eBook Gratuit

APPRENEZ protractor

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#protractor

Table des matières

À propos

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

Chapitre 1: Commencer avec le rapporteur

Remarques

Protractor est un framework de test de bout en bout pour les applications AngularJS.

Protractor est un wrapper (construit sur le dessus) autour de Selenium WebDriver. Il contient donc toutes les fonctionnalités disponibles dans Selenium WebDriver. De plus, Protractor fournit de nouvelles stratégies et fonctions de localisation qui sont très utiles pour automatiser l'application AngularJS. Les exemples incluent des choses comme: waitForAngular, By.binding, By.repeater, By.textarea, By.model, WebElement.all, WebElement.evaluate, etc.

Versions

Version	Données de version
0.0.1	2016-08-01

Examples

Installation et configuration de Protractor (sous Windows)

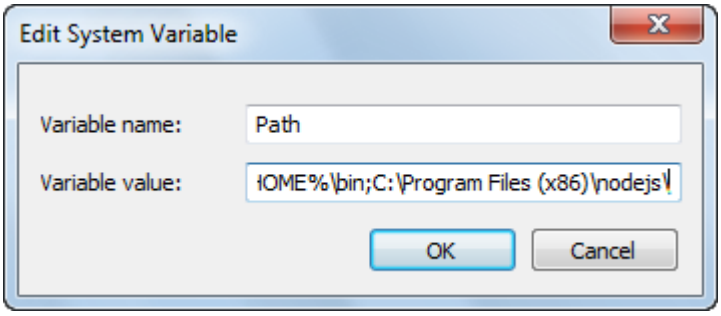
Configuration requise: Protractor requiert l'installation des dépendances suivantes avant l'installation:

- Java JDK 1.7 ou supérieur
- Node.js v4 ou supérieur

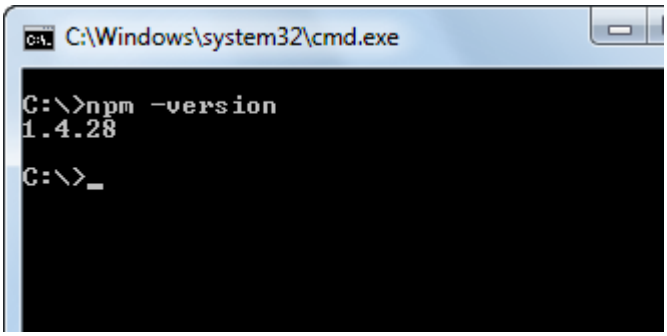
Installation:

Téléchargez et installez Node.js à partir de cette URL: https://nodejs.org/en/

Pour voir si l'installation de Node.js est réussie, vous pouvez aller vérifier les variables d'environnement. Le chemin sous Variables système sera automatiquement mis à jour.



Vous pouvez également vérifier la même chose en tapant la commande `npm -version` dans l'invite de commande qui vous donnera la version installée.



Maintenant, Protractor peut être installé de deux manières: localement ou globalement.

Nous pouvons installer un rapporteur dans un dossier ou un répertoire de projet spécifique. Si nous installons dans un répertoire de projet, chaque fois que nous courons, nous ne devrions courir que depuis cet emplacement.

Pour installer localement dans le répertoire du projet, accédez au dossier du projet et tapez la commande

```
npm install protractor
```

Pour installer Protractor, exécutez globalement la commande:

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

Cela installera deux outils de ligne de commande, `protractor` et `webdriver-manager` .
Exécutez le `protractor --version` pour vous assurer que le rapporteur a bien été installé.

`webdriver-manager` est utilisé pour télécharger les fichiers binaires du pilote du navigateur et démarrer le serveur sélénium.
Téléchargez les fichiers binaires du pilote de navigateur avec:

```
$ webdriver-manager update
```

Démarrer le serveur de sélénium avec:

```
$ webdriver-manager start
```

Pour télécharger le pilote Internet Explorer, exécutez la commande `webdriver-manager update --ie` dans l'invite de commande. Cela va télécharger IEDriverServer.exe dans votre dossier sélénium

Premier test avec Protractor

Protractor n'a besoin que de deux fichiers pour exécuter le premier test, le fichier de spécifications (code de test) et le fichier de configuration. Le fichier de spécification contient le code de test et

l'autre contient les détails de configuration tels que le chemin du fichier de spécification, les détails du navigateur, l'URL de test, les paramètres du framework, etc. , timeout, le framework sera ramassé aux valeurs par défaut.

Le navigateur par défaut pour Protractor est Chrome.

conf.js - Fichier de configuration

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

spec.js - **Fichier** Spec (code de test)

```
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 - Chemin du serveur sur lequel le serveur webdriver est exécuté.

specs - Un élément de tableau qui contient le chemin des fichiers de test. Les chemins multiples peuvent être spécifiés par des valeurs séparées par des virgules.

describe - Syntaxe du framework Jasmine . `describe` syntaxe sta

Ecrivez un test du rapporteur

Ouvrez une nouvelle ligne de commande ou une fenêtre de terminal et créez un dossier propre à tester.

Le rapporteur a besoin de deux fichiers à exécuter, un fichier de spécifications et un fichier de configuration.

Commençons par un test simple qui navigue vers l'exemple de liste de tâches sur le site Web AngularJS et ajoute un nouvel élément todo à la liste.

Copiez ce qui suit dans `spec.js`

describe ('angularjs homepage todo list', function () {it ('devrait ajouter 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);});});
```

Tests de fonctionnement sélectif

Protractor peut exécuter des groupes de tests de manière sélective en utilisant fdescribe () au lieu de describe ().

```
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
    });
});
```

Le rapporteur peut exécuter des tests de manière sélective dans des groupes en utilisant fit () au lieu de celui-ci ().

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

S'il n'y a pas de fit () dans un fdescribe (), alors chaque it () sera exécuté. Cependant, un fit () bloquera les appels () dans le même describe () ou 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
    });
});
```

Même si un fit () est dans un describe () au lieu d'un fdescribe (), il s'exécutera. De même, tout it () dans un fdescribe () qui ne contient pas de fit () sera exécuté.

```
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
    });
});
```

Tests en attente

Le rapporteur permet de définir les tests en attente. Cela signifie que le rapporteur n'exécutera pas le test, mais produira à la place:

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

Ou, si désactivé avec xdescribe ():

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

Combinaisons

- Un xit () dans un xdescribe () affichera la réponse xit ().
- Un xit () dans un fdescribe () sera toujours traité comme étant en attente.
- Un fit () dans un xdescribe () sera toujours exécuté et aucun test en attente ne produira quoi que ce soit.

Rapporteur: test E2E pour applications angulaires d'entreprise

Installation et configuration du rapporteur

Étape 1 : Téléchargez et installez NodeJS à partir d'ici. Assurez-vous d'avoir la dernière version du noeud. Ici, j'utilise le noeud v7.8.0. Le kit de développement Java (JDK) doit être installé pour exécuter le sélénium.

Étape 2 : Ouvrez votre terminal et entrez la commande suivante pour installer le rapporteur globalement.

```
npm install -g protractor
```

Cela installera deux outils tels que le rapporteur et le gestionnaire Webdriver. Vous pouvez vérifier l'installation de votre `protractor -version.` en suivant la commande suivante: `protractor -version.`

Si Protractor est installé avec succès, le système affichera la version installée (c.-à-d. La version 5.1.1). Sinon, vous devrez revérifier l'installation. Étape 3: Mettez à jour le gestionnaire webdriver pour télécharger les fichiers binaires nécessaires.

```
webdriver-manager update
```

Étape 4: La commande suivante lancera un serveur Selenium. Cette étape exécutera le gestionnaire de pilotes Web en arrière-plan et écoutera tous les tests exécutés via le rapporteur.

webdriver-manager start Vous pouvez voir des informations sur l'état du serveur à l' `http://localhost:4444/wd/hub/static/resource/hub.html`.

Ecriture du premier cas de test à l'aide du rapporteur:

Avant de passer à l'écriture du scénario de test, nous devons préparer deux fichiers, à savoir le fichier de configuration et le fichier de spécifications.

Dans le fichier de configuration:

```
//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'
    }
}
```

Compréhension de base des terminologies utilisées dans le fichier de configuration:

baseUrl - Une URL de base pour votre application en cours de test.

seleniumAddress - Pour vous connecter à un serveur Selenium déjà en cours d'exécution.

specs - Emplacement de votre fichier de spécifications

directConnect : true - Pour se connecter directement au navigateur Drivers.

Capacités - Si vous testez sur un seul navigateur, utilisez l'option des fonctionnalités. Si vous testez sur plusieurs navigateurs, utilisez le tableau multiCapabilities.

Vous pouvez trouver plus d'options de configuration à partir d' ici . Ils ont décrit toute la terminologie possible avec sa définition.

Dans le fichier Spec:

```
//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');
      });
    });
```

Compréhension de base des terminologies utilisées dans les fichiers de spécifications:

Par défaut, Protractor utilise le framework jasmine pour son interface de test. La syntaxe 'describe' et 'it' provient du framework Jasmin. Vous pouvez en apprendre plus d'ici. Exécution du premier test:

Avant de lancer le test, assurez-vous que votre gestionnaire webdriver et votre application s'exécutent dans différents onglets de votre terminal.

Maintenant, lancez le test avec:

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

Vous devriez voir que le navigateur chrome s'ouvre avec l'URL de votre application et se ferme. Le résultat du test doit être 1 test, 1 assertion, 0 échec.

Bravo! Vous avez réussi votre premier scénario de test.

Lire Commencer avec le rapporteur en ligne:
https://riptutorial.com/fr/protractor/topic/933/commencer-avec-le-rapporteur

Chapitre 2: Attentes explicites avec browser.wait ()

Examples

browser.sleep () vs browser.wait ()

Quand il s'agit de gérer un problème de synchronisation, il est tentant et facile de mettre un "quick" `browser.sleep(<timeout_in_milliseconds>)` et de continuer.

Le problème est que cela échouerait un jour. Il n'y a pas de règle d'or / générique sur le délai de mise en veille à définir et, par conséquent, en raison de problèmes de réseau, de performances ou autres, le chargement d'une page ou son élément peut prendre plus de temps. le temps, vous finiriez par attendre plus que vous ne devriez réellement.

`browser.wait()` d'autre part fonctionne différemment. Vous fournissez une fonction Condition attendue pour Protractor / WebDriverJS à exécuter et attendez que le résultat de la fonction soit évalué à true. *Protractor exécutera continuellement la fonction et s'arrêtera une fois que le résultat de la fonction aura la valeur true ou qu'un délai d'attente configurable aura été atteint.*

Il existe plusieurs conditions attendues intégrées, mais vous pouvez également en créer et en utiliser une personnalisée (exemple ici).

Lire Attentes explicites avec browser.wait () en ligne:
https://riptutorial.com/fr/protractor/topic/8297/attentes-explicites-avec-browser-wait---

Chapitre 3: Contrôle des flux et des promesses

Introduction

Protractor / WebDriverJS dispose de ce mécanisme appelé Control Flow - il s'agit d'une file d'attente interne de promesses, qui permet d'organiser l'exécution du code.

Examples

Comprendre le flux de contrôle

Considérez le test suivant:

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

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

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

Dans le test suivant, lorsque la `console.log()` est exécutée et que vous voyez `HERE` sur la console, aucune des commandes du rapporteur depuis les lignes précédentes n'a été exécutée. C'est un comportement entièrement *asynchrone* . Les commandes sont représentées sous la forme de promesses et placées sur le flux de contrôle, ce qui permet d'exécuter et de résoudre les promesses l'une après l'autre.

Voir plus à Promises et le flux de contrôle .

Lire Contrôle des flux et des promesses en ligne:
https://riptutorial.com/fr/protractor/topic/8580/controle-des-flux-et-des-promesses

Chapitre 4: Débogueur du rapporteur

Syntaxe

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

Remarques

Cette section explique comment déboguer les tests de rapporteur.

Examples

Utiliser browser.pause ()

La méthode `pause()` est l'une des solutions les plus simples que Protractor vous propose pour déboguer le code. Pour pouvoir l'utiliser, vous devez l'ajouter dans votre code où vous souhaitez interrompre l'exécution. Une fois l'exécution en pause:

1. Vous pouvez utiliser `c` (type C) pour avancer. Soyez prudent lorsque vous l'utilisez, vous devez écrire cette commande sans délai, car votre bibliothèque d'assertion risque de générer une erreur de temporisation si vous avez retardé d'appuyer sur c.

2. Tapez `repl` pour entrer en mode interactif. Le mode interactif est utilisé pour envoyer les commandes du navigateur directement à l'instance ouverte du navigateur. Par exemple, en mode interactif, vous pouvez émettre une commande comme celle-ci:

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

Notez que la sortie de la commande ci-dessus apparaît directement là-bas, ce qui vous permet de savoir si votre commande est correcte.

Remarque: Si vous avez ouvert les outils de développement Chrome, vous devez les fermer avant de poursuivre le test, car ChromeDriver ne peut pas fonctionner lorsque les outils de développement sont ouverts.

3. Quittez le mode de débogage en utilisant `CTRL+C` , vous pouvez sortir du mode de débogage en utilisant la commande classique 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. Appuyez sur d pour continuer à l'instruction de débogage suivante

Utiliser browser.debugger ()

Vous pouvez utiliser browser.debugger () pour arrêter l'exécution. Vous pouvez l'insérer à n'importe quel endroit dans votre code et cela arrêtera l'exécution après cette ligne jusqu'à ce que vous ne demandiez plus de continuer.

Remarque: Pour exécuter les tests en mode débogueur, vous devez exécuter la commande suivante:

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

Entrez `c` pour commencer l' exécution et continuer après le point d' arrêt ou entrez `next` commande.Système étapes de commande à côté de la ligne suivante dans le flux de contrôle.

Le débogueur utilisé dans Protractor utilise le débogueur de noeud et met en pause l'exécution de manière asynchrone. Par exemple, dans le code ci-dessous, le `browser.debugger()` sera appelé lorsque `username.sendKeys('username')` a été exécuté.

Remarque: Étant donné qu'il s'agit de tâches asynchrones, vous devez augmenter le délai d'attente par défaut de vos spécifications, sinon une exception de délai d'attente par défaut serait émise!

```
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');
});
```

On peut entrer dans le mode `repl` en entrant la commande-

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

Cela exécutera la commande sendKeys en tant que tâche suivante, puis entrera de nouveau dans le débogueur.

On peut changer le `Port no.` ils veulent déboguer leurs scripts simplement en passant le port à la méthode du débogueur-

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

La méthode `debugger()` injecte un côté client de Protractor au navigateur et vous pouvez exécuter quelques commandes dans la console du navigateur pour récupérer les éléments. L'un des exemples d'utilisation du script côté client est:

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

Lire Débogueur du rapporteur en ligne: https://riptutorial.com/fr/protractor/topic/3910/debogueur-du-rapporteur

Chapitre 5: Éléments de localisation

Introduction

Pour pouvoir interagir avec une page, vous devez indiquer à Protractor quel élément rechercher. Les bases utilisées pour sélectionner les éléments sont les localisateurs. Protractor, tout comme les sélecteurs génériques Selenium, possède également des localisateurs spécifiques à Angular, plus robustes et persistants aux modifications. Cependant, parfois, même dans une application angulaire, des localisateurs réguliers doivent être utilisés.

Paramètres

Paramètre	Détail
sélecteur	Une chaîne qui spécifie la valeur du sélecteur (dépend du localisateur)

Examples

Localisateurs spécifiques au rapporteur (pour les applications à base angulaire)

Ces localisateurs doivent être utilisés en priorité lorsque cela est possible, car ils sont plus persistants dans les modifications d'une application que dans les localisateurs basés sur css ou xpath, ce qui peut facilement se casser.

Localisateur de reliure

Syntaxe

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

Exemple

Vue

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

Localisateur

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

Prend également en charge les correspondances partielles

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

Localisateur de reliure exacte

Similaire à la `binding` , sauf que les correspondances partielles ne sont pas autorisées.

Syntaxe

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

Exemple

Vue

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

Localisateur

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

Localisateur de modèle

Sélectionne un élément avec une directive de modèle angulaire

Syntaxe

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

Exemple

Vue

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

Localisateur

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

Localisateur de texte de bouton

Sélectionne un bouton basé sur son texte. Ne doit être utilisé que si le texte du bouton ne change pas souvent.

Syntaxe

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

Exemple

Vue

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

Localisateur

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

Localisateur de texte de bouton partiel

Similaire à `buttonText` , mais permet des correspondances partielles. Ne doit être utilisé que si le texte du bouton ne change pas souvent.

Syntaxe

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

Exemple

Vue

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

Localisateur

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

Localisateur de répéteur

Sélectionne un élément avec une directive de répéteur angulaire

Syntaxe

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

Exemple

Vue

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

Localisateur

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

Prend également en charge les correspondances partielles

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

Localisateur de répéteur exact

Similaire au `repeater` , mais ne permet pas les correspondances partielles

Syntaxe

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

Exemple

Vue

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

Localisateur

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

CSS et localisateur de texte

Un localisateur CSS étendu où vous pouvez également spécifier le contenu textuel de l'élément.

Syntaxe

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

Exemple

Vue

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

Localisateur

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

Localisateur d'options

Sélectionne un élément avec une directive d'options angulaires

Syntaxe

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

Exemple

Vue

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

Localisateur

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

Deep CSS locator

Localisateur CSS qui s'étend dans le DOM de l' ombre

Syntaxe

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

Exemple

Vue

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

Localisateur

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

Notions de base sur les localisateurs

Les localisateurs eux-mêmes ne renvoient pas un élément avec lequel il est possible d'interagir dans Protractor, ce sont simplement des instructions indiquant comment Protractor trouve l'élément.

Pour accéder à l'élément lui-même, utilisez cette syntaxe:

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

Remarque: le ou les éléments ne sont pas réellement accessibles tant qu'une action n'a pas été exécutée - c'est-à-dire que Protractor ne récupèrera l'élément que si une action telle que getText () est appelée sur l'élément.

Si vous souhaitez sélectionner un seul élément à l'aide d'un localisateur, utilisez l' `element` . Si votre localisateur pointe vers plusieurs éléments, l' `element` renverra le premier trouvé. `element` renvoie un `ElementFinder` .

Si vous souhaitez sélectionner plusieurs éléments à l'aide d'un localisateur, `element.all` renverra tous les éléments trouvés. `element.all` retourne un `ElementArrayFinder` , et chaque élément du tableau est accessible en utilisant différentes méthodes - par exemple, la `map` fonction.

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

Localisateurs de chaînage

Vous pouvez enchaîner plusieurs localisateurs pour sélectionner un élément dans une application complexe. Vous ne pouvez pas enchaîner directement les objets de `locator` , vous devez enchaîner les éléments `ElementFinders` :

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

Il n'y a pas de limite au nombre de chaînes que vous pouvez utiliser. à la fin, vous serez toujours recevoir un `ElementFinder` ou et `ElementArrayFinder` , selon vos localisateurs.

Lire Éléments de localisation en ligne: https://riptutorial.com/fr/protractor/topic/10825/elements-de-localisation

Chapitre 6: Fichier de configuration du rapporteur

Introduction

Le fichier de configuration contient des informations que Protractor utilise pour exécuter votre script de test. Ici, je vais essayer de donner quelques variantes.

Examples

Fichier de configuration 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;
```

Fichier de configuration avec capacités - 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;
```

fichier de configuration shardTestFiles - Chrome

Cette configuration vous permet d'exécuter votre total de fichiers de spécifications dans deux instances de navigateur en parallèle. Cela permet de réduire le temps d'exécution du test global.

Modifiez les maxInstances en fonction de vos besoins.

Remarque : *Assurez-vous que vos tests sont indépendants.*

```
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;
```

fichier de configuration multi-capacités émuler - 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;
```

Lire Fichier de configuration du rapporteur en ligne:
https://riptutorial.com/fr/protractor/topic/9745/fichier-de-configuration-du-rapporteur

Chapitre 7: Objets de la page

Introduction

Les objets de page sont un modèle de conception qui se traduit par moins de doublons de code, une maintenance facile et une meilleure lisibilité.

Examples

Objet de la première page

```
/* 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;
```

Utilisons notre fichier objet de première page dans notre test.

```
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');
    });

});
```

Lire Objets de la page en ligne: https://riptutorial.com/fr/protractor/topic/9747/objets-de-la-page

Chapitre 8: Sélecteurs CSS

Syntaxe

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

Paramètres

Paramètre	Détails
css-selector	Un sélecteur CSS tel que `'.class-name'` pour sélectionner l'élément sur la base du nom de la classe
id	Identifiant de l'élément dom
modèle	Modèle utilisé pour l'élément dom
contraignant	Nom de la liaison utilisée pour lier certains éléments

Remarques

Comment écrire des sélecteurs CSS?

Les attributs les plus importants pour écrire des sélecteurs CSS sont class et id of dom. Pour une instance si un dom HTML ressemble à l'exemple ci-dessous:

```
<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>
```

Ensuite, pour sélectionner le champ de saisie, vous pouvez écrire le sélecteur CSS de la manière suivante:

1. **Utilisation du nom de la classe** : Le nom de la classe dans le sélecteur CSS commence par un caractère spécial (point). Le sélecteur CSS pour cela sera comme ceci `.form-control` .

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

Comme la classe de `form-control` est partagée par les deux éléments d'entrée, cela soulève un problème de duplicité dans les localisateurs. Donc, dans ce cas, si id est disponible, vous devriez toujours préférer utiliser id au lieu du nom de la classe.

2. **Utilisation de l'ID** : L'identifiant dans le sélecteur CSS commence par le caractère spécial # (hachage). Ainsi, le sélecteur CSS utilisant l'ID pour l'élément de saisie de courrier électronique sera écrit comme ci-dessous:

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

3. **Utilisation de plusieurs noms de classe** : Si l'élément dom a plusieurs classes, vous pouvez combiner plusieurs classes en tant que sélecteur CSS. Par exemple, si l'élément dom est comme ceci:

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

4. **Utilisation du nom de balise avec d'autres attributs** : L'expression générale pour écrire le sélecteur CSS à l'aide du nom de balise et d'autres attributs est `tagname[attribute-type='attribute-vallue']` . Donc, suivant l'expression le bouton de localisation css pour le bouton de connexion peut être formé comme ceci:

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

Examples

Raccourcis du localisateur de sélecteur $ et $$ CSS

L'API Protractor permet aux localisateurs d'éléments CSS d'utiliser la notation de raccourci de type jQuery `$()` .

Normal Element Locator CSS :

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

Raccourci `$()` Localisateur d'éléments CSS :

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

Pour trouver plusieurs éléments sous un localisateur, utilisez la notation de raccourci `$$()` .

Normal Element Locator CSS :

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

Shortcut `$$()` CSS Element Locator :

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

Introduction aux localisateurs

Un localisateur dans Protractor est utilisé pour effectuer des actions sur les éléments HTML dom. Les plus populaires et les meilleurs localisateurs utilisés dans Protractor sont css, id, model et binding. Par exemple, les localisateurs couramment utilisés sont:

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

Sélectionner un élément par une valeur d'attribut HTML exacte

Pour sélectionner un élément par un attribut HTML exact, utilisez le modèle de localisation css [attribut = valeur]

```
//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"]'));
```

Sélectionner un élément par un attribut HTML contenant une valeur spécifiée

Pour sélectionner un élément par un attribut HTML contenant une valeur spécifiée, utilisez le modèle de localisation css [attribut * = valeur]

```
//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']'));
```

Lire Sélecteurs CSS en ligne: https://riptutorial.com/fr/protractor/topic/1524/selecteurs-css

Chapitre 9: Sélecteurs XPath dans Protractor

Examples

Sélection d'un élément DOM à l'aide du rapporteur

En dehors des sélecteurs CSS, model et binding, le rapporteur peut également localiser des éléments à l'aide de xpath View

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

Code

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

Sélection d'éléments avec des attributs spécifiques

Les sélecteurs XPath peuvent être utilisés pour sélectionner des éléments avec des attributs spécifiques, tels que la classe, l'ID, le titre, etc.

Par classe

Vue:

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

Code:

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

Cependant, un élément peut avoir plusieurs classes. Dans de tels cas, la solution de contournement «contient» peut être utilisée

Vue:

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

Code:

```
var theLionKing= element(by.xpath('//div[conatins(@class,"Hakuna")]'));
```

```
expect(theLionKing.getText()).to.eventually.equal('Hakuna Matata', "Text not found");
```

Le morceau de code ci-dessus renverra des éléments contenant à la fois 'class = "HakunaMatata"' et 'class = "Hakuna Matata"'. Si votre texte de recherche fait partie d'une liste séparée par des espaces, la solution suivante peut être utilisée:

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

Par id

L'ID reste le localisateur le plus simple et le plus précis permettant de sélectionner un élément.

Vue:

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

Code:

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

Comme pour les classes, la fonction contains peut être utilisée pour trouver un élément contenant le texte donné.

Autres attributs

Recherche d'un élément avec un attribut de **titre** donné

Vue

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

Code

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

Sélection d'un élément avec un texte spécifique

Vue

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

Code

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

Comme avec d'autres recherches basées sur du texte, la fonction contains peut être utilisée pour sélectionner des éléments avec text () contenant la correspondance requise.

Vue

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

Code

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

Sélection d'un élément avec un attribut de nom spécifique

Vue

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

Code

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

Lire Sélecteurs XPath dans Protractor en ligne:
https://riptutorial.com/fr/protractor/topic/7205/selecteurs-xpath-dans-protractor

Chapitre 10: Tester des applications non angulaires avec Protractor

Introduction

Le rapporteur est conçu pour tester les applications angulaires. Cependant, il est toujours possible de tester des applications non angulaires avec Protractor si nécessaire.

Examples

Modifications nécessaires pour tester une application non angulaire avec Protractor

Utilisez `browser.driver` au lieu du `driver`

Utilisez `browser.driver.ignoreSynchronization = true`

Raison : Protractor attend que les composants angulaires se chargent complètement sur une page Web avant de commencer toute exécution. Cependant, étant donné que nos pages ne sont pas angulaires, Protractor attend le chargement angulaire jusqu'à ce que le test échoue avec le délai d'attente. Donc, nous devons dire explicitement au rapporteur de ne pas attendre "angulaire"

Lire Tester des applications non angulaires avec Protractor en ligne: https://riptutorial.com/fr/protractor/topic/8830/tester-des-applications-non-angulaires-avec-protractor

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec le rapporteur	Bhoomi Bhalani, Community, Devmati Wadikar, Manuli Piyalka, olyv, Peter Stegnar, Praveen, Priyanshu Shekhar, SilentLupin, sonhu, Stephen Leppik
2	Attentes explicites avec browser.wait ()	alecxe
3	Contrôle des flux et des promesses	alecxe
4	Débogueur du rapporteur	Devmati Wadikar, Priyanshu Shekhar, Ram Pasala, Sakshi Singla, Stephen Leppik
5	Éléments de localisation	Sébastien Dufour-Beauséjour
6	Fichier de configuration du rapporteur	Barney
7	Objets de la page	Barney, Suresh Salloju
8	Sélecteurs CSS	alecxe, Droogans, leon, Priyanshu Shekhar, sonhu
9	Sélecteurs XPath dans Protractor	Shubhang
10	Tester des applications non angulaires avec Protractor	Sakshi Singla