Testing Frameworks: Comparison among QUnit, Jasmine and Mocha

Kejun Liu

Because of the market pressure, the testing of Web-based applications is often neglected by developers, as it consumes a lot of time and lacks in significance [1]. This overlook will definitely lead to the decrease of application quality. Besides, making one modification in the application may cause big trouble to the whole system, Therefore, test-driven-development has been popular as it can reduce the ‘fear’ of breaking the whole system. When a change is made in the application, all that must be done is to run the existing test cases to see if the change has influenced any other part of the application [2]. There are many testing frameworks now in web development, such as QUnit, Jasmine and Mocha. In this paper, comparisons will be made among these three testing frameworks in unit test.

1. Introduction

1.1 QUnit

QUnit is the oldest among these three testing frameworks, which is released in 2008. QUnit was originally developed by John Resig as part of jQuery and in 2008, it was extracted from the jQuery but it does not rely on jQuery.

QUnit is a rather easy-to-use framework. It is a library like bootstrap which you can include it from CDN or download the qunit.js and qunit.css file from website and add it to a html file. Test cases should be included in a script tag or a JavaScript file. The testing result will be displayed in a pretty nice looking format in the website through opening the html file.

QUnit provides asynchronous test via QUnit.asyncTest(), QUnit.start() and QUnit.stop(). The test runner will automatically stop when meet QUnit.ayncTest() and it will not be invoked until QUnit.start() is called. A test can be stopped using the method QUnit.stop(). To achieve testing asynchronously, QUnit.start() should be called before each asynchronous function and QUnit.stop() should be called after it.

1.2 Jasmine

Jasmine is released in 2010 and it’s built to be easy to set up and use in almost any scenario. It has a standalone release so in this case it does not depend on other frameworks. That means jasmine library is all what developers need to study. Besides, jasmine support node.js, ruby, python as well so it can be run through karma or other runners. Jasmine has lots of extensionalities – 3th party libraries like jasmine-jquery, jasmine-dom, jasmine-react etc. It does not rely on any browsers, DOM, or JavaScript frameworks. Therefore, it’s suited for websites, Node.js projects, or anywhere JavaScript can run on.

1.3 Mocha

Mocha relies on third party libraries. It requires developers to select and set up assertion libraries and mocking utilities, such as Chai and Sinon. Although it is a tough work to choose appropriate libraries in the beginning, mocha will make testing easier because developers can choose libraries that they are familiar with and libraries with better syntax and functionality, which will make the testing process more fluent and less troublesome.

2. Comparison

In this part, comparison among QUnit, Jasmine and Mocha will be made in following aspects: speed of setup, functionality, flexibility and simplicity.

2.1 Speed of Setup

QUnit is perhaps easiest to start from scratch among these three frameworks which only requires including a JavaScript and CSS file in the html file. It only takes few hours to master its usage. The API is simple to read and it does not have fancy but complex functions like the promise function in Mocha.

Jasmine can also be set up like QUnit for It has standalone release. It contains everything needed to do unit test including assertion, asynchronous testing, built in mocking functionality called spy. To test with Jasmine, developers only need to download particular version and replace the source/spec files with their own test and source code. Besides, node.js, ruby and python are supported in Jasmine which provides developers flexible methods to set up testing environment as they want.

Compared with QUnit and Jasmine, Mocha is quite difficult to set up and study for the fact that Mocha requires developers to select and set up third party libraries such as assertion libraries and mocking utilities themselves. Thus, for a beginner, it is hard to choose which libraries to use and learn. Besides, test runners are required to work along with Mocha in testing, which indicates additional environment setup is mandatory for testing with Mocha.

Thus, it is obvious that Mocha’s environment is much more complicated to set up compared with QUnit and Jasmine. QUnit is the easiest one among these three.

2.2 Functionality

All these three frameworks have good functionality in supporting almost all JavaScript unit tests. Whereas there are a lot of differences in their implementation, which leads to their various performance. Differences can be classified into three main parts: assertions, mocking and asynchronous testing.

2.2.1 Assertions

The differences of these libraries start with assertions. QUnit and Jasmine both have built in assertion libraries. Mocha does not come with a built-in assertion library. Where developers can use QUnit and Jasmine as is to write their tests, they have to load another assertion library with Mocha [3]. Although QUnit has built in simple assertion, it appears that it does not clear syntax like Jasmine and Mocha when developers need to group related tests under a single label.

2.2.2 Mocking

In terms of mocking, Jasmine has its built-in mocking functionality, which is spy to be specifically. While both Mocha and QUnit do not come with built in mocking. Third party library such as Sinon is required when developers want to use mocking/spies with Mocha or QUnit. Sinon is a very powerful library which is equivalent to Jasmine spies with some additional features.

Both Sinon and Jasmine-spies support mocking but in different levels. Sinon has a very nice API for a mock server which is quite handy if developers want to test REST calls without having a real backend. This allows us to setup fake responses to AJAX requests made for certain URLs. While Jasmine-spies are mostly function-level spying. Fake server functionality could be implemented through Jasmine-spies but a little bit complicated [4]. Thus, Sinon is obviously a better tool used to mock a fake server for front-end testing when back-end is still under development.

2.2.3 Asynchronous

Asynchronous tests will be useful when there is Ajax calls. Jasmine and Mocha both have the same approach for asynchronous testing using a callback method (done) [5]. Instead of the callback method, QUnit also has its own asynchronous support through functions QUnit.start() and Qunit.stop(). Although different in syntax, all three frameworks can equally satisfy almost all asynchronous testing scenarios.

3. Conclusion

QUnit, Jasmine and Mocha all have good performance in unit testing despite the fact that they are quite different from setup to detailed functionality. Jasmine is a more complete all-in-one solution, containing an assertion library with custom methods, callback spies support. Mocha has the opposite approach: basic functionalities built-in and extensibility through plugins [6]. While QUnit might not be the shiniest of these testing frameworks, it works perfectly and requires almost no setup [7]. Mocha has more flexibility, but it requires more complex set up and additional dependencies.

In our rice book application unit testing, Mocha is chosen along with Chai, Sinon. I think it is a brilliant choice. Firstly, although it is known that Mocha is a headache to set up, our node.js environment has already been set up which means we can use Mocha based on previous node.js environment, no additional configuration is required. Secondly, the front-end and back-end are separately developed. In this case, as I mentioned above, Sinon is a better choice for testing front-end here as it can provide us with a fake server. Lastly, Mocha is more flexible, which is what we need in rice book testing. For testing React component rendering, we imported enzyme, react-addons-test-utils, react-dom. For mocking, we used mockery. For automating browser testing, we used selenium. Although a lot of work is needed to learn these libraries, the convenience these libraries provide us should not be ignored.

As simplicity and flexibility cannot be achieved together, making decision which testing framework to use depends on web functionality and testing requirement. Of course, the decision largely depends on other personal preferences, such as syntax and running environment.

Reference

[1]. Dutta P, Verma A, Prithviraj J. Karma-The Test Runner, for Automated Testing of Web Based Applications[J].
[2]. http://www.base36.com/2012/07/benefits-of-test-driven-development/
[3]. http://marcofranssen.nl/jasmine-vs-mocha/index.html
[4]. http://stackoverflow.com/questions/12216053/whats-the-advantage-of-using-sinon-js-over-jasmines-built-in-spys
[5]. https://marcofranssen.nl/jasmine-vs-mocha/
[6]. http://stackoverflow.com/questions/24391462/what-are-the-differences-between-mocha-chai-karma-jasmine-should-js-etc-te/24404675
[7]. http://blog.trackets.com/2014/09/24/unit-testing-javascript-with-qunit.html