**Scrapy框架中Spider的用法**

在Scrapy里面，要抓取网站的链接配置、抓取逻辑、解析逻辑里，都是在Spider里面去完成的。

**一、Spider的运行流程**

在实现Scrapy爬虫项目里面，最核心的就是Spider类了，它定义了如何爬取某个网站的流程和解析方式。简单来讲，Spider就做两件事情：

* 定义爬取的动作
* 分析爬取下来的网页

对于Spider类来说，整个流程如下，可以参考[中文官方文档](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/spiders.html)：

* 以初始的URL初始化Request，并设置回调函数。 当该request下载完毕并返回时，将生成response，并作为参数传给该回调函数。spider中初始的request是通过调用 [**start\_requests()**](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/spiders.html#scrapy.spider.Spider.start_requests) 来获取的。 [**start\_requests()**](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/spiders.html#scrapy.spider.Spider.start_requests) 读取 [**start\_urls**](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/spiders.html#scrapy.spider.Spider.start_urls) 中 的URL， 并以 [**parse**](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/spiders.html#scrapy.spider.Spider.parse) 为回调函数生成 [**Request**](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/request-response.html#scrapy.http.Request) 。
* 在回调函数内分析返回的(网页)内容，返回 [**Item**](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/items.html#scrapy.item.Item) 对象或者 [**Request**](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/request-response.html#scrapy.http.Request) 或者一个包括二者的可迭代容器。 返回的Request对象之后会经过Scrapy处理，下载相应的内容，并调用设置的callback函数(函数可相同)。
* 在回调函数内，您可以使用 [选择器(Selectors)](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/selectors.html#topics-selectors) (您也可以使用BeautifulSoup, lxml 或者您想用的任何解析器) 来分析网页内容，并根据分析的数据生成item。
* 最后，由spider返回的item将被存到数据库(由某些 [Item Pipeline](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/item-pipeline.html#topics-item-pipeline) 处理)或使用 [Feed exports](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/feed-exports.html#topics-feed-exports) 存入到文件中。

通过以上几步的循环往复的进行，就可以完成站点的爬取。

**二、Spider类分析**

[**scrapy.Spider.[namespider**](http://scrapy-chs.readthedocs.io/zh_CN/0.24/topics/spiders.html)**] 例如**class JdbraSpider(scrapy.Spider)中JdbraSpider就是namespider的名字，在使用scrapy genspider Jdbra时，会自动创建这样一个类。Spider并没有提供什么特殊的功能。 其仅仅请求给定的 **start\_urls**/**start\_requests** ，并根据返回的结果(resulting responses)调用spider的 **parse** 方法。

**2.1 name**

定义spider名字的字符串(string)。spider的名字定义了Scrapy如何定位(并初始化)spider，所以其必须是唯一的。 不过您可以生成多个相同的spider实例(instance)，这没有任何限制。 name是spider最重要的属性，而且是必须的。

如果该spider爬取单个网站(single domain)，一个常见的做法是以该网站(domain)(加或不加 [后缀](http://en.wikipedia.org/wiki/Top-level_domain) )来命名spider。 例如，如果spider爬取 **mywebsite.com** ，该spider通常会被命名为 **mywebsite** 。

**2.2 allow\_domains**

允许爬取的域名，是可选配置，不在此域名范围的链接，不会被跟进爬取。

**2.3 start\_urls**

URL列表。当没有制定特定的URL时，spider将从该列表中开始进行爬取。 因此，第一个被获取到的页面的URL将是该列表之一。 后续的URL将会从获取到的数据中提取。

**2.4 custom\_settings**

它是一个字典，是专属于Spider的配置，此方法会覆盖全局的配置，此设置必须在初始化前被更新，必须定义成类变量。

**理解的操作方式：**

在之前的项目里面，创建一个知乎的Spider：

在start\_urls的链接改成start\_urls = ['http://www.zhihu.com/explore']

在settings.py里面DEFAULT\_REQUEST\_HEADERS加入user-agent信息

DEFAULT\_REQUEST\_HEADERS = {

'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8',

'Accept-Language': 'en',

'user-agent':'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36',

}

先运行下给spider看看有什么问题：

会看到，返回200的状态码：

在zhihu.py文件里面加入custom\_settings相关配置：

custom\_settings={

'DEFAULT\_REQUEST\_HEADERS':{

'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8',

'Accept-Language': 'en',

'user-agent': None,

}

}

在运行zhihu看看有什么问题：

请求时出现了400的状态码，说明custom\_settings相关配置已经生效，得到了错误的响应。

说明custom\_settings用来覆盖settings的全局配置。

**2.5 crawler**

它是由from\_crawler方法设置的，代表本Spider对应的Crawler对象，包含了许多项目组件。我们可以利用它来获取项目中的一些配置信息，最常见的就是从settings.py里面获取项目的配置信息。

**2.6 from\_crawler()**

使用该方法可以获取Crawler对象里面的项目组件配置信息。此方法和Pipeline里面使用是一样的。

**2.7 start\_requests()**

此方法用于生成初始请求，它必须必须返回一个可迭代对象。此方法会默认使用start\_urls里面的URL来构建Request，而且Request是以GET方式进行请求。如果我们想在启动时，想以POST的请求方式访问某个网站，可以直接重写这个方法。

**理解的操作方式：（http.org是一个进行http请求方式模拟的一个网站）**

重新创建一个httpbin的Spider：

原封不动进行执行：

看效果，完成的是get请求：

先对start\_urls改成start\_urls = ['http://httpbin.org/post']

再执行，出现了以下情况：

接下来对start\_requests()进行改写：

def start\_requests(self):

yield scrapy.Request(url='http://httpbin.org/post',method='POST',callback=self.parse\_post)

def parse(self, response):

pass

def parse\_post(self,response):

print('请求成功：',response.status)

再次执行，可以看到如下信息：

**2.8 make\_requests\_from\_url(*url*)**

该方法接受一个URL并返回用于爬取的 [**Request**](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/request-response.html#scrapy.http.Request) 对象。 该方法在初始化request时被 [**start\_requests()**](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/spiders.html#scrapy.spiders.Spider.start_requests) 调用，也被用于转化url为request。

默认未被复写(overridden)的情况下，该方法返回的Request对象中， [**parse()**](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/spiders.html#scrapy.spiders.Spider.parse) 作为回调函数，dont\_filter参数也被设置为开启。 (详情参见 [**Request**](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/request-response.html#scrapy.http.Request))。

**理解的操作方式：**

重新创建一个baidu的Spider：

改写make\_requests\_from\_url()，改变回调函数：

def make\_requests\_from\_url(self, url):

return scrapy.Request(url=url,callback=self.parse\_page)

def parse(self, response):

pass

def parse\_page(self,response):

print(response.status)

进行执行：

通过上面该写可以看到，url直接来走start\_urls里面的元素，还可以直接改变回调函数

如果我们在加入改写的start\_request()，就不会再调用make\_requests\_from\_url()方法。

def start\_requests(self):

yield scrapy.Request(url='http://www.baidu.com/',callback=self.parse\_index)

def parse\_index(self,response):

print('调用start\_resquest()方法')

def make\_requests\_from\_url(self, url):

return scrapy.Request(url=url,callback=self.parse\_page)

def parse(self, response):

pass

def parse\_page(self,response):

print(response.status）

self.logger.info(response.status)

执行spider之后就会看到如下信息：

**2.9 parse()**

当response没有指定回调函数时，该方法是Scrapy处理下载的response的默认方法。

**parse** 负责处理response并返回处理的数据以及(/或)跟进的URL。 [**Spider**](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/spiders.html#scrapy.spiders.Spider) 对其他的Request的回调函数也有相同的要求。

该方法及其他的Request回调函数必须返回一个包含 [**Request**](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/request-response.html#scrapy.http.Request)、dict 或 [**Item**](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/items.html#scrapy.item.Item) 的可迭代的对象。

**2.10 logger()**

日志输出的方法，有info()和DEBUG()方法，可以输出日志的诶输出信息：方法见2.8最后一段代码。

**2.11 close()**

当spider关闭时，该函数被调用。有一个参数reason，表示当前参数中断的原因。

def close(self,spider, reason):  
 print('++++++++++++')  
 self.logger.debug(reason)