# Лабораторная работа № 25-26
# Selenium IDE как инструмент автоматизации тестирования

## 1. Цель работы

Изучить принципы работы со средством автоматизированного тестирования Selenium IDE.

## 2. Общие сведения и ход работы

**Selenium** – инструмент для тестирования Web-приложений. Позволяет автоматизировать GUI-тестирование.

**Ход выполнения работы:**

0) Установить браузер FireFox.

1) Через браузер FireFox открываем страницу - http://seleniumhq.org/download/;

2) С нее скачиваем последнюю версию Selenium IDE;

3) FireFox самостоятельно предложит установить данный плагин;

4) Устанавливаем -> перезагружаем браузер;

5) Установка завершена.

**Основные возможности Selenium IDE:**

1) Простая запись сценария при помощи мыши;

2) Простая и наглядная возможность редактирования сценария;

3) Возможность сохранить записанный сценарий в нужном формате;

4) Возможность использования дополнительных расширений (user extensions).

**Запись сценария:**

Вызываем Selenium IDE из меню 'Инструменты -> Selenium IDE'. Появится следующее окн (рисунок 23):

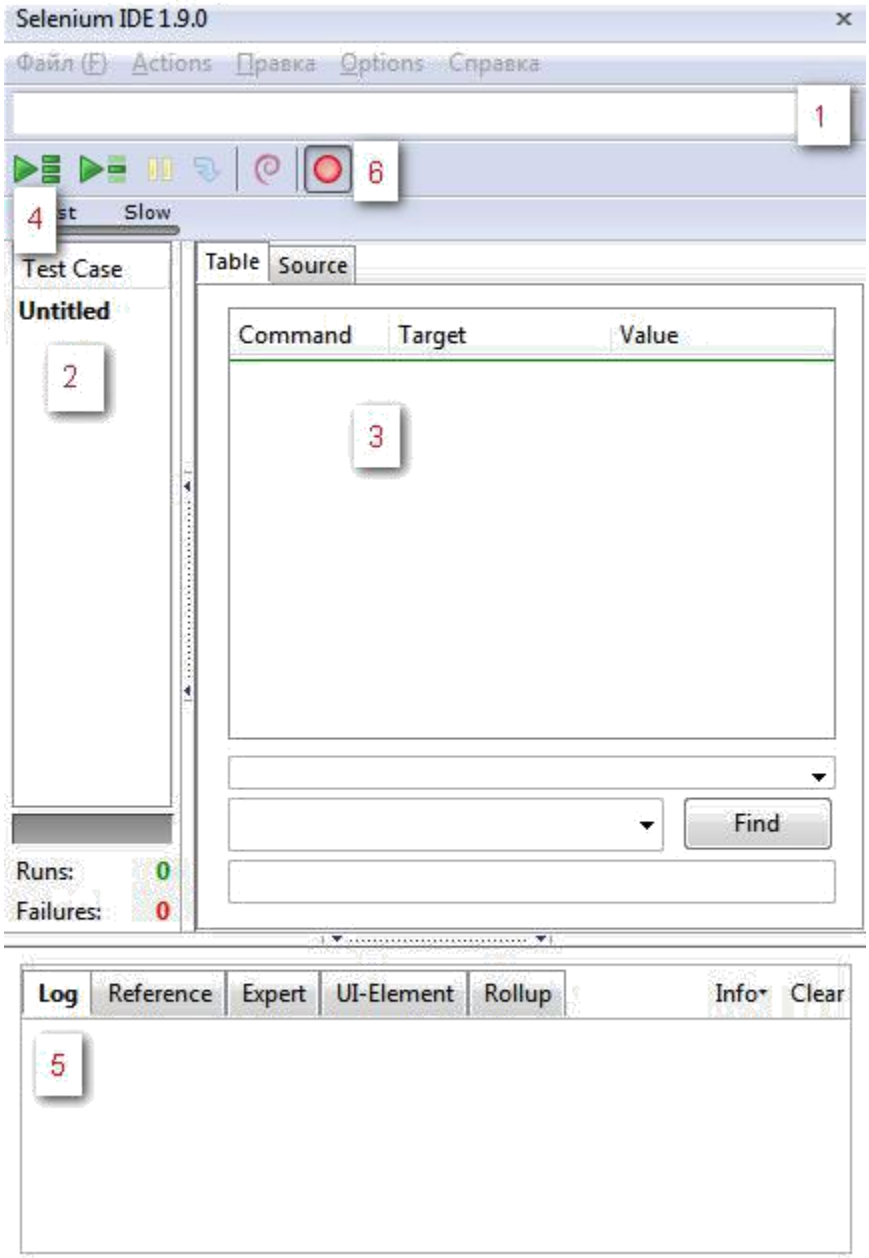


Рисунок 23 – Окно Selenium

Можно выделить следующие элементы окна:

1. Base URL - базовая страница;

2. Test Case - список тест кейсов;

3. Вкладка Table - шаги тест кейса;

4. 2 кнопки play - первая "прогоняет" все тест кейсы, вторая тот, с которым идет работа

5. Log - логи;

6. Красная кнопка (в виде кружка) сверху справа (под строкой Base URL):

а) если нажата - действия пользователя в браузере записываются в тест кейс;

б) если не нажата - действия пользователя не записываются.

**Пример.** Рассмотрим запись на примере сайта Rambler.ru.

1) Открываем сайт;

2) Запускаем Selenium IDE;

3) Мышью кликаем в поле поиска на странице и вводим слово "Selenium";

4) Кликаем мышью по кнопке "Найти".

Результат будет следующим (рисунок 24):

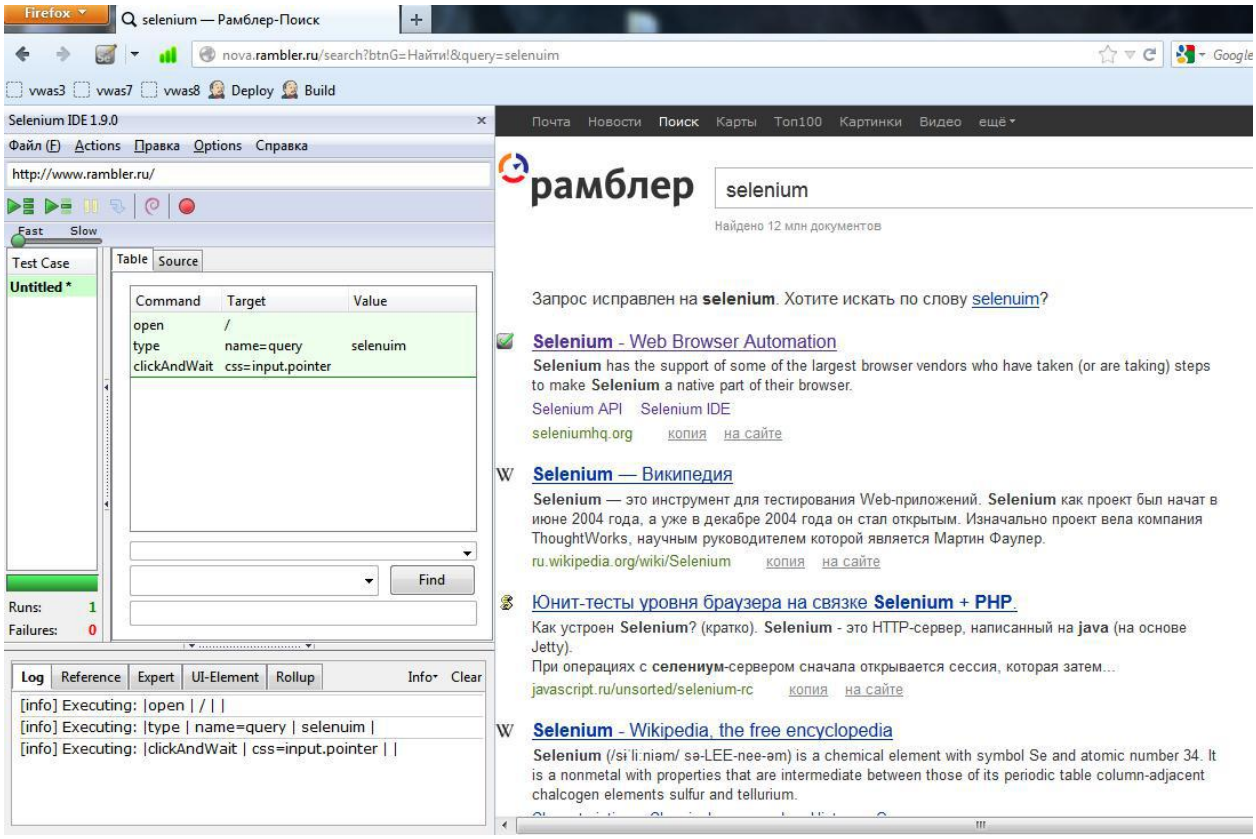


Рисунок 24 – Запись действий пользователя при помощи Selenium

Первая стока на вкладке "Table" указывает на то, что будет открыта базовая страница. Вторая - вводим в поле поиска (с name='query') слово "Selenium". Третья - кликаем на кнопке "Найти".

Соответственно:

- Первый столбец – действие (Command);

- Второй столбец - цель, над которой будет совершено действие (Target);

- Третий столбец - значение, которое будет введено/сравнено/сохранено... согласно цели и действию над ним (Value).

Строки тест кейса легко редактируются, при клике по любой из них данные для редактирования появляются в полях, которые находятся чуть ниже таблицы.

**Как сохранить и вызвать значение?**

Бывают ситуации, когда нам необходимо на определенном шаге сохранить данные, которые будет необходимо использовать в дальнейшем.

Для этого необходимо кликнуть правой кнопкой мыши по элементу (скажем по картинке) и выбрать одну из store команд (пусть будет storeText - запомним значение параметра alt). После чего появится окно, в котором необходимо ввести имя, по которому можно будет обратиться к сохраненному значению (пусть это будет myStored). Что обратиться к ней необходимо использовать конструкцию stroredVars['myStored'].

`echo storedVars['myStored']` - выведет в логах сохраненное значение.

**Как узнать идентификатор цели?**

Selenium IDE не всегда получается правильно выставить идентификатор цели (например если для того или иного элемента уникальный идентификатор не обязателен).

Для этого можно установить FireBug и FireFinder.

FireBug со страницы - https://addons.mozilla.org/ru/firefox/addon/firebug/

FireFinder - https://addons.mozilla.org/en-us/firefox/addon/firefinder-for-firebug/

Используя данные плагины легко определить идентификатор элемента, по которому к нему будет легко обратиться. Во-первых, мы можем кликнуть правой кнопкой мыши по любому элементу и в контекстном меню выбрать "Анализировать элемент" - после этого по элементу будет выведена детальная информация (id, name, value...), которую мы можем использовать. Во-вторых, мы можем искать элементы не только по уникальным идентификаторам, но и по части текста внутри них (например есть label, содержание которого "'date_' +Date()", т.е. при каждом обновлении страницы конец фразы будет меняться) или если мы ищем элемент, который находится внутри родительского, то мы можем производить поиск в рамках родительского элемента или использовать так называемые оси.

**Поиск через XPath.**

Вводим в поле target -

```
//div[starts-with(@id, 'date_')]
```

где div - это тип тегов, в котором будет производиться поиск; starts-with - параметр, указывающий на то, что нужно искать последовательность в начале;

@id - тип идентификатора, по которому будет производиться поиск; 'date_' - что ищем.

Параметры бывают:

1) starts-with, 2) ends-with, 3) contains.

Можно рассмотреть и более простой пример. Допустим, нам необходимо найти элемент input, у которого параметр class равен "someClass"; в этом случае искать мы будем следующим способом:

```
//input[@class='someClass']
```

Так мы сможем обратиться к данному элементу, через один из его дополнительных параметров. Но будьте осторожны. Если на странице находится не один элемент данного типа с таким параметром, то selenium будет использовать первый элемент, который ему попадется. Поэтому либо комбинируйте параметры для поиска, либо ищите другие признаки.

Развивая тему, можно упомянуть о том, что поиск можно производить и с использованием так называемых осей (Axis).

**Пример:**

```
//input[@value='Button']/following-
sibling::input[@value='Sibling Button']
```

В этом примере реализуется возможность обращения к полю input с значением параметра value='Sibling button', как к следующему полю типа Input за первым полем input с параметром value='Button'.

**Типы осей:**

*Таблица 5 – Типы осей*

| Axis name | Result |
|---|---|
| ancestor | выделяет всех предков |
| descendant | выделяет всех потомков элемента |
| following | выделяет все элементы текущего типа |
| following-sibling | выделит элементы указанного типа |
| parent | выделит всех родителей текущего типа |
| preceding | выделит все элементы, что до текущего элемента |
| preceding-sibling | выделит все элементы, что того же типа |

Рассмотрим пример, когда у нас есть несколько элементов внутри тега div без уникальных идентификаторов, а нам необходимо обратиться ко второму элементу Input (пусть внутри тега div будут два поля типа input):

```
//div[@class='someClass']/input[2]
```

## Поиск через CSS selector

Поиск элемента на странице через FireFinder:

```
div.name input
```

```
[какой тег].[класс] [что ищем]
```

Поиск элемента на странице с помощью selenium через css (то, что вводится в столбец target):

```
css=div.name>input
```

так мы найдем первое поле типа input внутри тега div с class='name'

А если нам необходимо выбрать другой элемент Input внутри тега div? Делаем так:

```
css=div.name>input#but1+br+input
```

В этом случае после символа # идет значение id, а потом через "+" теги, вплоть до того элемента по порядку, к которому необходимо обратиться.

Естественно можно использовать и более простые формулировки:

```
css=input[value='someValue']
```

Как и в случае с XPath мы можем использовать поиск элементов по части их значения (речь, например, о starts-with...). Ниже приведена таблица 6 аналогии css vs xpath:

*Таблица 6 – css vs xpath*

| CSS | XPath |
|---|---|
| ^= | starts-with |
| $= | ends-with |
| *= | contains |

Пример использования: `css=div[id*='date_']`

Для поиска N-го элемента внутри какого либо тега:

```
css=div#someid    *:nth-child(2)      ----      то же самое, что и
//div[@id='someid']/sometag[2]
```

Можно искать элементы и по тексту, который содержится внутри тегов (например  Some text inside ):

```
css=div:contains('me tex')
```

## Pattern Matching

Рассмотрим 4 инструмента:

- Exact

- Glob

- Regexp

- Wildcards

Для чего это нужно? Например для того, чтобы сравнивать данные между собой. Сразу рассмотрим пример:

## Exact

| command | target | value |
|---|---|---|
| verifyText | //div[@id='someText'] | exact:texttexttext |

Здесь сравниваются значения цели и то, что мы ввели в столбец value. Если сходятся, то шаг пройдет успешно. Не сходятся - не пройден.

## Glob

Используем так же как и exact

| command | target | value |
|---|---|---|
| verifyText | //div[@id='someText'] | glob:*sometextinside* |

Здесь используются спецсимволы:

| Символ | Вид | Описание |
|---|---|---|
| * | glob:*text* | найдет данный текст внутри текстовой строки. |
| ? | glob:?xt | найдет текст с окончание xt, например text |

## Regexp

Инструмент используется для сравнивания количества символов в символьной строке. Пример:

| command | target | value |
|---|---|---|
| verifyText | centerdiv | regexp:\w{3}\d{2} |

Если в проверяемом выражении есть пробелы, их необходимо ставить и в выражении, которым проверяем (т.е. в столбце value).

| Символ | Описание |
|---|---|
| \w{n} | ищем n букв подряд |

| | |
|---|---|
| \d{n} | ищем n цифр подряд |
| \s | пробел |

## Wildcards

Маски используются в регулярных выражениях (regexp).

**Пример:**

`regexp:\w.*` (или вместо * ставить +), где:
. - обязательный элемент (ставить всегда при использовании масок);

* - от 0 до n;

+ - от 1 до n;

[a-z] - используется при проверке на соответствие наличия лишь букв.

## JavaScript

Использование javascript зачастую очень может облегчить жизнь, в случае, если необходимо использовать данные, которые должны формироваться динамически (например, текущее время или математические расчеты и т.д.). Рассмотрим пример, в котором в поле с id currentTime вставим текущее время:

| Command | Target | Value |
|---|---|---|
| type | currentTime | javascript{d=new Date(); d.getHours()} |

Так же рассмотрим математические расчеты. В поле с id mathOp вставим результат произведения сохраненной величины savedVar (пусть она будет 5) и 100:

| Command | Target | Value |
|---|---|---|
| type | //div[@id=mathOp] | javascript{+storedVars['savedVar']*100} |

, где + перед storedVars обозначает, что мы привели величину к типу int.

## BrowserBot

Обращение к javascript напрямую производится через browserbot. Рассмотрим пример:

| Command | Target | Value |
|---|---|---|
| getEval | this.browserbot.getUserWindow().[метод] | |

Теперь используя выше описанный шаблон рассмотрим "боевой" пример:

| Command | Target | Value |
|---|---|---|
| getEval | this.browserbot.getUserWindow().document.getElementById('someId').options.length | 4 |

В этом примере мы сравниваем текущее количество элементов внутри комбобокса с id = someId с тем количеством, которое необходимо (т.е. внутри столбца target мы получаем сколько элементов в данный момент и сравниваем со значением внутри столбца value).

## Firing events

Бывает необходимо проверить действия системы (скажем при наведении мыши на текст), возникающие при обработке событий. Производится это при помощи firing events:

| Command | Target | Value |
|---|---|---|
| fireEvent | //div[@id=someId] | mouseover |
| verifyAlert | text inside the alert | |

В данном примере мы выполняем событие onMouseOver при наведении мыши на объект с id=someId и проверяем его содержимое наличие описанного нами текста внутри.

Можно работать со следующими событиями:

- onFocus;
- onChange;
- onMouseOut;
- onBlur;
- onSubmit;
- onMouseOver.

## Расширения пользователя

Расширения используются в тех случаях, когда необходимо часто использовать какую либо функцию, а описывать постоянно внутри Selenium

IDE не хочется. Т.е. можно описать заранее алгоритм и вызывать его, когда необходимо.

Для создания расширения необходимо в файле с расширением js описать данную функцию и указать в настройках selenium (Options -> Options -> Selenium Core Extentions) путь к данному файлу. Пример функции:

```
Selenium.prototype.doOurRandom = function
    (nameOfVar){ random = Math.floor(Math.random());
    storedVars\[nameOfVar\]=random;
}
```

| Command | Target | Value |
|---|---|---|
| ourRandom | myRandom | |
| echo | storedVars[myRandom] | |

В примере наглядно показано, как создавать функцию:

- Selenium.prototype.doOurRandom -- обязательная строка, где doOurRandom - название нашего метода, через которое мы будем вызывать его в Selenium IDE, к тому же вызывать будем по имени ourRandom (не doOurRandom). do - параметр, обозначающий, что далее следует имя метода.

- function (nameOfVar) - здесь все понятно. Можно упомянуть лишь о том, что внутри можно разместить и два параметра, которые можно получить (соответственно первый получаем из столбца Target, второй - Value).

И как работать с функцией в Selenium IDE: передаем методу ourRandom имя переменной, которую позденее выводим в логах.

Использование локаторов (id, name...)

Еще один небольшой пример:

```
Selenium.prototype.doTypeTodaysDate = function (locator)
    { var dates = new Date();
    var day = dates.getDate();
    if (day < 10){
      day = '0' + day;
    }
    month = dates.getMonth() + 1;
    if (month<10){
```

```
    month = '0' + month;
  }
  var year = dates.getFullYear();
  var prettyDay = day + '/' + month + '/' + year;
  this.doType(locator, prettyDay);
}
```

## 3. Содержание отчета по лабораторной работе

1. Цель работы.
2. Описание задания на автоматизацию (необходимо выбрать любой сайт и автоматизировать там действие пользователя: отправку сообщений, осуществление заказа в интернет-магазине, покупка билета и т.д. Действие должно повторяться больше одного раза).
3. Автоматизированные скрипты с пояснениями.
4. Выводы по лабораторной работе.