

¿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

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

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

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



» Estás en: Inicio Tutoriales JsTestDriver: testea tu código Javascript



Miguel Arlandy Rodríguez

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en [Autentia](#): Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE



[Ver todos los tutoriales del autor](#)



Fecha de publicación del tutorial: 2012-03-23

Tutorial visitado 4 veces [Descargar en PDF](#)

JsTestDriver: testea tu código Javascript.

0. Índice de contenidos.

- 1. Introducción.
- 2. Entorno.
- 3. El escenario.
- 4. Escribiendo los test.
- 5. Arrancando el servidor.
- 6. Ejecutando los test.
- 7. Referencias.
- 8. Conclusiones.

1. Introducción

Todos tenemos muy claro la importancia de tener un código bien testeado que sea robusto y flexible a cambios. Que si cambio una cosa no me cargue el funcionamiento de otra. Estamos acostumbrados a usar frameworks como JUnit para testear nuestro código Java, pero... ¿qué pasa con nuestro código Javascript?. Con Javascript, en caso de que se produzca un error, por norma general no vamos a tener una traza de error en nuestro servidor (aunque hay librerías de log's para js). Lo más probable es que al usuario no le funcione bien la aplicación o que, directamente la pantalla no haga nada (caso de que salte una excepción). Además, en algunas ocasiones, el código no se comporta de la misma forma en diferentes navegadores (sobre todo cuando interactuamos con elementos del DOM).

En este tutorial vamos a intentar explicar cómo testear nuestro código Javascript con diferentes navegadores gracias al framework JsTestDriver.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 15' (2.2 Ghz Intel Core I7, 8GB DDR3).
- Sistema Operativo: Mac OS Snow Leopard 10.6.7
- Entorno de desarrollo: Eclipse 3.7 Indigo.
- JsTestDriver 1.3.4.
- Mozilla Firefox 10.0.2.
- Google Chrome 17.



- Safari 5.1.

3. El escenario.

Supongamos que, en un directorio llamado js de nuestra aplicación, tenemos una librería Javascript llamada validador.js con las siguientes características:

- Contiene una clase a la que hemos llamado Validador, cuyos objetos se construyen recibiendo como parámetro un formulario.
- La clase tiene un método validar que validará todos los elementos del formulario.
- Si un elemento tiene un className igual a obligatorio (class="obligatorio") se comprobará que su valor no sea vacío.
- Si todos los elementos se validaron correctamente devolverá el mensaje "OK".
- Si un elemento no cumpliera con la validación se devolverá un mensaje informando de que ese campo tiene un valor incorrecto.

El código es el siguiente:

```

1  function Validador(formulario) {
2      this.form = formulario;
3  }
4
5  Validador.prototype.validar = function () {
6      var elementos = this.form.childNodes;
7      for (var i = 0; i < elementos.length; i++) {
8          var elemento = elementos[i];
9          if (elemento.className == 'obligatorio' && elemento.value == '') {
10             return "El campo " + elemento.name + " es obligatorio";
11         }
12     }
13     return "OK";
14 };
15
16

```

Y un ejemplo de un formulario que se podría validar con este código sería el siguiente:

```

1  <form action="loquesea">
2      Nombre: <input type="text" class="obligatorio" name="nombre"/>
3      Cargo: <select class="obligatorio" name="cargo">
4          <option value=""></option>
5          <option value="Jefe">Jefe</option>
6          <option value="Becario">Becario</option>
7      </select>
8      Edad: <input type="text" name="edad"/>
9  </form>

```

Obsérvese que los campos Nombre y Cargo están marcados como obligatorios.

4. Escribiendo los test.

Para escribir los tests que se ejecutarán sobre el código del punto anterior, creamos un directorio js-test a la misma altura del directorio js donde teníamos nuestro validador.js. Dentro del dicho directorio, creamos un fichero validador-test.js donde escribiremos nuestros test. Vamos a utilizar como escenario el formulario propuesto en el punto anterior. Probaremos lo siguiente:

- Si se intenta validar un formulario con el valor del campo "nombre" igual a vacío debe devolver un mensaje de error indicando que el nombre es obligatorio.
- Si se intenta validar un formulario con el valor del campo "nombre" distinto a vacío pero con el valor del campo "cargo" igual a vacío debe dar un mensaje de error indicando que dicho campo es obligatorio.
- Si los campos "nombre" y "cargo" no son vacíos debe devolver OK.

El código de nuestro validador-test.js es el siguiente.

```

1  ValidadorTest = TestCase("ValidadorTest");
2
3  ValidadorTest.prototype.testDebeAvisarmeSiElNombreNoEstaPuesto = function() {
4      /*:DOC form = <form action="loquesea">
5          Nombre: <input type="text" class="obligatorio" name="nombre"/>
6          Cargo: <select class="obligatorio" name="cargo">
7              <option value=""></option>
8              <option value="Jefe">Jefe</option>
9              <option value="Becario">Becario</option>
10             </select>
11             Edad: <input type="text" name="edad"/>
12         </form>
13     */
14     var validador = new Validador(this.form);
15     assertEquals("El campo nombre es obligatorio", validador.validar());
16 };
17

```

```

18 ValidadorTest.prototype.testDebeAvisarmeSiElCargoNoEstaPuesto = function() {
19     /*:DOC form = <form action="loquesea">
20         Nombre: <input type="text" class="obligatorio" name="nombre" val
21         Cargo: <select class="obligatorio" name="cargo">
22             <option value=""></option>
23             <option value="Jefe">Jefe</option>
24             <option value="Becario">Becario</option>
25         </select>
26         Edad: <input type="text" name="edad"/>
27     </form>
28     */
29     var validador = new Validador(this.form);
30     assertEquals("El campo cargo es obligatorio", validador.validar());
31 };
32
33 ValidadorTest.prototype.testDebeAvisarmeSiEstaTodoOK = function() {
34     /*:DOC form = <form action="loquesea">
35         Nombre: <input type="text" class="obligatorio" name="nombre" val
36         Cargo: <select class="obligatorio" name="cargo">
37             <option value=""></option>
38             <option value="Jefe" selected="selected">Jefe</option>
39             <option value="Becario">Becario</option>
40         </select>
41         Edad: <input type="text" name="edad"/>
42     </form>
43     */
44     var validador = new Validador(this.form);
45     assertEquals("OK", validador.validar());

```

Sobre el código anterior debemos tener en cuenta las siguientes consideraciones:

- El nombre del test (del método) debe empezar por "test" para que el framework sepa que es un test (ej: testDebeAvisarmeSiEstaTodoOK).
- Obsérvese que para hacer que los test actúen sobre un formulario HTML con unas características concretas lo hacemos introduciéndolo de la forma /*:DOC form = código HTML */ dentro del método del test.
- Si quisiésemos hacer inicializaciones antes de cada test, deberíamos hacerlo con un método llamado setUp (ej: ValidadorTest.prototype.setUp)

5. Arrancando el servidor.

JsTestDriver nos provee de un ligerísimo servidor web donde deben conectarse nuestros navegadores para que el código Javascript que vamos a testear pueda ser ejecutado en ellos y así poder evaluar el comportamiento del código.

Pues bien, lo primero que debemos hacer es [descargarnos la librería](#) que arrancará el servidor y ejecutará los test. Seleccionamos el enlace que pone "Self-contained executable jar".



| Filename ▼ | Summary + Labels ▼ |
|---|--|
| testisolation-nodeeps-1.3.4.b.jar | Coverage Plugin without bundled dependencies Featured |
| coverage-nodeeps-1.3.4.b.jar | Coverage Plugin without bundled dependencies Featured |
| coverage-1.3.4.b.jar | Coverage Plugin Featured |
| JsTestDriver-1.3.4.b-src.jar | Source code Featured |
| JsTestDriver-nodeeps-1.3.4.b.jar | Jar without bundled dependencies Featured |
| JsTestDriver-1.3.4.b.jar | Self-contained executable jar Featured |

Lo siguiente que hacemos es copiar el archivo descargado (.jar) en el directorio desde el que vamos a arrancar el servidor y ejecutar los test.

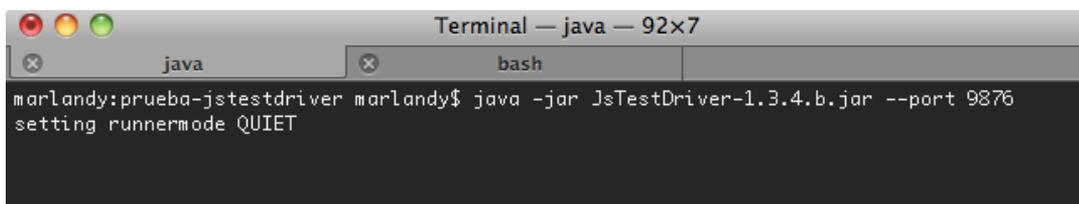
| | |
|--------------------------|-----|
| js | Hoy |
| validador.js | Hoy |
| js-test | Hoy |
| validador-test.js | Hoy |
| JsTestDriver-1.3.4.b.jar | Hoy |
| jsTestDriver.conf | Hoy |

A continuación, creamos un fichero en el mismo directorio donde hemos copiado el jar, llamado jsTestDriver.conf. En este fichero configuraremos las características de la ejecución de nuestros tests. En nuestro caso únicamente le diremos donde está el servidor, la ruta de los ficheros a testear y la ruta de los tests.

```
1 server: http://localhost:9876
2
3 load:
4   - js/*.js
5   - js-test/*.js
```

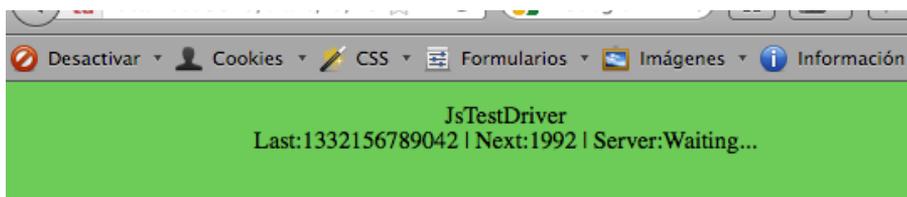
Ahora ya podemos arrancar el servidor. Vamos al directorio donde tenemos el jar y el fichero de configuración y escribimos lo siguiente:

```
1 java -jar JsTestDriver-1.3.4.b.jar --port 9876
```



Y por último nos conectamos con los navegadores que deseamos evaluar al servidor con la siguiente URL: localhost:9876/capture

Nos aparecerá una pantalla como la siguiente:



También podemos hacerlo todo de golpe, arrancando el servidor e indicándole la ruta de los navegadores (separados por comas) con los que queremos testear el código de la siguiente forma:

```
1 java -jar JsTestDriver-1.3.4.b.jar --port 9876 --browser /Applications/Google\ Chrome.ap...
```

6. Ejecutando los test.

Una vez que ya tenemos el servidor arrancado y los navegadores conectados al servidor, lo último que queda es lanzar los test. Para ello, en el directorio del jar y el .conf escribimos lo siguiente:

```
1 java -jar JsTestDriver-1.3.4.b.jar --tests all --verbose
```

```

Terminal — bash — 113x24
marlandy:prueba-jstestdriver marlandy$ java -jar JsTestDriver-1.3.4.b.jar --tests all --verbose
setting runnermode QUIET
Safari 534.54.16 Mac OS [PASSED] ValidadorTest.testDebeAvisarmeSiElNombreNoEstaPuesto
Safari 534.54.16 Mac OS [PASSED] ValidadorTest.testDebeAvisarmeSiElCargoNoEstaPuesto
Safari 534.54.16 Mac OS [PASSED] ValidadorTest.testDebeAvisarmeSiEstaTodoOK
Chrome 17.0.963.79 Mac OS loaded /test/js-test/validador-test.js
Firefox 10.0.2 Mac OS loaded /test/js-test/validador-test.js
Chrome 17.0.963.79 Mac OS [PASSED] ValidadorTest.testDebeAvisarmeSiElNombreNoEstaPuesto
Chrome 17.0.963.79 Mac OS [PASSED] ValidadorTest.testDebeAvisarmeSiElCargoNoEstaPuesto
Chrome 17.0.963.79 Mac OS [PASSED] ValidadorTest.testDebeAvisarmeSiEstaTodoOK
Firefox 10.0.2 Mac OS [PASSED] ValidadorTest.testDebeAvisarmeSiElNombreNoEstaPuesto
Firefox 10.0.2 Mac OS [PASSED] ValidadorTest.testDebeAvisarmeSiElCargoNoEstaPuesto
Firefox 10.0.2 Mac OS [PASSED] ValidadorTest.testDebeAvisarmeSiEstaTodoOK
Total 9 tests (Passed: 9; Fails: 0; Errors: 0) (2,00 ms)
  Firefox 10.0.2 Mac OS: Run 3 tests (Passed: 3; Fails: 0; Errors 0) (2,00 ms)
  Chrome 17.0.963.79 Mac OS: Run 3 tests (Passed: 3; Fails: 0; Errors 0) (2,00 ms)
  Safari 534.54.16 Mac OS: Run 3 tests (Passed: 3; Fails: 0; Errors 0) (1,00 ms)
marlandy:prueba-jstestdriver marlandy$

```

Y como se ve en la imagen, todos los test han pasado en los tres navegadores en los que se ha testeado: Safari, Firefox y Google Chrome.

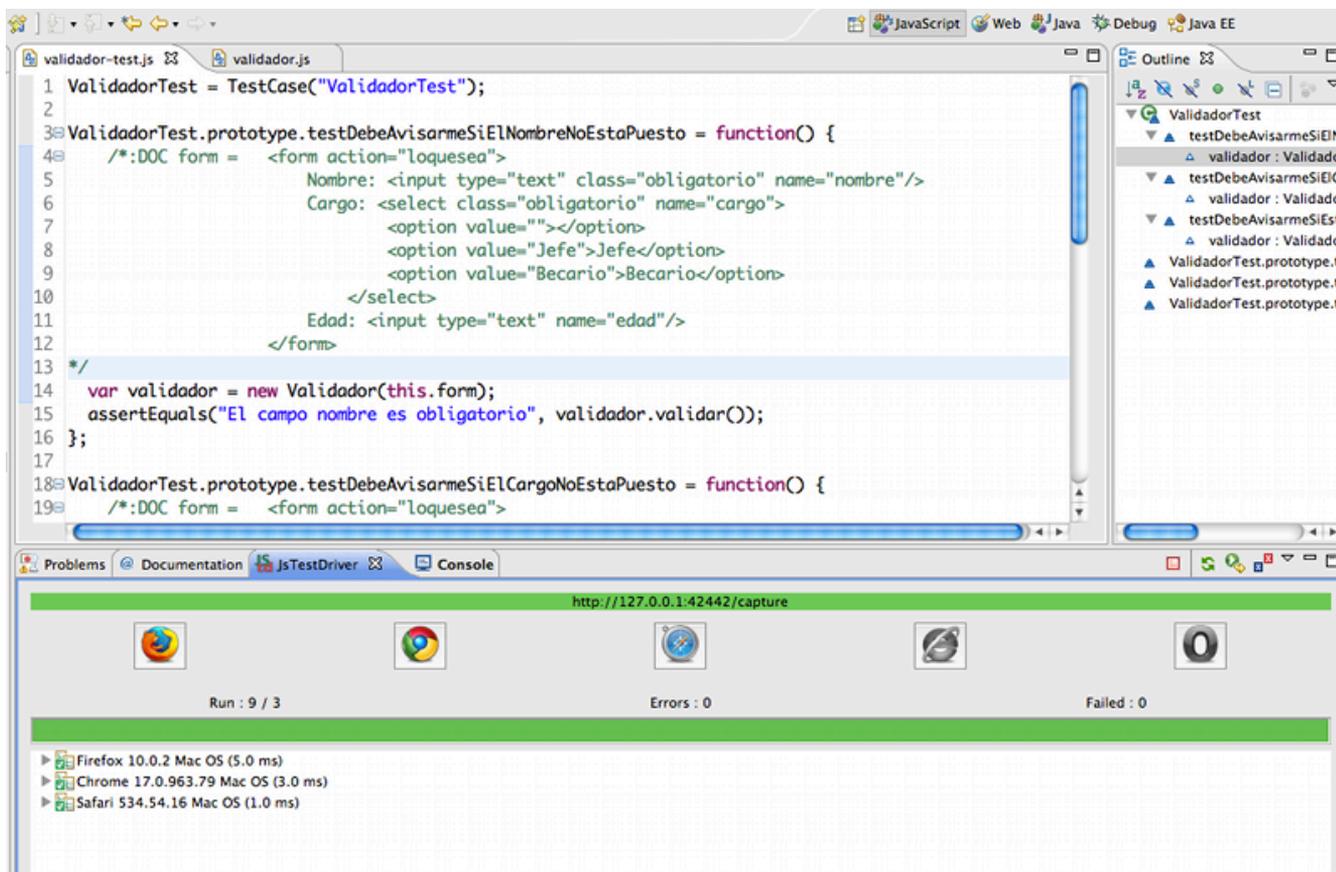
7. Referencias.

- [JsTestDriver](#)
- [Javascript testing with JsTestDriver](#)

8. Conclusiones.

En este tutorial hemos visto cómo podemos testear nuestro código con el framework de test para Javascript JsTestDriven para ayudarnos a que nuestras aplicaciones js sean más robustas y flexibles a cambios. Además cabe destacar la gran facilidad con la que este framework prueba en diferentes navegadores. Recordemos que muchas veces el comportamiento de un navegador no es exactamente igual al de otro (todos nos acordamos de IE...).

Nótese que este framework dispone de plugins para entornos de desarrollo como [Eclipse](#) e [IntelliJ IDEA](#), donde el servidor se arranca y los test's se ejecutan directamente desde el entorno. Aquí os dejo un pantallazo del plugin de Eclipse:



Espero que este tutorial os haya sido de ayuda. Un saludo.

Miguel Arlandy

marlandy@autentia.com

Twitter: @m_arlandy

A continuación puedes evaluarlo:

[Regístrate para evaluarlo](#)

Por favor, vota +1 o compártelo si te pareció interesante

Share |

0

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

» [Regístrate](#) y accede a esta y otras ventajas «



Esta obra está licenciada bajo licencia [Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

PUSH THIS

Page Pushers

Community

Help?

0 people brought clicks to this page

no clicks

+ + + + + + + +

powered by [karmacracy](#)

Copyright 2003-2012 © All Rights Reserved | [Texto legal y condiciones de uso](#) | [Banners](#) | [Powered by Autentia](#) |

