Agular 单元测试编写入门指南

Angular 是一个mobile和桌面web应用的开发平台。核心概念包括组件、依赖注入、绑定。在测试方面提供了基于TestBed测试库，借助通用的单元测试框架jasmine等可以开展单元测试工作。本文介绍如何使用TestBed库进行单元测试用例的编写。

单元测试用例编写基本过程

单元测试用例编写和运行的前提是：配置好Agular项目的文件结构，单元测试代码统一放置在 组件.spec.ts中，同时在开发测试框架中进行相关配置，这些配置可能包括编译、nodejs、karma、jasmine等配置。

测试用例编写通用步骤:

- 导入Angular相关包、被测组件类、支撑组件类（包括打桩类、第三方类）
- 在代码编写框架下配置测试环境
- 编写测试条目

一、导入Angular相关包、被测组件类、支撑组件类（包括打桩类、第三方类）

在导入相关类时，注意导入的类与被测组件类有关。

> 如果组件中无路由，则可以不导入路由类，如果不选择DOM，可以不导入By，对所有导入的类需要有一定的了解。

下面一步步详细说明：

1. 导入Angular 包，根据调用情况变化，常见的Component组件，Directive指令，Injector 注入，NgModule，NO*ERRORS*SCHEMA浅测试

```
import { Component, Directive, Injector, NgModule, OpaqueToken,NO_ERRORS_SCHEMA, Pipe,
PlatformRef, SchemaMetadata, Type } from '@angular/core';
```

2. 导入 async异步测试、ComponentFixture装置和TestBed测试床

```
import {async，ComponentFixture,TestBed }  from "@angular/core/testing"
```

3. 导入css选择器类By，元素类el，通过DOM测试时需要这些

```
import {By}   from '@angular/platform-browser'
import {el} from "@angular/platform-browser/testing/browser_util";
```

4. 导入Router, ActivatedRoute路由、HTTP、FormsModule表单类等

```
import {Router, ActivatedRoute} from "@angular/router";
import {Http, HttpModule, ConnectionBackend, BaseRequestOptions, XHRBackend} from "@angular/http";
import {Location} from '@angular/common';
import {FormsModule} from "@angular/forms";
import {fakeAsync, tick} from "@angular/core/testing/fake_async";
import {inject, getTestBed} from "@angular/core/testing";
import {MockBackend, MockConnection} from "@angular/http/testing/mock_backend";
```

   下面导入与被测对象相关的类

5. 实际被测组件类、服务类

```
import {ModifyDevComponent} from "./modifydevice.component";/*被测组件-这是个修改设备的组件*/
import {DeviceConfigService} from "../../../project/set/deviceconfig.service";/*这是个修改配置的服务*/
```

6. 被测对象打桩类

```
import {RouterStub} from "../../../../../UTStub/RouterStub";
import {ObservableStub} from "../../../../../UTStub/ObservableStub";
import {HTTPStub} from "../../../../../UTStub/HttpStub";
```

```
import {ActivatedRouteStub} from "../../../../../UTStub/ActivedRouteStub";
import {LocationStub} from "../../../../../UTStub/LocationStub";
```

二、在代码编写框架下配置测试环境

1. 按照jasmine的标准写法，先是测试项的描述describe部分

```
describe('XXX测试项的名称', () => {
/*实际测试代码*/
}
```

2. 配置测试条目item的环境配置

```
/*实际测试代码*/
TestBed.configureTestingModule({
    /*环境配置主体部分*/
})
```

> 测试代码的第一部分就是配置测试条目item的环境配置（类似用例设计中的预置条件），用jasmine的beforeEach() 或者 beforeAll()等函数中调用TestBed.configureTestingModule来配置测试的组件以及提供商等。

环境配置的主体内容

- declarations: [被测类声明],/*被测组件等对象与被测组件相关的类
- imports: [支撑类],/*测试中使用到的提供的支撑类*/
- schemas: [NO*ERRORS*SCHEMA],/*设置浅测试方式，后文描述*/
- providers: [] /*这是个数组，列出了提供者和提供的类，用于打桩类的重定位*/

```
declarations: [ModifyDevComponent,TurnWhiteDirective],
imports: [FormsModule,HttpModule],
schemas: [NO_ERRORS_SCHEMA],
providers:    [
{
    /*使用自定义的HTTPStub 来替换原有的Http 服务*/
    provide:Http,useClass:HTTPStub
},
{
    provide:ActivatedRoute,useClass:ActivatedRouteStub
},
]
```

> 黑色加粗字体为固定，不可修改的。
>
> beforeEach等jasmine函数的写法和作用，参见官网

- 如果还想再覆盖组件的服务来简化或模拟测试，还可以继续往下写

```
TestBed.configureTestingModule({
        declarations: [ModifyDevComponent,TurnWhiteDirective],
        imports: [FormsModule,HttpModule],
        schemas: [NO_ERRORS_SCHEMA],
        providers:    [
        {
            /*使用自定义的HTTPStub 来替换原有的Http 服务*/
            provide:Http,useClass:HTTPStub
        },
        {
            provide:ActivatedRoute,useClass:ActivatedRouteStub
        },
        ]
```

```
})
// override component's service begin
.overrideComponent(ModifyDevComponent,{
set:{
    providers:[ {provide:DeviceConfigService,useClass:DeviceConfigServiceOver} ]
}
```

至此，准备工作基本完成。

三、编写测试条目

准备工作完成后，就可以进行测试条目的编写和测试，it函数标识了测试条目，我们以“#组件-测试编号 测试主题 与测试结果”来命名，你也可以用自己习惯的命名方式。

1. 测试条目it函数

```
it('#modifydevice-0001 组件异步创建测试 async created  shoud be succcess',
async(() => {
    /*测试条目主体*/
}));
```

2. 通过TestBed.complieComponents编译

```
TestBed
       .compileComponents()   /*TestBed.complieComponents编译（异步）*/
       .then(() => {
});
```

3. 通过TestBed.createComponent等创建组件，createComponent的返回fixture

```
TestBed
.compileComponents()
.then(() => {
    let fixture = TestBed.createComponent(ModifyDevComponent);
});
```

4. 通过fixture获取组件或DOM的实例

```
TestBed
.compileComponents()
.then(() => {
    let fixture = TestBed.createComponent(ModifyDevComponent);
    modifydevicecompInst = fixture.componentInstance;
    let modifydevicecompDOMEl = fixture.debugElement.nativeElement;
});
```

5. 触发动作或者调用函数，将当前值与目标值进行断言校验

```
TestBed
.compileComponents()
.then(() => {
    let fixture = TestBed.createComponent(ModifyDevComponent);
    modifydevicecompInst = fixture.componentInstance;
    let modifydevicecompDOMEl = fixture.debugElement.nativeElement;
    expect(modifydevicecompInst).toBeTruthy();
    expect(modifydevicecompDOMEl.querySelectorAll('label')[0].textContent).toEqual('产品名称');/*通过css选择器获取DOM元素*/
});
```

恭喜你，可以开始编写代码，也意味着你进入了一次次掉坑、填坑的痛并快乐的旅途中。

后记：你以为我在讲测试？No，我其实是练习写md。缩进不友好啊。