### 爬虫(Spider)，反爬虫(Anti-Spider)，反反爬虫(Anti-Anti-Spider) 之间恢宏壮阔的斗争...

#### Day 1

* 小莫想要某站上所有的电影，写了标准的爬虫(基于HttpClient库)，不断地遍历某站的电影列表页面，根据 Html 分析电影名字存进自己的数据库。
* 这个站点的运维小黎发现某个时间段请求量陡增，分析日志发现都是 IP(xxx.xxx.xxx.xxx)这个用户，并且 user-agent 还是 Python-urllib/2.7 ，基于这两点判断非人类后直接在服务器上封杀。

#### Day 2

* 小莫电影只爬了一半，于是也针对性的变换了下策略：1. user-agent 模仿百度("Baiduspider...")，2. IP每爬半个小时就换一个IP代理。
* 小黎也发现了对应的变化，于是在服务器上设置了一个频率限制，每分钟超过120次请求的再屏蔽IP。 同时考虑到百度家的爬虫有可能会被误伤，想想市场部门每月几十万的投放，于是写了个脚本，通过 hostname 检查下这个 ip 是不是真的百度家的，对这些 ip 设置一个白名单。

#### Day 3

* 小莫发现了新的限制后，想着我也不急着要这些数据，留给服务器慢慢爬吧，于是修改了代码，随机1-3秒爬一次，爬10次休息10秒，每天只在8-12，18-20点爬，隔几天还休息一下。
* 小黎看着新的日志头都大了，再设定规则不小心会误伤真实用户，于是准备换了一个思路，当3个小时的总请求超过50次的时候弹出一个验证码弹框，没有准确正确输入的话就把 IP 记录进黑名单。

#### Day 4

* 小莫看到验证码有些傻脸了，不过也不是没有办法，先去学习了图像识别（关键词 PIL，tesseract），再对验证码进行了二值化，分词，模式训练之后，总之最后识别了小黎的验证码（关于验证码，验证码的识别，验证码的反识别也是一个恢弘壮丽的斗争史...），之后爬虫又跑了起来。
* 小黎是个不折不挠的好同学，看到验证码被攻破后，和开发同学商量了变化下开发模式，数据并不再直接渲染，而是由前端同学异步获取，并且通过 JavaScript 的加密库生成动态的 token，同时加密库再进行混淆（比较重要的步骤的确有网站这样做，参见淘宝和微博的登陆流程）。

#### Day 5

* 混淆过的加密库就没有办法了么？当然不是，可以慢慢调试，找到加密原理，不过小莫不准备用这么耗时耗力的方法，他放弃了基于 HttpClient的爬虫，选择了内置浏览器引擎的爬虫(关键词：PhantomJS，Selenium)，在浏览器引擎运行页面，直接获取了正确的结果，又一次拿到了对方的数据。
* 小黎：.....

### 爬虫与发爬虫的斗争还在继续...

通常情况下，在爬虫与反爬虫的对弈中，爬虫一定会胜利。

换言之，只要人类能够正常访问的网页，爬虫在具备同等资源的情况下就一定可以抓取到。

### 关于爬虫部分一些建议：

1. 尽量减少请求次数，能抓列表页就不抓详情页，减轻服务器压力，程序员都是混口饭吃不容易。
2. 不要只看 Web 网站，还有手机 App 和 H5，这样的反爬虫措施一般比较少。
3. 实际应用时候，一般防守方做到根据 IP 限制频次就结束了，除非很核心的数据，不会再进行更多的验证，毕竟成本的问题会考虑到。
4. 如果真的对性能要求很高，可以考虑多线程(一些成熟的框架如 Scrapy都已支持)，甚至分布式...

**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，类型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。

可以从 PyPI 网站下载 Selenium库[https://pypi.python.org/simple/selenium](https://pypi.python.org/simple/selenium/) ，也可以用 第三方管理器 pip用命令安装：pip install selenium

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

**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/)

**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。

# IPython2 测试代码

# 导入 webdriver

from selenium import webdriver

# 要想调用键盘按键操作需要引入keys包

from selenium.webdriver.common.keys import Keys

# 调用环境变量指定的PhantomJS浏览器创建浏览器对象

driver = webdriver.PhantomJS()

# 如果没有在环境变量指定PhantomJS位置

# driver = webdriver.PhantomJS(executable\_path="./phantomjs"))

# get方法会一直等到页面被完全加载，然后才会继续程序，通常测试会在这里选择 time.sleep(2)

driver.get("http://www.baidu.com/")

# 获取页面名为 wrapper的id标签的文本内容

data = driver.find\_element\_by\_id("wrapper").text

# 打印数据内容

print data

# 打印页面标题 "百度一下，你就知道"

print driver.title

# 生成当前页面快照并保存

driver.save\_screenshot("baidu.png")

# id="kw"是百度搜索输入框，输入字符串"长城"

driver.find\_element\_by\_id("kw").send\_keys(u"缝纫机乐队")

# id="su"是百度搜索按钮，click() 是模拟点击

driver.find\_element\_by\_id("su").click()

# 获取新的页面快照

driver.save\_screenshot("缝纫机乐队.png")

# 打印网页渲染后的源代码

print driver.page\_source

# 获取当前页面Cookie

print driver.get\_cookies()

# ctrl+a 全选输入框内容

driver.find\_element\_by\_id("kw").send\_keys(Keys.CONTROL,'a')

# ctrl+x 剪切输入框内容

driver.find\_element\_by\_id("kw").send\_keys(Keys.CONTROL,'x')

# 输入框重新输入内容

driver.find\_element\_by\_id("kw").send\_keys("itxdl")

# 模拟Enter回车键

driver.find\_element\_by\_id("su").send\_keys(Keys.RETURN)

# 清除输入框内容

driver.find\_element\_by\_id("kw").clear()

# 生成新的页面快照

driver.save\_screenshot("itxdl.png")

# 获取当前url

print driver.current\_url

# 关闭当前页面，如果只有一个页面，会关闭浏览器

# driver.close()

# 关闭浏览器

driver.quit()

**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']")

**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. By ID
2. <div id="coolestWidgetEvah">...</div>
   * 实现
   * element = driver.find\_element\_by\_id("coolestWidgetEvah")
   * ------------------------ or -------------------------
   * from selenium.webdriver.common.by import By
   * element = driver.find\_element(by=By.ID, value="coolestWidgetEvah")
3. By Class Name
4. <div class="cheese"><span>Cheddar</span></div><div class="cheese"><span>Gouda</span></div>
   * 实现
   * cheeses = driver.find\_elements\_by\_class\_name("cheese")
   * ------------------------ or -------------------------
   * from selenium.webdriver.common.by import By
   * cheeses = driver.find\_elements(By.CLASS\_NAME, "cheese")
5. By Tag Name
6. <iframe src="..."></iframe>
   * 实现
   * frame = driver.find\_element\_by\_tag\_name("iframe")
   * ------------------------ or -------------------------
   * from selenium.webdriver.common.by import By
   * frame = driver.find\_element(By.TAG\_NAME, "iframe")
7. By Name
8. <input name="cheese" type="text"/>
   * 实现
   * cheese = driver.find\_element\_by\_name("cheese")
   * ------------------------ or -------------------------
   * from selenium.webdriver.common.by import By
   * cheese = driver.find\_element(By.NAME, "cheese")
9. By Link Text
10. <a href="http://www.google.com/search?q=cheese">cheese</a>
    * 实现
    * cheese = driver.find\_element\_by\_link\_text("cheese")
    * ------------------------ or -------------------------
    * from selenium.webdriver.common.by import By
    * cheese = driver.find\_element(By.LINK\_TEXT, "cheese")
11. By Partial Link Text
12. <a href="http://www.google.com/search?q=cheese">search for cheese</a>>
    * 实现
    * 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")
13. By CSS
14. <div id="food"><span class="dairy">milk</span><span class="dairy aged">cheese</span></div>
    * 实现
    * 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")
15. By XPath
16. <input type="text" name="example" />
17. <INPUT type="text" name="other" />
    * 实现
    * inputs = driver.find\_elements\_by\_xpath("//input")
    * ------------------------ or -------------------------
    * from selenium.webdriver.common.by import By
    * inputs = driver.find\_elements(By.XPATH, "//input")

**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：

示例：

#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()

**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

全部取消选择怎么办呢？很简单:

select.deselect\_all()

**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()

**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)

**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

**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()

**页面等待**

注意：这是非常重要的一部分！！

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。

**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，你可以直接调用这些条件，而不用自己写某些等待条件了。

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

**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。