EBook Gratis

APRENDIZAJE protractor

Free unaffiliated eBook created from
Stack Overflow contributors.

#protractor

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: protractor

It is an unofficial and free protractor ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official protractor.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el transportador

Observaciones

Protractor es un marco de prueba de extremo a extremo para aplicaciones AngularJS.

Protractor es una envoltura (construida en la parte superior) alrededor de Selenium WebDriver, por lo que contiene todas las funciones que están disponibles en Selenium WebDriver. Además, Protractor proporciona algunas nuevas estrategias y funciones de localización que son muy útiles para automatizar la aplicación AngularJS. Los ejemplos incluyen cosas como: waitForAngular, By.binding, By.repeater, By.textarea, By.model, WebElement.all, WebElement.evaluate, etc.

Versiones

Versión	Datos de lanzamiento
0.0.1	2016-08-01

Examples

Instalación y configuración del transportador (en Windows)

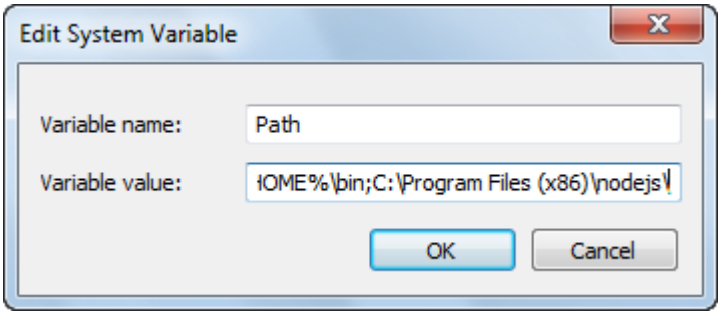
Requisitos: Protractor requiere que se instalen las siguientes dependencias antes de la instalación:

- Java JDK 1.7 o superior
- Node.js v4 o superior

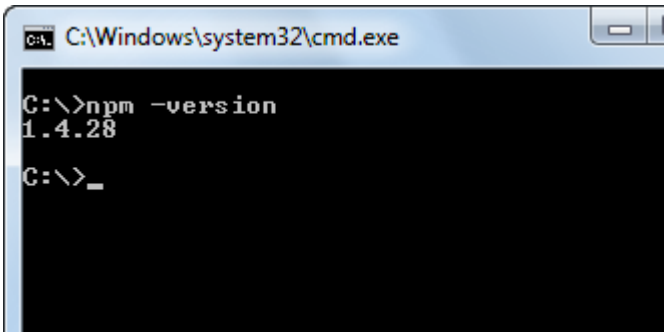
Instalación:

Descargue e instale Node.js desde esta URL: https://nodejs.org/en/

Para ver si la instalación de Node.js es exitosa, puede ir y verificar las variables de entorno. La 'Ruta' en Variables del sistema se actualizará automáticamente.



También puede verificar lo mismo escribiendo el comando `npm -version` en el símbolo del sistema que le dará la versión instalada.



Ahora Protractor se puede instalar de dos maneras: local o globalmente.

Podemos instalar el transportador en una carpeta específica o en la ubicación del directorio del proyecto. Si instalamos en un directorio de proyecto, cada vez que ejecutamos, deberíamos ejecutar desde esa ubicación solamente.

Para instalar localmente en el directorio del proyecto, navegue a la carpeta del proyecto y escriba el comando

```
npm install protractor
```

Para instalar Protractor globalmente ejecute el comando:

```
$ npm install -g protractor
```

Esto instalará dos herramientas de la línea de comandos, el `protractor` y el `webdriver-manager` . Ejecute `protractor --version` para asegurarse de que el transportador se haya instalado correctamente.

`webdriver-manager` se utiliza para descargar los binarios del controlador del navegador e iniciar el servidor Selenium.
Descargue los binarios del controlador del navegador con:

```
$ webdriver-manager update
```

Inicia el servidor de selenio con:

```
$ webdriver-manager start
```

Para descargar el controlador de Internet Explorer, ejecute el comando `webdriver-manager update --ie` en el símbolo del sistema. Esto descargará IEDriverServer.exe en su carpeta de Selenium

Primera prueba usando transportador

El transportador necesita solo dos archivos para ejecutar la primera prueba, el archivo de

especificaciones (código de prueba) y el archivo de configuración. El archivo de especificaciones contiene el código de prueba y el otro contiene detalles de configuración como la ruta del archivo de especificación, los detalles del navegador, la url de la prueba, los parámetros del marco, etc. Para escribir la primera prueba, solo proporcionaremos la dirección del servidor de selenio y la ruta del archivo de especificaciones. Los otros parámetros como el navegador , tiempo de espera, marco se recogerá a los valores predeterminados.

El navegador predeterminado para Protractor es Chrome.

conf.js - archivo de configuración

```
exports.config = {
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['spec.js']
};
```

spec.js - **archivo** Spec (código de prueba)

```
describe('first test in protractor', function() {
  it('should verify title', function() {
    browser.get('https://angularjs.org');

    expect(browser.getTitle()).toEqual('AngularJS — Superheroic JavaScript MVW Framework');
  });
});
```

---

seleniumAddress : ruta al servidor donde se ejecuta el servidor webdriver.

especificaciones : un elemento de matriz que contiene la ruta de los archivos de prueba. Las múltiples rutas pueden ser especificadas por valores separados por comas.

describir - Sintaxis desde el marco de Jasmine . `describe` sintaxis de sta

Escribe una prueba de transportador

Abra una nueva línea de comandos o ventana de terminal y cree una carpeta limpia para probar.

Protractor necesita dos archivos para ejecutarse, un archivo de especificaciones y un archivo de configuración.

Comencemos con una prueba simple que navega al ejemplo de la lista de tareas pendientes en el sitio web de AngularJS y agrega un nuevo elemento de tareas pendientes a la lista.

Copia lo siguiente en `spec.js`

describe ('angularjs homepage todo list', function () {it ('debería agregar un todo', function () {browser.get (' https://angularjs.org ');

```
element(by.model('todoList.todoText')).sendKeys('write first protractor test');
element(by.css('[value="add"]')).click();
```

```
var todoList = element.all(by.repeater('todo in todoList.todos'));
expect(todoList.count()).toEqual(3);
expect(todoList.get(2).getText()).toEqual('write first protractor test');

// You wrote your first test, cross it off the list
todoList.get(2).element(by.css('input')).click();
var completedAmount = element.all(by.css('.done-true'));
expect(completedAmount.count()).toEqual(2);});});
```

Pruebas de funcionamiento selectivo

El transportador puede ejecutar grupos de pruebas de forma selectiva utilizando fdescribe () en lugar de describir ().

```
fdescribe('first group',()=>{
    it('only this test will run',()=>{
        //code that will run
    });
});
describe('second group',()=>{
    it('this code will not run',()=>{
        //code that won't run
    });
});
```

El transportador puede ejecutar pruebas de forma selectiva dentro de los grupos utilizando fit () en lugar de it ().

```
describe('first group',()=>{
    fit('only this test will run',()=>{
        //code that will run
    });
    it('this code will not run',()=>{
        //code that won't run
    });
});
```

Si no hay un ajuste () dentro de un fdescribe (), entonces se ejecutará cada uno (). Sin embargo, un ajuste () lo bloqueará () llamadas dentro del mismo describe () o fdescribe ().

```
fdescribe('first group',()=>{
    fit('only this test will run',()=>{
        //code that will run
    });
    it('this code will not run',()=>{
        //code that won't run
    });
});
```

Incluso si un ajuste () está en un describe () en lugar de un fdescribe (), se ejecutará. Además, cualquier it () dentro de un fdescribe () que no contenga un ajuste () se ejecutará.

```
fdescribe('first group',()=>{
```

```
    it('this test will run',()=>{
        //code that will run
    });
    it('this test will also run',()=>{
        //code that will also run
    });
});
describe('second group',()=>{
    it('this code will not run',()=>{
        //code that won't run
    });
    fit('this code will run',(){
        //code that will run
    });
});
```

Pruebas pendientes

El transportador permite que las pruebas se establezcan como pendientes. Esto significa que el transportador no ejecutará la prueba, sino que generará:

```
Pending:
1) Test Name
Temporarily disabled with xit
```

O, si está deshabilitado con xdescribe ():

```
Pending:
1) Test Name
No reason given
```

Combinaciones

- Un xit () dentro de un xdescribe () generará la respuesta xit ().
- Un xit () dentro de un fdescribe () aún se tratará como pendiente.
- Un ajuste () dentro de un xdescribe () todavía se ejecutará, y ninguna prueba pendiente generará nada.

Transportador: E2E Testing para aplicaciones angulares empresariales.

Instalación y configuración del transportador

Paso 1 : Descargue e instale NodeJS desde aquí. Asegúrese de tener la última versión de nodo. Aquí, estoy usando el nodo v7.8.0. Deberá tener instalado el Kit de desarrollo de Java (JDK) para ejecutar selenium.

Paso 2 : Abra su terminal y escriba el siguiente comando para instalar el transportador global.

```
npm install -g protractor
```

Esto instalará dos herramientas tales como el transportador y el administrador del controlador de

web. Puede verificar la instalación de su transportador siguiendo el comando: `protractor -version`. Si Protractor se instala correctamente, el sistema mostrará la versión instalada (es decir, la versión 5.1.1). De lo contrario, tendrá que volver a verificar la instalación. Paso 3: Actualice el administrador de webdriver para descargar los binarios necesarios.

```
webdriver-manager update
```

Paso 4: El siguiente comando iniciará un servidor Selenium. Este paso ejecutará el administrador del controlador web en segundo plano y escuchará cualquier prueba que se ejecute a través del transportador.

webdriver-manager start Puede ver información sobre el estado del servidor en `http://localhost:4444/wd/hub/static/resource/hub.html`.

Escribiendo el primer caso de prueba usando el transportador:

Antes de saltar a la escritura del caso de prueba, tenemos que preparar dos archivos que son el archivo de configuración y el archivo de especificaciones.

En el archivo de configuración:

```
//In conf.js
exports.config = {
    baseUrl: 'http://localhost:8800/adminapp',
    seleniumAddress: 'http://localhost:4444/wd/hub',
    specs: ['product/product_test.js'],
    directConnect : true,
    capabilities :{
        browserName: 'chrome'
    }
}
```

Comprensión básica de las terminologías utilizadas en el archivo de configuración:

baseUrl - Una URL base para su aplicación bajo prueba.

seleniumAddress : para conectarse a un servidor Selenium que ya se está ejecutando.

Especificaciones - Ubicación de su archivo de especificaciones

directConnect : true - Para conectarse directamente a los controladores del navegador.

Capacidades : si está probando en un solo navegador, use la opción de capacidades. Si está probando en varios navegadores, use la matriz multiCapabilities.

Puedes encontrar más opciones de configuración desde aquí . Han descrito toda la terminología posible con su definición.

En el archivo de especificaciones:

```
//In product_test.js
```

```
describe('Angular Enterprise Boilerplate', function() {
  it('should have a title', function() {
    browser.get('http://localhost:8800/adminapp');
    expect(browser.getTitle()).toEqual('Angular Enterprise Boilerplate');
  });
});
```

Comprensión básica de las terminologías utilizadas en el archivo de especificaciones:

De forma predeterminada, Protractor utiliza el marco jasmine para su interfaz de prueba. La sintaxis de 'describe' y 'it' es de jasmine framework. Puedes aprender más desde aquí. Ejecución del primer caso de prueba:

Antes de ejecutar el caso de prueba, asegúrese de que su administrador de web y su aplicación se ejecuten en diferentes pestañas de su terminal.

Ahora, ejecuta la prueba con:

```
Protractor app/conf.js
```

Debería ver que el navegador Chrome se abre con la URL de su aplicación y se cierra solo. La salida de la prueba debe ser 1 pruebas, 1 aserción, 0 fallas.

¡Bravo! Has ejecutado con éxito tu primer caso de prueba.

Lea Empezando con el transportador en línea:
https://riptutorial.com/es/protractor/topic/933/empezando-con-el-transportador

Capítulo 2: Archivo de configuración del transportador

Introducción

El archivo de configuración contiene información que utiliza Protractor para ejecutar su script de prueba. Aquí trataré de dar algunas variaciones diferentes.

Examples

Archivo de configuración simple - Chrome

```
var config = {};
var timeout = 120000;

config.framework = 'jasmine2';
config.allScriptsTimeout = timeout;
config.getPageTimeout = timeout;
config.jasmineNodeOpts.isVerbose = true;
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;
config.specs = ['qa/**/*Spec.js'];
config.browserName = 'chrome';

exports.config = config;
```

Archivo de configuración con capacidades - Chrome

```
var config = {};
var timeout = 120000;

config.framework = 'jasmine2';
config.allScriptsTimeout = timeout;
config.getPageTimeout = timeout;
config.jasmineNodeOpts.isVerbose = true;
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;
config.specs = ['qa/**/*Spec.js'];
config.capabilities = {
    browserName: 'chrome',
    'chromeOptions': {
        'args': ['start-minimized', 'window-size=1920,1080']
    }
};

exports.config = config;
```

archivo de configuración shardTestFiles - Chrome

Esta configuración le permite ejecutar sus archivos de especificaciones totales en dos instancias de navegador en paralelo. Ayuda a reducir el tiempo total de ejecución de la prueba. Cambie las

maxInstances en función de su necesidad.

Nota : *asegúrese de que sus pruebas son independientes.*

```
var config = {};
var timeout = 120000;

config.framework = 'jasmine2';
config.allScriptsTimeout = timeout;
config.getPageTimeout = timeout;
config.jasmineNodeOpts.isVerbose = true;
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;
config.specs = ['qa/**/*Spec.js'];
config.capabilities = {
    browserName: 'chrome',
    shardTestFiles: true,
    maxInstances: 2,
    'chromeOptions': {
        'args': ['start-minimized', 'window-size=1920,1080']
    }
};

exports.config = config;
```

Emulación de múltiples capacidades del archivo de configuración - Chrome

```
var config = {};
var timeout = 120000;

config.framework = 'jasmine2';
config.allScriptsTimeout = timeout;
config.getPageTimeout = timeout;
config.jasmineNodeOpts.isVerbose = true;
config.jasmineNodeOpts.defaultTimeoutInterval = timeout;
config.specs = ['qa/**/*Spec.js'];
config.multiCapabilities = [{
        browserName: 'chrome',
        shardTestFiles: true,
        maxInstances: 2,
        'chromeOptions': {
            'args': ['start-minimized', 'window-size=1920,1080']
        }
    },
    {
        browserName: 'chrome',
        shardTestFiles: true,
        maxInstances: 1,
        'chromeOptions': {
            'args': ['show-fps-counter=true'],
            'mobileEmulation': {
                'deviceName': 'Apple iPhone 6'
            }
        }
    }
];

exports.config = config;
```

Lea Archivo de configuración del transportador en línea:
https://riptutorial.com/es/protractor/topic/9745/archivo-de-configuracion-del-transportador

Capítulo 3: Depurador del transportador

Sintaxis

- browser.pause ()
- browser.debugger ()

Observaciones

Esta sección explica cómo podemos depurar las pruebas del transportador.

Examples

Usando browser.pause ()

El método `pause()` es una de las soluciones más fáciles que ofrece Protractor para depurar el código. Para usarlo, debe agregarlo en el código donde desea pausar la ejecución. Una vez que la ejecución se encuentra en estado de pausa:

1. Puedes usar `c` (tipo C) para avanzar. Tenga cuidado al usarlo, debe escribir este comando sin demora ya que podría obtener un error de tiempo de espera de su biblioteca de afirmaciones si se demora en presionar c.

2. Escriba `repl` para entrar en el modo interactivo. El modo interactivo se utiliza para enviar comandos del navegador directamente a la instancia abierta del navegador. Por ejemplo, en el modo interactivo puede emitir un comando como este:

```
> element(by.css('#username')).getText()
> NoSuchElementError: No element found using locator: by.username("#username")
```

La salida de aviso del comando anterior aparece directamente allí, lo que le permite saber la corrección de su comando.

Nota: Si ha abierto las Herramientas de desarrollo de Chrome, debe cerrarlas antes de continuar con la prueba porque ChromeDriver no puede funcionar cuando las Herramientas de desarrollo están abiertas.

3. Salga del modo de depuración utilizando `CTRL+C` , puede salir del modo de depuración utilizando el comando clásico CTRL + C.

```
it('should pause when we use pause method', function () {
   browser.get('/index.html');

   var username = element(by.model('username'));
   username.sendKeys('username');
   browser.pause();
```

```
    var password = element(by.model('password'));
    password.sendKeys('password');
    browser.pause();
});
```

4. Presione d para continuar con la siguiente instrucción del depurador

Utilizando browser.debugger ()

Puede usar browser.debugger () para detener la ejecución. Puede insertarlo en cualquier lugar de su código y detendrá la ejecución después de esa línea hasta que no ordene continuar.

Nota: para ejecutar las pruebas en modo depurador, debe emitir un comando como este:

```
    `protractor debug <configuration.file.js>`
```

Ingrese `c` para iniciar la ejecución y continuar después de que el punto de interrupción o entrar `next` command.The pasos al lado de mando a la siguiente línea en el flujo de control.

El depurador utilizado en Protractor utiliza el depurador de nodos y detiene la ejecución de forma asíncrona. Por ejemplo, en el siguiente código, se llamará al `browser.debugger()` cuando se haya ejecutado `username.sendKeys('username') .`

Nota: ya que estas son tareas asíncronas, tendría que aumentar el tiempo de espera predeterminado de sus especificaciones, de lo contrario, se lanzaría la excepción de tiempo de espera predeterminado.

```
it('should pause when we use pause method', function () {
browser.get('/index.html');

var username = element(by.model('username'));
username.sendKeys('username');
browser.debugger();

var password = element(by.model('password'));
password.sendKeys('password');
});
```

Uno puede entrar en el modo de `repl` ingresando el comando

```
debug > repl
> element(by.model('abc')).sendKeys('xyz');
```

Esto ejecutará el comando sendKeys como la siguiente tarea, luego volverá a ingresar al depurador.

Uno puede cambiar el `Port no.` quieren depurar sus scripts simplemente pasando el puerto al método del depurador-

```
browser.debugger(4545); //will start the debugger in port 4545
```

El método `debugger()` inyecta un lado del cliente desde Protractor al navegador y puede ejecutar algunos comandos en la consola del navegador para recuperar los elementos. Uno de los ejemplos para usar el script del lado del cliente es:

```
window.clientSideScripts.findInputs('username');
```

Lea Depurador del transportador en línea:
https://riptutorial.com/es/protractor/topic/3910/depurador-del-transportador

Capítulo 4: Elementos de localización

Introducción

Para poder interactuar con una página, debe decirle a Protractor exactamente qué elemento buscar. La base utilizada para seleccionar los elementos son los localizadores. El transportador, además de incluir los selectores genéricos de Selenium, también tiene localizadores angulares específicos que son más robustos y persistentes a los cambios. Sin embargo, a veces, incluso en una aplicación Angular, se deben usar localizadores regulares.

Parámetros

Parámetro	Detalle
selector	Una cadena que especifica el valor del selector (depende del localizador)

Examples

Localizadores específicos del transportador (para aplicaciones basadas en Angular)

Estos localizadores deben usarse como una prioridad cuando sea posible, porque son más persistentes a los cambios en una aplicación que los localizadores basados en css o xpath, que pueden romperse fácilmente.

Localizador de encuadernación

Sintaxis

```
by.binding('bind value')
```

Ejemplo

Ver

```
<span>{{user.password}}</span>
<span ng-bind="user.email"></span>
```

Locador

```
by.binding('user.password')
by.binding('user.email')
```

También soporta coincidencias parciales

```
by.binding('email')
```

Localizador de enlace exacto

Similar a `binding` , excepto que no se permiten coincidencias parciales

Sintaxis

```
by.exactBinding('exact bind value')
```

Ejemplo

Ver

```
<span>{{user.password}}</span>
```

Locador

```
by.exactBinding('user.password')
by.exactBinding('password') // Will not work
```

Localizador de modelos

Selecciona un elemento con una directiva de modelo angular.

Sintaxis

```
by.model('model value')
```

Ejemplo

Ver

```
<input ng-model="user.username">
```

Locador

```
by.model('user.username')
```

Botón de localizador de texto

Selecciona un botón en función de su texto. Se debe usar solo si no se espera que el texto del botón cambie con frecuencia.

Sintaxis

```
by.buttonText('button text')
```

Ejemplo

Ver

```
<button>Sign In</button>
```

Locador

```
by.buttonText('Sign In')
```

Localizador de texto de botón parcial

Similar a `buttonText` , pero permite coincidencias parciales. Se debe usar solo si no se espera que el texto del botón cambie con frecuencia.

Sintaxis

```
by.partialButtonText('partial button text')
```

Ejemplo

Ver

```
<button>Register an account</button>
```

Locador

```
by.partialButtonText('Register')
```

Localizador de repetidores

Selecciona un elemento con una directiva de repetidor angular.

Sintaxis

```
by.repeater('repeater value')
```

Ejemplo

Ver

```
<tbody ng-repeat="review in reviews">
    <tr>Movie was good</tr>
    <tr>Movie was ok</tr>
    <tr>Movie was bad</tr>
</tbody>
```

Locador

```
by.repeater('review in reviews')
```

También soporta coincidencias parciales

```
by.repeater('reviews')
```

Localizador de repetidor exacto

Similar al `repeater` , pero no permite coincidencias parciales.

Sintaxis

```
by.exactRepeater('exact repeater value')
```

Ejemplo

Ver

```
<tbody ng-repeat="review in reviews">
    <tr>Movie was good</tr>
    <tr>Movie was ok</tr>
    <tr>Movie was bad</tr>
</tbody>
```

Locador

```
by.exactRepeater('review in reviews')
by.exactRepeater('reviews') // Won't work
```

CSS y localizador de texto

Un localizador de CSS extendido donde también puede especificar el contenido de texto del elemento.

Sintaxis

```
by.cssContainingText('css selector', 'text of css element')
```

Ejemplo

Ver

```
<ul>
    <li class="users">Mike</li>
    <li class="users">Rebecca</li>
</ul>
```

Locador

```
by.cssContainingText('.users', 'Rebecca') // Will return the second li only
```

Localizador de opciones

Selecciona un elemento con una directiva de opciones angulares.

Sintaxis

```
by.options('options value')
```

Ejemplo

Ver

```
<select ng-options="country.name for c in countries">
    <option>Canada</option>
    <option>United States</option>
    <option>Mexico</option>
</select>
```

Locador

```
by.options('country.name for c in countries')
```

Localizador de CSS profundo

Localizador CSS que se extiende hacia la sombra DOM

Sintaxis

```
by.deepCss('css selector')
```

Ejemplo

Ver

```
<div>
    <span id="outerspan">
    <"shadow tree">
        <span id="span1"></span>
        <"shadow tree">
            <span id="span2"></span>
        </>
    </>
</div>
```

Locador

```
by.deepCss('span') // Will select every span element
```

Conceptos básicos del localizador

Los localizadores por sí mismos no devuelven un elemento con el que se pueda interactuar en el Transportador, son simplemente instrucciones que indican al Transportador cómo encontrar el elemento.

Para acceder al elemento en sí, use esta sintaxis:

```
element(locator);
element.all(locator);
```

Nota: no se accede realmente al elemento (s) hasta que se realiza una acción en él, es decir, el Transportador solo irá a recuperar el elemento cuando se llame a una acción como getText () en el elemento.

Si desea seleccionar solo un elemento usando un localizador, use el `element` . Si su localizador apunta a múltiples elementos, el `element` devolverá el primero encontrado. `element` devuelve un `ElementFinder` .

Si desea seleccionar varios elementos utilizando un localizador, `element.all` devolverá todos los elementos encontrados. `element.all` devuelve un `ElementArrayFinder` , y se puede acceder a cada elemento de la matriz utilizando diferentes métodos, por ejemplo, la función de `map` .

```
element.all(locator).map(function(singleElement) {
        return singleElement.getText();
    }
});
```

Encadenadores localizadores

Puede encadenar varios localizadores para seleccionar un elemento en una aplicación compleja. No puede encadenar directamente objetos `locator` , debe encadenar `ElementFinders` :

```
element(by.repeater('movie in movies').element(by.linkText('Watch Frozen on Netflix')
```

No hay límite a la cantidad de cadenas que puedes usar; al final, aún recibirás un solo `ElementFinder` o `ElementArrayFinder` , dependiendo de tus localizadores.

Lea Elementos de localización en línea:
https://riptutorial.com/es/protractor/topic/10825/elementos-de-localizacion

Capítulo 5: Espera explícita con browser.wait ()

Examples

browser.sleep () vs browser.wait ()

Cuando se trata de lidiar con el problema del tiempo, es tentador y fácil poner un `browser.sleep(<timeout_in_milliseconds>)` "rápido" `browser.sleep(<timeout_in_milliseconds>)` y seguir adelante.

El problema es que algún día fallaría. No hay una regla de oro / genérica sobre qué tiempo de espera de suspensión se debe configurar y, por lo tanto, en algún momento debido a la red o al rendimiento u otros problemas, puede tomar más tiempo para que una página se cargue o se convierta en un elemento visible, etc. El tiempo, terminarías esperando más de lo que realmente deberías.

`browser.wait()` por otro lado funciona de manera diferente. Proporciona una función de Condición esperada para que Protractor / WebDriverJS ejecute y espere a que el resultado de la función se evalúe como verdadero. *El transportador ejecutaría continuamente la función y se detendría una vez que el resultado de la función se evalúe como verdadero o se haya alcanzado un tiempo de espera configurable.*

Existen múltiples Condiciones esperadas integradas, pero también puede crear y usar una personalizada (muestra aquí).

Lea Espera explícita con browser.wait () en línea:
https://riptutorial.com/es/protractor/topic/8297/espera-explicita-con-browser-wait---

Capítulo 6: Flujo de control y promesas

Introducción

Protractor / WebDriverJS tiene este mecanismo llamado Control Flow : es una cola interna de promesas, mantiene la ejecución del código organizada.

Examples

Entendiendo el flujo de control

Considere la siguiente prueba:

```
it('should test something', function() {
  browser.get('/dashboard/');

  $("#myid").click();
  expect(element(by.model('username')).getText()).toEqual('Test');

  console.log("HERE");
});
```

En la siguiente prueba, cuando se ejecuta el `console.log()` y ve `HERE` en la consola, no se ha ejecutado ninguno de los comandos del Transportador de las líneas anteriores. Este es un comportamiento totalmente *asíncrono* . Los comandos se representan como promesas y se pusieron en el flujo de control que ejecutaría y resolvería las promesas de forma secuencial, una por una.

Ver más en Promesas y el flujo de control .

Lea Flujo de control y promesas en línea: https://riptutorial.com/es/protractor/topic/8580/flujo-de-control-y-promesas

Capítulo 7: Objetos de página

Introducción

Los objetos de página son un patrón de diseño que produce menos duplicados de código, mantenimiento fácil y más legibilidad.

Examples

Objeto de primera página

```
/* save the file in 'pages/loginPage'
var LoginPage = function(){

};

/*Application object properties*/
LoginPage.prototype = Object.create({}, {
    userName: {
        get: function() {
            return browser.driver.findElement(By.id('userid'));
        }
    },
    userPass: {
        get: function() {
            return browser.driver.findElement(By.id('password'));
        }
    },
    submitBtn: {
        get: function() {
            return browser.driver.findElement(By.id('btnSubmit'));
        }
    }
});

/* Adding functions */
LoginPage.prototype.login = function(strUser, strPass) {
    browser.driver.get(browser.baseUrl);
    this.userName.sendKeys(strUser);
    this.userPass.sendKeys(strPass);
    this.submitBtn.click();
};

module.exports = LoginPage;
```

Vamos a usar nuestro primer archivo de objeto de página en nuestra prueba.

```
var LoginPage = require('../pages/loginPage');
describe('User Login to Application', function() {
    var loginPage = new LoginPage();

    beforeAll(function() {
        loginPage.login(browser.params.userName, browser.params.userPass);
```

```
    });

    it('and see a success message in title', function() {
        expect(browser.getTitle()).toEqual('Success');
    });

});
```

Lea Objetos de página en línea: https://riptutorial.com/es/protractor/topic/9747/objetos-de-pagina

Capítulo 8: Probando aplicaciones no angulares con Protractor

Introducción

El transportador está hecho para probar aplicaciones angulares. Sin embargo, todavía es posible probar aplicaciones no angulares con Protractor si es necesario.

Examples

Cambios necesarios para probar la aplicación no angular con Protractor

Use `browser.driver` lugar de `driver`

Utilice `browser.driver.ignoreSynchronization = true`

Motivo : el transportador espera a que los componentes angulares se carguen completamente en una página web antes de que comience la ejecución. Sin embargo, dado que nuestras páginas no son angulares, el transportador sigue esperando a que se cargue 'angular' hasta que la prueba falla con el tiempo de espera. Por lo tanto, debemos decirle explícitamente al Transportador que no espere a que aparezca "angular"

Lea Probando aplicaciones no angulares con Protractor en línea:
https://riptutorial.com/es/protractor/topic/8830/probando-aplicaciones-no-angulares-con-protractor

Capítulo 9: Selectores de CSS

Sintaxis

- por.css ('css-selector')
- por.id ('id')
- por.model ('modelo')
- by.binding ('binding')

Parámetros

Parámetro	Detalles
selector de css	Un selector css como `'.class-name'` para seleccionar el elemento en la base del nombre de clase
carné de identidad	Id del elemento dom
modelo	Modelo utilizado para el elemento dom
Unión	Nombre del enlace que se utiliza para enlazar a cierto elemento

Observaciones

¿Cómo escribir selectores css?

Los atributos más importantes para escribir selectores css son class y id of dom. Para una instancia si un dom html se ve como el siguiente ejemplo:

```
<form class="form-signin">
    <input type="text" id="email" class="form-control" placeholder="Email">
    <input type="password" id="password" class="form-control" placeholder="Password">
    <button class="btn btn-block" id="signin-button" type="submit">Sign in</button>
</form>
```

Luego, para seleccionar el campo de entrada de correo electrónico, puede escribir el selector css de la siguiente manera:

1. **Usando el nombre de la clase** : el nombre de la clase en el selector css comienza con un carácter especial. (Punto). El selector css para eso será como este `.form-control` .

   ```
   by.css('.form-control')
   ```

Dado que la clase de `form-control` es compartida por ambos elementos de entrada, esto plantea una preocupación de duplicidad en los localizadores. Entonces, en tal situación, si la identificación

está disponible, siempre debería preferir usar la identificación en lugar del nombre de la clase.

2. **Utilizando ID** : El identificador en el selector css comienza con un carácter especial # (hash). Por lo tanto, el selector css que utiliza id para el elemento de entrada de correo electrónico se escribirá como a continuación:

   ```
   by.css('#email')
   ```

3. **Uso de varios nombres de clase** : si el elemento dom tiene varias clases, puede combinarlas con el selector css. Por ejemplo, si el elemento dom es así:

```
<input class="username-class form-control">
// css selector using multiple classes
by.css('.username-class.form-control')
```

4. **Usando el nombre de la etiqueta con otros atributos** : La expresión general para escribir el selector css usando el nombre de la etiqueta y otros atributos es `tagname[attribute-type='attribute-vallue']` . Así que siguiendo la expresión, el localizador css para el botón de inicio de sesión se puede formar de esta manera:

```
by.css("button[type='submit']") //or
by.css("button[id='signin-button']")
```

Examples

Accesos directos del localizador de selector $ y $$ CSS

La API de Protractor permite que los localizadores de elementos CSS utilicen la notación de acceso directo similar a jQuery `$()` .

Localizador normal de elementos CSS :

```
element(by.css('h1.documentation-text[ng-bind="title"]'));
element(by.css('[ng-click="submit"]'));
```

Acceso directo `$()` Localizador de elementos CSS :

```
$('h1.documentation-text[ng-bind="title"]');
$('[ng-click="submit"]');
```

Para encontrar múltiples elementos en un localizador, use la notación de acceso directo `$$()` .

Localizador normal de elementos CSS :

```
element.all(by.css('h1.documentation-text[ng-bind="title"]'));
element.all(by.css('[ng-click="submit"]'));
```

Acceso directo `$$()` Localizador de elementos CSS :

```
$$('h1.documentation-text[ng-bind="title"]');
$$('[ng-click="submit"]');
```

Introducción a los localizadores.

Se utiliza un localizador en Protractor para realizar acciones en elementos de dom HTML. Los localizadores más comunes y mejores utilizados en Protractor son css, id, model y binding. Por ejemplo, los localizadores de uso común son:

```
by.css('css-selector')
by.id('id')
```

Seleccione el elemento por un valor de atributo HTML exacto

Para seleccionar un elemento por un atributo HTML exacto, use el patrón del localizador css [atributo = valor]

```
//selects the first element with href value '/contact'
element(by.css('[href="/contact"]'));

//selects the first element with tag option and value 'foo'
element(by.css('option[value="foo"]'));

//selects all input elements nested under the form tag with name attribute 'email'
element.all(by.css('form input[name="email"]'));
```

Seleccione el elemento por un atributo HTML que contenga un valor especificado

Para seleccionar un elemento por un atributo HTML que contenga un valor específico, use el patrón del localizador css [atributo * = valor]

```
//selects the first element with href value that contains'cont'
element(by.css('[href*="cont"]'));

//selects the first element with tag h1 and class attribute that contains 'fo'
element(by.css('h1[class*="fo"]'));

//selects all li elements with a title attribute that contains 'users'
element.all(by.css('li[title*='users']'));
```

Lea Selectores de CSS en línea: https://riptutorial.com/es/protractor/topic/1524/selectores-de-css

Capítulo 10: Selectores XPath en transportador

Examples

Seleccionando un elemento DOM usando un transportador

Además de los selectores de CSS, modelo y enlace, el transportador también puede localizar elementos utilizando la vista xpath

```
<ul>
<li><a href='http://www.google.com'>Go to google</a></li>
</ul>
```

Código

```
var googleLink= element(by.xpath('//ul/li/a'));
expect(element.getText()).to.eventually.equal('Go to google','The text you mention was not
found');
```

Selección de elementos con atributos específicos.

Los selectores XPath se pueden usar para seleccionar elementos con atributos específicos, como clase, id, título, etc.

Por clase

Ver:

```
<div class="HakunaMatata"> Hakuna Matata </div>
```

Código:

```
var theLionKing= element(by.xpath('//div[@class="HakunaMatata"]'));
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Sin embargo, un elemento puede tener múltiples clases. En tales casos, se puede utilizar la solución 'contiene'

Ver:

```
<div class="Hakuna Matata"> Hakuna Matata </div>
```

Código:

```
var theLionKing= element(by.xpath('//div[conatins(@class,"Hakuna")]'));
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

El fragmento de código anterior devolverá elementos que contienen 'class = "HakunaMatata"' y 'class = "Hakuna Matata"'. Si su texto de búsqueda es parte de una lista separada por espacios, entonces se puede usar la siguiente solución:

```
var theLionKing= element(by.xpath('//div[contains(concat(' ',normalize-space(@class),' '),
"Hakuna")]'));
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Por id

ID sigue siendo el localizador más sencillo y preciso que se puede usar para seleccionar un elemento.

Ver:

```
<div id="HakunaMatata">Hakuna Matata</div>
```

Código:

```
var theLionKing= element(by.xpath('//div[@id="HakunaMatata"]');
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Al igual que con las clases, la función contiene se puede usar para encontrar un elemento que contenga el texto dado.

Otros atributos

Encontrar un elemento con un atributo de **título** dado

Ver

```
<div title="Hakuna Matata">Hakuna Matata</div>
```

Código

```
var theLionKing= element(by.xpath('//div[@title="Hakuna Matata"]'));
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Seleccionando un elemento con un texto específico.

Ver

```
<div class="Run Simba Run">Run Simba</div>
```

Código

```
var runSimba= element(by.xpath('//div[text()="Run Simba"]'));
```

Al igual que con otras búsquedas basadas en texto, la función contiene se puede usar para seleccionar elementos con texto () que contenga la coincidencia requerida.

Ver

```
<div class="Run Simba Run">Run Simba,run</div>
```

Código

```
 var runSimba= element(by.xpath('//div[contains(text(),"Run Simba")]'));
expect(runSimba.getText()).to.eventually.equal('Run Simba, run', "Text not found"); //true
```

Selección de un elemento con un atributo de nombre específico

Ver

```
<input type="text" name="FullName"></input>
```

Código

```
var fullNameInput= element(by.xpath('//input[@name="FullName"]'));
fullNameInput.sendKeys("John Doe");
```

Lea Selectores XPath en transportador en línea:
https://riptutorial.com/es/protractor/topic/7205/selectores-xpath-en-transportador

Creditos

S. No	Capítulos	Contributors
1	Empezando con el transportador	Bhoomi Bhalani, Community, Devmati Wadikar, Manuli Piyalka, olyv, Peter Stegnar, Praveen, Priyanshu Shekhar, SilentLupin, sonhu, Stephen Leppik
2	Archivo de configuración del transportador	Barney
3	Depurador del transportador	Devmati Wadikar, Priyanshu Shekhar, Ram Pasala, Sakshi Singla, Stephen Leppik
4	Elementos de localización	Sébastien Dufour-Beauséjour
5	Espera explícita con browser.wait ()	alecxe
6	Flujo de control y promesas	alecxe
7	Objetos de página	Barney, Suresh Salloju
8	Probando aplicaciones no angulares con Protractor	Sakshi Singla
9	Selectores de CSS	alecxe, Droogans, leon, Priyanshu Shekhar, sonhu
10	Selectores XPath en transportador	Shubhang