2. Test on Component

测试component的时候，需要测试的是什么?

- 【组件渲染】当页面渲染好后，重要的、应该存在的控件存在，而且初始值正确
- 【组件绑定】当页面有数据绑定时候，当模拟属性赋值，视图是否能正确的绑定并跟随变化。反向的时候，当页面控件变化，绑定数据是否正确更新并获取
- 【组件动作】当页面控件有动作的时候，除了页面关联变化，绑定属性变化，是否还正确触发了其他后台动作（如service调用）

参考默认生成的测试案例

When generate component, by default it already come together with test case structure.

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';
import { DashboardPageComponent } from './dashboard-page.component';
import { RouterModule } from '@angular/router';

describe('DashboardPageComponent', () => {
  let component: DashboardPageComponent;
  let fixture: ComponentFixture<DashboardPageComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ DashboardPageComponent ],
      imports: [ RouterModule.forRoot([]) ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(DashboardPageComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it(`should have as title 'demo'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('demo');
  });

  it('should render title in a h1 tag', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('Welcome to demo!');
  });
});
```

It is the test target for html contain below tag

```
<div style="text-align:center">
<h1>
  Welcome to {{ title }}!
</h1>
```

And ts contain below property

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'demo';
}
```

当需要测试【渲染】/【绑定】/【动作】的时候，不可避免就是需要在页面上选择控件，从而获取里面的值作判断。

常见控件获取方式

```
//Via nativeElement
h1 = fixture.nativeElement.querySelector('h1');
h1_content = fixture.debugElement.nativeElement.querySelector('h1').textContent;
const button = fixture.debugElement.nativeElement.querySelector('button');
button.click();

//By.css
const bannerDe: DebugElement = fixture.debugElement;
const paragraphDe = bannerDe.query(By.css('p'));

//模拟输入->发布事件->侦测变化
const hostElement = fixture.nativeElement;
const nameInput: HTMLInputElement = hostElement.querySelector('input');
nameInput.value = 'quick BROWN  fOx';

nameInput.dispatchEvent(newEvent('input'));
fixture.detectChanges();
expect(nameDisplay.textContent).toBe('Quick Brown  Fox');
```

NativeElement: 原生DOM元素

- fixture.nativeElement 的值是 any 型別的。
- DebugElement.nativeElement 也同樣是 any 型別的。
- nativeElement 的屬性取決於執行環境。 你可以在沒有 DOM，或者其 DOM 模擬器無法支援全部 HTMLElement API 的平臺上執行這些測試。
- nativeElement后面可以接上querySelector/querySelectorAll

DebugElement: 由Angular封装的抽象层

> Angular 依賴於 DebugElement 這個抽象層，就可以安全的橫跨其支援的所有平臺。 Angular 不再建立 HTML 元素樹，而是建立 DebugElement 樹，其中包裹著相應執行平臺上的原生元素。 nativeElement 屬性會解開 DebugElement，並返回平臺相關的元素物件。

- debugElement后面可以接上query