**Разработка функциональных тестов при помощи Selenium**

**Структура статьи:**

1. [Подходы к автоматизации функционального тестирования.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_1)

    1.1 [План разработки автоматических функциональных тестов.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_1_1)

    1.2. [Соглашения.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_1_2)

          1.2.1. [Расстановка приоритетов тестов.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_1_2_1)

          1.2.2. [Правило идентификации тестов.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_1_2_2)

          1.2.3. [Правило формирования идентификаторов элементов.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_1_2_3)

          1.2.4. [Структура функционального теста.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_1_2_4)

2. [Подготовка к тестированию с использованием Selenium IDE.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_2)

    2.1. [Настройка Selenium IDE для Mozilla.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_2_1)

    2.2. [Настройка Linux-сервера для тестирования без использования графики.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_2_2)  
    2.3. [Структура тестов Selenium.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_2_3)  
    2.4. [Основы работы с Selenium IDE.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_2_4)  
3. [Логические операции в Selenium.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_3)

4. [Реализация ветвления в Selenium.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_4)

5. [Реализация циклов в Selenium.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_5)

6. [Примеры типичных обращений к элементам GWT-формы.](http://forworktests.blogspot.ru/2013/03/selenium.html#id_6)

**1. Подходы к автоматизации функционального тестирования**

Под ***функциональным тестированием***будем понимать проверку соответствия реализованной функциональности приложения заявленному в ТЗ, или условиям, поставленным в соответствующей задаче по разработке. При этом чтобы избежать многократного повторения ручного тестирования, разрабатываются функциональные автоматические тесты.

**1.1. План разработки автоматических функциональных тестов**

1. Анализируется приложение и составляется функциональная схема логики его работы: выделяются основные формы и группируются по реализуемому ими функционалу (справочники, подсистемы администрирования и авторизации, формы ввода данных, формы для выгрузки данных и т.п.). **Результатом является тест-план.**
2. Создаются комплекты шаблонов для тестов требуемых категорий. **Результатом являются Init-тесты**, файлы «пустышки», содержащие заголовки и основную структуру теста.
3. Разработчик теста выбирает для тестируемой формы соответствующий ей шаблон и заполняет его проверками, требуемыми для конкретной категории теста. **Результатом является готовый тест.**
4. Формируются тестовые наборы, пишутся скрипты для их запуска и **настраивается автозапуск тестов** на сервере сборки TeamCity после каждой успешной сборки приложения.

**1.2. Соглашения**

Перед началом работ по автоматизации, группе тестирования рекомендуется принять некоторые технические соглашения, например, предложенные ниже.

**1.2.1. Расстановка приоритетов тестов**

**0** - (наивысший) приоритет для теста инициализации и авторизации;

**1** - (высокий) приоритет для тестов главных модулей приложения (справочники и основные формы);

**2** - (нормальный) приоритет для второстепенных форм приложения (отдельных вкладок в модулях приложения);

**3** - (низкий) приоритет для тестов проверки отдельных элементов форм, сценариев или тестов проверки корректной реализации дополнительной функциональности приложения.

**1.2.2. Правило идентификации тестов**

Данное правило идентификации может свободно изменяться. Главное отмечать эти изменения в тест-плане.

**Идентификатор теста** (имя файла) состоит из **10 цифр**, разделенных знаком подчеркивания:

**1 цифра - тип теста**(0 - CRUD, 1 - Regression, 2 - Positive, 3 - Negative, 4 - End-to-end),

**2 цифра - логический модуль приложения**(0 - подсистемы безопасности, 1 - справочники, 2 - формы карточек и т.п.),

**3, 4 цифры** - номер формы (00 - форма справочника или список карточек, 01 - Форма 1, ..., 10 - Форма 10),

**5, 6, 7 цифры - номер вкладки в форме или номер справочника**(7-я цифра запасная, по умолчанию 0, 010 - первая вкладка или справочник, ..., 100 - десятая вкладка, ...),

**8, 9, 10 цифры - номер подвкладки**(10-я цифра запасная, по умолчанию 0 (000 - если подвкладок нет, 010 - первая подвкладка, ..., 100 - десятая подвкладка, ...).

Дополнительно можно указать **примечание**, через знак подчеркивания (\_db - проверка справочника, \_Instrumental\_control - подвкладка инструментального контроля, ...).

Например, **0\_2\_06\_010\_010\_Main** - идентификатор для CRUD-теста Формы 6, первой вкладки, первой подвкладки - Общие сведения, **0\_1\_00\_020\_000\_db** - идентификатор для CRUD-теста второго справочника, в котором нет подвкладок.

Пример разработки тестплана, составленного по данному правилу, можно посмотреть на страничке: [Общие сведения о тестировании web-приложений](http://forworktests.blogspot.ru/2013/03/web_3.html).

**1.2.3. Правило формирования идентификаторов элементов**

Для облегчения автоматизации каждый элемент web-страницы должен быть идентифицирован уникальным id. **Имена id-шников** для управляющих элементов разработчики могут формировать, например, в виде:

**testid="ELEMENTTYPE\_YYYYMMDDhhmmss"**

Например, получаемый код HTML может выглядеть так:

**<div style=combobox\_field testid="combo\_20110913160925">**

В качестве**типов** ELEMENTTYPE, можно писать префиксы, соответствующие управляющим элементам формы:

**combo\_**, **grid\_**, **btn\_**, **text\_**, **form\_**, **chk\_**, и т.п.

**1.2.4. Структура функционального теста**

Каждый test-case (тест-кейс, тестовый случай) обязательно должен содержать:

* **заголовок** (можно вставить комментарием в HTML-файл с тестом), хранящий служебную информацию:

**AUTHOR:**разработчик(-и) теста;

**TEST PRIORITY:**приоритет исполнения теста, при необходимости выбора - в первую очередь должны исполняться тесты более высокого приоритета;

**SPEC ID:**спек (спецификация, ТЗ) - документ, на который ориентировались, при создании теста;

**OVERVIEW:**краткое, но понятное описание теста для других специалистов-разработчиков;

**IDEA:**идея, логика работы теста, кратко описанная цель теста;

**SETUP&INFO:**предварительные настройки для теста.

**Например:**

AUTHOR: Т.М. Гильмуллин;

TEST PRIORITY: 2;

SPEC ID: 'ТЗ - АИС\*\*\*\* - 20110512 1430.doc';

OVERVIEW: Проверяется работоспособность интерфейса подсистемы "Форма № 5. Котельные водогрейные", вкладка "Измерительная часть";

IDEA: Все управляющие элементы, присутствующие на вкладке и каждой вложенной вкладке, должны быть работоспособны;

SETUP&INFO: тесты 000-100\_\*\*\* выполнены успешно;

* **блок инициализации теста**, где выполняются авторизация в системе, предварительные настройки для теста, иные подготовительные действия;
* **тело теста**, где непосредственно выполняются основные проверки, соответствующие типу теста;
* **блок завершения теста**, где выполняются необходимые действия для приведения системы в первоначальное состояние, корректного выхода из системы.

**2. Подготовка к тестированию с использованием Selenium IDE**

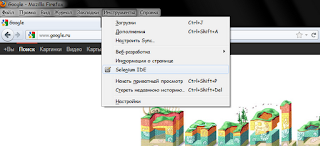
**Selenium IDE** представляет собой визуальную среду разработки, предназначенную для автоматизации тестирования Web-приложений. Он реализован в виде плагина для Firefox и позволяет записывать, редактировать и отлаживать тесты.

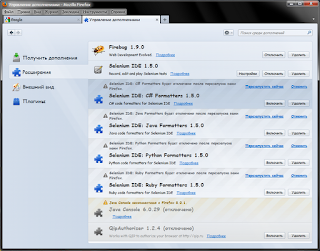
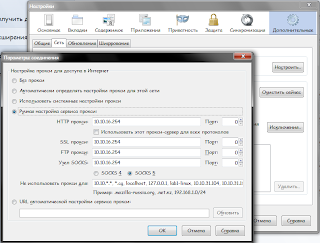
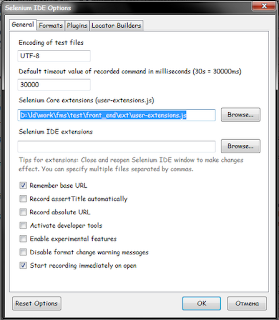
Подробная русская документация по Selenium: <http://selenium2.ru/docs.html>.

Рекомендуется разрабатывать тесты от начала до конца только в Selenium IDE, не используя сторонних средств. Дополнительный функционал всегда можно реализовать на JavaScript и добавить в специальный файл расширений: **user-extensions.js**, подключаемый как для Selenium IDE, так и для запуска в режиме Selenium RC (консольный запуск, при котором TestRunner запускает скрипт Selenium, используя Selenium Server для интерпретации команд).  
  
**Шаблон структуры**, которой можно придерживаться для упрощения организации автоматизированного тестирования web-приложений  с использованием Selenium IDE, содержится в каталоге **front\_end**, который можно скачать здесь (не забудьте обновить .jar-файл с Selenium Server в каталоге autorun и прописать соответствующие настройки в скриптах config\_for\_all\_test\_suites\*, каталог с профилем мозиллы может быть пустым):  
[**https://docs.google.com/open?id=0B-1rf8K04ZS5RmNlNEhNMTFMNU0**](https://docs.google.com/open?id=0B-1rf8K04ZS5RmNlNEhNMTFMNU0)  
Код в GitHub:  
[**https://github.com/Tim55667757/front\_end\_tests**](https://github.com/Tim55667757/front_end_tests)  
 **Структура тестового фреймворка для Selenium IDE:**  
**front\_end** - корень проекта, содержит файлы с отдельными тест-кейсами и тест-сьютами.  
        **/autorun** - содержит каталог с профилем мозиллы и bash/shell скрипты для запуска тестов.  
            **config\_for\_all\_test\_suites.\*** - общие настройки для всех скриптов запуска тестов.  
**autorun\_\*.\*** - скрипты для запуска различных типов тестов.  
            **run\_\*.\*** - скрипты для запуска smoke-тестов.  
        **/ext** - содержит файл с пользовательскими расширениями функциональности Selenium IDE.  
        **/log** - каталог с результатами работы формируется автоматически в процессе отработки тестов.  
            **selDebug.txt** - файл по умолчанию для отладочной информации Selenium Server.  
            **testResult.html** - файл по умолчанию с результатами прохождения тестов.  
        **0\_1\_00\_000\_Enter\_system\_1.html**- примеры тест-кейсов для CRUD-тестов.  
        **\*\_Regression.html** - примеры тест кейсов для регрессионных тестов.  
        **\*\_Positive.html** - примеры тест кейсов для позитивных тестов.  
        **\*\_Negative** - примеры тест кейсов для негативных тестов.  
        **\*\_End\_to\_end** - примеры тест кейсов для End-to-end сценариев.  
        **projectname\_Test\_suite\_\*.html** - примеры тестовых наборов для различных типов тестов.

**2.1. Настройка Selenium IDE для Mozilla**

1. Скачать и установить актуальную версию **Selenium IDE** с сайта <http://seleniumhq.org/download/>.  В меню браузера во вкладке "Инструменты" должен появиться пункт Selenium IDE.

[](http://3.bp.blogspot.com/-SjXTjJTVGdU/UEOSqGbNQhI/AAAAAAAAAHU/Y5SxQ5bSYDk/s1600/9.png)

1. Отключить в Mozilla все ненужные надстройки: *Инструменты -> Дополнения -> Расширения.*Для отладки тестов достаточно оставить Firebug и Selenium. Для прогона тестов их тоже можно отключить.  
   [](http://3.bp.blogspot.com/-oPkyK7oSL0E/UEOSiCM4F1I/AAAAAAAAAGc/nhC-nchFcUQ/s1600/10.png)
2. Настроить в Mozilla Default Profile, при необходимости proxy-сервер и исключения для локальных адресов: *Инструменты -> Настройки -> Дополнительные -> Сеть -> Соединение Настроить -> Ручная настройка сервиса прокси.  
   [](http://3.bp.blogspot.com/-V5m5rQtlxuA/UEOSjVtVULI/AAAAAAAAAGg/TLJvtFk4r8A/s1600/11.png)*
3. Открыть Selenium. В настройках прописать путь к файлу с расширениями user-extensions.js: *Options -> Options... -> General*. Например, он может выглядеть так: D:\ld\work\fms\test\front\_end\ext\user-extensions.js (файл с расширениями для тестов по некоторому проекту, для каждого проекта может быть указан свой файл).  
   [](http://2.bp.blogspot.com/-qLHRJpycW_8/UEOSgzcdwkI/AAAAAAAAAGU/leR6weDSD-Y/s1600/1.png)
4. На той же вкладке проверить, что используется кодировка UTF-8. Сохранить изменения.
5. Скачать с сайта Selenium .jar-файл с **Selenium Server** (**Selenium RC Server**) для запуска тестов в консольном режиме.
6. Для написания тестов возможно потребуется также установка в Mozilla плагина Firebug для более удобного поиска пути к элементу (<http://firebug.ru/>).

**2.2. Настройка Linux-сервера для тестирования без использования графики**

При необходимости прогона тестов Selenium в Linux без использования графики можно  настроить сервер по следующей инструкции (для CentOS Linux):

1. Установить шрифты из репозитория:

**yum install xorg-x11-fonts\***

1. Установить сервер виртуального экрана Xvfb из репозитория:

**yum install xorg-x11-server-Xvfb**

1. Установить mozilla версии 4.0 или выше:

скачать можно с <ftp://ftp.mozilla.org/> затем распаковать в **/usr/local/firefox**.

1. В переменную среды PATH добавить путь к mozilla, для чего отредактировать файл:

**~/.bash\_profile**

и добавить, например, в конце файла:

**PATH=$PATH:$HOME/bin:/usr/local/firefox:**

1. Выполнить команду для указания пути к библиотекам firefox:

**export LIBPATH=/usr/local/firefox**

1. Скопировать в каталог с тестами **профиль mozilla** по умолчанию (можно использовать уже имеющийся в Windows).
2. Запустить сервер виртуального экрана как сервис:

**Xvfb :1 -fp /usr/share/X11/fonts/misc -screen 1 1280x1024x32 &**

1. Указать вывод на дисплей 1:

**export DISPLAY=:1**

1. Проверить корректный запуск mozilla с профилем по умолчанию и, если требуется, обновить пакеты:

**firefox -P <Profile\_path>**

**2.3. Структура тестов Selenium**

**Тест Selenium** – набор команд прикладного уровня модели OSI, имитирующих действия пользователя в web-приложении.

Selenium сохраняет файлы с тестами в обычных HTML-файлах с простой структурой содержащей одну таблицу из трех колонок, что позволяет редактировать тесты в любом редакторе:

<table cellpadding="1" cellspacing="1" border="1">  
<tr><td rowspan="1" colspan="3">**Test head**</td></tr>  
<tr>  
    <td>**Command**</td>  
    <td>**Target**</td>  
    <td>**Value**</td>  
</tr>  
...  
<tr>  
...  
</tr>  
</table>

**Test head** - заголовок теста, **Command** - команда языка Selenium, **Target** - цель, это элемент над которым должно выполняться действие (обычно указывается как xPath на элемент), **Value** - параметр, при необходимости передаваемый в команду. Более подробно о назначении полей Target и Value указано в справке для конкретной команды.

Итогом разработки тестов является формирование различных тестовых наборов (Test-suites), содержащих тесты различных типов, или тесты для отдельных модулей приложения. Для каждого такого тест-сьюта можно написать скрипт для его запуска и настроить автоматический запуск тестов по расписанию (например, в системе TeamCity).

|  |  |
| --- | --- |
| [http://4.bp.blogspot.com/-zM6lCf_gMGc/UEObSjMmruI/AAAAAAAAAHs/5dOLh-sBZZs/s320/teamcity.png](http://4.bp.blogspot.com/-zM6lCf_gMGc/UEObSjMmruI/AAAAAAAAAHs/5dOLh-sBZZs/s1600/teamcity.png) | [http://4.bp.blogspot.com/-5cvVQTSjoW4/UEObRaUtOtI/AAAAAAAAAHo/-GUa2NSfKmU/s320/selTestResult.png](http://4.bp.blogspot.com/-5cvVQTSjoW4/UEObRaUtOtI/AAAAAAAAAHo/-GUa2NSfKmU/s1600/selTestResult.png) |

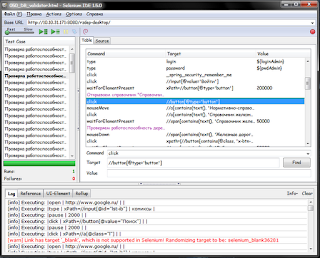
После прогонки тест-сьютов Selenium можно ознакомиться с лог-файлом - отчётом по тестированию, который включает в себя:

* общий результат прогонки тестового набора (passed / failed);
* общее время тестирования;
* общее число выполненных тест-кейсов, и число успешных и неуспешных из них;
* число успешных, неуспешных тестовых команд и команд с ошибками;

Используя оглавление отчёта можно перейти к конкретному тест-кейсу и ознакомиться с более детализированной информацией по нему.

**2.4. Основы работы с Selenium IDE**

После установки и настройки программы согласно инструкции выше, можно приступать к работе. Открываем Selenium. Окно программы делится на 3 области. В левом верхнем окне показывается открытый **Test Case** и список Test Case-ов в открытом **Test Suite**-е. В основном правом верхнем окне отображается список шагов открытого Test Case'а.

[](http://4.bp.blogspot.com/-HexJ6ki0H4A/UEOSkYXvS2I/AAAAAAAAAGo/5f2c5yAbs9U/s1600/3.png)

Основное окно имеет две вкладки - **Table** и **Source**. На вкладке Table команды представлены в виде таблицы, а на вкладке Source - в виде html-кода.

В нижнем окне на вкладке **Log** показываются логи действий теста, на вкладке **Reference** - пояснения к выбранной команде.

|  |  |
| --- | --- |
| [http://2.bp.blogspot.com/-HREnKpug6jc/UEOSmzwfSRI/AAAAAAAAAG4/DzsDxmlkF4k/s320/5.png](http://2.bp.blogspot.com/-HREnKpug6jc/UEOSmzwfSRI/AAAAAAAAAG4/DzsDxmlkF4k/s1600/5.png) | [http://1.bp.blogspot.com/-JpOzY0Ug1uA/UEOSlWQf1aI/AAAAAAAAAG0/b8WEv-dWi-U/s320/4.png](http://1.bp.blogspot.com/-JpOzY0Ug1uA/UEOSlWQf1aI/AAAAAAAAAG0/b8WEv-dWi-U/s1600/4.png) |

На панели, находящейся над окнами Test Case и Table, расположены кнопки управления тестом. Здесь можно менять скорость прогонки теста, запускать Test Suite, запускать Test Case, приостанавливать работу. При нажатии на кнопку **Record**в тест производится запись всех действий пользователя в браузере.

[http://1.bp.blogspot.com/-pjElvUGnJiY/UEOSo39nkLI/AAAAAAAAAHI/_K8cChPE-ZE/s640/8.png](http://1.bp.blogspot.com/-pjElvUGnJiY/UEOSo39nkLI/AAAAAAAAAHI/_K8cChPE-ZE/s1600/8.png)

Основная работа происходит в окне **Table**. Каждый шаг Test Case'а состоит из трех частей:

**Command** - команда или действие, которое нужно совершить;

**Target**  - путь к элементу, к которому применяется команда (обычно используется xPath, подробнее об идентификации элементов читайте страничку: [xPath: идентификация элементов на web-форме](http://forworktests.blogspot.com/p/xpath-web.html));

**Value** - значение, передаваемый команде параметр (может использоваться для пользовательских комментариев, если параметр для команды не требуется).

Кнопка **Find** помогает проверить правильность выбранного элемента, подсвечивая его.

**3. Логические операции в Selenium**

В Selenium можно использовать логические операции И, ИЛИ, НЕ, используя синтаксис JavaScript. Эти операции можно использовать в вычислениях (команда **storeEval**).

* **&, &&** - символы операции **И**,
* **|, ||** - символы операции **ИЛИ**,
* **!**- символ операции **НЕ**.

Пример выражения:

**storeEval  (${a} <= 0) || (${b} <= 0) || (${c} <= 0) || (${a} + ${b} <= ${c}) || (${a} + ${c} <= ${b}) || (${b} + ${c} <= ${a})  flag**

В этом примере проверяется существование треугольника со сторонами a, b, c. Если логическая переменная flag = 0 (false, условие не выполнилось), то треугольник с такими сторонами построить можно, если flag = 1 (true, условие выполнилось), то треугольник с такими сторонами построить нельзя.

**storeEval  (${a} == ${b}) && (${b} == ${c})  flag**

Если логическая переменная flag = 1 (true, условие выполнилось), то треугольник равносторонний.

Затем, полученные переменные-флаги можно использовать, например, для изменения логики работы теста:

**gotoIf  !storedVars['flag']  jump\_1**

Если условие не выполнилось, то перейти на нужную метку.

**4. Реализация ветвления в Selenium**

Команды ветвления могут быть реализованы в файле расширений **user-extensions.js** (как это сделать, можно прочитать здесь: <http://wiki.openqa.org/display/SEL/flowControl>).  
Ветвления рекомендуется добавлять для избежания ситуаций отсутствующих данных. Например, если предыдущие команды добавляли данные, которые по некоторым причинам не сохранились, а дальнейшие команды пытаются к ним обратиться. В этом случае, чтобы тест не проваливался можно добавить проверку наличия созданного элемента и, если его нет, перейти на метку в конец теста.

Использование команды **gotoIf**в теле теста:

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Target** | **Value** | **Комментарий** |
| ... | ... | ... | Некоторые команды выше. |
| store\* | <выражение> | flag | Любая команда для вычисления значения <выражения> из семейства команд store\*, результат кладется в переменную flag типа Boolean. В зависимости от задачи можно использовать команды: store (сохранить произвольное значение в переменную flag), storeElementPresent (результат - тип Boolean, помещается в переменную flag), storeEval (вычислить произвольное выражение и поместить результат в переменную flag). Подробнее о точной записи команд семейства store\* имеется во внутреннем справочнике Selenium. |
| gotoIf | storedVars['flag'] | Label\_1 | Если переменная flag = true, то перейти на метку Label\_1, которая может быть в любом месте теста. Также можно использовать выражение !storedVars['flag'] - отрицание значения flag. |
| ... | ... | ... | Произвольные команды. |
| label | Label\_1 |  | Описание метки Label\_1. Достаточно добавить аналогичную строку в любом месте теста, куда требуется совершать переходы. |
| ... | ... | ... | Некоторые команды ниже. |

**Пример** использования условного перехода в тесте. **Задача:** реализовать удаление элементов, в случае, если они были созданы Selenium и отображены на форме, иначе - перейти на выход.

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Target** | **Value** | **Комментарий** |
| <инициализация теста> | ... | ... | Какие-либо команды предварительной инициализации теста. |
| <создание элемента> | xPath=//span[contains(text(), '\_Sel')] |  | Блок команд для создания элемента, имеющего в результате указанный xPath, содержащий текст '\_Sel'. |
| storeElementPresent | xPath=//span[contains(text(), '\_Sel')] | f | Определяем существование элемента с указанным xPath и результат заносим в переменную f типа Boolean. |
| gotoIf | !storedVars['f'] | L\_Break\_1 | Если выполняется условие "элемент НЕ существует" = true, то переходим на метку L\_Break\_1. |
| mouseDown | xPath=//span[contains(text(), '\_Sel')] |  | Выделяем мышью нужный элемент, содержащий текст '\_Sel'. |
| click | xPath=//table[@testid='btn\_Удалить'] |  | Нажимаем на кнопку меню 'Удаление' элемента. |
| waitForElementPresent | xPath=//span[contains(text(), 'Подтверждение удаления')] | 50000 | Ожидаем появление диалога удаления. |
| click | xPath=/button[contains(text(), 'Да')] |  | Нажимаем на кнопку 'Да' в диалоге удаления. |
| waitForElementNotPresent | xPath=//span[contains(text(), 'Подтверждение удаления')] | 50000 | Ожидаем, пока диалог удаления не закроется. |
| pause | 2000 |  | Добавляем небольшую паузу, для того, чтобы команда на удаление успела уйти и обновился список элементов на форме, после удаления. |
| ... | ... | ... | Прочие команды в тесте. |
| label | L\_Break\_1 |  | Описание метки перехода L\_Break\_1 |
| <обновить данные> |  |  | Обновление данных на форме. |
| <выйти из системы> |  |  | Команды для корректного выхода из системы. |

**5. Реализация циклов в Selenium**

Команды циклов могут быть реализованы в файле расширений user-extensions.js (как это сделать, можно прочитать здесь: <http://wiki.openqa.org/display/SEL/flowControl>).

В CRUD-тестах рекомендуется выносить в циклы все команды для многократного создания случайных данных для полей форм, а также при необходимости удаления множества элементов. При необходимости можно вложить цикл внутрь условия, содержащего gotoIf, но не наоборот (не допускается выход из циклов по команде gotoIf и вообще не рекомендуется использовать её внутри циклов).

Использование команд while - endWhile в теле теста:

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Target** | **Value** | **Комментарий** |
| ... | ... | ... | Некоторые команды выше цикла. |
| store | 0 | i | Инициализируем счётчик цикла некоторым числом (часто обнуляют). |
| while | <условие> |  | Открываем цикл типа 'Пока': пока <условие> = true, выполняются действия ниже. |
| ... | ... | ... | Произвольные команды в теле цикла. |
| storeEval | ${i}+1 | i | Увеличиваем счетчик на 1 (или уменьшаем). Для этого вычисляется значение i+1 используя стандартную команду storeEval и результат помещается в переменную i. |
| endWhile |  |  | Конец блока команд в цикле. |
| ... | ... | ... | Некоторые команды ниже цикла. |

**Пример** использования циклов в тесте. **Задача:** реализовать добавление нескольких случайных элементов на форму.

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Target** | **Value** | **Комментарий** |
| <инициализация теста> | ... | ... | Какие-либо предварительной инициализации теста. |
| ... | ... | ... | Некоторые команды выше цикла. |
| store | 0 | i | Обнуляем счетчик цикла i. |
| while | storedVars['i']<=3 |  | Открываем цикл типа 'Пока': пока условие 'storedVars['i']<=3' = true, выполнять действия ниже. |
| click | xPath=//table[@testid='btn\_Добавить'] |  | Нажимаем кнопку меню 'Добавить' для открытия диалога добавления элемента. |
| waitForElementPresent | xPath=//span[contains(text(), 'Добавление элемента)] | 50000 | Ожидаем появления диалога для добавления нового элемента. При этом обычно открывается некоторая форма для заполнения сведений об элементе. |
| <заполняем форму данными> | ... | ... | Команды для заполнения полей формы случайными данными, используя команды randomString и randomInt (обе они из файла расширений user-extensions.js). |
| randomString | 5 | x | Например, генерируем случайную строку длины 5 и сохраняем её в переменную x, для того, чтобы подставить в некоторое текстовое поле. |
| randomInt | 10 | y | Аналогичная команда, но она генерирует строку длины 10, состоящую только из цифр 0..9. |
| type | xPath=//input[contains(@type, 'text')] | ${x} | Печатаем сгенерированную строку x в некоторое текстовое поле. ${x} - подставляет значения x в это поле. |
| ... | ... | ... | Иные команды цикла. |
| click | xPath=//button[contains(text(), 'Сохранить')] |  | Нажимаем кнопку 'Сохранить' в диалоге добавления. |
| pause | 2000 |  | Добавляем небольшую паузу, чтобы команда на добавление элемента была отправлена и при необходимости обновились данные на форме. |
| storeEval | ${i}+1 | i | Увеличиваем переменную счётчик на 1. |
| endWhile |  |  | Команда окончания цикла. |
| ... | ... | ... | Прочие команды ниже цикла. |

**6. Примеры типичных обращений к элементам GWT-формы**

В таблицу ниже сведено описание основных наиболее употребляемых в тестах команд Selenium и комментарии к ним. Подробнее о стандартных командах можно почитать во встроенном справочнике Selenium IDE. В угловые скобки < > взяты выражения, изменяющиеся в зависимости от контекста.

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Target** | **Value** | **Комментарий** |
| open | <URL> |  | Перейти в браузере по указанному в Target URL страницы. |
| click | <xPath элемента> |  | Щелчок левой кнопкой мыши на указанном в Target элементе. |
| waitForPageToLoad | <time> |  | Ожидать полной загрузки контента страницы с таймаутом указанным в поле Target, в миллисекундах. |
| windowMaximize |  |  | Развернуть окно браузера на весь экран. |
| fireEvent | <xPath элемента> | <{focus}, {blur}> | Если установлено значение focus, то команда генерирует событие установки фокуса на указанный элемент и вызывает обработчик значений данного поля, при значении blur - команда генерирует событие снятия фокуса с поля и также вызывает соответствующий обработчик (например, проверки корректности значения, имеющегося в поле). |
| open | ${mainProject}/j\_spring\_security\_logout |  | Команду нужно вставлять в самом конце теста для гарантированного разлогинивания и окончания текущей сессии. |
| click | xPath=(//table[@testid='btn\_201110261212\_Добавить'])/.//button |  | Нажатие на кнопку 'Добавить', используя атрибут @testid, расположенный выше кнопки. |
| type | xpath=//div[label='Код по ОКВЭД:']/div/div/input | 012345 | Запись значения '012345' в текстовое поле, у которого рядом имеется комментарий 'Код по ОКВЭД:'. |
| click | xPath=//label[contains(text(), 'Наличие объекта:')]/..//input |  | Поставить галочку в чекбокс около (справа) надписи 'Наличие объекта'. |
| click | xPath=//legend[span[contains(text(), 'Наличие объекта')]]/input |  | Поставить галочку в чекбокс ранее (слева) от надписи 'Наличие объекта'. |
| click | xPath=//label['Учебная дисциплина:']/..//img |  | Вариант нажатия на кнопку выпадающего списка, справа от надписи 'Учебная дисциплина'. |
| waitForTextPresent | Отображение 1 - \* из \* | 50000 | Один из способов ожидания появления содержимого выпадающего списка, снизу которого отображается количество видимых строк. |
| click waitForElementPresent mouseDown | xPath=(//img[@class='x-form-trigger x-form-trigger-arrow '])[2] xPath=(//div[@role='listitem'])[${i}+1] xPath=(//div[@role='listitem'])[${i}+1] | 50000 | Один из способов выбора значений в комбобоксе по порядку: 1. Нажимаем на кнопку выпадающего списка для комбобокса №2, используя класс. 2. Ожидаем появления (i+1)-й строки в комбобоксе, используя атрибут role. 3. Нажимаем на (i+1)-ю строку в комбобоксе, используя атрибут role. |
| click waitForElementPresent mouseDown | xPath=//img[@class='x-form-trigger x-form-trigger-arrow '] xPath=//div[contains(text(), 'Функциональный филиал ЖД')] xPath=//div[contains(text(), 'Функциональный филиал ЖД')] | 50000 | Можно обращаться к строкам комбобокса по содержимому: 1. Нажимаем на кнопку выпадающего списка для комбобокса №1, используя класс. 2. Ожидаем появления строки в комбобоксе с текстом 'Функциональный филиал ЖД'. 3. Нажимаем на строку в комбобоксе с текстом 'Функциональный филиал ЖД'. |
| click waitForElementPresent mouseDown | xPath=//div[label='Тип здания:']/div/div/img xPath=(//div[contains(@class, 'x-combo-list-item')])[3] xPath=(//div[contains(@class, 'x-combo-list-item')])[3] | 50000 | Еще один пример обращения к строке выпадающего списка: 1. Нажимаем на кнопку выпадающего списка, через лейбл рядом с ним. 2. Ожидаем появления третьей строки комбобокса. 3. Нажимаем на третью строку комбобокса, через класс, содержащий 'x-combo-list-item'. |
| mouseDown | xpath=//span[contains(text(), 'Редактирование записей')]/following::table[@class='x-grid3-row-table']/..//span |  | Выделяем мышью первую строку (с любым содержимым) таблицы, содержащейся в форме диалоге с заголовком 'Редактирование записей'. |
| click | xPath=//span[contains(text(), 'Редактирование подсказки')]/following::button[contains(text(), 'Сохранить')] |  | Нажимаем на кнопку 'Сохранить', находящуюся внутри диалога контекстного меню. |
| mouseDown | xPath=//span[contains(text(), 'Наименование здания')]/following::span[contains(text(), '\_Sel')] |  | Выделяем первую строку таблицы, если у неё в колонке 'Наименование здания' есть текст '\_Sel'. |
| type | xPath=//div[contains(@class, 'focus')]/.//input | Текст | Запись текста в ячейку таблицы, класс которой содержит 'focus'. |
| type | xPath=//input[contains(@class, 'focus')] | Текст | Аналогичное обращение к инпуту ячейки, с атрибутом class, содержащим 'focus'. |
| click | xPath=//span[contains(text(), 'Подтверждение удаления')]/following::button[contains(text(), 'Да')] |  | Нажатие на кнопку 'Да', содержащуюся внутри диалога 'Подтверждение удаления'. |
| click waitForElementNotPresent waitForElementPresent waitForElementNotPresent  assertElementNotPresent | xPath=//button[contains(text(), 'Сохранить')] xPath=//span[contains(text(), 'Добавление здания')] xPath=//div[contains(@class, 'x-component x-abs-layout-container')]/..//img[@class='gwt-Image x-component'] xPath=//div[contains(@class, 'x-component x-abs-layout-container')]/..//img[@class='gwt-Image x-component'] | 50000 50000 50000 | Пример сохранения данных с учетом прогресс-бара: 1. Нажимаем кнопку 'Сохранить' на форме диалога. 2. Ожидаем пока форма диалога исчезнет. 3. Ожидаем появления прогресс-бара. 4. Ожидаем пока прогресс-бар не исчезнет. 5. Проверяем, что не появилась надпись об ошибке. Аналогичные команды и для компоненты редактирования. |
| click waitForElementPresent waitForElementNotPresent assertElementNotPresent | xPath=(//table[@testid='btn\_201110261212\_Обновить'])[2] xPath=//div[contains(text(), 'Загрузка...')] xPath=//div[contains(text(), 'Загрузка...')] //span[contains(text(), 'Ошибка')] | 50000 50000 | Пример обновления данных с учетом прогресс-бара: 1. Нажимаем кнопку 'Обновить'. 2. Ожидаем появления надписи 'Загрузка...' при прогресс баре. 3. Ожидаем исчезания надписи 'Загрузка...'. 4. Проверяем, что не появилась надпись об ошибке. |
| type fireEvent | xPath=//div[label='Наименование здания или комплекса:']/div/div/input xPath=//div[label='Наименование здания или комплекса:']/div/div/input | ${x} blur | Обращение к полю input, на котором "висит биндинг": 1. Печатаем что-нибудь в инпут. Ошибка для поля остается. 2. Действие по перепроверке введенных значений в поле input. Если значения были введены корректные, ошибка для поля пропадает. |
| waitForElementPresent | xPath=//span[contains(@class, 'x-info-header-text')][contains(text(), 'Информация')] | 50000 | Сложное условие span [ ] [ ]  заменяет союз И. Данная команда означает, в частности, ожидание появления всплывающего окна с заголовком 'Информация' об успехе операции. |
| click | xPath=(//div[span[contains(text(), 'Список договоров')]]/..)[2]/..//input[@type='checkbox'] |  | Сложное условие для обращения к элементам внутри таблицы с заголовком (если их на странице несколько). Обращаемся к родителю от заголовка 'Список договоров' (для второго одноименного заголовка) - это будет сама таблица, затем в любом месте внутри неё щелкаем на первый чекбокс. |
| mouseDown | xPath=//span[contains(text(), 'Протокол')]/ancestor::div[@class='x-grid3-viewport']/.//span[contains(text(), '\_Sel')] |  | Способ обращения к ячейкам только данной таблицы (ни выше, ни ниже) содержащим текст '\_Sel' и для которой имеется заголовок столбца - 'Протокол'. |
| click | xPath=//\*[@testid='id123']/ancestor::tr[@class='x-toolbar-left-row']/..//button |  | Способ обращения к элементам выше (или левее) данного. В частности, здесь идёт обращение к элементу (кнопке) имеющему атрибут @testid='id123', которая находится внутри панели инструментов, и затем обращение к первой кнопке этой панели. Может пригодится, если testid есть только у некоторых кнопок. |
| waitForElementPresent | xPath=//span[contains(text(), 'Создание линии электропередач')]/following::div[@class='x-combobox-loading'][contains(@style, 'display: none;')] | 50000 | Пример ожидания окончания загрузки в комбобоксе для диалога с заголовком 'Создание линии электропередач'. При этом пропадает значок загрузки. |
| verifyElementPresent | xPath=//div[contains(@class, 'x-modal x-component')] |  | Проверка того, что диалог открылся в модальном режиме. |
| click | xPath=(//span[contains(text(), '\_Sel')])[last()] |  | Щелчок мыши на последний элемент в строке таблицы, содержащий текст '\_Sel'. |
| verifyElementNotPresent | xPath=//input[contains(@class, 'invalid')] |  | Проверка того, что после ввода данных в поле оно не подсвечивается с ошибкой. |
| verifyElementPresent | //div[@class='x-panel-body'][contains(@style, 'overflow: auto;')] |  | Проверяем наличие скролла. За наличие полосы прокрутки отвечает стиль: style="width: 1006px; height: 251px; overflow: auto;". overflow: auto; - означает, есть или нет полоса прокрутки. |