Плохой софт **не имеет** документации. Отличный софт **не нуждается** в документа

Мы с гордостью заявляем, что Selenide настолько прост, что не нужно читать тонны документации, чтобы начать с ним
Вся работа с Selenide состоит всего из трёх простых шагов:

Три простых шага:

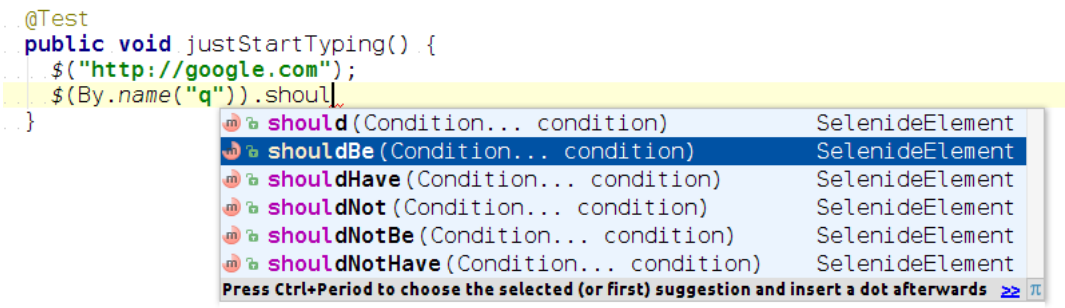
1. открыть(страницу)
2. $(элемент).совершитьДействие()
3. $(элемент).проверитьУсловие()

```
open("/login");
$("#submit").click();
$(".message").shouldHave(text("Привет"));
```

Используй мощь IDE!

Весь Selenide API состоит из небольшого числа классов. Мы предлагаем прекратить читать, открыть твою любимую IDE и просто начать печатать.

Просто набери: `$(selector).` - и IDE предложит все возможные опции.



Используй всю мощь современных средств разработки вместо того, чтобы забивать себе голову документацией!

Selenide API

Здесь можно найти Selenide javadoc.

Для справки, ниже приведены некоторые классы Selenide с описанием основных определенных в них методов, которые тебе, возможно, понадобятся:

com.codeborne.selenide.Selenide [src] [javadoc]

Ядро библиотеки Selenide. Основные методы - это `open`, `$` и `$$`:

- open(URL)
- $(String cssSelector) - возвращает объект типа SelenideElement, который представляет первый найденный по CSS селектору элемент на странице
- $(By) - возвращает "первый SelenideElement" по локатору типа By
- $$(String cssSelector) - возвращает объект типа ElementsCollection, который представляет коллекцию всех элементов найденных по CSS селектру
- $$(By) - возвращает "коллекцию элементов" по локатору типа By

Обычно, когда ты получаешь с помощью доллара объект SelenideElement, ты можешь либо совершить с ним какое-то действие:

- `$(byText("Sign in")).click();`

и даже несколько действий сразу:

- `$(byName("password")).setValue("qwerty").pressEnter();`

либо проверить какое-то условие:

- `$(".wellcome-message").shouldHave(text("Welcome, user!"))`.

“Два доллара” же может быть удобно использовать когда нужный элемент является одним из группы однотипных элементов. Например вместо:

```
$(byXpath("//*[@id='search-results']//a[contains(text(),'selenide.org')]")).click();
```

можно использовать более читабельный и лаконичный вариант:

```
$$("#search-results a").findBy(text("selenide.org")).click();
```

Большинство операций над элементами полученными с помощью `$` и `$$` имеют встроенные неявные ожидания в зависимости от контекста. Это позволяет в большинстве случаев не отвлекаться на явные ожидания загрузки элементов при тестировании динамических веб приложений.

Не стесняйся поискать и найти намного больше методов внутри класса Selenide, которые могут тебе понадобиться;), просто набрав в любимом IDE `Selenide.`,

Вот лишь несколько примеров: sleep(long milliseconds), refresh(), title(), executeJavaScript(String jsCode, Object... arguments).

Более подробная информация доступна в Selenide gitbook

com.codeborne.selenide.SelenideElement [src] [javadoc]

Класс SelenideElement - описывает элемент найденный на странице. Его обьект можно например получить с помощью команды `$`. В классе описанны следующие полезные методы.

Методы поиска внутренних элементов

- find(String cssSelector) / $(String cssSelector)
- find(By) / $(By)
- findAll(String cssSelector) / $$(String cssSelector)
- findAll(By) / $$(By)

Здесь $ и $$ просто лаконичные “алиасы” (синонимы) для соответствующих команд.

Таким образом, можно пошагово уточнять - какой внутренний элемент необходимо получить внутри внешнего элемента, строя цепочку последовательных вызовов, например:

```
$("#header").find("#menu").findAll(".item")
```

Методы-проверки состояния элемента - assertions

- should(Condition) / shouldBe(Condition) / shouldHave(Condition)
- shouldNot(Condition) / shouldNotBe(Condition) / shouldNotHave(Condition)

Рекомендуется выбирать такой метод, чтобы строка кода легко воспринималась, как обычная фраза, например:

```
$("input").should(exist);
$("input").shouldBe(visible);
$("input").shouldHave(exactText("Some text"));
```

Проверки играют роль явных ожиданий (explicit waits) в Selenide. Они **ждут** до удовлетворения условия (visible, enabled, text(“some text”)) пока не истечет таймаут (значение `Configuration.timeout`, которое

установлено по умолчанию в 4000 миллисекунд).

Можно использовать проверки явно - с целью ожиданий нужного состояния у элементов перед действием, например: `$("#submit").shouldBe(enabled).click();`

Есть версии явных ожиданий с указанием таймаута:

- waitUntil(Condition, milliseconds)
- waitWhile(Condition, milliseconds)

Методы-действия над элементом

- click()
- doubleClick()
- contextClick()
- hover()
- setValue(String) / val(String)
- pressEnter()
- pressEscape()
- pressTab()
- selectRadio(String value)
- selectOption(String)
- append(String)
- dragAndDropTo(String)
- …

Большинство действий также возвращают обьект SelenideElement, позволяя использовать цепочки вызовов, например: `$("#edit").setValue("text").pressEnter();` .

Методы получения статусов элементов и значений их атрибутов

- getValue() / val()
- data()
- attr(String)
- text() // возвращает “видимый текст на странице”
- innerText() // возвращает “текст элемента в DOM”
- getSelectedOption()
- getSelectedText()
- getSelectedValue()
- isDisplayed() //возвращает false, если элемент либо невидимый, либо его нет в DOM
- exists() //возвращает true, если элемент есть в DOM, иначе - false

Другие полезные команды

- uploadFromClasspath(String fileName)
- download()
- toWebElement()
- uploadFile(File…)

Более подробная информация доступна в Selenide gitbook

com.codeborne.selenide.Condition [src] [javadoc]

Условия используются в конструкциях should / shouldNot / waitUntil / waitWhile. Мы рекомендуем статически импортировать используемые условия, чтобы получить все преимущества читаемого кода.

- visible / appear // e.g. $(“input”).shouldBe(visible)
- present / exist // условия присутствия элемента в DOM
- hidden / disappear // not(visible)
- readonly // e.g. $(“input”).shouldBe(readonly)
- name // e.g. $(“input”).shouldHave(name(“fname”))
- value // e.g. $(“input”).shouldHave(value(“John”))
- type // e.g. $(“#input”).shouldHave(type(“checkbox”))
- id // e.g. $(“#input”).shouldHave(id(“myForm”))
- empty // e.g. $(“h2”).shouldBe(empty)

- attribute(name) // e.g. $("#input").shouldHave(attribute("required"))
- attribute(name, value) // e.g. $("#list li").shouldHave(attribute("class", "active checked"))
- cssClass(String) // e.g. $("#list li").shouldHave(cssClass("checked"))
- focused
- enabled
- disabled
- selected
- matchText(String regex)
- text(String substring)
- exactText(String wholeText)
- textCaseSensitive(String substring)
- exactTextCaseSensitive(String wholeText)

Более подробная информация доступна в Selenide gitbook

com.codeborne.selenide.Selectors [src] [javadoc]

Класс содержит некоторые `By` селекторы для поиска элементов по тексту или атрибуту (которых не хватает в стандартном Selenium WebDriver API):

- byText - поиск элемента по точному тексту
- withText - поиск элемента по тексту (подстроке)
- by - поиск элемента по атрибуту
- byTitle - поиск по атрибуту "title"
- byValue - поиск по атрибуту "value"

```
// Пример использования:
$(byText("Login")).shouldBe(visible);
$(By.xpath("//div[text()='Login']")).shouldBe(visible); // можно использовать любой org.openqa.seleniu
$(byXpath("//div[text()='Login']")).shouldBe(visible); // или его аналог из Selectors
```

Более подробная информация доступна в Selenide gitbook

com.codeborne.selenide.ElementsCollection [src] [javadoc]

Класс ElementsCollection - описывает коллекцию элементов на странице, которую обычно можно получить с помощью вызова `$$` . Класс предоставляет набор весьма полезных методов.

Методы-проверки - assertions

- shouldBe - e.g. `$$(".errors").shouldBe(empty)`
- shouldHave - e.g. `$$("#mytable tbody tr").shouldHave(size(2))`

Проверки играют роль явных ожиданий (explicit waits). Они **ждут** до удовлетворения условия (size, empty) пока не истечет таймаут (значение `Configuration.collectionsTimeout` , которое установлено по умолчанию в 6000 миллисекунд).

Методы получения статусов и значений коллекции элементов

- size()
- isEmpty()
- getTexts() // возвращает массив видимых текстов, e.g. для элементов `<li>a</li><li hidden>b</li><li>c</li>` вернет массив вида ["a", "", "c"]

Методы выборки внутренних элементов

- filterBy(Condition) возвращает коллекцию (как ElementsCollection) из только тех элементов оригинальной коллекции, которые удовлетворяют условию - например `$$("#multirowTable tr").filterBy(text("Norris"))`
- excludeWith(Condition) - например `$$("#multirowTable tr").excludeWith(text("Chuck"))`
- get(int) - возвращает n-ый элемент как `SelenideElement`
- findBy(Condition) - возвращает элемент коллекции как `SelenideElement` который удовлетворяет условию

Более подробная информация доступна в Selenide gitbook

com.codeborne.selenide.CollectionCondition [src] [javadoc]

Условия используются в конструкциях shouldBe / shouldHave для коллекции - объекта ElementsCollection. Рекомендуется статически импортировать используемые условия, чтобы получить все преимущества читаемого кода.

- empty // e.g. $$("#list li").shouldBe(empty)
- size(int) // e.g. $$("#list li").shouldHave(size(10))
- sizeGreaterThan(int)
- sizeGreaterThanOrEqual(int)
- sizeLessThan(int)
- sizeLessThanOrEqual(int)
- sizeNotEqual(int)
- texts(String... substrings)
- exactTexts(String... wholeTexts)

Более подробная информация доступна в Selenide gitbook

com.codeborne.selenide.WebDriverRunner [src] [javadoc]

Этот класс содержит некоторые функции, относящиеся к браузеру:

- isChrome()
- isFirefox()
- isHeadless()
- url() - возвращает текущий url
- source() - возвращает исходный HTML текущего окна
- getWebDriver() - возвращает обьект WebDriver (созданный Selenide автоматически или установленный пользователем), таким образом отдавая "чистый" интерфейс к Selenium, если нужно
- setWebDriver(WebDriver) - указывает Selenide использовать драйвер, созданный пользователем. С этого момента пользователь сам ответственен за закрытие драйвера.

Более подробная информация доступна в Selenide gitbook

com.codeborne.selenide.Configuration [src] [javadoc]

Этот класс содержит конфигурации для запуска тестов, например:

- timeout - время ожидания в миллисекундах, которое используется в явных (should/shouldNot/waitUntil/waitWhile) и неявных ожиданиях для SelenideElement, может быть изменено во время исполнения, e.g. `Configuration.timeout = 6000;`
- collectionsTimeout - время ожидания в миллисекундах, которое используется в явных (should/shouldNot) ожиданиях для ElementsCollection, может быть изменено во время исполнения, e.g. `Configuration.collectionsTimeout = 8000;`
- browser (e.g. `"chrome"` , `"ie"` , `"firefox"`)
- baseUrl
- reportsFolder
- ...

Также можно передать конфигурационные параметры как system properties для использования в CI (continuous integration) задачах (например `-Dselenide.baseUrl=http://staging-server.com/start`)

Более подробная информация доступна в Selenide gitbook

Оставайся на связи!

The project is maintained by

well-crafted software