

Автоматизация тестирования Web UI на Java

# Selenium WebDriver

## Часть 2

[Java 11]

---



# На этом уроке

1. Поговорим о добавлении зависимостей в проект.
2. Рассмотрим принципы работы WebDriver.
3. Разберём типы ожиданий.

## Оглавление

[Добавление зависимостей в проект](#)

[Установка конкретного драйвера](#)

[Использование WebDriverManager](#)

[Поиск элементов](#)

[Использование WebDriver](#)

[Основные методы драйвера](#)

[Взаимодействие с элементами](#)

[Ожидания](#)

[Неявные ожидания](#)

[Явные ожидания](#)

[Практическое задание](#)

[Используемые источники](#)

# Добавление зависимостей в проект

Для управления драйвером из клиентского кода нам нужно подключить требуемую для используемого языка программирования реализацию библиотеки WebDriver.

На [этой странице](#) есть все доступные версии библиотеки.

Так как мы условились использовать сборщик Maven на текущем проекте, добавим в pom.xml, а затем в блок dependencies реквизиты библиотеки selenium-java последней стабильной версии.

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>3.141.59</version>
</dependency>
```

На этом подключение библиотеки selenium считается завершенным. Осталось скачать сам драйвер.

## Установка конкретного драйвера

Рассматривать все тонкости работы драйверов различных браузеров мы не станем, так как эти различия незначительны. Если выбрать наиболее популярный браузер, доступный во всех ОС, то во всех примерах будет использоваться браузер Chrome.






Скачать Google Chrome на официальном [сайте](#).

**Важно!** Перед началом работы требуется обновить текущую версию браузера до актуальной.

Чтобы узнать версию браузера, нужно ввести в адресную строку chrome://version/.

На момент написания этого методического материала актуальная версия Google Chrome — v.86.

Для загрузки драйвера, совместимого с конкретной ОС, нужно перейти на официальный [сайт](#) ChromeDriver.

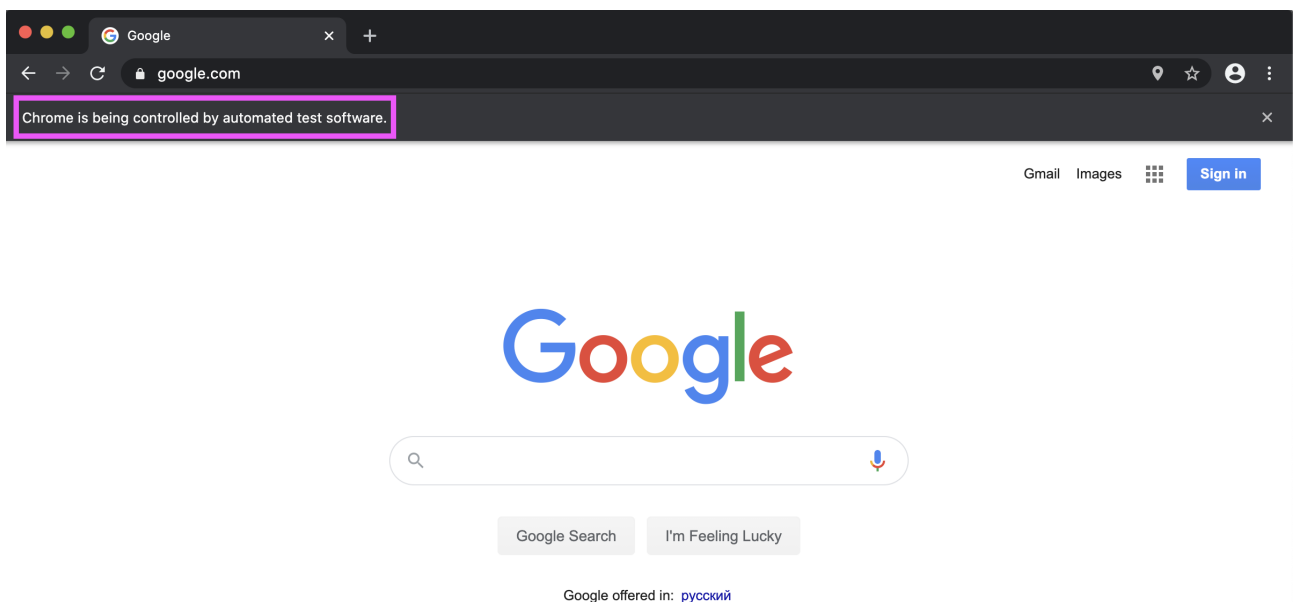
Name	Last modified	Size	ETag
 <a href="#">Parent Directory</a>		-	
 <a href="#">chromedriver_linux64.zip</a>	2020-09-03 19:26:44	5.20MB	0b080bc38261fe200602d8c525931ea7
 <a href="#">chromedriver_mac64.zip</a>	2020-09-03 19:26:45	7.44MB	f9ed96fe96d5a23979f3ae9648643047
 <a href="#">chromedriver_win32.zip</a>	2020-09-03 19:26:47	5.00MB	bfb2a4b657284571b390002086b5eb3
 <a href="#">notes.txt</a>	2020-09-03 19:26:51	0.00MB	c4e583db0d7ff14f4f91760ad364be0d

После распаковки архива переместим файл драйвера в папку resources.

В методе `main` передадим в статический метод `setProperty` класса `System` идентификатор драйвера и его относительный путь. Затем инициализируем новый экземпляр `ChromeDriver` и вызовем метод `get`, передав URL в виде строки.

```
public static void main(String[] args) {  
    System.setProperty(  
        "webdriver.chrome.driver", "src/main/resources/chromedriver"  
    );  
    WebDriver driver = new ChromeDriver();  
    driver.get("https://google.com");  
}
```

Если после запуска появится новое окно Google Chrome с уведомлением, что текущая браузерная сессия находится под контролем программного обеспечения для автоматизированного тестирования — всё получилось.



## Использование WebDriverManager

Способ выше отлично работает, пока используется одна версия конкретного браузера. Однако в реальном мире подобное управление драйверами сулит большими хлопотами.

В 2015 году разработчик [Boni Garcia](#) написал и опубликовал библиотеку [WebDriverManager](#), в настоящий момент активно поддерживаемую сообществом.

Разработчики отлично задокументировали этот проект, этапы подключения и настройки с примерами находятся в файле README на главной странице проекта.

# Поиск элементов

Для поиска элементов в Selenium используются локаторы. Локатор — это строка, уникально идентифицирующая элемент страницы. В библиотеке для языка Java локаторы создаются посредством класса **By** — название неслучайно и соответствует семантике. WebDriver предоставляет несколько способов использования локаторов для поиска элементов.

1. `By.className`.
2. `By.id`.
3. `By.name`.
4. `By.tagName`.
5. `By.linkText`.
6. `By.cssSelector`.
7. `By.xpath`.

Таким образом, чтобы найти этот элемент:

```
<input name="email" class="form-control"/>
```

Используется следующая конструкция:

```
driver.findElement(By.name("email"));
```

Метод **findElement** возвращает экземпляр класса **WebElement** в случае успешного поиска, либо бросает исключение **NoSuchElementException**.

Если задача — это поиск списка элементов, следует использовать метод **findElements**.

**Важно!** Чтобы проверить наличие элемента по указанному локатору без обработки исключительной ситуации, выполняется проверка на длину полученного списка.

```
driver.findElements(By.className("form-control")).size() > 0;  
// true - как минимум один элемент, удовлетворяющий условию, найден
```

Локаторы подробно обсуждались на курсе «Тестирование веб-приложений», поэтому подробно останавливаться на них в рамках этого методического пособия не будем.

Если вы не изучали этот курс, прочитайте дополнительные материалы:

- [Здесь вам нужен раздел XPath](#)
- [Здесь рассказывается про путь к элементам](#)

# Использование WebDriver

## Основные методы драйвера

Интерфейс WebDriver содержит следующие методы для управления браузерной сессией. В приведенной ниже таблице указываются сигнатуры методов и краткое описание назначения. В рамках наших практических занятий мы рассмотрим каждый из этих методов.

<code>Options manage();</code>	Предоставляет интерфейс Options для управления параметрами драйвера: размер окна, ожидания, добавление или удаление cookie-файлов.
<code>void get(String url);</code>	Загружает веб-страницу в текущем окне.
<code>String getCurrentUrl();</code>	Возвращает текущий URL.
<code>WebElement findElement(By by);</code>	Возвращает WebElement.
<code>List&lt;WebElement&gt; findElements(By by);</code>	Возвращает список WebElements.
<code>String getTitle();</code>	Возвращает заголовок текущей страницы.
<code>Set&lt;String&gt; getWindowHandles();</code>	Возвращает множество дескрипторов открытых окон браузера. Используется и для итерирования.
<code>TargetLocator switchTo();</code>	Переключение на другой Frame или Window.
<code>void close();</code>	Закрывает текущее окно.
<code>void quit();</code>	Закрывает браузер.

## Взаимодействие с элементами

Пользователь — тот, чью работу с веб-страницей мы собираемся имитировать. С его точки зрения, наиболее часто встречаемые действия:

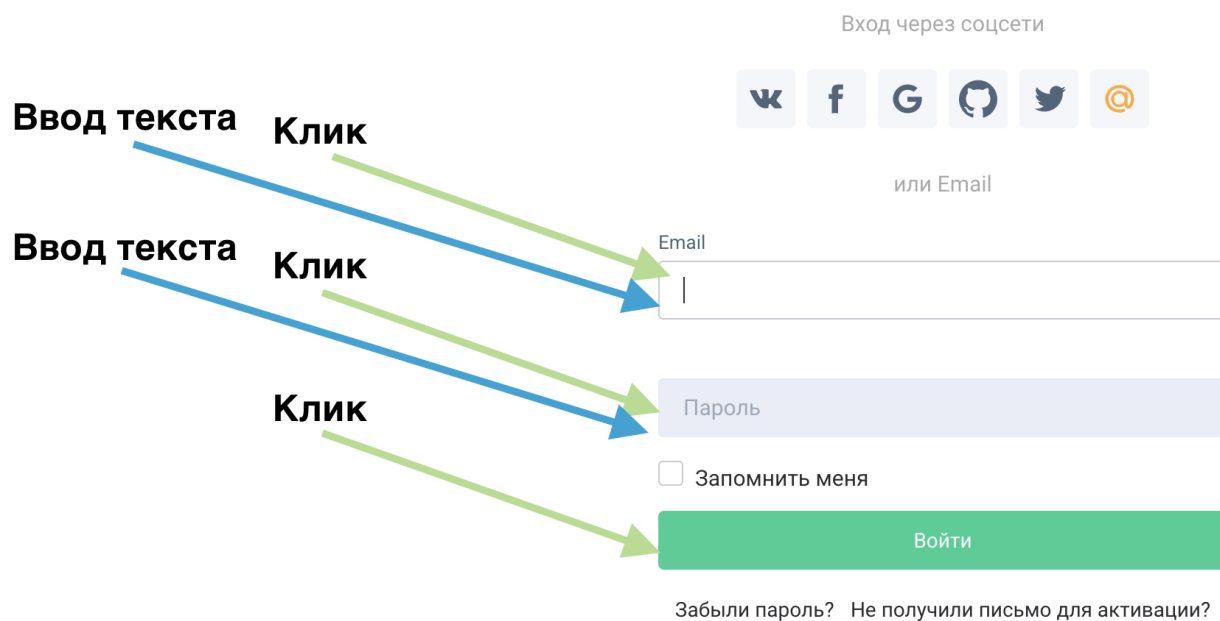
- клики — по кнопкам, чекбоксам, полям ввода;
- набор текста в поле ввода.

Для удовлетворения этих целей интерфейс WebElement содержит следующие методы:

```
void sendKeys(CharSequence... keysToSend);  
void click();
```

Рассмотрим гипотетический вариант использования этих методов на примере экрана авторизации на сайте нашей платформы:

## Вход



```
// Клик и последующий ввод  
emailInputField.click();  
emailInputField.sendKeys("student001@mail.ru");  
  
// Клик и последующий ввод  
passwordInputField.click();  
passwordInputField.sendKeys("foobar");  
  
// Клик  
loginButton.click();
```

Неужели такой мощный инструмент как Selenium умеет только отправлять клики и текст в поля ввода? Разумеется, нет. API Selenium содержит методы для имитации простых и сложных действий пользователя — двойной клик, движения курсором, перетаскивания объектов (drag-n-drop) и т. д. В следующем уроке мы разберём эту тему.

# Ожидания

Вследствие неустойчивого интернет-соединения или долгого ответа от сервера, нужная нам страница или элемент во время выполнения скрипта могут подгрузиться не сразу. Это может и, скорее всего, вызвать падение программы с **NoSuchElementException**.

С точки зрения программирования, простейшее решение — приостановка выполнения программы методом **Thread.sleep()**. Однако этот метод несёт в себе больше недостатков, чем достоинств. Даже если приостановить исполнение на 5 секунд, нет гарантии, что элемент не загрузится через 6. А если элемент загрузится и обнаружится мгновенно, мы вхолостую ждём эти секунды. Они в масштабах сессии регрессионного тестирования из нескольких тысяч тестов дадут нам лишние десятки минут ожидания.

К счастью для нас, корректные способы обработки таких ситуаций давно появились.

## Неявные ожидания

Неявные ожидания конфигурируют экземпляр драйвера периодически проверять наличие искомого элемента в течение обозначенного времени без выброса исключений.

```
driver.manage().timeouts().implicitlyWait(3, TimeUnit.SECONDS);
```

Здесь драйвер конфигурируется на 3 секунды ожидания. Это значит, что везде по ходу выполнения программы драйвер будет ожидать появления элемента 3 секунды и некоторое количество миллисекунд опрашивать страницу на предмет наличия искомого. Если элемент найдётся, ожидание остановится. В этом заключается ключевое отличие ожиданий драйвера от метода **sleep** класса **Thread**.

Неявные ожидания обычно настраиваются сразу после создания экземпляра **WebDriver**. Они действуют в течение всей жизни этого экземпляра, но переопределить их можно в любой момент.

## Явные ожидания

Явные ожидания — **Explicit Waits** — это код, который ждёт наступления какого-то события, прежде чем продолжит выполнение команд скрипта. Такое ожидание срабатывает один раз в указанном месте.

```
new WebDriverWait(driver, 5).until(ExpectedConditions.urlContains("/create"));
```

Пример выше в течение 5 секунд ожидает появления строки “/create” в URL текущей страницы.

Рассмотрим его более детально:



1. `new WebDriverWait(driver, 5)` — создание нового экземпляра класса `WebDriverWait`. В качестве параметров конструктора класс принимает текущий экземпляр драйвера и количество секунд для ожидания.
2. `.until(ExpectedConditions.urlContains("/create"))` — вызов метода `until` (до тех пор), принимающего в качестве параметра `ExpectedConditions` — статический класс, содержащий часто используемые условия для ожидания.

Наиболее употребляемые параметры класса `ExpectedConditions` позволяют писать самые разные условия, основанные на видимости, невидимости или кликабельности элемента, на появлении требуемого текста, изменении заголовка страницы, её URL и т. д.

- `titleContains(String title);`
- `presenceOfElementLocated(By locator);`
- `presenceOfAllElementsLocatedBy(By locator);`
- `visibilityOfElementLocated(By locator);`
- `visibilityOf(WebElement element);`
- `textToBePresentInElement(By locator, String text);`
- `invisibilityOfElementLocated(By locator);`
- `elementToBeClickable(By locator).`

## Практическое задание

1. Перенесите два сценария для CRM и два сценария из своего проекта из Selenium IDE в код проектов.
2. Сдайте ссылки на оба репозитория преподавателю.

## Используемые источники

1. Yujun Liang & Alex Collins «Selenium WebDriver: From Foundations to Framework».
2. [Официальная документация.](#)