RxJS: Testing with Fake Time

Nicholas Jamieson Follow
Jun 27, 2018 · 3 min read

Photo by rawpixel on Unsplash

Angular, Jasmine, Jest and Sinon.JS all provide APIs for running tests with fake time. Their APIs differ, but they are broadly similar:

- Before each test, time-related functions like `setTimeout` and `setInterval` are patched to use fake time instead of actual time.
- Within a test, an API call can be made to advance the clock by a specified number of fake milliseconds.
- After each test, the patched functions are restored.

Running tests with fake time avoids having to wait for actual time to elapse and it also makes the tests much simpler, as they run synchronously.

So what does this have to do with RxJS?

RxJS has its own concept of fake time—which is named virtual time. In RxJS, all time-related functionality is implemented in schedulers and there is a particular scheduler for virtual time: the `VirtualTimeScheduler`.

To test RxJS-based code with virtual time, any schedulers used—either explicitly or implicitly—need to be swapped for an instance of the `VirtualTimeScheduler`.

Unfortunately, that's not always easy to do. And using the `VirtualTimeScheduler` won't help if the code under test also includes time-related, non-RxJS code, as the virtual and fake time concepts differ significantly.

To solve this problem, I've added a `fakeSchedulers` function to `rxjs-marbles`, so that tests can use fake time for situations in which a marble test would be too complicated to write.

Let's have a look at how it can be used.

Testing an Angular component

Here's a simple Angular component that uses Reactive forms:

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms
import { Observable } from 'rxjs';
import { debounceTime, distinctUntilChanged, pluck } f

@Component({
  selector: 'some-component',
  template: `
  <form [formGroup]="form">
    <input formControlName="term" type="text">
  </form>
  <div class="searching" *ngIf="term$ | async as term"
    <span>Searching for {{ term }}</span>
  </div>
  `
})
export class SomeComponent {
  form: FormGroup;
  term$: Observable<string>;
  constructor(formBuilder: FormBuilder) {
```

Whenever the search term's `input` changes, the form's value is debounced and repeated values are ignored. If the resultant search term isn't an empty string, the searching indicator is shown. The component doesn’t do anything useful; it does just enough to give us something to test.

We could test that the searching indicator exhibits the expected behaviour with a test something like this:

```
import { TestBed, async } from '@angular/core/testing'
import { FormsModule, ReactiveFormsModule } from '@ang
import { fakeSchedulers } from 'rxjs-marbles/jasmine/a
import { SomeComponent } from './some.component';

describe("SomeComponent", () => {

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [SomeComponent],
      imports: [FormsModule, ReactiveFormsModule]
    }).compileComponents();
  }));

  it('should indicate when searching', fakeSchedulers(
    const fixture = TestBed.createComponent(SomeCompon
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElemen
    const input = compiled.querySelector('input');
    expect(compiled.querySelector('.searching')).toBeN
```

Internally, `fakeSchedulers` calls Angular’s `fakeAsync`, so fake time is advanced the same way: by calling `tick`.

The above test triggers the form’s `valueChanges` by dispatching an event. It could also trigger the change using an explicit call to the form’s `patchValue` method, like this:

```
it('should indicate when searching', fakeSchedulers(()
  const fixture = TestBed.createComponent(SomeComponen
  fixture.detectChanges();
  const compiled = fixture.debugElement.nativeElement;
  const input = compiled.querySelector('input');
  expect(compiled.querySelector('.searching')).toBeNul
  fixture.componentInstance.form.patchValue({ term: 'f
  fixture.detectChanges();
  expect(compiled.querySelector('.searching')).toBeNul
  tick(400);
```

Testing a React component

Here’s a React version of the Angular component:

```
import * as React from "react";
import { componentFromStream, createEventHandler } fro
import { from } from "rxjs";
import { debounceTime, distinctUntilChanged, map, star

export const SomeComponent = () => {
  const { handler, stream } = createEventHandler();
  const SearchingComponent = componentFromStream(props
    map(event => event.target.value),
    debounceTime(400),
    distinctUntilChanged(),
    startWith(""),
    map(term => !term ? null : (
      <div className="searching">
        <span>Searching for {term}</span>
      </div>
```

It uses the `createEventHandler` and `componentFromStream` functions from `recompose` to compose an observable-based component that emits DOM elements when the `input` changes.

If you’ve not seen how `recompose` can be used to compose observable-based components in React, this talk by Andrew Clark is well worth watching.

To test the component with fake time, we could do something like this:

```
import { mount } from "enzyme";
import * as React from "react";
import { fakeSchedulers } from "rxjs-marbles/jest";
import { SomeComponent } from "./SomeComponent";

describe("SomeComponent", () => {

  beforeEach(() => jest.useFakeTimers());

  it("should indicate when searching", fakeSchedulers(
    const wrapper = mount(<SomeComponent />);
    expect(wrapper.find('.searching')).toHaveLength(0)
    wrapper.find("input").simulate("change", { target:
```

Unlike Angular, Jasmine and Sinon.JS, Jest does not patch `Date`. In particular, it does not patch `Date.now`.

That means `fakeSchedulers` needs to keep track of the current fake time—as the RxJS scheduler implementations depend upon `Date.now`. To do this, fake time needs to be advanced by calling the `advance` function that's passed to the test, instead of `jest.advanceTimersByTime`.

Testing with AVA, Mocha or Tape

These test frameworks don't include built-in support for testing with fake time, but Sinon.JS supports it and it's easy to use.

For example, this is what testing with fake time using Sinon.JS looks like in Mocha:

```
import { expect } from "chai";
import { fakeSchedulers } from "rxjs-marbles/mocha";
import { timer } from "rxjs";
import * as sinon from "sinon";

describe("timer", () => {

  let clock: sinon.SinonFakeTimers;

  beforeEach(() => {
    clock = sinon.useFakeTimers();
  });

  it("should be testable with fake time", fakeSchedule
    let received: number | undefined;
    timer(100).subscribe(value => received = value);
    clock.tick(50);
```

· · ·

After writing this article, a related PR was merged into the RxJS repository. The PR fixes the one problem that prevented the RxJS schedulers from working with Angular's `fakeAsync`. The problem was that RxJS captured `Date.now` before it could be patched by `fakeAsync`.

So with RxJS versions later than 6.2.1, `fakeSchedulers` should not be required for Angular tests—just use `fakeAsync`, instead. However, `fakeSchedulers` will still be necessary for any non-Angular tests run using Jasmine and for any tests run using other frameworks, when fake time is needed.