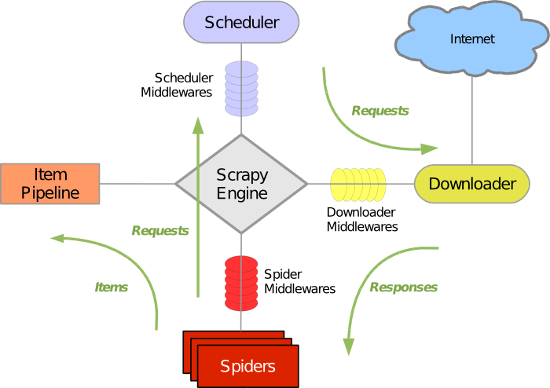
Scrapy，Python开发的一个快速,高层次的屏幕抓取和web抓取框架，用于抓取web站点并从页面中提取结构化的数据。Scrapy用途广泛，可以用于数据挖掘、监测和[自动化测试](http://baike.baidu.com/view/1303916.htm" \t "_blank)。

Scrapy吸引人的地方在于它是一个框架，任何人都可以根据需求方便的修改。它也提供了多种类型爬虫的基类，如BaseSpider、sitemap爬虫等，最新版本又提供了web2.0爬虫的支持。

Scratch，是抓取的意思，这个Python的爬虫框架叫Scrapy，大概也是这个意思吧，就叫它：小刮刮吧。

Scrapy 使用了 Twisted异步网络库来处理网络通讯。整体架构大致如下

Scrapy主要包括了以下组件：

* **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*
* **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。*
* **下载器中间件(Downloader Middlewares)**  
  *位于Scrapy引擎和下载器之间的框架，主要是处理Scrapy引擎与下载器之间的请求及响应。*
* **爬虫中间件(Spider Middlewares)**  
  *介于Scrapy引擎和爬虫之间的框架，主要工作是处理蜘蛛的响应输入和请求输出。*
* **调度中间件(Scheduler Middewares)**  
  *介于Scrapy引擎和调度之间的中间件，从Scrapy引擎发送到调度的请求和响应。*

Scrapy运行流程大概如下：

1. 引擎从调度器中取出一个链接(URL)用于接下来的抓取
2. 引擎把URL封装成一个请求(Request)传给下载器
3. 下载器把资源下载下来，并封装成应答包(Response)
4. 爬虫解析Response
5. 解析出实体（Item）,则交给实体管道进行进一步的处理
6. 解析出的是链接（URL）,则把URL交给调度器等待抓取

**一、安装**

因为python3并不能完全支持Scrapy，因此为了完美运行Scrapy，我们使用python2.7来编写和运行Scrapy。



|  |  |
| --- | --- |
| 1 | pip install Scrapy |

注：windows平台需要依赖pywin32，请根据自己系统32/64位选择下载安装，https://sourceforge.net/projects/pywin32/

其它可能依赖的安装包：lxml-3.6.4-cp27-cp27m-win\_amd64.whl，VCForPython27.msi百度下载即可

**二、基本使用**

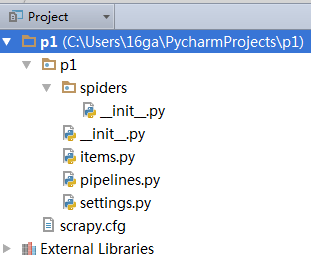
1、创建项目

运行命令:



|  |  |
| --- | --- |
| 1 | scrapy startproject p1（your\_project\_name） |

2.自动创建目录的结果：



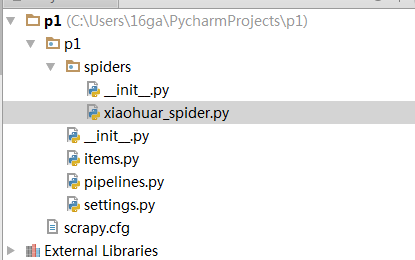
文件说明：

* scrapy.cfg  项目的配置信息，主要为Scrapy命令行工具提供一个基础的配置信息。（真正爬虫相关的配置信息在settings.py文件中）
* items.py    设置数据存储模板，用于结构化数据，如：Django的Model
* pipelines    数据处理行为，如：一般结构化的数据持久化
* settings.py 配置文件，如：递归的层数、并发数，延迟下载等
* spiders      爬虫目录，如：创建文件，编写爬虫规则

注意：一般创建爬虫文件时，以网站域名命名

3、编写爬虫

在spiders目录中新建 xiaohuar\_spider.py 文件



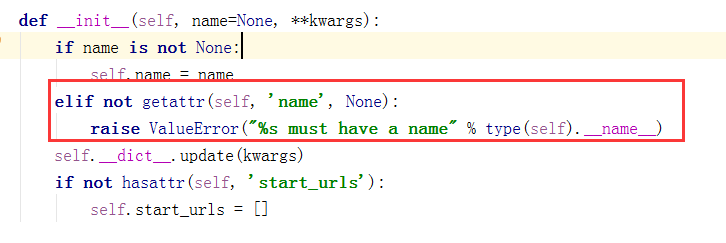
示例代码：



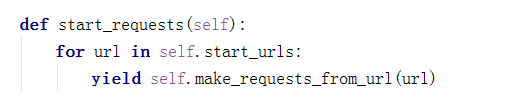
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #!/usr/bin/env python  # -\*- coding:utf-8 -\*-  import scrapy  class XiaoHuarSpider(scrapy.spiders.Spider):      name = "xiaohuar"      allowed\_domains = ["xiaohuar.com"]      start\_urls = [          "http://www.xiaohuar.com/hua/",      ]      def parse(self, response):          # print(response, type(response))          # from scrapy.http.response.html import HtmlResponse          # print(response.body\_as\_unicode())          current\_url = response.url #爬取时请求的url          body = response.body  #返回的html          unicode\_body = response.body\_as\_unicode()#返回的html unicode编码 |

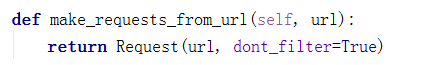
备注：

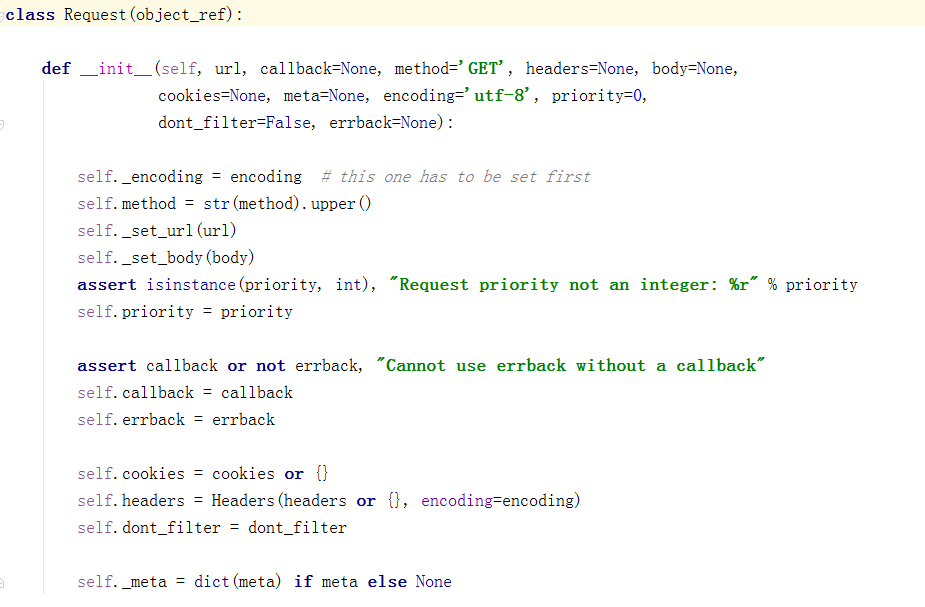
* 1.爬虫文件需要定义一个类，并继承scrapy.spiders.Spider
* 2.必须定义name，即爬虫名，如果没有name，会报错。因为源码中是这样定义的：



* 3.编写函数parse，这里需要注意的是，该函数名不能改变，因为Scrapy源码中默认callback函数的函数名就是parse；
* 4.定义需要爬取的url，放在列表中，因为可以爬取多个url，Scrapy源码是一个For循环，从上到下爬取这些url，使用生成器迭代将url发送给下载器下载url的html。源码截图：







4、运行

进入p1目录，运行命令



|  |  |
| --- | --- |
| 1 | scrapy crawl xiaohau --nolog |

格式：scrapy crawl+爬虫名  –nolog即不显示日志

5.scrapy查询语法：

当我们爬取大量的网页，如果自己写正则匹配，会很麻烦，也很浪费时间，令人欣慰的是，scrapy内部支持更简单的查询语法，帮助我们去html中查询我们需要的标签和标签内容以及标签属性。下面逐一进行介绍：

* 查询子子孙孙中的某个标签(以div标签为例)：//div
* 查询儿子中的某个标签(以div标签为例)：/div
* 查询标签中带有某个class属性的标签：//div[@class=’c1′]即子子孙孙中标签是div且class=‘c1’的标签
* 查询标签中带有某个class=‘c1’并且自定义属性name=‘alex’的标签：//div[@class=’c1′][@name=’alex’]
* 查询某个标签的文本内容：//div/span/text() 即查询子子孙孙中div下面的span标签中的文本内容
* 查询某个属性的值（例如查询a标签的href属性）：//a/@href

示例代码：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | def parse(self, response):         # 分析页面         # 找到页面中符合规则的内容（校花图片），保存         # 找到所有的a标签，再访问其他a标签，一层一层的搞下去         hxs = HtmlXPathSelector(response)#创建查询对象         # 如果url是 http://www.xiaohuar.com/list-1-\d+.html         if re.match('http://www.xiaohuar.com/list-1-\d+.html', response.url): #如果url能够匹配到需要爬取的url，即本站url             items = hxs.select('//div[@class="item\_list infinite\_scroll"]/div') #select中填写查询目标，按scrapy查询语法书写             for i in range(len(items)):                 src = hxs.select('//div[@class="item\_list infinite\_scroll"]/div[%d]//div[@class="img"]/a/img/@src' % i).extract()#查询所有img标签的src属性，即获取校花图片地址                 name = hxs.select('//div[@class="item\_list infinite\_scroll"]/div[%d]//div[@class="img"]/span/text()' % i).extract() #获取span的文本内容，即校花姓名                 school = hxs.select('//div[@class="item\_list infinite\_scroll"]/div[%d]//div[@class="img"]/div[@class="btns"]/a/text()' % i).extract() #校花学校                 if src:                     ab\_src = "http://www.xiaohuar.com" + src[0]#相对路径拼接                     file\_name = "%s\_%s.jpg" % (school[0].encode('utf-8'), name[0].encode('utf-8')) #文件名，因为python27默认编码格式是unicode编码，因此我们需要编码成utf-8                     file\_path = os.path.join("/Users/wupeiqi/PycharmProjects/beauty/pic", file\_name)                     urllib.urlretrieve(ab\_src, file\_path) |

注：urllib.urlretrieve(ab\_src, file\_path) ，接收文件路径和需要保存的路径，会自动去文件路径下载并保存到我们指定的本地路径。

5.递归爬取网页

上述代码仅仅实现了一个url的爬取，如果该url的爬取的内容中包含了其他url，而我们也想对其进行爬取，那么如何实现递归爬取网页呢？

示例代码：



|  |  |
| --- | --- |
| 1  2  3  4  5 | # 获取所有的url，继续访问，并在其中寻找相同的url          all\_urls = hxs.select('//a/@href').extract()          for url in all\_urls:              if url.startswith('http://www.xiaohuar.com/list-1-'):                  yield Request(url, callback=self.parse) |

即通过yield生成器向每一个url发送request请求，并执行返回函数parse，从而递归获取校花图片和校花姓名学校等信息。

注：可以修改settings.py 中的配置文件，以此来指定“递归”的层数，如： DEPTH\_LIMIT = 1

6.scrapy查询语法中的正则：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | from scrapy.selector import Selector  from scrapy.http import HtmlResponse  html = """<!DOCTYPE html>  <html>  <head lang="en">      <meta charset="UTF-8">      <title></title>  </head>  <body>      <li class="item-"><a href="link.html">first item</a></li>      <li class="item-0"><a href="link1.html">first item</a></li>      <li class="item-1"><a href="link2.html">second item</a></li>  </body>  </html>  """  response = HtmlResponse(url='http://example.com', body=html,encoding='utf-8')  ret = Selector(response=response).xpath('//li[re:test(@class, "item-\d\*")]//@href').extract()  print(ret) |

* 语法规则：Selector(response=response查询对象).xpath(‘//li[re:test(@class, “item-d\*”)]//@href’).extract()，即根据re正则匹配，test即匹配，属性名是class，匹配的正则表达式是”item-d\*”，然后获取该标签的href属性。



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63 | #!/usr/bin/env python  # -\*- coding:utf-8 -\*-    import scrapy  import hashlib  from tutorial.items import JinLuoSiItem  from scrapy.http import Request  from scrapy.selector import HtmlXPathSelector      class JinLuoSiSpider(scrapy.spiders.Spider):      count = 0      url\_set = set()        name = "jluosi"      domain = 'http://www.jluosi.com'      allowed\_domains = ["jluosi.com"]        start\_urls = [          "http://www.jluosi.com:80/ec/goodsDetail.action?jls=QjRDNEIzMzAzOEZFNEE3NQ==",      ]        def parse(self, response):          md5\_obj = hashlib.md5()          md5\_obj.update(response.url)          md5\_url = md5\_obj.hexdigest()          if md5\_url in JinLuoSiSpider.url\_set:              pass          else:              JinLuoSiSpider.url\_set.add(md5\_url)              hxs = HtmlXPathSelector(response)              if response.url.startswith('http://www.jluosi.com:80/ec/goodsDetail.action'):                  item = JinLuoSiItem()                  item['company'] = hxs.select('//div[@class="ShopAddress"]/ul/li[1]/text()').extract()                  item['link'] = hxs.select('//div[@class="ShopAddress"]/ul/li[2]/text()').extract()                  item['qq'] = hxs.select('//div[@class="ShopAddress"]//a/@href').re('.\*uin=(?P<qq>\d\*)&')                  item['address'] = hxs.select('//div[@class="ShopAddress"]/ul/li[4]/text()').extract()                    item['title'] = hxs.select('//h1[@class="goodsDetail\_goodsName"]/text()').extract()                    item['unit'] = hxs.select('//table[@class="R\_WebDetail\_content\_tab"]//tr[1]//td[3]/text()').extract()                  product\_list = []                  product\_tr = hxs.select('//table[@class="R\_WebDetail\_content\_tab"]//tr')                  for i in range(2,len(product\_tr)):                      temp = {                          'standard':hxs.select('//table[@class="R\_WebDetail\_content\_tab"]//tr[%d]//td[2]/text()' %i).extract()[0].strip(),                          'price':hxs.select('//table[@class="R\_WebDetail\_content\_tab"]//tr[%d]//td[3]/text()' %i).extract()[0].strip(),                      }                      product\_list.append(temp)                    item['product\_list'] = product\_list                  yield item                current\_page\_urls = hxs.select('//a/@href').extract()              for i in range(len(current\_page\_urls)):                  url = current\_page\_urls[i]                  if url.startswith('http://www.jluosi.com'):                      url\_ab = url                      yield Request(url\_ab, callback=self.parse)    选择器规则Demo    选择器规则Demo |

选择器规则Demo



|  |  |
| --- | --- |
| 1  2  3  4  5 | def parse(self, response):      from scrapy.http.cookies import CookieJar      cookieJar = CookieJar()      cookieJar.extract\_cookies(response, response.request)      print(cookieJar.\_cookies) |

获取响应cookie

更多选择器规则：http://scrapy-chs.readthedocs.io/zh\_CN/latest/topics/selectors.html

7、格式化处理

上述实例只是简单的图片处理，所以在parse方法中直接处理。如果对于想要获取更多的数据（获取页面的价格、商品名称、QQ等），则可以利用Scrapy的items将数据格式化，然后统一交由pipelines来处理。即不同功能用不同文件实现。

items：即用户需要爬取哪些数据，是用来格式化数据，并告诉pipelines哪些数据需要保存。

示例items.py文件：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # -\*- coding: utf-8 -\*-  # Define here the models for your scraped items  #  # See documentation in:  # http://doc.scrapy.org/en/latest/topics/items.html  import scrapy  class JieYiCaiItem(scrapy.Item):      company = scrapy.Field()      title = scrapy.Field()      qq = scrapy.Field()      info = scrapy.Field()      more = scrapy.Field() |

即：需要爬取所有url中的公司名，title，qq，基本信息info，更多信息more。

上述定义模板，以后对于从请求的源码中获取的数据同样按照此结构来获取，所以在spider中需要有一下操作：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | #!/usr/bin/env python  # -\*- coding:utf-8 -\*-    import scrapy  import hashlib  from beauty.items import JieYiCaiItem  from scrapy.http import Request  from scrapy.selector import HtmlXPathSelector  from scrapy.spiders import CrawlSpider, Rule  from scrapy.linkextractors import LinkExtractor      class JieYiCaiSpider(scrapy.spiders.Spider):      count = 0      url\_set = set()        name = "jieyicai"      domain = 'http://www.jieyicai.com'      allowed\_domains = ["jieyicai.com"]        start\_urls = [          "http://www.jieyicai.com",      ]        rules = [          #下面是符合规则的网址,但是不抓取内容,只是提取该页的链接(这里网址是虚构的,实际使用时请替换)          #Rule(SgmlLinkExtractor(allow=(r'http://test\_url/test?page\_index=\d+'))),          #下面是符合规则的网址,提取内容,(这里网址是虚构的,实际使用时请替换)          #Rule(LinkExtractor(allow=(r'http://www.jieyicai.com/Product/Detail.aspx?pid=\d+')), callback="parse"),      ]        def parse(self, response):          md5\_obj = hashlib.md5()          md5\_obj.update(response.url)          md5\_url = md5\_obj.hexdigest()          if md5\_url in JieYiCaiSpider.url\_set:              pass          else:              JieYiCaiSpider.url\_set.add(md5\_url)                hxs = HtmlXPathSelector(response)              if response.url.startswith('http://www.jieyicai.com/Product/Detail.aspx'):                  item = JieYiCaiItem()                  item['company'] = hxs.select('//span[@class="username g-fs-14"]/text()').extract()                  item['qq'] = hxs.select('//span[@class="g-left bor1qq"]/a/@href').re('.\*uin=(?P<qq>\d\*)&')                  item['info'] = hxs.select('//div[@class="padd20 bor1 comard"]/text()').extract()                  item['more'] = hxs.select('//li[@class="style4"]/a/@href').extract()                  item['title'] = hxs.select('//div[@class="g-left prodetail-text"]/h2/text()').extract()                  yield item                current\_page\_urls = hxs.select('//a/@href').extract()              for i in range(len(current\_page\_urls)):                  url = current\_page\_urls[i]                  if url.startswith('/'):                      url\_ab = JieYiCaiSpider.domain + url                      yield Request(url\_ab, callback=self.parse)    spider    spider |

spider

上述代码中：对url进行md5加密的目的是避免url过长，也方便保存在缓存或数据库中。

此处代码的关键在于：

* 将获取的数据封装在了Item对象中
* yield Item对象 （一旦parse中执行yield Item对象，则自动将该对象交个pipelines的类来处理）



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 | # -\*- coding: utf-8 -\*-    # Define your item pipelines here  #  # Don't forget to add your pipeline to the ITEM\_PIPELINES setting  # See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html    import json  from twisted.enterprise import adbapi  import MySQLdb.cursors  import re    mobile\_re = re.compile(r'(13[0-9]|15[012356789]|17[678]|18[0-9]|14[57])[0-9]{8}')  phone\_re = re.compile(r'(\d+-\d+|\d+)')    class JsonPipeline(object):        def \_\_init\_\_(self):          self.file = open('/Users/wupeiqi/PycharmProjects/beauty/beauty/jieyicai.json', 'wb')          def process\_item(self, item, spider):          line = "%s  %s\n" % (item['company'][0].encode('utf-8'), item['title'][0].encode('utf-8'))          self.file.write(line)          return item    class DBPipeline(object):        def \_\_init\_\_(self):          self.db\_pool = adbapi.ConnectionPool('MySQLdb',                                               db='DbCenter',                                               user='root',                                               passwd='123',                                               cursorclass=MySQLdb.cursors.DictCursor,                                               use\_unicode=True)        def process\_item(self, item, spider):          query = self.db\_pool.runInteraction(self.\_conditional\_insert, item)          query.addErrback(self.handle\_error)          return item        def \_conditional\_insert(self, tx, item):          tx.execute("select nid from company where company = %s", (item['company'][0], ))          result = tx.fetchone()          if result:              pass          else:              phone\_obj = phone\_re.search(item['info'][0].strip())              phone = phone\_obj.group() if phone\_obj else ' '                mobile\_obj = mobile\_re.search(item['info'][1].strip())              mobile = mobile\_obj.group() if mobile\_obj else ' '                values = (                  item['company'][0],                  item['qq'][0],                  phone,                  mobile,                  item['info'][2].strip(),                  item['more'][0])              tx.execute("insert into company(company,qq,phone,mobile,address,more) values(%s,%s,%s,%s,%s,%s)", values)        def handle\_error(self, e):          print 'error',e    pipelines    pipelines |

上述代码中多个类的目的是，可以同时保存在文件和数据库中，保存的优先级可以在配置文件settings中定义。



|  |  |
| --- | --- |
| 1  2  3  4  5 | ITEM\_PIPELINES = {      'beauty.pipelines.DBPipeline': 300,      'beauty.pipelines.JsonPipeline': 100,  }  # 每行后面的整型值，确定了他们运行的顺序，item按数字从低到高的顺序，通过pipeline，通常将这些数字定义在0-1000范围内。 |