selenium

- selenium是什么
  - Selenium是一个用于Web应用程序测试的工具。
  - Selenium 测试直接运行在浏览器中，就像真正的用户在操作一样
  - 这个工具的主要功能包括：测试与浏览器的兼容性——测试应用程序看是否能够很好地工作在不同浏览器和操作系统之上。测试系统功能——创建回归测试检验软件功能和用户需求。支持自动录制动作和自动生成.Net、Java、Perl等不同语言的测试脚本。
- 为什么使用selenium
  - 模拟浏览器功能，自动执行网页中的js代码，实现动态加载
  - 在爬虫领域 selenium 是一把利器，能够解决大部分反网页的反爬问题
    - 对于逆向的参数可以不用管 因为我们是模拟正常用户
  - 自动化处理验证码此项简单
  - 对于用请求库(requests)不好爬取的网站可以用selenium 但是不要所有的网站都用 效率不高且吃资源
- selenium安装
  - pip install selenium==3.141.0
    - 推荐下3.x版本的
    - 直接pip install selenium安装的是4.x的
    - 4.x版本有些方式使用起来需要再进行导包 不方便
- 安装浏览器驱动（根据自身浏览器版本下载对应驱动）
  - 针对不同的浏览器，需要安装不同的驱动 课程主要讲的谷歌浏览器 以谷歌浏览器为准
  - 确定浏览器版本
    - 谷歌浏览器右上角三个点 -->帮助-->关于
    - 根据系统和版本进行下载
      - 如window64位 浏览器版本为111.0
      - index of /111.0.5563.15/

Name	Last modified	Size	Etag
Parent Directory			
chromedriver_linux64.zip	2023-01-31 21:00	7.8M	7e0ecaa1f7ecaeca-gzip
chromedriver_mac64.zip	2023-01-31 21:00	8.2M	7e0ecaa1f7ecaeca-gzip
chromedriver_mac64_m1.zip	2023-01-31 21:00	7.9M	7e0ecaa1f7ecaeca-gzip
chromedriver_win32.zip	2023-01-31 21:00	6.6M	7e0ecaa1f7ecaeca-gzip
      - 64位 把window下载32位的压缩包没问题 能用
      - 浏览器版本前两位重要 后面小版本没有太大影响（前两位是111.0 后面的0563.15不重对上，13没啥影响）
  - 谷歌浏览器驱动下载链接
    - http://chromedriver.storage.googleapis.com/index.html
  - 驱动存放位置
    - 方法1 推荐
      - 解压压缩包 得到exe文件 放到python的安装目录下
      - 如 E:\python38 和python.exe同级
    - 方法2 不推荐
      - 每次实例化指定驱动路径
        - webdriver.Chrome(executable_path="驱动exe路径")
  - 谷歌浏览器自动更新问题
    - 方法1
      - 如果浏览器自动更新导致驱动版本和浏览器版本不一致就不能使用
      - 谷歌浏览器自动更新后就去下载对应版本的驱动
    - 方法2
      - 禁用谷歌浏览器更新
      - 此电脑右键 -> 管理 -> 服务 把Google的两个更新服务 禁用
      - 有些电脑服务可能没有这两个服务 或者只有一个 可以用新一下浏览器是否有自动更新 更新就使用方法1
- 基本方法与使用
  - # 导包
    from selenium import webdriver
    # Chrome浏览器 实例化
    browser= webdriver.Chrome()
    browser.get('url') #访问页面
  - browser.maximize_window() # 窗口最大化
  - browser.minimize_window() # 最小化到任务栏
  - browser.set_window_size(600,700) # 自定义大小 浏览器宽600 高700
  - browser.save_screenshot('image.png') # 当前窗口截图
    - 以png后缀结尾 源码写的！！！
  - browser.refresh() # 刷新页面
  - browser.back() # 后退 浏览器左上角的后退按钮
  - browser.forward() # 前进 浏览器左上角的前进按钮
  - browser.close() # 关闭当前标签页
  - browser.quit() # 关闭浏览器
  - browser.title # 当前标签页标题
  - browser.current_url # 当前url
  - .size 返回元素大小
  - .text 获取元素的文本
  - get_attribute('xxx') 获取属性值.xxx：要获取的属性
  - .is_display() 判断元素是否可见
  - .is_enabled() 判断元素是否可用
  - 注意
    - size、text、title、current_url：为属性，调用时无括号；如：xxx.size
    - title、current_url：使用浏览器实例化对象直接调用。 如：browser.title
- 元素定位
  - browser.find_element_by_id('id值') # ID定位
  - browser.find_element_by_class_name('class属性值') # class 定位
  - browser.find_element_by_name('name值') # name 定位
  - browser.find_element_by_css_selector('css选择器') # css选择器定位
  - browser.find_element_by_xpath('xpath规则') # xpath定位
- 窗口切换
  - 在 selenium 操作页面的时候，可能会因为点击某个链接页面转到一个新的页面（打开了一个新标签页），这时候 selenium 实际还是处于上一个页面的，需要我们进行切换才能够定位最新页面上的元素
  - 标签页切换
    - 1. 获取所有标签页的窗口句柄
      - window = browser.window_handles # 返回列表
    - 2. 利用窗口句柄 切换到句柄指向的标签页
      - browser.switch_to.window(window_list)
    - 注意 多个窗口切换的情况 按打开的窗口最倒序插入
      - 如：js代码打开三个新窗口 第一个打开百度 第二个打开豆瓣 句柄列表顺序为[js代页面，豆瓣，百度]
      - get页面默认就是第一个
  - 表单切换
    - iframe 页面嵌套了页面 获取内页面的内容直接是定位不到
    - 需要进入到内页面才能定位
    - browser.switch_to.frame('id')
      - # 传iframe id的值 如果没有 也可以写元素定位
      - # 元素定位写法
        browser.switch_to.frame(browser.find_element_by_xpath('//iframe[@id="iframeResult"]'))
- selenium动作
  - send_keys() 模拟输入指定内容
  - clear() 清除文本内容
  - click() 单击左键
  - from selenium.webdriver import ActionChains
    browser = webdriver.Chrome() # 实例化
    action = ActionChains(browser)
    - click_and_hold() 点击且长按指定标签
    - drag_and_drop() 拖动
    - move_to_element() 鼠标移动到某个元素
    - move_by_offset() 鼠标从当前位置移动到某个坐标
    - release() 在某个位置松开鼠标左键
    - perform() 执行链中的所有动作
    - move_to_element_with_offset() 移动到距某个元素（左上角坐标）多少距离的位置
    - key_up() 松开某个键
- selenium配置项
  - from selenium import webdriver
    option = webdriver.ChromeOptions() # 实例化配置对象
    browser = webdriver.Chrome(options =option)
  - option.add_argument('--headless') # 指定无头模式 无界面启动
  - # 防检测
    browser.execute_cdp_cmd("Page.addScriptToEvaluateOnNewDocument", {"source": """ Object.defineProperty(navigator, 'webdriver', { get: () => undefined }) """})
    - window.navigator.webdriver 为true代表检测到自动化
    - 这个代码是修改 window.navigator.webdriver为undefined 可以改为false
  - option.add_argument('User-Agent={}'.format('Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.70 Safari/537.36')) # 更换UA
  - option.add_argument('--proxy-server=http://ip:port') # 更换ip
- 元素等待
  - 为什么需要设置元素等待
    - 因为：网站加载页面 会加载很多的html css js文件
    - 但是：加载肯定有加载顺序，大型网站很快会一秒内就把所有东西加载出来，不仅如此，加载速度也受网络波动影响
    - 因此：当我们要在页面中做一元素定位一定时候，有可能刚打开了网页但元素未加载出来，这个时候就定位不到元素，就会报错
    - 所以：我们需要设置元素等待，意思就是：等待指定元素已被加载出来之后，再去才去定位该元素，就不会出现定位失败的现象了
  - 等待方法1 time.sleep(3) 强制等待 必须等3秒
  - 等待方法2 隐式等待 implicitly_wait()
    - 如果在规定时间内网页加载完成，则执行下一步，否则一直等到时间截止，然后执行下一步，抛出异常
    - 隐式等待是全局性的 只需要设置一次
    - 即运行过程中，如果元素可以定位到，这个命令时不起作用，但如果定位不到，则它会以轮询的方式不断地动问元素直到元素被找到，若超过指定时间，则抛出异常。
  - 等待方法3 显式等待
    - 可以它看成是局部变量，作用于指定元素
    - 就是直到元素出现才去操作，如果超时则报异常
    - 显示等待需要使用 WebDriverWait，同时配合 until 或 not until
    - until(method, message='')
      - method：指定预期条件的判断方法，在等待期间，每隔一段时间调用该方法，判断元素是否存在，直到元素出现
      - message 如果超时，抛出 TimeoutException ，并显示 message 中的内容
    - until_not(method, message='')
      - until_not 正好相反，当元素消失或指定条件不成立，则继续执行后续代码
    - 主要方法看其他课件代码

XMind:ZEN 试用模式