Бесплатная электронная книга

УЧУСЬ

protractor

Free unaffiliated eBook created from
Stack Overflow contributors.

#protractor

Около

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

глава 1: Начало работы с транспортиром

замечания

Транспортир представляет собой **сквозную** тестовую платформу для приложений AngularJS.

Транспортир - это обертка (построенная сверху) вокруг Selenium WebDriver, поэтому она содержит все функции, доступные в Selenium WebDriver. Кроме того, Protractor предоставляет несколько новых стратегий и функций локатора, которые очень полезны для автоматизации приложения AngularJS. Примеры включают такие вещи, как waitForAngular, By.binding, By.repeater, By.textarea, By.model, WebElement.all, WebElement.evaluate и т. Д.

Версии

Версия	Сведения о выпуске
0.0.1	2016-08-01

Examples

Установка и настройка транспортира (в Windows)

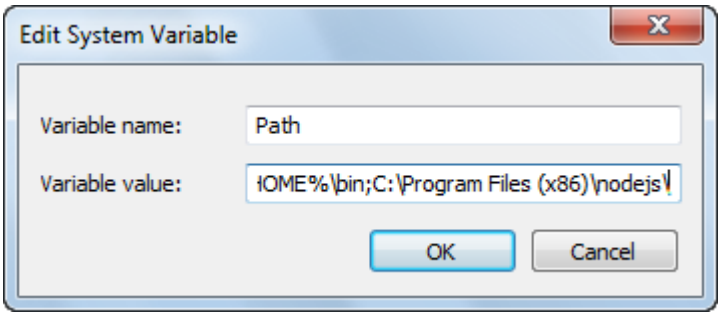
Требования: Транспортир требует установки следующих зависимостей перед установкой:

- Java JDK 1.7 или выше
- Node.js v4 или выше

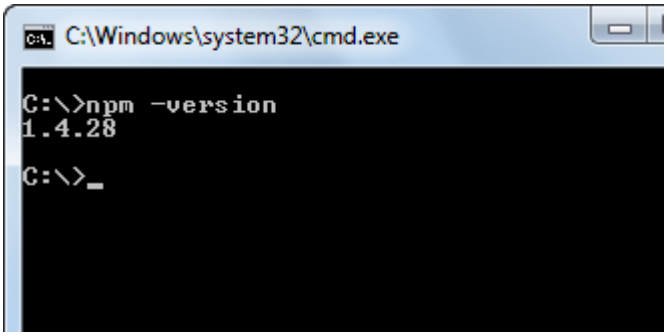
Монтаж:

Загрузите и установите Node.js с этого URL-адреса: https://nodejs.org/en/

Чтобы узнать, успешна ли установка Node.js, вы можете пойти и проверить переменные среды. «Путь» под системными переменными будет автоматически обновляться.



Вы также можете проверить это, введя команду `npm -version` в командной строке, которая предоставит вам установленную версию.



Теперь Protractor можно установить двумя способами: локально или глобально.

Мы можем установить транспортир в указанной папке или каталоге каталога проекта. Если мы устанавливаем в каталоге проекта, каждый раз, когда мы запускаем, мы должны запускаться только из этого местоположения.

Чтобы установить локально в каталог проекта, перейдите в папку проекта и введите команду

```
npm install protractor
```

Чтобы установить Protractor глобально выполните команду:

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

Это установит два инструмента командной строки, `protractor` и `webdriver-manager` . Запустите `protractor --version` чтобы убедиться, что транспортир успешно установлен.

`webdriver-manager` используется для загрузки двоичных файлов драйвера браузера и запуска сервера selenium.
Загрузите файлы драйверов браузера:

```
$ webdriver-manager update
```

Запустите сервер selenium с помощью:

```
$ webdriver-manager start
```

Чтобы загрузить драйвер Internet explorer, запустите команду `webdriver-manager update --ie` в командной строке. Это загрузит IEDriverServer.exe в папку selenium

Первый тест с использованием Protractor

Транспортеру нужны только два файла для запуска первого тестового, специфицированного (тестового кода) файла и файла конфигурации. Файл спецификации содержит тестовый код, а другой содержит данные конфигурации, такие как путь к файлам спецификаций, сведения о браузере, тестовый URL-адрес, параметры структуры и т. Д. Чтобы написать первый тест, мы будем предоставлять только адрес сервера селена и путь к файлу спецификации. Другие параметры, такие как браузер , тайм-аут, рамки будут подобраны до значений по умолчанию.

Браузер по умолчанию для Protractor - Chrome.

conf.js - Файл конфигурации

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

spec.js - Спецификация (тестовый код) файла

```
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 - путь к серверу, на котором запущен сервер webdriver.

specs - элемент массива, который содержит путь к тестовым файлам. Несколько путей могут быть заданы значениями, разделенными запятыми.

Опишите - Синтаксис из структуры Жасмин . `describe` синтаксис sta

Напишите тест-протрактор

Откройте новую командную строку или окно терминала и создайте чистую папку для тестирования.

Транспортеру нужны два файла для запуска, специфицированный файл и файл конфигурации.

Начнем с простого теста, который переместится к примеру списка todo на веб-сайте AngularJS и добавит новый элемент todo в список.

Скопируйте следующее в `spec.js`

описать ('angularjs homepage todo list', function () {it ('должен добавить 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);});});
```

Селективные тесты

Транспортир может выборочно запускать группы тестов, используя fdescribe () вместо описания ().

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

Транспортир может выборочно запускать тесты внутри групп, используя fit () вместо него ().

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

Если в fdescribe () нет fit (), тогда будет выполняться каждое его (). Тем не менее, fit ()

заблокирует его () вызовы внутри одного и того же описания () или 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
    });
});
```

Даже если fit () находится в описании () вместо fdescribe (), он будет запущен. Кроме того, будет выполняться любое его () внутри fdescribe (), которое не содержит fit ().

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

Ожидающие испытания

Транспортир позволяет проверять тесты в ожидании. Это означает, что транспортир не выполнит тест, но вместо этого выведет:

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

Или, если отключено с помощью xdescribe ():

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

Комбинации

- A xit () в xdescribe () выводит ответ xit ().
- A xit () внутри fdescribe () все равно будет рассматриваться как ожидающий.
- Подгонка () внутри xdescribe () все равно будет выполняться, и никакие ожидающие

тесты ничего не выдадут.

Транспортир: тестирование E2E для корпоративных угловых применений

Установка и настройка транспортировщика

Шаг 1. Загрузите и установите NodeJS отсюда. Убедитесь, что у вас установлена последняя версия узла. Здесь я использую узел v7.8.0. Для запуска селена вам потребуется установить Java Development Kit (JDK).

Шаг 2. Откройте терминал и введите следующую команду, чтобы установить транспортир по всему миру.

```
npm install -g protractor
```

Это установит два инструмента, таких как транспортир и менеджер webdriver. Вы можете проверить свою установку укладчика, выполнив следующую команду: `protractor -version`. Если Protractor установлен успешно, система отобразит установленную версию (например, версию 5.1.1). В противном случае вам придется перепроверить установку. Шаг 3. Обновите менеджер webdriver, чтобы загрузить необходимые бинарные файлы.

```
webdriver-manager update
```

Шаг 4: Следующая команда запустит сервер Selenium. Этот шаг запустит менеджер веб-драйверов в фоновом режиме и будет прослушивать любые тесты, выполняемые через транспортир.

Начало webdriver-manager Вы можете увидеть информацию о статусе сервера по адресу `http://localhost:4444/wd/hub/static/resource/hub.html`.

Написание первого тестового примера с использованием Транспортера:

Прежде чем перейти к написанию тестового примера, мы должны подготовить два файла, которые являются файлом конфигурации и спецификацией.

В файле конфигурации:

```
//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'
    }
}
```

Основные понятия терминов, используемых в файле конфигурации:

baseUrl - базовый URL для тестируемого приложения.

seleniumAddress - для подключения к серверу Selenium, который уже запущен.

specs - Местоположение вашего файла спецификации

directConnect : true - для прямого подключения к драйверам браузера.

возможности. Если вы тестируете в одном браузере, используйте параметр возможностей. Если вы тестируете несколько браузеров, используйте массив multiCapabilities.

Вы можете найти больше параметров конфигурации от здесь . Они описали всю возможную терминологию с ее определением.

В файле 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');
      });
    });
```

Основные понятия терминов, используемых в файле spec:

По умолчанию Protractor использует структуру jasmine для своего интерфейса тестирования. Синтаксис «описать» и «он» - из структуры жасмина. Вы можете узнать больше отсюда. Запуск первого тестового примера:

Перед запуском тестового примера убедитесь, что ваш менеджер webdriver и ваше приложение работают на разных вкладках вашего терминала.

Теперь запустите тест с помощью:

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

Вы должны увидеть, что браузер Chrome открывается с вашим url приложения и закрывается. Выход теста должен быть 1 тест, 1 утверждение, 0 сбоев.

Браво! Вы успешно запускаете свой первый тестовый пример.

Прочитайте Начало работы с транспортиром онлайн:
https://riptutorial.com/ru/protractor/topic/933/начало-работы-с-транспортиром

глава 2: Контроль потока и обещаний

Вступление

У Protractor / WebDriverJS есть этот механизм, называемый Control Flow - это внутренняя очередь обещаний, он сохраняет выполнение кода.

Examples

Понимание потока управления

Рассмотрим следующий тест:

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

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

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

В следующем тесте, когда `console.log()` выполняется, и вы видите `HERE` на консоли, ни одна из команд Protractor из предыдущих строк не была выполнена. Это совершенно *асинхронное* поведение. Команды представлены в виде обещаний и помещаются в поток управления, который будет выполнять и разрешать обещания последовательно, один за другим.

См. Больше в Promises и Flow Control .

Прочитайте Контроль потока и обещаний онлайн:
https://riptutorial.com/ru/protractor/topic/8580/контроль-потока-и-обещаний

глава 3: Объекты страницы

Вступление

Объекты страницы - это шаблон проектирования, который приводит к уменьшению количества дубликатов кода, простому обслуживанию и большей удобочитаемости.

Examples

Объект первой страницы

```
/* 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;
```

Давайте используем наш первый объектный файл страницы в нашем тесте.

```
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');
    });

});
```

Прочитайте Объекты страницы онлайн: https://riptutorial.com/ru/protractor/topic/9747/объекты-страницы

глава 4: Отладка отрывника

Синтаксис

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

замечания

В этом разделе объясняется, как мы можем отлаживать тесты транспортировщика.

Examples

Использование browser.pause ()

Метод `pause()` является одним из самых простых решений, с помощью которых Protractor позволяет вам отлаживать код, чтобы его использовать, вам нужно добавить его в свой код, где вы хотите приостановить выполнение. Когда выполнение приостановлено, выполните следующее:

1. Вы можете использовать `c` (тип C) для перемещения вперед. Будьте осторожны при использовании, вы должны написать эту команду без каких-либо задержек, поскольку вы можете получить ошибку тайм-аута из вашей библиотеки утверждений, если вы задерживаете нажимать c.

2. Введите `repl` для входа в интерактивный режим. Интерактивный режим используется для отправки команд браузера напрямую, чтобы открыть экземпляр браузера. Например, в интерактивном режиме вы можете выдать команду следующим образом:

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

Заметьте, что выводимый выше вывод появляется непосредственно там, что позволяет узнать правильность вашей команды.

Примечание. Если вы открыли инструменты Chrome Dev, вы должны закрыть их перед продолжением теста, потому что ChromeDriver не может работать, когда инструменты Dev открыты.

3. Выйдите из режима отладки с помощью `CTRL+C` , вы можете выйти из режима отладки, используя классическую команду 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. Нажмите d, чтобы перейти к следующему отладчику

Использование browser.debugger ()

Вы можете использовать browser.debugger (), чтобы остановить выполнение. Вы можете вставить его в любое место в своем коде, и оно прекратит выполнение после этой строки, пока вы не приступите к продолжению.

Примечание. Чтобы запустить тесты в режиме отладчика, вы должны выполнить команду следующим образом:

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

Введите `c` , чтобы начать выполнение и продолжить после останова или введите `next` command.The действие следующей команды к следующей строке в потоке управления.

Отладчик, используемый в Protractor, использует отладчик узлов и приостанавливает выполнение асинхронным способом. Например, в нижнем коде `browser.debugger()` будет вызван, когда будет выполнено `username.sendKeys('username')` .

Примечание. Поскольку это асинхронные задачи, вам придется увеличить таймаут по умолчанию для ваших спецификаций, иначе будет выбрано исключение по тайм-ауту по умолчанию!

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

Можно войти в режим `repl` , введя команду-

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

Это запустит команду sendKeys в качестве следующей задачи, а затем повторно введите отладчик.

Можно изменить `Port no.` они хотят отлаживать свои сценарии, просто передав порт методу отладчика,

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

Метод `debugger()` внедряет клиентскую сторону из Protractor в браузер, и вы можете запускать несколько команд в консоли браузера для извлечения элементов. Одним из примеров использования сценария на стороне клиента является:

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

Прочитайте Отладка отрывника онлайн: https://riptutorial.com/ru/protractor/topic/3910/отладка-отрывника

глава 5: Поиск элементов

Вступление

Чтобы иметь возможность взаимодействовать со страницей, вам нужно указать Protractor точно, какой элемент искать. Основой, используемой для выбора элементов, являются локаторы. Транспортир, а также, в том числе, селектор Селен, также имеют углообразные локаторы, которые являются более надежными и постоянными для изменений. Однако иногда даже в угловом приложении необходимо использовать обычные локаторы.

параметры

параметр	подробность
селектор	Строка, которая задает значение селектора (зависит от локатора)

Examples

Специфические указатели трактора (для приложений на основе углов)

Эти локаторы должны использоваться как приоритет, когда это возможно, потому что они более устойчивы к изменениям в приложении, а затем локаторы на основе css или xpath, которые могут легко сломаться.

Локатор привязки

Синтаксис

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

пример

Посмотреть

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

Локатор

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

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

Также поддерживает частичные совпадения

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

Точный привязывающий локатор

Подобно `binding` , кроме частичных совпадений, не допускается.

Синтаксис

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

пример

Посмотреть

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

Локатор

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

Локатор модели

Выбирает элемент с директивой «Угловая модель»

Синтаксис

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

пример

Посмотреть

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

Локатор

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

Текстовый локатор

Выбирает кнопку на основе ее текста. Следует использовать только в том случае, если текст кнопки не ожидается часто меняться.

Синтаксис

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

пример

Посмотреть

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

Локатор

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

Частичный текстовый локатор

Подобно `buttonText` , но допускает частичные совпадения. Следует использовать только в том случае, если текст кнопки не ожидается часто меняться.

Синтаксис

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

пример

Посмотреть

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

Локатор

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

Локатор ретранслятора

Выбирает элемент с указателем углового репитера

Синтаксис

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

пример

Посмотреть

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

Локатор

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

Также поддерживает частичные совпадения

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

Точный локатор ретранслятора

Подобно `repeater` , но не допускает частичных совпадений

Синтаксис

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

пример

Посмотреть

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

Локатор

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

CSS и текстовый локатор

Расширенный CSS-локатор, где вы также можете указать текстовое содержимое элемента.

Синтаксис

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

пример

Посмотреть

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

Локатор

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

Локатор опций

Выбирает элемент с директивой «Угловые параметры»

Синтаксис

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

пример

Посмотреть

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

Локатор

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

Глубокий локатор CSS

Локатор CSS, который распространяется на тень DOM

Синтаксис

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

пример

Посмотреть

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

Локатор

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

Основы локатора

Локаторы сами по себе не возвращают элемент, с которым можно взаимодействовать в Protractor, это просто инструкции, указывающие на то, что Protractor, как найти элемент.

Чтобы получить доступ к самому элементу, используйте этот синтаксис:

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

Примечание: элемент (ы) на самом деле не обращается к нему до тех пор, пока на нем не будет выполнено действие - то есть, Прототранс только начнет возвращать элемент, когда на элемент вызывается действие, такое как getText ().

Если вы хотите выбрать только один элемент с помощью локатора, используйте `element` . Если ваш локатор указывает на несколько элементов, `element` вернет первый найденный. `element` возвращает `element ElementFinder` .

Если вы хотите выбрать несколько элементов с помощью локатора, `element.all` вернет все найденные элементы. `element.all` возвращает `ElementArrayFinder` , и каждый элемент в

массиве может быть доступен с использованием разных методов - например, функции `map` .

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

Цепочные локаторы

Вы можете связать несколько локаторов, чтобы выбрать элемент в сложном приложении. Вы не можете напрямую `locator` объекты `locator` , вы должны `ElementFinders` :

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

Нет предела тому, сколько цепей вы можете использовать; в конце концов, вы все равно получите один `ElementFinder` или `ElementArrayFinder` , в зависимости от ваших локаторов.

Прочитайте Поиск элементов онлайн: https://riptutorial.com/ru/protractor/topic/10825/поиск-элементов

глава 6: Селекторы CSS

Синтаксис

- by.css ('CSS-селектор')
- by.id ('ID')
- by.model ('модель')
- by.binding ('связывание')

параметры

параметр	подробности
CSS-селектор	Селектор css, например `'.class-name'` чтобы выбрать элемент на основе имени класса
Я бы	Идентификатор элемента dom
модель	Модель, используемая для элемента dom
переплет	Имя привязки, которое используется для привязки к определенному элементу

замечания

Как написать селектор css?

Наиболее важными атрибутами для записи css-селекторов являются класс и id dom. Для примера, если html dom выглядит как пример:

```
<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>
```

Затем, чтобы выбрать поле ввода электронной почты, вы можете написать селектор css следующим образом:

1. **Использование имени класса** : Имя класса в селекторе css начинается со специального символа (точка). Селектор css для этого будет выглядеть как `.form-control` .

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

Поскольку класс `form-control` разделяется обоими элементами ввода, поэтому возникает проблема двуличия в локаторах. Поэтому в такой ситуации, если id доступен, вы всегда должны использовать id вместо имени класса.

2. **Использование идентификатора** : идентификатор в селекторе css начинается с специального символа # (хеш). Таким образом, css-селектор, использующий id для элемента ввода электронной почты, будет написан следующим образом:

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

3. **Использование нескольких имен классов** . Если элемент dom имеет несколько классов, то вы можете с комбинацией классов как css-селектор. Например, если элемент dom выглядит следующим образом:

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

4. **Использование имени тега с другими атрибутами** : Общее выражение для записи css-селектора с использованием имени тега и других атрибутов - `tagname[attribute-type='attribute-vallue']` . Итак, следуя выражению, css-локатор для кнопки входа может быть сформирован следующим образом:

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

Examples

$ и $$ быстрые ссылки на селектор CSS

API-интерфейс Protractor позволяет локаторам элементов CSS использовать обозначение ярлыка jQuery типа `$()` .

Обычный CSS-элемент :

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

Ярлык `$()` Элемент элемента CSS :

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

Для нахождения нескольких элементов под локатором используйте обозначение ярлыка `$$()` .

Обычный CSS-элемент :

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

Ярлык `$$()` Элемент элемента CSS :

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

Введение в локаторы

Локатор в Protractor используется для выполнения действий над элементами HTML dom. Наиболее распространенными и лучшими локаторами, используемыми в Protractor, являются css, id, model и binding. Например, обычно используемые локаторы:

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

Выбрать элемент с помощью точного значения атрибута HTML

Чтобы выбрать элемент с помощью точного атрибута HTML, используйте шаблон локатора css [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"]'));
```

Выбрать элемент по атрибуту HTML, который содержит указанное значение

Чтобы выбрать элемент с помощью атрибута HTML, который содержит указанное значение, используйте шаблон локатора css [attribute * = value]

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

Прочитайте Селекторы CSS онлайн: https://riptutorial.com/ru/protractor/topic/1524/селекторы-css

глава 7: Селекторы XPath в Protractor

Examples

Выбор элемента DOM с помощью транспортира

Помимо селекторов CSS, модели и привязки, транспортир также может находить элементы, используя xpath View

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

Код

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

Выбор элементов с определенными атрибутами

Селектора XPath могут использоваться для выбора элементов с определенными атрибутами, такими как класс, идентификатор, заголовок и т. Д.

По классу

Посмотреть:

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

Код:

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

Однако элемент может иметь несколько классов. В таких случаях может быть использовано обходное решение «содержит»

Посмотреть:

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

Код:

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

Вышеупомянутый фрагмент кода возвращает элементы, содержащие как «class =» HakunaMatata «», так и «class =» Hakuna Matata ». Если ваш текст поиска является частью списка, разделенного пробелами, может использоваться следующее обходное решение:

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

По идентификатору

ID остается самым простым и точным локатором, который можно использовать для выбора элемента.

Посмотреть:

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

Код:

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

Как и в случае с классами, функция contains может использоваться для нахождения элемента, содержащего данный текст.

Другие атрибуты

Поиск элемента с заданным атрибутом **заголовка**

Посмотреть

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

Код

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

Выбор элемента с определенным текстом

Посмотреть

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

Код

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

Как и в других текстовых поисках, функция contains может использоваться для выбора элементов с текстом (), содержащим требуемое совпадение.

Посмотреть

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

Код

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

Выбор элемента с определенным атрибутом имени

Посмотреть

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

Код

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

Прочитайте Селекторы XPath в Protractor онлайн:
https://riptutorial.com/ru/protractor/topic/7205/селекторы-xpath-в-protractor

глава 8: Тестирование не угловых приложений с помощью Protractor

Вступление

Транспортир предназначен для испытания угловых применений. Тем не менее, по-прежнему можно протестировать немагнитные приложения с помощью Транспортера, если это необходимо.

Examples

Изменения, необходимые для тестирования негласного приложения с помощью Protractor

Использовать `browser.driver` вместо `driver`

Использовать `browser.driver.ignoreSynchronization = true`

Причина : Транспортир ждет, пока угловые компоненты полностью загрузятся на веб-странице, прежде чем она начнет выполнение. Однако, поскольку наши страницы не являются угловыми, Protractor продолжает ждать «углового» для загрузки до тех пор, пока тест не завершится с таймаутом. Итак, нам нужно явно сказать, что Протрактор не должен ждать «углового»,

Прочитайте Тестирование не угловых приложений с помощью Protractor онлайн: https://riptutorial.com/ru/protractor/topic/8830/тестирование-не-угловых-приложений-с-помощью-protractor

глава 9: Файл конфигурации транспортира

Вступление

Файл конфигурации содержит информацию, которую использует Protractor для запуска тестового сценария. Здесь я попытаюсь дать несколько разных вариантов.

Examples

Простой файл конфигурации - 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;
```

Файл конфигурации с возможностями - 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;
```

Файл конфигурации shardTestFiles - Chrome

Эта конфигурация позволяет «запускать ваши общие файлы спецификаций в двух экземплярах браузера параллельно. Это помогает сократить общее время выполнения теста. Измените maxInstances на основе ваших потребностей.

Примечание . *Убедитесь, что ваши тесты независимы.*

```
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;
```

конфигурационный файл мульти-возможностей эмулировать - хром

```
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;
```

Прочитайте Файл конфигурации транспортира онлайн:

https://riptutorial.com/ru/protractor/topic/9745/файл-конфигурации-транспортира

глава 10: Явные ожидания с помощью browser.wait ()

Examples

browser.sleep () vs browser.wait ()

Когда дело доходит до решения проблемы синхронизации, заманчиво и легко поставить «быстрый» `browser.sleep(<timeout_in_milliseconds>)` и двигаться дальше.

Проблема в том, что когда-нибудь это произойдет. Существует не золотое / общее правило о том, какой тайм-аут ожидания установить, и, следовательно, в какой-то момент из-за сети или производительности или других проблем, может потребоваться больше времени для загрузки страницы или элемента, чтобы стать видимым и т. Д. Кроме того, большая часть время, вы в конечном итоге ожидаете больше, чем вы на самом деле должны.

`browser.wait()` другой стороны, `browser.wait()` работает по-разному. Вы предоставляете функцию ожидаемого условия для выполнения Protractor / WebDriverJS и ожидаете, что результат функции будет равен true. *Транспортир непрерывно выполнял функцию и останавливался, как только результат функции оценивался как истина или был достигнут настраиваемый тайм-аут.*

Существует несколько встроенных ожидаемых условий, но вы также можете создать и использовать пользовательский (образец здесь).

Прочитайте Явные ожидания с помощью browser.wait () онлайн: https://riptutorial.com/ru/protractor/topic/8297/явные-ожидания-с-помощью-browser-wait---

кредиты

S. No	Главы	Contributors
1	Начало работы с транспортиром	Bhoomi Bhalani, Community, Devmati Wadikar, Manuli Piyalka, olyv, Peter Stegnar, Praveen, Priyanshu Shekhar, SilentLupin, sonhu, Stephen Leppik
2	Контроль потока и обещаний	alecxe
3	Объекты страницы	Barney, Suresh Salloju
4	Отладка отрывника	Devmati Wadikar, Priyanshu Shekhar, Ram Pasala, Sakshi Singla, Stephen Leppik
5	Поиск элементов	Sébastien Dufour-Beauséjour
6	Селекторы CSS	alecxe, Droogans, leon, Priyanshu Shekhar, sonhu
7	Селекторы XPath в Protractor	Shubhang
8	Тестирование не угловых приложений с помощью Protractor	Sakshi Singla
9	Файл конфигурации транспортира	Barney
10	Явные ожидания с помощью browser.wait ()	alecxe