

Debugging Spiders

PART 1

Scrapy spider'ımızı debug etmek için kullanılacak yöntemlerden bahsedilecek, aşağıdaki linkte bazı tekniklerden zaten bahsedilmiş, önce bunları açıklayacağız daha sonra part 2'de ekstra yöntemlere bakacağız.

<https://docs.scrapy.org/en/0.22/topics/debug.html>

1.Parse Command

Spider outputunu check etmenin en basit yolu budur. Temelde bu komutun yaptığı şey şu, specified link'i base alarak spider outputunu check etmek.

Parse komutunu aşağıdaki gibi execute edebiliriz:

```
scrapy parse --spider=mypider -c parse_item -d 2  
<item_url>
```

hangi spider'ı execute edeceğimizi, callback methodu ve depth'i belirttikten sonra ilgili url'yi yazıyoruz ve output check ediyoruz.

Depth parametresi şuna karşılık geliyor: Diyelim ki spider'ımızın link A'dan → link B'ye oradan → link C'ye oradan da → link D'ye gidiyor bu spider'ın depthi 3 demektir.

Uygulamaya bakalım, parse commandini daha önce build edilen countries spider'ı için kullanalım, aşağıda countries spider'ını görüyoruz.

Parse_country methodu için parse command'i çalıştırıp yield edilen outputu görmek istiyoruz.

```
class CountriesSpider(scrapy.Spider):
    name = 'countries'
    allowed_domains = ['www.worldometers.info']
    start_urls = [
        'https://www.worldometers.info/world-population/population-by-country/']

    def parse(self, response):
        countries = response.xpath("//td/a")
        for country in countries:
            name = country.xpath("../text()").get()
            link = country.xpath("../@href").get()

            # absolute_url = f"https://www.worldometers.info/{link}"
            # absolute_url = response.urljoin(link)

            yield response.follow(url=link, callback=self.parse_country, meta={'country_name': name})

    def parse_country(self, response):
        name = response.request.meta['country_name']
        rows = response.xpath(
            "(/table[@class='table table-striped table-bordered table-hover table-condensed table-list'])[1]/tbody/tr")
        for row in rows:
            year = row.xpath("../td[1]/text()").get()
            population = row.xpath("../td[2]/strong/text()").get()
            yield {
                'country_name': name,
                'year': year,
                'population': population
            }
```

Öncelikle terminali açıyoruz ve aşağıdaki kodu yazıyoruz.



```
PS C:\Users\Ahmed\projects\worldometers> scrapy parse --spider=countries -c parse_country https://www.worldometers.info/world-population/
```

```
scrapy parse --spider=countries -c parse_country
https://www.worldometers.info/world-population/china-
population/
```

Burada verilen link China'nın country link'i, parse country'e normalde de country link requestlerinin response'ları geridönüyordu. Depth'i belirtmedik şart değil.

Yani bizim yukarıdaki komutla yaptığımız şey şu oldu, komut içindeki linke git, daha sonra o linkin response'unu al o response'u parse_country methoduna ver ve parse country methodunu çalıştır.

İyi ama dikkat edersen parse country methodunun sağlıklı çalışabilmesi için country_name verisinin meta ile parse methodundan pass edilmesi gerekiyordu, meta dictionarysi içinde request ile gönderilen bu veriler response ile catch ediliyordu.

Biz şuan parse methodunu çalıştırmadan artificial olarak parse_country'i çalıştırdığımız için method bu meta parametrelerine sahip değil bu yüzden yukarıdaki haliyle komutu çalıştırırsak hata alırız.

Yapmamız gereken istenilen meta parametresini manuel olarak komuta belirtmek, böylece meta parametresi alınmış gibi method çalışacak:

```
scrapy parse --spider=countries -c --  
meta='{\"country_name\": \"China\"}' parse_country  
https://www.worldometers.info/world-population/china-population/
```

Bu şekilde meta içerisinde “China” country name'ini manuel olarak geçirmiş olduk ve sonucunda parse_country tarafından yield edilen değerleri aşağıdaki gibi görebiliyoruz:

```
py.cfg      population': '926,240,885', 'year  
' : '1975'),  
{'country_name': 'China', '  
population': '827,601,394', 'year  
' : '1970'),  
{'country_name': 'China', '  
population': '724,218,968', 'year  
' : '1965'),  
{'country_name': 'China', '  
population': '660,408,056', 'year  
' : '1960'),  
{'country_name': 'China', '  
population': '612,241,554', 'year  
' : '1955']}]  
  
# Requests -----  
[ ]
```

2. Scrapy Shell

İkinci debug yöntemi de: scrapy shell'i spider'ın içerisinde invoke etmek.

Bunu yapmak için öncelikle spider'ın en başında aşağıdaki gibi inspect_response modülünü import ediyoruz.

```
from scrapy.shell import inspect_response
```

Daha sonra parse_country methodunu inspect etmek istediğim için bu methodun içeriğini commente alıyorum ve içine

```
inspect_response(response, self)
```

İlk argüman inspect edeceğimiz response objesi olacak, ikinci argüman ise spider objesi olacak, biz burada current spider object'i self olarak veriyoruz.

```
yield response.follow(url=link, callback=self.parse_country, meta={'country_name': name})

def parse_country(self, response):
    inspect_response(response, self)
    # name = response.request.meta['country_name']
    # rows = response.xpath(
    #     "(/table[@class='table table-striped table-bordered table-hover table-condensed table-list'])[1]/tbody/tr
    # for row in rows:
    #     year = row.xpath("./td[1]/text()").get()
    #     ong/text()").get()
```

Bu değişikliklerden sonra, terminalden standard şekilde spider'ı çalıştırdığımızda yani:

`scrapy crawl countries` dediğimizde `inspect_response` methodu bize aşağıdaki scrapy shell'i açıyor aşağıdaki gibi

```
2019-09-29 10:35:50 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.worldometers.info/world-population/population-by-country/>
[s] Available Scrapy objects:
[s] scrapy      scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s] crawler      <scrapy.crawler.Crawler object at 0x000001D7A43595C8>
[s] item          {}
[s] request       <GET https://www.worldometers.info/world-population/china-population/>
[s] response      <200 https://www.worldometers.info/world-population/china-population/>
[s] settings      <scrapy.settings.Settings object at 0x000001D7A446EB48>
[s] spider        <CountriesSpider 'countries' at 0x1d7a459e548>
[s] Useful shortcuts:
[s] shelp()        Shell help (print this help)
[s] view(response) View response in a browser
In [1]:
```

Bu noktadan sonra normal shell'de yapılan işlemleri yapabilirim mesela

`request.headers` ile request headers'ı yazdırabiliriz veya

`response.body` ile response'un html markup'ını görebiliriz.

veya `view(response)` ile response'u browser'da görebiliriz.

Burada göreceğimiz response ilk country link'in response'u olacaktır, çünkü `country_parse` ilk kez country link response ile çağırılıyor.

Peki ben direk scrapy Shell başlatsam burada yaptığımızı yapamaz mıyım? Hayır yapamam, çünkü shell başlatırsam, hangi url'ye request gönderceğimi direkt vermem gerek, burada ise biz spider içindeki `country_parse` methodunu debug etmek istiyoruz, bu methoda gelen response'un request'ini

kendimiz oluşturmadık, bir üstündeki parse methodu içerisinde request gönderiliyor alınan response bu methoda gönderiliyor, debug bu yüzden önemli, spider içerisindeki ilk methodu debug etmek daha kolay olabilir, Shell ile bunu yapabiliriz ama ikinci veya daha sonraki methodlar için debugging yöntemlerini kullanmak avantajlı olacak.

3. Open in browser

Yukarıda shell ile response'u nasıl browser'da açabildiğimizi gördük :

`view(response)` bunun bir başka yolu da `open_in_browser` methodunu kullanmak.

Bu methodu kullanmak için öncelikle import ediyorum:

```
from scrapy.utils.response import open_in_browser
```

Daha sonra spider içinde istediğim bir response'u `open_in_browser(response)` komutu ile view edebilirim.

Örneğin bu methodu aşağıdaki gibi `parse_country` içerisinde çağırırsam, her country response browser'da view edilecektir, çünkü her response alındıktan sonra bu method çağırılacak.

```
def parse_country(self, response):
    open_in_browser(response)
    # name = response.request.meta['country_name']
    # rows = response.xpath(
    #     "(//table[@class='table table-striped table-bordered table-hov
    # for row in rows:
    #     year = row.xpath("./td[1]/text()").get()
    #     population = row.xpath("./td[2]/strong/text()").get()
    #     yield {
    #         'country_name': name,
    #         'year': year,
```


4. Logging

Spider'ımızı debug etmenin son documented yolu da logging module.

Şimdi bunu anlayalım. Öncelikle gerekli module'u spider'ın başında import ediyoruz:

```
import logging
```

Daha sonra yine spider içerisinde istediğimiz bir response'u inspect etmek için

`logging.info(response.status)` komutunu kullanabiliriz.

```
def parse_country(self, response):
    logging.info(response.status)
    # name = response.request.meta['start_urls']
    # rows = response.xpath(
    #     "(/table[@class='table table-striped table-bordered table-hover table-condensed'])"
    # for row in rows:
    #     year = row.xpath("./td[1]/text()").get()
    #     population = row.xpath("./td[2]/strong/text()").get()
```

Sonuçta

`scrapy crawl countries` ile spider'ı çalıştırınca ilgili response'la ilgili aşağıdaki gibi bazı bilgileri elde edebiliyoruz.

```
2019-09-29 10:50:15 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min)
2019-09-29 10:50:15 [scrapy.extensions.telnet] INFO: Telnet console listening on 127.0.0.1:5555
2019-09-29 10:50:16 [scrapy.core.engine] DEBUG: Crawled (404) <GET https://www.worldometers.info/world-population/population-by-country/>
2019-09-29 10:50:16 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.worldometers.info/world-population/population-by-country/>
2019-09-29 10:50:17 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.worldometers.info/world-population/population-by-country/>
2019-09-29 10:50:17 [root] INFO: 200
2019-09-29 10:50:17 [scrapy.core.engine] INFO: Closing spider (finished)
2019-09-29 10:50:17 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 823,
 'downloader/request_count': 3,
 'downloader/request_method_count/GET': 3,
 'downloader/response_bytes': 34197,
 'downloader/response_count': 3,
```

PART 2

Bu kısımda ise, VSCode editörünü kullanarak spider'ımızı nasıl debug edebileceğimizi göreceğiz:

Bunun için yapmamız gereken spider'ımızı programatik olarak çalıştırmalıyız, yani terminalden değil, bir başka .py dosyasının içinden spider'ı çalıştırmalıyız ki VSCode kullanarak debugging yapabilelim.

Project root directory'de (settings.py, pipelines.py, middlewares.py gibi dosyaların olduğu klasörde) `runner.py` adından bir runner scripti oluşturuyorum.

Daha sonra bu script içerisinde aşağıdaki gibi bazı importları yapacağım:

```
→import scrapy
```

```
→import scrapy.crawler import CrawlerProcess
```

```
→from scrapy.utils.project import  
get_project_settings
```

Daha sonra debug edeceğim spider class'ımı import edeceğim

```
→from worldometers.spiders.countries import  
CountriesSpider
```


Daha sonra bir CrawlerProcess class instantiate edelim:

```
process
```

```
=CrawlerProcess(settings=get_project_settings())
```

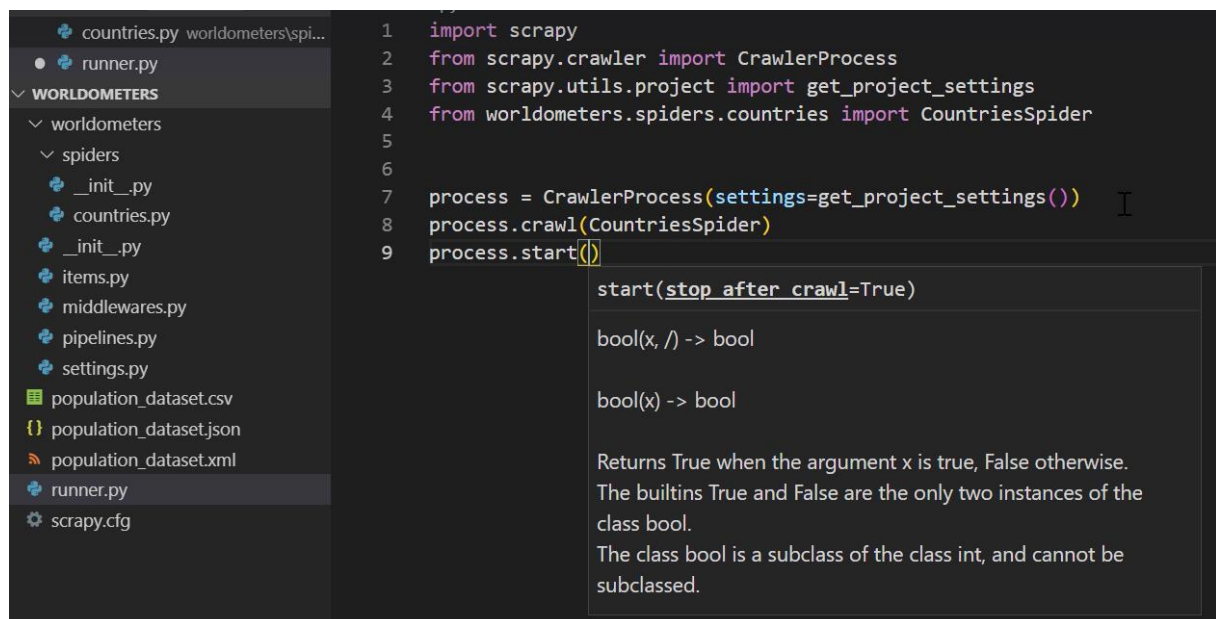
process ismindeki crawlerprocess objesi input olarak settings.py içerisindeki bilgileri settings argumantı aracılığı ile aldı.

Daha sonra spider'ı çalıştırmak için:

```
process.crawl(CountriesSpider)
```

```
process.start()
```

Sonuçta runner.py'nin içi aşağıdaki gibi oldu.



```
1 import scrapy
2 from scrapy.crawler import CrawlerProcess
3 from scrapy.utils.project import get_project_settings
4 from worldometers.spiders.countries import CountriesSpider
5
6
7 process = CrawlerProcess(settings=get_project_settings())
8 process.crawl(CountriesSpider)
9 process.start()
```

start(stop_after_crawl=True)

bool(x, /) -> bool

bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

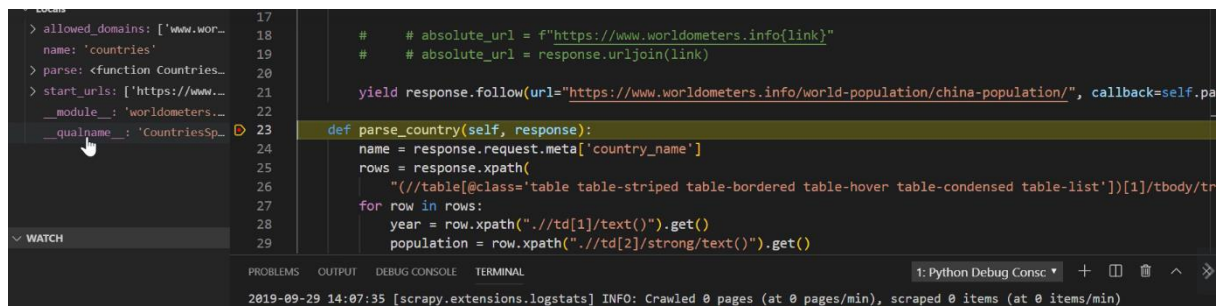
Son olarak debug edilecek noktaya işaret koyuyoruz, bu nokta spider'ın içindeki parse_country methodu olacak, yani bu method çağırıldığında program ilerlemek için bizim komutlarımızı bekleyecek:

Bu işareti aşağıdaki gibi spider class'ımızın içindeki parse_country methodunun yanına tıklayarak koyabiliriz.

```
21         yield response.follow(url="https://www.worldometers.info/world-population/china-population/", callback=self.parse_country)
22
23     def parse_country(self, response):
24         name = response.request.meta['country_name']
25         rows = response.xpath(
26             "(/table[@class='table table-striped table-bordered table-hover table-condensed table-list'])[1]/tbody/tr"
27         )
28         for row in rows:
29             year = row.xpath("./td[1]/text()").get()
30             population = row.xpath("./td[2]/strong/text()").get()
31             yield {
32                 'country_name': name,
33                 'year': year,
34                 'population': population
35             }
```

Kaydettikten sonra, runner içerisinde yukarıdan **Debug > Start Debugging>Python File** ile debug işlemini başlatabiliriz.

Sonuçta aşağıdaki gibi bir ekranla karşılaşırız, sol tarafta ilgili anda programın hangi local variable'ları tuttuğunu ve bunların değerlerini görebiliriz.



```
> allowed_domains: ['www.worldometers.info']
> name: 'countries'
> parse: <function CountriesSpider.parse_country at 0x7f8b1c1c1c1c>
> start_urls: ['https://www.worldometers.info/world-population/china-population/']
> _module_: 'worldometers.info.world-population.china-population'
> _qualname_: 'CountriesSpider.parse_country'
> WATCH
```

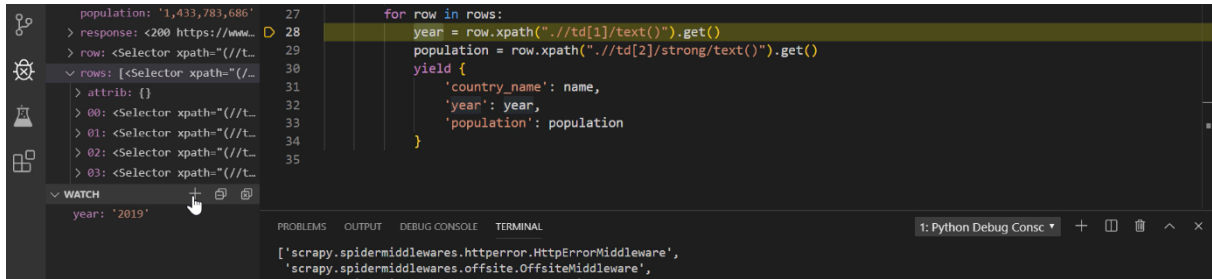
```
17         yield response.follow(url="https://www.worldometers.info/world-population/china-population/", callback=self.parse_country)
18
19     # # absolute_url = f"https://www.worldometers.info/{link}"
20     # # absolute_url = response.urljoin(link)
21
22     yield response.follow(url="https://www.worldometers.info/world-population/china-population/", callback=self.parse_country)
23
24     def parse_country(self, response):
25         name = response.request.meta['country_name']
26         rows = response.xpath(
27             "(/table[@class='table table-striped table-bordered table-hover table-condensed table-list'])[1]/tbody/tr"
28         )
29         for row in rows:
30             year = row.xpath("./td[1]/text()").get()
31             population = row.xpath("./td[2]/strong/text()").get()
32             yield {
33                 'country_name': name,
34                 'year': year,
35                 'population': population
36             }
```

```
2019-09-29 14:07:35 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2019-09-29 14:07:35 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
```

Henüz parse_country içinde değiliz, yukarıdakilerle birkaç adım ilerleyince parse_country içine girebiliriz ve burada local olarak response

objesini ve self objesini görebiliriz, ilerdikçe method içinde oluşan değişkenleri ve değerlerini de görebiliriz.

Diyelim ki bir değişkenin değerini takip etmek istiyorum: Sol alttaki watch kısmı bunun için yapılmış, mesela year değişkenini gözlemlemek istiyorum, watch kısmına year değişkeni elimle yazarak eklersem her step için bu değişkenin değeri görüntülenebilir.



Crawl Spider Structure

Intro

Şimdiye kadar bir spider yarattığımızda oluşan hazır class template'i **BASIC TEMPLATE** olarak isimlendirilir

Bu class aşağıdakine benzer bir formda:

```
class CountriesSpider(scrapy.Spider):
    name = 'countries'
    allowed_domains = ['www.worldometers.info']
    start_urls = [
        'https://www.worldometers.info/world-population/population-by-country/']

    def parse(self, response):
        countries = response.xpath("//td/a")
        for country in countries:
            name = country.xpath("../text()").get()
            link = country.xpath("../@href").get()

            # absolute_url = f"https://www.worldometers.info/{link}"
            # absolute_url = response.urljoin(link)

            yield response.follow(url=link, callback=self.parse_country, meta={'country_name': name})

    def parse_country(self, response):
        name = response.request.meta['country_name']
        rows = response.xpath(
            "(/table[@class='table table-striped table-bordered table-hover table-condensed table-list'])[1]/tbody/tr")
        for row in rows:
            year = row.xpath("../td[1]/text()").get()
            population = row.xpath("../td[2]/strong/text()").get()
            yield {
                'country_name': name,
                'year': year,
                'population': population
            }
```

Scrapy.Spider'ı inherit eden, name, allowed domains, start_urls'i olan veya start request methodu override edilmiş, en az bir parse methodu olan bildiğimiz klasik class.

```
class SpecialOffersSpider(scrapy.Spider):
    name = 'special_offers'
    allowed_domains = ['www.tinydeal.com.hk']

    def start_requests(self):
        yield scrapy.Request(url='https://www.tinydeal.com.hk/specials.html', callback=self.parse, headers={
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome'
        })

    def parse(self, response):
        for product in response.xpath("//ul[@class='productlisting-ul']/div/li"):
            yield {
                'title': product.xpath("../a[@class='p_box_title']/text()").get(),
                'url': response.urljoin(product.xpath("../a[@class='p_box_title']/@href").get()),
                'discounted_price': product.xpath("../div[@class='p_box_price']/span[1]/text()").get(),
                'original_price': product.xpath("../div[@class='p_box_price']/span[2]/text()").get(),
                'User-Agent': response.request.headers['User-Agent']
            }
```

Basic Template'ın yanında 3 farklı template daha vardır, bunları terminaleden `scrapy genspider -l` komutu ile görebiliriz:

- Basic
- Crawl
- Csvfeed
- Xmlfeed

Son ikisi'ni csv ve xml feed template'leri isminden anlaşılacağı gibi csv ve xml files'ı scrape etmek için kullanılır ama sanıyorum biz bunları kullanmayacağız.

Bu kısımda **Crawl Template**'i anlamaya çalışacağız.

Crawl Spider

Şimdi yeni bir proje yaratalım ve crawl spider'ı anlayalım.

Terminalde proje klasöründeyken:

`scrapy startproject imdb` ile imdb isimli bir proje başlatalım.

`cd imdb` ile cd'yi ayarlayalım ve aşağıdaki şekilde crawler spider'ımızı yaratalım.

`scrapy genspider -t crawl best_movies imdb.com`

name ve link dışında -t crawl ile template belirttik o kadar.

Spider'ımız oluştu, şimdi daha önce yaptığımız gibi vscode ile bu projeyi açıp spider'a ulaşabiliriz.

Oluşan crawl spider'ın orijinal hali aşağıdaki gibidir.

```
# -*- coding: utf-8 -*-
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule

class BestMoviesSpider(CrawlSpider):
    name = 'best_movies'
    allowed_domains = ['imdb.com']
    start_urls = ['http://imdb.com/']

    rules = (
        Rule(LinkExtractor(allow=r'Items/'), callback='parse_item', follow=True),
    )

    def parse_item(self, response):
        item = {}
        #item['domain_id'] = response.xpath('//input[@id="sid"]/@value').get()
        #item['name'] = response.xpath('//div[@id="name"]').get()
        #item['description'] = response.xpath('//div[@id="description"]').get()
        return item
```

Bu class'ın inherit ettiği class farklıdır, onun yanında name allowed domains ve start_urls propertyleri yine var ve bir parse methodu var.

Ayrıca rules isminde yeni bir tuple tipli property söz konusu.

Rule Objects

Tuple immutable bir yapı, yani içindekileri değiştiremiyoruz, rule attribute'u içinde en az bir Rule objesi olan bir tuple olmalı.

Bu Rule objesi spider'a hangi linkleri follow etmesi gerektiğini söyler.

Default rule objesinin içine bakarsak:

```
Rule(LinkExtractor(allow=r'Items/'), callback='parse_item', follow=True),
```

- Rule objesinin 3 farklı argument'i var.
- İlki bir LinkExtractor objesi, diğer callback methodu, üçüncüsü de bir boolean.
- Callback methodunun string olarak verildiğine dikkat et.
- follow=True demek extract edilen linkleri follow et, yani request yolla demek!
- LinkExtractor ile extract etmek istediğimiz veya istemediğimiz linkleri specify ederiz.

```
LinkExtractor(allow=r'Items/')
```

Burada allow argument'i içine bir regular expression girilmiş.

Bu hali ile Rule'un dediği şey şu: items kelimesini içeren linkleri extract et, follow et ve response'u parse_item methoduna döndür.

```
LinkExtractor(deny=r'Items/')
```

Dersek, içinde items geçen linkleri deny et yani extract etme demek.

LinkExtractor içinde Xpath expressions veya CSS Selectors ile de hangi linklerin extract ediliş edilmeyeceğini belirtebiliriz:

```
LinkExtractor(restrict_xpaths=('//a[@class="active"]'))
```

Burada söylenilen şey şu active class'ına dahil olan a elemanlarının href parametrelerindeki linki extract et. Burada xpath içinde @href diye belirtmedik ancak LinkExtractor bunu otomatik yapıyor, sadece href'leri arıyor.

Ayrıca restrict_xpaths'in bir tuple olduğunu unutma buraya birden fazla Xpath tanımlayabiliriz.

Ayrıca istersek CSS selectors da kullanabiliriz:

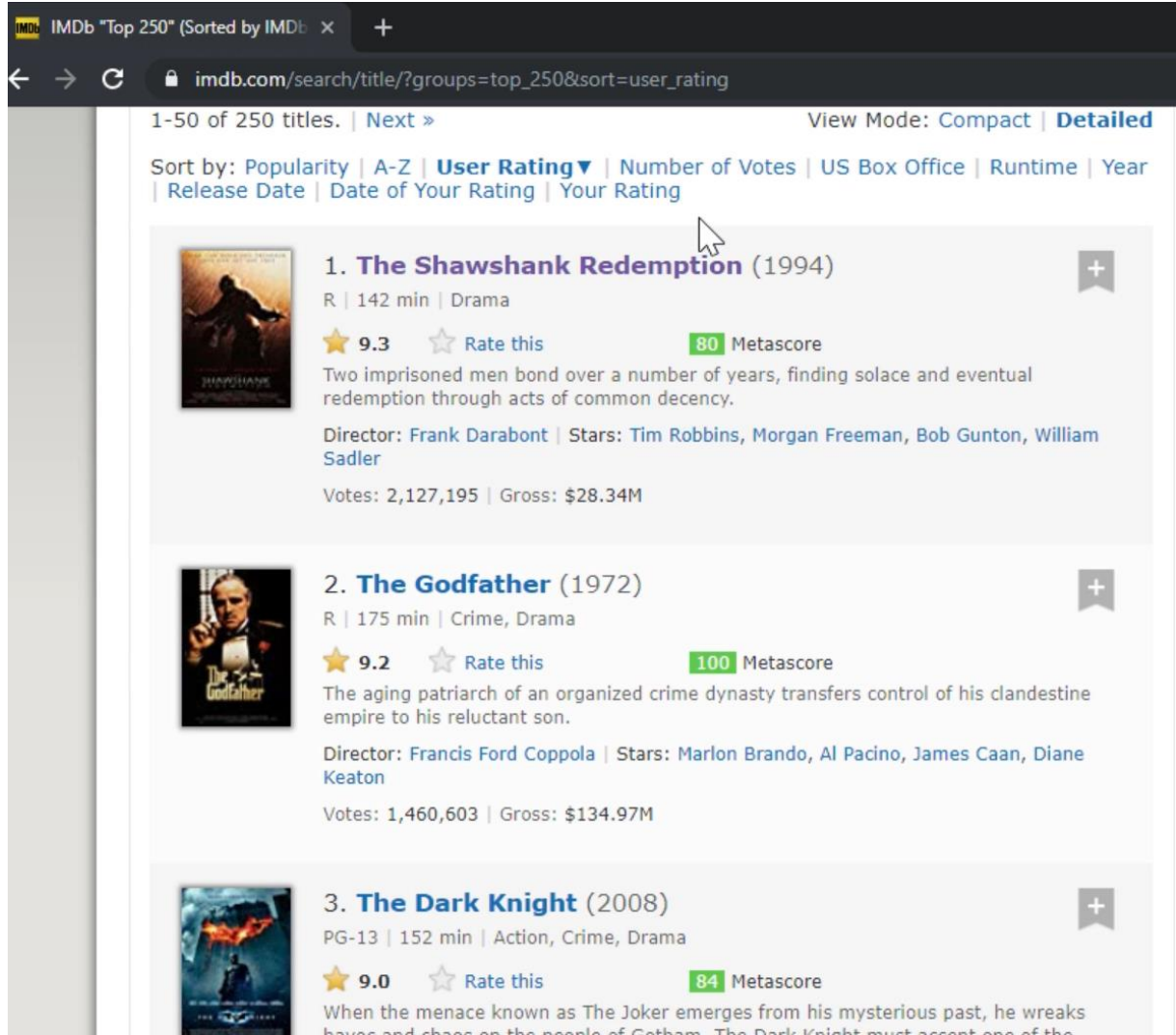
```
restrict_css=('//a[@class="active"]'))
```

Son olarak şunu söyleyelim rules tuple'ı içerisinde birden fazla Rule objesi olabilir her bir Rule objesi spesifik linklerin follow ediliş edilmemesinden sorumlu olabilir.

```
rules = (  
    Rule(LinkExtractor(restrict_css=('//a[@class="active"]')), callback='parse_item', follow=True),  
)
```

The Rule Object with IMDB Project

Amacımız IMDB Top250 sayfasına gidip, her film linkine tek tek gidip,



IMDb "Top 250" (Sorted by IMDb) x +

imdb.com/search/title/?groups=top_250&sort=user_rating

1-50 of 250 titles. | Next » View Mode: Compact | Detailed

Sort by: Popularity | A-Z | **User Rating ▼** | Number of Votes | US Box Office | Runtime | Year | Release Date | Date of Your Rating | Your Rating

- 1. The Shawshank Redemption (1994)**
R | 142 min | Drama
★ 9.3 ☆ Rate this 80 Metascore
Two imprisoned men bond over a number of years, finding solace and eventual redemption through acts of common decency.
Director: Frank Darabont | Stars: Tim Robbins, Morgan Freeman, Bob Gunton, William Sadler
Votes: 2,127,195 | Gross: \$28.34M
- 2. The Godfather (1972)**
R | 175 min | Crime, Drama
★ 9.2 ☆ Rate this 100 Metascore
The aging patriarch of an organized crime dynasty transfers control of his clandestine empire to his reluctant son.
Director: Francis Ford Coppola | Stars: Marlon Brando, Al Pacino, James Caan, Diane Keaton
Votes: 1,460,603 | Gross: \$134.97M
- 3. The Dark Knight (2008)**
PG-13 | 152 min | Action, Crime, Drama
★ 9.0 ☆ Rate this 84 Metascore
When the menace known as The Joker emerges from his mysterious past, he wreaks havoc and chaos on the people of Gotham. The Dark Knight must accept one of the

gidilen film sayfasından aşağıdaki gibi isim, puan, duration, genre gibi bilgileri çekmek.

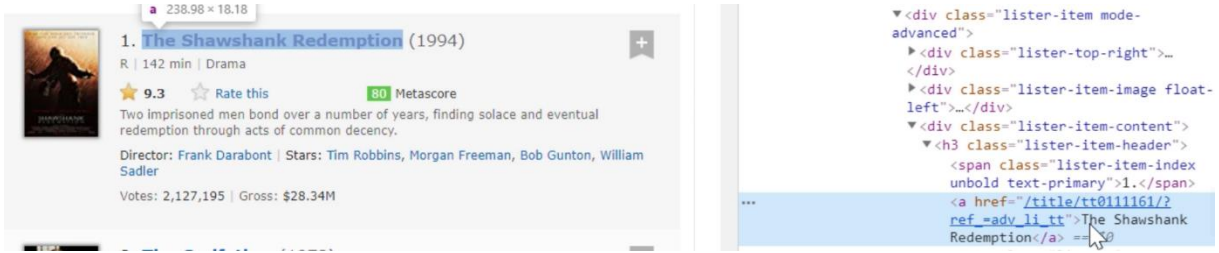


FULL CAST AND CREW | TRIVIA | USER REVIEWS | IMDbPro | MORE » SHARE

The Shawshank Redemption (1994)
R | 2h 22min | Drama | 14 October 1994 (USA)
★ 9.3 / 2,127,195 ☆ Rate This

Bunun için yapacağım şey şu olacak, öncelikle start_urls'i top250 sayfasının url'i ile değiştireceğim.

Ardından, rules içindeki tek Rule objemin LinkExtractor objesi içerisinde öyle bir xpath expression tanımlayacağım ki ilgili top250'nin ilk sayfa linkindeki tüm movie links alınsın.



Sonuçta `//h3[@class='lister-item-header']/a` ile movie linkinin href'inde gömülü olduğu a elemanını seçebiliriz.

İlk sayfada 50 film olduğu için 50 tane a elemanı yani 50 link görmeliyiz.

Şimdi bu ifadeyi Rule içine koyuyoruz, unutmama LinkExtractor zaten verilen elemanın içindeki href elemanını extract ediyor.

```
rules = (  
    Rule(LinkExtractor(restrict_xpaths=("//h3[@class='lister-item-header']/a")), callback='parse_item', follow=True)  
)
```

Tek bir xpath expression kullanılacaksa restrict_xpaths'i bir tuple olarak değil direkt tek xpath olarak da tanımlayabiliriz.

Parse_item içerisinde de sadece response'un url'ini print ettirdik.

```
class BestMoviesSpider(CrawlSpider):
    name = 'best_movies'
    allowed_domains = ['imdb.com']
    start_urls = ['https://www.imdb.com/search/title/?groups=top_250&sort=user_rating']

    rules = (
        Rule(LinkExtractor(restrict_xpaths="//h3[@class='lister-item-header']/a"), callback='parse_item', follow=True)
    )

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

Şimdi ilgili spider'ı daha önce olduğu gibi **scrapy crawl best_movies** şeklinde terminalden çalıştırırsak, ilk sayfa url'lerini göreceğiz.

Yani bu şekilde yapılan şey, ilk sayfadaki her movie link rule object ile extract edildi ve link follow edildi her follow sonrası da parse_item methodu çalıştırıldı, bunu basic spider ile de yapabildik, orada Rule objesi yerine bir başka parse methodu ile önce linkleri extract edecektik daha sonra loop ile linkleri dönüp tek tek request yollayacaktık, sonuçta sanıyorum aynı şeyi tek satırla yapmış olduk.

Şimdi istenilen movie properties'i scrape etmek için daha önce yaptığımız gibi specific movie response'u üzerinden xpath ile elemanları seçeceğiz:

Burada time elemanı başta ve sonda boşluklar ile yield ediliyordu bunun için aşağıdaki gibi **normalize-space** fonksiyonundan yararlanıldı.

```
class BestMoviesSpider(CrawlSpider):
    name = 'best_movies'
    allowed_domains = ['imdb.com']
    start_urls = ['https://www.imdb.com/search/title/?groups=top_250&sort=user_rating']

    rules = (
        Rule(LinkExtractor(restrict_xpaths="//h3[@class='list-item-header']/a"), callback='parse_item', follow=True)
    )

    def parse_item(self, response):
        yield {
            'title': response.xpath("//div[@class='title_wrapper']/h1/text()").get(),
            'year': response.xpath("//span[@id='titleYear']/a/text()").get(),
            'duration': response.xpath("normalize-space(//time[1]/text())").get(),
            'genre': response.xpath("//div[@class='subtext']/a[1]/text()").get(),
            'rating': response.xpath("//span[@itemprop='ratingValue']/text()").get(),
            'movie_url': response.url,
        }
```

Son olarak, title için **unicode encoding** problemi ile karşılaşabiliriz, bunu çözmek için settings.py'ye aşağıdaki satırı ekliyoruz bundan daha önce de bahsetmiştik.

```
FEED_EXPORT_ENCODING = 'utf-8'
```


Yukarıdaki kodla, spider'ımız ilk sayfadaki tüm linkleri extract etti, her linki follow etti, ve her follow sonrası parse_item içerisinde belirtilen özellikler scrape edildi.

Şimdi pagination'ı halledeceğiz.

Pagination

Crawl spider ile pagination'ı da çözelim, bunu yapmak için bize gereken şey yeni bir rule'dan fazlası değil.

Öncelikle top250 ilk sayfasında yani ana request sayfasındaki next butonunu keşfediyoruz, ve bu next linkini elde edecek XPath'i elde ediyoruz.



50. **The Great Dictator** (1940)
Passed | 125 min | Comedy, Drama, War
★ 8.5 ☆ Rate this
Dictator Adenoid Hynkel tries to expand his empire while a poor Jewish barber tries to avoid persecution from Hynkel's regime.
Director: Charles Chaplin | Stars: Charles Chaplin, Paulette Goddard, Jack Oakie, Reginald Gardiner
Votes: 183,834

1-50 of 250 titles. | [Next](#)

```
<div class="listner-item mode-advanced">...</div>
<div class="listner-item mode-advanced">...</div>
<div class="listner-item mode-advanced">...</div>
<div class="listner-item mode-advanced">...</div>
<div class="listner-item mode-advanced">...</div>
<div class="listner-item mode-advanced">...</div>
<div class="listner-item mode-advanced">...</div>
</div>
<br class="clear">
<div class="desc">
  <span>1-50 of 250 titles.</span>
  <span class="ghost">|</span>
  <a href="/search/title/?groups=top_250&sort=user_rating,desc&start=51&ref_=adv_nxt" class="listner-page-next next-page">Next </a> == $0
</div>
```

... #main div div a.listner-page-next.next-page

@class='listner-page-next next-page'] [2] 1 of 1 Cancel

Daha sonra spider'a dönüp bu xpath expression'ı yeni oluşturulan rule içinde kullanacağız:

```
rules = (
    Rule(LinkExtractor(restrict_xpaths="//h3[@class='list-item-header']/a"), callback='parse_item', follow=True),
    Rule(LinkExtractor(restrict_xpaths="//a[@class='list-item-page-next next-page']")[2])
)

def parse_item(self, response):
    yield {
        'title': response.xpath("//div[@class='title_wrapper']/h1/text()").get(),
        'year': response.xpath("//span[@id='titleYear']/a/text()").get(),
        'duration': response.xpath("normalize-space(//time[1]/text())").get(),
        'genre': response.xpath("//div[@class='subtext']/a[1]/text()").get(),
        'rating': response.xpath("//span[@itemprop='ratingValue']/text()").get(),
        'movie_url': response.url,
    }
```

Sonuçta spider'ın yaptığı şey şu, önce ilk rule ile belirlenen, linkleri extract ediyor, her birini follow edip callback'i çağırıyor yani ilk sayfadaki her film linkine gidip properties'i çekiyor,

Daha sonra ilk sayfa tamamlanınca rule 2 execute ediliyor, rule 2'nin follow'u default olarak true yani rule 2 ile yapılan şey next linkini follow etmek, rule2 execute edilince tekrar rule1'e geri dönülüyor, ve yeni sayfadaki tüm linklerle işlem yapılıyor her link için callback çağırılıyor vesaire.

Yani rule içindeki LinkExtractor linkleri extract ediyor, daha sonra bu linkler sırasıyla follow edilip belirtilen callback çağırılıyor, bu liste tamamlanınca eğer başka rule yoksa, çalışma sonlanıyor, eğer varsa sıradaki rule'a geçiliyor, eğer son rule execute edilmiş ve tamamlanmışsa başa dönülüyor taa ki extract edilecek link kalmayana kadar.?

Merak ettiğim şey şu: tek rule varken, bu rule tekrar tekrar çalışıyor mu? Yoksa tekrarlama olayı yalnızca birden fazla rule varsa mı oluyor? Ayrıca next butonu bulunamayınca ne oluyor?

Spoofing Request Headers

Bu başlığı daha önce görmüştük, request headers'ı değiştirerek, siteye giriş yaparken banlanmayı engelleyebiliriz demiştik.

Bunun için bazı yöntemler göstermiştik. Bu yöntemlerin iki tanesinde settings.py içinde değişiklik yaparak user-agent header'ını değiştirmiştik, diğesinde de spider içinde start_requests() methodunu override etmiştik:

İlk method:

```
# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'tinydeal (+http://www.yourdomain.com)'
```

İkinci method:

```
40
41 # Override the default request headers:
42 DEFAULT_REQUEST_HEADERS = {
43     'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
44     'Accept-Language': 'en',
45 }
46
```

```
# Override the default request headers:
DEFAULT_REQUEST_HEADERS = {
    'User-Agent': ''
}
```

Son method:

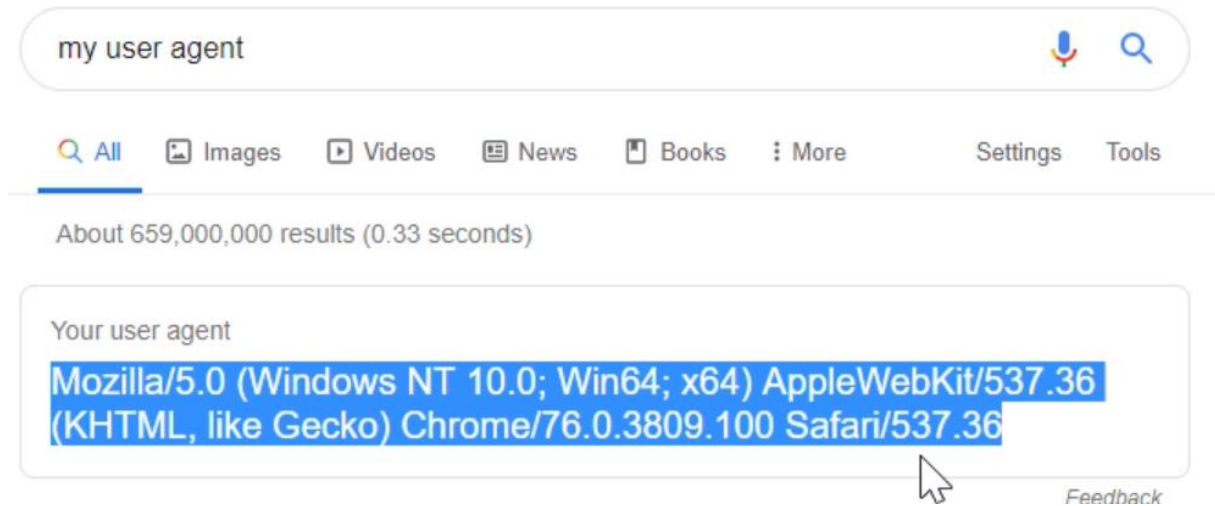
```
class SpecialOffersSpider(scrapy.Spider):
    name = 'special_offers'
    allowed_domains = ['www.tinydeal.com.hk']
    start_urls = ['https://www.tinydeal.com.hk/specials.html']

    def start_requests(self):
        yield scrapy.Request(url='https://www.tinydeal.com.hk/specials.html', callback=self.parse, headers={
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
        })

    def parse(self, response):
        for product in response.xpath("//ul[@class='productlisting-ul']/div/li"):
            yield f
```

İlk iki method'u crawl spider için de kullanabiliriz, ancak sonuncu method'u crawl spider için nasıl kullanacağımızı göreceğiz:

-Öncelikle google'a şunu yazıp çıkan user-agent value'yu kopyalıyoruz:



-Ardından spider içerisinde user_agent değişkeni yaratıp değerini yukarıdaki kopyalanan değer olarak atıyoruz:

```
user_agent = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36'

def start_requests(self):
    yield scrapy.Request(url='https://www.imdb.com/search/title/?groups=top_250&sort=user_rating'; headers={
        'User-Agent': self.user_agent
    })
```

-Bu önceki ile neredeyse birebir aynı, sadece callback tanımlamadık, çünkü önceden ilk request main url'ye gönderilecek response'un hangi parse'a verileceği önemliydi, crawl spider için ise, response direkt rule'lara veriliyor anladığım kadarıyla onun için callback tanımlamadık.

-Sonuçta yukarıdaki değişiklik ile start_urls attribute'unu silebiliriz, ve main request burada belirtilen adrese belirtilen headers ile gönderilecek.

Main request istediğimiz header ile gönderilmiş olsa da, response alınacak ve rule'lar response üzerinde çalışmaya başlayacak. Bu rule'lar içinde gönderilen request'lerin header'ları hala eskisi gibi duruyor şimdi de bu header'ları override edeceğiz:

Bu işlem için Rule objesi içine bir `process_request` argümanı tanımlıyoruz ve bu argümana bir callback method atıyoruz, bu methodu da yine kendimiz tanımlayacağız:

```
restrict_xpaths="//h3[@class='lister-item-header']/a"), callback='parse_item', follow=True, process_request='set_user_agent'),  
restrict_xpaths="//a[@class='lister-page-next next-page']][2]"))
```

Elbette bu `process_request` argümanını her rule için tanımlayacağım, daha sonra `set_user_agent` methodunu da aşağıdaki gibi tanımlayabilirim.

```
rules = (  
    Rule(LinkExtractor(restrict_xpaths="//h3[@class='lister-item-header']/a"), callback='parse_item', follow=True),  
    Rule(LinkExtractor(restrict_xpaths="//a[@class='lister-page-next next-page']][2]"), process_request='set_user_agent'),  
)  
  
def set_user_agent(self, request):  
    request.headers['User-Agent'] = self.user_agent  
    return request
```

Artık, spider içinden gönderilen main request ve rule'lar içinden gönderilen request'lerin hepsi için user-agent header'ı istediğimiz gibi modifiye edilmiş oldu.