Kostenloses eBook

LERNEN protractor

Free unaffiliated eBook created from
Stack Overflow contributors.

#protractor

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit dem Winkelmesser

Bemerkungen

Der Winkelmesser ist ein **durchgängiges** Test-Framework für AngularJS-Anwendungen.

Bei dem Winkelmesser handelt es sich um einen Wrapper (auf der Oberseite aufgebaut) um Selenium WebDriver. Daher enthält er alle Funktionen, die im Selenium WebDriver verfügbar sind. Darüber hinaus bietet Protractor einige neue Locator-Strategien und -Funktionen, die für die Automatisierung der AngularJS-Anwendung sehr hilfreich sind. Beispiele sind Dinge wie: waitForAngular, By.binding, By.repeater, By.textarea, By.model, WebElement.all, WebElement.evaluate usw.

Versionen

Ausführung	Daten freigeben
0.0.1	2016-08-01

Examples

Installieren und Einrichten von Winkelmesser (unter Windows)

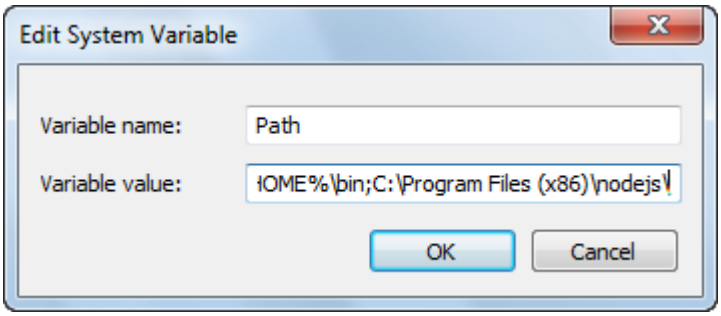
Voraussetzungen: Für den Winkelmesser müssen vor der Installation die folgenden Abhängigkeiten installiert werden:

- Java JDK 1.7 oder höher
- Node.js v4 oder höher

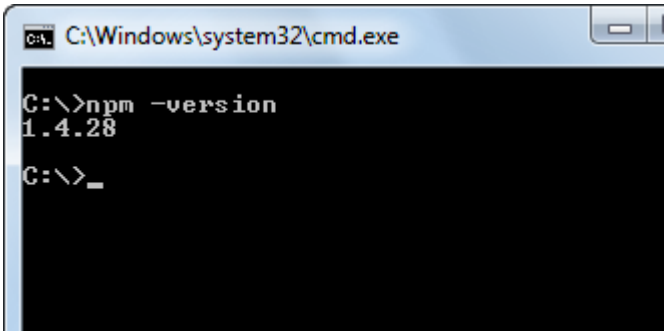
Installation:

Laden Sie Node.js von dieser URL herunter und installieren Sie sie: https://nodejs.org/de/

Um zu überprüfen, ob die Installation von Node.js erfolgreich ist, können Sie die Umgebungsvariablen überprüfen. Der 'Pfad' unter Systemvariablen wird automatisch aktualisiert.



Sie können dies auch überprüfen, indem `npm -version` in der Eingabeaufforderung den Befehl `npm -version` eingeben, der die installierte Version `npm -version` .



Protractor kann jetzt auf zwei Arten installiert werden: lokal oder global.

Wir können den Winkelmesser in einem bestimmten Ordner oder in einem Projektverzeichnis installieren. Wenn wir in einem Projektverzeichnis installieren, sollten wir bei jeder Ausführung nur von diesem Speicherort aus ausführen.

Um lokal im Projektverzeichnis zu installieren, navigieren Sie zum Projektordner und geben Sie den Befehl ein

```
npm install protractor
```

Um Protractor global zu installieren, führen Sie den Befehl aus:

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

Dadurch werden zwei Befehlszeilen-Tools installiert: `protractor` und `webdriver-manager` .
Führen Sie die `protractor --version` um sicherzustellen, dass der Winkelmesser erfolgreich installiert wurde.

`webdriver-manager` wird verwendet, um die Binärdateien des Browsertreibers herunterzuladen und den `webdriver-manager` starten.
Laden Sie die Browser-Treiber-Binärdateien herunter mit:

```
$ webdriver-manager update
```

Starten Sie den Selenserver mit:

```
$ webdriver-manager start
```

Führen Sie zum Herunterladen des Internet Explorer-Treibers den Befehl `webdriver-manager update --ie` in der Eingabeaufforderung aus. Dadurch wird IEDriverServer.exe in Ihrem Selenium-Ordner heruntergeladen

Erster Test mit Winkelmesser

Der Winkelmesser benötigt nur zwei Dateien, um die erste Test-, Spezifikations- (Testcode-) und Konfigurationsdatei auszuführen. Die Spezifikationsdatei enthält Testcode und die andere enthält Konfigurationsdetails wie Spezifikationsdateipfad, Browserdetails, Test-URL, Rahmenparameter usw. Zum Schreiben des ersten Tests werden nur die Adresse des Selen-Servers und der Spezifikationsdateipfad angegeben Bei einem Timeout wird das Framework auf die Standardwerte gesetzt.

Der Standardbrowser für Protractor ist Chrome.

conf.js - Konfigurationsdatei

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

spec.js - **Spezifikationsdatei** (**Testcode**)

```
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 - Pfad zum Server , auf den WebDriver Server ausgeführt wird .

specs - Ein Arrayelement, das den Pfad der Testdateien enthält. Die mehreren Pfade können durch kommagetrennte Werte angegeben werden.

beschreiben - Syntax aus Jasmin- Framework. `describe` Syntax sta

Schreiben Sie einen Winkelmesser-Test

Öffnen Sie eine neue Befehlszeile oder ein neues Terminalfenster und erstellen Sie einen sauberen Ordner zum Testen.

Der Winkelmesser benötigt zwei Dateien, eine Spezifikationsdatei und eine Konfigurationsdatei.

Beginnen wir mit einem einfachen Test, der zu dem ToDo-Listenbeispiel auf der AngularJS-

Website navigiert und der Liste ein neues ToDo-Element hinzufügt.

Kopieren Sie Folgendes in `spec.js`

beschreiben ('anglejs homepage todo list') function () {it ('sollte ein todo hinzufügen') 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);});});
```

Selektive laufende Tests

Der Winkelmesser kann selektiv Testgruppen mit fdescribe () anstelle von compare () ausführen.

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

Der Winkelmesser kann innerhalb von Gruppen gezielt Tests ausführen, indem er anstelle von it () fit () verwendet.

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

Wenn es in einem fdescribe () kein fit () gibt, wird jedes it () ausgeführt. Ein fit () blockiert jedoch it () -Aufrufe innerhalb der gleichen configure () oder 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
    });
});
```

Auch wenn ein fit () in einem description () anstelle eines fdescribe () enthalten ist, wird es ausgeführt. Außerdem wird jedes it () in einem fdescribe () ausgeführt, das kein fit () enthält.

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

Ausstehende Tests

Mit dem Winkelmesser können Tests als anstehend festgelegt werden. Das bedeutet, dass der Winkelmesser den Test nicht ausführt, sondern stattdessen Folgendes ausgibt:

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

Oder, wenn mit xdescribe () deaktiviert:

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

Kombinationen

- Ein xit () in einem xdescribe () gibt die Antwort von xit () aus.
- Ein xit () innerhalb eines fdescribe () wird immer noch als ausstehend behandelt.
- Ein fit () innerhalb eines xdescribe () wird weiterhin ausgeführt, und ausstehende Tests geben nichts aus.

Winkelmesser: E2E-Test für Enterprise-Angular-Anwendungen

Winkelmesser Installation und Setup

Schritt 1 : Laden Sie NodeJS hier herunter und installieren Sie es. Stellen Sie sicher, dass Sie

über die neueste Version des Knotens verfügen. Hier verwende ich Knoten v7.8.0. Sie müssen das Java Development Kit (JDK) installiert haben, um Selen ausführen zu können.

Schritt 2 : Öffnen Sie Ihr Terminal und geben Sie den folgenden Befehl ein, um den Winkelmesser global zu installieren.

```
npm install -g protractor
```

Dadurch werden zwei Tools installiert, z. B. der Winkelmesser- und der Web-Treiber-Manager. Sie können Ihre Winkelmesserinstallation mit folgendem Befehl überprüfen: `protractor –version.` Wenn Protractor erfolgreich installiert wurde, zeigt das System die installierte Version an (z. B. Version 5.1.1). Andernfalls müssen Sie die Installation erneut überprüfen. Schritt 3: Aktualisieren Sie den Webtreiber-Manager, um die erforderlichen Binärdateien herunterzuladen.

```
webdriver-manager update
```

Schritt 4: Der folgende Befehl startet einen Selenium Server. Bei diesem Schritt wird der Webtreiber-Manager im Hintergrund ausgeführt und alle Tests werden abgehört, die über den Winkelmesser ausgeführt werden.

webdriver-manager start Informationen zum Status des Servers finden Sie unter `http://localhost:4444/wd/hub/static/resource/hub.html.`

Schreiben des ersten Testfalls mit dem Winkelmesser:

Bevor wir mit dem Schreiben des Testfalls beginnen, müssen wir zwei Dateien vorbereiten, die Konfigurationsdatei und Spezifikationsdatei sind.

In der Konfigurationsdatei:

```
//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'
    }
}
```

Grundlegendes zu den in der Konfigurationsdatei verwendeten Terminologien:

baseUrl - Eine Basis-URL für Ihre zu **testende** Anwendung.

SeleniumAddress - Um eine Verbindung zu einem Selenium Server herzustellen, der bereits läuft.

specs - Ort Ihrer Spezifikationsdatei

directConnect : true - Zum direkten Verbinden mit den Browsertreibern.

Fähigkeiten - Wenn Sie mit einem einzelnen Browser testen, verwenden Sie die Option "Funktionen". Wenn Sie mit mehreren Browsern testen, verwenden Sie das multiCapabilities-Array.

Weitere Konfigurationsmöglichkeiten finden Sie hier . Sie haben alle möglichen Begriffe mit ihrer Definition beschrieben.

In Spec-Datei:

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

Grundlegendes zu den in der Spezifikationsdatei verwendeten Terminologien:

Standardmäßig verwendet Protractor das Jasmin-Framework für seine Testschnittstelle. Die Beschreibungs- und Es-Syntax stammt vom Jasmin-Framework. Hier erfahren Sie mehr. Erster Testfall läuft:

Bevor Sie den Testfall ausführen, vergewissern Sie sich, dass Ihr Webdriver-Manager und Ihre Anwendung auf verschiedenen Registerkarten Ihres Terminals laufen.

Führen Sie nun den Test aus mit:

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

Sie sollten sehen, dass sich der Chrome-Browser mit Ihrer Anwendungs-URL öffnet und sich selbst schließt. Die Testausgabe sollte 1 Test, 1 Assertion, 0 Fehler sein.

Bravo! Sie führen Ihren ersten Testfall erfolgreich aus.

Erste Schritte mit dem Winkelmesser online lesen:
https://riptutorial.com/de/protractor/topic/933/erste-schritte-mit-dem-winkelmesser

Kapitel 2: CSS-Selektoren

Syntax

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

Parameter

Parameter	Einzelheiten
CSS-Selektor	Ein CSS-Selektor wie `'.class-name'` zur Auswahl des Elements auf der Basis des Klassennamens
Ich würde	ID des dom-Elements
Modell-	Modell für dom-Element
Bindung	Name der Bindung, die zur Bindung an ein bestimmtes Element verwendet wird

Bemerkungen

Wie schreibe ich CSS-Selektoren?

Die wichtigsten Attribute zum Schreiben von CSS-Selektoren sind Klasse und ID von Dom. Für eine Instanz, wenn ein HTML-Dom wie folgt aussieht:

```
<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>
```

Um das E-Mail-Eingabefeld auszuwählen, können Sie den CSS-Selector folgendermaßen schreiben:

1. **Verwenden des Klassennamens** : Der Klassenname im CSS-Selektor beginnt mit einem Sonderzeichen (Punkt). Die `.form-control` für diese `.form-control` ist wie diese `.form-control` .

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

Da die `form-control` von beiden Eingabeelementen gemeinsam genutzt wird, besteht ein Problem hinsichtlich der Duplizität der Locators. Wenn also id vorhanden ist, sollten Sie immer id anstelle

des Klassennamens verwenden.

2. **ID verwenden** : Die ID im CSS-Selektor beginnt mit dem Sonderzeichen # (Hash). Der css-Selektor, der die ID für E-Mail-Eingabeelemente verwendet, wird wie folgt geschrieben:

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

3. **Mehrere Klassennamen verwenden** : Wenn das dom-Element mehrere Klassen hat, können Sie Klassen als CSS-Selektor verwenden. Wenn zum Beispiel das dom-Element so ist:

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

4. **Tag-Namen mit anderen Attributen verwenden** : Der allgemeine Ausdruck zum Schreiben von CSS-Selektoren mit Tag-Namen und anderen Attributen lautet `tagname[attribute-type='attribute-vallue']` . Nach dem Ausdruck kann der css-Locator für die Anmeldeschaltfläche folgendermaßen geformt werden:

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

Examples

$ und $$ CSS-Auswahl-Locator-Shortcuts

Die Protractor-API ermöglicht es CSS-Element-Locators, die jQuery-artige Abkürzungsnotation `$()` .

Normaler CSS-Element-Locator :

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

Shortcut `$()` CSS Element Locator :

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

Um mehrere Elemente unter einem Locator zu finden, verwenden Sie die Abkürzungsnotation `$$()` .

Normaler CSS-Element-Locator :

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

Abkürzung `$$()` CSS Element Locator :

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

Einführung in Locators

Ein Locator in Protractor wird zum Ausführen von Aktionen für HTML-Dom-Elemente verwendet. Die gebräuchlichsten und besten Locators, die in Protractor verwendet werden, sind CSS, ID, Modell und Bindung. Häufig verwendete Locators sind beispielsweise:

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

Wählen Sie das Element anhand eines genauen HTML-Attributwerts aus

Um ein Element anhand eines genauen HTML-Attributs auszuwählen, verwenden Sie das CSS-Locator-Muster [attribute = value]

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

Wählen Sie das Element anhand eines HTML-Attributs aus, das einen angegebenen Wert enthält

Um ein Element anhand eines HTML-Attributs auszuwählen, das einen angegebenen Wert enthält, verwenden Sie das CSS-Locator-Muster [Attribut * = Wert].

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

CSS-Selektoren online lesen: https://riptutorial.com/de/protractor/topic/1524/css-selektoren

Kapitel 3: Elemente suchen

Einführung

Um mit einer Seite interagieren zu können, müssen Sie Protractor genau sagen, nach welchem Element gesucht werden soll. Die Basis für die Auswahl von Elementen sind Locators. Der Winkelmesser sowie die generischen Selenium-Selektoren verfügen auch über winkelspezifische Locators, die robuster und beständiger gegen Änderungen sind. Manchmal müssen jedoch auch in einer Angular-Anwendung reguläre Locators verwendet werden.

Parameter

Parameter	Detail
Wähler	Eine Zeichenfolge, die den Wert des Selektors angibt (abhängig vom Locator)

Examples

Winkelmesser-spezifische Locators (für Angular-basierte Anwendungen)

Diese Locators sollten nach Möglichkeit als Priorität verwendet werden, da sie dauerhafter für Änderungen in einer Anwendung sind als Locators, die auf css oder xpath basieren und leicht brechen können.

Binding Locator

Syntax

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

Beispiel

Aussicht

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

Locator

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

Unterstützt auch teilweise Übereinstimmungen

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

Genaues Bindungssuchgerät

Ähnlich wie die `binding` , ausgenommen, dass partielle Übereinstimmungen nicht zulässig sind.

Syntax

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

Beispiel

Aussicht

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

Locator

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

Modell Locator

Wählt ein Element mit einer Angular-Direktive aus

Syntax

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

Beispiel

Aussicht

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

Locator

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

Schaltfläche Text Locator

Wählt eine Schaltfläche basierend auf ihrem Text aus. Sollte nur verwendet werden, wenn der Schaltflächentext nicht häufig geändert wird.

Syntax

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

Beispiel

Aussicht

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

Locator

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

Text-Locator für teilweise Schaltflächen

Ähnlich wie `buttonText` , erlaubt jedoch teilweise Übereinstimmungen. Sollte nur verwendet werden, wenn der Schaltflächentext nicht häufig geändert wird.

Syntax

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

Beispiel

Aussicht

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

Locator

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

Repeater Locator

Wählt ein Element mit einer Angular-Repeater-Direktive aus

Syntax

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

Beispiel

Aussicht

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

Locator

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

Unterstützt auch teilweise Übereinstimmungen

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

Exakter Repeater Locator

Ähnlich wie `repeater` , erlaubt jedoch keine partiellen Übereinstimmungen

Syntax

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

Beispiel

Aussicht

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

Locator

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

CSS und Text Locator

Ein erweiterter CSS-Locator, in dem Sie auch den Textinhalt des Elements angeben können.

Syntax

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

Beispiel

Aussicht

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

Locator

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

Optionen Locator

Wählt ein Element mit einer Angular-Optionsanweisung aus

Syntax

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

Beispiel

Aussicht

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

Locator

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

Deep CSS Locator

CSS-Locator, der sich in das Schatten-DOM erstreckt

Syntax

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

Beispiel

Aussicht

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

Locator

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

Locator-Grundlagen

Locators allein geben kein Element zurück, mit dem in Protractor interagiert werden kann. Sie sind lediglich Anweisungen, die angeben, wie das Element gefunden wird.

Um auf das Element selbst zuzugreifen, verwenden Sie diese Syntax:

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

Hinweis: Auf das Element bzw. die Elemente wird erst dann tatsächlich zugegriffen, wenn eine Aktion ausgeführt wird. Das heißt, Protractor ruft das Element nur dann ab, wenn eine Aktion wie getText () für das Element aufgerufen wird.

Wenn Sie nur ein Element mit einem Locator auswählen möchten, verwenden Sie das `element` . Wenn Ihr Locator auf mehrere Elemente verweist, gibt `element` das zuerst gefundene zurück. `element` gibt einen `ElementFinder` .

Wenn Sie mehrere Elemente mit einem Locator auswählen möchten, gibt `element.all` alle gefundenen Elemente zurück. `element.all` gibt einen `ElementArrayFinder` , und auf jedes Element im Array kann mit verschiedenen Methoden zugegriffen werden, z. B. mit der `map` Funktion.

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

Verkettung von Lokatoren

Sie können mehrere Locators miteinander verketten, um ein Element in einer komplexen Anwendung auszuwählen. Sie können `locator` Objekte nicht direkt `ElementFinders` , `ElementFinders` müssen `ElementFinders` :

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

Es gibt keine Begrenzung für die Anzahl der Ketten, die Sie verwenden können. Am Ende erhalten Sie immer noch einen einzelnen `ElementFinder` oder `ElementArrayFinder` , abhängig von Ihren Locators.

Elemente suchen online lesen: https://riptutorial.com/de/protractor/topic/10825/elemente-suchen

Kapitel 4: Explizites Warten mit browser.wait ()

Examples

browser.sleep () vs browser.wait ()

Wenn es um Timing- `browser.sleep(<timeout_in_milliseconds>)` geht, ist es verlockend und einfach, einen "schnellen" `browser.sleep(<timeout_in_milliseconds>)` und `browser.sleep(<timeout_in_milliseconds>)` .

Das Problem ist, es würde eines Tages versagen. Es gibt keine allgemeine / generische Regel für die Einstellung des Sleep-Timeouts. Daher kann es an einem bestimmten Punkt aufgrund des Netzwerks oder der Leistung oder anderer Probleme dauern, bis eine Seite geladen wird oder ein Element sichtbar wird Die Zeit würde Sie am Ende mehr warten, als Sie eigentlich sollten.

`browser.wait()` hingegen funktioniert anders. Sie stellen eine Funktion Expected Condition für Protractor / WebDriverJS bereit, die ausgeführt werden soll, und warten, bis das Ergebnis der Funktion den Wert true ergibt. *Der Winkelmesser führt die Funktion kontinuierlich aus und stoppt, wenn das Ergebnis der Funktion als wahr ausgewertet wird oder ein konfigurierbarer Timeout erreicht wurde.*

Es gibt mehrere integrierte erwartete Bedingungen, Sie können jedoch auch eine benutzerdefinierte Bedingung erstellen und verwenden (Beispiel hier).

Explizites Warten mit browser.wait () online lesen:
https://riptutorial.com/de/protractor/topic/8297/explizites-warten-mit-browser-wait---

Kapitel 5: Kontrollfluss und Versprechen

Einführung

Protractor / WebDriverJS hat diesen Mechanismus namens Control Flow - es handelt sich um eine interne Warteschlange mit Versprechen, die die Ausführung des Codes organisiert.

Examples

Den Steuerfluss verstehen

Betrachten Sie den folgenden Test:

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

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

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

Wenn im folgenden Test die `console.log()` ausgeführt wird und Sie `HERE` auf der Konsole sehen, wurden keine Protractor-Befehle aus den vorherigen Zeilen ausgeführt. Dies ist ein völlig *asynchrones* Verhalten. Die Befehle werden als Versprechen dargestellt und in den Kontrollfluss gestellt, der die Versprechen nacheinander einzeln ausführen und lösen würde.

Sehen Sie mehr unter Versprechen und den Kontrollfluss .

Kontrollfluss und Versprechen online lesen:
https://riptutorial.com/de/protractor/topic/8580/kontrollfluss-und-versprechen

Kapitel 6: Seitenobjekte

Einführung

Seitenobjekte sind ein Entwurfsmuster, das zu weniger Code-Duplikaten, einfacher Wartung und mehr Lesbarkeit führt.

Examples

Erstes Seitenobjekt

```
/* 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;
```

Verwenden wir in unserem Test unsere erste Seitenobjektdatei.

```
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');
    });

});
```

Seitenobjekte online lesen: https://riptutorial.com/de/protractor/topic/9747/seitenobjekte

Kapitel 7: Testen von nicht eckigen Apps mit dem Winkelmesser

Einführung

Der Winkelmesser dient zum Testen von Winkelanwendungen. Es ist jedoch immer noch möglich, Anwendungen mit Winkelmesser bei Bedarf zu testen.

Examples

Änderungen erforderlich, um nicht winkelige App mit Protractor zu testen

Verwenden Sie `browser.driver` anstelle des `driver`

Verwenden Sie `browser.driver.ignoreSynchronization = true`

Grund : Der Winkelmesser wartet, bis die Winkelkomponenten vollständig auf einer Webseite geladen sind, bevor er mit der Ausführung beginnt. Da unsere Seiten jedoch nicht winklig sind, wartet Protractor solange auf das Laden des Winkels, bis der Test mit Timeout fehlschlägt. Also müssen wir dem Winkelmesser explizit sagen, nicht auf "Winkel" zu warten.

Testen von nicht eckigen Apps mit dem Winkelmesser online lesen: https://riptutorial.com/de/protractor/topic/8830/testen-von-nicht-eckigen-apps-mit-dem-winkelmesser

Kapitel 8: Winkelmesser-Debugger

Syntax

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

Bemerkungen

In diesem Abschnitt wird erläutert, wie Sie Tests für Winkelmesser durchführen können.

Examples

Browser.pause () verwenden

Die `pause()` -Methode ist eine der einfachsten Lösungen, die Protractor zum Debuggen des Codes bereitstellt. Um ihn zu verwenden, müssen Sie ihn in den Code einfügen, an dem Sie die Ausführung anhalten möchten. Sobald die Ausführung angehalten ist:

1. Sie können `c` (Typ C) verwenden, um vorwärts zu gehen. Seien Sie vorsichtig bei der Verwendung. Sie müssen diesen Befehl ohne Verzögerung schreiben, da Sie möglicherweise Zeitüberschreitungsfehler aus Ihrer Assertionsbibliothek erhalten, wenn Sie verzögert haben, c zu drücken.

2. `repl` , um den interaktiven Modus zu `repl` . Der interaktive Modus wird verwendet, um Browser-Befehle direkt an die Browser-Instanz zu senden. Im interaktiven Modus können Sie beispielsweise einen Befehl wie folgt ausgeben:

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

Die Hinweisausgabe des obigen Befehls wird direkt dort angezeigt, sodass Sie die Richtigkeit Ihres Befehls erkennen können.

Hinweis: Wenn Sie die Chrome Dev Tools geöffnet haben, müssen Sie sie schließen, bevor Sie den Test fortsetzen, da ChromeDriver nicht ausgeführt werden kann, wenn die Dev Tools geöffnet sind.

3. Wenn Sie den Debug-Modus mit `CTRL+C` , können Sie sich mit dem klassischen STRG + C-Befehl aus dem Debug-Modus entfernen.

```
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. Drücken Sie d, um mit der nächsten Debugger-Anweisung fortzufahren

Browser.debugger () verwenden

Sie können browser.debugger () verwenden, um die Ausführung zu stoppen. Sie können es an beliebiger Stelle in Ihren Code einfügen. Nach dieser Zeile wird die Ausführung angehalten, bis Sie nicht zum Fortfahren aufgefordert werden.

Hinweis: Um die Tests im Debugger-Modus auszuführen, müssen Sie einen Befehl wie folgt ausgeben:

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

Geben Sie `c` Ausführung starten und weiterhin nach dem Haltepunkt oder geben Sie `next` Befehl nächste Befehlsschritte in der nächsten Zeile in dem Steuerfluss.

Der in Protractor verwendete Debugger verwendet den Knoten-Debugger und unterbricht die Ausführung auf asynchrone Weise. Im folgenden Code wird der `browser.debugger()` aufgerufen, wenn `username.sendKeys('username')` ausgeführt wurde.

Hinweis: Da es sich um asynchrone Aufgaben handelt, müssen Sie das Standard-Timeout Ihrer Angaben erhöhen, da sonst ein Standard-Timeout-Ausnahmefehler ausgelöst wird!

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

Sie können den `repl` Modus durch Eingabe des Befehls aufrufen.

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

Dadurch wird der Befehl sendKeys als nächste Aufgabe ausgeführt und der Debugger erneut aufgerufen.

Man kann die `Port no.` ändern `Port no.` Sie möchten ihre Skripts debuggen, indem sie den Port einfach an die Debugger-Methode übergeben.

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

Die `debugger()` -Methode fügt einen Client von Protractor in den Browser ein. Sie können einige Befehle in der Browserkonsole ausführen, um die Elemente abzurufen. Ein Beispiel für die Verwendung eines clientseitigen Skripts ist:

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

Winkelmesser-Debugger online lesen:
https://riptutorial.com/de/protractor/topic/3910/winkelmesser-debugger

Kapitel 9: Winkelmesser-Konfigurationsdatei

Einführung

Die Konfigurationsdatei enthält Informationen, mit denen Protractor Ihr Testskript ausführt. Hier versuche ich einige Variationen zu geben.

Examples

Einfache Konfigurationsdatei - 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;
```

Konfigurationsdatei mit Funktionen - 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;
```

Konfigurationsdatei shardTestFiles - Chrome

Mit dieser Konfiguration können Sie Ihre gesamten Spezifikationsdateien in zwei Browser-Instanzen parallel ausführen. Es hilft, die gesamte Testausführungszeit zu reduzieren. Ändern Sie die maxInstances je nach Bedarf.

Hinweis : *Stellen Sie sicher, dass Ihre Tests unabhängig sind.*

```
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;
```

Konfigurationsdatei Multi-Funktionen emulieren - 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;
```

Winkelmesser-Konfigurationsdatei online lesen:
https://riptutorial.com/de/protractor/topic/9745/winkelmesser-konfigurationsdatei

Kapitel 10: XPath-Selektoren in Winkelmesser

Examples

Auswählen eines DOM-Elements mit einem Winkelmesser

Neben CSS-, Modell- und Bindungsselektoren kann der Winkelmesser auch Elemente mithilfe von xpath View finden

```
<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');
```

Elemente mit bestimmten Attributen auswählen

XPath-Selektoren können verwendet werden, um Elemente mit bestimmten Attributen wie Klasse, ID, Titel usw. auszuwählen.

Nach Klasse

Aussicht:

```
<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");
```

Ein Element kann jedoch mehrere Klassen haben. In solchen Fällen kann die Problemumgehung "Enthält" verwendet werden

Aussicht:

```
<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");
```

Der obige Code gibt Elemente zurück, die sowohl 'class = "HakunaMatata"' als auch 'class = "Hakuna Matata"' enthalten. Wenn Ihr Suchtext Teil einer durch Leerzeichen getrennten Liste ist, kann die folgende Problemumgehung verwendet werden:

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

Von id

Die ID bleibt der einfachste und genaueste Locator, mit dem ein Element ausgewählt werden kann.

Aussicht:

```
<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");
```

Wie bei Klassen kann mit der Funktion contains ein Element gesucht werden, das den angegebenen Text enthält.

Andere Attribute

Das Finden eines Elements mit einem bestimmten Attribut **title**

Aussicht

```
<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");
```

Ein Element mit einem bestimmten Text auswählen

Aussicht

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

Code

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

Wie bei anderen textbasierten Suchläufen können mit der contains-Funktion Elemente ausgewählt werden, deren Text () die gewünschte Übereinstimmung enthält.

Aussicht

```
<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
```

Auswählen eines Elements mit einem bestimmten Namensattribut

Aussicht

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

Code

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

XPath-Selektoren in Winkelmesser online lesen:
https://riptutorial.com/de/protractor/topic/7205/xpath-selektoren-in-winkelmesser

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit dem Winkelmesser	Bhoomi Bhalani, Community, Devmati Wadikar, Manuli Piyalka, olyv, Peter Stegnar, Praveen, Priyanshu Shekhar, SilentLupin, sonhu, Stephen Leppik
2	CSS-Selektoren	alecxe, Droogans, leon, Priyanshu Shekhar, sonhu
3	Elemente suchen	Sébastien Dufour-Beauséjour
4	Explizites Warten mit browser.wait ()	alecxe
5	Kontrollfluss und Versprechen	alecxe
6	Seitenobjekte	Barney, Suresh Salloju
7	Testen von nicht eckigen Apps mit dem Winkelmesser	Sakshi Singla
8	Winkelmesser-Debugger	Devmati Wadikar, Priyanshu Shekhar, Ram Pasala, Sakshi Singla, Stephen Leppik
9	Winkelmesser-Konfigurationsdatei	Barney
10	XPath-Selektoren in Winkelmesser	Shubhang