Методы сбора и обработки данных при помощи Python

Selenium в Python





Оглавление

Введение

История создания

Программные продукты Selenium

Selenium WebDriver

Начало работы

Загрузка Selenium для Python

Пошаговый разбор примера

Навигация

Взаимодействие со страницей

Заполнение форм

Перетаскивание

Переключение между окнами и фреймами

Всплывающие окна

Навигация: история и локация

Куки (cookies)

Поиск элементов

Поиск по Id

Поиск по Name

Поиск по XPath

Поиск гиперссылок по тексту

Поиск элементов по тегу

Поиск элементов по классу

Поиск элементов по CSS-селектору

Ожидания

Явные ожидания

Неявные ожидания

Ускорение работы

Глоссарий

Домашнее задание

Используемая литература

Введение

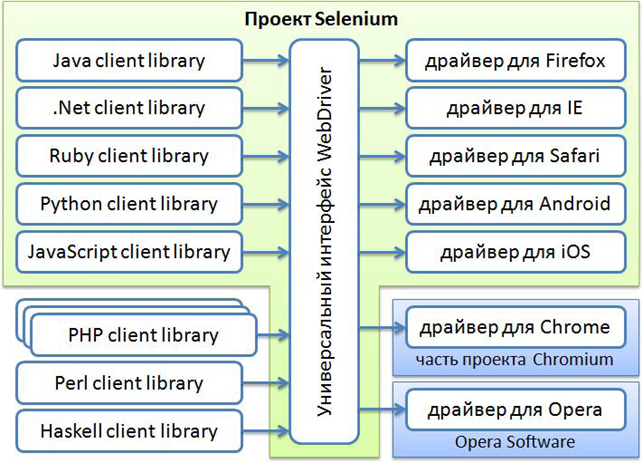
Сегодня мы познакомимся с Selenium’ом - это инструмент, который чаще сего используется для автоматизации тестирования. Однако для сбора данных он тоже подходит. Хочу обратить ваше внимание на то, что если вам требуется селениум для сбора данных, скорее всего, вы пытаетесь собрать данные с сайта, который активно этому сопротивляется. Помните, пожалуйста, об этичном парсинге и постарайтесь не ломать сайты.

Официальная страница проекта — [SeleniumHQ Browser Automation](http://docs.seleniumhq.org/)

Selenium WebDriver

Selenium WebDriver (Selenium 2) — это программная библиотека для управления браузерами, основной продукт в рамках проекта Selenium. Она включает набор модулей для разработки ПО.

Selenium WebDriver состоит из набора драйверов и клиентских библиотек для таких браузеров, как Firefox, Safari, Chrome и Edge.



Чаще всего Selenium WebDriver используют для тестирования функционала веб-сайтов/веб-ориентированных приложений. Но мы будем его использовать для парсинга.

Загрузка Selenium для Python

Вы можете установить Селениум через pip, однако лучше зайти на [pypi.org](http://pypi.org), найти там селениум и не только установить его, но и скачать браузер, через который мы будем осуществлять парсинг. Я предпочитаю использовать firefox, для которого у селениума есть гекодрайвер

После скачивания драйвера и установки селениума давайте создадим новую папку селениум. Для удобства я положу в эту папку и гекодрайвер, с помощью которого мы будем запускать наш селениум.

Сначала мы научимся логиниться на сайт, затем научимся парсить страницы с бесконечной прокруткой. То есть, например, такие сайты, как ТикТок, где лента имеет бесконечную прокрутку.

Итак, сначала нам надо научиться авторизовываться на сайте. Для этого воспользуемся ссылкой [quotes.toscrape.com/login](http://quotes.toscrape.com/login)

Первое, что мы делаем - это импортируем необходимые библиотеки.

**from selenium import webdriver**

**from selenium.webdriver.common.by import By**

**from selenium.webdriver.support.ui import WebDriverWait**

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

Класс By поможет нам в поиске необходимых тегов, WebDriverWait позволит нам ждать загрузку элементов на странице, а EC позволит устанавливать условия, выполнение которых необходимо для успешной работы парсера.

Теперь, первое, что нам надо сделать - это создать дравйер, с помощью которого мы будем открывать страницу и выполнять необходимые действия. В executable\_path указываем путь до гекодрайвера.

**driver = webdriver.Firefox(executable\_path=‘./geckodriver')**

Затем с помощью метода get() мы передаем ссылку на сайт, который надо открыть.

**driver.get('<https://quotes.toscrape.com/login>')**

Хочу обратить ваше внимание, что браузер в конце надо закрывать, по тому же принципу, что и работа с базами данных, например. Так что возьмите за правило в конце скрипта обязательно сразу закрывать браузер, иначе вам придется это делать руками. Закрываем браузер с помощью метода quit()

**driver.quit()**

После того, как драйвер перейдет по ссылке, указанной в методе гет, нам надо заполнить поля для юзернейма и пароля и нажать на кнопку логин. Если мы сразу выполним эти действия, скорее всего, наш скрипт сломается. Потому что драйверу потребуется какое-то время на загрузку страницы, а наш код будет требовать немедленно найти необходимые поля. Чтобы избежать таких ошибок мы будем использовать класс WebDriverWait. Он позволяет дожидаться появления на странице определенного элемента и только после этого выполнять дальнейшие действия с кодом.

Посмотрим, что у нас должно быть на нашей странице логина, чтобы мы могли авторизоваться. Допустим, мы будем ждать появления формы логина, а именно тега input с айди username и таким же именем.

**WebDriverWait(driver, 30).until(EC.presence\_of\_element\_located((By.ID, ‘username')))**

То есть код будет ждать 30 секунд, пока не появится элемент с айдишником юзернейм. Если элемент появится на 2ой секунде - код продолжит свое выполнение, если после 30 секунд элемент не появится - код завершит свое выполнение с ошибкой. Такие ожидания лучше заворачивать в try/except, но в данном курсе мы этого не будем касаться. Также лучше всегда указывать айди при поиске элемента, так как имен одинаковых может быть несколько, а вот айди на странице с именем юзернейм - один единственный.

После того, как появится форма на странице, нам надо найти поля логин и пароль и ввести туда значения. Так как это тестовый сайт, вы можете ввести абсолютно любые значения и в любом случае сможете залогиниться.

Давайте сначала найдем поле для ввода юзернейма. Оно находится, как мы уже знаем, в теге input с именем и айди name. Будем искать это поле с помощью xpath’a.

**login = driver.find\_element\_by\_xpath(«//input[@id='username']")**

Теперь посмотрим, где лежит поле для ввода пароля. Оно находится в теге инпут с именем password. Давайте это поле найдем с использованием метода find\_element\_by\_id. Вообще у селениума очень большой выбор методов, с помощью которых вы можете находить элементы на странице. Я предпочитаю xpath, но есть и поиск по имени тега, по его айдишнику, поиск по имени класса и многие другие. Это всё можно найти и изучить в документации. Итак, ищем поле пароля по айди тега.

**password = driver.find\_element\_by\_id(«password»)**

После того, как мы нашли эти поля, нам надо ввести в них логин и пароль, допустим, это будет admin/admin и нажать на кнопку логин. Сначала введем данные, используя метод send\_keys()

**login.send\_keys('admin')**

**password.send\_keys(‘admin')**

Теперь найдем кнопку логина и нажмем ее. Кнопка находится в теге инпут с атрибутом value равным login. Воспользуемся поискам по xpath.

**login\_btn = driver.find\_element\_by\_xpath(«//input[@value='Login']")**

После того, как элемент найден, используем метод click()

**login\_btn.click()**

Теперь, чтобы убедиться, что мы залогинились, давайте вернем список цитат и html страницы. Соберем просто html цитат, не будем углубляться в подробный сбор. И выведем длину получившегося списка.

Обратите внимание, после логина нам так же надо подождать какое-то время, пока страница загрузится. Так что скопируем написанный выше класс ожидания. И теперь будем ждать, пока на странице не появится элемент с классом quote.

**WebDriverWait(driver, 120).until(EC.presence\_of\_element\_located((By.CLASS\_NAME, ‘quote')))**

После этого получим html

**html = driver.page\_source**

И цитаты.

**quotes = driver.find\_elements\_by\_class\_name(‘quote')**

Обратите внимание, мы написали find elementS, в таком случае нам будет возвращен список всех найденных элементов. Если мы напишем element - нам вернется всего один элемент с указанным классом.

Выведем html и длину списка с цитатами

**print(html)**

**print(f'Количество цитат равно {len(quotes)}’)**

Проверяем, что не забыла закрыть драйвер

**driver.quit()**

И запускаем скрипт. Как видите, мы всё написали правильно и нам вывелся html и количество цитат - 10.

Теперь приступим к парсингу страниц с бесконечной прокруткой. Давайте в этой же папке создадим новый файлик - infinite\_scroll.py

Импортируем известные нам библиотеки, скопировав их из предыдущего скрипта, только еще добавим импорт time

**import time**

**from selenium import webdriver**

**from selenium.webdriver.common.by import By**

**from selenium.webdriver.support.ui import WebDriverWait**

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

**Создаем драйвер, так же, как и в предыдущем скрипте, можем просто скопировать**

**driver = webdriver.Firefox(executable\_path='./geckodriver')**

**driver.get('https://quotes.toscrape.com/scroll')**

А вот ссылку будем использовать другую - [quotes.toscrape.com/scroll](http://quotes.toscrape.com/scroll). Давайте посмотрим на нее в браузере. Как видите, мы можем прокручивать страницу почти бесконечно, при этом после прокрутки количество элементов с классом quote становится больше, так как добавляются новые, появившиеся при скролинге. Обратите внимание, что скролить мы можем как мышкой, так и нажимая пробел.

Мы попробуем спарсить данные, используя оба способа.

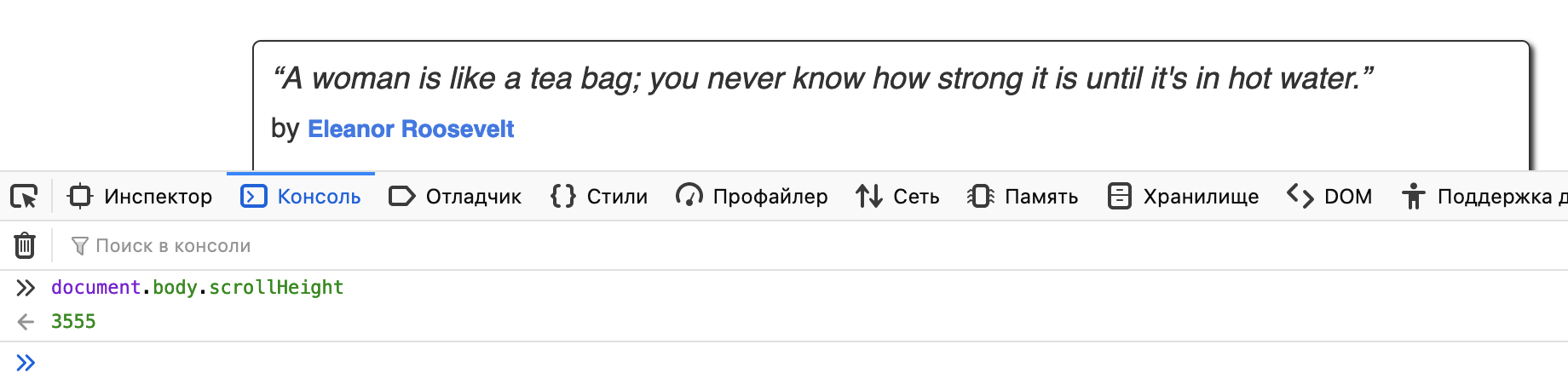
Для начала снова дождемся появления элементов на странице. Давайте скопируем из предыдущего скрипта - ожидание элемента с классом quote и немного изменим его. Теперь мы будем ждать, пока на странице появится див с классом quotes, так что допишем s в конце.

**WebDriverWait(driver, 30).until(EC.presence\_of\_element\_located((By.CLASS\_NAME, ‘quotes’)))**

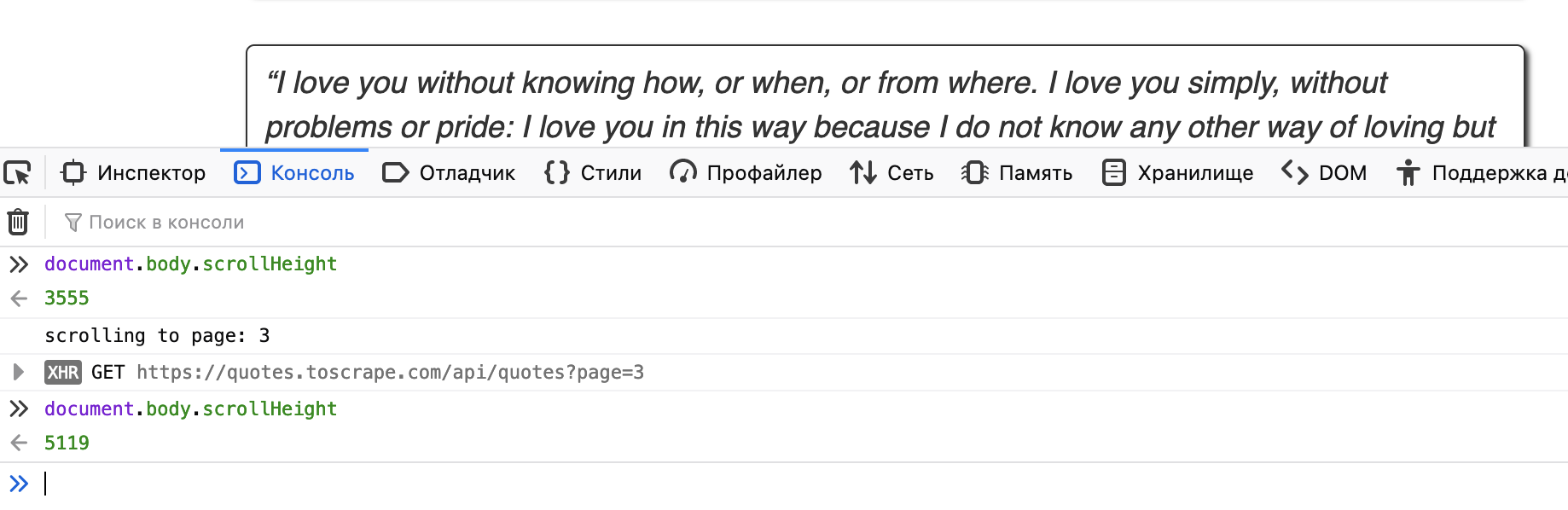
Смотрите, 30 секунд - это изменяемый параметр, вы можете ждать как 1 секунду, так и 1 минуту, учитывайте скорость загрузки страницы при установке лимита ожидания.

Теперь объясню вам логику работы скролинга в нашем скрипте. Нам надо, чтобы браузер прокручивал страницу вниз до того момента, пока прокрутка станет невозможной, то есть мы достигнем конца страницы. Чтобы определить момент выхода из цикла прокрутки мы будем использовать высоту тега body.

Сейчас я покажу вам наглядно. Идем на сайт и открываем консоль. Вводим

**document.body.scrollHeight**

Нажимаем enter. Вернулась высота тега body.

Теперь прокрутим страницу вниз до появления новых элементов. Снова введем document.body.scrollHeight, как видите, высота изменилась.

Давайте докрутим до самого конца. Еще раз вводим document.body.scrollHeight. Видите, теперь высота совсем другая, чем была в начале. Именно эту высоту мы и будем использовать как ориентир конца страницы.

Для начала еще до цикла мы получим и сохраним в переменную высоту нашего тега, используя метод execute\_script()

**body\_height = driver.execute\_script("return document.body.scrollHeight»)**

Теперь создадим цикл while True. В этом цикле мы будем прокручивать страницу вниз, получать высоту тега body и сравнивать нашу первую высоту с вновь полученной. Если высота будет различаться, мы будем перезаписывать body\_height и снова выполнять прокрутку. Если высота не будет меняться, это будет означать, что мы достигли конца страницы, - в таком случае мы будем выходить из цикла. Запишем

**while True:**

**driver.execute\_script(f"window.scrollTo(0, {last\_height});»)**

**new\_height = driver.execute\_script("return document.body.scrollHeight»)**

**if new\_height == body\_height:**

**break**

**body\_height = new\_height**

В driver.execute\_script мы выполняем команду скролинга окна - window.scrollTo. Внутри мы указываем две координаты: первая - координата по горизонтальной оси документа. Так как нам не надо скролить влево или вправо, мы указываем 0. Вторая координата - вертикальная ось документа, в которую мы передаем высоту нашего элемента body.

Теперь нам надо обязательно делать паузы между скролингом. Во-первых, чтобы дать элементам время на загрузку, во-вторых, чтобы не перегружать сайт. Запомните, чем больше и тяжелее у вас элементы на странице, тем медленнее элементы будут подгружаться в html. Поэтому если вам надо долго скролить, старайтесь использовать большие паузы.

Определим время паузы до начала цикла while

**pause\_time = 0.5**

И вставим это время внутри цикла

**driver.execute\_script(f"window.scrollTo(0, {last\_height});»)**

**pause\_time = 0.5**

**new\_height = driver.execute\_script("return document.body.scrollHeight»)**

И давайте после завершения цикла соберем все цитаты и вернем их длину. Для этого можем скопировать строчки из предыдущего кода.

**quotes = driver.find\_elements\_by\_class\_name("quote")**

**print(f'Количество цитат равно {len(quotes)}’)**

Так, запускаем парсер. Как видите, всё работает. Давайте теперь я вам покажу еще один способ скролить страницу, а именно, используя кнопку пробел. Для этого нам надо импортировать еще два класса.

**from selenium.webdriver.common.keys import Keys**

**from selenium.webdriver.common.action\_chains import ActionChains**

**Теперь создаем переменную actions, которая позволит нам выполнять действия в драйвере.**

**actions = ActionChains(driver)**

Получение высоты элемента и ее обновление в цикле мы оставляем без изменений, так же как и время паузы. А вот выполнение скрипта скролинга давайте закомментируем и напишем выполнение нажатия кнопки пробел.

**# driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")**

**actions.send\_keys(Keys.SPACE).perform()**

Запускаем скрипт. Как видите, он работает точно так же.

Еще есть несколько опций запуска драйвера, которые могут быть полезны при парсинге. Это опция отключения картинок и всплывающих окон. Для этого в начале скрипта нам надо создать профиль, задать профилю нужные значения, а затем при создании драйвера указать этот профиль.

**profile = webdriver.FirefoxProfile()**

**profile.set\_preference('permissions.default.image', 2)**

**profile.set\_preference('dom.ipc.plugins.enabled.libflashplayer.so', 'false')**

**driver = webdriver.Firefox(profile, executable\_path=‘./geckodriver')**

set\_preference('permissions.default.image', 2) - отключает картинки на странице

set\_preference('dom.ipc.plugins.enabled.libflashplayer.so', ‘false') - отключает всплывающие окна. В нашем примере это никак не повлияет на работу скрипта, так как у нас нет ни картинок, ни всплывающих окон. Однако в реальной жизни эти опции часто бывают полезны.

Еще есть опция запуска драйвера без графического интерфейса. Для этого надо импортировать класс options и прописать запуск без графического интерфейса.

**options = Options()**

**options.add\_argument('--headless')**

**driver = webdriver.Firefox(options=options, executable\_path=‘./geckodriver')**

Это ускорит работу скрипта и снизит нагрузку на систему, а также сэкономит объём загружаемых данных. Однако в большинстве случаев сайты распознают такие моменты и моментально блокируют парсеры. На этом наш урок, посвященный селениуму, подошел к концу. Безусловно его функционал не граничивается теми методами, что мы здесь разобрали. Это только самые простые и необходимые. И прошу вас не забывать об этичном парсинге. Работая с селениумом можно легко перегрузить сайт, создав видимость реальных запросов.

Глоссарий

**Selenium** — инструмент для автоматизации действий веб-браузера, в его рамках представлены такие продукты, как Selenium WebDriver, Selenium RC, Selenium Server, Selenium Grid, Selenium IDE.

**Selenium WebDriver** — программная библиотека для управления браузерами, основной продукт в рамках проекта Selenium. Она включает набор модулей для разработки ПО.

**XPath** — язык, использующийся для поиска узлов дерева XML-документа.

**Явное ожидание** — указание WebDriver ожидать возникновение определённого условия до начала действий.

**Неявное ожидание** — указание WebDriver опрашивать DOM определённое количество времени.

Дополнительные материалы

1. [Современная веб-автоматизация при помощи Python и Selenium](https://python-scripts.com/web-automation-with-python-and-selenium).

Домашнее задание

Залогиниться на [сайте](https://www.scrapethissite.com/pages/advanced/?gotcha=login). Вывести сообщение, которое появляется после логина (связка логин/пароль может быть любой).

Используемая литература

1. [The Selenium Browser Automation Project :: Documentation for Selenium](https://www.seleniumhq.org/docs/)

2. [The Selenium Browser Automation Project :: Documentation for Selenium](https://www.selenium.dev/documentation/en/)

3. [Open source record and playback test automation for the web](https://www.seleniumhq.org/selenium-ide/)

4. [Пробный запуск браузера](https://python-scripts.com/web-automation-with-python-and-selenium%252523zapusk-brauzera-headless)