How do I test nativeElement.focus in Angular?

Posted At : January 28, 2020 9:00 AM | Posted By : Jeffry Houser
Related Categories: Professional, Jasmine, Angular, Karma

In last week's post I wrote about setting focus on an element with Angular.

This week I wanted to write tests behind that code.

A Review

First, let's review what we want to test. We have a loop of inputs in the HTML Template and access them in the component class using the ViewChildren() metadata:

```
@ViewChildren('input') inputs: QueryList<ElementRef>;
```

Then we have an onClick() method which accepts an input and determines which element to set focus too:

```
onClick(index) {
    this.inputs.toArray()[index].nativeElement.focus();
  }
```

As a good developer, I of course, want to write tests against that onClick() method.

Write the Tests

If you're building off last week's project, you can open up the app.component.spec.ts file. You probably already have a beforeEach() which creates the basic application. Let's add a new describe block to the file:

```
describe('onClick()', () => {
});
```

Good start, let's add some variables:

```
let app;
const e1: ElementRef = new ElementRef({focus() {} } );
const e2: ElementRef = new ElementRef({focus() {} } );
```

I created one app variable to reference the component, and two ElementRef constants. When creating an ElementRef we pass in a nativeElement object. For the purposes of this test, I created a mock object with a focus() method. In production code I would have formalized a mock class to represent a nativeElement.

Now, let's add a beforeEach:

```
beforeEach(() => {
});
</code>
```

Nothing interesting here. Inside the beforeEach() let's get access to the app:

```
<code
const fixture = TestBed.createComponent(AppComponent);
app = fixture.debugElement.componentInstance;
```

This gets access to the fixture and saves the app for future use. Now detectChanges():

```
fixture.detectChanges();
```

This forces the component to render once, which will create all the viewChildren and populate the inputs variable.

Now, add spys for the focus method on our custom elementRef objects:

```
spyOn(e1.nativeElement, 'focus');
spyOn(e2.nativeElement, 'focus');
```

Our code will check on these spys to make sure the appropriate element was focused on based on our inputs in the onClick() method.

I want to do one more spy:

```
spyOn(app.inputs, 'toArray').and.returnValue([e1, e2]);
```

When our onClick() method tries to convert the inputs into an array, this gives us complete control over what the onClick() method is accessing. Even though the elements in the view are properly rendered, using this spy with controlled return values gives us granular control over the test.

Now let's write a test:

```
it('should call focus on element 0', () => {
  app.onClick(0);
  expect(e1.nativeElement.focus).toHaveBeenCalled();
  expect(e2.nativeElement.focus).not.toHaveBeenCalled();
});
```

This call s the onClick() method with a 0 index. It verifies that focus was called on e1, but not on e2.

We can write a second test using a different index value:

```
it('should call focus on element 1', () => {
  app.onClick(1);
  expect(e1.nativeElement.focus).not.toHaveBeenCalled();
  expect(e2.nativeElement.focus).toHaveBeenCalled();
});
```

This calls the same onClick() method, with a different index, and swaps the checks making sure that focus() was called on the e2 element and not the e1 element.

Run the code:

Success!

That is one way to test how to set focus on an element using Angular.

Play with the code here