Конфигурирование и запуск Selenium-тестов JepRia-приложений с помощью TestNG

Оглавление

[1. Общие сведения об используемом ПО 3](#_Toc452025728)

[2. Поддержка автоматизации тестирования приложений на прикладном уровне 3](#_Toc452025729)

[2.1 Присвоение идентификаторов web-элементам 3](#_Toc452025730)

[2.2 Создание интерфейса для тестирования класса (Class Under Test, CUT) 4](#_Toc452025731)

[2.3 Разработка тестов JepRia-приложений 5](#_Toc452025732)

[2.3.1 Тесты, управляемые данными 7](#_Toc452025733)

[2.4 Конфигурирование тестов для TestNG 10](#_Toc452025734)

[2.4.1 Группировка тестов в java-классах 11](#_Toc452025735)

[2.4.2 Фильтрация запускаемых тестов в test-suite-файлах 11](#_Toc452025736)

[2.4.3 Настройка параметров запуска тестов в файле test.properties 13](#_Toc452025737)

[3. Запуск тестов с помощью Ant 14](#_Toc452025738)

[3.1 Определение параметров запуска тестов в командной строке 14](#_Toc452025739)

[3.2 Описание цели для Selenium-тестирования в build-файле 14](#_Toc452025740)

[4. Тестовые отчёты 15](#_Toc452025741)

[5. Дистрибутивы 15](#_Toc452025742)

[5.1 Selenium Standalone Server 15](#_Toc452025743)

[5.2 Браузер Firefox 16](#_Toc452025744)

[6. Литература по теме 16](#_Toc452025745)

# Общие сведения об используемом ПО

Для тестирования JepRia-приложений на уровне пользовательского интерфейса используется:

* *Selenium Standalone Server*, который содержит реализацию WebDriver для наиболее распространённых браузеров, а также все необходимые ему библиотеки и тестовый framework TestNG (ссылки на дистрибутив здесь: 5.1).
* Браузер Firefox (ссылки на дистрибутив здесь: 5.2)

Selenium WebDriver (далее по тексту - просто Selenium), строго говоря, не является средством автоматизации тестирования, он является средством автоматизации работы с браузером, то есть, средством эмуляции действий пользователя, работающего с Web-приложением. Поэтому Selenium может использоваться не только для тестов, но и, например, для разработки роботов, имеющих различное назначение. Тесты используют Selenium как средство воздействия на Web-приложения и как средство анализа реакции Web-приложений на эти воздействия.

Подход к автоматизированному тестированию на основе Selenium, применяющийся в JepRia, основан на разделении функционала поддержки тестов, на два уровня: уровень собственно тестов и уровень автоматизации - эмуляции действий пользователя. При этом в целях максимального упрощения тестов реализация автоматизации приложения вынесена в отдельный уровень системного и прикладного кода, что избавляет создателей тестов (которые не обязательно являются профессиональными разработчиками) от необходимости знания особенностей Selenium.  
Уровень автоматизации тестов - не просто обёртка над Selenium, он поддерживает структуру и особенности поведения JepRia-приложений.

# Поддержка автоматизации тестирования приложений на прикладном уровне

В целях упрощения разработки тестов на Selenium системный и прикладной уровень JepRia предоставляют высокоуровневый интерфейс работы с приложениями: системный уровень поддерживает работу в таких терминах предметной области JepRia как: клиентский модуль, Toolbar, Statusbar, детальная форма, поля детальной формы, списочная форма, и т.п. Прикладной уровень поддерживает работу в терминах предметной области JepRia-приложения, например, с конкретными наборами полей детальной формы. Для такой высокоуровневой поддержки автоматизации на системном уровне создан интерфейс JepRiaModuleAuto и его реализация JepRiaModuleAutoImpl. Высокоуровневую поддержку автоматизации на прикладном уровне проиллюстрируем небольшим фрагментом интерфейса GoodsAuto полноэкранного модуля Goods.

## Присвоение идентификаторов web-элементам

Для идентификации элементов во время автоматизированного тестирования необходимо назначить им идентификаторы (пример из реализации детальной формы):

import static com.technology.jep.jepriashowcase.custom.client.JRSCCustomAutomationConstant.JRSC\_GOODS\_NAME\_TEXT\_FIELD\_ID;

import static com.technology.jep.jepriashowcase.custom.client.JRSCCustomAutomationConstant.JRSC\_GOODS\_TYPE\_COMBOBOX\_FIELD\_ID;

...

public class GoodsDetailFormViewImpl extends DetailFormViewImpl implements GoodsDetailFormView {

...

JepTextField goodsNameTextField = new JepTextField(**JRSC\_GOODS\_NAME\_TEXT\_FIELD\_ID**, goodsText.goods\_detail\_goods\_name());

goodsNameTextField.setMaxLength(255);

JepComboBoxField goodsTypeComboBoxField = new JepComboBoxField(**JRSC\_GOODS\_TYPE\_COMBOBOX\_FIELD\_ID**, goodsText.goods\_detail\_goods\_type());

...

}

## Создание интерфейса для тестирования класса (Class Under Test, CUT)

public interface GoodsAuto extends JepRiaModuleAuto {

/\*\*

\* Заполнение поля *Наименование товара*

\*

\* @param goodsName значение поля *Наименование товара*

\*/

void setGoodsName(String goodsName);

/\*\*

\* Получение значения поля *Наименование товара*

\* @return значение поля *Наименование товара*

\*/

String getGoodsName();

...

/\*\*

\* Заполнение формы создания

\*

\* @param goodsName Наименование товара

\* @param goodsType Тип товара

\* @param unit Единица измерения

\* @param motivation Мотивация

\* @param purchasingPrice закупочная цена

\* @param goodsPhoto фото товара

\* @param goodsSpecification спецификация товара

\*/

void fillCreateForm(

String goodsName,

String goodsType,

String unit,

String motivation,

String purchasingPrice,

String goodsPhoto,

String goodsSpecification);

}

## Разработка тестов JepRia-приложений

Тесты для приложений JepRia разрабатываются для их выполнения на фреймворке [TestNG](http://testng.org/doc/index.html). Ниже приводится пример теста фасадного компонента автоматизации, проверяющего правильность заполнения поля *GoodsName* детальной формы создания методом setGoodsNameOnCreate().

...

public class GoodsAutoTest {

...

private JepRiaShowcaseAutoImpl automationManager;

private GoodsAuto cut;

/\*\*

\* Конфигурирование теста

\*

\* @param baseUrl - URL запуска приложения

\* @param browserName - используемый браузер

\* @param browserVersion - версия браузера

\* @param browserPlatform - платформа, для которой реализован браузер

\* @param jepriaVersion - версия JepRia

\* @param forceNewBrowser - условие запуска нового браузера: если true - запускать

\* @param forceLogin - условие перелогинивания: если true - перелогиниваться

\* @param username - имя пользователя

\* @param password - пароль пользователя

\*/

@Parameters({

"baseUrl",

"browserName",

"browserVersion",

"browserPlatform",

"jepriaVersion",

"forceNewBrowser",

"forceLogin",

"username",

"password"})

@BeforeMethod(groups = {"find", "create", "delete", "edit", "goto", "list", "setAndGetTextField"})

public void setUp(

String baseUrl,

String browserName,

@Optional("fake") String browserVersion,

@Optional("fake") String browserPlatform,

String jepriaVersion,

@Optional("No") String forceNewBrowser,

@Optional("No") String forceLogin,

String username,

String password) {

automationManager = startAutomationManager(automationManager, baseUrl, browserName, browserVersion, browserPlatform, jepriaVersion, forceNewBrowser, forceLogin, username, password);

cut = getCut();

if("Yes".equalsIgnoreCase(forceLogin) || !cut.isLoggedIn()) {

cut.login(username, password);

}

}

/\*\*

\* Действия после окончания тестового метода

\*

\* @param forceNewBrowser - условие запуска нового браузера: если true - запускать

\* @param forceLogin - условие перелогинивания: если true - перелогиниваться

\*/

@AfterMethod(groups = {"find", "create", "delete", "edit", "goto", "list", "setAndGetTextField"})

@Parameters({

"forceNewBrowser",

"forceLogin"})

public void tearDown(

@Optional("No") String forceNewBrowser,

@Optional("No") String forceLogin) {

if("Yes".equalsIgnoreCase(forceNewBrowser)) {

automationManager.stop();

} else {

if("Yes".equalsIgnoreCase(forceLogin) && cut.isLoggedIn()) {

cut.logout();

} else {

cut.goTo(WorkstateEnum.SEARCH); // Приведение модуля в исходное состояние

}

}

}

...

@Test

public void setGoodsNameOnCreate() {

cut.goTo(CREATE);

String GOODS\_NAME = "Вино";

cut.setGoodsName(GOODS\_NAME);

AssertJUnit.assertEquals(GOODS\_NAME, cut.getGoodsName());

}

...

}

### Тесты, управляемые данными

DDT (Data Driven Testing - тесты, управляемые данными) – подход к тестированию, при котором тестовые данные хранятся отдельно от самих тестов, часто в документах Excel, файлах CSV или в базе данных. В TestNG подход DDT поддерживается реализацией *провайдеров данных*, обеспечивающих поток входных данных для тестовых методов. Провайдеры данных реализуются в виде Java-методов, обозначаемых аннотацией *@DataProvider* и возвращающих один из двух типов: Object[][] или Iterator<Object[]>. Для того, чтобы тестовый метод мог работать в режиме DDT, в его аннотации @Test нужно указать атрибут *dataProvider* присвоив ему соответствующее значение, например:

public class LocaleUtilsTest extends Assert {

@DataProvider

public Object[][] parseLocaleData() {

return new Object[][]{

{null, null},

{"", LocaleUtils.ROOT\_LOCALE},

{"en", Locale.ENGLISH},

{"en\_US", Locale.US},

{"en\_GB", Locale.UK},

{"ru", new Locale("ru")},

{"ru\_RU\_some\_variant", new Locale("ru", "RU", "some\_variant")},

};

}

@Test(dataProvider = "parseLocaleData")

public void testParseLocale(String locale, Locale expected) {

final Locale actual = LocaleUtils.parseLocale(locale);

assertEquals(actual, expected);

}

}

В приведённом примере тест *testParseLocale* будет вызван столько раз, сколько массивов данных-параметров будет передано ему из провайдера данных *parseLocaleData* - в данном случае - 7 раз. Если метод-провайдер данных и тестовый метод находятся в разных классах, то в аннотацию @Test нужно добавить атрибут, указывающий класс, в котором находится провайдер данных (в данном случае - *LocaleUtilsTestData.class*):

public class LocaleUtilsTest extends Assert {

...

@Test(dataProvider = "parseLocaleData, dataProviderClass = LocaleUtilsTestData.class")

public void testParseLocale(String locale, Locale expected) {

final Locale actual = LocaleUtils.parseLocale(locale);

assertEquals(actual, expected);

}

}

Наибольшая гибкость при тестировании, управляемом данными, достигается, когда для каждого тестового метода можно указать отдельный поток входных параметров. В JepRia это делается при помощи использования файлового провайдера данных *dataFromFile* и дополнительной аннотации *@DataProviderArguments*, в которой указывается файл данных провайдера *dataFromFile*для соответствующего тестового метода. При этом провайдер данных и тестовый метод выглядят следующим образом:

...

public class JepFileDataProvider {

@DataProvider(name = "dataFromFile")

public static Iterator getDataFromFile(Method testMethod) throws Exception {

Map arguments = DataProviderUtils.resolveDataProviderArguments(testMethod);

List lines = JepFileDataProvider.getRawLinesFromFile(arguments.get("filePath"));

List data = new ArrayList();

for (String line : lines) {

line = line.trim();

if(line.length() == 0 || line.charAt(0) == '#') { // Пропускаем комментарий

continue;

}

data.add(line.split("\\|"));

}

return data.iterator();

}

...

/\*\*

\* Тест заполнения формы создания

\*/

@DataProviderArguments("filePath=test/resources/com/technology/jep/jepriashowcase/goods/auto/GoodsAutoTest.fillCreateForm.method.data")

@Test(dataProviderClass = JepFileDataProvider.class, dataProvider="dataFromFile")

public void fillCreateForm(String goodsName, String goodsType, String unit, String motivation, String purchasingPrice) {

cut.goTo(CREATE);

cut.fillCreateForm(

goodsName,

goodsType,

unit,

motivation,

purchasingPrice);

assertEquals(goodsName, cut.getGoodsName());

assertEquals(goodsType, cut.getGoodsType());

assertEquals(unit, cut.getUnit());

assertEquals(motivation, cut.getMotivation());

assertEquals(purchasingPrice, cut.getPurchasingPrice());

}

#### Формат тестовых файлов данных

Правила создания тестовых файлов данных хорошо иллюстрируются примером файла *GoodsAutoTest.fillCreateForm.method.data*, использующегося для тестирования заполнения формы создания в модуле Goods. Тестовый файл данных представлят собой обычный текст и выглядит следующим образом:

#Наименование товара|Тип товара|Единицы измерения|Мотивация|Закупочная цена

Вино|Продукты питания|Литры|Процент с дохода|12345.12

Водка|Продукты питания|Литры|Процент с дохода|100

Каждая строка файла описывает параметры одного вызова тестового метода. Все параметры, разумеется, текстовые. В качестве разделителя параметров используется символ '|'. Строки, начинающиеся c '#', считаются комментариями. Первую строку файла данных целесообразно использовать для комментария, описывающего формат строки параметров.

#### Именование тестовых файлов данных

Правила именования, приводимые ниже, не имеют обязательного характера, но полезны для системной организации тестовых данных. Файлы данных могут создаваться для группы тестовых методов (имеющих одно и то же значение атрибута *group*), в этом случае они именуются с использованием расширения *group*:   
*TestClass.GroupName.group.data*, где *TestClass* - имя Java-класса, содержащего тест, а *GroupName* - имя TestNG-группы, для которой этот файл предназначен, например, ***GoodsAutoTest.create.group.data***.   
  
Если файл данных создаётся для одного или нескольких тестовых методов, не объединённых группой, он именуется с использованием расширения *method*:   
*TestClass.testMethodName.method.data*, где *testMethodName* - имя теста, для которого предназначен этот тестовый файл данных, например, ***GoodsAutoTest.fillCreateForm.method.data***.

#### Размещение тестовых файлов данных

Правила размещения файлов, приводимые ниже, не имеют обязательного характера, но полезны для системной организации тестовых данных. В аннотации *@DataProviderArguments* указывается относительный путь к тестовому файлу данных. Размещение тестовых данных выполняется по схеме:   
*%APP\_HOME%/%TEST\_RESOURSE\_HOME%/%TEST\_CLASS\_JAVA\_PACKAGE%/TEST\_DATA\_FILE*, где   
*APP\_HOME* - директория App-части приложения   
*TEST\_RESOURSE\_HOME* - директория тестовых данных App-части приложения   
*TEST\_CLASS\_JAVA\_PACKAGE* - директория, соответствующая директории пакета тестового Java-класса.   
*TEST\_DATA\_FILE* - тестовый файл данных

## Конфигурирование тестов для TestNG

ТеstNg отличается возможностью очень гибкого конфигурирования в xml-описателях конфигурации тестирования (далее по тексту - в test-suite-файлах) наборов используемых тестовых методов: при помощи указания:

* используемых тестовых Java-классов
* используемых тестовых методов тестовых классов
* произвольно создаваемых групп тестовых методов

В дополнение к этому стоит отметить возможность запуска из ant и командной строки произвольного «набора наборов» TestSuites.

### Группировка тестов в java-классах

Ниже приведен пример размещения тестового метода *find* в группах *find* и *mandatory,* осуществляемого при помощи параметра аннотации *@Test groups*:

/\*\*

\* Тест операции поиска по пустому шаблону

\*/

@Test(groups = {"find", "mandatory"})

**public** **void** find() {

. . .

}

В конфигурационных test-suite-файлах указание имён таких групп используется для фильтрации запускаемых тестовых методов.

### Фильтрация запускаемых тестов в test-suite-файлах

Ниже приводится пример конфигурационного test-suite-файла:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="GoodsAutoTestSuite" parallel="false">

<test name="GoodsAutoTest">

<parameter name="baseUrl" value="http://${host}/{$appName}{$testModule}"/>

<parameter name="browserName" value="${browserName}"/>

<parameter name="browserPath" value="${browserPath}"/>

<parameter name="username" value="${testUsername}"/>

<parameter name="password" value="${testPassword}"/>

<parameter name="jepriaVersion" value="${jepriaVersion}"/>

<parameter name="forceNewBrowser" value="No"/>

<parameter name="forceLogin" value="No"/>

<groups>

<define name="all">

<include name="find" />

<include name="create" />

<include name="delete" />

<include name="edit" />

<include name="goto" />

<include name="list" />

</define>

<run>

<!-- <include name="all" /> -->

<!-- <include name="edit" /> -->

<include name="setAndGetTextField" />

</run>

</groups>

<classes>

<class name="com.technology.jep.jepriashowcase.goods.auto.GoodsAutoTest">

<methods>

<!-- <include name="fillSearchForm" /> -->

…

</methods>

</class>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

где в теге *<test>* определяются параметры и набор запускаемых тестов.

Параметры запуска задаются в тегах *<parameter>*.

В test-suite-файлах множество запускаемых тестов определяется тегами *<groups>* и *<classes>*.

#### Теги <parameter>

В тегах *<parameter>* определяются *параметры запуска тестов* (не путать с параметрами тестов).

Параметры запуска тестов описывают основные условия запуска: тип браузера (chrome, firefox, …), на котором будут выполняться тесты, пользователя, из-под которого будет выполняться тестовый вход и т.п.

#### Тег <groups>

##### Тег <define>

В теге *<define>* можно определять дополнительные – составные запускаемые группы тестов.

##### Тег <run>

В теге *<run>* определяется набор запускаемых групп тестов.

#### Тег <classes>

В теге *<classes>* определяется набор java-классов, содержащих тестовые методы.

##### Тег <сlass>/<methods>

В подтеге *<class>/<methods>* с помощью тега *<include>* определяется набор используемых методов java-классов.

Быстрое изменение запускаемого набора тестов (фильтрацию тестов) удобно выполнять комментированием/раскомментированием строк файла (с тегами *<include>*).

### Настройка параметров запуска тестов в файле test.properties

В директории App прикладного приложения находится файл test.properties, содержащий параметры запуска тестов:

host=srvt16.d.t # имя сервера

appName=JepRiaShowcase # имя приложения на сервере

browserName=firefox # имя браузера, с помощью которого будет проводиться тестирование

browserPath=C:/path/to/firefox.exe # путь до исполняемого файла браузера

jepriaVersion=9.0.0 # (задел на будущее?)

# Обязателен любой непустой общий постфикс в названиях свойств

testModule01=/JepRiaShowcase.jsp?em=Goods

# адрес тестируемого модуля (полный URL = host + "/" + appName + testModule01)

testSuite01=\*\*/auto/\*\*/GoodsAutoTest.xml

# маска xml-файлов (test suite), которые будут участвовать в тестировании

testUsername01=NagornyyS # пользователь для входа в тестируемый модуль

testPassword01=123 # пароль для входа в тестируемый модуль

Последнюю группу параметров можно размножать, изменяя общий постфикс. Это полезно, например, для тестирования различных модулей одного и того же приложения, или для входа под разными пользователями.

# Запуск тестов с помощью Ant

Запуск selenium-тестов приложений JepRia из командной строки с помощью Ant выполняется командой

ant test-by-selenium

## Определение параметров запуска тестов в командной строке

Иногда бывает нужно выполнить быструю проверку работы тестов в конфигурациях, отличных от той, которая определена в используемых test-suite. Для этого не обязательно редактировать test-suite.

Перекрыть параметры запуска тестов (см. 3) текущих test-suite можно параметрами команды запуска ant – эти параметры имеют больший приоритет, чем соответствующие параметры test-suite.

Для того, чтобы это сделать, при запуске ant нужно задать нужные внешние переменные с использованием ключа *-D*, например, значение параметра *browserName* можно изменить командой

ant –DbrowserName=”chrome” test-by-selenium

## Описание цели для Selenium-тестирования в build-файле

Цель (target) ***test-by-selenium*** для Selenium-тестирования JepRia-приложения определяется следующим образом:

<target name="test-by-selenium" depends="testng-compile">

<taskdef classname="org.testng.TestNGAntTask" name="testng">

<classpath>

<pathelement location="${SELENIUM\_HOME}/libs/testng-6.8.5.jar" />

</classpath>

</taskdef>

<propertyselector property="modules" match="testModule(.\*)" select="\1"/>

<for list="${modules}" param="suiteId">

<sequential>

<testng classpathref="run-testng-libs"

outputDir="${testng.report.dir}"

haltOnFailure="true"

verbose="2"

delegateCommandSystemProperties="true">

<classpath location="${test.src.dir}" />

<classpath location="${JEP\_RIA}/lib/jepria-test.jar" />

<classpath refid="selenium-libs" />

<sysproperty key="browserName" value="${browserName}"/>

<sysproperty key="browserPath" value="${browserPath}"/>

<sysproperty key="jepriaVersion" value="${jepriaVersion}"/>

<sysproperty key="host" value="${host}"/>

<sysproperty key="appName" value="${appName}"/>

<sysproperty key="testModule" value="${testModule@{suiteId}}"/>

<sysproperty key="testUsername" value="${testUsername@{suiteId}}"/>

<sysproperty key="testPassword" value="${testPassword@{suiteId}}"/>

<xmlfileset dir="${test.conf.dir}" includes="${testSuite@{suiteId}}"/>

</testng>

</sequential>

</for>

</target>

Группа тегов *<sysproperty>* используется для определения внешних переменных, передаваемых в test-suite (см. соответствующую группу тегов *<parameter>* в описателе test-suite: 2.4.2). Значения этих переменных задаются в файле свойств ant *test.properties*, доступном из *build.xml* (см.[Настройка параметров запуска тестов в файле test.properties](#_Настройка_параметров_запуска)).

# Тестовые отчёты

Отчёт о результатах прохождения последнего запуска тестов находится в директории приложения *App/test-output* и доступен по ссылке [Отчёт о прохождении тестов](file:///D:\workspace\git\JEP\Module\JepRiaShowcase\App\test-output\index.html)

# Расширенное конфигурирование тестов

## Добавление нового параметра в test.properties для его использования при тестировании

Для добавления нового параметра (например, с именем newVar) в test.properties необходимо определить его в следующих местах кода[[1]](#footnote-1):

* Непосредственно в test.properties  
    
  newVar0=Its Value
* В build.xml, в цели test-by-selenium в раздел объявления sysproperty  
    
  <sysproperty key="newVar1" value="${newVar0}"/>
* В ModuleNameAutoTest.xml, в раздел определения параметров теста  
    
  <suite...>  
   <test...>  
   <parameter name=*"newVar2"* value=*"${newVar1}"*/>  
   </test>  
  </suite>
* В аннотацию @Parameters метода setUp класса JepAutoTest, в список явных параметров этого метода и (*обязательно ли?*) во все переопределения этого метода наследниками указанного класса  
    
  @Parameters({..., "newVar2", ...})  
  **public** **void** setUp(..., *[@Optional]* String newVar3, ...) {  
   useVar(newVar3);  
  }
* В список явных параметров конструктора класса WDAutoAbstract и в список set-методов в его теле  
   **public** WDAutoAbstract(..., String newVar4, ...) {  
   ...  
   *set*(*NEW\_VAR\_KEY*, newVar4);  
   ...  
  }  
  + Константа создается в классе JepAutoProperties  
      
    **public** **static** String *NEW\_VAR\_KEY* = "newVar5";

В этом случае новый параметр можно использовать в классе WebDriverFactory с помощью метода *get*(*NEW\_VAR\_KEY*).

# Дистрибутивы

## Selenium Standalone Server

Используемый дистрибутив 2.43.1 находится здесь:

[X:\!IT DEPARTMENT\Divisions\Applications\Distrib\Selenium\selenium-2.43.1](file:///X:\!IT%20DEPARTMENT\Divisions\Applications\Distrib\Selenium\selenium-2.43.1)

Последние версии скачиваются с официальной страницы отсюда: <http://www.seleniumhq.org/download/>.

## Браузер Firefox

Не все версии Firefox нормально работают с Selenium.

Firefox 33 работает нормально, лежит здесь:

[X:\!IT DEPARTMENT\Divisions\Applications\Distrib\FireFox\Firefox Setup 33.0.exe](file:///X:\!IT%20DEPARTMENT\Divisions\Applications\Distrib\FireFox\Firefox%20Setup%2033.0.exe)

# Литература по теме

* [Что такое Selenium?](http://habrahabr.ru/post/152653/)
* [Selenium 2.0 и WebDriver](http://selenium2.ru/docs/webdriver.html)
* [Тестирование в Java. TestNG](http://habrahabr.ru/post/121234/0)

1. В данном примере использованы разные индексы переменной newVar исключительно для демонстрации непересекающихся пространств имен одной и той же переменной. Разумеется, все упоминания имени переменной могут (*и должны*) быть одинаковы, без индексов вообще. [↑](#footnote-ref-1)