Tutorial

This is a simple tutorial that shows you how to set up Protractor and start running tests.

Prerequisites

Protractor is a Node.js (http://nodejs.org/) program. To run, you will need to have Node.js installed. You will download Protractor package using npm (https://www.npmjs.org/), which comes with Node.js. Check the version of Node.js you have by running `node --version` . Then, check the compatibility notes (https://github.com/angular/protractor#compatibility) in the Protractor README to make sure your version of Node.js is compatible with Protractor.

By default, Protractor uses the Jasmine (http://jasmine.github.io/) test framework for its testing interface. This tutorial assumes some familiarity with Jasmine, and we will use version 2.4.

This tutorial will set up a test using a local standalone Selenium Server to control browsers. You will need to have the Java Development Kit (JDK) (http://www.oracle.com/technetwork/java/javase/downloads/index.html) installed to run the standalone Selenium Server. Check this by running `java -version` from the command line.

Setup

Use npm to install Protractor globally with:

```
npm install -g protractor
```

This will install two command line tools, `protractor` and `webdriver-manager` . Try running `protractor --version` to make sure it's working.

The `webdriver-manager` is a helper tool to easily get an instance of a Selenium Server running. Use it to download the necessary binaries with:

```
webdriver-manager update
```

Now start up a server with:

```
webdriver-manager start
```

This will start up a Selenium Server and will output a bunch of info logs. Your Protractor test will send requests to this server to control a local browser. Leave this server running throughout the tutorial. You can see information about the status of the server at `http://localhost:4444/wd/hub` .

Step 0 - write a test

Open a new command line or terminal window and create a clean folder for testing.

Protractor needs two files to run, a **spec file** and a **configuration file**.

Let's start with a simple test that navigates to an example AngularJS application and checks its title. We'll use the Super Calculator application at http://juliemr.github.io/protractor-demo/ (http://juliemr.github.io/protractor-demo/).

Copy the following into spec.js:

```
// spec.js
describe('Protractor Demo App', function() {
  it('should have a title', function() {
    browser.get('http://juliemr.github.io/protractor-demo/');

    expect(browser.getTitle()).toEqual('Super Calculator');
  });
});
```

The `describe` and `it` syntax is from the Jasmine framework. `browser` is a global created by Protractor, which is used for browser-level commands such as navigation with `browser.get`.

Now create the configuration file. Copy the following into conf.js:

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

This configuration tells Protractor where your test files (`specs`) are, and where to talk to your Selenium Server (`seleniumAddress`). It specifies that we will be using Jasmine for the test framework. It will use the defaults for all other configuration. Chrome is the default browser.

Now run the test with

```
protractor conf.js
```

You should see a Chrome browser window open up and navigate to the Calculator, then close itself (this should be very fast!). The test output should be `1 tests, 1 assertion, 0 failures`. Congratulations, you've run your first Protractor test!

Step 1 - interacting with elements

Now let's modify the test to interact with elements on the page. Change spec.js to the following:

```
// spec.js
describe('Protractor Demo App', function() {
  it('should add one and two', function() {
    browser.get('http://juliemr.github.io/protractor-demo/');
    element(by.model('first')).sendKeys(1);
    element(by.model('second')).sendKeys(2);

    element(by.id('gobutton')).click();

    expect(element(by.binding('latest')).getText()).
        toEqual('5'); // This is wrong!
  });
});
```

This uses the globals `element` and `by`, which are also created by Protractor. The `element` function is used for finding HTML elements on your webpage. It returns an ElementFinder object, which can be used to interact with the element or get information from it. In this test, we use `sendKeys` to type into `<input>`s, `click` to click a button, and `getText` to return the content of an element.

`element` takes one parameter, a Locator, which describes how to find the element. The `by` object creates Locators. Here, we're using three types of Locators:

- `by.model('first')` to find the element with `ng-model="first"`. If you inspect the Calculator page source, you will see this is `<input type="text" ng-model="first">`.
- `by.id('gobutton')` to find the element with the given id. This finds `<button id="gobutton">`.
- `by.binding('latest')` to find the element bound to the variable `latest`. This finds the span containing `{{latest}}`

  Learn more about locators and ElementFinders.

Run the tests with

```
protractor conf.js
```

You should see the page enter two numbers and wait for the result to be displayed. Because the result is 3, not 5, our test fails. Fix the test and try running it again.

Step 2 - writing multiple scenarios

Let's put these two tests together and clean them up a bit. Change spec.js to the following:

```
// spec.js
describe('Protractor Demo App', function() {
  var firstNumber = element(by.model('first'));
  var secondNumber = element(by.model('second'));
  var goButton = element(by.id('gobutton'));
  var latestResult = element(by.binding('latest'));

  beforeEach(function() {
    browser.get('http://juliemr.github.io/protractor-demo/');
  });

  it('should have a title', function() {
    expect(browser.getTitle()).toEqual('Super Calculator');
  });

  it('should add one and two', function() {
    firstNumber.sendKeys(1);
    secondNumber.sendKeys(2);

    goButton.click();

    expect(latestResult.getText()).toEqual('3');
  });

  it('should add four and six', function() {
    // Fill this in.
    expect(latestResult.getText()).toEqual('10');
  });

  it('should read the value from an input', function() {
    firstNumber.sendKeys(1);
    expect(firstNumber.getAttribute('value')).toEqual('1');
  });
});
```

Here, we've pulled the navigation out into a `beforeEach` function which is run before every `it` block. We've also stored the ElementFinders for the first and second input in nice variables that can be reused. Fill out the second test using those variables, and run the tests again to ensure they pass.

In the last assertion we read the value from the input field with `firstNumber.getAttribute('value')` and compare it with the value we have set before.

Step 3 - changing the configuration

Now that we've written some basic tests, let's take a look at the configuration file. The configuration file lets you change things like which browsers are used and how to connect to the Selenium Server. Let's change the browser. Change conf.js to the following:

```
// conf.js
exports.config = {
  framework: 'jasmine',
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['spec.js'],
  capabilities: {
    browserName: 'firefox'
  }
}
```

Try running the tests again. You should see the tests running on Firefox instead of Chrome. The `capabilities` object describes the browser to be tested against. For a full list of options, see the config file (https://github.com/angular/protractor/blob/5.4.1/lib/config.ts).

You can also run tests on more than one browser at once. Change conf.js to:

```
// conf.js
exports.config = {
  framework: 'jasmine',
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['spec.js'],
  multiCapabilities: [{
    browserName: 'firefox'
  }, {
    browserName: 'chrome'
  }]
}
```

Try running once again. You should see the tests running on Chrome and Firefox simultaneously, and the results reported separately on the command line.

Step 4 - lists of elements

Let's go back to the test files. Feel free to change the configuration back to using only one browser.

Sometimes, you will want to deal with a list of multiple elements. You can do this with `element.all` , which returns an ElementArrayFinder. In our calculator application, every operation is logged in the history, which is implemented on the site as a table with `ng-repeat` . Let's do a couple of operations, then test that they're in the history. Change spec.js to:

```
// spec.js
describe('Protractor Demo App', function() {
  var firstNumber = element(by.model('first'));
  var secondNumber = element(by.model('second'));
  var goButton = element(by.id('gobutton'));
  var latestResult = element(by.binding('latest'));
  var history = element.all(by.repeater('result in memory'));

  function add(a, b) {
    firstNumber.sendKeys(a);
    secondNumber.sendKeys(b);
    goButton.click();
  }

  beforeEach(function() {
    browser.get('http://juliemr.github.io/protractor-demo/');
  });

  it('should have a history', function() {
    add(1, 2);
    add(3, 4);

    expect(history.count()).toEqual(2);

    add(5, 6);

    expect(history.count()).toEqual(0); // This is wrong!
  });
});
```

We've done a couple things here - first, we created a helper function, `add`. We've added the variable `history`. We use `element.all` with the `by.repeater` Locator to get an ElementArrayFinder. In our spec, we assert that the history has the expected length using the `count` method. Fix the test so that the second expectation passes.

`ElementArrayFinder` has many methods in addition to `count`. Let's use `last` to get an ElementFinder that matches the last element found by the Locator. Change the test to:

```
  it('should have a history', function() {
    add(1, 2);
    add(3, 4);

    expect(history.last().getText()).toContain('1 + 2');
    expect(history.first().getText()).toContain('foo'); // This is wrong!
  });
```

Since the Calculator reports the oldest result at the bottom, the oldest addition (1 + 2) be the last history entry. We're using the `toContain` Jasmine matcher to assert that the element text contains "1 + 2". The full element text will also contain the timestamp and the result.

Fix the test so that it correctly expects the first history entry to contain the text "3 + 4".

ElementArrayFinder also has methods `each`, `map`, `filter`, and `reduce` which are analogous to JavaScript Array methods. Read the API for more details (http://angular.github.io/protractor/#/api?view=ElementArrayFinder).

Where to go next

This should get you started writing tests. To learn more, see the documentation Table of Contents.