这篇文章假设你的计算机中已经安装好了Scrapy,如果没有安装的好，你可以参考这里 <http://doc.scrapy.org/intro/install.html#intro-install>

///

备注\_ 个人在kubuntu下安装测试，一次按命令顺序安装以下部件 (假设你已经安装python)

下载 tar scrapy, python setup.py安装即可

部件\_ sudo apt-get install python-twisted python-libxml python-simplejson

安装完这些异步等框架后 再装 easy\_install –U w3lib

///

我们将展示如何抓取Open directory project (dmoz)

这篇教程将会带你做以下任务:

1.创建一个新的Scrapy project

2.定义提取的项目

3.写一个抓取站点和提取项目的爬虫

4.编写条目管道(Item Pipeline) 来存储提取的项目

Scrapy由Python语言编写，如果你是Python新手，你也许因为想了解 Scrapy而首先了解Python这门语言。如果你已经对其他的编程语言比较熟悉，想要快速的学习python,我们建议你读 Dive into Python…

**创建一个工程项目**

在你开始编写Scrapy的程序钱，你首先应该建立一个新的Scrapy工程

scrapy startproject dmoz

这样将会在dmoz目录下创建如下结构

dmoz/

scrapy.cfg

dmoz/

\_\_init\_\_.py

items.py

pipelines.py

settings.py

spiders/

\_\_init\_\_.py

...

逐条解释:

Scrapy.cfg : 项目配置文件

Domz/ : 主module，从这里导入代码

Dmoz/items.py : 管道文件

Dmoz/settings.py : 项目设置文件

Dmoz/spiders/ : 爬虫所在

**定义Item**

条目(Items)是一个我们与抓取的数据中间的一个交互，比较像Python的字典，但它提供更多附加的特性比如提供默认值.

它通过建立一个scrapy.item.Item的类来生命，定义它的属性为scrpiy.item.Field对象，就像你在一个ORM中.

我们通过将我们需要的条目模型化来控制从dmoz.org获得的数据，比如我们要获得网站的名字，url和网站描述，我们定义这三种属性的范围，为了达到目的，我们编辑在dmoz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()

开始看起来可能会有些疑惑，但是定义这些能让你用其他组件的时候知道items是如何定义的

**Your First Spider !** (ps 期待 amazing spider man!)

爬虫用来定义url列表来下载，跟踪链接，解析网页来获取条目。为了建立一个爬虫，你要为scrapy.spider.BaseSpider创建一个子类，定义三个属性

[**name**](http://doc.scrapy.org/topics/spiders.html#scrapy.spider.BaseSpider.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**](http://doc.scrapy.org/topics/spiders.html#scrapy.spider.BaseSpider.start_urls):这是一个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。 下面计算的所有子URL将会从这些URL中开始计算

[**parse()**](http://doc.scrapy.org/topics/spiders.html#scrapy.spider.BaseSpider.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response数据匹配抓取的数据(解析为item)和其他的URL

The [**parse()**](http://doc.scrapy.org/topics/spiders.html#scrapy.spider.BaseSpider.parse) method is in charge of processing the response and returning scraped data (as [**Item**](http://doc.scrapy.org/topics/items.html#scrapy.item.Item) objects) and more URLs to follow (as [**Request**](http://doc.scrapy.org/topics/request-response.html#scrapy.http.Request) objects).

**from** **scrapy.spider** **import** BaseSpider

**class** **DmozSpider**(BaseSpider):

name = "dmoz.org"

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)

这是第一个爬虫，保存名为dmoz\_sprder.py在dmoz/spiders文件夹下

**Scrape**

返回项目主目录执行抓取命令 ! ( 与你的spiders文件夹同级)

Scrapy crawl dmoz

获得抓取状态的输出 :

注意有[dmoz.org]的输出，可以看见在start\_url中定义的一些URL的一些输出。因为这些URL是开始页面，所以没有引用，所以在每行的末尾你会看到referer:<None>.

有趣的是，在我们的解析 (parse) 方法的作用下, books和Resources被创建，这两个文件中有着URL的页面内容.

What’s happening ?

Scrapy为爬虫属性中的 start\_urls中的每个URL创建了一个 [scrapy.http.Request](http://doc.scrapy.org/topics/request-response.html" \l "scrapy.http.Request" \o "scrapy.http.Request) 对象 , 为他们指定爬虫的解析方法作为回调

Request被执行, 之后 [scrapy.http.Response](http://doc.scrapy.org/topics/request-response.html" \l "scrapy.http.Response" \o "scrapy.http.Response) 对象通过**[parse()](http://doc.scrapy.org/topics/spiders.html" \l "scrapy.spider.BaseSpider.parse" \o "scrapy.spider.BaseSpider.parse)** 方法返回给spider.

**提取 Items !**

 Selectors入门

为了从网站中提取数据，我们有很多方法. Scrapy 使用基于 [XPath](http://www.w3.org/TR/xpath) 的名字叫做 [XPath selectors](http://doc.scrapy.org/topics/selectors.html#topics-selectors)的机制。如果你想了解更多selectors和其他机制你可以查阅资料

传送门[http://doc.scrapy.org/topics/selectors.html#topics-selectors](http://doc.scrapy.org/topics/selectors.html" \l "topics-selectors)

简单介绍一些XPath表达式的例子和他们分别的含义

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

/html/head/title/text(): 选择在 <title> 元素下面的标签内容.

//td: 选择所有 <td> 标签

//div[@class="mine"]: 选择所有 class="mine" 属性的div 标签

这些只是你可以通过XPath可以做的简单例子,但是XPath实际上非常强大

更多XPATH的内容，我们推荐这个教程<http://www.w3schools.com/XPath/default.asp>

为了方便使用XPaths, Scrapy提供[XPathSelector](http://doc.scrapy.org/topics/selectors.html" \l "scrapy.selector.XPathSelector" \o "scrapy.selector.XPathSelector) 类, 一共有两种, [HtmlXPathSelector](http://doc.scrapy.org/topics/selectors.html" \l "scrapy.selector.HtmlXPathSelector" \o "scrapy.selector.HtmlXPathSelector) (HTML数据解析) 和[XmlXPathSelector](http://doc.scrapy.org/topics/selectors.html" \l "scrapy.selector.XmlXPathSelector" \o "scrapy.selector.XmlXPathSelector) (XML数据解析). 为了使用他们你必须通过一个 [Response](http://doc.scrapy.org/topics/request-response.html#scrapy.http.Response) 对象对他们进行实例化操作

发现Selector对象展示了文档的节点结构.所以，首先被实例化的selector与跟节点或者是整个目录有关 。

Selectors 有三种方法

[**select()**](http://doc.scrapy.org/topics/selectors.html#scrapy.selector.XPathSelector.select): 返回selectors列表, 每一个战士了通过xpath参数表达式而

选择的节点.<http://doc.scrapy.org/topics/selectors.html#scrapy.selector.XPathSelector.select>

[**extract()**](http://doc.scrapy.org/topics/selectors.html#scrapy.selector.XPathSelector.extract): 返回通过XPath选择器选择返回的数据的unicode字符串

[**re()**](http://doc.scrapy.org/topics/selectors.html#scrapy.selector.XPathSelector.re): 返回根据正则表达式匹配的一个unicode编码字符串列表

在交互环境中使用Selectors

为了举例说明Selectors的用法我们将用到 [Scrapy shell](http://doc.scrapy.org/topics/shell.html" \l "topics-shell), 需要使用ipython (一个扩展python交互环境) 。

为了使用交互环境，进入你的项目目录，然后命令

scrapy shell http://www.dmoz.org/Computers/Programming/Languages/Python/Books/

输出结果这样:

[ ... Scrapy log here ... ]

…

…

交互环境载入后,你将会有一个在本地变量 response 中提取的response , 所以如果你输入 response.body 你将会看到response的body部分,或者你可以输入 response.headers 来查看它的 headers.

**提取数据**

现在我们尝试从网页中提取数据.

你可以尝试在控制台输入 response.body , 检查这些 XPaths 来查找你需要的内容.然而，去检查这些节点是一件很枯燥的事情，为了使事情变得简单，你可以使用一些浏览器的工具(比如firebug).

参考如下文档：[http://doc.scrapy.org/topics/firefox.html#topics-firefox](http://doc.scrapy.org/topics/firefox.html" \l "topics-firefox)

<http://doc.scrapy.org/topics/firebug.html#topics-firebug>

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

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')

然后是网页描述:

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

网站标题:

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

网站超级链接:

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

每个 select() 调用返回一个selectors列表, 所以可以结合 select() 调用去查找更深的node. 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

在spider爬虫中加入:

**from** **scrapy.spider** **import** BaseSpider

**from** **scrapy.selector** **import** HtmlXPathSelector

**class** **DmozSpider**(BaseSpider):

name = "dmoz.org"

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('//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

现在再次去抓去dmoz.org

scrapy crawl dmoz.org

使用条目(Item)

[**Item**](http://doc.scrapy.org/topics/items.html#scrapy.item.Item) 实质是python中的字典; 你可以查看某个字典的特定值，通过像下面这样的简单的语法:

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

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

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

'Example title'

Spiders将会返回在 [**Item**](http://doc.scrapy.org/topics/items.html#scrapy.item.Item) 中抓取的信息 ，所以为了返回我们抓取的信息，spider的内容应该是这样:

**from** **scrapy.spider** **import** BaseSpider

**from** **scrapy.selector** **import** HtmlXPathSelector

**from** **dmoz.items** **import** DmozItem

**class** **DmozSpider**(BaseSpider):

name = "dmoz.org"

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('//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

现在我们再次抓取 :

…

…

保存抓取的信息

最简单的保存信息的方法是通过 [Feed exports](http://doc.scrapy.org/topics/feed-exports.html#topics-feed-exports), 命令如下:

scrapy crawl dmoz.org --set FEED\_URI=items.json --set FEED\_FORMAT=json

将会计算出一个包含所有抓取items的 items.json 文件, 已经被[JSON](http://en.wikipedia.org/wiki/JSON)化.

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](http://doc.scrapy.org/topics/item-pipeline.html#topics-item-pipeline)*,* 在建立一个条目的时候，一个专门为项目的一个管道文件已经为这些items建立，目录在  dmoz/pipelines.py. 这样，如果你只是打算将这些抓取后的items保存，就不需要去检查任何的pipeline…

教程简要介绍了Scrapy的使用，但是远远不够…

期待和你一起研究