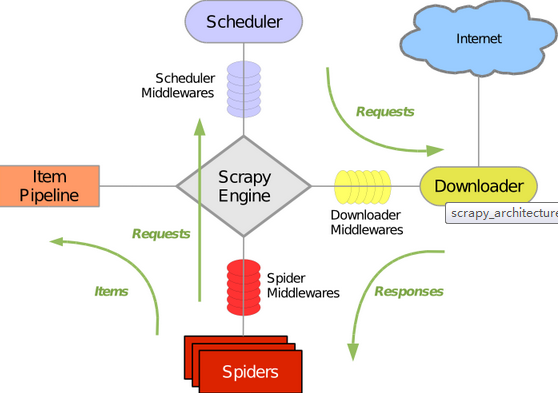
1. 开始使用scrapy:



**新建工程**

**在抓取之前，你需要新建一个Scrapy工程。进入一个你想用来保存代码的目录，然后执行：**

**Microsoft Windows XP [Version 5.1.2600]**

**(C) Copyright 1985-2001 Microsoft Corp.**

**T:\>scrapy startproject tutorial**

**T:\>**

**这个命令会在当前目录下创建一个新目录tutorial，它的结构如下：**

**T:\tutorial>tree /f**

**Folder PATH listing**

**Volume serial number is 0006EFCF C86A:7C52**

**T:.**

**│ scrapy.cfg**

**│**

**└─tutorial**

**│ items.py**

**│ pipelines.py**

**│ settings.py**

**│ \_\_init\_\_.py**

**│**

**└─spiders**

**\_\_init\_\_.py**

**这些文件主要是：**

**scrapy.cfg: 项目配置文件**

**tutorial/: 项目python模块, 呆会代码将从这里导入**

**tutorial/items.py: 项目items文件**

**tutorial/pipelines.py: 项目管道文件**

**tutorial/settings.py: 项目配置文件**

**tutorial/spiders: 放置spider的目录**

**定义Item**

**Items是将要装载抓取的数据的容器，它工作方式像python里面的字典，但它提供更多的保护，比如对未定义的字段填充以防止拼写错误。**

**它通过创建一个scrapy.item.Item类来声明，定义它的属性为scrpy.item.Field对象，就像是一个对象关系映射(ORM).**

**我们通过将需要的item模型化，来控制从dmoz.org获得的站点数据，比如我们要获得站点的名字，url和网站描述，我们定义这三种属性的域。要做到这点，我们编辑在tutorial目录下的items.py文件，我们的Item类将会是这样**

**from scrapy.item import Item, Field**

**class DmozItem(Item):**

**title = Field()**

**link = Field()**

**desc = Field()**

**刚开始看起来可能会有些困惑，但是定义这些item能让你用其他Scrapy组件的时候知道你的 items到底是什么。**

**我们的第一个爬虫(Spider)**

**Spider是用户编写的类，用于从一个域（或域组）中抓取信息。**

**他们定义了用于下载的URL的初步列表，如何跟踪链接，以及如何来解析这些网页的内容用于提取items。**

**要建立一个Spider，你必须为scrapy.spider.BaseSpider创建一个子类，并确定三个主要的、强制的属性：**

**name：爬虫的识别名，它必须是唯一的，在不同的爬虫中你必须定义不同的名字.**

**start\_urls：爬虫开始爬的一个URL列表。爬虫从这里开始抓取数据，所以，第一次下载的数据将会从这些URLS开始。其他子URL将会从这些起始URL中继承性生成。**

**parse()：爬虫的方法，调用时候传入从每一个URL传回的Response对象作为参数，response将会是parse方法的唯一的一个参数,**

**这个方法负责解析返回的数据、匹配抓取的数据(解析为item)并跟踪更多的URL。**

**这是我们的第一只爬虫的代码，将其命名为dmoz\_spider.py并保存在tutorial\spiders目录下。**

**from scrapy.spider import BaseSpider**

**class DmozSpider(BaseSpider):**

**name = "dmoz"**

**allowed\_domains = ["dmoz.org"]**

**start\_urls = [**

**"http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",**

**"http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"**

**]**

**def parse(self, response):**

**filename = response.url.split("/")[-2]**

**open(filename, 'wb').write(response.body)**

**为了让我们的爬虫工作，我们返回项目主目录执行以下命令**

**T:\tutorial>scrapy crawl dmoz**

**crawl dmoz 命令从dmoz.org域启动爬虫。 你将会获得如下类似输出**

**T:\tutorial>scrapy crawl dmoz**

**Scrapy为爬虫的 start\_urls属性中的每个URL创建了一个 scrapy.http.Request 对象 ，并将爬虫的parse 方法指定为回调函数。**

**这些 Request首先被调度，然后被执行，之后通过parse()方法，scrapy.http.Response 对象被返回，结果也被反馈给爬虫。**

**提取Item**

**选择器介绍**

**我们有很多方法从网站中提取数据。Scrapy 使用一种叫做 XPath selectors的机制，它基于 XPath表达式。如果你想了解更多selectors和其他机制你可以查阅资料http://doc.scrapy.org/topics/selectors.html#topics-selectors**

**这是一些XPath表达式的例子和他们的含义**

**/html/head/title: 选择HTML文档<head>元素下面的<title> 标签。**

**/html/head/title/text(): 选择前面提到的<title> 元素下面的文本内容**

**//td: 选择所有 <td> 元素**

**//div[@class="mine"]: 选择所有包含 class="mine" 属性的div 标签元素**

**这只是几个使用XPath的简单例子，但是实际上XPath非常强大。如果你想了解更多XPATH的内容，我们向你推荐这个XPath教程http://www.w3schools.com/XPath/default.asp**

**为了方便使用XPaths，Scrapy提供XPathSelector 类， 有两种口味可以选择， HtmlXPathSelector (HTML数据解析) 和XmlXPathSelector (XML数据解析)。 为了使用他们你必须通过一个 Response 对象对他们进行实例化操作。你会发现Selector对象展示了文档的节点结构。因此，第一个实例化的selector必与根节点或者是整个目录有关 。**

**Selectors 有三种方法**

**select()：返回selectors列表, 每一个select表示一个xpath参数表达式选择的节点.**

**extract()：返回一个unicode字符串，该字符串为XPath选择器返回的数据**

**re()： 返回unicode字符串列表，字符串作为参数由正则表达式提取出来**

**尝试在shell中使用Selectors**

**为了演示Selectors的用法，我们将用到 内建的Scrapy shell，这需要系统已经安装IPython (一个扩展python交互环境) 。**

**附IPython下载地址：http://pypi.python.org/pypi/ipython#downloads**

**要开始shell，首先进入项目顶层目录，然后输入**

**T:\tutorial>scrapy shell http://www.dmoz.org/Computers/Programming/Languages/Python/Books/**

**Shell载入后，你将获得回应，这些内容被存储在本地变量 response 中，所以如果你输入response.body 你将会看到response的body部分，或者输入response.headers 来查看它的 header部分。**

**提取数据**

**现在我们尝试从网页中提取数据。**

**你可以在控制台输入 response.body， 检查源代码中的 XPaths 是否与预期相同。然而，检查HTML源代码是件很枯燥的事情。为了使事情变得简单，我们使用Firefox的扩展插件Firebug。更多信息请查看Using Firebug for scraping 和Using Firefox for scraping.**

**txw1958注：事实上我用的是Google Chrome的Inspect Element功能，而且可以提取元素的XPath。**

**检查源代码后，你会发现我们需要的数据在一个 <ul>元素中，而且是第二个<ul>。**

**我们可以通过如下命令选择每个在网站中的 <li> 元素:**

**hxs.select('//ul/li')**

**然后是网站描述:**

**hxs.select('//ul/li/text()').extract()**

**网站标题:**

**hxs.select('//ul/li/a/text()').extract()**

**网站链接:**

**hxs.select('//ul/li/a/@href').extract()**

**如前所述，每个select()调用返回一个selectors列表，所以我们可以结合select()去挖掘更深的节点。我们将会用到这些特性，所以:**

**sites = hxs.select('//ul/li')**

**for site in sites:**

**title = site.select('a/text()').extract()**

**link = site.select('a/@href').extract()**

**desc = site.select('text()').extract()**

**print title, link, desc**

**Note**

**更多关于嵌套选择器的内容，请阅读Nesting selectors 和 Working with relative XPaths**

**将代码添加到爬虫中：**

**txw1958注：代码有修改，绿色注释掉的代码为原教程的，你懂的**

**from scrapy.spider import BaseSpider**

**from scrapy.selector import HtmlXPathSelector**

**class DmozSpider(BaseSpider):**

**name = "dmoz"**

**allowed\_domains = ["dmoz.org"]**

**start\_urls = [**

**"http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",**

**"http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"**

**]**

**def parse(self, response):**

**hxs = HtmlXPathSelector(response)**

**sites = hxs.select('//fieldset/ul/li')**

**#sites = hxs.select('//ul/li')**

**for site in sites:**

**title = site.select('a/text()').extract()**

**link = site.select('a/@href').extract()**

**desc = site.select('text()').extract()**

**#print title, link, desc**

**print title, link**

**现在我们再次抓取dmoz.org，你将看到站点在输出中被打印 ，运行命令**

**T:\tutorial>scrapy crawl dmoz**

**使用条目(Item)**

**Item 对象是自定义的python字典，使用标准字典类似的语法，你可以获取某个字段(即之前定义的类的属性)的值:**

**>>> item = DmozItem()**

**>>> item['title'] = 'Example title'**

**>>> item['title']**

**'Example title'**

**Spiders希望将其抓取的数据存放到Item对象中。为了返回我们抓取数据，spider的最终代码应当是这样:**

**from scrapy.spider import BaseSpider**

**from scrapy.selector import HtmlXPathSelector**

**from tutorial.items import DmozItem**

**class DmozSpider(BaseSpider):**

**name = "dmoz"**

**allowed\_domains = ["dmoz.org"]**

**start\_urls = [**

**"http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",**

**"http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"**

**]**

**def parse(self, response):**

**hxs = HtmlXPathSelector(response)**

**sites = hxs.select('//fieldset/ul/li')**

**#sites = hxs.select('//ul/li')**

**items = []**

**for site in sites:**

**item = DmozItem()**

**item['title'] = site.select('a/text()').extract()**

**item['link'] = site.select('a/@href').extract()**

**item['desc'] = site.select('text()').extract()**

**items.append(item)**

**return items**

**保存抓取的数据**

**保存信息的最简单的方法是通过Feed exports，命令如下：**

**T:\tutorial>scrapy crawl dmoz -o items.json -t json**

**所有抓取的items将以JSON格式被保存在新生成的items.json 文件中**

**在像本教程一样的小型项目中，这些已经足够。然而，如果你想用抓取的items做更复杂的事情，你可以写一个 Item Pipeline(条目管道)。因为在项目创建的时候，一个专门用于条目管道的占位符文件已经随着items一起被建立，目录在tutorial/pipelines.py。如果你只需要存取这些抓取后的items的话，就不需要去实现任何的条目管道。**

**实现PipeLine**

PipeLine用来对Spider返回的Item列表进行保存操作，可以写入到文件、或者数据库等。

PipeLine只有一个需要实现的方法：process\_item，例如我们将Item保存到一个文件中：

**[python]**[view plaincopyprint?](http://blog.csdn.net/zbyufei/article/details/7554322)

1. def \_\_init\_\_(self):
2. self.file = open('jingdong.txt', 'wb')
4. def process\_item(self, item, spider):
5. self.file.write(item['title'] + '\t'+ item['link'] + '\t' + item['desc']+'\n')

def \_\_init\_\_(self):    self.file = open('jingdong.txt', 'wb') def process\_item(self, item, spider):    self.file.write(item['title'] + '\t'+ item['link'] + '\t' + item['desc']+'\n')

到现在，我们就完成了一个基本的爬虫的实现，可以输入下面的命令来启动这个Spider:

**[python]**[view plaincopyprint?](http://blog.csdn.net/zbyufei/article/details/7554322)

1. scrapy crawl dmoz.org

scrapy crawl dmoz.org

Scrapy之URL解析与递归爬取：

前面介绍了Scrapy如何实现一个最简单的爬虫，但是这个Demo里只是对一个页面进行了抓取。在实际应用中，爬虫一个重要功能是”发现新页面”，然后递归的让爬取操作进行下去。

发现新页面的方法很简单，我们首先定义一个爬虫的入口URL地址，比如Scrapy入门教程中的start\_urls，爬虫首先将这个页面的内容抓取之后，解析其内容，将所有的链接地址提取出来。这个提取的过程是很简单的，通过一个html解析库，将这样的节点内容提取出来，href参数的值就是一个新页面的URL。获取这个URL值之后，将其加入到任务队列中，爬虫不断的从队列中取URL即可。这样，只需要为爬虫定义一个入口的URL，那么爬虫就能够自动的爬取到指定网站的绝大多数页面。

当然，在具体的实现中，我们还需要对提取的URL做进一步处理:

**1. 判断URL指向网站的域名**，如果指向的是外部网站，那么可以将其丢弃  
**2. URL去重**，可以将所有爬取过的URL存入数据库中，然后查询新提取的URL在数据库中是否存在，如果存在的话，当然就无需再去爬取了。

下面介绍一下如何在Scrapy中完成上述这样的功能。

我们只需要改写spider的那个py文件即可，修改parse()方法代码如下：

**[python]**[view plaincopyprint?](http://blog.csdn.net/zbyufei/article/details/7554322)

1. from scrapy.selector import HtmlXPathSelector
3. def parse(self, response):
4. hxs = HtmlXPathSelector(response)
5. items = []
7. newurls = hxs.select('//a/@href').extract()
8. validurls = []
9. for url in newurls:
10. #判断URL是否合法
11. if true:
12. validurls.append(url)
14. items.extend([self.make\_requests\_from\_url(url).replace(callback=self.parse) for url in validurls])
16. sites = hxs.select('//ul/li')
17. items = []
18. for site in sites:
19. item = DmozItem()
20. item['title'] = site.select('a/text()').extract()
21. item['link'] = site.select('a/@href').extract()
22. item['desc'] = site.select('text()').extract()
23. items.append(item)
25. return items