7.26APScheduler

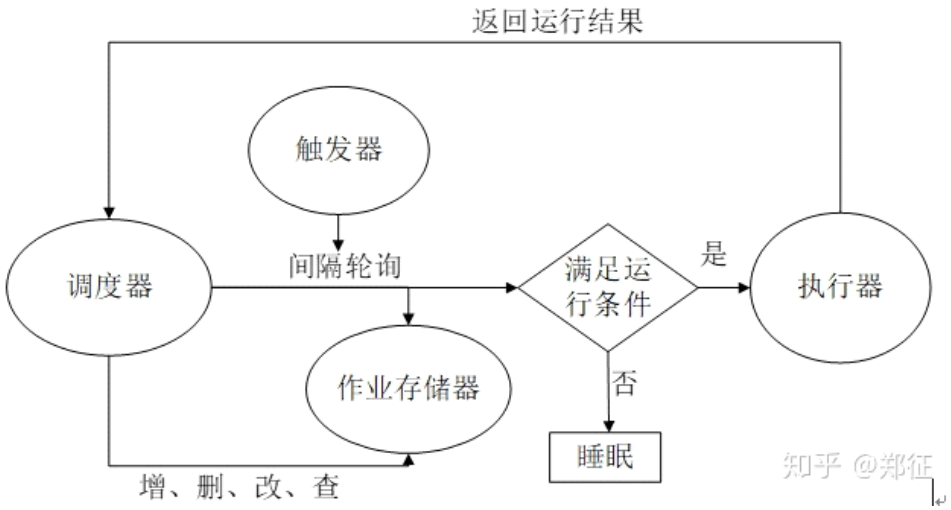
2020年7月26日　23:16

学习资源

知乎专栏：https://zhuanlan.zhihu.com/p/46948464

归纳

APScheduler总的来说是一个轻量级的定时任务框架，其基本组成部分调度器（Scheduler）触发器（Trigger）作业存储器（JobStore）执行器（Executor）。他们都基本关系与运作流程如下图所示。我认为这其中的逻辑关系还是十分清楚的。



调度器的初始化完成后完成作业存储器内容的装载。调度器开始运行后其主循环不断询问各个作业存储器是否有需要执行的任务有则提交至执行器按时间点执行。（这是比较粗略的理解，细节还请见上文专栏链接）

我们需要关注的内容以及需要考虑的内容主要是以下几点：

- 调度器的配置
- 作业存储器的存储模式（DB）
- 事件监听

详情见上文专栏链接。

带有注释的源代码

```
from apscheduler.events import EVENT_JOB_EXECUTED, EVENT_JOB_ERROR
from apscheduler.schedulers.blocking import BlockingScheduler
import datetime
from apscheduler.jobstores.memory import MemoryJobStore
from apscheduler.executors.pool import ThreadPoolExecutor, ProcessPoolExecutor
import logging

# 日志
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    filename='log1.txt',
                    filemode='a')

# 任务
def my_job(id, num):
    print(num, id, '-->', datetime.datetime.now())


# 报错任务
def error_job(id):
    print(id, '-->', datetime.datetime.now())
```

```
            print(1 / 0)


        # 监听事件
        def my_listener(event):
            if event.exception:
                print("jobs error")
            else:
                print("normal")


        # 配置调度器
        jobstores = {
            'default': MemoryJobStore()
        }
        executors = {
            'default': ThreadPoolExecutor(20),
            'processpool': ProcessPoolExecutor(10)
        }
        job_defaults = {
            'coalesce': False,
            'max_instances': 3
        }

        # 根据配置初始化调度器
        scheduler = BlockingScheduler(jobstores=jobstores, executors=executors, job_defaults=job_defaults)
        # 亦可如下省略配置
        # scheduler = BlockingScheduler()

        # 往作业存储器里添加作业
        # 周期触发器
        scheduler.add_job(func=my_job, args=['interval', '1', ], id='job_interval', trigger='interval', seconds=3,
                          replace_existing=True)
        # cron任务，灵活
        scheduler.add_job(my_job, args=['cron', '2', ], id='job_cron', trigger='cron', month='4-8,11-12', hour='7-20',
                          second='*/10',
                          end_date='2020-07-30')
        # 未指定触发器，启动时触发
        scheduler.add_job(my_job, args=['once_now', '3', ], id='job_once_now')
        # 日期触发器
        scheduler.add_job(my_job, args=['date_once', '4', ], id='job_date_once', trigger='date', run_date='2020-07-26 18:23:00')
        # 报错
        scheduler.add_job(func=error_job, args=('error',), id='job_error',
                          next_run_time=datetime.datetime.now() + datetime.timedelta(seconds=5))
        # 添加监听器
        scheduler.add_listener(my_listener, EVENT_JOB_EXECUTED | EVENT_JOB_ERROR)
```

7.27request+pyquery+selenium（Demo的运行尝试与过程理解）

2020年7月27日　21:41

遇到的问题与解决

- Selenium无法连接到浏览器问题

  报错：`[WinError 10061] 由于目标计算机积极拒绝,无法连接`

  网络上有很多解释，有的说是浏览器设置的局域网代理问题等等，总的来说就是程序无法连接到目标浏览器。我的错误原因是因为WebDriver安装错误的问题，简单致命。重新下载正确的Driver并通过设置路径的方式解决了该问题。

- WebDriverWait错误&页面加载不全

  最开始的情况，采用的是`Edge`的WebDriver。当程序运行到`WebDriverWait`时程序便停滞不前。在除去这一块再次调试时发现有页面加载不完全的情况。

  后面有组员@金尝试了`PhantomJS`作为`WebDriver`来进行下一步操作，从结果上来看是成功了，具体原因现在尚不明了。

- buffer问题

  报错：`write() argument must be str, not bytes`

  在给出的demo源码中，image的输出似乎经由了`cStringIO`这个包来进行。但是经了解，这个包适用于Python 3以下的情况。在我的`Python3.8`环境中并不兼容，因此将它删去并采用PIL更为直接的图片保存方式就成功了。

- PIL包图片保存问题

  在通过`PIL`包对爬取的网页快照截取时，发现最后的保存操作并不能正确执行

  ```
  im.save('sc.png', format="PNG")
  ```

  后改为

  ```
  im.save('C:/Github/summer.apsbug/images/b.png')
  ```

  成功执行，目前原因尚不明了，但是组内成员能够成功运行代码，估计是通过pip装载的包本身存在问题。

Request

学习链接：

https://blog.csdn.net/shanzhizi/article/details/50903748

目前的理解：

`Request`向目标服务器发送各类请求（`POST`，`GET`...）目标服务器向我们返回`Response`（信息源），我们获取`Response`。就目前来看似乎是比较简单的机制。

适合抓取静态页面，在面对动态渲染的页面（如本次demo）时往往需要借助`Selenium`。

Selenium

学习链接：

https://zhuanlan.zhihu.com/p/111859925

目前的理解:

Selenium模拟了浏览器访问服务器的过程（自动化），此举能够让我获得动态页面经渲染后的页面，配置WebDriver的过程比较坑爹。其余过程目前来说给我感受与Request相差不大，后续有新的理解再进行添加。

PyQuery

学习链接:
https://www.cnblogs.com/lei0213/p/7676254.html
目前的理解：
用于对捕获的信息源（往往是HTML网页）进行解析，以便于对页面内元素进行操作。其语法与JQuery十分类似。需要加以记忆的是CSS选择器的选择规则，在使用感受上比Selenium的获取元素的途径上要简洁许多。

总结

在学习一块知识时，我比较注重一方面知识的完整性。它们之间是如何协调工作的，它们各自存在的意义是什么，它们负责的功能的输入是什么，输出是什么。
就这一块内容的学习而言，现在我的理解是，
Request与Selenium负责了通过指定域名（服务器地址）进行对网络资源的获取，而它们各自的应用场景有所不同，而PyQuery更专注于对获取的资源进行解析以获取我们需要的部分（WebDriver与WebElement的使用）。这时让我们重新考虑爬虫的功能要求，可以发现通过这些提供基础工具的包（当然也有别的包也能够胜任这些工作）我们可以实现一个网络爬虫应具有的基础功能。
我们解决了最基本的爬虫的设计的问题。而在面对真正的功能需求时，我们往往需要对大量的网络爬虫进行高效的配置与管理，也就是接下来引入框架例如Scrapy与通过DB与Shell进行辅助控制的目的。

7.27jobstores配置DB&Shell控制

2020年7月27日　21:40

jobstores进行DB存储配置

为了实现Job_Store的持久化存储，进行DataBase数据库配置（采用MySQL）。
配置完成并成功与数据库建立连接后，jobs作业就会被保存在数据库中，第二次运行程序时会自动地从数据库jobstore中读取作业信息并运作。

jobstores配置代码

```
jobstores = {
        'default': SQLAlchemyJobStore(url='mysql+pymysql://root:Ji1234@101.132.131.184/apscheduler?charset'
                                                                                    '=utf8', tablename='api_job')
}
```

遇到的问题以及解决

- 连接数据库出现 No module named 'MySQLdb'

  MySQL-Python不支持Python3引起的错误。

  执行 `pip install PyMySQL`,

  将数据库连接改为 `mysql+pymysql://username:password@server/db`

Shell

认识

Shell 是一个用 C 语言编写的程序，它是用户使用 Linux 的桥梁。Shell 既是一种命令语言，又是一种程序设计语言。
Shell 是指一种应用程序，这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务。

来自 <https://www.runoob.com/linux/linux-shell.html>
在通过一系列的文档查阅后，在我的理解下，Shell是一种相对于我们编写的程序来说更为'高级'的一种存在，它负责对各程序的调度与命令，而不受程序语言差别的约束。因此可以对编写完成的各功能进行配置化管理。

但是就目前而言，对这块缺乏系统的认识，只是对之后应用场景有个十分模糊的概念。

问题

Shell常用于Linux系统。那么在Windows系统下要如何编写使用Shell脚本呢?
学习链接:
https://blog.csdn.net/M__L__/article/details/92066132
即通过安装Git Bash来控制解释运行Shell脚本。

几条简单的尝试，尚未进行解决问题程度的探索


28345@DESKTOP-61GOMIQ MINGW64 ~/Desktop/test
$ touch 2.sh
28345@DESKTOP-61GOMIQ MINGW64 ~/Desktop/test
$ echo “"Hello World"
□□Hello World
28345@DESKTOP-61GOMIQ MINGW64 ~/Desktop/test
$ sh 2.sh
28345@DESKTOP-61GOMIQ MINGW64 ~/Desktop/test
$ sh 1.sh
1111

7.28APScheduler&爬虫自动化

2020年7月29日　14:22

源代码

test.py

```
from apscheduler import job
from apscheduler.events import EVENT_JOB_EXECUTED, EVENT_JOB_ERROR
from apscheduler.jobstores.sqlalchemy import SQLAlchemyJobStore
from apscheduler.schedulers.blocking import BlockingScheduler
import datetime
from apscheduler.jobstores.memory import MemoryJobStore
from apscheduler.executors.pool import ThreadPoolExecutor, ProcessPoolExecutor
import logging
import spider

# 日志
logging.basicConfig(level=logging.INFO,
                                        format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s',
                                        datefmt='%Y-%m-%d %H:%M:%S',
                                        filename='log1.txt',
                                        filemode='a')



# 任务
def my_job(id, num):
        pass
        # print(num, id, '-->', datetime.datetime.now())


# 报错任务
def error_job(id):
        # print(id, '-->', datetime.datetime.now())
        print(1 / 0)


# 监听事件
def my_listener(event):
        if event.exception:
                print("jobs error")
        else:
                print("normal")


# 配置调度器
jobstores = {
        'default': SQLAlchemyJobStore(url='mysql+pymysql://root:Ji1234@101.132.131.184/apscheduler?charset'
                                                                                '=utf8', tablename='api_job')
}
executors = {
        'default': ThreadPoolExecutor(20),
        'processpool': ProcessPoolExecutor(10)
}
job_defaults = {
        'coalesce': False,
        'max_instances': 3
}

if __name__ == "__main__":
        scheduler = BlockingScheduler(jobstores=jobstores, executors=executors, job_defaults=job_defaults)
        # 添加监听器
        scheduler.add_listener(my_listener, EVENT_JOB_EXECUTED | EVENT_JOB_ERROR)
        # 日志打印
        scheduler._logger = logging
```

```
60
61          scheduler.add_job(func=spider.update, id='spider', trigger='interval', seconds=3,
62                                              replace_existing=True)
63          # 启动调度器
64          scheduler.start
```

spider.py

```
1  from selenium.webdriver.edge.service import Service
2  from selenium import webdriver
3
4
5  def update():
6          print("spider update")
7          service = Service('C:/Users/28345/AppData/Local/Programs/Python/Python38/Scripts/msedgedriver.exe')
8          service.start()
9          driver = webdriver.Remote(service.service_url)
10         driver.get('http://weather.sina.com.cn')
11         data = driver.find_elements_by_class_name('slider_degree')[0]
12         print(data.text)
13         driver.quit()
```

运行结果

控制台打印

```
normal
spider update
normal
33℃
spider update
normal
normal
normal
33℃
spider update
normal
```

数据库状态

	id	next_run_time	job_state	
☐	spider	1596004088.711411	(Binary/Image)	475B
☐	job_interval	1596004088.186387	(Binary/Image)	499B
☐	job_cron	1596004090	(Binary/Image)	1K
*	*(NULL)*	*(NULL)*	*(NULL)*	0K

问题

似乎没有遇到有同学碰到过的线程问题。

从控制台打印的结果来看，`APScheduler`这个框架本身的作用调用应该就是多线程（或是异步？）的。

7.28Scrapy框架

2020年7月29日　0:23

Scrapy

学习链接

具体的操作步骤教程给的很详细，这里不再赘述。
https://www.runoob.com/w3cnote/scrapy-detail.html

必要性

在之前，我们似乎通过`request+pyquery+selenium`的协作使用就已经完成了我们期待的爬虫的一切功能。
但是在之后进行真正的项目开发时，我们面对的并不是单独的爬虫需求，而可能是需要同时对多个服务器进行信息抓取并生成结构性数据。而框架的存在为我们省去了重复造轮子的麻烦，并为我们提供了简单高效的开发手段。
而且经过简单地学习之后，发现该框架提供的功能确实简洁强大。

安装框架

```
pip install Scrapy
```

这期间可能会遇到下载失败的情况。
可行的方案：

- 重新安装
- 挂梯子

遇到的问题以及解决

- `scrapy 'ItcastItem does not support field: name'`
  链接：https://blog.csdn.net/beyond_f4/article/details/79745954
  检查`Items`中对应的结构类的`Field`是否完整正确地定义了。

```
name = each.xpath("h3/text()").extract()
title = each.xpath("h4/text()").extract()
info = each.xpath("p/text()").extract()

# xpath返回的是包含一个元素的列表
item['name'] = name[0]
item['title'] = title[0]
item['info'] = info[0]
```

该处获取了`item`的`name`，`title`，`info`信息。
那么在`item`中也应对应的有`name`，`title`，`info`域

```
class ItcastItem(scrapy.Item):
    name = scrapy.Field()
    title = scrapy.Field()
    info = scrapy.Field()
```

7.29第一阶段学习总结

2020年7月29日 14:20

- 环境搭建Anaconda&Pycharm
  - Anaconda安装与环境配置
  - Pycharm安装与项目配置
- 任务调度框架APScheduler
  - 基本框架理解
  - 多种调度模式cron,interval...
  - DB持久化jobstores
  - shell脚本控制
- 实现简单爬虫功能Request+Pyquery+Selenium
  - Request获取静态页面（Response）
  - Selenium模拟浏览器浏览页面获取动态渲染页面信息（WebDriver）
  - Pyquery解析获取的数据以获得目标数据（CSS选择器）
- Scrapy框架
  - 完成Scrapy框架的安装
  - Scrapy爬虫制作流程
  - Scrapy+APScheduler定时爬虫运行
  - Scrapy+Selenium动态页面爬取
- 其他（数据处理）
  - Image处理图片
  - 正则表达式筛选数据

8.10Scrapy基本配置要素实现demo

2020年8月9日　　22:53

Github

https://github.com/vanot313/scrapy.demo

简述

并没有深入源码了解，仅从功能理解上谈述。

`Setting.py`配置各类基础信息，其生成的源项目中有详细的备注，另外可以自己增加配置信息在各组件中基本通过`crawler.settings.get`可以获取相应参数。

`Items.py`配置需要的数据信息格式，类似于`modle`之于数据库，可以简单理解为数据库信息在项目中的模具，获取到相应信息后传回数据实例保存至数据库。

`Piplines.py`配置'管道'，在功能上这是很形象的描述，在`items`实体被传回后经过'管道'，进行加工处理或者是将其存入数据库或是本地文件（持久化）。

`Middlewares.py`配置中间下载器，配合`Selenium`实现对动态渲染页面的抓取。

`Spiders`爬虫对`middleware`中获取的网页数据进行进一步的业务处理，以获取需要的信息并生成`items`实例再投递至`piplines`管道。

配置与实现清单

- `MySQL`的连接配置
- `WebDriver`的路径配置
- `WebDriver`的参数配置
- `piplines`本地存储与数据库存储模块
- `middlewares`配置`Selenium`动态页面访问模块
- `items`数据实体
- `spiders`爬虫业务逻辑实现

实现功能

实现的是一个整体流程都经过的Scrapy工程模板，集成了基础selenium。

可以基本实现对动态页面的数据爬取与持久化。

待实现功能

对具体网页进行具体操作的爬虫业务逻辑。例如对分页页面的爬取，对反爬取页面爬取的反制手段。

遇到的问题以及解决

9/13：闲置过久。待下次学习重启编写。

8.12selenium处理request时的问题

2020年8月12日 23:35

1.selenium 不支持 post 请求方式
2.selenium 不支持自定义 request header

解决

见Scrapy ajax post解决

但事实上并没有正面解决这两个问题，而是采取了别的方案实现预期效果

Scrapy ajax post解决

2020年8月25日　　23:45

Github

https://github.com/vanot313/scrapy.demo

// TODO 问题与分析

9/13：这块闲置时间有点久了，出于学习效率，估计下次再来补充需要到再次用到这方面知识的时候。不过这块源代码写的比较详细，并且错误情况与解决方案有做记录，有空可以把这块补上。

Scrapy+Scheduler定时爬虫任务问题

2020年8月27日　　12:44

- ValueError : Signal only works in main thread
  似乎是cmd命令只能在主线程运行的原因，那么除了cmd直接触发的方式还有什么方法可以控制爬虫的运行呢?

  解决方案：Scheduler+Flask

  Flask尚未学习

  9/13：这块闲置时间有点久了，出于学习效率，估计下次再来补充需要到再次用到这方面知识的时候。