# 课程介绍

* 多线程爬虫（掌握）
* 动态HTML页面数据获取（掌握）
* Selenium（掌握）
* PhantomJS（掌握）

1. 为什么需要多线程

urllib.urlopen()、requests.get()、requests.post()都是同步网络请求，也就是说，这些操作在收到对应url的响应将其封装为响应结果对象返回或者访问超时后，才会继续执行后续的代码，发生了“阻塞”。

这种阻塞是没有必要和效率低下的，就像网购下单后一定得等到货物送到签收后，才允许再次下单。

有比如我们通过网购买东西，我们下单后并不会就一直坐着等快递到，而是继续该干啥干啥；也不会就要等到快递收到后才下单买其他东西，而是可能同时下了多个订单。而这些订单都是并发处理的，具体商品到达的顺序和订单发出的顺序往往并不一致。最后，快递小哥一个电话，我们就高高兴兴去收快递了。

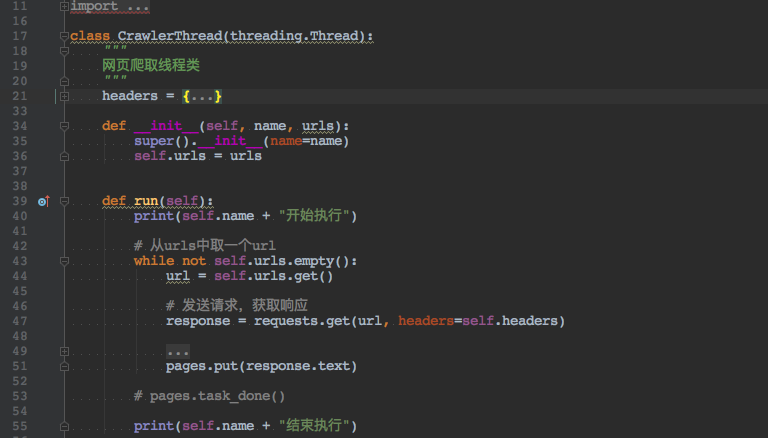
更好的做法：同时发多个请求，同时进行多种处理。

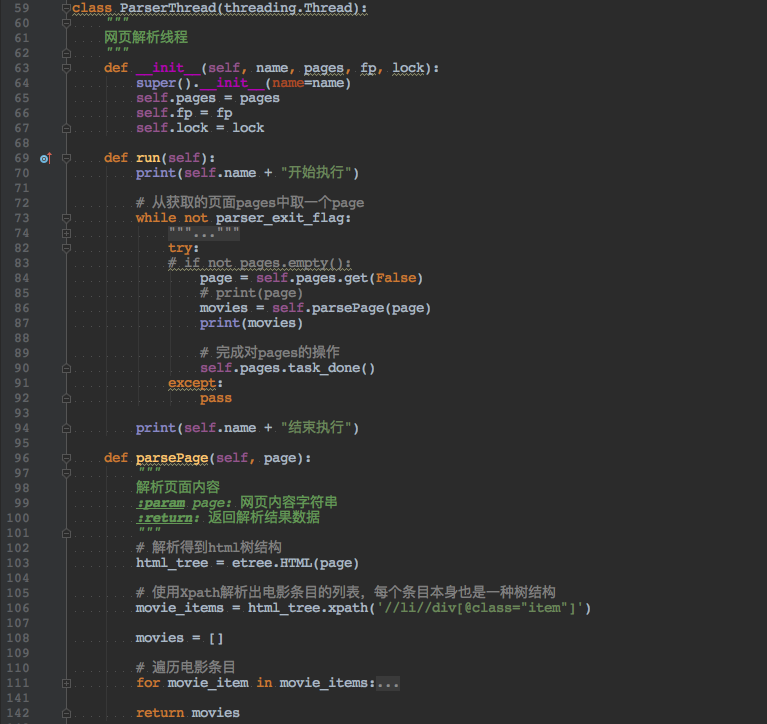
1. python多线程回顾
   1. Queue（队列对象）

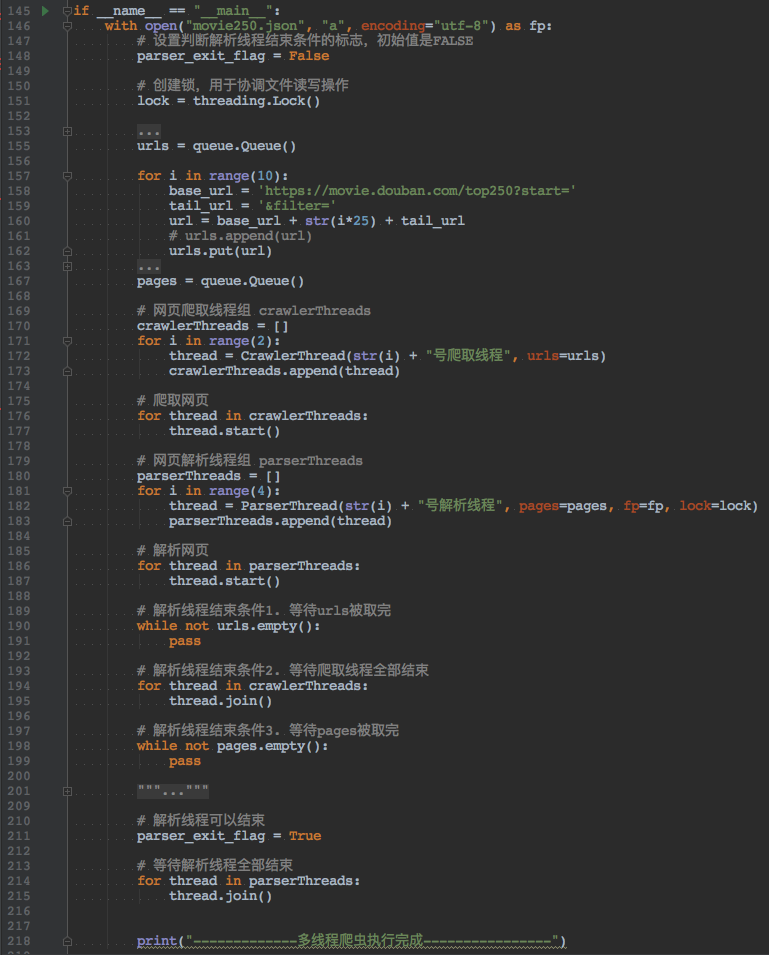
Queue是python中的标准库，可以直接import Queue引用;队列是线程间最常用的交换数据的形式

对于资源，加锁是个重要的环节。因为python原生的list,dict等，都是not thread safe的。而Queue，是线程安全的，因此在满足使用条件下，建议使用队列

1. 初始化： class Queue.Queue(maxsize)，遵循FIFO（先进先出）
2. 包中的其他常用方法:
   * Queue.qsize() 返回队列的大小
   * Queue.empty() 如果队列为空，返回True,反之False
   * Queue.full() 如果队列满了，返回True,反之False，与 maxsize对应
   * Queue.get([block[, timeout]])获取队列，timeout等待时间
3. 创建队列对象
   * import Queue
   * myqueue = Queue.Queue(maxsize = 10)
4. 将一个值放入队列中（总是队尾）
   * myqueue.put(3.14)
5. 将一个值从队列中取出（总是队首）
   * myqueue.get()
   1. 多线程爬虫案例：爬取豆瓣电影







1. 动态HTML页面的数据获取

我们目前爬取的网站页面多是静态页面，页面内容和时间等状态没有密切关系。

而还有一大类网站页面，其内容和时间息息相关，比如：各种股票网站、今日头条等网站会产生动态HTML页面。如果直接用爬虫爬取HTML文件，实际上只是爬取到了产生数据的代码，而绝大多数数据是通过这些代码动态产生的。

* 1. 常见动态页面分类
     1. JavaScript

JavaScript 是网络上最常用也是支持者最多的客户端脚本语言。它可以收集 用户的跟踪数据,不需要重载页面直接提交表单，在页面嵌入多媒体文件，甚至运行网页游戏。

我们可以在网页源代码的<scripy>标签里看到，比如：

<script type="text/javascript" src="https://statics.huxiu.com/w/mini/static\_2015/js/sea.js?v=201601150944"></script>

* + 1. jQuery

jQuery 是一个十分常见的库,70% 最流行的网站(约 200 万)和约 30% 的其他网站(约 2 亿)都在使用。一个网站使用 jQuery 的特征,就是源代码里包含了 jQuery 入口,比如:

<script type="text/javascript" src="https://statics.huxiu.com/w/mini/static\_2015/js/jquery-1.11.1.min.js?v=201512181512"></script>

如果一个网站网页源码中出现了jQuery，那么采集这个网站数据的时候要格外小心。因为jQuery可以动态地创建HTML内容,这些内容只有在JavaScript代码执行之后才会产生和显示。如果使用传统的方法采集页面内容,就只能获得JavaScript代码执行之前页面上的内容。

* + 1. Ajax

我们与网站服务器通信的唯一方式，就是发出HTTP/HTTPS请求获取新页面。如果提交表单或从服务器获取信息之后，网站的页面不需要刷新，那么该网站就使用了Ajax 技术。

Ajax其实并不是一门语言,而是用来完成网络任务(可以认为 它与网络数据采集差不多)的一系列技术。Ajax（Asynchronous JavaScript and XML，异步JavaScript和XML)，网站不需要使用单独的页面请求就可以和网络服务器进行交互。

* + 1. DHTML

类似Ajax，动态 HTML(Dynamic HTML, DHTML)也是一系列用于解决网络问题的 技术集合。DHTML 使用客户端语言改变页面的 HTML 元素(HTML或者CSS或者二者同时)。比如页面上的按钮只有当用户移动鼠标之后才出现,背景色可能每次点击都会改变，或者用一个Ajax请求触发页面加载一段新内容，网页是否属于DHTML，关键要看有没有用JavaScript控制HTML和CSS元素。

总之，眼睛看到的不一定是html代码中存在的数据，有些数据是通过javascipt代码动态生成的。

* 1. 对应处理方法

那些使用了 Ajax 或 DHTML 技术改变 / 加载内容的页面，可能有一些采集手段。但是用 Python 解决这个问题只有两种途径:

* + - 1. 直接从 JavaScript 代码里采集内容，费时费力
      2. 用 Python 的 第三方库运行 JavaScript，直接采集你在浏览器里看到的页面。

既然浏览器能拿到数据，那么，我们可以模拟一个浏览器，从浏览器中拿到数据。

1. Selenium与PhantomJS
   1. Selenium

[Selenium](http://www.seleniumhq.org/)是一个Web的自动化测试工具，最初是为网站自动化测试而开发的，就像我们玩游戏用的按键精灵，可以按指定的命令自动操作，不同是Selenium 可以直接运行在浏览器上，它支持所有主流的浏览器（包括PhantomJS这些无界面的浏览器）。

Selenium 可以根据我们的指令，让浏览器自动加载页面，获取需要的数据，甚至页面截屏，或者判断网站上某些动作是否发生。

Selenium 自己不带浏览器，不支持浏览器的功能，它需要与第三方浏览器结合在一起才能使用。但是我们有时候需要让它内嵌在代码中运行，所以我们可以用一个叫 PhantomJS 的工具代替真实的浏览器。

推荐使用pip安装：pip install selenium

注意：由于最新的selenium取消了PhantomJS的支持，所以我们需要安装一个早期版本，推荐安装2.47.3:

pip install selenium==2.47.3

Selenium 官方参考文档：<http://selenium-python.readthedocs.io/index.html>

* 1. PhantomJS

[PhantomJS](http://phantomjs.org/) 是一个基于Webkit的“无界面”(headless)浏览器，它会把网站加载到内存并执行页面上的 JavaScript，因为不会展示图形界面，所以运行起来比完整的浏览器要高效。

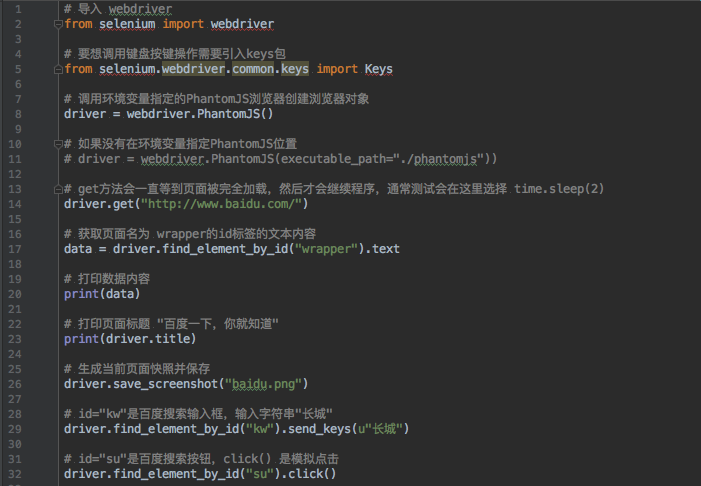
如果我们把 Selenium 和 PhantomJS 结合在一起，就可以运行一个非常强大的网络爬虫了，这个爬虫可以处理 JavaScrip、Cookie、headers，以及任何我们真实用户需要做的事情。

注意：PhantomJS 只能从它的官方网站[http://phantomjs.org/download.html)](http://phantomjs.org/download.html) 下载。 因为 PhantomJS 是一个功能完善(虽然无界面)的浏览器而非一个 Python 库，所以它不需要像 Python 的其他库一样安装，但我们可以通过Selenium调用PhantomJS来直接使用。

PhantomJS 官方参考文档：[http://phantomjs.org/documentation](http://phantomjs.org/documentation/)

* 1. 快速入门

Selenium 库里有个叫 WebDriver 的 API。WebDriver 有点儿像可以加载网站的浏览器，但是它也可以像 BeautifulSoup 或者其他 Selector 对象一样用来查找页面元素，与页面上的元素进行交互 (发送文本、点击等)，以及执行其他动作来运行网络爬虫。





* 1. 页面操作

Selenium 的 WebDriver提供了各种方法来寻找元素，假设下面有一个表单输入框：

<input type="text" name="user-name" id="passwd-id" />

那么：

# 获取id标签值

element = driver.find\_element\_by\_id("passwd-id")

# 获取name标签值

element = driver.find\_element\_by\_name("user-name")

# 获取标签名值

element = driver.find\_elements\_by\_tag\_name("input")

# 也可以通过XPath来匹配

element = driver.find\_element\_by\_xpath("//input[@id='passwd-id']")

* + 1. 定位UI元素 (WebElements)

关于元素的选取，有如下的API 单个元素选取

find\_element\_by\_id

find\_elements\_by\_name

find\_elements\_by\_xpath

find\_elements\_by\_link\_text

find\_elements\_by\_partial\_link\_text

find\_elements\_by\_tag\_name

find\_elements\_by\_class\_name

find\_elements\_by\_css\_selector

* + - 1. 通过ID

<div id="coolestWidgetEvah">...</div>

代码

element = driver.find\_element\_by\_id("coolestWidgetEvah")

------------------------ or -------------------------

from selenium.webdriver.common.by import By

element = driver.find\_element(by=By.ID, value="coolestWidgetEvah")

* + - 1. 通过class名字

<div class="cheese"><span>Cheddar</span></div><div class="cheese"><span>Gouda</span></div>

代码

cheeses = driver.find\_elements\_by\_class\_name("cheese")

------------------------ or -------------------------

from selenium.webdriver.common.by import By

cheeses = driver.find\_elements(By.CLASS\_NAME, "cheese")

* + - 1. 通过tag名字

<iframe src="..."></iframe>

代码

frame = driver.find\_element\_by\_tag\_name("iframe")

------------------------ or -------------------------

from selenium.webdriver.common.by import By

frame = driver.find\_element(By.TAG\_NAME, "iframe")

* + - 1. 通过name属性

<input name="cheese" type="text"/>

代码

cheese = driver.find\_element\_by\_name("cheese")

------------------------ or -------------------------

from selenium.webdriver.common.by import By

cheese = driver.find\_element(By.NAME, "cheese")

* + - 1. 通过Link Text

<a href="http://www.google.com/search?q=cheese">cheese</a>

代码

cheese = driver.find\_element\_by\_link\_text("cheese")

------------------------ or -------------------------

from selenium.webdriver.common.by import By

cheese = driver.find\_element(By.LINK\_TEXT, "cheese")

* + - 1. 通过Partial Link Text

<a href="http://www.google.com/search?q=cheese">search for cheese</a>>

代码

cheese = driver.find\_element\_by\_partial\_link\_text("cheese")

------------------------ or -------------------------

from selenium.webdriver.common.by import By

cheese = driver.find\_element(By.PARTIAL\_LINK\_TEXT, "cheese")

* + - 1. 通过CSS

<div id="food"><span class="dairy">milk</span><span class="dairy aged">cheese</span></div>

代码

cheese = driver.find\_element\_by\_css\_selector("#food span.dairy.aged")

------------------------ or -------------------------

from selenium.webdriver.common.by import By

cheese = driver.find\_element(By.CSS\_SELECTOR, "#food span.dairy.aged")

* + - 1. 通过XPath

<input type="text" name="example" />

<INPUT type="text" name="other" />

代码

inputs = driver.find\_elements\_by\_xpath("//input")

------------------------ or -------------------------

from selenium.webdriver.common.by import By

inputs = driver.find\_elements(By.XPATH, "//input")

* + 1. 鼠标动作链

有些时候，我们需要再页面上模拟一些鼠标操作，比如双击、右击、拖拽甚至按住不动等，我们可以通过导入 ActionChains 类来做到：

比如：

#导入 ActionChains 类

from selenium.webdriver import ActionChains

# 鼠标移动到 ac 位置

ac = driver.find\_element\_by\_xpath('element')

ActionChains(driver).move\_to\_element(ac).perform()

# 在 ac 位置单击

ac = driver.find\_element\_by\_xpath("elementA")

ActionChains(driver).move\_to\_element(ac).click(ac).perform()

# 在 ac 位置双击

ac = driver.find\_element\_by\_xpath("elementB")

ActionChains(driver).move\_to\_element(ac).double\_click(ac).perform()

# 在 ac 位置右击

ac = driver.find\_element\_by\_xpath("elementC")

ActionChains(driver).move\_to\_element(ac).context\_click(ac).perform()

# 在 ac 位置左键单击hold住

ac = driver.find\_element\_by\_xpath('elementF')

ActionChains(driver).move\_to\_element(ac).click\_and\_hold(ac).perform()

# 将 ac1 拖拽到 ac2 位置

ac1 = driver.find\_element\_by\_xpath('elementD')

ac2 = driver.find\_element\_by\_xpath('elementE')

ActionChains(driver).drag\_and\_drop(ac1, ac2).perform()

* + 1. 填充表单

我们已经知道了怎样向文本框中输入文字，但是有时候我们会碰到<select> </select>标签的下拉框。直接点击下拉框中的选项不一定可行。

<select id="status" class="form-control valid" onchange="" name="status">

<option value=""></option>

<option value="0">未审核</option>

<option value="1">初审通过</option>

<option value="2">复审通过</option>

<option value="3">审核不通过</option>

</select>

Selenium专门提供了Select类来处理下拉框。 其实 WebDriver 中提供了一个叫 Select 的方法，可以帮助我们完成这些事情：

# 导入 Select 类

from selenium.webdriver.support.ui import Select

# 找到 name 的选项卡

select = Select(driver.find\_element\_by\_name('status'))

#

select.select\_by\_index(1)

select.select\_by\_value("0")

select.select\_by\_visible\_text(u"未审核")

以上是三种选择下拉框的方式，它可以根据索引来选择，可以根据值来选择，可以根据文字来选择。注意：

* index 索引从 0 开始
* value是option标签的一个属性值，并不是显示在下拉框中的值
* visible\_text是在option标签文本的值，是显示在下拉框的值

实现全部取消选择:

select.deselect\_all()

* + 1. 弹窗处理

当你触发了某个事件之后，页面出现了弹窗提示，处理这个提示或者获取提示信息方法如下：

alert = driver.switch\_to\_alert()

* + 1. 页面切换

一个浏览器肯定会有很多窗口，所以我们肯定要有方法来实现窗口的切换。切换窗口的方法如下：

driver.switch\_to.window("this is window name")

也可以使用 window\_handles 方法来获取每个窗口的操作对象。例如：

for handle in driver.window\_handles:

driver.switch\_to\_window(handle)

* + 1. 页面前进和后退

操作页面的前进和后退功能：

driver.forward() #前进

driver.back() # 后退

* + 1. Cookies

获取页面每个Cookies值，用法如下

for cookie in driver.get\_cookies():

print "%s -> %s" % (cookie['name'], cookie['value'])

删除Cookies，用法如下

# By name

driver.delete\_cookie("CookieName")

# all

driver.delete\_all\_cookies()

* 1. 页面等待

注意：此处非常重要

现在的网页越来越多采用了 Ajax 技术，这样程序便不能确定何时某个元素完全加载出来了。如果实际页面等待时间过长导致某个dom元素还没出来，但是你的代码直接使用了这个WebElement，那么就会抛出NullPointer的异常。

为了避免这种元素定位困难而且会提高产生 ElementNotVisibleException 的概率。所以 Selenium 提供了两种等待方式，一种是隐式等待，一种是显式等待。

隐式等待是等待特定的时间，显式等待是指定某一条件直到这个条件成立时继续执行。

* + 1. 显式等待

显式等待指定某个条件，然后设置最长等待时间。如果在这个时间还没有找到元素，那么便会抛出异常了。

from selenium import webdriver

from selenium.webdriver.common.by import By

# WebDriverWait 库，负责循环等待

from selenium.webdriver.support.ui import WebDriverWait

# expected\_conditions 类，负责条件出发

from selenium.webdriver.support import expected\_conditions as EC

driver = webdriver.Chrome()

driver.get("http://www.xxxxx.com/loading")

try:

# 页面一直循环，直到 id="myDynamicElement" 出现

element = WebDriverWait(driver, 10).until(

EC.presence\_of\_element\_located((By.ID, "myDynamicElement"))

)

finally:

driver.quit()

如果不写参数，程序默认会 0.5s 调用一次来查看元素是否已经生成，如果本来元素就是存在的，那么会立即返回。

下面是一些内置的等待条件，可以直接调用，而不用自己写这些等待条件。

title\_is

title\_contains

presence\_of\_element\_located

visibility\_of\_element\_located

visibility\_of

presence\_of\_all\_elements\_located

text\_to\_be\_present\_in\_element

text\_to\_be\_present\_in\_element\_value

frame\_to\_be\_available\_and\_switch\_to\_it

invisibility\_of\_element\_located

element\_to\_be\_clickable – it is Displayed and Enabled.

staleness\_of

element\_to\_be\_selected

element\_located\_to\_be\_selected

element\_selection\_state\_to\_be

element\_located\_selection\_state\_to\_be

alert\_is\_present

* + 1. 隐式等待

隐式等待比较简单，就是简单地设置一个等待时间，单位为秒。

from selenium import webdriver

driver = webdriver.Chrome()

driver.implicitly\_wait(10) # seconds

driver.get("http://www.xxxxx.com/loading")

myDynamicElement = driver.find\_element\_by\_id("myDynamicElement")

当然如果不设置，默认等待时间为0。

1. 课程总结
   1. 重点
      * 1. 多线程爬虫。
        2. Selenium和PhantomJS的使用。
   2. 难点
      * 1. 多线程程序中共享数据结构的加锁
        2. 多线程程序中线程结束条件的判定
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
   2. 选做题（请在能于12点前睡觉的前提下尝试）
      * 1. 在多线程爬虫程序中添加动态获取url地址添加到urls队列中的功能:

* 从下一页标签中获取其他页面的url
* 从页面中获取电影详细页面url

1. 面试题
   * + 1. 用 Python 匹配 HTML的tag 的时候，<.\*> 和 <.\*?> 有什么区别？