Лекция 28. Отладка и тестирование Angular приложений

[**Отладка Angular приложения**](#_vunjwy5lj90c) **3**

[RxJS .tap()](#_nli9dq5kqdsm) 3

[ng.probe($0).componentInstance](#_jqwebn17ppip) 3

[Augury](#_2au68iu8fom9) 3

[Redux chrome extension](#_mf1erv929pzw) 3

[**Тестирование кода**](#_9fq623orwh64) **4**

[Unit-тестирование](#_v968haw9fvg) 4

[Как писать unit-тесты](#_zdwfuzvjwlll) 4

[Behavior-тестирование](#_dp8pw9ydlwww) 5

[**Задание для закрепления материала**](#_hj2fwexzuuul) **6**

[**Дополнительная информация**](#_wozml8fbqll0) **7**

# Отладка Angular приложения

## RxJS .tap()

При использовании RxJS и Observable имеется возможность отлаживать каждый итерируемый элемент при помощи pipe-а .tap(). Он ничего не делает, но позволяет просмотреть и продиагностировать элемент на каждом этапе:

**items$ = http.get(‘assets/data.js’).pipe(  
 tap( item => {console.log( item )} ),  
 map( item => item \* item )  
 tap( item => { console.log( item ) } )  
)**

Т.е. оператор tap позволяет отследить элемент Observable на каждом этапе.

## ng.probe($0).componentInstance

Для того, чтобы проинспектировать компонент Angular приложения, необходимо выделить элемент в инспекторе ( вкладка Elements ), перейти во вкладку Console и набрать команду **ng.probe($0).componentInstance.** Это позволит отследить состояние элемента

## Augury

Расширение Chrome для отладки Angular приложений

## Redux chrome extension

Если в приложении используется Redux, расширение для Chrome будет очень полезным.

# Тестирование кода

Для того, чтобы гарантировать работоспособность кода, применяют тестирование различного рода: Unit тестирование, Behavior тестирование, etc.

## Unit-тестирование

Помимо очевидной цели - гарантировать работоспособность кода, есть цель неочевидная - гарантировать работоспособность измененного кода. Т.е. у вас есть код, который вы поддерживаете, написанные unit-тесты уже гарантируют, что код у вас работает и работает правильно. В случае если вы проводите рефакторинг, старые тесты все также гарантируют, что код у вас работает и работает правильно, даже если вы его изменили.

В том случае, если результат полученный в измененном отличается от ожидаемого в тестах, остается 2 варианта:

* ваш код работает неправильно
* ваши тесты устарели и не соответствуют бизнес-требованиям.

В любом случае, внесенные изменения будут осознанными. Это гарантируется тестами.

Unit-тесты проверяют функционал каждой функции, каждого класса, каждого сервиса и т.д. Каждый тест должен быть атомарным, т.е. независимым от других тестов.

### Как писать unit-тесты

Тесты будут подготовлены с применением фрэймворка Codeception (PHP), просто в качестве примера.

Давайте возьмем за пример класс калькулятора:

**class Calculator {  
 plusAction( x:number, y:number ) {**

**return x + y;  
 }**

**minusAction() { .... }**

**multiplyAction() { .... }**

**…..  
}**

Напишем тесты для метода plusAction:

class CalculatorTest {

testPlusActionFirstParameterIsRequired() {

this.expectException( … ); // тут мы указываем, что ожидается ошибка

Calculator.plusAction( , 10);  
 }

testPlusActionSecondParameterIsRequired() {  
 this.expectException( … ); // тут мы указываем, что ожидается ошибка

Calculator.plusAction( 1 );  
 }

testPlusActionFirstParameterIsNumberOnly() {

this.expectException( … ); // тут мы указываем, что ожидается ошибка

Calculator.plusAction( ‘string’, 10 );  
 }  
  
 testPlusActionSecondParameterIsNumberOnly() {

this.expectException( … ); // тут мы указываем, что ожидается ошибка

Calculator.plusAction( 10, ‘string’ );  
 }

testPlusActionReturnsCorrectResult() {  
 // тут мы говорим какой результат ожидаем, т.е. возвращаемый результат будет равен ожидаемому( )

this.assertEquals( 10, Calculator.plusAction(8, 2) );  
 this.assertEquals( 3, Calculator.plusAction(1, 2) );  
 this.assertEquals( 145, Calculator.plusAction(15, 130) );

this.assertEquals( 5.2, Calculator.plusAction(1.2, 4) );

}

}

Заметьте, что на один метод у нас написано 5 тестов-кейсов, это не предел. Каждый кейс описывает свою ситуацию и является атомарным.

## Behavior-тестирование

Behavior тесты отличаются от Unit тем, что тестируют не сам код, а результат его выполнения, т.е. то, что видит пользователь. Например страница должна отображать:

* меню (тег nav или menu)
* заголовок страницы (тег h1)
* текст (блок main)
* Реакцию на отправку формы.

# Задание для закрепления материала

Доделать игру “Морской бой”.

# Дополнительная информация

<https://angular.io/guide/testing>

Отладка Angular приложений

<https://medium.com/front-end-hacking/a-guide-to-debugging-angular-applications-5a36bd88b4cf>

Фрэймворки:  
<http://unitjs.com/>

<https://karma-runner.github.io/>

<https://mochajs.org/>