Scrapy 基本概念

Request

可以先阅读一下scrapy有关Request类的源文件：

In [ ]:
```
# Scrapy Request部分代码
class Request(object_ref):

    def __init__(self, url, callback=None, method='GET', headers=None,
body=None,
                 cookies=None, meta=None, encoding='utf-8', priority=0,
                 dont_filter=False, errback=None):

        self._encoding = encoding  # this one has to be set first
        self.method = str(method).upper()
        self._set_url(url)
        self._set_body(body)
        assert isinstance(priority, int), "Request priority not an integ
er: %r" % priority
        self.priority = priority

        assert callback or not errback, "Cannot use errback without a ca
llback"
        self.callback = callback
        self.errback = errback

        self.cookies = cookies or {}
        self.headers = Headers(headers or {}, encoding=encoding)
        self.dont_filter = dont_filter

        self._meta = dict(meta) if meta else None

    @property
    def meta(self):
        if self._meta is None:
            self._meta = {}
        return self._meta
```

Request类常用参数

- url: 就是需要请求，并进行下一步处理的url
- callback: 指定该请求返回的Response，由那个函数来处理。
- method: 请求一般不需要指定，默认GET方法，可设置为"GET", "POST", "PUT"等，且保证字符串大写
- headers: 请求时，包含的头文件。一般不需要。内容一般如下：
  - Host: media.readthedocs.org
  - User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:33.0) Gecko/20100101 Firefox/33.0

    Accept: text/css,/;q=0.1
  - Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
  - Accept-Encoding: gzip, deflate
  - Referer: http://scrapy-chs.readthedocs.org/zh_CN/0.24/
  - Cookie: _ga=GA1.2.1612165614.1415584110;
  - Connection: keep-alive
  - If-Modified-Since: Mon, 25 Aug 2014 21:59:35 GMT
  - Cache-Control: max-age=0
- meta: 比较常用，在不同的请求之间传递数据使用的。字典dict型。例如：

```
request_with_cookies = Request(
          url="http://www.example.com",
          cookies={'currency': 'USD', 'country': 'UY'},
          meta={'dont_merge_cookies': True}
        )
```

- encoding: 使用默认的 'utf-8' 就行。
- dont_filter: 表明该请求不由调度器过滤。这是当你想使用多次执行相同的请求,忽略重复的过滤器。默认为False。
- errback: 指定错误处理函数

Response

Scrapy定义的Response部分源代码如下：

In [ ]:
```
# 部分代码
class Response(object_ref):
    def __init__(self, url, status=200, headers=None, body='', flags=No
ne, request=None):
        self.headers = Headers(headers or {})
        self.status = int(status)
        self._set_body(body)
        self._set_url(url)
        self.request = request
        self.flags = [] if flags is None else list(flags)

    @property
    def meta(self):
        try:
            return self.request.meta
        except AttributeError:
            raise AttributeError("Response.meta not available, this resp
onse " \
                "is not tied to any request")
```

可以看出来，Response与Response的参数很多是相同的。特殊的有以下几个：

- status: 响应码
- _set_body(body)： 响应体
- _set_url(url)：响应url
- self.request = request

发送POST请求

之前学习得scrapy方法都是发送GET型HTTP请求，我们还可以使用 yield scrapy.FormRequest(url, formdata, callback)方法发送POST请求。

如果希望程序执行一开始就发送POST请求，可以重写Spider类的start_requests(self) 方法，并且不再调用start_urls里的url。

In [ ]:
```
"""在start_request() 中使用scrapy.FormRequest() 发送POST请求"""
class mySpider(scrapy.Spider):
    # start_urls = ["http://www.example.com/"]

    def start_requests(self):
        url = 'http://www.renren.com/PLogin.do'

        # FormRequest 是Scrapy发送POST请求的方法
        yield scrapy.FormRequest(
            url = url,
            formdata = {"email" : "mr_mao_hacker@163.com", "password" :
"axxxxxxxe"},
            callback = self.parse_page
        )
    def parse_page(self, response):
        # do something
```

模拟登陆

可以使用FormRequest.from_response()方法模拟用户登录。

通常网站通过 实现对某些表单字段（如数据或是登录界面中的认证令牌等）的预填充。

使用Scrapy抓取网页时，如果想要预填充或重写像用户名、用户密码这些表单字段， 可以使用 FormRequest.from_response() 方法实现。

下面是使用这种方法的爬虫例子:

In [ ]:
```
"""使用FormRequest.form_response()模拟用户登录"""

import scrapy

class LoginSpider(scrapy.Spider):
    name = 'example.com'
    start_urls = ['http://www.example.com/users/login.php']

    def parse(self, response):
        return scrapy.FormRequest.from_response(
            response,
            formdata={'username': 'john', 'password': 'secret'},
            callback=self.after_login
        )

    def after_login(self, response):
        # check login succeed before going on
        if "authentication failed" in response.body:
            self.log("Login failed", level=log.ERROR)
            return

        # continue scraping with authenticated session...
```

知乎爬虫案例

1. 编写Spider类

In [ ]:
```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
from scrapy.spiders import CrawlSpider, Rule
from scrapy.selector import Selector
from scrapy.linkextractors import LinkExtractor
from scrapy import Request, FormRequest
from zhihu.items import ZhihuItem

class ZhihuSipder(CrawlSpider) :
    name = "zhihu"
    allowed_domains = ["www.zhihu.com"]
    start_urls = [
        "http://www.zhihu.com"
    ]
    rules = (
        Rule(LinkExtractor(allow = ('/question/\d+#.*?', )), callback = 
'parse_page', follow = True),
        Rule(LinkExtractor(allow = ('/question/\d+', )), callback = 'par
se_page', follow = True),
    )

    headers = {
    "Accept": "*/*",
    "Accept-Encoding": "gzip,deflate",
    "Accept-Language": "en-US,en;q=0.8,zh-TW;q=0.6,zh;q=0.4",
    "Connection": "keep-alive",
    "Content-Type":" application/x-www-form-urlencoded; charset=UTF-8",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) Apple
WebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.111 Safari/537.36",
    "Referer": "http://www.zhihu.com/"
    }

    #重写了爬虫类的方法, 实现了自定义请求, 运行成功后会调用callback回调函数
    def start_requests(self):
        return [Request("https://www.zhihu.com/login", meta = {'cookieja
r' : 1}, callback = self.post_login)]

    def post_login(self, response):
        print 'Preparing login'
        #下面这句话用于抓取请求网页后返回网页中的_xsrf字段的文字, 用于成功提交表单
        xsrf = Selector(response).xpath('//input[@name="_xsrf"]/@value')
.extract()[0]
        print xsrf
        #FormRequeset.from_response是Scrapy提供的一个函数, 用于post表单
        #登陆成功后, 会调用after_login回调函数
        return [FormRequest.from_response(response,   #"http://www.zhihu
.com/login",
                            meta = {'cookiejar' : response.meta['cookieja
r']},
                            headers = self.headers,  #注意此处的headers
                            formdata = {
                            '_xsrf': xsrf,
                            'email': '1095511864@qq.com',
                            'password': '123456'
                            },
```

```
                            callback = self.after_login,
                            dont_filter = True
                            )]

    def after_login(self, response) :
        for url in self.start_urls :
            yield self.make_requests_from_url(url)

    def parse_page(self, response):
        problem = Selector(response)
        item = ZhihuItem()
        item['url'] = response.url
        item['name'] = problem.xpath('//span[@class="name"]/text()').ext
ract()
        print item['name']
        item['title'] = problem.xpath('//h2[@class="zm-item-title zm-edi
table-content"]/text()').extract()
        item['description'] = problem.xpath('//div[@class="zm-editable-c
ontent"]/text()').extract()
        item['answer']= problem.xpath('//div[@class=" zm-editable-conten
t clearfix"]/text()').extract()
        return item
```

1. Item类设置

In [ ]:
```
from scrapy.item import Item, Field

class ZhihuItem(Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    url = Field()  #保存抓取问题的url
    title = Field()  #抓取问题的标题
    description = Field()  #抓取问题的描述
    answer = Field()  #抓取问题的答案
    name = Field()  #个人用户的名称
```

1. setting.py 设置抓取间隔

```
BOT_NAME = 'zhihu'

SPIDER_MODULES = ['zhihu.spiders']
NEWSPIDER_MODULE = 'zhihu.spiders'
DOWNLOAD_DELAY = 0.25    #设置下载间隔为250ms
```

Downloader 中间件

下载中间件是处于引擎(crawler.engine)和下载器(crawler.engine.download())之间的一层组件，可以有多个下载中间件被加载运行。

1. 当引擎传递请求给下载器的过程中，下载中间件可以对请求进行处理 （例如增加http header信息，增加proxy信息等）；
2. 在下载器完成http请求，传递响应给引擎的过程中， 下载中间件可以对响应进行处理（例如进行gzip的解压等）。

中间件的激活

只有激活的下载器中间件组件，将其加入到 DOWNLOADER_MIDDLEWARES 设置中才能使用。

这一设置是一个字典(dict)，键为中间件类的路径，值为其中间件的顺序(order)。

例如:

```
DOWNLOADER_MIDDLEWARES = {
    'mySpider.middlewares.MyDownloaderMiddleware': 543,
}
```

编写下载器中间件十分简单。每个中间件组件是一个定义了以下一个或多个方法的Python类:

> class scrapy.contrib.downloadermiddleware.DownloaderMiddleware

中间件类中的主要方法

process_request(self, request, spider)

当每个request通过下载中间件时，该方法被调用。

process_request() 必须返回以下其中之一:

- 一个 None
- 一个 Response 对象
- 一个 Request 对象 或 raise IgnoreRequest:

函数功能:

1. 如果其返回 None ，Scrapy将继续处理该request，执行其他的中间件的相应方法，直到合适的下载器处理函数(download handler)被调用， 该request被执行(其response被下载)。
2. 如果其返回 Response 对象，Scrapy将不会调用任何其他的 process_request() 或 process_exception() 方法，或相应地下载函数； 其将返回该response。 已安装的中间件的 process_response() 方法则会在每个response返回时被调用。
3. 如果其返回 Request 对象，Scrapy则停止调用 process_request方法并重新调度返回的request。当新返回的request被执行后， 相应地中间件链将会根据下载的response被调用。
4. 如果其raise一个 IgnoreRequest 异常，则安装的下载中间件的 process_exception() 方法会被调用。如果没有任何一个方法处理该异常， 则request的errback(Request.errback)方法会被调用。如果没有代码处理抛出的异常， 则该异常被忽略且不记录(不同于其他异常那样)。

参数:

- request (Request 对象) – 处理的request
- spider (Spider 对象) – 该request对应的spider

process_response(self, request, response, spider)

当下载器完成http请求，传递响应给引擎的时候调用

process_request() 必须返回以下其中之一:

- 返回一个 Response 对象

- 返回一个 Request 对象或raise一个 IgnoreRequest 异常。

函数功能:

1. 如果其返回一个 Response (可以与传入的response相同，也可以是全新的对象)，该response会被在链中的其他中间件的 process_response() 方法处理。
2. 如果其返回一个 Request 对象，则中间件链停止， 返回的request会被重新调度下载。处理类似于 process_request() 返回request所做的那样。
3. 如果其抛出一个 IgnoreRequest 异常，则调用request的errback(Request.errback)。 如果没有代码处理抛出的异常，则该异常被忽略且不记录(不同于其他异常那样)。

参数:

- request (Request 对象) – response所对应的request
- response (Response 对象) – 被处理的response
- spider (Spider 对象) – response所对应的spider

Middlewares案例

1. 创建middlewares.py文件。

Scrapy代理IP、Uesr-Agent的切换都是通过DOWNLOADER_MIDDLEWARES进行控制，我们在settings.py同级目录下创建middlewares.py文件，包装所有请求。

In [ ]:
```
# middlewares.py

#!/usr/bin/env python
# -*- coding:utf-8 -*-

import random
import base64

from settings import USER_AGENTS
from settings import PROXIES

# 随机的User-Agent
class RandomUserAgent(object):
    def process_request(self, request, spider):
        useragent = random.choice(USER_AGENTS)

        request.headers.setdefault("User-Agent", useragent)

class RandomProxy(object):
    def process_request(self, request, spider):
        proxy = random.choice(PROXIES)

        if proxy['user_passwd'] is None:
            # 没有代理账户验证的代理使用方式
            request.meta['proxy'] = "http://" + proxy['ip_port']
        else:
            # 对账户密码进行base64编码转换
            base64_userpasswd = base64.b64encode(proxy['user_passwd'])
            # 对应到代理服务器的信令格式里
            request.headers['Proxy-Authorization'] = 'Basic ' + base64_u
serpasswd
            request.meta['proxy'] = "http://" + proxy['ip_port']
```

为什么**HTTP**代理要使用**base64**编码

HTTP代理的原理很简单，就是通过HTTP协议与代理服务器建立连接，协议信令中包含要连接到的远程主机的IP和端口号，如果有需要身份验证的话还需要加上授权信息，服务器收到信令后首先进行身份验证，通过后便与远程主机建立连接，连接成功之后会返回给客户端200，表示验证通过，就这么简单，下面是具体的信令格式：

> CONNECT 59.64.128.198:21 HTTP/1.1 Host: 59.64.128.198:21 Proxy-Authorization: Basic bGV2I1TU5OTIz User-Agent: OpenFetion

其中Proxy-Authorization是身份验证信息，Basic后面的字符串是用户名和密码组合后进行base64编码的结果，也就是对username:password进行base64编码。

> HTTP/1.0 200 Connection established

客户端收到收面的信令后表示成功建立连接，接下来要发送给远程主机的数据就可以发送给代理服务器了，代理服务器建立连接后会在根据IP地址和端口号对应的连接放入缓存，收到信令后再根据IP地址和端口号从缓存中找到对应的连接，将数据通过该连接转发出去。

1. 修改settings.py配置USER_AGENTS和PROXIES

- 添加USER_AGENTS：

```
 USER_AGENTS = [
    "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64
; Trident/5.0; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET CLR 2
.0.50727; Media Center PC 6.0)",
    "Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.
0; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30
729; .NET CLR 3.0.30729; .NET CLR 1.0.3705; .NET CLR 1.1.4322)",
    "Mozilla/4.0 (compatible; MSIE 7.0b; Windows NT 5.2; .NET CLR
1.1.4322; .NET CLR 2.0.50727; InfoPath.2; .NET CLR 3.0.04506.30)"
,
    "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN) AppleWebKit/5
23.15 (KHTML, like Gecko, Safari/419.3) Arora/0.3 (Change: 287 c9
dfb30)",
    "Mozilla/5.0 (X11; U; Linux; en-US) AppleWebKit/527+ (KHTML,
like Gecko, Safari/419.3) Arora/0.6",
    "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2pre
) Gecko/20070215 K-Ninja/2.1.1",
    "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9) Gecko
/20080705 Firefox/3.0 Kapiko/3.0",
    "Mozilla/5.0 (X11; Linux i686; U;) Gecko/20070322 Kazehakase/
0.4.5"
    ]
```

- 添加代理IP设置PROXIES：

免费代理IP可以网上搜索，或者付费购买一批可用的私密代理IP：

```
PROXIES = [
    {'ip_port': '111.8.60.9:8123', 'user_passwd': 'user1:pass1'},
```

```
    {'ip_port': '101.71.27.120:80', 'user_passwd': 'user2:pass2'},
    {'ip_port': '122.96.59.104:80', 'user_passwd': 'user3:pass3'},
    {'ip_port': '122.224.249.122:8088', 'user_passwd': 'user4:pass
4'},
]
```

- 除非特殊需要，禁用cookies，防止某些网站根据Cookie来封锁爬虫。

  > COOKIES_ENABLED = False

- 设置下载延迟

  > DOWNLOAD_DELAY = 3

- 最后设置setting.py里的DOWNLOADER_MIDDLEWARES，添加自己编写的下载中间件类。

```
DOWNLOADER_MIDDLEWARES = {
    #'mySpider.middlewares.MyCustomDownloaderMiddleware': 543,
    'mySpider.middlewares.RandomUserAgent': 1,
    'mySpider.middlewares.ProxyMiddleware': 100
}
```

Settings

Scrapy设置(settings)提供了定制Scrapy组件的方法。可以控制包括核心(core)，插件(extension)，pipeline及spider组件。比如 设置Json Pipeliine、LOG_LEVEL等。参考文档：http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html#topics-settings-ref

下面是一些主要的内置设置参考：

- BOT_NAME
  - 默认: 'scrapybot'
  - 当您使用 startproject 命令创建项目时其也被自动赋值。
- CONCURRENT_ITEMS
  - 默认: 100
  - Item Processor(即 Item Pipeline) 同时处理(每个response的)item的最大值。
- CONCURRENT_REQUESTS
  - 默认: 16
  - Scrapy downloader 并发请求(concurrent requests)的最大值。
- DEFAULT_REQUEST_HEADERS
  - 默认: 如下

```
{
'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,
*/*;q=0.8',
'Accept-Language': 'en',
}
```

- Scrapy HTTP Request使用的默认header。

- DEPTH_LIMIT

    - 默认: 0
    - 爬取网站最大允许的深度(depth)值。如果为0，则没有限制。
- DOWNLOAD_DELAY
    - 默认: 0
    - 下载器在下载同一个网站下一个页面前需要等待的时间。该选项可以用来限制爬取速度， 减轻服务器压力。同时也支持小数:

        > DOWNLOAD_DELAY = 0.25 # 250 ms of delay

    - 默认情况下，Scrapy在两个请求间不等待一个固定的值， 而是使用0.5到1.5之间的一个随机值 * DOWNLOAD_DELAY 的结果作为等待间隔。
- DOWNLOAD_TIMEOUT
    - 默认: 180
    - 下载器超时时间(单位: 秒)。
- ITEM_PIPELINES
    - 默认: {}
    - 保存项目中启用的pipeline及其顺序的字典。该字典默认为空，值(value)任意，不过值(value)习惯设置在0-1000范围内，值越小优先级越高。

```
ITEM_PIPELINES = {
'mySpider.pipelines.SomethingPipeline': 300,
'mySpider.pipelines.ItcastJsonPipeline': 800,
}
```

- LOG_ENABLED
    - 默认: True
    - 是否启用logging。
- LOG_ENCODING
    - 默认: 'utf-8'
    - logging使用的编码。
- LOG_LEVEL
    - 默认: 'DEBUG'
    - log的最低级别。可选的级别有: CRITICAL、ERROR、WARNING、INFO、DEBUG 。
- USER_AGENT
    - 默认: "Scrapy/VERSION (+http://scrapy.org)"
    - 爬取的默认User-Agent，除非被覆盖。
- PROXIES： 代理设置
    - 示例:

```
PROXIES = [
{'ip_port': '111.11.228.75:80', 'password': ''},
{'ip_port': '120.198.243.22:80', 'password': ''},
{'ip_port': '111.8.60.9:8123', 'password': ''},
{'ip_port': '101.71.27.120:80', 'password': ''},
{'ip_port': '122.96.59.104:80', 'password': ''},
{'ip_port': '122.224.249.122:8088', 'password':''},
]
```

- COOKIES_ENABLED = False
    - 禁用Cookies

Scrapy 实战

1. 阳光热线问政平台内容爬取 http://wz.sun0769.com/index.php/question/questionType?type=4

目标：爬取投诉帖子的编号、帖子的url、帖子的标题，和帖子里的内容。 要求：综合使用scrapy框架方法，实现多页内容爬取。

1. 新浪分类资讯爬取

目标：爬取新浪网导航页所有下所有大类、小类、小类里的子链接，以及子链接页面的新闻内容。