Парсинг сайтов с помощью фреймворка Scrapy

Обсудить в форуме Комментариев — 0

Эта страница опубликована в основном списке статей сайта
по адресу http://gis-lab.info/qa/scrapy.html

Содержание

Введение

Синтаксический анализ (парсинг) сайтов хоть и не имеет прямого отношения к пространственным данным, но владение основами которого полезно любому, работающему с ними. В свете роста числа онлайн-ресурсов, публикующих открытые данные, ценность умения извлекать из них необходимую информацию многократно повышается. Приведём небольшой пример. Допустим, нам необходимо составить набор тематических карт, отражающих результаты Выборов Президента Российской Федерации 2012. Исходные данные можно получить на сайте ЦИК России. Однако, согласитесь, что непосредственно использовать данные, предоставляемые в таком виде, очень сложно. Плюс это усложняется тем, что данные по разным регионам расположены на разных страницах. Гораздо удобнее было бы, чтобы вся эта информация была представлена, например, в виде одного или нескольких структурированных CSV или XML файлов (в идеале, конечно было бы иметь еще и некоторый API, позволяющий выполнять запросы к таким ресурсам), однако зачастую формирование подобных файлов отдаётся на откуп конечному пользователю (почему так происходит - это вопрос отдельный). Именно проблеме создания таких вот аггрегированных наборов данных и посвещена данная статья. В связи с недавними событиями в качестве целевого сайта, который мы будем парсить, выбран сайт ДетскиеДомики.ру, а именно его раздел Детские учреждения. Предполагается, что информация, расположенная на этом сайте будет в ближайшее время очень востребованной.

В качестве инструмента, которым будет выполняться парсинг, выбран Scrapy - очень гибкий фреймворк, написанный на Python и позволяющий решать широкий спектр задач. Информации о Scrapy на русском языке не так много, например и ещё, но этот пробел компенсируется отличной документацией.

В данной статье мы не будем подробно заострять внимание на всех технических возможностях Scrapy, а просто рассмотрим поэтапно, как была решена определённая задача. Если кто-то захочет использовать данный материал как отправную точку для решения собственной задачи, но не найдёт здесь ответов на свой вопрос - спрашивайте в форуме, постараемся помочь.

Установка Scrapy

Unix

В *nix-системах установка Scrapy - тривиальная задача:

```
cd ~
mkdir scrapy
```

```
cd scrapy
virtualenv --no-site-packages env
source ./env/bin/activate
pip install Scrapy
```

Windows

В Windows всё сложнее:

1. Перейдите на страницу Win32 OpenSSL
2. Скачайте и установите Visual C++ 2008 redistributables для вашей версии Windows и архитектуры
3. Скачайте OpenSSL для вашей версии Windows и архитектуры (вам нужна обычная версия - не light)
4. Добавьте путь c:\openssl-win32\bin (или тот куда вы установили OpenSSL) к переменной PATH
5. Установите twisted отдельно (например для 2.7)
6. Установите lxml отдельно (например для 2.7)
7. если easy_install не установлен, установите сначала его самого, он ставится в C:\Python27\Scripts\ и запускается из cmd, а не из Python

далее установите:

```
easy_install pip
pip install Scrapy
pip install zope.interface
pip install w3lib
```

Создание проекта

После того, как Scrapy будет установлен, нужно создать каталог проекта. Для этого, находясь в каталоге *~/projects/scrapy*, выполните команду:

```
scrapy startproject orphanage
```

В результате чего будет создана директория *orphanage* (соответствует имени проекта), имеющая следующую структуру:

```
orphanage/
    scrapy.cfg
    orphanage/
        __init__.py
        items.py
        pipelines.py
        settings.py
        spiders/
            __init__.py
            ...
```

- *scrapy.cfg* - настройки проекта;
- *orphanage/* - Python модуль проекта;
- *orphanage/items.py* - классы, описывающие модель собираемых данных;
- *orphanage/pipelines.py* - используется в основном для описания пользовательских форматов сохранения результатов парсинга;
- *orphanage/settings.py* - пользовательские настройки паука;
- *orphanage/spiders/* - директория, в которой хранятся файлы с классами пауков. Каждого паука принято писать в отдельном файле.

Описание модели данных

Модель представляет собой отдельный класс, содержащий перечень атрибутивных полей собираемых данных. Прежде чем описывать модель, необходимо определиться во-первых с объектом парсинга, а во-вторых с набором его атрибутов, которые мы хотим извлечь из целевого ресурса. Объектом парсинга в нашем случае будут являться детские дома, а набором атрибутов - их характеристики (в случае если у каждого объекта перечень доступных атрибутов различный, то итоговым набором будет являться объединение множеств атрибутов всех объектов). Находим страницу, содержащую наиболее полный перечень атрибутов (в

данном случае факт полноты был определён путём сопоставления представленных атрибутов и анкеты детского учреждения) и выписываем их: "Рег. номер", "Регион", "Район", "Тип учреждения", "Название", "Почтовый адрес" и т.д. (всего 34 атрибута). После того, как мы определились с перечнем атрибутов, отразим их в специальном классе. Для этого открываем файл *items.py* и описываем класс (название - произвольное):

```
from scrapy.item import Item, Field

class OrphanageItem(Item):
    # define the fields for your item here like:
    # name = Field()
    id                = Field()
    region            = Field()
    district          = Field()
    type              = Field()
    name              = Field()
    post              = Field()
    phone             = Field()
    director          = Field()
    bank              = Field()
    parent            = Field()
    foundation        = Field()
    activities        = Field()
    history           = Field()
    staff             = Field()
    publications      = Field()
    children          = Field()
    age               = Field()
    orphans           = Field()
    deviated          = Field()
    principle         = Field()
    education         = Field()
    treatment         = Field()
    holidays          = Field()
    communication     = Field()
    buildings         = Field()
    vehicles          = Field()
    farming           = Field()
    working_cabinet   = Field()
    library           = Field()
    computers         = Field()
    toys              = Field()
    patronage         = Field()
    needs             = Field()
    volunteers        = Field()
    url               = Field()
```

Как можно увидеть представленный класс содержит записи вида *имя атрибута = Field()*, где в качестве *имя атрибута* рекомендуется использовать английский вариант названия соответствующего атрибута. Кроме того, в класс был добавлен ещё один атрибут *url*, который предназначен для хранения URL той страницы, из которой были извлечены данные.

Создание паука

Паук - это основная часть нашей системы, представляющий собой отдельный класс, описывающий способ обхода ресурса и собирающий необходимую информацию в соответствии с описанной на предыдущем этапе моделью.

Переходим в директорию *orphanage/spiders/* и создаём файл с описанием паука *detskiedomiki.py* (название произвольное). Внутри файла описываем класс (имя класса - произвольное):

```
from scrapy.contrib.spiders import CrawlSpider, Rule
from scrapy.contrib.linkextractors.sgml import SgmlLinkExtractor
from scrapy.contrib.loader.processor import TakeFirst
```

```
from scrapy.contrib.loader import XPathItemLoader
from scrapy.selector import HtmlXPathSelector
from orphanage.items import OrphanageItem


class OrphanSpider(CrawlSpider):
    name = "detskiedomiki"
    allowed_domains = ["www.detskiedomiki.ru"]
    start_urls = ["http://www.detskiedomiki.ru/guide/child/"]

    rules = (
        Rule(SgmlLinkExtractor(allow=('act=home_reg', 'act=home_zone')), follow=True),
        Rule(SgmlLinkExtractor(allow=('act=home_more')), callback='parse_item'),
    )

    def parse_item(self, response):
        ...
```

- *name* - уникальный идентификатор паука, именно он будет использоваться в качестве аргумента при запуске процесса парсинга;
- *allowed_domains* - список доменов по которым может ходить паук;
- *start_urls* - список URL, с которых начинается обход ресурса;
- *rules* - список правил обхода ресурса. В данном случае данный список содержит 2 правила - первое будет срабатывать при попадании паука на страницы, URL которых содержит *act=home_reg* или *act=home_zone*. Под срабатыванием в данном случае подразумевается переход по ссылкам, извлеченным из этих страниц (за что отвечает аргумент *follow=True*). Второе правило будет срабатывать при попадании паука на страницы, URL которых содержит *act=home_more* (именно на этих страницах содержится информация, которую мы хотим извлечь), например http://detskiedomiki.ru/?act=home_more&id=6278&z_id=3&part_id=65 . В данном случае у правила не указан аргумент *follow*, что означает, что при попадании паука на данную страницу ссылки из неё не извлекаются, вместо этого содержимое страницы передаётся на вход функции, указанной в аргументе *callback* - назовём её в нашем случае *parse_item*.

Прежде чем переходить к рассмотрению функции *parse_item*, сделаем небольшое техническое отступление. Существет несколько библиотек, предназначенных для извлечения данных из HTML-документов. Наиболее распространённые это BeautifulSoup (популярная, но при этом обладающая одним недостатком - очень медленная) и lxml. В Scrapy используется собственный механизм извлечения данных (основанный так же как и lxml на libxml2) из HTML-документов - селекторы (*selectors*). Фактически, селекторы - это отдельные классы, при создании экземпляров которых на вход передаётся объект класса Response (представляющий собой ответ сервера). В Scrapy доступно 2 селектора - HtmlXPathSelector и XmlXPathSelector, предназначенных для парсинга HTML и XML документов соответственно.

Для того, чтобы понять, как работают селекторы, перейдите в директорию *~/projects/scrapy/orphanage* и выполните команду:

```
scrapy shell "http://detskiedomiki.ru/?act=home_more&id=6278&z_id=3&part_id=65"
```

В результате выполнения данной команды будет осуществлен запрос к указанной странице, после чего вы попадете в интерактивную консоль, в которой уже будет ряд созданных Scrapy объектов, в том числе и объект *hxs* класса *HtmlXPathSelector*. Данную консоль очень удобно использовать для составления XPath выражений - основного инструмента доступа к элементам HTML-документа, используемого в Scrapy. Например, мы хотим извлечь значение атрибута *Рег. номер* из нашей страницы, для этого в консоли вызываем метод select объекта *hxs* и в качестве аргумента используем соответствующую XPath-строку:

```
In [10]: hxs.select("//td[text()='%s']/following-sibling::td/text()" % "Рег.
номер:".decode('utf-8'))
Out[10]: [<HtmlXPathSelector xpath=u"//td[text()='\u0420\u0435\u0433.
\u043d\u043e\u043c\u0435\u0440:']/following-sibling::td/text()" data=u'01-04-15-01'>]
```

В результате получим массив, состоящий из списка объектов класса *HtmlXPathSelector*, то есть метод *select* объекта класса *HtmlXPathSelector* возвращает список объектов класса *HtmlXPathSelector* (что очень удобно в случае парсинга вложенных данных). Для извлечения непосредственно данных необходимо применить метод

extract:

```
In [11]: hxs.select("//td[text()='%s']/following-sibling::td/text()" % "Рег.
номер:".decode('utf-8')).extract()
Out[11]: [u'01-04-15-01']
```

В результате мы получим список значений тех элементов, которые удовлетворяют XPath-выражению.

А теперь рассмотрим нашу функцию *parse_item*:

```
class OrphanLoader(XPathItemLoader):
    default_output_processor = TakeFirst()

def parse_item(self, response):
    hxs = HtmlXPathSelector(response)
    l = OrphanLoader(OrphanageItem(), hxs)

    #
    l.add_xpath('id', "//td[text()='%s']/following-sibling::td/text()" % u"Рег. номер:")
    ...

    l.add_value('url', response.url)

    return l.load_item()
```

Загрузчики (Item Loaders) - специальные классы, облегчающие заполнение объекта класса модели данных (в нашем случае *OrphanageItem*). В данном случае мы расширяем базовый класс загрузчика XPathItemLoader путём создания нового класса *OrphanLoader* и устанавливаем свойство default_output_processor в значение TakeFirst, это сделано из следующих соображений. При извлечении данных из HTML-документа результат возвращается в виде массива значений (даже если этот массив состоит из одного элемента как в нашем случае), использование процессора *TakeFirst* позволяет из полученного массива значений извлекать первый элемент.

Следующей строкой мы создаём объект класса *OrphanLoader*:

```
l = OrphanLoader(OrphanageItem(), hxs)
```

Первый аргумент - объект класса модели данных *OrphanageItem*, второй класса *HtmlXPathSelector*. Следующая (и последующие) строка вида:

```
l.add_xpath('id', "//td[text()='%s']/following-sibling::td/text()" % u"Рег. номер:")
```

говорит о том, что атрибут *id* объекта класса нашей модели данных будет извлекаться в соответствии с переданным выражением XPath.

```
l.add_value('url', response.url)
```

Метод *add_value* позволяет задать значение указанного (*url*) атрибута вручную. Следующая строка:

```
return l.load_item()
```

заполняет атрибуты объекта *OrphanageItem* в соответствии с настройками загрузчика.

Итоговый вариант файла *orphanage/spiders/detskiedomiki.py* :

```
# -*- encoding: utf-8 -*-

from scrapy.contrib.spiders import CrawlSpider, Rule
from scrapy.contrib.linkextractors.sgml import SgmlLinkExtractor
from scrapy.contrib.loader.processor import TakeFirst
from scrapy.contrib.loader import XPathItemLoader
from scrapy.selector import HtmlXPathSelector
from orphanage.items import OrphanageItem
```

```
class OrphanLoader(XPathItemLoader):
    default_output_processor = TakeFirst()

class OrphanSpider(CrawlSpider):
    name = "detskiedomiki"
    allowed_domains = ["www.detskiedomiki.ru"]
    start_urls = ["http://www.detskiedomiki.ru/guide/child/"]

    rules = (
        Rule(SgmlLinkExtractor(allow=('act=home_reg', 'act=home_zone')), follow=True),
        Rule(SgmlLinkExtractor(allow=('act=home_more')), callback='parse_item'),
    )

    def parse_item(self, response):
        hxs = HtmlXPathSelector(response)
        l = OrphanLoader(OrphanageItem(), hxs)

        #
        l.add_xpath('id', "//td[text()='%s']/following-sibling::td/text()" % u"Рег.
номер:")
        l.add_xpath('region', "//td[text()='%s']/following-sibling::td/text()" %
u"Регион:")
        l.add_xpath('district', "//td[text()='%s']/following-sibling::td/text()" %
u"Район:")
        l.add_xpath('type', "//td[text()='%s']/following-sibling::td/text()" % u"Тип
учреждения:")
        l.add_xpath('name', "//td[text()='%s']/following-sibling::td/strong/text()" %
u"Название:")
        l.add_xpath('post', "//td[text()='%s']/following-sibling::td/text()" % u"Почтовый
адрес:")
        l.add_xpath('phone', "//td[text()='%s']/following-sibling::td/text()" %
u"Телефоны:")
        l.add_xpath('director', "//td[text()='%s']/following-sibling::td/text()" %
u"Руководство:")
        l.add_xpath('bank', "//td[text()='%s']/following-sibling::td/text()" %
u"Банковские реквизиты:")
        l.add_xpath('parent', "//td[text()='%s']/following-sibling::td/text()" %
u"Вышестоящая организация:")

        #
        l.add_xpath('foundation', "//td[text()='%s']/following-sibling::td/text()" %
u"Дата основания:")
        l.add_xpath('activities', "//td[text()='%s']/following-sibling::td/text()" %
u"Направления деятельности:")
        l.add_xpath('history', "//td[text()='%s']/following-sibling::td/text()" %
u"История:")
        l.add_xpath('staff', "//td[text()='%s']/following-sibling::td/text()" %
u"Персонал:")
        l.add_xpath('publications', "//td[text()='%s']/following-sibling::td/text()" %
u"Публикации в СМИ:")

        #
        l.add_xpath('children', "//td[text()='%s']/following-sibling::td/text()" %
u"Количество детей в учреждении:")
        l.add_xpath('age', "//td[text()='%s']/following-sibling::td/text()" % u"Возраст
детей:")
        l.add_xpath('orphans', "//td[text()='%s']/following-sibling::td/text()" %
u"Количество детей-сирот:")
        l.add_xpath('deviated', "//td[text()='%s']/following-sibling::td/text()" %
u"Количество детей с отклонениями в развитии:")
        l.add_xpath('principle', "//td[text()='%s']/following-sibling::td/text()" %
u"Принцип формирования группы:")
        l.add_xpath('education', "//td[text()='%s']/following-sibling::td/text()" %
u"Обучение детей:")
```

```
        l.add_xpath('treatment', "//td[text()='%s']/following-sibling::td/text()" %
u"Лечение детей:")
        l.add_xpath('holidays', "//td[text()='%s']/following-sibling::td/text()" %
u"Летний отдых:")
        l.add_xpath('communication', "//td[text()='%s']/following-sibling::td/text()" %
u"Общение детей:")

        #
        l.add_xpath('buildings', "//td[text()='%s']/following-sibling::td/text()" %
u"Здания:")
        l.add_xpath('vehicles', "//td[text()='%s']/following-sibling::td/text()" %
u"Автотранспорт:")
        l.add_xpath('farming', "//td[text()='%s']/following-sibling::td/text()" %
u"Подсобное хозяйство:")
        l.add_xpath('working_cabinet', "//td[text()='%s']/following-sibling::td/text()" %
u"Кабинеты труда:")
        l.add_xpath('library', "//td[text()='%s']/following-sibling::td/text()" %
u"Библиотека:")
        l.add_xpath('computers', "//td[text()='%s']/following-sibling::td/text()" %
u"Компьютеры:")
        l.add_xpath('toys', "//td[text()='%s']/following-sibling::td/text()" % u"Игрушки
и игры")

        #
        l.add_xpath('patronage', "//td[text()='%s']/following-sibling::td/text()" %
u"Шефство, помощь:")
        l.add_xpath('needs', "//td[text()='%s']/following-sibling::td/text()" %
u"Потребности учреждения:")
        l.add_xpath('volunteers', "//td[text()='%s']/following-sibling::td/text()" %
u"Привлечение добровольцев:")

        l.add_value('url', response.url)

        return l.load_item()
```

Формирование выходных данных

Для запуска нашей программы необходимо, находясь в каталоге *~/projects/scrapy/orphanage* выполнить команду вида:

```
scrapy crawl [имя паука]
```

В нашем случае:

```
scrapy crawl detskiedomiki
```

При таком запуске результаты парсинга будут обрабатываться методами, описанными в специальных классах, расположенных в файле *pipelines.py*. То есть мы можем написать свой класс, который будет принимать на вход объекты класса *OrphanageItem* и обрабатывать их нужным образом. Однако в большинстве случаев оказывается достаточным того функционала, который предоставляет Scrapy по выгрузке данных в JSON, XML или CSV. Так, чтобы результаты парсинга в нашем случае сохранились в файле формата CSV, необходимо выполнить команду:

```
scrapy crawl detskiedomiki -o scarped_data_utf8.csv -t csv
```

Результат

Результаты парсинга доступны по адресу: http://gis-lab.info/share/DR/scarped_data_utf8.csv.7z

Ссылки

Собираем данные с помощью Scrapy
2. Scrapy text encoding

Обсудить в форуме Комментариев — 0

Последнее обновление: 2014-05-15 01:45

Дата создания: 02.01.2013
Автор(ы): Денис Рыков