1. 课程介绍

* Scrapy框架简介（了解）
* Scrapy安装配置（掌握）
* Scrapy项目的创建（掌握）
* Scrapy框架的使用（掌握）

1. Scrapy框架介绍
   1. 简介

Scrapy是一个针对爬取网站内容、提取结构性数据，而用纯Python实现的开源应用框架，用途非常广泛。借助该框架的力量，用户只需要定制开发几个模块就可以较轻松的实现一个爬虫，用来抓取网页内容以及各种图片，都非常方便。

Scrapy 使用了 Twisted(和Tornado竞争关系)异步网络框架来处理网络通讯，可以加快我们的下载速度，不用自己去实现异步框架，并且包含了各种中间件接口，可以灵活的完成各种需求。

* 1. 架构



其中，绿线表示数据流向。

* Scrapy Engine(引擎): 处理整个系统的数据流处理、触发事务，包括Spider、ItemPipeline、Downloader、Scheduler中间的通讯，信号、数据传递等，是整个框架的核心。
* Scheduler(调度器): 接受引擎发送过来的Request请求，并按照一定的方式进行整理将其加入队列中，当引擎需要时，交还给引擎。
* Downloader（下载器）：负责下载Scrapy Engine(引擎)发送的所有Requests请求，并将其获取到的Responses交还给Scrapy Engine(引擎)，由引擎交给Spider来处理，
* Spider（爬虫）：它负责处理所有Responses,从中分析提取数据，获取Item字段需要的数据，并将需要跟进的URL提交给引擎，再次进入Scheduler(调度器)，
* Item Pipeline(管道)：它负责处理Spider中获取到的Item，并进行进行后期处理（详细分析、过滤、存储等）的地方.
* Downloader Middlewares（下载中间件）：你可以当作是一个可以自定义扩展下载功能的组件。
* Spider Middlewares（Spider中间件）：你可以理解为是一个可以自定扩展和操作引擎和Spider中间通信的功能组件（比如进入Spider的Responses;和从Spider出去的Requests）
  1. 编写Scrapy爬虫的步骤
     + 1. 新建项目 (scrapy startproject xxx)：新建一个新的爬虫项目
       2. 设置目标 （items.py）：明确你想要抓取的目标
       3. 制作爬虫 （spiders/xxspider.py）：制作爬虫开始爬取网页
       4. 存储内容 （pipelines.py）：设计管道存储爬取内容

1. Scrapy的安装和配置
   1. Windows
      1. pip安装

升级pip版本：pip install --upgrade pip

通过pip 安装 Scrapy 框架pip install Scrapy

* + 1. conda-forge安装

建议在某一conda环境下安装。

conda install -c conda-forge scrapy

注意：要确保conda环境被激活。

activate conda *环境名字*

* 1. 基于Ubuntu的操作系统
     1. apt-get + pip安装

需要安装如下依赖：

sudo apt-get install python-dev python-pip libxml2-dev libxslt1-dev zlib1g-dev libffi-dev libssl-dev python3-dev

然后执行sudo pip install scrapy，最好在某一虚拟环境(virtualenv)下安装。

* + 1. conda-forge安装

建议通过conda-forge安装到某一conda环境下。

conda install -c conda-forge scrapy

注意：如果通过conda安装，一定要确保该conda环境被激活。

source activate *conda环境名字*

* 1. MacOS
     1. conda-forge安装

建议通过conda-forge安装到某一conda环境下。

conda install -c conda-forge scrapy

注意：如果通过conda安装，一定要确保该conda环境被激活。

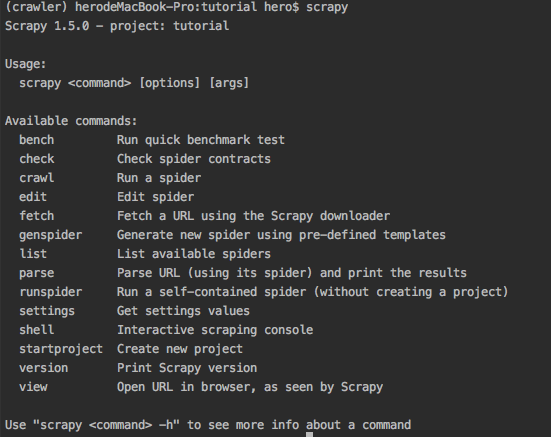
source activate *conda环境名字*

* 1. Manjaro等其他Linux和类Linux系统

你都这么厉害了，就自己搞定吧！

* 1. 安装验证

安装后，只要在命令终端输入 scrapy，提示类似以下结果，代表已经安装成功



注意：如果通过conda安装，一定要确保scrapy安装到的conda环境被激活。

1. Scrapy框架
   1. 入门案例

目标：使用Scrapy爬取网站<http://quotes.toscrape.com/>的指定内容。这是一个专用于练习爬虫编程的网站，网站页面代码简洁，适合初步练习用。

* + 1. 新建项目

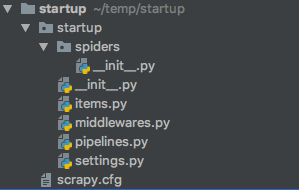
命令行进入建立好的项目目录中，运行如下命令：

scrapy startproject *yourspider*

其中，*yourspider*是你的爬虫项目名称。

注意：如果通过conda安装，一定要确保该conda环境被激活。

比如，运行scrapy startproject startup后，产生如下项目文件的目标结构

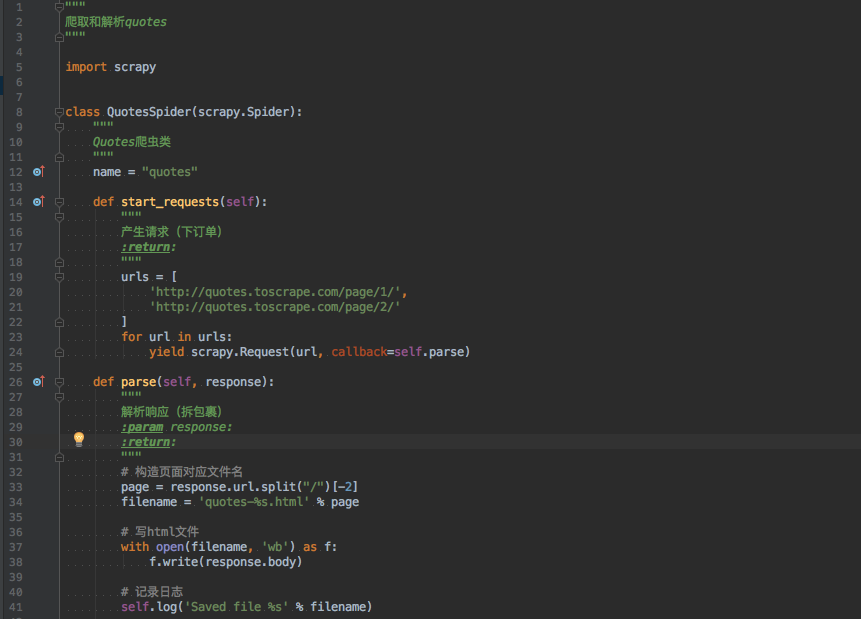


* + 1. 设置目标

我们打算抓取<http://quotes.toscrape.com/>里的所有名言（quote），获取每条名言的内容、作者和标签列表；抓取每位作者的名字、生日和简历。

* + 1. 制作爬虫

在*yourspider*/spiders目录下创建quotes\_spider.py，编写如下代码：



通过在项目目录下，运行如下命令运行编写的爬虫：

scrapy crawl *yourspider*

会生成两个html文件和一堆日志信息。

* + 1. 保存数据

**scrapy保存信息的最简单的方法主要有四种，-o 输出指定格式的文件，，命令如下：**

* + - 1. json格式，默认为Unicode编码

scrapy crawl *spidername* -o *filename*.json

* + - 1. json lines格式，默认为Unicode编码

scrapy crawl *spidername* -o *filename.jsonl*

或者

scrapy crawl *spidername* -o *filename.jl*

* + - 1. csv 逗号表达式，可用Excel打开

scrapy crawl *spidername* -o *filename.csv*

* + - 1. xml格式

scrapy crawl *spidername* -o *filename.xml*

* 1. Scrapy Shell

Scrapy终端是一个交互终端，我们可以在未启动spider的情况下尝试及调试代码，也可以用来测试XPath或CSS表达式，查看他们的工作方式，方便我们爬取的网页中提取的数据。

* + 1. 启动Scrapy Shell

进入项目的根目录，执行下列命令来启动shell:

scrapy shell 'http://quotes.toscrape.com/page/1/'

Scrapy Shell根据下载的页面会自动创建一些方便使用的对象，例如 Response 对象，以及 Selector 对象 (对HTML及XML内容)。

* 当shell载入后，将得到一个包含response数据的本地 response 变量，输入 response.body将输出response的包体，输出 response.headers 可以看到response的包头。
* 输入 response.selector 时， 将获取到一个response 初始化的类 Selector 的对象，此时可以通过使用 response.selector.xpath()或response.selector.css() 来对 response 进行查询。
* Scrapy也提供了一些快捷方式, 例如 response.xpath()或response.css()同样可以生效（如之前的案例）。
  + 1. Selectors选择器

实际上，scrapy的css选择器用的是XPath实现。

* + 1. 爬数据

要建立一个Spider， 你必须用scrapy.Spider类创建一个子类，并确定了三个强制的属性 和 一个方法。

* name = "" ：这个爬虫的识别名称，必须是唯一的，在不同的爬虫必须定义不同的名字。
* allow\_domains = [] 是搜索的域名范围，也就是爬虫的约束区域，规定爬虫只爬取这个域名下的网页，不存在的URL会被忽略。
* start\_urls = () ：爬取的URL元祖/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(self, response) ：解析的方法，每个初始URL完成下载后将被调用，调用的时候传入从每一个URL传回的Response对象来作为唯一参数，主要作用如下：
  1. 负责解析返回的网页数据(response.body)，提取结构化数据(生成item)
  2. 生成需要下一页的URL请求。
  3. Item Pipeline

当Item在Spider中被收集之后，它将会被传递到Item Pipeline，这些Item Pipeline组件按定义的顺序处理Item。

每个Item Pipeline都是实现了简单方法的Python类，比如决定此Item是丢弃而存储。以下是item pipeline的一些典型应用：

* + - 1. 验证爬取的数据(检查item包含某些字段，比如说name字段)
      2. 查重(并丢弃)
      3. 将爬取结果保存到文件或者数据库中
    1. 编写item pipeline

编写item pipeline很简单，item pipiline组件是一个独立的Python类，其中process\_item()方法必须实现:

import something

class SomethingPipeline(object):

def \_\_init\_\_(self):

# 可选实现，做参数初始化等

# doing something

def process\_item(self, item, spider):

# item (Item 对象) – 被爬取的item

# spider (Spider 对象) – 爬取该item的spider

# 这个方法必须实现，每个item pipeline组件都需要调用该方法，

# 这个方法必须返回一个 Item 对象，被丢弃的item将不会被之后的pipeline组件所处理。

return item

def open\_spider(self, spider):

# spider (Spider 对象) – 被开启的spider

# 可选实现，当spider被开启时，这个方法被调用。

def close\_spider(self, spider):

# spider (Spider 对象) – 被关闭的spider

# 可选实现，当spider被关闭时，这个方法被调用

* + 1. 完善之前的案例
       1. 将item写入JSON文件

以下pipeline将所有(从所有'spider'中)爬取到的item，存储到一个独立的items.json 文件，每行包含一个序列化为'JSON'格式的'item':

import json

class ItcastJsonPipeline(object):

def \_\_init\_\_(self):

self.file = open('teacher.json', 'wb')

def process\_item(self, item, spider):

content = json.dumps(dict(item), ensure\_ascii=False) + "\n"

self.file.write(content)

return item

def close\_spider(self, spider):

self.file.close()

* + 1. 启用一个Item Pipeline组件

为了启用Item Pipeline组件，必须将它的类添加到 settings.py文件ITEM\_PIPELINES 配置，就像下面这个例子:

# Configure item pipelines

# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html

ITEM\_PIPELINES = {

"mySpider.pipelines.ItcastJsonPipeline":300

}

分配给每个类的整型值，确定了他们运行的顺序，item按数字从低到高的顺序，通过pipeline，通常将这些数字定义在0-1000范围内（0-1000随意设置，数值越低，组件的优先级越高）

* 1. Spiders

Spider类定义了如何爬取某个(或某些)网站。包括了爬取的动作(例如:是否跟进链接)以及如何从网页的内容中提取结构化数据(爬取item)。 换句话说，Spider就是您定义爬取的动作及分析某个网页(或者是有些网页)的地方。

class scrapy.Spider是最基本的类，所有编写的爬虫必须继承这个类。

主要用到的函数及调用顺序为：

\_\_init\_\_() : 初始化爬虫名字和start\_urls列表

start\_requests() 调用make\_requests\_from\_url():生成Requests对象交给Scrapy下载并返回response

parse() : 解析response，并返回Item或Requests（需指定回调函数）。Item传给Item pipline持久化 ， 而Requests交由Scrapy下载，并由指定的回调函数处理（默认parse())，一直进行循环，直到处理完所有的数据为止。

* + 1. 源码参考

#所有爬虫的基类，用户定义的爬虫必须从这个类继承

class Spider(object\_ref):

#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。

#name是spider最重要的属性，而且是必须的。

#一般做法是以该网站(domain)(加或不加 后缀 )来命名spider。 例如，如果spider爬取 mywebsite.com ，该spider通常会被命名为 mywebsite

name = None

#初始化，提取爬虫名字，start\_ruls

def \_\_init\_\_(self, name=None, \*\*kwargs):

if name is not None:

self.name = name

# 如果爬虫没有名字，中断后续操作则报错

elif not getattr(self, 'name', None):

raise ValueError("%s must have a name" % type(self).\_\_name\_\_)

# python 对象或类型通过内置成员\_\_dict\_\_来存储成员信息

self.\_\_dict\_\_.update(kwargs)

#URL列表。当没有指定的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。

if not hasattr(self, 'start\_urls'):

self.start\_urls = []

# 打印Scrapy执行后的log信息

def log(self, message, level=log.DEBUG, \*\*kw):

log.msg(message, spider=self, level=level, \*\*kw)

# 判断对象object的属性是否存在，不存在做断言处理

def set\_crawler(self, crawler):

assert not hasattr(self, '\_crawler'), "Spider already bounded to %s" % crawler

self.\_crawler = crawler

@property

def crawler(self):

assert hasattr(self, '\_crawler'), "Spider not bounded to any crawler"

return self.\_crawler

@property

def settings(self):

return self.crawler.settings

#该方法将读取start\_urls内的地址，并为每一个地址生成一个Request对象，交给Scrapy下载并返回Response

#该方法仅调用一次

def start\_requests(self):

for url in self.start\_urls:

yield self.make\_requests\_from\_url(url)

#start\_requests()中调用，实际生成Request的函数。

#Request对象默认的回调函数为parse()，提交的方式为get

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

return Request(url, dont\_filter=True)

#默认的Request对象回调函数，处理返回的response。

#生成Item或者Request对象。用户必须实现这个类

def parse(self, response):

raise NotImplementedError

@classmethod

def handles\_request(cls, request):

return url\_is\_from\_spider(request.url, cls)

def \_\_str\_\_(self):

return "<%s %r at 0x%0x>" % (type(self).\_\_name\_\_, self.name, id(self))

\_\_repr\_\_ = \_\_str\_\_

* + 1. 主要属性和方法
       1. name

定义spider名字的字符串。

例如，如果spider爬取 mywebsite.com ，该spider通常会被命名为 mywebsite

* + - 1. allowed\_domains

包含了spider允许爬取的域名(domain)的列表，可选。

* + - 1. start\_urls

初始URL元祖/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。

* + - 1. start\_requests(self)

该方法必须返回一个可迭代对象(iterable)。该对象包含了spider用于爬取（默认实现是使用 start\_urls 的url）的第一个Request。

当spider启动爬取并且未指定start\_urls时，该方法被调用。

* + - 1. parse(self, response)

当请求url返回网页没有指定回调函数时，默认的Request对象回调函数。用来处理网页返回的response，以及生成Item或者Request对象。

* + - 1. log(self, message[, level, component])

使用 scrapy.log.msg() 方法记录(log)message。 更多数据请参见 [logging](http://4.7.html/)

* + 1. 案例：腾讯招聘网自动翻页采集
       1. 创建一个新的爬虫：

scrapy genspider tencent "tencent.com"

* + - 1. 编写items.py

获取职位名称、详细信息、

class TencentItem(scrapy.Item):

name = scrapy.Field()

detailLink = scrapy.Field()

positionInfo = scrapy.Field()

peopleNumber = scrapy.Field()

workLocation = scrapy.Field()

publishTime = scrapy.Field()

* + - 1. 编写tencent.py

# tencent.py

from mySpider.items import TencentItem

import scrapy

import re

class TencentSpider(scrapy.Spider):

name = "tencent"

allowed\_domains = ["hr.tencent.com"]

start\_urls = [

"http://hr.tencent.com/position.php?&start=0#a"

]

def parse(self, response):

for each in response.xpath('//\*[@class="even"]'):

item = TencentItem()

name = each.xpath('./td[1]/a/text()').extract()[0]

detailLink = each.xpath('./td[1]/a/@href').extract()[0]

positionInfo = each.xpath('./td[2]/text()').extract()[0]

peopleNumber = each.xpath('./td[3]/text()').extract()[0]

workLocation = each.xpath('./td[4]/text()').extract()[0]

publishTime = each.xpath('./td[5]/text()').extract()[0]

#print name, detailLink, catalog, peopleNumber, workLocation,publishTime

item['name'] = name.encode('utf-8')

item['detailLink'] = detailLink.encode('utf-8')

item['positionInfo'] = positionInfo.encode('utf-8')

item['peopleNumber'] = peopleNumber.encode('utf-8')

item['workLocation'] = workLocation.encode('utf-8')

item['publishTime'] = publishTime.encode('utf-8')

curpage = re.search('(\d+)',response.url).group(1)

page = int(curpage) + 10

url = re.sub('\d+', str(page), response.url)

# 发送新的url请求加入待爬队列，并调用回调函数 self.parse

yield scrapy.Request(url, callback = self.parse)

# 将获取的数据交给pipeline

yield item

* + - 1. 编写pipeline.py文件

import json

#class ItcastJsonPipeline(object):

class TencentJsonPipeline(object):

def \_\_init\_\_(self):

#self.file = open('teacher.json', 'wb')

self.file = open('tencent.json', 'wb')

def process\_item(self, item, spider):

content = json.dumps(dict(item), ensure\_ascii=False) + "\n"

self.file.write(content)

return item

def close\_spider(self, spider):

self.file.close()

* + - 1. 在 setting.py 里设置ITEM\_PIPELINES

ITEM\_PIPELINES = {

#'mySpider.pipelines.SomePipeline': 300,

#"mySpider.pipelines.ItcastJsonPipeline":300

"mySpider.pipelines.TencentJsonPipeline":300

}

* + - 1. 执行爬虫：

scrapy crawl tencent

* + 1. 请思考 parse()方法的工作机制：
       1. 因为使用的yield，而不是return。parse函数将会被当做一个生成器使用。scrapy会逐一获取parse方法中生成的结果，并判断该结果是一个什么样的类型；
       2. 如果是request则加入爬取队列，如果是item类型则使用pipeline处理，其他类型则返回错误信息。
       3. scrapy取到第一部分的request不会立马就去发送这个request，只是把这个request放到队列里，然后接着从生成器里获取；
       4. 取尽第一部分的request，然后再获取第二部分的item，取到item了，就会放到对应的pipeline里处理；
       5. parse()方法作为回调函数(callback)赋值给了Request，指定parse()方法来处理这些请求 scrapy.Request(url, callback=self.parse)
       6. Request对象经过调度，执行生成 scrapy.http.response()的响应对象，并送回给parse()方法，直到调度器中没有Request（递归的思路）
       7. 取尽之后，parse()工作结束，引擎再根据队列和pipelines中的内容去执行相应的操作；
       8. 程序在取得各个页面的items前，会先处理完之前所有的request队列里的请求，然后再提取items。

1. 课程总结
   1. 重点
      * 1. Scrapy框架的架构和数据流。
        2. Scrapy框架的使用。
   2. 难点
      * 1. Scrapy框架个组件的关系。
2. 课后练习
   1. **必做题**
      * 1. 实现上课代码，尽量领会后独立实现，包括自己简要的注释，变量命名要做到“见名知意”
        2. 分别使用使用Scrapy爬取网站<http://quotes.toscrape.com/>和所有后继页面以及作者介绍页面，并分别解析出。这是一个专用于练习爬虫编程的网站，网站页面代码简洁，适合初步练习用。