Урок 7

Selenium в Python

Введение

Рассмотрим одно из самых популярных средств автоматизации — Selenium.

Этот проект популярен, так как он условно бесплатный. Selenium предоставляет программные продукты, которые можно свободно скачать и установить. Однако для оптимизации под задачи пользователя их необходимо дорабатывать, а это влечёт затраты на оплату работы программистов.

Selenium — инструмент для автоматизации действий веб-браузера. Чаще всего используется для тестирования. Официальная страница проекта — http://docs.seleniumhq.org/

История создания

В 2004 году разработчик Джейсон Хаггинс создал на языке JavaScript библиотеку JavaScriptTestRunner. Она служит для запуска тестов в браузере и сегодня известна как Selenium Core. Хаггинс работал в компании ThoughtWorks в Стамбуле. Его проект заинтересовал коллег и они активно включились в работу. Название Selenium закрепилось после того, как Джейсон Хаггинс с иронией высказался о конкурирующем проекте Mercury Interactive QuickTest Professional. Так как

mercury в переводе с английского означает «ртуть», разработчик нового ПО пошутил, что отравление ртутью лечат приёмом селениума. Код проекта открыли в конце 2014 года.

Программные продукты Selenium

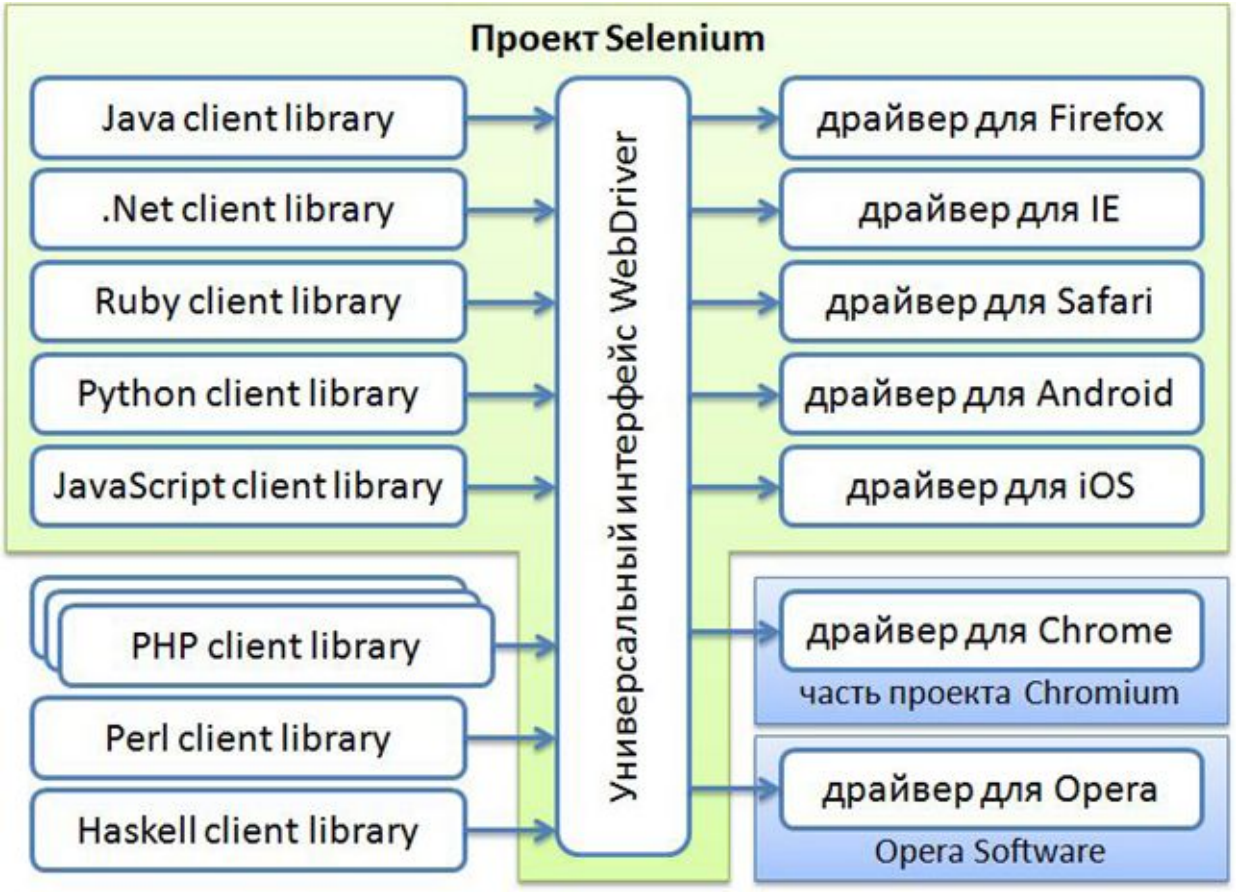
В рамках проекта Selenium разработана серия программных продуктов с открытым исходным кодом (open source):

- Selenium WebDriver;
- Selenium RC;
- Selenium Server;
- Selenium Grid;
- Selenium IDE.

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, Internet Explorer и Safari, а также мобильных приложений, работающих с операционными системами Android и iOS. Драйвер для Google Chrome разрабатывается в проекте Chromium, а для Opera (включая мобильные версии) разработку ведёт компания Opera Software.



Чаще всего Selenium WebDriver используют для тестирования функционала веб-сайтов/веб-ориентированных приложений. Автоматизированное тестирование удобно, потому что позволяет многократно запускать повторяющиеся тесты. Регрессионное тестирование, то есть, проверка, что код не перестал работать правильно после внесения изменений — типичный пример, когда необходима автоматизация. WebDriver предоставляет все необходимые методы, обеспечивает высокую скорость теста и гарантирует корректность проверки (поскольку человеческий фактор исключён). В официальной документации Selenium приводятся такие плюсы автоматизированного тестирования веб-приложений:

- возможность чаще проводить регрессионное тестирование;
- быстрое предоставление разработчикам отчета о состоянии продукта;
- потенциально бесконечное числа прогонов тестов;
- поддержка Agile и экстремальных методов разработки;
- сохранение строгой документации тестов;
- обнаружение ошибок, которые были пропущены на стадии ручного тестирования.

Функционал WebDriver позволяет использовать его не только для тестирования, но и для администрирования веб-сервисов, сократив до возможного предела количество действий, производимых вручную. Selenium WebDriver — незаменимый помощник в случаях, когда, например, ядро сайта устарело и требует от модераторов большого количества операций для реализации мелких фич (например, загрузки галереи фото).

Одна из незаменимых особенностей Selenium WebDriver — ожидание загрузки страницы. Сюда можно отнести случаи, когда парсинг данных на странице невозможен из-за страниц перенаправления или ожидания, содержащих примерно такой текст: «Подождите, страница загружается». Это нецелевые страницы для парсинга однако часто обойти их не представляется возможным. Естественно, без Selenium WebDriver. Он позволяет в таких случаях «ожидать», как ожидал бы человек, пока на странице не появится элемент с необходимым именем.

Ещё один плюс Selenium — действия веб-драйвера видны визуально и требуют минимального времени нахождения на странице, это позволяет с добством демонстрировать функционал сайта, когда необходима презентация сервиса.

Начало работы

Привязка Selenium к Python представляет собой простой API-интерфейс программирования приложений для написания тестов с использованием веб-драйвера Selenium WebDriver. С помощью Selenium Python API вы можете интуитивно получить доступ ко всему функционалу Selenium WebDriver.

Связка Python-Selenium предоставляет удобный API для доступа к таким веб-драйверам Selenium, как Firefox, IE, Chrome, Remote и другим. Сейчас поддерживаются версии Python 2.7, 3.2, 3.3 – 3.7.

Загрузка Selenium для Python

Вы можете загрузить привязку Selenium к Python со страницы пакета selenium на PyPI. Однако лучшим способом будет использование модуля pip. Python 3.4 содержит pip в стандартной библиотеке. Используя pip, вы можете установить Selenium следующей командой:

```
pip install selenium
```

Если вы установили привязку Selenium к Python, можете начать её использовать с помощью интерпретатора Python.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
driver = webdriver.Chrome()
driver.get("http://www.python.org")
assert "Python" in driver.title
elem = driver.find_element_by_name("q")
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
assert "No results found." not in driver.page_source
driver.close()
```

Пошаговый разбор примера

Модуль selenium.webdriver предоставляет весь функционал WebDriver. Сейчас он поддерживает реализации Firefox, Chrome, IE и Remote. Класс Keys обеспечивает взаимодействие с командами клавиатуры, такими, как RETURN, F1, ALT и т. д.

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

Далее создаётся элемент класса Chrome WebDriver.

```
driver = webdriver.Chrome()
```

Метод driver.get перенаправляет к странице URL в параметре. WebDriver будет ждать, пока страница не загрузится полностью (то есть событие onload игнорируется), прежде чем передать контроль вашему тесту или скрипту. Стоит отметить, что если страница использует много AJAX-кода при загрузке, то WebDriver может не распознать, загрузилась ли она полностью:

```
driver.get("http://www.python.org")
```

Следующая строка — это утверждение (англ. assertion), что заголовок содержит слово Python. Assert позволяет проверять предположения о значениях произвольных данных в произвольном месте программы. По своей сути команда напоминает констатацию факта, расположенного посреди кода программы. Если утверждение ошибочно, assert вызывает исключение. Такое поведение позволяет контролировать выполнение программы в строго определённом русле. Отличие assert от условий заключается в том, что программа с assert не приемлет иного хода событий, считая дальнейшее выполнение программы или функции бессмысленным:

```
assert "Python" in driver.title
```

WebDriver предоставляет ряд способов получить элементы с помощью методов find_element_by_*. Для примера, элемент ввода текста input может быть найден по его атрибуту name методом find_element_by_name.

```
elem = driver.find_element_by_name("q")
```

После этого мы посылаем сигнал нажатия клавиш (аналогично использованию клавиатуры). Специальные команды могут быть переданы с помощью класса Keys, импортированного из selenium.webdriver.common.keys:

```
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
```

После ответа страницы вы получите результат, если он был запрошен. Чтобы удостовериться, что мы его получили, добавим утверждение:

```
assert "No results found." not in driver.page_source
```

При завершении программы окно браузера закрывается. Вы можете также вызывать метод quit вместо close. Он закроет браузер полностью, в то время как close закроет одну вкладку. Однако в случае, когда открыта только одна вкладка, большинство браузеров закрываются полностью:

```
driver.close()
```

Навигация

Для перехода по ссылке с помощью WebDriver используют метод get:

```
driver.get("http://www.google.com")
```

Взаимодействие со страницей

Сама по себе возможность переходить по ссылке не так уж полезна. Нас интересует взаимодействие со страницей, точнее, с элементами HTML на ней. Прежде всего необходимо найти их. WebDriver предоставляет ряд способов для поиска элементов. К примеру, на странице есть элемент, определённый таким образом:

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

Его можно найти, используя любой из следующих методов:

```
element = driver.find_element_by_id("passwd-id")
element = driver.find_element_by_name("passwd")
element = driver.find_element_by_xpath("//input[@id='passwd-id']")
```

Также вы можете искать адрес гиперссылки по её тексту, но будьте бдительны: текст должен совпадать в точности. Также будьте внимательны при использовании XPath в WebDriver. Если существует больше одного элемента, удовлетворяющего условиям запроса, вернётся только первый найденный. Если ничего не найдено, будет вызвано исключение NoSuchElementException.

WebDriver имеет объектно-ориентированное API. Если вы вызовете метод, используя его неправильно (например, используете setSelected() для мета-тега), WebDriver вызовет исключение.

Итак, мы получили элемент. Что вы можете с ним сделать? Первым делом можно ввести какой-нибудь текст в текстовое поле:

```
element.send_keys("some text")
```

Также вы можете имитировать нажатие клавиш-стрелок клавиатуры с помощью класса Keys:

```
element.send_keys(" and some", Keys.ARROW_DOWN)
```

Метод send_keys можно вызвать для любого элемента, который позволяет проверить сочетания клавиш, как те, что используются в GMail. Существует побочный эффект, заключающийся в том, что новый ввод в текстовое поле не очищает его автоматически. Вместо этого то, что вы набираете на клавиатуре, будет дописываться к тому, что уже есть в поле. Очистить содержимое текстового поля или текстовой области textarea можно с помощью метода clear:

```
element.clear()
```

Заполнение форм

Мы уже рассмотрели ввод текста в текстовую область или текстовое поле, а как быть с другими элементами? Вы можете попробовать развернуть выпадающий список, после чего использовать setSelected для выделения тегов вроде OPTION. Работать с тегами SELECT не так уж сложно:

```
element = driver.find_element_by_xpath("//select[@name='name']")
all_options = element.find_elements_by_tag_name("option") for option
in all_options: print("Value is: %s" % option.get_attribute("value"))
option.click()
```

Такой код найдёт первый элемент SELECT на странице и в цикле пройдет по всем тегам OPTION поочерёдно, сообщая их значения и выделяя их.

Как можно заметить, это не самый быстрый способ работы с элементами SELECT. Поддерживаемые WebDriver классы содержат один, называющийся Select, он предоставляет более удобные способы взаимодействия:

```
from selenium.webdriver.support.ui import Select
select = Select(driver.find_element_by_name('name'))
select.select_by_index(index)
select.select_by_visible_text("text")
select.select_by_value(value)
```

Также WebDriver предоставляет возможность снятия выделения со всех элементов выпадающего списка:

```
select = Select(driver.find_element_by_id('id'))
select.deselect_all()
```

Этот код снимает выделение со всех тегов OPTION первого тега SELECT на странице.

Допустим, для теста вам необходим список всех выделенных по умолчанию опций. Класс Select предоставляет такое свойство (возвращает список):

```
select = Select(driver.find_element_by_xpath("xpath"))
all_selected_options = select.all_selected_options
```

Для получения всех доступных опций используйте:

```
options = select.options
```

После заполнения формы, вы, вероятно, захотите сохранить изменения. Один из способов сделать это — найти кнопку submit и кликнуть по ней:

```
# Предположим, ID кнопки равен "submit"
driver.find_element_by_id("submit").click()
```

В качестве альтернативы первому методу можно использовать метод submit, доступный для каждого элемента. Если вызвать его для элемента внутри формы, WebDriver пройдет по всей структуре DOM, пока не найдёт закрывающийся тег формы, а затем вызовет для неё submit. Если элемент находится вне формы, тогда будет вызвано исключение NoSuchElementException:

```
element.submit()
```

Перетаскивание

Есть два варианта перемещения элементов: на определённую величину или на другой элемент:

```
element = driver.find_element_by_name("source")
target = driver.find_element_by_name("target")
from selenium.webdriver import ActionChains
action_chains = ActionChains(driver)
action_chains.drag_and_drop(element, target)
```

Переключение между окнами и фреймами

Современные веб-приложения редко обходятся без фреймов (frame) и ограничиваются одним окном. WebDriver поддерживает переключение между именованными окнами с помощью метода switch_to_window:

```
driver.switch_to_window("windowName")
```

Все вызовы, начинающиеся с driver, теперь будут истолкованы как обращённые к полученному окну. Но откуда вам знать имя окна? Взгляните на код JS или ссылку, которые открывают окно:

```
<a href="кое_где.html" target="имяОкна">
Нажмите сюда, чтобы открыть новое окно
</a>
```

Также вы можете послать «дескриптор окна» методу switch_to_window(). Пользуясь этой особенностью, вы можете использовать цикл для перебора всех открытых окон, к примеру, так:

```
for handle in driver.window_handles:
driver.switch_to_window(handle)
```

Можно переходить между фреймами (frame или iframes):

```
driver.switch_to_frame("frameName")
```

Можно получить доступ к подчинённым фреймам, подавая путь, разделяемый точкой, или можно получить фрейм по индексу:

```
driver.switch_to_frame("frameName.0.child")
```

Следующий код перенаправит к фрейму с именем child, который в свою очередь принадлежит первому подчиненному фрейму фрейма frameName. **Пути к фреймам описываются полностью — от верхнего уровня**:

```
driver.switch_to_frame("frameName.0.child")
```

Когда работа с фреймами будет завершена, необходимо переключиться обратно к главному фрейму:

```
driver.switch_to_default_content()
```

Всплывающие окна

Selenium WebDriver из упаковки поддерживает управление всплывающими диалоговыми окнами. После того, как вы инициируете запуск, откроется окно, управлять которым можно так:

```
alert = driver.switch_to_alert()
```

Код вернёт объект текущего открытого окна. С этим объектом вы можете принять или отклонить вопрос окна, прочитать его содержимое или ввести текст. Интерфейс взаимодействия со всплывающими окнами работает одинаково хорошо как для предупреждений (alerts), так для запросов к подтверждению (confirms) и приглашений к вводу (prompts). За дополнительной информацией обратитесь к документации API.

Навигация: история и локация

Чуть раньше мы упомянули о навигации по ссылке с использованием команды get (driver.get (www.example.com)). Как вы уже могли заметить, WebDriver для отдельных случаев предоставляет узконаправленные, специализированные интерфейсы взаимодействия, и навигация не исключение. Чтобы перейти по ссылке, вы можете воспользоваться методом get:

```
driver.get("http://www.example.com")
```

Чтобы перейти вперёд или назад по истории вкладки:

```
driver.forward()
driver.back()
```

Имейте в виду, что этот функционал полностью зависит от используемого драйвера. Вы можете получить непредвиденный результат, если привыкли к поведению какого-либо конкретного браузера, а работаете с другим.

Куки (cookies)

Рассмотрим, как использовать куки. Прежде всего, вам необходим домен, использующий их:

```
# Перейти на необходимый домен
driver.get("http://www.example.com")
# Установить куки. Следующий cookie действителен для всего домена
cookie = {"ключ": "значение"}
driver.add_cookie(cookie)
# И теперь получим все доступные куки для текущего адреса URL
all_cookies = driver.get_cookies()
for cookie_name, cookie_value in all_cookies.items():
print("%s -> %s", cookie_name, cookie_value)
```

Поиск элементов

Существует ряд способов поиска элементов на странице. Вы вправе использовать наиболее уместные для конкретных задач. Selenium предоставляет такие методы поиска элементов на странице:

- find_element_by_id;
- find_element_by_name;
- find_element_by_xpath;
- find_element_by_link_text;
- find_element_by_partial_link_text;
- find_element_by_tag_name;
- find_element_by_class_name;
- find_element_by_css_selector.

Чтобы найти все элементы, удовлетворяющие условию поиска, используйте эти методы (возвращается список):

- 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.

Как вы могли заметить, во втором списке нет поиска по id. Это обусловливается особенностью свойства id для элементов HTML: идентификаторы элементов страницы всегда уникальны.

Помимо общедоступных (public) методов, перечисленных выше, есть два приватных (private) метода, которые при знании указателей объектов страницы могут быть очень полезны: find_element и find_elements. Пример использования:

```
from selenium.webdriver.common.by import By
driver.find_element(By.XPATH, '//button[text()="Some text"]')
driver.find_elements(By.XPATH, '//button')
```

Для класса By доступны следующие атрибуты:

- ID = "id";
- XPATH = "xpath";
- LINK_TEXT = "link text";
- PARTIAL_LINK_TEXT = "partial link text";
- NAME = "name";
- TAG_NAME = "tag name";
- CLASS_NAME = "class name";
- CSS_SELECTOR = "css selector";

Поиск по id

Используйте этот способ, когда известен id элемента. Если ни один элемент не удовлетворяет заданному значению id, будет вызвано исключение NoSuchElementException.

Для примера, рассмотрим такой исходный код страницы:

```
<html>
<body>
<form id="loginForm">
<input name="username" type="text" />
<input name="password" type="password" />
<input name="continue" type="submit" value="Login" />
</form>
</body>
<html>
```

Элемент form может быть определён так:

```
login_form = driver.find_element_by_id('loginForm')
```

Поиск по Name

Используйте этот способ, когда когда у элемента известен атрибут name. Результатом будет первый элемент с искомым значением атрибута name. Если ни один элемент не удовлетворяет заданному значению name, будет вызвано исключение NoSuchElementException.

Для примера, рассмотрим исходный код страницы:

```
<html>
 <body>
  <form id="loginForm">
   <input name="username" type="text" />
   <input name="password" type="password" />
   <input name="continue" type="submit" value="Login" />
   <input name="continue" type="button" value="Clear" />
  </form>
 </body>
<html>
```

Элементы с именами username и password могут быть определены так:

```
username = driver.find_element_by_name('username')
password = driver.find_element_by_name('password')
```

Такой код получит кнопку Login, находящуюся перед кнопкой Clear:

```
continue = driver.find_element_by_name('continue')
```

Поиск по XPath

XPath — это язык, использующийся для поиска узлов дерева XML-документа. Поскольку в основе HTML чаще всего структура XML (XHTML), пользователи Selenium могут с помощью этого языка отыскивать элементы в веб-приложениях. XPath выходит за рамки простых методов поиска по атрибутам id или name (и в то же время поддерживает их) и открывает спектр новых возможностей, например, поиск третьего чекбокса (checkbox) на странице.

Одно из веских оснований использовать XPath — ситуации, когда у вас нет пригодных атрибутов, чтобы использовать их в качестве указателей, например id или name для элемента, который вы хотите получить. Вы можете использовать XPath для поиска элемента как по абсолютному пути (не рекомендуется), так и по относительному (для элементов с заданными id или name). XPath-указатели могут быть использованы, в том числе, для определения элементов с помощью атрибутов, отличных от id и name.

Абсолютный путь XPath содержит все узлы дерева от корня (html) до необходимого элемента и, как следствие, подвержен ошибкам в результате малейших корректировок исходного кода страницы. Если найти ближайший элемент с атрибутами id или name (в идеале, один из элементов-родителей),

можно определить искомый элемент, используя связь «родитель-подчинённый». Эти связи будут куда стабильнее и сделают ваши тесты устойчивыми изменениям в исходном коде страницы.

Для примера рассмотрим следующий исходный код страницы:

```
<html>
 <body>
  <form id="loginForm">
   <input name="username" type="text" />
   <input name="password" type="password" />
   <input name="continue" type="submit" value="Login" />
   <input name="continue" type="button" value="Clear" />
  </form>
 </body>
<html>
```

Элемент form может быть определён следующими способами:

1. Абсолютный путь (сломается при малейшем изменении структуры HTML-страницы).

```
login_form = driver.find_element_by_xpath("/html/body/form[1]")
```

2. Первый элемент form в странице HTML.

```
login_form = driver.find_element_by_xpath("//form[1]")
```

3. Элемент form, для которого определён атрибут с именем id и значением loginForm.

```
login_form = driver.find_element_by_xpath("//form[@id='loginForm']")
```

Элемент username может быть найден так:

1. Первый элемент form с дочерним элементом input, для которого определён атрибут с именем name и значением username.

```
username = driver.find_element_by_xpath("//form[input/@name='username']")
```

2. Первый дочерний элемент input элемента form, для которого определён атрибут с именем id и значением loginForm.

```
username = river.find_element_by_xpath("//form[@id='loginForm']/input[1]")
```

3. Первый элемент input, для которого определён атрибут с именем name и значением username.

```
username = driver.find_element_by_xpath("//input[@name='username']")
```

Кнопка Clear может быть найдена следующими способами:

1. Элемент input, для которого заданы атрибут с именем name, значением continue и атрибут с именем type и значением button.

```
clear_button =
driver.find_element_by_xpath("//input[@name='continue'][@type='button']")
```

2. Четвёртый дочерний элемент input элемента form, для которого задан атрибут с именем id и значением loginForm.

```
clear_button = driver.find_element_by_xpath("//form[@id='loginForm']/input[4]")
```

Поиск гиперссылок по тексту гиперссылки

Используйте этот способ, когда известен текст внутри анкер-тега (anchor tag). С помощью такого способа вы получите первый элемент с искомым значением текста тега. Если ни один элемент не удовлетворяет искомому значению, будет вызвано исключение NoSuchElementException.

Для примера, рассмотрим следующий исходный код страницы:

```
<html>
 <body>
  <p>Are you sure you want to do this?</p>
  <a href="continue.html">Continue</a>
  <a href="cancel.html">Cancel</a>
 </body>
<html>
```

Элемент-гиперссылка с адресом continue.html может быть получен так:

```
continue_link = driver.find_element_by_link_text('Continue')
continue_link = driver.find_element_by_partial_link_text('Conti')
```

Поиск элементов по тегу

Используйте этот способ, когда вы хотите найти элемент по его тегу. Таким способом вы получите первый элемент с указанным именем тега. Если поиск не даст результатов, будет вызвано исключение NoSuchElementException.

Для примера рассмотрим следующий исходный код страницы:

```
<html>
 <body>
  <h1>Welcome</h1>
  <p>Site content goes here.</p>
 </body>
```

```
<html>
```

Элемент заголовка h1 может быть найден так:

```
heading1 = driver.find_element_by_tag_name('h1')
```

Поиск элементов по классу

Используйте этот способ в случаях, когда хотите найти элемент по значению атрибута class. Таким способом вы получите первый элемент с искомым именем класса. Если поиск не даст результата, будет вызвано исключение NoSuchElementException.

Для примера рассмотрим исходный код страницы:

```
<html>
 <body>
  <p class="content">Site content goes here.</p>
 </body>
<html>
```

Элемент «p» может быть найден следующим образом:

```
content = driver.find_element_by_class_name('content')
```

Поиск элементов по CSS-селектору

Используйте этот способ, когда хотите получить элемент с использованием синтаксиса CSS-селекторов — это формальное описание относительного пути до элемента/элементов HTML. Обычно селекторы используются для задания правил стиля. В случае с WebDriver, существование самих правил не обязательно, он использует синтаксис CSS только для поиска. Этим способом вы получите первый элемент, удовлетворяющий CSS-селектору. Если ни один элемент ему не удовлетворяет, будет вызвано исключение NoSuchElementException.

Для примера рассмотрим следующий исходный код страницы:

```
<html>
 <body>
  <p class="content">Site content goes here.</p>
 </body>
<html>
```

Элемент «p» может быть определён так:

```
content = driver.find_element_by_css_selector('p.content')
```

Ожидания

В наши дни большинство веб-приложений используют AJAX-технологии. Когда страница загружена в браузере, элементы на ней могут подгружаться с различными временными интервалами. Это затрудняет их поиск, и, если они не присутствуют в DOM, возникает исключение ElementNotVisibleException. Используя ожидания, мы можем решить эту проблему. Оно даёт временной интервал между поиском или любой другой операцией с элементом.

Selenium WebDriver предоставляет два типа ожиданий — неявное (implicit) и явное (explicit). Явное ожидание заставляет WebDriver ожидать возникновение определённого условия до начала действий. Неявное ожидание заставляет WebDriver опрашивать DOM в течение определённого времени в период поиска элемента.

Явные ожидания

Явное ожидание — это код, которым вы определяете, какое необходимое условие должно произойти, чтобы дальнейший код исполнился. Худший пример такого кода — использование команды time.sleep(), которая устанавливает точное время ожидания. Существуют более удобные методы, которые помогут написать код, ожидающий ровно столько, сколько необходимо. WebDriverWait в комбинации с ExpectedCondition — один из таких способов.

```
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
driver = webdriver.Firefox()
driver.get("http://somedomain/url_that_delays_loading")
try:
element = WebDriverWait(driver, 10).until(
EC.presence_of_element_located((By.ID, "myDynamicElement"))
)
finally:
driver.quit()
```

Этот код будет ждать 10 секунд до того, как отдаст исключение TimeoutException или, если найдёт элемент за эти 10 секунд, то вернёт его. WebDriverWait по умолчанию вызывает ExpectedCondition каждые 500 миллисекунд до тех пор, пока не получит успешный return, который для ExpectedCondition имеет тип boolean и возвращает значение true, либо возвращает not null для всех других ExpectedCondition-типов.

Ожидаемые условия. Существуют некие условия, которые часто встречаются при автоматизации веб-сайтов. Ниже перечислены реализации каждого. Связки в Selenium Python предоставляют набор удобных методов, так что вам не придется писать класс expected_condition самостоятельно или создавать собственный пакет утилит.

- 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.

```
from selenium.webdriver.support import expected_conditions as EC
wait = WebDriverWait(driver, 10)
element = wait.until(EC.element_to_be_clickable((By.ID,'someid')))
```

Модуль expected_conditions уже содержит набор предопределённых условий для работы с WebDriverWait.

Неявные ожидания

Неявное ожидание указывает WebDriver опрашивать DOM определённое время, когда пытается найти элемент или элементы, которые недоступны в данный момент. Значение по умолчанию равно 0. Неявное ожидание устанавливает срок жизни объекта WebDriver.

```
from selenium import webdriver
driver = webdriver.Firefox()
driver.implicitly_wait(10) # seconds
driver.get("http://somedomain/url_that_delays_loading")
myDynamicElement = driver.find_element_by_id("myDynamicElement")
```

Ускорение работы

WebDriver можно запускать и без графического интерфейса. Это ускорит работу вашего приложения и снизит нагрузку на систему, а также сэкономит объём загружаемых данных. Для этого:

```
from selenium.webdriver.chrome.options import Options
chrome_options = Options()
chrome_options.add_argument("--headless")  #Режим без интерфейса
driver = webdriver.Chrome(options=chrome_options)
```

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.

Домашнее задание

1. Написать программу, которая собирает входящие письма из своего или тестового почтового ящика, и сложить информацию о письмах в базу данных (от кого, дата отправки, тема письма, текст письма).
2. Написать программу, которая собирает «Хиты продаж» с сайтов техники М.видео, ОНЛАЙН ТРЕЙД и складывает данные в БД. Магазины можно выбрать свои. Главный критерий выбора: динамически загружаемые товары.

Используемая литература

1. The Selenium Browser Automation Project :: Documentation for Selenium
2. https://www.selenium.dev/documentation/en/
3. Open source record and playback test automation for the web
4. Пробный запуск браузера