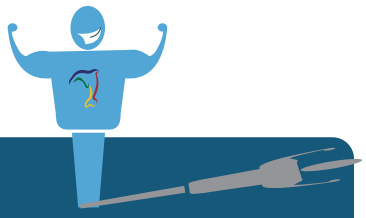
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

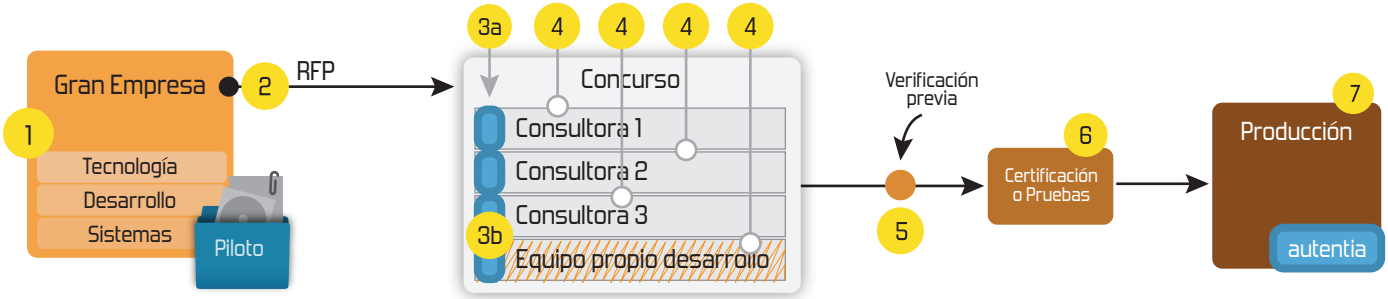
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

DESARROLLADO POR:
Juan Alonso Ramos

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero Técnico en Informática de Gestión e Ingeniero en Informática, especialidad en Ingeniería del Software

Puedes encontrarme en Autentia: Ofrecemos de servicios soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Fecha de publicación del tutorial: 2009-02-26

Introducción a Selenium 2 y WebDriver

Índice de contenidos.

1. Introducción

Selenium es un conjunto de herramientas para automatizar test en aplicaciones web con las que poder grabar, configurar, lanzar y comprobar que nuestras aplicaciones web hacen lo que realmente deben hacer. Continuando con nuestros tutoriales sobre testing y concretamente con pruebas con Selenium, en esta ocasión veremos las novedades que trae la versión 2.

Algunas de las nuevas características a destacar son:

1. Inclusión de WebDriver dentro de Selenium. La principal contribución de WebDriver son los controladores nativos que dan soporte a distintos navegadores (Internet Explorer, Firefox, Chrome y próximamente Opera y Safari). Debido a que Selenium tiene las restricciones propias del Javascript (lenguaje con el que está hecho), WebDriver va más allá y dependiendo del navegador que queramos probar utiliza el mecanismo más apropiado, por ejemplo en Firefox se implementa como una extensión, para Internet Explorer hace uso de los objetos propios de automatización. Incluso puede hacer uso de las características de los navegadores desde el punto de vista del sistema operativo. Gracias a WebDriver ya no necesitamos de un navegador web real para lanzar los test sino que utiliza una aplicación basada en HtmlUnit para simular el navegador. A partir de la versión 2 de Selenium disponemos de toda la funcionalidad que teníamos hasta ahora y adicionalmente, si la necesitamos, las capacidades de WebDriver.
2. Permite test de Selenium en dispositivos móviles iPhone y Android. Cada vez son más populares este tipo de dispositivos por lo que Selenium 2 incluye un emulador para poder testear las aplicaciones.
3. Dispone de una API más sencilla. Una simplificación del interfaz con dos clases principales: WebDriver (para el control de los distintos navegadores) y WebElement (para los elementos que componen la página web).
4. Arquitectura mejorada. La nueva arquitectura introduce una seria de características para facilitar la realización de los test: incluye un teclado nativo, soporte a eventos de ratón, manejo de popups, frames, etc.
5. Conexión remota a navegadores en otras máquinas por si en el entorno de pruebas no se dispusiera de ellos.
6. Ejecución de Javascript, acceso al objeto Window, temporizadores, clases con funcionalidad básica para facilitar los test, etc.
7. Compatibilidad de los test de Selenium en JUnit a la nueva API de WebDriver a través de la clase WebDriverBackedSelenium

2. Entorno

- MacBook Pro 15' (2.4 GHz Intel Core i5, 4GB DDR3 SDRAM).
- Sistema Operativo: Mac OS X Snow Leopard 10.6.6
- Java 1.6.0_22
- Maven 3.0.2
- Selenium 2.0 Beta 2
- JUnit 4.8.2

3. Crear un proyecto de pruebas

Para empezar las pruebas lo primero que debemos tener es un proyecto por lo que usaremos maven para crear uno. Para ello utilizamos un arquetipo mediante el comando **mvn archetype:generate**. Concretamente utilizaremos el archetipo 101 que nos creará una proyecto maven estándar.

Terminal — java — 98×23

```
94: remote -> maven-archetype-j2ee-simple (An archetype which contains a simplifed sample J2EE app
lication.)
95: remote -> maven-archetype-marmalade-mojo (-)
96: remote -> maven-archetype-mojo (An archetype which contains a sample a sample Maven plugin.)
97: remote -> maven-archetype-plugin (An archetype which contains a sample Maven plugin.)
98: remote -> maven-archetype-plugin-site (An archetype which contains a sample Maven plugin site.
 This archetype can be layered upon an
    existing Maven plugin project.)
99: remote -> maven-archetype-portlet (An archetype which contains a sample JSR-268 Portlet.)
100: remote -> maven-archetype-profiles (-)
101: remote -> maven-archetype-quickstart (An archetype which contains a sample Maven project.)
102: remote -> maven-archetype-site (An archetype which contains a sample Maven site which demonst
rates some of the supported document types like
    APT, XDoc, and FML and demonstrates how to i18n your site. This archetype can be layered
    upon an existing Maven project.)
103: remote -> maven-archetype-site-simple (An archetype which contains a sample Maven site.)
104: remote -> maven-archetype-webapp (An archetype which contains a sample Maven Webapp project.)
105: remote -> myfaces-archetype-helloworld (Archetype to create a new webapp based on myfaces)
106: remote -> myfaces-archetype-helloworld-facelets (Archetype to create a new webapp based on My
Faces and Facelets)
107: remote -> myfaces-archetype-helloworld-portlets (Archetype to create a new portlet webapp bas
ed on myfaces)
108: remote -> myfaces-archetype-helloworld20 (Archetype to create a new webapp based on MyFaces 2
```

Una vez creado el arquetipo del proyecto abrimos el pom.xml y añadimos la dependencia del selenium-server. Selenium-server a su vez tiene las dependencias al selenium-htmlunit-driver, selenium-ie-driver, selenium-firefox-driver, selenium-chrome-driver, selenium-iphone-driver, etc. Si no se ve necesaria la inclusión de alguna de estas librerías se pueden excluir aunque como estas dependencias están con el scope de test no pasa nada porque nos las incluya. De paso actualizamos la versión de JUnit a la 4.8.2 en el pom.xml

```
01 <dependency>
02     <groupid>org.seleniumhq.selenium</groupid>
```

```
    <artifactid>selenium-server</artifactid>
    <version>2.0b2</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupid>junit</groupid>
    <artifactid>junit</artifactid>
    <version>4.8.2</version>
    <scope>test</scope>
</dependency>
```

4. Crear un test básico

Un primer test para irnos familiarizando con el API sería conectarnos a la web de AdictosAlTrabajo y comprobar que llegamos a ella, por ejemplo comprobando el título de la página:

```
public class SeleniumTest {

    private WebDriver driver;

    @Before
    public void setUp() {
        driver = new HtmlUnitDriver();
    }

    @After
    public void tearDown() {
        driver.close();
    }

    @Test
    public void connectToAdictos()  {
        driver.get("http://www.adictosaltrabajo.com");

        Assert.assertEquals("Adictos al Trabajo. Formación y desarrollo | JAVA, JEE, UML, XML |. Tutoriales sobre nuevas tecnologias.",
                            driver.getTitle());
    }
}
```

El código resultante del test es sencillo, basta con crear una instancia del driver correspondiente que hace las veces de motor o controlador del navegador. En este caso para no tener necesidad de utilizar un navegador se ha utilizado el HtmlUnitDriver que es un simulador de un navegador en memoria. Utilizando el método get le decimos que se conecte al dominio indicado y posteriormente comprobamos que el título es el esperado mediante un método assert.

Recordar que si en la máquina donde se lancen los test no está instalado un navegador de los soportados por WebDriver y en un test se utiliza el driver de ese navegador, el test fallará. Por ejemplo si en un Mac lanzamos un test que utilice el InternetExplorerDriver (sólo disponible desde Windows), este test fallará debido a que el driver no encontrará este programa instalado en el sistema operativo dando el siguiente error: org.openqa.selenium.WebDriverException: java.io.IOException: Unable to locate: IEDriver.dll. Por ello es importante tener en cuenta el entorno de ejecución de los test. Para solucionar este problema se puede lanzar el test contra un servidor remoto que sí tenga instalado el sistema operativo. Esto lo veremos en futuros tutoriales.

Si lo que queremos es probar nuestra aplicación desde un entorno no Windows pero con el driver de InternetExplorer también se le puede indicar al HtmlUnitDriver que simule este navegador con la versión que queramos:

```
final WebDriver driver = new HtmlUnitDriver(BrowserVersion.INTERNET_EXPLORER_6);
```

5. Explorando el API para interactuar con la página HTML

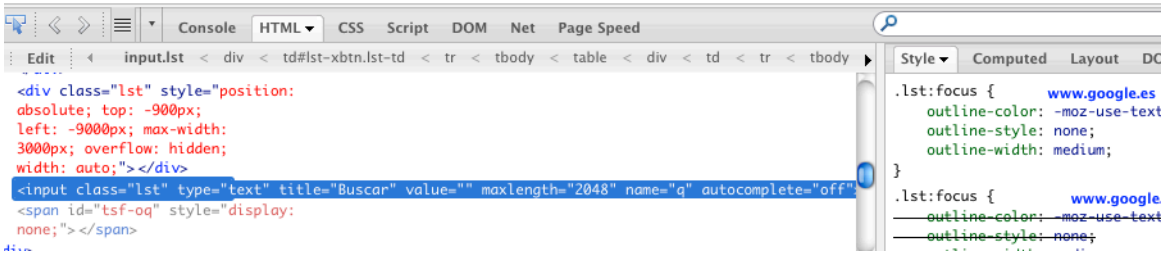
Hemos visto hasta ahora la introducción a Selenium 2 que incluye el API WebDriver y hemos creado un proyecto de test. Hasta hemos hecho nuestro primer test sencillo de conexión a una página web. Ahora vamos a mostrar un poco más las opciones de este API para las pruebas de aplicaciones web.

En el test anterior utilizamos la clase WebDriver que nos da soporte para conectarnos a nuestra aplicación web. Una vez que nos conectamos podemos interactuar con la página y los distintos elementos que la componen: elementos de un formulario, enlaces, capas, etc:

```
<input type="text" name="usuario" id="idUser" class="inputClass">
```

Para interactuar con la caja de texto hay múltiples formas, desde buscarla dentro del DOM por nombre (name), por identificador (id), selector css indicado explicitamente por el atributo (class), selector css aplicable al componente (input.clase), nombre de etiqueta (input), etc. Para las pruebas con la página de Google el código quedaría asi:

```
<input type="text" title="Buscar" value="" maxlength="2048" name="q" class="lst" autocomplete="off">
```



Para interactuar con esta caja de texto podemos utilizar cualquiera de las siguientes opciones:

```
@Test
public void connectToGoogle()  {
    final WebDriver driver = new HtmlUnitDriver();
    driver.get("http://www.google.es");

    Assert.assertEquals("Google", driver.getTitle());

    //<input type="text" title="Buscar" value="" maxlength="2048" name="q" class="lst" autocomplete="off">

    WebElement webElementById = driver.findElement(By.name("q"));
    WebElement webElementByClassName = driver.findElement(By.className("lst"));
    WebElement webElementByCssSelector = driver.findElement(By.cssSelector("input[name=\"q\"]"));
    WebElement webElementByTagName = driver.findElement(By.tagName("input"));
    WebElement webElementByXPath = driver.findElement(By.xpath("//*[@name=\"q\"]"));
}
```

Una vez que accedemos a los elementos de la página podemos rellenar los campos de texto, cambiar los valores de los combos, hacer click en los botones, etc. Se muestra a continuación un ejemplo de uso de cómo se rellena un formulario:

El formulario en cuestión:

El código del test que prueba el formulario se encarga de meter los datos en el mismo, llamar al método que hace el click del botón y capturar el alert comprobando que el mensaje es el correcto. Para las pruebas he metido el código de este tutorial en un servidor apache.

```
01 @Test
02 public void fillForm() throws UnknownHostException {
03     final WebDriver driver = new FirefoxDriver();
04     driver.get("http://" + InetAddress.getLocalHost().getHostAddress() + "/~jalonso/selenium/Selenium2.html");
05
06     final String OPTION_MADRID = "Madrid";
07     driver.findElement(By.id("nombre")).sendKeys("Juan");
08     driver.findElement(By.id("apellidos")).sendKeys("Alonso");
09     driver.findElement(By.id("email")).sendKeys("jalonso@autentia.com");
10
11     final List<WebElement> options = driver.findElements(By.tagName("option"));
12     for (WebElement option : options) {
13         if (option.getText().equals(OPTION_MADRID)) {
14             option.setSelected();
15         }
16     }
17
18     final List<WebElement> radios = driver.findElements(By.name("newsletters"));
19     for (WebElement radio : radios) {
20         if (radio.getValue().equals("true")) {
21             radio.setSelected();
22         }
23     }
24
25     driver.findElement(By.id("enviar")).click();
26
27     final Alert alert = driver.switchTo().alert();
28     if (!alert.getText().equals("OK")) {
29         Assert.fail(alert.getText());
30     }
31 }
```

Mediante el método switchTo del driver podemos posicionarnos en los distitos objetos que componen la página web: alert, frame, window, history, cookies. Es importante destacar que el HtmlUnitDriver no soporta el método alert.

HtmlUnit no utiliza el mismo motor JavaScript y la interpretación del DOM que el resto de navegadores por lo que los resultados de las pruebas podrían no ser iguales. Esta interpretación, aunque sigue el estándar marcado por el W3C, tiene sus propias peculiaridades como los demás navegadores. Por defecto el HtmlUnitDriver tiene el JavaScript deshabilitado por lo que para habilitarlo bastaría con llamar a este constructor **public HtmlUnitDriver(boolean enableJavascript)** o bien utilizar el método **driver.setJavascriptEnabled(true);**

6. Conclusiones

Una vez más hacemos especial incapié en las pruebas de software. En este caso las pruebas de integración o validación se pueden realizar con Selenium que en su versión 2, a fecha de hoy aún en fase beta, no deja de superarse.

Si la aplicación es especialmente complicada siempre se pueden grabar los test con Selenium IDE y exportar posteriormente a código Java (Junit4) para facilitarnos la tarea de realización del test. Ojo ya que el código que nos genera por defecto es con sintaxis Selenium por lo que será necesario instalar esta extensión que sirve para generar el test con API WebDriver.

Espero que te haya sido de ayuda.

Un saludo. Juan.

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

» **Registrate** y accede a esta y otras ventajas «

COMENTARIOS