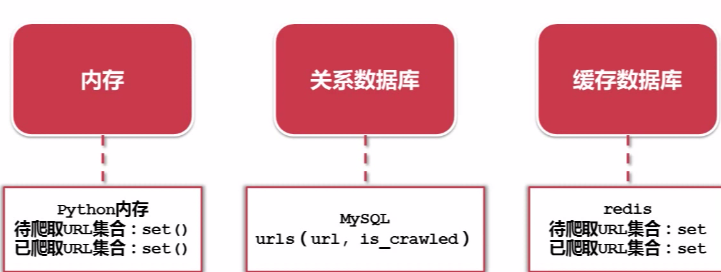


### URL管理器

管理待抓取的URL集合和已抓取的URL集合，防止重复抓取、防止循环抓取



存储：



### 网页下载器

urllib下载网页方法：

##### urllib.request.urlopen(url)

urllib的request模块可以非常方便地抓取URL内容，也就是发送一个GET请求到指定的页面，然后返回HTTP的响应：

|  |
| --- |
| from urllib import request  url = 'https://api.douban.com/v2/book/2129650'  res = request.urlopen(url) #返回响应对象  print('Status:', res.status, res.reason) # 200状态码 ok状态码详情  print(res.getheaders()) #list里包tumple的格式 [('Content-Length', '2058'), ('Connection', 'close')]  for k, v in res.getheaders(): #循环遍历显示响应头信息  print('%s: %s' % (k, v))  data = res.read() #返回响应数据并赋值给data  print(data) #字节字符串  print(data.decode('utf-8')) #文本字符串 (json格式） |

##### 模拟浏览器发送GET请求：

如果我们要想模拟浏览器发送GET请求，就需要使用Request对象，通过往Request对象添加HTTP头，我们就可以把请求伪装成浏览器。

|  |
| --- |
| from urllib import request  req = request.Request('http://www.douban.com/') #创建request对象，  req.add\_header('User-Agent', 'Mozilla/6.0 (iPhone; CPU iPhone OS 8\_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/8.0 Mobile/10A5376e Safari/8536.25')  response = request.urlopen(req) #发送请求，将req传过去，获取响应对象  print('Status:', response.status, response.reason)  for k, v in response.getheaders():  print('%s: %s' % (k, v))  print('Data:', response.read().decode('utf-8')) #返回的数据是bytes字符串，不要解码 |

##### 模拟浏览器发送POST请求：

只需要data以bytes形式传入，其他跟GET类似

|  |
| --- |
| from urllib import request,parse  url = 'https://api.douban.com/v2/book/2129650'  req = request.Request(url)  req.add\_header('User-Agent','Mozilla/5.0')  data = parse.urlencode([('a','1'),('b',2)]) # a=1&b=2  response = request.urlopen(req,data=data.encode('utf-8'))  print(response.status,response.reason)  print('Data:', response.read().decode(‘utf-8’)) |

##### urllib.request.urlretrieve(url,filenam)

抓取url的内容，将它保存到本地filename

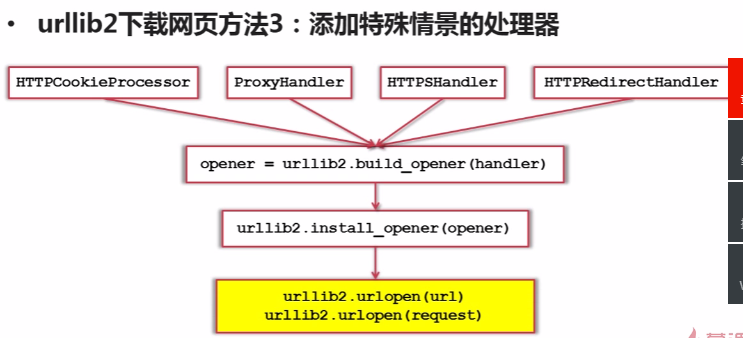
参数：

url：待抓取的url地址

filename：需要保存的文件名

|  |
| --- |
| local\_filename, headers = urllib.request.urlretrieve('http://python.org/',”outer.html”)  local\_filename返回文件名  headers返回header头信息 |

##### Handler



python3中：

|  |
| --- |
| import urllib.request,http.cookiejar  #创建cookie容器  cj = http.cookiejar.CookieJar()  #创建一个opener  opener = urllib.request.build\_opener(urllib.request.HTTPCookieProcessor(cj))  #给urllib安装opener  urllib.request.install\_opener(opener)  #使用带有cookie的urllib访问网页  response = urllib.request.urlopen('http://www.baidu.com')  for k,v in response.getheaders():  print(k,v) |

##### 综合

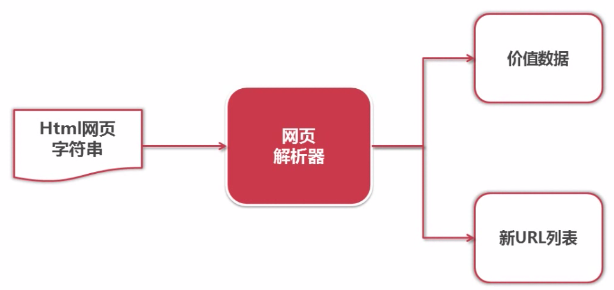
|  |
| --- |
| import urllib.request,http.cookiejar  url = 'http://www.baidu.com'  print('第一种方法')  res1 = urllib.request.urlopen(url)  print(res1.status,res1.reason)  print(len(res1.read()))  print('第二种方法') #模拟浏览器get请求  req = urllib.request.Request(url)  req.add\_header('user-agent','Mozilla=5.0') #模拟浏览器  res2 = urllib.request.urlopen(req)  print(res2.status,res2.reason)  print(len(res2.read()))  print('第三种方法')  cj = http.cookiejar.CookieJar()  opener = urllib.request.build\_opener(urllib.request.HTTPCookieProcessor(cj))  urllib.request.install\_opener(opener)  res3 = urllib.request.urlopen(url)  print(res3.status,res3.reason)  print(cj)  print(res3.read().decode('utf-8')) |

##### 使用requests模块下载

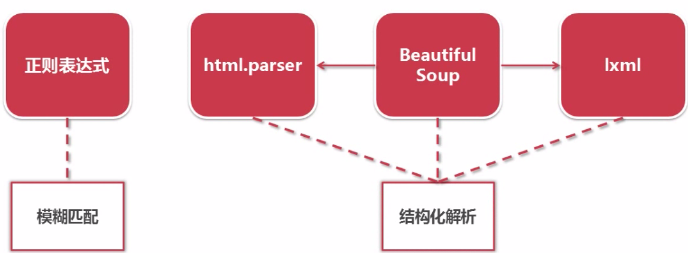
|  |
| --- |
| import requests  response = requests.get(URL) #返回一个Response对象  response.status\_code # 成功返回200  response.text #下载页面的字符串  response.raise\_for\_status() #检查错误，如果出错抛出异常  #保存下载的文件到硬盘  playFile = open('playFile.html','wb') #将下载的文件保存到文件，此处必须用‘写二进制’的方式打开文件，  for chunk in response.iter\_content(10000): #response的这个方法，一次可以读取指定数量的字符，不用一次性把整个文件读取到缓存，节省内存  playFile.write(chunk) |

### 网页解析器

从网页中提取有价值数据的工具



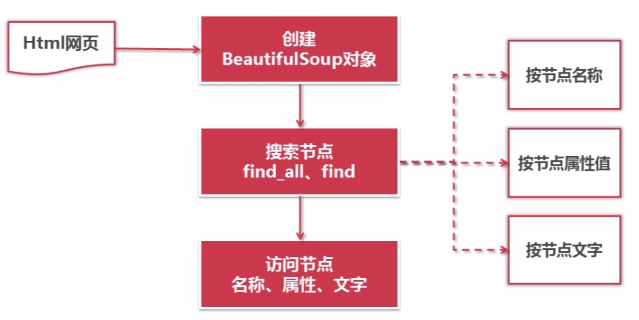
python的几种解析器：



#### BeautifulSoup4

使用pycharm的同学，直接输入import bs4，然后会报错，提示模块不存在

按 alt+回车 出现错误修正提示,选择模块安装即可



|  |
| --- |
| 借用4.5中html\_doc=res1.read().decode(‘utf-8’)  from bs4 import BeautifulSoup  #1、根据HTML网页字符串创建BeautifulSoup对象  soup = BeautifulSoup(  html\_doc, #HTML文档字符串，或File对象  'html.parser', #HTML解析器  from\_encoding='utf-8' #HTML文档编码  )  2.查找节点  # find\_all(name,attrs,string|text,limit)  # name 节点名称  # attrs 节点属性  # string 节点文本  #limit搜索几条  #查找所有标签为a的节点  node = soup.find\_all('a')  #查找标签为a，链接符合http://e.baidu.com/?refer=888形式的节点  node= soup.find\_all('a',href='http://e.baidu.com/?refer=888')  #查找所有有id属性的a标签  node = soup.find\_all(‘a’,href=True)  #定义一个属性字典  node=soup.find\_all(‘a’,attrs={‘id’:’soup’})  #使用正则（每个参数都可以使用正则）  node = soup.find\_all('a',href=re.compile(r'http://www.baidu.com/.\*'))  node = soup.find\_all(name = re.compile(r'\ba\b')) #a为标签  #查找标签文本是‘意见反馈’的a标签  node = soup.find\_all('a',string='意见反馈')  #还能根据多个属性综合查询，中间可以传多个属性，class是python关键字，所以用class\_  node = soup.find\_all(name=‘a’,href=’’,class\_=’’,string=’’)  print(node) #是个list列表，里面存放所有查到的节点  3、访问节点信息  # print(node[0].name) 返回标签名  # print(node[0]['href']) 返回标签属性值  # print(node[0]['id'])  print(node[0].get\_text()) 返回标签的文本 |

find(name,attrs,string)，用法同find\_all,查找一个，直接返回结果Tag

find能多次调用，例：soup.find(‘p’).find(‘a’)

find与find\_all的区别：

find\_all返回一个列表，而find直接返回结果，find\_all不能多次调用，但find可以

按选择器查找：

|  |
| --- |
| node = soup.select(CSS选择器) #类似于js的选择器，返回一个list列表  node[0].attrs #获取属性  node[0].get(‘id’) #获取属性id值  node[0].getText() #返回标签文本  例：  soup.select(‘body div a’) |

### 实例