**Немного про современный веб**

Разработчики хорошо потрудились, чтобы в 2022 году веб-страницы выглядели красиво и быстро открывались, а переходы между страницами были практически незаметны. Страницы сайтов интерактивны и мгновенно реагируют на действия пользователя. Для реализации такого комфортного пользовательского опыта чаще всего используют подход [Single-Page Application](https://en.wikipedia.org/wiki/Single-page_application) (или [одностраничных приложений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)), что в общем случае означает наличие одной страницы на сайте. Содержимое страницы при этом динамически обновляется с помощью JavaScript, который незаметно обменивается с сервером информацией, например, посредством REST API.

В целом все довольны. Разве что создателям автотестов на интерфейсы приходится туго. Неожиданно появляющиеся или пропадающие элементы на странице, непредсказуемое время полной отрисовки страницы, изменяющийся текст в кнопках или в сообщениях веб-сайта — эти особенности работы SPA-приложений приходится учитывать в автотестах, и, стоит признать, это является одним из самых сложных и головоломных аспектов разработки автотестов на Selenium (да и в других фреймворках для написания end-to-end тестов тоже).

В этом уроке мы рассмотрим подробнее самые распространенные проблемы и познакомимся со способами их решения.

## Как работают методы get и find\_element

Разберем еще один простой тест на WebDriver, проверяющий работу кнопки.

Тестовый сценарий выглядит так:

1. Открыть страницу <http://suninjuly.github.io/wait1.html>
2. Нажать на кнопку "Verify"
3. Проверить, что появилась надпись "Verification was successful!"

Для открытия страницы мы используем метод get, затем находим нужную кнопку с помощью одного из методов find\_element\_by\_ и нажимаем на нее с помощью метода click. Далее находим новый элемент с текстом и проверяем соответствие текста на странице ожидаемому тексту.

Вот как выглядит код автотеста:

from selenium import webdriver

from selenium.webdriver.common.by import By

browser = webdriver.Chrome()

browser.get("http://suninjuly.github.io/wait1.html")

button = browser.find\_element(By.ID, "verify")

button.click()

message = browser.find\_element(By.ID, "verify\_message")

assert "successful" in message.text

Попробуйте сначала выполнить тест вручную, а затем запустить автотест. В первом случае, вы завершите тест успешно, во втором случае автотест упадет с сообщением NoSuchElementException для элемента c **id="verify"**. Почему так происходит?

Команды в Python выполняются синхронно, то есть, строго последовательно. Пока не завершится команда get, не начнется поиск кнопки. Пока кнопка не найдена, не будет сделан клик по кнопке и так далее.

Но тест будет работать абсолютно стабильно, только если в данной веб-странице не используется JavaScript (что маловероятно для современного веба). Метод get дожидается информации от браузера о том, что страница загружена, и только после этого наш тест переходит к поиску кнопки. Если страница интерактивная, то браузер будет считать, что страница загружена, при этом продолжат выполняться загруженные браузером скрипты. Скрипт может управлять появлением кнопки на странице и показывать ее, например, с задержкой, чтобы кнопка красиво и медленно возникала на странице. В этом случае наш тест упадет с уже известной нам ошибкой NoSuchElementException, так как в момент выполнения команды button = browser.find\_element(By.ID, "verify") элемент с **id="verify"** еще не отображается на странице. На данной странице пауза перед появлением кнопки установлена на 1 секунду, метод **find\_element()** сделает только одну попытку найти элемент и в случае неудачи уронит наш тест.

## Давайте быстрее это починим: time.sleep()

Теперь, когда мы уже знаем, что кнопка появляется с задержкой, мы можем добавить паузу до начала поиска элемента. Мы уже использовали библиотеку **time** ранее. Давайте применим ее и сейчас:

from selenium import webdriver

from selenium.webdriver.common.by import By

import time

browser = webdriver.Chrome()

browser.get("http://suninjuly.github.io/wait1.html")

time.sleep(1)

button = browser.find\_element(By.ID, "verify")

button.click()

message = browser.find\_element(By.ID, "verify\_message")

assert "successful" in message.text

Теперь тест проходит. Но что если элемент с сообщением тоже будет появляться с задержкой? Добавить еще один **time.sleep()** перед поиском сообщения? А если изменится время задержки при появлении кнопки? Увеличим длительность паузы? А еще на разных машинах с разной скоростью интернета кнопка может появляться через разные промежутки времени. Можно перед каждым действием добавить задержку, но тогда значительную часть времени прогона тестов будут занимать бесполезные ожидания, при этом с увеличением количества тестов эта проблема будет только расти.

## Есть способы получше: Selenium Waits (Implicit Waits)

Надеемся, вы поняли, что решение с**time.sleep()**плохое: оно не масштабируемое и трудно поддерживаемое.

Идеальное решение могло бы быть таким: нам всё равно надо избежать ложного падения тестов из-за асинхронной работы скриптов или задержек от сервера, поэтому мы будем ждать появление элемента на странице в течение заданного количества времени (например, 5 секунд). Проверять наличие элемента будем каждые 500 мс. Как только элемент будет найден, мы сразу перейдем к следующему шагу в тесте. Таким образом, мы сможем получить нужный элемент в идеальном случае сразу, в худшем случае за 5 секунд.

В Selenium WebDriver есть специальный способ организации такого ожидания, который позволяет задать ожидание при инициализации драйвера, чтобы применить его ко всем тестам. Ожидание называется **неявным** (**Implicit wait**), так как его не надо явно указывать каждый раз, когда мы выполняем поиск элементов, оно автоматически будет применяться при вызове каждой последующей команды.

Улучшим наш тест с помощью неявных ожиданий. Для этого нам нужно будет убрать time.sleep() и добавить одну строчку с методом **implicitly wait**:

from selenium import webdriver

from selenium.webdriver.common.by import By

browser = webdriver.Chrome()

# говорим WebDriver искать каждый элемент в течение 5 секунд

browser.implicitly\_wait(5)

browser.get("http://suninjuly.github.io/wait1.html")

button = browser.find\_element(By.ID, "verify")

button.click()

message = browser.find\_element(By.ID, "verify\_message")

assert "successful" in message.text

Теперь мы можем быть уверены, что при небольших задержках в работе сайта наши тесты продолжат работать стабильно. На каждый вызов команды **find\_element** WebDriver будет ждать 5 секунд до появления элемента на странице прежде, чем выбросить исключение **NoSuchElementException**.

## Задание: Про Exceptions

Теперь мы знаем, как настроить ожидание поиска элемента. Во время поиска WebDriver каждые 0.5 секунды проверяет, появился ли нужный элемент в DOM-модели браузера (Document Object Model — «объектная модель документа», интерфейс для доступа к HTML-содержимому сайта). Если произойдет ошибка, то WebDriver выбросит одно из следующих исключений (**exceptions**):

* Если элемент не был найден за отведенное время, то мы получим **NoSuchElementException.**
* Если элемент был найден в момент поиска, но при последующем обращении к элементу DOM изменился, то получим **StaleElementReferenceException**. Например, мы нашли элемент **Кнопка** и через какое-то время решили выполнить с ним уже известный нам метод click. Если кнопка за это время была скрыта скриптом, то метод применять уже бесполезно — элемент "устарел" (stale) и мы увидим исключение.
* Если элемент был найден в момент поиска, но сам элемент невидим (например, имеет нулевые размеры), и реальный пользователь не смог бы с ним взаимодействовать, то получим **ElementNotVisibleException**.

Знание причин появления исключений помогает отлаживать тесты и понимать, где находится баг в случае его возникновения.

**Задание:**

Какую ошибку вы увидите в консоли, если попытаетесь выполнить команду**browser.find\_element(By.ID,** **"button")** после открытия страницы <http://suninjuly.github.io/cats.html>?

Коротко о том, как запомнить кто есть кто:

* NoSuchElementException - нет такого вообще
* StaleElementReferenceException -  был элемент да сплыл
* ElementNotVisibleException - видишь элемент? И я не вижу, а он есть.

## Explicit Waits (WebDriverWait и expected\_conditions)

В предыдущем шаге мы решили проблему с ожиданием элементов на странице. Однако методы **find\_element** проверяют только то, что элемент появился на странице. В то же время элемент может иметь дополнительные свойства, которые могут быть важны для наших тестов. Рассмотрим пример с кнопкой, которая отправляет данные:

* Кнопка может быть неактивной, то есть её нельзя кликнуть;
* Кнопка может содержать текст, который меняется в зависимости от действий пользователя. Например, текст "Отправить" после нажатия кнопки поменяется на "Отправлено";
* Кнопка может быть перекрыта каким-то другим элементом или быть невидимой.

Если мы хотим в тесте кликнуть на кнопку, а она в этот момент неактивна, то WebDriver все равно проэмулирует действие нажатия на кнопку, но данные не будут отправлены.

Давайте попробуем запустить следующий тест:

from selenium import webdriver

from selenium.webdriver.common.by import By

browser = webdriver.Chrome()

# говорим WebDriver ждать все элементы в течение 5 секунд

browser.implicitly\_wait(5)

browser.get("http://suninjuly.github.io/wait2.html")

button = browser.find\_element(By.ID, "verify")

button.click()

message = browser.find\_element(By.ID, "verify\_message")

assert "successful" in message.text

Мы видим, что WebDriver смог найти кнопку с**id="verify"**и кликнуть по ней, но тест упал на поиске элемента "**verify\_message**" с итоговым сообщением:

no such element: Unable to locate element: {"method":"id","selector":"verify\_message"}

Это произошло из-за того, что WebDriver быстро нашел кнопку и кликнул по ней, хотя кнопка была еще неактивной. На странице мы специально задали программно паузу в 1 секунду после загрузки сайта перед активированием кнопки, но неактивная кнопка в момент загрузки — обычное дело для реального сайта.

Чтобы тест был надежным, нам нужно не только найти кнопку на странице, но и дождаться, когда кнопка станет кликабельной. Для реализации подобных ожиданий в Selenium WebDriver существует понятие **явных** ожиданий (**Explicit Waits**), которые позволяют задать специальное ожидание для конкретного элемента. Задание явных ожиданий реализуется с помощью инструментов WebDriverWait и **expected\_conditions**. Улучшим наш тест:

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

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

from selenium import webdriver

browser = webdriver.Chrome()

browser.get("http://suninjuly.github.io/wait2.html")

# говорим Selenium проверять в течение 5 секунд, пока кнопка не станет кликабельной

button = WebDriverWait(browser, 5).until(

EC.element\_to\_be\_clickable((By.ID, "verify"))

)

button.click()

message = browser.find\_element(By.ID, "verify\_message")

assert "successful" in message.text

**element\_to\_be\_clickable**вернет элемент, когда он станет кликабельным, или вернет **False**в ином случае.

Обратите внимание, что в объекте WebDriverWait используется функция **until**, в которую передается правило ожидания, элемент, а также значение, по которому мы будем искать элемент. В модуле **expected\_conditions** есть много других правил, которые позволяют реализовать необходимые ожидания:

* 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
* 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

Описание каждого правила можно найти на [сайте](https://selenium-python.readthedocs.io/api.html#module-selenium.webdriver.support.expected_conditions).

Если мы захотим проверять, что кнопка становится неактивной после отправки данных, то можно задать негативное правило с помощью метода **until\_not**:

# говорим Selenium проверять в течение 5 секунд пока кнопка станет неактивной

button = WebDriverWait(browser, 5).until\_not(

EC.element\_to\_be\_clickable((By.ID, "verify"))

)

## Задание: ждем нужный текст на странице

Попробуем теперь написать программу, которая будет бронировать нам дом для отдыха по строго заданной цене. Более высокая цена нас не устраивает, а по более низкой цене объект успеет забронировать кто-то другой.

В этой задаче вам нужно написать программу, которая будет выполнять следующий сценарий:

1. Открыть страницу <http://suninjuly.github.io/explicit_wait2.html>
2. Дождаться, когда цена дома уменьшится до $100 (ожидание нужно установить не меньше 12 секунд)
3. Нажать на кнопку "Book"
4. Решить уже известную нам математическую задачу (используйте ранее написанный код) и отправить решение

Чтобы определить момент, когда цена аренды уменьшится до $100, используйте метод **text\_to\_be\_present\_in\_element** из библиотеки **expected\_conditions**.

Если все сделано правильно и быстро, то вы увидите окно с числом. Отправьте его в качестве ответа на это задание.

## Итоги урока

В этом уроке мы изучили способы сделать наши тесты менее зависимыми от внешних условий, на которые мы не можем влиять: например, особенности работы JavaScript и непредсказуемая длительность сетевых запросов. Также узнали, почему надо избегать использования **time.sleep()** в автотестах, стали пользоваться неявными и явными ожиданиями WebDriver (**Implicit** и**Explicit Waits**) и научились разбираться в исключениях WebDriver.