

Автоматизация тестирования Web UI на Java

Selenium WebDriver

Часть 2

[Java 11]



На этом уроке

1. Узнаем, как работает класс Actions.
2. Рассмотрим применение JavaScript-кода.
3. Выясним, как работать с окнами.
4. Разберём файлы cookie.

Оглавление

[Класс Actions — сложные взаимодействия](#)

[Исполнение JavaScript-кода](#)

[Подробнее о работе с окнами](#)

[Работа с окнами браузера](#)

[Работа с фреймами](#)

[Работа с файлами cookie](#)

[Тесты](#)

[Практическое задание](#)

[Дополнительная информация](#)

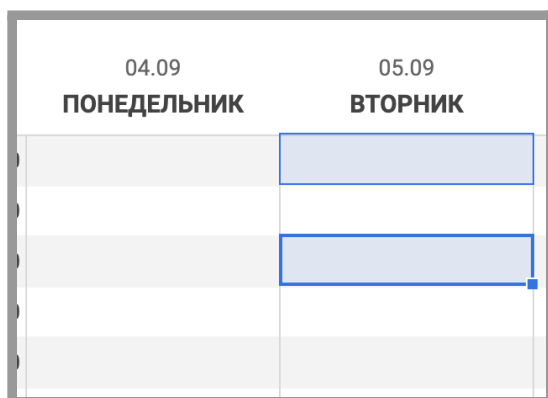
[Используемые источники](#)

Класс Actions — сложные взаимодействия

На прошлом занятии, посвящённом взаимодействию с элементами, мы познакомились с методами `click` и `sendKeys`, но сделали оговорку, что драйвер позволяет выполнять гораздо более интересные действия. Для этих целей есть специальный класс `Actions`, который реализует паттерн «Строитель» для создания комплексных действий, включающих в себя несколько простых.

Рассмотрим пример выделения нескольких строк таблицы. Будучи опытными пользователями, такие действия мы иногда делаем на автомате. Но если разложить процесс на составляющие действия, то получим примерно следующее:

1. Кликнуть левой кнопкой мыши на первую строчку таблицы.
2. Зажать CTRL.
3. Кликнуть левой кнопкой мыши на другую строчку таблицы.
4. Отпустить клавишу CTRL.



04.09 ПОНЕДЕЛЬНИК	05.09 ВТОРНИК

Посмотрим, как выглядит этот пример в коде:

```
// Создадим экземпляр класса Actions
Actions builder = new Actions(driver);

// Построим цепочку действий
builder
    .click(tableRows.get(0)) // кликаем на первую строчку
    .keyDown(Keys.CONTROL)   // зажимаем CTRL
    .click(tableRows.get(2)) // кликаем на другую (третью) строчку
    .keyUp(Keys.CONTROL)     // отпускаем клавишу
    .build()                 // завершаем формирование цепочки действий
    .perform();              // отдаём команду на исполнение
```

При создании любой цепочки действий построение завершается методом `build`. Метод `perform` — это спусковой крючок, после вызова которого собранное действие выполняется.

Класс `Keys`, также использованный в примере выше, служит для представления не буквенно-цифровых клавиш клавиатуры: `CTRL`, `SHIFT`, `FN`, `F1` и т. д.

Рассмотрим основных представителей класса, не задействованных в учебном примере:

```
// Отправка нажатий в текстовое поле
Actions sendKeys(CharSequence... keys)

// Клик с удержанием
Actions clickAndHold(WebElement target)

// Отпустить зажатую клавишу мыши
Actions release()

// Двойной клик
Actions doubleClick(WebElement target)

// Наведение фокуса на элемент
Actions moveToElement(WebElement target)
Actions moveToElement(WebElement target, int xOffset, int yOffset)

// Перемещение фокуса, указывая смещение относительно верхнего левого угла
Actions moveByOffset(int xOffset, int yOffset)

// Клик правой кнопкой мыши
Actions contextClick(WebElement target)
Actions contextClick()

// Перетаскивание объекта
Actions dragAndDrop(WebElement source, WebElement target)
Actions dragAndDropBy(WebElement source, int xOffset, int yOffset)

// Пауза
Actions pause(Duration duration)
```

Тип возвращаемого значения всех методов — `Actions`. Возвращение своего типа в методах характерно для классов, реализующих паттерн «Строитель».

Изучите исходный код класса `Actions`, обратив внимание на реализацию метода `dragAndDrop`.

Исполнение JavaScript-кода

`WebDriver` имеет довольно богатый API, предоставляющий удобный способ взаимодействия с веб-страницами в коде тестов. Тем не менее есть вещи, которые он делать не умеет.

Иногда мы упираемся в ограничение реализации драйвера для конкретного браузера. Например, когда новый инструментарий из стандарта WebDriver ещё не выполнен. Порой API не предоставляет способа получить интересующую нас информацию. Для таких случаев припасено секретное оружие — JavaScript.

Интерфейс `JavaScriptExecutor` представляет собой контракт из двух методов — `executeScript` и `executeAsyncScript`. Эти методы отличаются, как это видно из названия, способом выполнения кода — последовательное и асинхронное. Они (методы) возвращают экземпляр класса `Object`, что обязывает тестировщика делать явное приведение типов в конкретных случаях.

Все распространённые реализации `WebDriver` имплементируют этот интерфейс, так что для использования достаточно привести экземпляр `driver` к требуемому типу:

```
JavaScriptExecutor jsExecutor = (JavaScriptExecutor) driver;  
long windowWidth = (long) jsExecutor.executeScript("return window.innerWidth");
```

Если присмотреться к сигнатурам методов (к слову, аналогичных), то на примере вышеупомянутого метода увидим следующее:

```
Object executeScript(String script, Object... args);
```

Первым аргументом метод принимает строковое представление js-кода, на втором месте — аргументы, передаваемые в скрипт. Конструкция `VarArgs` не обязывает пользователя передавать что-либо в случае отсутствия аргументов, поэтому на все случаи жизни нам хватает одного метода без лишних перегрузок.

```
private boolean isImageLoaded(WebElement image) {  
    JavaScriptExecutor jsExecutor = (JavaScriptExecutor) driver;  
    return (boolean) jsExecutor.executeScript(  
        "return arguments[0].complete && " +  
        "typeof arguments[0].naturalWidth != 'undefined' && " +  
        "arguments[0].naturalWidth > 0", image);  
}
```

Приведённый выше пример иллюстрирует использование JavaScript в коде тестов для выполнения проверок с передачей аргумента в скрипт. Как видно из листинга метода, передавая экземпляр `image` класса `WebElement`, в скрипте не фигурирует название переменной, вместо него — указание индекса в массиве аргументов `arguments[0]`.

`JavaScriptExecutor` не требуется для возвращения значения из веб-страницы. Он полезен, когда надо создать новую вкладку браузера, сделать кастомный скролл внутри страницы или кастомного элемента (например, карусели), взаимодействовать с уведомлениями и так далее. Причина в том, что

любой современный браузер содержит встроенный интерпретатор JavaScript. Использование этой фичи ограничено, по сути, только фантазией.

Подробнее о работе с окнами

Зачастую процесс тестирования происходит в рамках одной веб-страницы. Но API веб-драйвера позволяет покрыть самые разные потребности пользователей, предоставляя удобные способы переключения контекста на вложенные фреймы, алерт-диалоги и даже другие вкладки браузера.

Если изучить исходный код интерфейса WebDriver, можно найти метод с названием `switchTo`, что дословно переводится, как «переключись на...». Рекомендуется изучать исходные коды используемых библиотек, так как все важные методы имеют говорящие названия.

Такой метод возвращает объект, имплементирующий интерфейс `TargetLocator`, который включает следующие методы:

<code>WebDriver frame(int index);</code>
<code>WebDriver frame(String nameOrId);</code>
<code>WebDriver frame(WebElement frameElement);</code>
<code>WebDriver parentFrame();</code>
<code>WebDriver window(String nameOrHandle);</code>
<code>WebDriver defaultContent();</code>
<code>WebElement activeElement();</code>
<code>Alert alert();</code>

Работа с окнами браузера

Начнём с окон. Судя по названиям, из всех сигнатур вышеперечисленных методов только один метод позволяет переключить окно — `window`. На вход метод ожидает получить строковое значение с так называемым дескриптором окна — своеобразным ID, по которому драйвер отдаёт распоряжение браузеру на переключение окна.

А где взять этот ID?

Для ответа на этот вопрос обратимся к методичке урока №3 — раздел «Основные методы драйвера», где есть два важных метода:

<code>Set<String> getWindowHandles(); // возвращает ID всех окон</code>

```
String getWindowHandle(); // возвращает ID текущего окна
```

Итак, чтобы переключиться на нужное окно, требуется знать его дескриптор. Итоговая команда выглядит примерно так:

```
// Получение списка идентификаторов окон
List<String> windowHandles = new ArrayList(driver.getWindowHandles());
// Получение конкретного идентификатора
String secondTab = windowHandles.get(1);

// Переключение
driver.switchTo().window(secondTab);
```

Работа с фреймами

Хотя для пользователя отдельный фрейм выглядит всего лишь чужаковатым элементом интерфейса веб-страницы, обычно их нет в современных сайтах, фреймы используют собственные DOM-деревья. И это не удивительно, ведь, по сути, фрейм представляет собой веб-страницу внутри веб-страницы. В связи с чем веб-драйверу, который ориентируется на DOM, требуется явно передать намерение к использованию другого DOM-дерева.

Возвращаясь к перечню методов интерфейса `TargetLocator`, можно заметить метод `frame` и две его перегрузки, предоставляющие следующий инструментарий:

- переключение в другой фрейм по его индексу;
- переключение в другой фрейм по его атрибуту `name` или `id`;
- переключение в другой фрейм, обернутый в объект `WebElement`.

Логика действий не отличается от того, как надо работать с окнами, не считая, что драйвер не умеет возвращать все `id` или `name` фреймов на странице. Но так как фреймы обозначаются тегами `<frame>` и `<iframe>`, поиск концептуально не отличается от поиска любого другого элемента.

Пример:

```
// Переключение фокуса в первый найденный фрейм
driver.switchTo().frame(0);
```

Если требуется возврат, переключение в родительский фрейм или корень страницы происходит с применением этих методов:

```
driver.switchTo().parentFrame()
```

Или:

```
driver.switchTo().defaultContent()
```

Работа с файлами cookie

В процессе интернет-сёрфинга можно столкнуться со всплывающим окном, оповещающем пользователя, что этот веб-ресурс использует файлы cookie.

После прохождения курса «Тестирование веб-приложений» мы узнали, что эти файлы представляют собой довольно маленькие по размеру фрагменты текста, полученные браузером от сервера и сохранённые в долговременной памяти компьютера. При повторном и последующем посещении веб-страницы, cookie-файлы позволяют «восстановить» пользовательскую сессию. Самый простой пример — корзина интернет-магазина. Если сайт делали грамотные веб-разработчики, закрыв сайт и открыв его снова, мы не потеряем набранную корзину даже без авторизации в личном кабинете.

Тесты

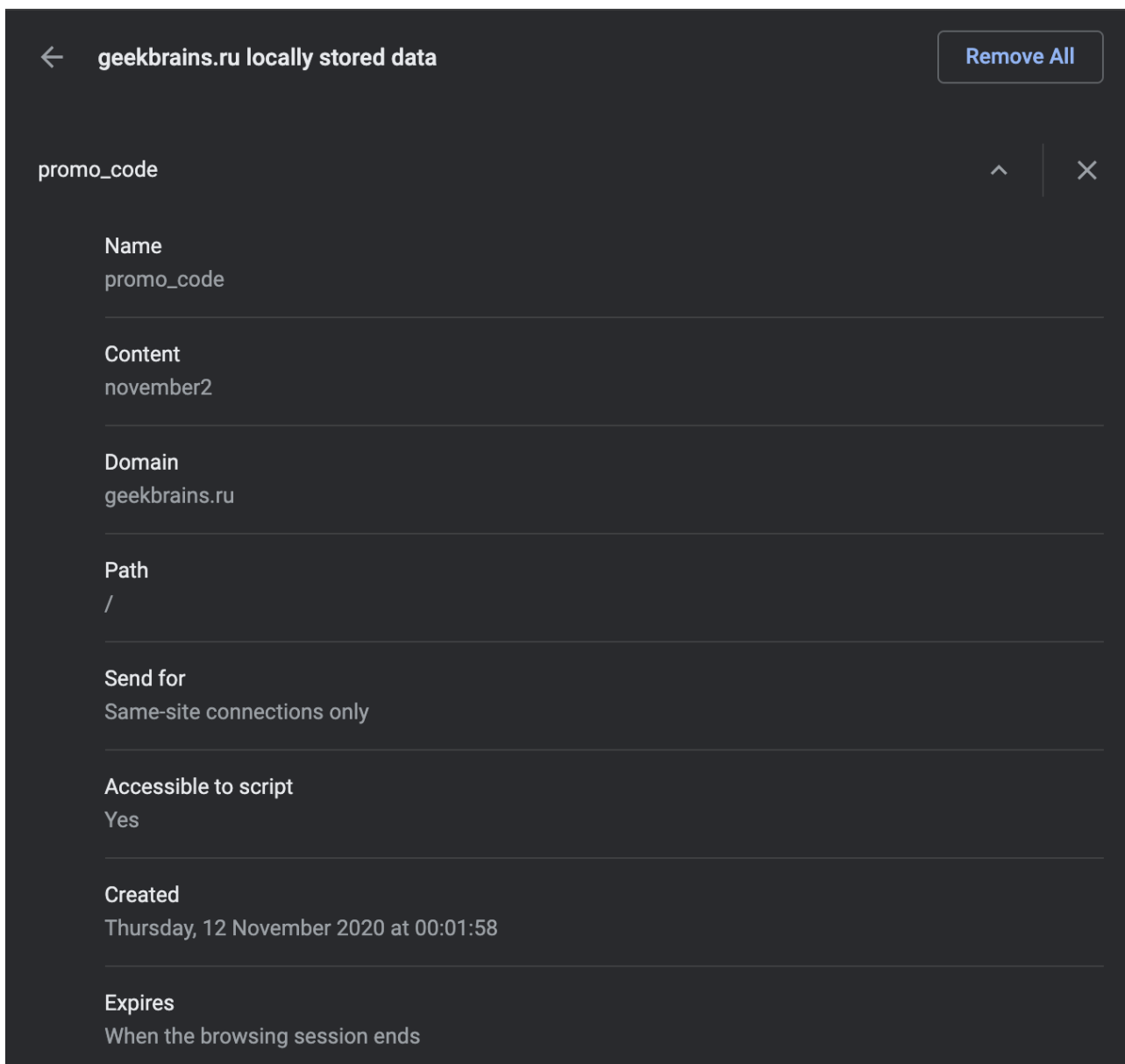
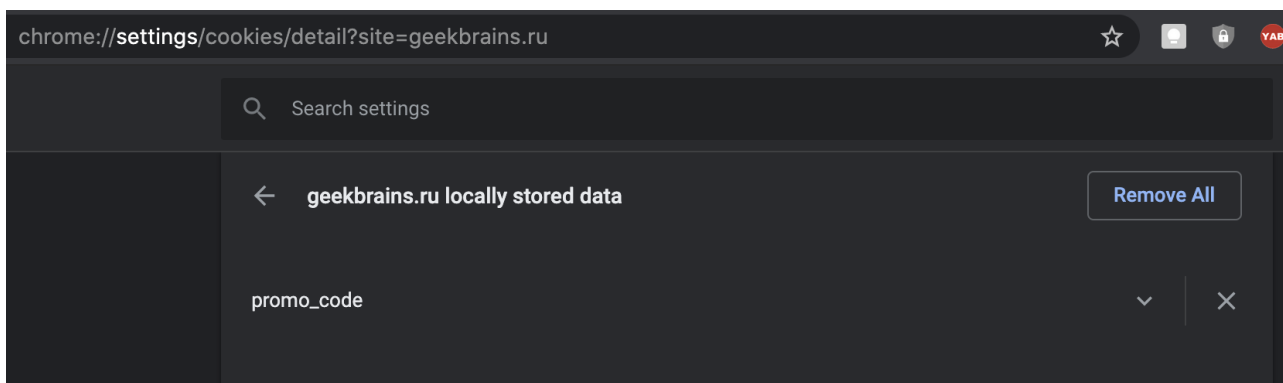
Если наличие или отсутствие тех или иных cookie-файлов может менять логику работы сайта, то их влияние также требуется тестировать. API веб-драйвера позволяет это сделать путём следующих нехитрых методов, позволяющих судя по названиям, получать, добавлять и удалять cookie из браузера.

```
Set<Cookie> getCookies();  
  
Cookie getCookieNamed(String name);  
  
void addCookie(Cookie cookie);  
  
void deleteCookieNamed(String name);  
  
void deleteCookie(Cookie cookie);  
  
void deleteAllCookies();
```

Пример вызова:

```
// Добавление куки файла с названием promo_code и значением november1 на текущую  
// страницу  
driver.get("https://geekbrains.ru/");  
driver.manage().addCookie(new Cookie("promo_code", "november2"));
```

Результат:



Чтобы посмотреть текущие cookie браузера в браузере Chrome, надо перейти в Settings → Cookies and other site data → See all cookies and site data. А именно: «Настройки» → «Конфиденциальность и безопасность» → «Файлы cookie и другие данные сайтов» → «Все файлы cookie и данные сайта».

Практическое задание

1. Перенесите сценарии для CRM в `src/test/java`. Добавьте ассерты.
2. Перенесите сценарии для своего проекта. Добавьте ассерты.
3. Напишите ещё два тест-кейса для своего проекта. Автоматизируйте основные проверки.

Дополнительная информация

1. Статья [Builder](#).

Используемые источники

1. Yujun Liang, Alex Collins. Selenium WebDriver: From Foundations To Framework.
2. Unmesh Gundecha. Selenium Testing Tools Cookbook.
3. [Официальная документация](#).