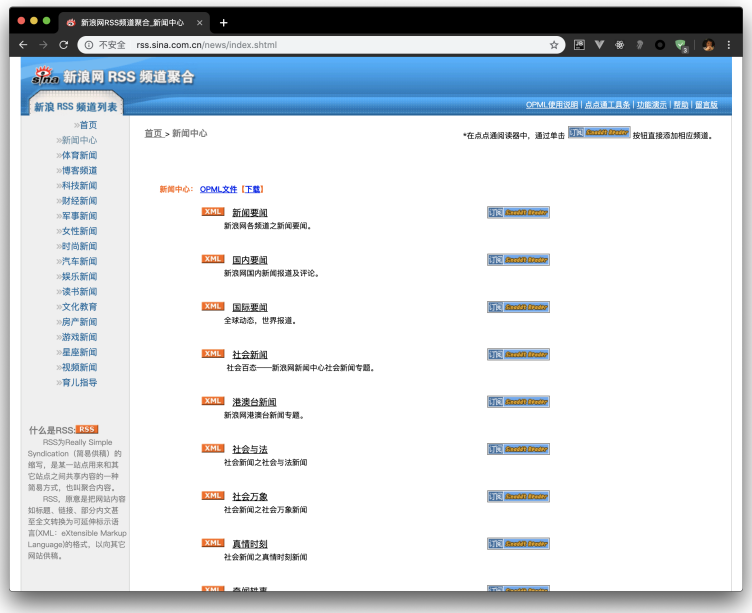
06 新闻供稿专用爬虫开发实践

更新时间：2019-07-04 10:26:37

> 困难只能吓倒懦夫懒汉，而胜利永远属于敢于等科学高峰的人。
>
> ——茅以升

本节课我们将学习如何用 `Item` 定义目标数据的提取结构，然后开始了解“蜘蛛”到底是长什么样子的，在 scrapy 中是怎么实现的，最后用 scrapy 内置的 `XMLFeedSpider` 快速实现新闻供稿爬虫。

本节将会以爬取新浪 RSS 频道聚合为示例，展开 Scrapy 爬虫的学习、开发之旅。先来看一下 RSS 频道聚合长什么样子：



开始之前我们先回顾一下在第一章所学到的设计爬虫的基本步骤：

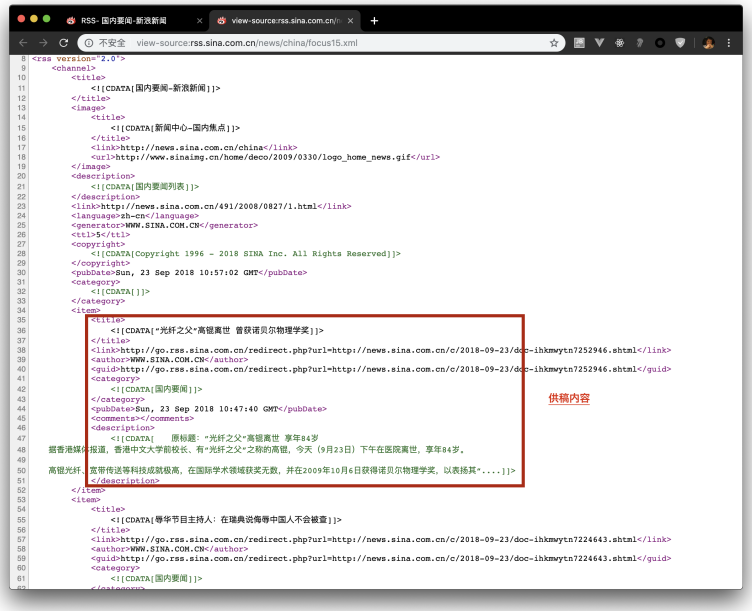
1. 确立爬取目标，分析种子页结构
2. 设计需要的存储的数据结构
3. 分析承载数据的页面结构，建立数据结构与元素选择器间的映射关系
4. 设计代码流程与编写思路

第一步，我们这个示例先定位于爬取单个频道的聚合内容，然后接下来就是需要分析存储的数据结构了。这个示例与之前我们接触过的 HTML 编写的页面源码有所不同，在爬取这个页面之前，我们需要先来了解一些关于 RSS（新闻供稿）格式的相关内容，这能有助于加深我们的理解与进一步的学习。

RSS 的基本知识

简易信息聚合（也叫聚合内容）是一种 RSS 基于 XML 标准，在互联网上被广泛采用的内容包装和投递协议。RSS(Really Simple Syndication) 是一种描述和同步网站内容的格式，是使用最广泛的 XML 应用。RSS 搭建了信息迅速传播的一个技术平台，使得每个人都成为潜在的信息提供者。发布一个 RSS 文件后，这个 RSS Feed 中包含的信息就能直接被其他站点调用，而且由于这些数据都是标准的XML 格式，所以也能在其他的终端和服务中使用，是一种描述和同步网站内容的格式。

为了更好的了解 RSS ，我们来打开新浪 RSS 频道聚合新闻中心分类的页面源码。来到新闻中心页面，右键查看网页源代码：



将上述文档格式用中文重新标识其中的内容作用，就可以了解整个文档的数据结构了：

```
<rss version="2.0">
    <channel>
        <title>频道标题名称</title>
        <image>
            <title>图片标题</title>
            <link>图片链接地址</link>
            <url>图片地址</url>
        </image>
        <description>新闻频道的详细描述
        </description>
        <link>本频道的链接地址</link>
        <language>语言</language>
        <item>
            <title>新闻标题</title>
            <link>新闻的原文链接</link>
            <description>新闻摘要信息
            </description>
            <pubDate>发布日期</pubDate>
        </item>
        <item>
         ...
        </item>
    </channel>
</rss>
```

是不是比较容易理解？

当然 RSS 的所有元素定义远远不止这么一点，为了表述方便，此处不对其全部的元素定义一一赘述了，有兴趣的读者可以到 W3CSchool 的RSS参考手册 上阅读更多内容。

为 RSS 设计通用的数据结构

对于 RSS 这种具有标准化格式的数据，可以说是爬虫项目中最简单的一种了，因为我们只需要做的就是下载一个 XML 文件，然后从 XML 文件中读取出对应标签内的内容就可以了。

按照我在第一章中介绍的方法先来建立的一个数据关系的对照表,先弄清楚哪些数据是我们需要的，这些数据又存在结果数据中的哪个位置。

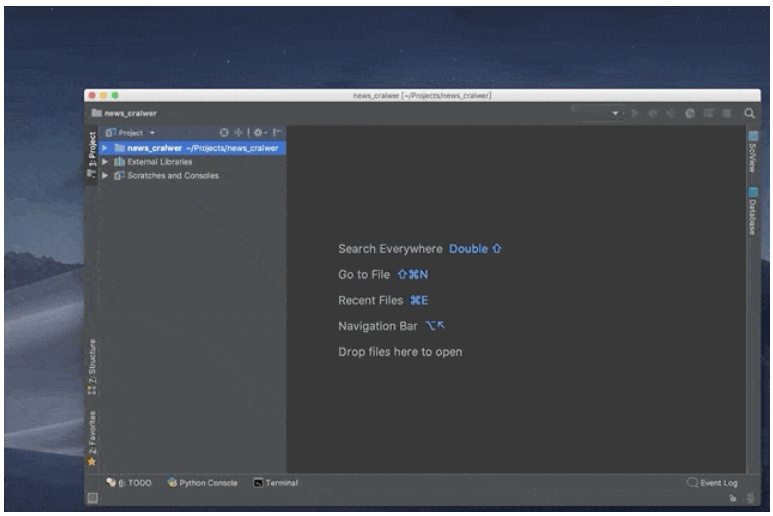
根据对路透社 RSS 的结构分析，所有的数据都存在 `<item>` 标记内，对此进行分析可以得到下表：

名称	字段	选择器(XPATH)
标题	title	item/title/text()
摘要	desc	item/description/text()
链接	link	item/link/text()
发表日期	pub_date	item/pubDate/text()

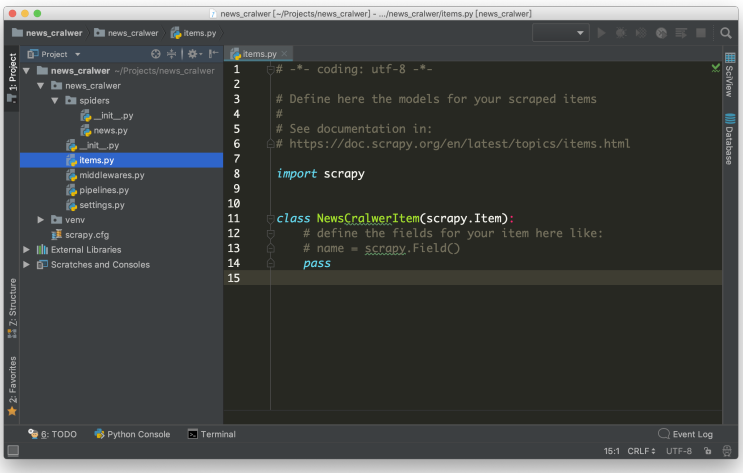
编写 `items.RSSFeedItem` - 浅释 Item

在 Scrapy 中提供一个类，用于帮助我们以面向对象的方式来定义用于存储分析后的结构化内容，这个类称为 `scrapy.item.Item`

首先，我们用 pyCharm 打开在上章中创建的 news_cralwer 项目,并在 pyCharm 中指定项目使用的 Python 解释器,具体操作如下:



然后在 pyCharm 中打开 `news_crawler/items.py` 文件:



这个文件内有一个 `NewsCralwerItem` 的类，是在创建 scrapy 项目时， `startproject` 指令为我们默认创建的，为了方便使用，先将类命名为 `NewsItem`。

Scrapy 的 `scrapy.item.Item` 类的属性需要用一个 `scrapy.item.Field` 对象来指明每个字段的元数据(metadata)。这样做是方便 Scrapy 对数据进行序列化操作。

根据字段映射表我们就可以动手定义这个 `Item` 类了，具体代码如下所示：

```
# -*- coding: utf-8 -*-
# news_crawler/news_crawler/items.py
from scrapy.item import Item, Field

class NewsItem(Item):
    title = Field()
    desc = Field()
    link = Field()
    pub_date = Field()
```

`scrapy.item.Field` 对象对接受的值没有任何限制。也正是因为这个原因，文档也无法提供所有可用的元数据的键(key)参考列表。 `scrapy.item.Field` 对象中保存的每个键可以由多个组件使用，并且只有这些组件知道这个键的存在。读者可以根据自己的需求，定义使用其他的 `Field` 键。设置 `Field` 对象的主要目的就是在一个地方定义好所有的元数据。一般来说，那些依赖某个字段的组件肯定使用了特定的键(key)。我们必须查看组件相关的文档，查看其用了哪些元数据键(metadata key)。

需要注意的是，用来声明 item 的 `scrapy.item.Field` 对象并没有被赋值为 class 的属性。不过可以通过 `Item.fields` 属性进行访问。

编写 蜘蛛 `RSSFeedSpider

接下来就是重头戏蜘蛛类的编写了，在 pyCharm 打开终端，使用我们在上一节中重点讲述的 `genspider` 指令生成一个基于 `xmlfeed` 模板的蜘蛛:

```
(venv) $ scrapy genspider -t xmlfeed news rss.sina.com.cn
```

然后打开 `news_crawler/news_cralwer/spiders/news.py` 文件:

了解 scrapy 蜘蛛的最基本写法

```
# -*- coding: utf-8 -*-
from scrapy.spiders import XMLFeedSpider

class NewsSpider(XMLFeedSpider):
    name = 'news'
    allowed_domains = ['rss.sina.com.cn']
    start_urls = ['http//:rss.sina.com.cn/feed.xml']
    iterator = 'iternodes' # you can change this; see the docs
    itertag = 'item' # change it accordingly

    def parse_node(self, response, selector):
        i = {}
        #i['url'] = selector.select('url').extract()
        #i['name'] = selector.select('name').extract()
        #i['description'] = selector.select('description').extract()
        return i
```

我们来细致地解读上述由 `genspider -t xmlfeed` 所生成的这个 `NewsSpider` 类的代码，了解清楚具体的逻辑以后，就知道应该从哪里着手写了。

首先，`scrapy.spiders.XMLFeedSpider` 是 Scrapy 的一个内置提供的爬虫类，它被设计用于通过迭代各个节点来分析XML源（XML feed ）。

认识 Spider 类的最基本构成

`NewsSpider` 类中定义的这几个属性分别具有以下这些作用：

```
name = 'news'                                          # 蜘蛛的名称用于`crawl`指令调用
allowed_domains = ['rss.sina.com.cn']                  # 指明允许蜘蛛能爬取的主域（可以是多个）
# start_urls = ['http://https://cn.reuters.com/feed.xml']  # 由genspider生成，但地址不并正确采用下一行这个
start_urls = ['http//:rss.sina.com.cn/feed.xml'] # 指定爬虫的种子页，可以是一个也可以是多个
iterator = 'iternodes'                                 # 迭代器，可以从 "iternodes", "xml" , "html"这三种值中选取
itertag = 'item'                                       # 指定一个标签名，迭代器会根据这个标签名去定位数据项，这里设置为ite
m意思就是定位所有<item>标记
```

> 注:不用太纠结于 `iterator` 的取值，无论你用 `iternodes`、`xml` 还是 `html` 最终还是会被转换成为以 xpath 方式对文档进行选择然后迭代，所以它们的速度几乎是没有差别的，甚至你可以将这>个属性的赋值删除了 `XMLFeedSpider` 是依然能正常工作。

`parse_node` 函数的作用

解释完这些属性的作用，是不是还感觉有点云里雾里？如果用 Scrapy 的方式来理解蜘蛛就很容易记住上面这些属性的真正作用了，Scrapy 是这样来执行蜘蛛的：

1. 用 `scrapy crawl news` 来启动蜘蛛
2. Scrapy 会根据 `start_urls` 内定义的种子页自动生成并发出网页请求，这些请求只能被限定在从属于 `allowed_domains` 内指定的域内的子页
3. 当请求返回之后，`XMLFeedSpider` 就会预先将返回结果载入一个 XML 文档，并按照 `iterator` 指定的迭代器来分析结果
4. 根据 `itertag` 中指定的标签名从 XML 文档中取出一个或多个的 XMLNode 对象
5. 最后由 `XMLFeedSpider` 根据提取出的 `XMLNode` 的个数**循环调用** `parse_node` 函数并将 XML 节点选择器实例传入

也就是说 `XMLFeedSpider` 预先为我们进行RSS文档的分析和提取工作，我们只需要从 `parse_node` 函数的 `selector` 参数中取得其包含的子标签的内容并生成 `NewsItem` 返回即可。

内容的提取，通过写代码认识 Selector 与 Response 对象

`parse_node( self, response, selector )` 中的 `response` 就是与发出请求对应的原生响应对象，`selector` 就是节点选择器。节点选择器又是什么鬼？这个节点选择器就与我们在第一章中学到的 css 选择器相类似，只是这个选择器是一个 XPath 选择器类型，采用的是 XPath 的选择语法。

接下来就是实现这个 `parse_node` 函数了，先从 `items` 中导入 `NewsItem` ，实例化 `NewsItem` 类，然后通过选择器将 `<item>` 内对应的字段读取出来并保存到 `NewsItem` 实例中，最后返回 `NewsItem` 实例，具体代码如下所示：

```
from ..items import NewsItem

def parse_node(self, response, selector):
    item = NewsItem()
    item['title'] = selector.xpath('title/text()').extract_first()
    item['link'] = selector.xpath('link/text()').extract_first()
    item['pub_date'] = selector.xpath('pubDate/text()').extract_first()
    item['desc'] = selector.xpath('description/text()').extract_first()
    return item
```

`selector.xpath` 可以匹配 xpath query 的节点，并返回 `SelectorList` 的一个实例结果，单一化其所有元素。列表元素也实现了 `Selector` 的接口。而 `extract_first()` 方法则是从 `SelectorList` 中提取出第一个元素，如果只调用 `extract()` 方法的话就会返回一个 `list` 对象。

使用 `scrapy crawl news` 执行蜘蛛

整个代码已经写完了，是不是感觉学的概念很多但实际写的代码很少呢？这就是 Scrapy 的强大之处，只要你完全掌握了它的相关概念实质，编码量是非常少的。

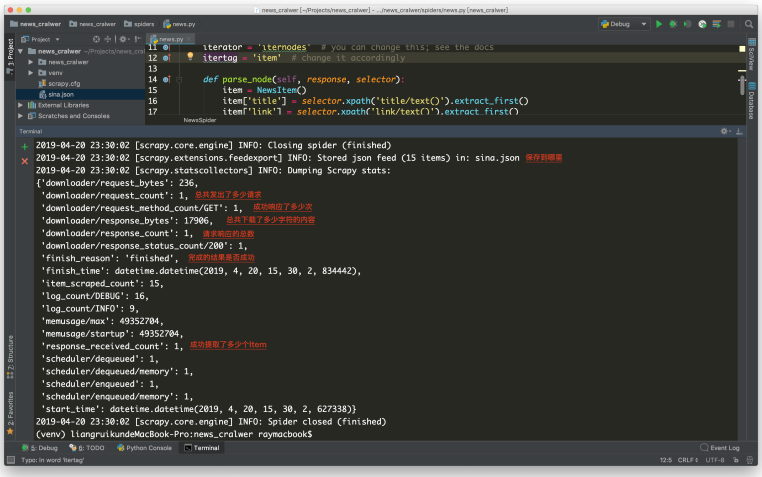
打开终端运行以下指令启动爬虫

```
(venv) $ scrapy crawl news -o sina.json
```

这里的 `-o sina.json` 是将爬取结果保存到JSON格式的文件中，当成功执行 `crawl` 后会看到一大堆的指令输出结果

在控制台输出的众多数据哪些是有用的？

如果你仔细看这些结果就会发现很多有用的信息，如下图所示：



如果你并不想看到这些令人眼花缭乱的输出记录可以加上一个 `--nolog` 参数，这样 `crawl` 指令就会停止日志的输出，这样做遇到海量数据爬取时是非常有用的。

如何让这个 news 爬虫变得通用起来？

既然 RSS 是一种通用格式，也就是说所有的提供 RSS 格式聚合内容的网站 news 爬虫都可以爬取吗？答案当然是肯定的。但现在 `start_urls` 这个参数被写死在蜘蛛代码内，是否能将它变成一个变量接收命令行参数的输入呢？

官方文档提出过可以通过 `-a` 参数将命令行输入的参数直接绑定到蜘蛛的一个属性值上。根据此原理我们来改写一下这个新闻蜘蛛让它变更得更通用一些：

```
class NewsSpider(XMLFeedSpider):
    name = 'news'
    url = 'http://rss.sina.com.cn/news/world/focus15.xml'
    start_urls = [url]
```

这里改写了两处，一是我将 `allowed_domains`、`iterator`、`itertag` 三个属性都删除了，因为对于本示例来说这三个属性直接用默认值运行就可以了，没有必要声明到类里面。然后增加了一个 `url` 属性用于承载输入参数。

通过这个简单的重构，就可以用以下的指令去爬取中新网新闻供稿(http://www.chinanews.com/rss/scroll-news.xml),如下所示：

```
$ scrapy crawl news -a url='http://www.chinanews.com/rss/scroll-news.xml'  -o chinanews.json
```

如何完美解决JSON输出的中文编码问题

当我们打开 `sina.json` 文件可能你又会觉得有点崩溃，因为又遇到中文乱码了，原因我在第一章第3小节也解释过。但在 Scrapy 我们又应如何处理呢？是不是要增加一个处理类来解决这个编码问题？当然你上百度或谷歌一下会出现一大堆建议，例如你去增加一个管道(`pipline`)类来解决这个编码问题，但我认为不写代码才是**完美解决**这个问题的办法。而且我们还没有讲述到 `pipline` 这一个重要的内容，所以我们同样可以用 `crawl` 的输入参数来解决这一问题。

方法非常简单，只要设置 scrapy 的默认运行配置项 `FEED_EXPORT_ENCODING='utf-8'` 就可以了，具体做法如下：

```
$ scrapy crawl news -a url='http://www.chinanews.com/rss/scroll-news.xml' -o chinanews.json  -s FEED_EXPORT_ENCODING='utf-8
'
```

`-s` 参数就是修改运行配置 `settings.py` 内的预置配置项的意思。再运行一次，问题就完美解决！这样你就得到了一个可以爬取任何内容聚合网站的强力爬虫了，是不是很简单呢？

小结

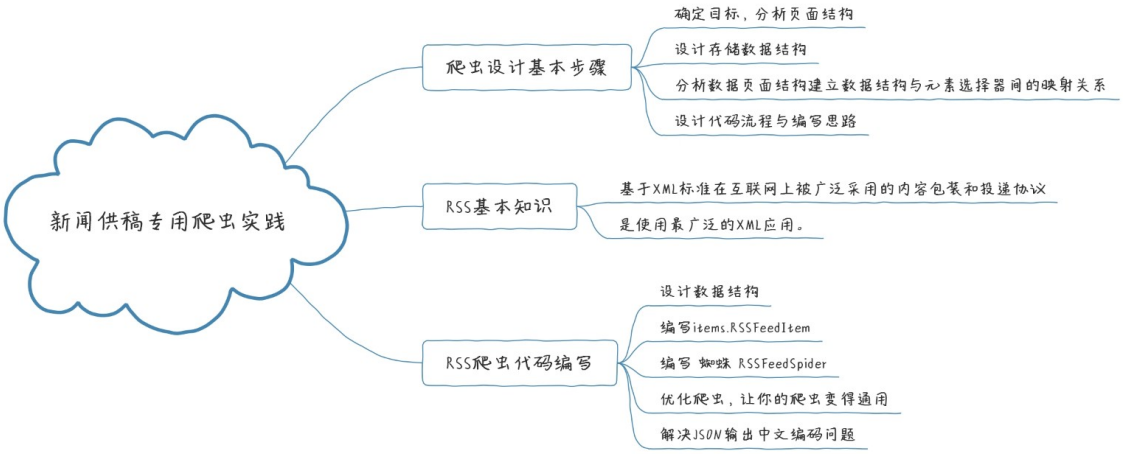
回想这个简单的通用爬虫实践，我们仍然是采用第一章中的开发思路，其实那就是爬虫的本质！无论爬虫用什么语言实现，用什么框架实现还是用这4个套路：

1. 确立爬取目标，分析种子页结构
2. 设计需要的存储的数据结构
3. 分析承载数据的页面结构，建立数据结构与元素选择器间的映射关系
4. 设计代码流程与编写思路

本节的示例只是一种对于爬取结构化数据的特例，毕竟互联网中非结构化数据的存量是远远大于结构化数据的，没有几个如此大方的网站会将数据按照统一标准做好给你去爬的，所以谨记以上4步，是设计一个良好爬虫项目的根本。

本节的重点则在于以下几点：

1. RSS 的数据结构
2. 学习用 `Item` 与 `Field` 来构建数据实体类
3. 采用 Scrapy 自带的 `XMLFeedSpider` 快速解决一切 RSS 网页的问题
4. 了解 Scrapy 的最基本的运作过程
5. 用好 `crawl` 指令可以省去你不少额外的代码



> 思考：本示例只是直接用了 RSS 的页来爬取，如果将种子页换成新浪 RSS 聚合频道页面那这个项目应该如何改写呢？

> 注: 你可以到 GitHub 获取本节源代码

05 Python 世界最流行的网络爬虫框架 Scrapy
07 常见爬网方式与网页结构分析思路