Лабораторная работа №2

### Практическое применение техник тест-дизайна при разработке тест-кейсов

*Цель работы:* научиться практическому применению техник тест-дизайна при формировании тест-кейсов

*Отчет по лабораторной работе:* 6 тест-кейсов, составленных в Selenium IDE

## Теоретическая часть

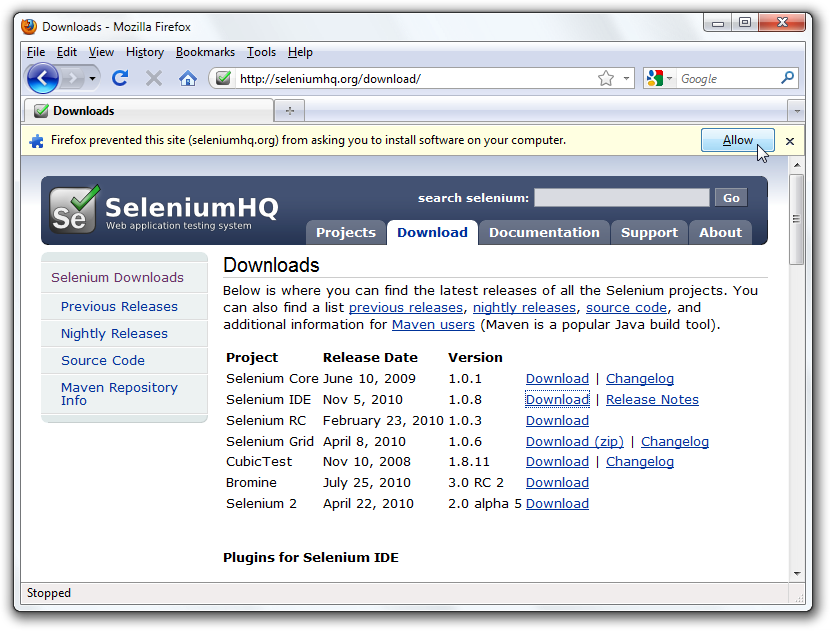
Для разработки тест-кейсов и их непосредственного выполнения на веб-сайте будет использоваться браузер Firefox с установленным плагином Selenuim IDE.

Selenium IDE (Integrated Development Environment, интегрированная среда разработки) — это инструмент, используемый для разработки тестовых сценариев. Он представляет собой простое в использовании дополнение к браузеру Firefox и, в целом, является наиболее эффективным способом разработки тестовых сценариев. Дополнение среди прочего содержит контекстное меню, позволяющее пользователю сначала выбрать любой элемент интерфейса на отображаемой браузером в данный момент странице, а затем выбрать команду из списка команд Selenium с параметрами, предустановленными в соответствии с выбранным элементом. Это не только экономит время, но и дает замечательную возможность для изучения языка команд Selenium.

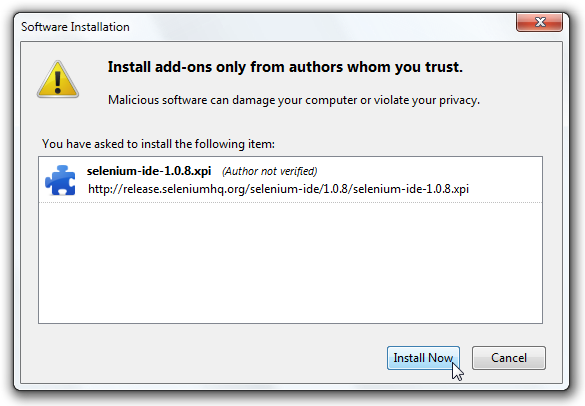
## Практическая часть

#### Установка IDE

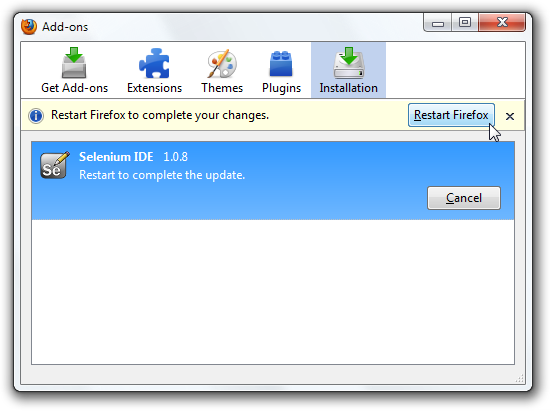
Запустите Firefox и скачайте IDE с [веб-сайта SeleniumHQ](http://seleniumhq.org/download/). Firefox предостерегает вас от установки дополнений с незнакомых ресурсов, потому для продолжения установки нажмите “Разрешить”, как показано на скриншоте.



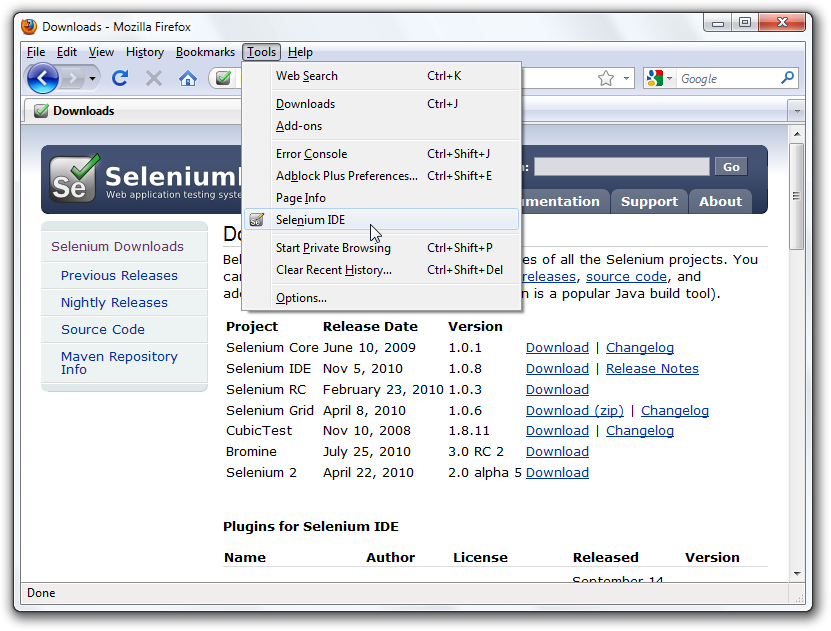
При загрузке дополнения через Firefox появится следующее окно.



Выберите “Установить сейчас”. Появится окно дополнений Firefox, сначала оно будет показывать индикатор выполнения, а после окончания скачивания появится следующее сообщение.

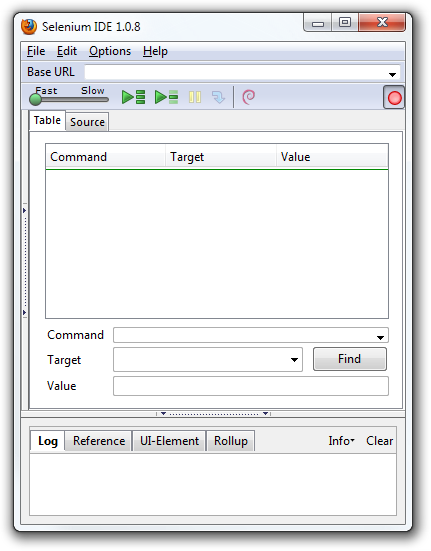


Перезапустите Firefox. После перезапуска Selenium IDE появится в меню “Веб-разработка”.



#### Запуск IDE

Чтобы запустить Selenium IDE, просто выберите его из меню “Веб-разработка” браузера Firefox. Дополнение откроет пустое окно, предназначенное для редактирования тестовых сценарием, а также меню для их загрузки или сохранения.



#### Возможности IDE

##### Панель меню

Меню “File” (“Файл”) имеет разделы “Test Case” (“Тестовый сценарий”) и “Test Suite” (“Набор тестов”). Эти разделы позволяют создать, открыть, сохранить или экспортировать тестовый сценарий в код на любом желаемом языке программирования. Также имеется возможность открыть недавно использованный тестовый сценарий. Аналогичные действия можно совершить и с набором тестов.

Меню “Редактирование” (“Edit”) позволяет выполнять операции копирования, вставки, удаления, отмены изменений и выбора всего набора команд вашего тестового сценария.

Меню “Опции” (“Options”) позволяет изменить настройки дополнения. В этом меню вы можете установить тайм-аут для определенных команд, добавить пользовательские расширения основного набора команд Selenium, а также настроить формат (язык), который будет использоваться для сохранения ваших тестовых сценариев в будущем.

Меню “Помощь” – это стандартное меню справки по Firefox, только опция “UI-Element Documentation” (“Документация по UI-Element”) относится к Selenium IDE.

Панель инструментов (Toolbar)

На панели инструментов находятся кнопки, с помощью которых можно управлять выполнением тестовых сценариев, в том числе пошаговым выполнением для отладки. ../_images/chapt3_img06_IDE_features1.png

Крайняя правая кнопка, на которой изображена красная точка – это кнопка записи.

speed control

“Управление скоростью”: позволяет контролировать скорость выполнения тестового сценария.

run all

“Запустить все”: Запускает весь набор тестов, в случае, если он содержит несколько тестовых сценариев.

run

“Запустить”: Запускает выбранный в данный момент тест. В случае, когда загружен только один тест, кнопки “Запустить” и “Запустить все” работают идентично.

pauseresume

“Пауза”/”Возобновить”: Останавливают и возобновляют исполнение текущего тестового сценария.

step

“Шаг”: Позволяет выполнять тест “по шагам”, то есть по одной команде за раз. Предназначено для отладки тестовых сценариев.

testrunner

Режим “TestRunner”: Эта команда позволяет запустить тестовый сценарий в браузере, используя Selenium Core TestRunner. TestRunner практически не используется в настоящее время, и его поддержка, скорее всего, будет прекращена. Данная кнопка нужна для оценки тестовых сценариев на обратную совместимость с TestRunner. Большинству пользователей, скорее всего, она не понадобится.

rollup

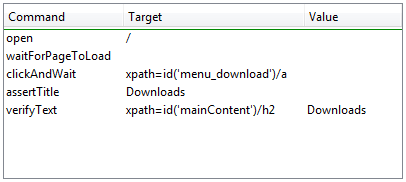
“Применить правила группировки”: Эта продвинутая функция дает возможность группировки повторяющихся последовательностей команд Selenium в одно действие. Более подробную информацию о правилах группирования можно найти в “Документации по расширению UI-Element” в разделе “Помощь”.

record

“Запись”: Записывает действия пользователя в браузере.

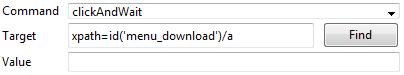
##### Панель тестового сценария

В этой панели отображается набор команд Selenium, составляющих тестовый сценарий. На ней расположены две вкладки, первая из которых, “Table” (“Таблица”), отображает команды и их параметры в удобном для восприятия табличном виде.



Вторая, “Source” (“Исходный код”), отображает тестовый сценарий в формате, используемом для сохранения файла. По умолчанию это HTML, однако он может быть изменен на язык программирования, такой как Java или C#, или же на скриптовый язык, такой как Python. Более подробную информацию вы найдете в меню “Options” (“Опции”). Вкладка “Source” (“Исходный код”) позволяет пользователю редактировать тестовый сценарий в текстовой форме, в том числе используя операции копирования, вырезания и вставки.

Поля ввода данных “Command” (“Команда”), “Target” (“Цель”) и “Value” (“Значение”) отображают выбранную в данный момент команду, а также ее параметры. С помощью этих полей можно модифицировать выбранную команду. Значение первого параметра, описанного во вкладке “Reference” (“Справка”) нижней панели, указывается в поле “Цель”. Если в “Справке” описан также второй параметр, то он всегда указывается в поле “Значение”.



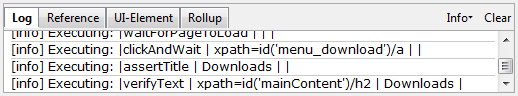
Когда вы начинаете вводить текст в поле “Команда”, появляется выпадающее меню, содержащее команды, начинающиеся с уже введенных символов и вы можете выбрать из этого списка нужную.

Вкладки “Log”, “Reference”, “UI-Element”, “Rollup” (“Лог”, “Справка”, “UI-Element”, “Группировка”)

Нижняя панель используется для четырёх различных функций: лога, справки, документациии по UI-Element и группирования — в зависимости от того, какая вкладка выбрана.

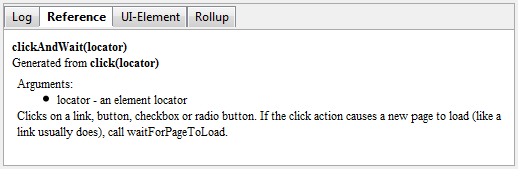
##### Лог

Когда пользователь запускает тестовый сценарий, в панели данной вкладки автоматически появляются информационные сообщения и уведомления об ошибках, даже если пользователь не выбрал вкладку “Лог” до этого. Эти сообщения часто полезны при отладке тестового сценария. Обратите внимание на кнопку “Очистить” (“Clear”), которая позволяет очистить лог. Также обратите внимание на то, что кнопка “Информация” (“Info”) является выпадающим меню, в котором можно выбрать различную степень детализации записи в лог.



##### Справка

Вкладка “Справка” выбирается по умолчанию каждый раз, когда пользователь вводит или модифицирует команды и параметры в табличном режиме. В этом режиме панель “Справка” отображает информацию о текущей команде. При вводе и изменении команд в табличном режиме или режиме правки исходного кода крайне важно удостовериться, что параметры, указанные в полях “Цель” и “Значение”, соответствуют указанным в списке параметров на панели “Справка”. Количество, порядок и типы указанных параметров должны соответствовать количеству, порядку и типам параметров,которые являются допустимыми для данной команды. В случае несоответствия любого из трех пунктов команда не будет выполнена корректно.



Несмотря на то, что “Справка” неоценима в качестве “шпаргалки”, зачастую пользователю следует обращаться к справочной [документации](http://release.seleniumhq.org/selenium-core/1.0.1/reference.html) Selenium.

#### Создание тестовых сценариев

Существует три основных способа разработки тестовых сценариев. Зачастую вам придется использовать все три.

##### Запись

Большинство начинающих пользователей начинают создание тестовых сценариев с записи своего взаимодействия с веб-сайтом. При первом запуске Selenium IDE запись по умолчанию включена. Если вы не хотите, чтобы Selenium IDE начинал запись автоматически, необходимо изменить настройки записи. Для этого нужно выбрать “Опции” > “Опции...” и снять флажок с пункта “Start recording immediately on open” (“Начинать запись при запуске”).

Во время записи Selenium IDE автоматически вставляет команды в тестовый сценарий, основываясь на действиях пользователя. Обычно это команды:

* при нажатии на ссылку – команды click или clickAndWait
* при вводе данных – команда type
* при выборе опции из выпадающего списка – команда select
* при нажатии на чекбокс или переключатель – команда click

Вот некоторые неочевидные вещи, о которых следует знать:

* Для того, чтобы команда type записалась, может потребоваться кликнуть где-нибудь на странице.
* Клик по ссылке обычно записывает команду click. Пользователю в большинстве случаев будет необходимо изменить ее на команду clickAndWait, чтобы удостовериться, что тестовый сценарий дождется загрузки новой страницы. В противном случае Selenium продолжит выполнять команды до того, как загрузятся все элементы пользовательского интерфейса. Это может привести к неожиданным сбоям тестового сценария.

##### Добавление проверок через контекстное меню

В тестовых сценариях бывает необходимо выполнить проверку параметров веб-страницы. Для этого необходимы командыassert и verify. Здесь мы лишь рассмотрим способ их добавления в тестовый сценарий.

При включенном в Selenium IDE режиме записи, переключитесь на браузер с тестируемым веб-приложением и щелкните правой кнопкой мыши в любом месте на странице. Появится контекстное меню с командами verify и/или assert.

При первом использовании Selenium, вероятно, в списке будет только одна команда. Однако, в процессе работы с IDE новые команды добавятся в этот список довольно быстро. Selenium IDE будет пытаться предугадать нужную команду и ее параметры, основываясь на выбранном пользователем элементе интерфейса.

Давайте посмотрим, как это работает. Откройте веб-страницу и выберите кусок текста. Для данной цели прекрасно подойдут заголовок или абзац. Кликните правой клавишей мыши на выбранном тексте. В контекстном меню должны появиться команды verifyTextPresent, а предложенным параметром будет сам текст.

Также обратите внимание на опцию “Show All Available Commands” (“Показать все доступные команды”). Ее выбор открывает список со множеством команд с параметрами, рекомендованными для проверки выбранного элемента.

Поэкспериментируйте с другими элементами интерфейса. Попробуйте кликнуть правой клавишей мыши на изображении или элементе управления, например, на кнопке или чекбоксе. Вам может понадобиться выбрать “Показать все доступные команды”, чтобы увидеть прочие команды, кроме verifyTextPresent. Регулярно выбирая команды из списка всех команд, вы увидите, что наиболее часто используемые появятся в контекстном меню на верхнем уровне. Например, после выбораverifyElementPresent для изображения, в следующий раз, когда вы кликнете правой клавишей мыши на изображении, эта команда будет предложена в контекстном меню на верхнем уровне.

##### Вставка команды

###### В табличном виде

Выберите место в тестовом сценарии, куда вы хотите вставить команду. Чтобы сделать это, кликните левой кнопкой мыши на той строке на панели тестового сценария, перед которой вы хотите вставить новую команду. Кликните правой кнопкой мыши и выберите “Вставить команду”. IDE добавит пустую строку перед выбранной вами. Теперь в поля ввода введите свою команду и ее параметры.

###### В режиме исходного кода

Выберите место в тестовом сценарии, куда вы хотите вставить команду. Чтобы сделать это, кликните левой кнопкой мыши между теми командами, где вы хотите вставить новую команду. Введите HTML-теги, необходимые для создания строки в три колонки, которые содержат команду, первый параметр (если он необходим команде) и второй параметр (опять таки, если он необходим). Перед переключением обратно в табличный вид убедитесь, что сохранили тест.

##### Вставка комментария

В Selenium существует возможность добавления комментариев для улучшения читаемости тестового сценария. Эти комментарии игнорируются во время выполнения теста.

Комментарии можно также использовать для добавления пустых строк (одной или нескольких). Пустая команда приведет к ошибке при выполнении теста, а пустой комментарий нет.

###### В табличном виде

Выберите строку в тестовом сценарии, куда вы хотите вставить комментарий. Кликните правой кнопкой мыши и выберите “Вставить комментарий”. Теперь в поле “Команда” введите нужный комментарий. Он будет выделен фиолетовым цветом.

###### В режиме исходного кода

Выберите место в тестовом сценарии, куда вы хотите вставить комментарий. Комментарий добавляется с помощью HTML-тегов, т.  е.: <!-- текст вашего комментария -->.

##### Редактирование команды или комментария

###### В табличном виде

Просто выберите строку, которую нужно изменить, и отредактируйте ее, введя нужные значения в поля “Команда”, “Цель” и “Значение”.

###### В режиме исходного кода

Так как во вкладке “Исходный код” работа с текстом происходит в режиме WYSIWYG, просто меняйте любые строки по своему желанию: команды, параметры или комментарии.

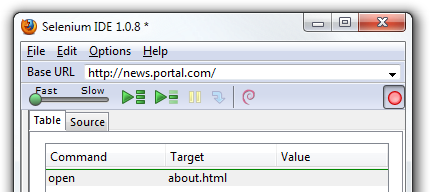
##### Как открыть и сохранить тестовый сценарий

Как и в большинстве других приложений, в Selenium IDE есть команды “Открыть” и “Сохранить” в меню “Файл”. Однако следует отметить, что Selenium IDE различает тестовые сценарии и наборы тестов. При сохранении тестов для последующего использования вы можете либо сохранить отдельные тестовые сценарии, либо целые наборы тестов. В случае, если тестовые сценарии из набора тестов не были сохранены, программа предложит пользователю сделать это перед сохранением набора тестов.

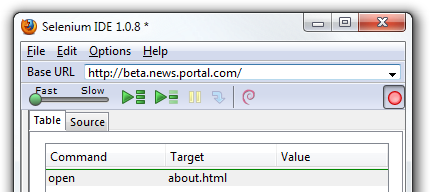
При открытии сохраненного ранее тестового сценария или набора тестов Selenium IDE отображает содержащиеся в нем команды Selenium в панели “Тестовый сценарий”.

#### Использование базового URL для запуска тестовых сценариев в других доменах

Поле Base URL (“Базовый URL”) вверху окна Selenium IDE очень полезно для запуска тестов в различных доменах. К примеру, представим, что сайт http://news.portal.com имеет тестовый веб-сайт http://beta.news.portal.com. Все тестовые сценарии для этих сайтов, начинающиеся с команды open, должны передавать относительный URL в качестве аргумента open, а не абсолютный URL (который начинается с протоколов http: или https:). Selenium IDE создаст абсолютный URL, добавив к базовому URL аргумент команды open. Например, тестовый сценарий, приведенный ниже, будет запущен на http://news.portal.com/about.html:



Этот же тестовый сценарий с модифицированным значением базового URL будет запущен на http://beta.news.portal.com/about.html:



#### Язык команд Selenium

Язык Selenium – это набор команд, которые составляют ваши тесты. Последовательность таких команд называется тестовый сценарий. В данном разделе эти команды будут рассмотрены более детально. Мы также покажем множество вариантов создания тестовых сценариев с помощью Selenium.

В Selenium существует широкий выбор команд для максимально полного тестирования веб-приложений.

Используя язык команд Selenium, пользователь может: проверить наличие элементов интерфейса пользователя по их HTML-тегам, проверить определенный контент, работу гиперссылок, полей ввода, меню, отправляемых форм, табличных данных и прочее. Команды Selenium поддерживают также проверку размеров окна, позиции курсора мыши, работу с диалоговыми окнами, элементами Ajax, всплывающими оконами, обработку событий и другие функции современых веб-приложений.

Команда сообщает Selenium, что нужно сделать. Команды Selenium бывают трех видов: **Действия** (**Actions**), **Считыватели** (**Accessors**) и **Проверки** (**Assertions**).

* **Действия** – это команды, которые обычно управляют состоянием приложения. Они совершают действия вроде “щелкнуть по той ссылке” или “выбрать эту опцию”. Если действие не может быть выполнено, либо выполняется с ошибкой, то текущий тест прерывается. К большей части действий можно добавить “AndWait” (“подождать”), к примеру, “clickAndWait”. Этот суффикс сообщает Selenium, что действие принудит браузер совершить запрос к серверу и что Selenium должен дождаться загрузки новой страницы.
* **Считыватели** анализируют состояние приложения и сохраняют результаты в переменные, к примеру, команда “storeTitle”. “Считыватели” также используются для автоматической генерации “Проверок”.
* **Проверки** похожи на “Считыватели”, однако они проверяют соответствие состояния приложения ожидаемому. Например, можно “удостовериться, что заголовок страницы Х” или “проверить, что вон тот чекбокс отмечен”. Все “Проверки” Selenium можно использовать в трёх режимах: “assert” (строгая проверка), “verify” (нестрогая проверка) и “waitFor” (ожидание). К примеру, имеются команды “assertText”, “verifyText” или “waitForText”. Если строгая проверка (“assert”) завершается неуспешно, тест прерывается. Если нестрогая проверка (“verify”) не проходит, тестовый сценарий продолжит выполняться с записью в лог об ошибке. К примеру, за одной командой “assert”, проверяющей, что приложение находится на нужной странице, могут идти многочисленные “verify”, проверяющие данные в полях ввода, надписи и т.д.

Команды ожидания (“waitFor”) сообщают Selenium о том, что необходимо дождаться выполнения определенного условия (это полезно для тестирования приложений на Ajax). Если условие выполняется, тест без остановки продолжит работу. А если условие не выполняется, тест будет остановлен до тех пор, пока оно не выполнится. А если условие так и не выполнится в течение установленного тайм-аута, оно будет считаться проваленным (см. для настроек тайм-аута описание команды “setTimeout”).

#### Синтаксис языка команд Selenium

Команды Selenium просты, они состоят из самой команды и двух параметров. К примеру:

|  |  |  |
| --- | --- | --- |
| verifyText | //div//a[2] | Логин |

Параметры не всегда необходимы, это зависит от самой команды. В некоторых случаях оба параметра необходимы, в других только один параметр, а в третьих использовать команду можно вовсе без параметров. Вот несколько примеров:

|  |  |  |
| --- | --- | --- |
| goBackAndWait |  |  |
| verifyTextPresent |  | Добро пожаловать на мою страницу |
| type | id=phone | (555) 666-7066 |
| type | id=address1 | ${myVariableAddress} |

Справка по командам описывает требования к параметрам для каждой команды.

Параметры бывают разные. Тем не менее, они обычно делятся на:

* локаторы для идентификации элементов пользовательского интерфейса на странице
* текстовые шаблоны для проверки с помощью команд “verify” или “assert” ожидаемого содержимого на странице
* текстовые шаблоны или переменные языка команд Selenium для ввода текста в поля ввода или для выбора элемента из выпадающего списка

Тестовые сценарии, исполняемые Selenium IDE, хранятся в текстовом формате HTML. Он представляет собой HTML-таблицу из трех столбцов. Первый столбец определяет команду Selenium, второй – ее цель, третий столбец содержит значение переменной. В зависимости от команды второй и третий столбцы могут быть необязательными для заполнения, однако они должны присутствовать в таблице. Каждая строка обозначает новую команду Selenium. Вот пример тестового сценария, который открывает страницу, проверяет ее заголовок (с помощь команды “assert”) и затем проверяет некоторые элементы содержимого страницы (с помощь команды “verify”):

**<table>**

**<tr><td>open</td><td>/download/</td><td></td></tr>**

**<tr><td>assertTitle</td><td></td><td>Downloads</td></tr>**

**<tr><td>verifyText</td><td>//h2</td><td>Downloads</td></tr>**

**</table>**

Отрисованный браузером в виде таблицы, код будет выглядеть так:

|  |  |  |
| --- | --- | --- |
| open | /download/ |  |
| assertTitle |  | Downloads |
| verifyText | //h2 | Downloads |

Основанный на HTML синтаксис языка команд Selenium может использоваться для написания и запуска тестовых сценариев без знания какого-либо языка программирования. Базовые знания языка команд Selenium и программы Selenium IDE позволяют в короткие сроки начать писать и выполнять тестовые сценарии.

#### Наборы тестов

Набор тестов – это комплект нескольких тестовых сценариев. Пользователь может запустить все доступные тестовые сценарии в наборе тестов как одно непрерывное пакетное задание.

При использовании Selenium IDE наборы тестов также могут быть определены, используя обычный файл HTML. Синтаксис довольно прост. Список тестовых сценариев задается таблицей, где в строках хранятся пути к файлу каждого тестового сценария. Например:

**<html>**

**<head>**

**<title>Набор сценариев тестирования функционала — Важность 1</title>**

**</head>**

**<body>**

**<table>**

**<tr><td><b>Suite Of Tests</b></td></tr>**

**<tr><td><a href="./Login.html">Вход</a></td></tr>**

**<tr><td><a href="./SearchValues.html">Тест поиска значений</a></td></tr>**

**<tr><td><a href="./SaveValues.html">Тест сохранения значений</a></td></tr>**

**</table>**

**</body>**

**</html>**

С помощью подобного файла пользователь может запустить в Selenium IDE все тестовые сценарии последовательно.

Пользователь также может запускать наборы тестов при работе с Selenium RC с помощью программирования. Есть несколько возможных вариантов. Для разработки тестового набора на языке Java в Selenium RC часто используется Junit. Для тестовых сценариев на C# можно использовать Nunit. При использовании интерпретируемого языка, к примеру, Python, в Selenium RC потребуются базовые навыки программирования для создания набора тестов. Так как основной целью применения Selenium RC является возможность использования программной логики для тестирования, то это не должно стать проблемой.

#### Часто используемые команды Selenium

В качестве заключения рассмотрим несколько типичных команд. Это будут, пожалуй, самые востребованные при создании тестов команды.

* **open** - открывает страницу по заданному URL
* **click/clickAndWait** - совершает клик, и, при необходимости, дожидается загрузки страницы
* **verifyTitle/assertTitle** - проверяет соответствие заголовка страницы ожидаемому.
* **verifyTextPresent** - проверяет наличие ожидаемого текста где-либо на странице.
* **verifyElementPresent** - проверяет страницу на наличие ожидаемого элемента по его HTML-тегу.
* **verifyText** - проверяет наличие на странице ожидаемого текста и соответствующего ему HTML-тега.
* **verifyTable** - проверяет таблицу на наличие ожидаемого содержимого
* **waitForPageToLoad** - временно прекращает выполнение теста до загрузки ожидаемой страницы. Автоматически вызывается при использовании команды clickAndWait
* **waitForElementPresent** - приостанавливает выполнение до появления ожидаемого элемента интерфейса пользователя с определенным HTML-тегом.

#### Выбор между командами “assert” и “verify”

Выбор между “assert” и “verify” определяется тем, насколько критичным является результат неуспешной проверки. Нет никакого смысла проверять правильность первого абзаца на странице, когда ваш тест уже провалился при проверке того, что браузер отображает нужную страницу. Если вы не на нужной странице, вы, вероятнее всего, остановите тест, чтобы разобраться в ситуации и как можно быстрее исправить проблему. С другой стороны, вам может понадобиться проверить множество свойств страницы и не прерывать тест, несмотря на то, что первая проверка провалилась, что позволит изучить сразу все ошибки на странице и принять соответствующие меры. Напомним, что “assert” при провале проверки остановит выполнение тестового сценария, в то время как с “verify” тестовый сценарий продолжит выполняться в любом случае.

Лучше всего использовать эти команды для логически сгруппированных проверок, начиная каждую группу с “assert”, за которой следуют одна или несколько “verify”. К примеру:

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| open | /download/ |  |
| assertTitle | Downloads |  |
| verifyText | //h2 | Downloads |
| assertTable | 1.2.1 | Selenium IDE |
| verifyTable | 1.2.2 | June 3, 2008 |
| verifyTable | 1.2.3 | 1.0 beta 2 |

Приведенный пример открывает страницу, затем выполняет проверку того, что загрузилась нужная страница, путем сравнения заголовка страницы (с помощью команды “assert”) с ожидаемым значением. Только при успешно выполненной проверке выполняется следующая по списку команда, которая проверяет с помощью “verify” наличие текста в определенном месте на странице. Далее тестовый сценарий проверяет с помощью “assert”, что первая колонка во второй строчке первой таблицы содержит ожидаемое значение, и только после удачно выполненной проверки этого элемента будут выполняться остальные команды, начинающиеся с префикса “verify”.

### verifyTextPresent

Команда verifyTextPresent используется, чтобы проверить наличие определенного текста в любом месте страницы. Принимает один аргумент — проверяемый текстовый шаблон. К примеру:

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| verifyTextPresent | Маркетинговое исследование |  |

Эта команда сообщает Selenium, что необходимо найти текст “Маркетинговое исследование” и проверить, что он существует где-либо на тестируемой странице. Используйте verifyTextPresent, когда вас интересует только присутствие текста где-либо на странице. Данная команда не подходит, если необходимо проверить, что текст находится в определенном месте на странице.

### verifyElementPresent

Эта команда используется, когда необходимо проверить наличие определенного элемента интерфейса, а не его содержимое. Команда проверяет не сам текст, а только его тэг HTML. Она часто используется для проверки наличия изображения.

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| verifyElementPresent | //div/p/img |  |

Команда выполняет проверку того, что изображение, заданное HTML-тегом <img>, находится на странице, и что перед ним следуют теги <div> и <p>. Первый (и единственный) параметр – это локатор, который используется для нахождения элемента. Функции локаторов описаны в следующем разделе.

Команду verifyElementPresent можно использовать для проверки наличия любого HTML-тега на странице. Пользователь может проверить наличие ссылок, параграфов, блоков <div> и прочего. Вот еще несколько примеров:

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| verifyElementPresent | //div/p |  |
| verifyElementPresent | //div/a |  |
| verifyElementPresent | id=Login |  |
| verifyElementPresent | link=Перейти к исследованию рынка |  |
| verifyElementPresent | //a[2] |  |
| verifyElementPresent | //head/title |  |

Эти примеры показывают использование различных методов проверки элементов интерфейса пользователя. Повторим, что локаторам посвящен следующий раздел.

### verifyText

Используйте команду verifyText, когда нужно проверить как текст, так и соответствующий ему элемент интерфейса пользователя. Для команды verifyText обязательно должен использоваться локатор. Если вы работаете с локаторами XPath или DOM, то вы можете выполнять проверку, что определенный текст находится в определенном месте относительно других элементов страницы.

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| verifyText | //table/tr/td/div/p | Это мой текст, и он находится сразу после div внутри таблицы. |

#### Нахождение элементов

Для многих команд Selenium необходимо задавать цель, идентифицирующую элемент на странице веб-приложения. Она выглядит как определение типа локатора (метода поиска) с последующим определением искомой позиции:locatorType=location. При этом тип локатора во многих случаях можно опустить. Различные типы локаторов перечислены ниже, с примерами их использования.

##### Нахождение по идентификатору

Это самый распространенный метод нахождения элементов, а также значение по умолчанию, в случае, когда тип локатора не задан явно. Этот метод находит первый элемент в коде, который подходит по id. Если элемент с соответствующим id отсутствует, то будет использован первый элемент с подходящим параметром “name”.

Предположим, в исходном коде страницы содержатся значения атрибутов “id” и “name”, как указано ниже:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **<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>** |

Тогда следующий метод поиска вернет такие элементы (в скобках указаны номера строк):

* identifier=loginForm (3)
* identifier=password (5)
* identifier=continue (6)
* continue (6)

Так как тип локатора identifier является типом по умолчанию, то identifier= в первых трех примерах писать не обязательно.

##### Нахождение по “id”

Данный тип локатора является более ограниченным, чем предыдущий, но и более точным. Используйте его, когда вам известно значение атрибута “id” элемента.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **<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>** |

* id=loginForm (3)

##### Нахождение по “name”

Тип локатора “name” ищет первый элемент с соответствующим запросу атрибутом “name”. Если у нескольких элементов одинаковое значение атрибута “name”, то тогда можно использовать фильтры, чтобы отсеять ненужные результаты. Тип фильтра по умолчанию – это значение атрибута “value”.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **<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>** |

* name=username (4)
* name=continue value=Clear (7)
* name=continue Clear (7)
* name=continue type=button (7)

В отличие от некоторых типов локаторов XPath и DOM, перечисленные выше три типа локаторов позволяют Selenium тестировать элементы пользовательского интерфейса вне зависимости от местоположения элемента на странице. Таким образом, если структура и организация страницы изменятся, тест все равно будет пройден. Может понадобиться как проверять, так и не проверять изменения в структуре страницы. Когда страница часто изменяется веб-дизайнерами, но необходимо проводить регрессионное тестирование функционала, тестирование по атрибутам “id” и “name” (или любым другим атрибутам HTML), становится очень важным.

##### Нахождение с помощью XPath

XPath – это язык, который используется для нахождения элементов в XML документах. Так как HTML может быть реализацией XML (XHTML), пользователи Selenium могут использовать этот богатый язык при работе с элементами в своих веб-приложениях. XPath выходит за рамки (в том числе и в плане поддержки) простых методов нахождения по атрибутам “id” и “name” и предоставляет огромные возможности для тестирования, такие как, скажем, нахождение третьего чекбокса на странице.

Одна из главных причин использования XPath – это отсутствие подходящего атрибута “id” или “name” для элемента, который нужно найти. XPath можно использовать для нахождения элемента с помощью абсолютного выражения (не рекомендуется) или относительно элемента, имеющего “id” или “name”. Локаторы XPath также можно использовать, чтобы определять элементы с помощью отличных от “id” и “name” атрибутов.

Абсолютное выражение содержит в себе путь XPath, начиная от корневого элемента (html), поэтому при малейшем изменении в приложении высока вероятность сбоя. Используя ближайший элемент с атрибутом “id” или “name” (в идеале родительский элемент), вы можете найти нужный вам элемент на основе их взаимного расположения. Вероятность того, что оно изменится, значительно меньше, так что ваши тесты станут более устойчивы к изменениям на странице.

Так как только локаторы xpath начинаются с “//”, то необязательно дописывать``xpath=`` для определения типа локатора.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **<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>** |

* xpath=/html/body/form[1] (3) – Абсолютный путь (перестанет работать, если в HTML будут внесены хотя бы самые незначительные изменения)
* //form[1] (3) – Первая форма на странице
* xpath=//form[@id='loginForm'] (3) – Форма с атрибутом “id”, имеющим значение “loginForm”
* xpath=//form[input/\@name='username'] (4) – Форма, в которой есть поле ввода с атрибутом “name”, имеющим значение “username”
* //input[@name='username'] (4) – Поле ввода с атрибутом “name”, имеющим значение “username”
* //form[@id='loginForm']/input[1] (4) – Первое поле ввода в форме с атрибутом “id”, имеющим значение “loginForm”
* //input[@name='continue'][@type='button'] (7) – Поле ввода с атрибутом “name”, имеющим значение “continue”, и с атрибутом “type”, имеющим значение “button”
* //form[@id='loginForm']/input[4] (7) – Четвертое поле ввода в формес атрибутом “id”, имеюшим значение “loginForm”

##### Нахождение гиперссылок по тексту ссылки

Используя метод поиска по тексту ссылки, находить гиперссылки на веб-странице довольно легко. Если на странице присутствуют две ссылки с одинаковым текстом, то будет использовано первое совпадение.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **<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>** |

* link=Continue (4)
* link=Cancel (5)

##### Нахождение по DOM

Объектная модель документа (DOM, Document Object Model) описывает структуру HTML-документа, взаимодействие с ней осуществляется посредством JavaScript. Этот метод поиска позволяет указать JavaScript-код, вычисляющий элемент страницы. Таким кодом может быть просто положение элемента в иерархии DOM, в виде пути от корня, где перечислены все промежуточные уровни, разделенные точками.

Так как только локаторы DOM начинаются со слова “document”, необязательно писать dom= при их определении.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **<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>** |

* dom=document.getElementById('loginForm') (3)
* dom=document.forms['loginForm'] (3)
* dom=document.forms[0] (3)
* document.forms[0].username (4)
* document.forms[0].elements['username'] (4)
* document.forms[0].elements[0] (4)
* document.forms[0].elements[3] (7)

##### Нахождение с помощью CSS

CSS (Cascading Style Sheets, каскадные таблицы стилей) – это язык, используемый для описания правил визуализации HTML и XML документов. Для привязки стилей к элементам документа, в CSS используются селекторы. Эти селекторы могут быть использованы Selenium в качестве еще одного метода поиска.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **<html>**  **<body>**  **<form id="loginForm">**  **<input class="required" name="username" type="text" />**  **<input class="required passfield" name="password" type="password" />**  **<input name="continue" type="submit" value="Login" />**  **<input name="continue" type="button" value="Clear" />**  **</form>**  **</body>**  **</html>** |

* css=form#loginForm (3)
* css=input[name="username"] (4)
* css=input.required[type="text"] (4)
* css=input.passfield (5)
* css=#loginForm input[type="button"] (7)
* css=#loginForm input:nth-child(2) (5)

Опытные пользователи Selenium рекомендуют отдавать предпочтение нахождению элементов с помощью CSS, так как это значительно быстрее, чем с помощью XPath и позволяет находить самые сложные объекты в документе HTML.

#### Проверка соответствия шаблону

Как и локаторы, шаблоны также часто бывают необходимыми аргументами команд Selenium. Команды, для которых необходимо указывать шаблоны: **verifyTextPresent**, **verifyTitle**, **verifyAlert**, **assertConfirmation**, **verifyText**, и **verifyPrompt**. Как уже было упомянуто, метод поиска по тексту ссылки может использовать шаблоны в локаторе. Шаблоны дают пользователю возможность описывать искомый текст с помощью специальных символов, вместо того, чтобы указывать этот текст в точности.

Существуют три типа шаблонов: подстановка (globbing), регулярные выражения (regular expressions) и точное совпадение (exact).

##### Подстановка

Многие знакомы с подстановкой, поскольку она используется для задания шаблонов имен файлов в утилитах командной строки DOS и Unix/Linux, например для команды ls \*.c). В этом примере подстановка используется для отображения всех файлов, заканчивающихся на .c, в текущем каталоге. Применение подстановок довольно ограничено. Selenium поддерживает только две конструкции подстановок:

**\*** соответствует чему угодно, т.е. отсутствию символа, одному или нескольким символам.

**[ ]** (набор символов) соответствует любому одному символу из указанных внутри квадратных скобок. Можно использовать дефис, чтобы задавать интервалы символов (при этом считается, что символы упорядочены по их ASCII-кодам).

Несколько примеров для разъяснения функциональности набора символов:

* [aeiou] соответствует любой гласной (латинского алфавита) в нижнем регистре
* [0-9] соответствует любой цифре
* [a-zA-Z0-9] соответствует любому буквенно-цифровому символу.

Подстановка обычно также использует третий спецсимвол, **?**, однако Selenium поддерживает только звездочку и набор символов.

Для передачи шаблона подстановки в команду языка Selenium, следует предварить шаблон префиксом **glob:**. Но поскольку подстановка и так используется по умолчанию, префикс пможно опустить, вместо этого просто указав сам шаблон.

Ниже приведены примеры двух команд, использующих подстановку. В действительности, текст ссылки на тестируемой странице был “Film/Television Department”, однако, благодаря использованию шаблона вместо точного текста, команда **click**будет выполнена, даже если текст изменится на “Film & Television Department” или “Film and Television Department”. Знак “\*” в шаблоне указывает на то, что следует искать соответствие “чему угодно или отсутствию чего-либо” между слов “Film” и “Television”.

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| click | link=glob:Film\*Television Department |  |
| verifyTitle | glob:\*Film\*Television\* |  |

Заголовок страницы, на которую мы перешли, кликнув по ссылке – “De Anza Film And Television Department – Menu”. Благодаря использованию шаблона вместо точного текста, проверка verifyTitle завершится успешно в том случае, если слова “Film” и “Television” (в такой последовательности) встретятся в любом месте в заголовке страницы. К примеру, если заголовок страницы будет сокращен до “Film & Television Department”, тест все равно будет пройден. Использование шаблона вместо точного текста и для перехода по ссылке, и для простой проверки результата действия (как это было сделано с командой verifyTitle в примере) поможет значительно сократить время, затрачиваемое на обслуживание подобных тестовых сценариев в будущем.

##### Шаблоны регулярных выражений

Шаблоны регулярных выражений – наиболее мощные из трех типов, поддерживаемых Selenium. Регулярные выражения также поддерживаются большинством высокоуровневых языков программирования, многими текстовыми редакторами, а также множеством приложений, в том числе такими утилитами командной строки в Linux/Unix, как **grep**, **sed** и **awk**. С помощью шаблонов регулярных выражений пользователь имеет возможность решать множество задач, которые в противном случае были бы значительно сложнее. Например, предположим, что в вашем тестовом сценарии необходимо убедиться, что определенная ячейка таблицы содержит число. regexp: [0-9]+ – простой шаблон, с помощью которого легко убедиться, что в ячейке находится число произвольной длины.

В то время как шаблоны подстановки Selenium поддерживают только **\*** и **[ ]** (набор символов), шаблоны регулярных выражений Selenium могут работать со всеми специальными символами, поддерживаемыми JavaScript. Ниже приведен их неполный список:

| **ШАБЛОН** | **СООТВЕТСТВИЕ** |
| --- | --- |
| . | один любой символ |
| [ ] | набор символов: один любой символ из указанных в скобках |
| \* | квантификатор: 0 или более предшествующих символов (или групп) |
| + | квантификатор: 1 или более предшествующих символов (или групп) |
| ? | квантификатор: 0 или 1 предшествующий символ (или группа) |
| {1,5} | квантификатор: от 1 до 5 предшествующих символов (или групп) |
| | | выбор: символ/группа слева или символ/группа справа |
| ( ) | группировка: часто используется с чередованием и/или с квантификатором |

Перед шаблонами регулярных выражений в языке команд Selenium должны стоять либо regexp:, либо regexpi:; regexp:чувствителен к регистру, regexpi: - нет.

Несколько следующих примеров помогут понять, как использовать регулярные выражения с командами Selenium. В первом примере показано использование наиболее популярного шаблона регулярных выражений – **.\***. Если последовательность из двух символов может быть расшифрована как “0 или более экземпляров любого символа” или проще – “что-либо или ничего”. Она является эквивалентом односимвольного шаблона подстановки **\*** (одна звездочка).

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| click | link=regexp:Film.\*Television Department |  |
| verifyTitle | regexp:.\*Film.\*Television.\* |  |

Пример выше функционально эквивалентен более раннему примеру, в котором использовалась подстановка для этого же тестового сценария. Отличаются эти два примера наличием **regexp:** вместо **glob:**, а также способом построения шаблона “что-либо или ничего” (**.\*** вместо **\***).

В более сложном примере ниже проверяется, что страница службы Yahoo! Weather для города Анкоридж, штат Аляска, содержит информацию о времени восхода солнца:

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| open | http://weather.yahoo.com/forecast/USAK0012.html |  |
| verifyTextPresent | regexp:Sunrise: \*[0-9]{1,2}:[0-9]{2} [ap]m |  |

Давайте рассмотрим части этого регулярного выражения:

|  |  |
| --- | --- |
| Sunrise: \* | Строка **Sunrise:**, после которой идет 0 или больше пробелов |
| [0-9]{1,2} | 1 или 2 цифры (для часов) |
| : | Символ **:** (спецсимволы не используются) |
| [``](http://selenium2.ru/docs/selenium-ide#id36)[0-9]{2} `` | 2 цифры (для минут), за которыми следует пробел |
| [ap]m | “a” или “p”, за которым следует “m” (“am” или “pm”) |

##### Точное совпадение

Шаблон **точного совпадения** бывает изредка полезен. В нем не используется никаких спецсимволов. В случае, если, например, необходимо найти символ “\*” (он используется как спецсимвол и для подстановок, и для регулярных выражений), точное соответствие будет единственным способом это сделать. К примеру, если необходимо найти пункт выпадающего меню под названием “Real \*”, то следующий код может выполниться успешно, а может и не заработать. Символ “\*” в шаблоне glob:Real \* означает “совпадение с чем-либо или ни с чем”. Таким образом, если бы в меню ранее присутствовала опция “Real Numbers,” (“Вещественные числа”), то вместо опции “Real \*” была бы выбрана она.

|  |  |  |
| --- | --- | --- |
| select | //select | glob:Real \* |

Чтобы удостовериться, что будет выбран “Real \*”, используется приписка exact: для создания шаблона **точного**совпадения, как показано ниже:

|  |  |  |
| --- | --- | --- |
| select | //select | exact:Real \* |

То же самое можно сделать, “защитив” символ “\*” в шаблоне регулярного выражения при помощи обратного слеша:

|  |  |  |
| --- | --- | --- |
| select | //select | regexp:Real \\* |

Скорее всего, большинству тестировщиков никогда не понадобится искать “\*” или несколько букв, заключенных в квадратные скобки (используется в подстановке, как набор символов). Зачастую большей части тестировщиков будет достаточно возможностей подстановки и регулярных выражений.

#### Команды “AndWait”

Разница между стандартной командой и ее AndWait разновидностью заключается в том, что после выполнения стандартной команды (например, click) Selenium как можно быстрее перейдет к выполнению следующей команды тестового сценария. Разновидность AndWait (например, clickAndWait) приказывает Selenium **дождаться** загрузки страницы после выполнения команды.

Разновидность AndWait используется всегда, когда действие требует от браузера перейти на другую страницу или обновить текущую страницу.

Следует быть внимательным, чтобы не применять разновидность команд AndWait для действий, которые не инициируют переход на другую страницу или обновление текущей страницы, иначе ваш тест не сможет успешно завершиться. Это случается из-за того, что Selenium, дождавшись конца тайм-аута AndWait, не видит никаких переходов/обновлений и вызывает исключение по тайм-ауту.

#### Команды waitFor в приложениях AJAX

В веб-приложениях на базе AJAX данные приходят с сервера без обновления страницы. Команды с AndWait не смогут нормально выполняться по этой причине. Приостановка выполнения тестового сценария на определенное время - неверный подход, поскольку веб-элемент может появиться на странице раньше или позже указанного времени, это зависит от скорости отклика системы, нагрузки либо иных не подконтрольных факторов, приводящих к сбоям теста. Лучшим выходом из ситуации является задание динамического периода ожидания появления элемента, а после удачной проверки – продолжение выполнения тестового сценария.

Разновидности команд waitFor, такие как waitForElementPresent либо waitForVisible, позволяют задавать динамическое время ожидания, каждую секунду проверяя указанное условие. После удачной проверки Selenium перейдет к выполнению следующей команды.

#### Команды сохранения и переменные Selenium

Переменные Selenium можно использовать для инициализации констант в начале скрипта. А в сочетании с управляемым данными тестовым сценарием (см. следующие разделы) – для хранения значений, принимаемых тестами из командной строки, другой программы или из файла.

store – это наиболее часто используемая из всех команд сохранения, она применяется для сохранения постоянного значения в переменную Selenium. Команда принимает два параметра: текстовое значение и имя переменной Selenium. В качестве названия переменной можно использовать только буквенно-цифровые символы.

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| store | [paul@mysite.org](mailto:paul%40mysite.org) | userName |

Для того чтобы позже использовать сохраненную переменную в тестовом сценарии, необходимо заключить ее имя в фигурные скобки “{}” и поставить перед ними знак доллара “$”, как это показано в примере.

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| verifyText | //div/p | ${userName} |

Переменные часто используются для хранения данных, предназначенных для заполнения полей ввода.

| **Команда** | **Цель** | **Значение** |
| --- | --- | --- |
| type | id=login | ${userName} |

Переменные Selenium можно использовать либо в первом, либо во втором параметре команды и они будут проинтерпретированы Selenium до того, как команда начнет выполняться. Переменные могут также использоваться внутри выражений локаторов.

Соответствующие команды сохранения существуют для всех команд проверок (“verify” или “assert”). Ниже показаны еще несколько наиболее часто используемых команд вида “store”.

##### storeElementPresent

Команда, которая соответствует “verifyElementPresent”. Она сохраняет булевское значение – “true” или “false” – в зависимости от того, найден элемент или нет.

##### storeText

“storeText” соответсвует “verifyText”. Команда использует локатор, чтобы найти определенный текст на странице. Если текст был найден, он будет сохранен в переменную. Команду “storeText” можно использовать для извлечения текста из тестируемой страницы.

##### storeEval

Эта команда принимает в качестве первого параметра фрагмент JavaScript. В следующем разделе более подробно рассмотрено внедрение JavaScript в язык команд Selenium. “storeEval” позволяет тестовому сценарию сохранять результат исполняемого скрипта в переменной.

#### Отладка

Отладка – это нахождение и исправление ошибок в тестовом сценарии. Это стандартный этап разработки тестов.

##### Точки останова и точки старта

В Selenium IDE есть возможность создать точки останова, а также запускать тест с любого места в тестовом сценарии. К примеру, можно выполнить сценарий до команды где-то в середине и проверить, как сценарий ведет себя в этой конкретной точке. Для этого нужно установить точку останова перед той командой, действие которой нужно проверить.

Чтобы установить точку останова, выберите нужную команду, кликните на ней правой клавишей мыши и из контекстного меню выберите пункт “Toggle Breakpoint” (“Установить точку останова”). После этого нажмите на кнопку “Запустить”, чтобы выполнить тестовый сценарий от начала и до точки останова.

Иногда также бывает полезно запустить тестовый сценарий с середины до конца теста или от точки старта до точки прерывания. Предположим, что ваш тестовый сценарий выстроен так, что вначале он авторизуется на сайте, а затем выполняет ряд команд, одна из которых нуждается в отладке. Авторизоваться нужно единожды, но после этого существует необходимость многократного перезапуска тестовых сценариев в процессе разработки. Решение: авторизуйтесь на сайте, потом запускайте ваш тестовый сценарий с точки старта, установленной после авторизации. Это будет более рационально, чем выходить из учетной записи вручную каждый раз перед запуском тестового сценария.

Чтобы установить точку старта, выберите нужную команду, кликните на ней правой клавишей мыши и в контекстном меню выберите пункт “Set/Clear Start Point” (“Установить/убрать точку старта”). После этого нажмите на кнопку “Запустить”, чтобы запустить тестовый сценарий с точки старта.

##### Пошаговое выполнение тестового сценария

Для того чтобы выполнять тестовый сценарий по одной команде за раз (“по шагам”):

1. Запустите тестовый сценарий с помощью кнопки “Запустить” (“Run”) на панели инструментов.

../_images/chapt3_img09_Run1.png

1. Сразу же остановите выполнение тестового сценария, нажав на кнопку “Пауза” (“Pause”).

../_images/chapt3_img10_Pause1.png

1. Нажимайте на кнопку “Шаг” (“Step”) для выполнения команд по одной.

../_images/chapt3_img12_Step1.png

##### Кнопка “Найти”

Кнопка “Найти” служит для того, чтобы узнать, какой элемент интерфейса пользователя на тестируемой странице (в браузере) соответствует локатору, указанному в качестве параметра выбранной команды Selenium. Эта функция полезна при отладке локаторов. Поиск можно использовать для любой команды, для которой в качестве значения первого параметра указан локатор элемента, например, click,clickAndWait, type, в том числе некоторые команды assert и verify.

В табличном режиме выберите любую команду с локатором в качестве параметра. Нажмите кнопку “Найти”. На веб-странице должен появиться светло-зеленый прямоугольник вокруг элемента, определенного локатором.

##### Просмотр исходного кода страницы при отладке

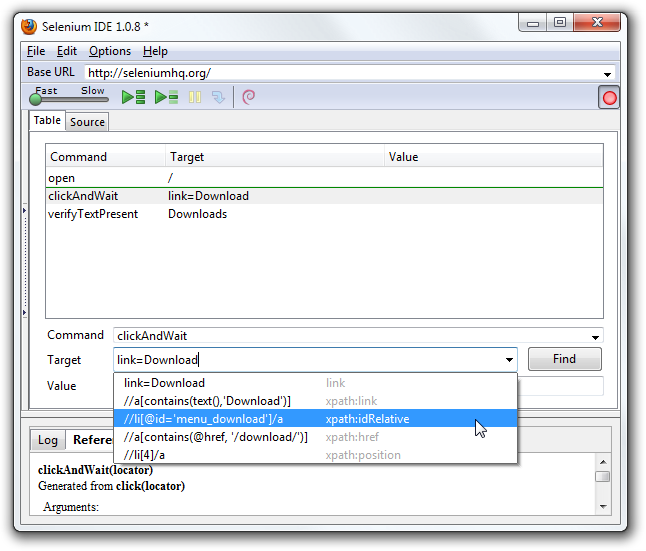
Зачастую при отладке тестовых сценариев необходимо посмотреть исходный код страницы для определения проблемы. В Firefox сделать это нетрудно. Щелкните правой клавишей мыши на странице и выберите “Исходный код страницы”. Код HTML будет показан в отдельном окне. Используйте функцию поиска (Правка=>Найти), чтобы отыскать тестируемый элемент интерфейса пользователя.

Также можно выделить только часть веб-страницы, чтобы посмотреть ее исходный код. После выделения кликните правой клавишей мыши и выберите “Исходный код выделенного фрагмента”. В данном случае в появившемся окне будет отображаться только небольшой отрывок HTML с подсвеченным участком, соответствующим вашему выделению.

##### Помощь в построении локаторов

Когда Selenium IDE записывает параметр типа “локатор”, он также сохраняет дополнительную информацию, чтобы пользователь имел возможность выбрать любой другой доступный тип локатора для данного элемента. Это может быть полезно как для изучения различных типов локаторов, так и для изменения типа локатора при составлении тестового сценария.

Изменить вид локатора можно из выпадающего списка в поле “Target” (“Цель”), когда это поле уже содержит записанный параметр типа “локатор”. На скриншоте ниже показано содержание выпадающего списка для команды. Обратите внимание, что в первом столбце списка расположены доступные альтернативные локаторы, а во втором содержится информация об их типах.



#### Составление набора тестов

Набор тестов – это список тестовых сценариев, который расположен в крайней левой панели Selenium IDE. Панель набора тестов можно вручную скрывать или открывать нажатием на небольшую полоску у правой границы панели (после закрытия панели она превращается в левую границу окна Selenium IDE).

Панель автоматически откроется, если будет открыт сохраненный набор тестов или когда пользователь выберет опцию “New Test Case” (“Новый тестовый сценарий”) из меню “Файл”. В последнем случае новый тестовый сценарий появится в списке после предыдущего тестового сценария.

Selenium IDE пока не поддерживает добавление сохраненных тестовых сценариев в набор тестов. Для того, чтобы создать или изменить набор тестов, добавив в него сохраненные тестовые сценарии, пользователю необходимо вручную отредактировать файл набора тестов.

Файл набора тестов – это файл HTML, в котором находится таблица в один столбец. Каждая ячейка каждой строки таблицы содержит ссылку на тестовый сценарий. Ниже приведен пример набора тестов с четырьмя тестовыми сценариями:

**<html>**

**<head>**

**<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">**

**<title>Пример набора тестов Selenium</title>**

**</head>**

**<body>**

**<table cellpadding="1" cellspacing="1" border="1">**

**<thead>**

**<tr><td>Тестовые сценарии для A- Z ссылок на директории</td></tr>**

**</thead>**

**<tbody>**

**<tr><td><a href="./a.html">A Links</a></td></tr>**

**<tr><td><a href="./b.html">B Links</a></td></tr>**

**<tr><td><a href="./c.html">C Links</a></td></tr>**

**<tr><td><a href="./d.html">D Links</a></td></tr>**

**</tbody>**

**</table>**

**</body>**

**</html>**

#### Задание

Разработать 6 тест-кейсов, которые выполняются в Selenuim IDE, согласно своему варианту. Варианты аналогичны лабораторной работе №1.