1

2

3

4

5

6

Seminar - Jasmine 2.0.0

https://jasmineseminar-calculator.herokuapp.com/

Danh sách sinh viên

1312582	Bùi Phạm Thiên Thư
1312595	Trịnh Xuân Tiến
1312596	Dương Tiễn
1312600	Nguyễn Hoàng Tín

1

2

3

4

5

6

Nội dung trình bày

- Tổng quan
- Nội dung tìm hiểu
  - Unit test
  - Jasmine
  - Demo

1

2

3

4

5

6

A. TỔNG QUAN

1

2

3

4

5

6

1. Chức năng đã làm

- Tính toán biểu thức liên quan đến +, -, *, /, !, log, ln, ^, căn bậc 2, lượng giác
- Chuyển đổi cơ số Bin, Hex, Dec, Oct
- Xuất thông báo lỗi với nhiều loại lỗi khác nhau (lỗi syntax, lỗi tràn số, lỗi cơ số)
- Xóa tất cả ký tự trong biểu thức
- Xóa ký tự cuối của biểu thức
- Sử dụng bàn phím để nhập các ký tự số, chữ (A đến F), tính toán kết quả

1
2
3
4
5
6

2. Những việc đã làm có sử dụng Jasmine

- [x] Viết 1 số test case kiểm tra giá trị biên
- [x] Viết test case kiểm tra một số biểu thức tính toán
- [x] Viết custom matcher để so sánh 2 số thực
- [x] Sử dụng kỹ thuật spy để kiểm tra các loại lỗi
- [x] Sử dụng kỹ thuật spy để giả lập 1 hàm chuyển đổi cơ số

1

2

3

4

5

6

B. NỘI DUNG TÌM HIỂU

3. Unit Test

- Được thực hiện bởi **Developer**.
- Mục đích là để chứng minh tính đúng đắn của các hàm.
- Các hoạt động **Unit Test**:
  - Bảo đảm contract.
  - Kiểm tra kết quả truyền dữ liệu.
  - Kiểm tra kết quả tính toán.
  - Bảo đảm lỗi bên ngoài được xử lý một cách chính xác.
- Một số khái niệm cơ bản:
  - *Test case*: Dùng để chỉ một trường hợp hoạt động của function.
  - *Setup*: Dùng để cài đặt các biến, khai báo dữ liệu trước khi chạy các test case.
  - *Teardown*: Dùng để giải phóng bộ nhớ sau khi các test case đã được thực hiện xong.
  - *Assert*: Câu lệnh dùng để kiểm tra tính đúng đắn của hàm.
  - *Mock*: Giả sử, trong chương trình, có 2 module A & B. Ta cần module A để test module B, nhưng module A chưa được xây dựng hoàn chỉnh. Mock chính là kỹ thuật giúp chúng ta giả lập module A, đủ để test module B, không cần phải đợi module A hoàn chỉnh.

4. Jasmine

- Là một framework dùng để thực hiện kiểm thử đơn vị trên những đoạn code JavaScript.
- Được xây dựng theo mô hình BDD.

4. Jasmine

4.1. Các khái niệm

Spec Suite

- Một bộ kịch bản test, bao gồm nhiều hàm test case.
- Jasmine sử dụng hàm `describe()` để định nghĩa 1 Spec Suite. Trong 1 Spec Suite, có thể có chứa những Spec Suite khác.
- Nếu chúng ta không muốn xem kết quả của 1 Spec Suite, ta thay lệnh `describe()` bằng lệnh `xdescribe()`.

4. Jasmine

4.1. Các khái niệm

Spec

- Trong Jasmine, một hàm kiểm thử được gọi là Spec. Spec nằm trong Spec Suite.
- Để định nghĩa 1 Spec, Jasmine sử dụng hàm `it()`.
- Trong nhiều trường hợ, test case có kết quả trả về chưa được xác định đầy đủ.
  Lúc này, ta có thể thay lệnh `it()` bằng câu lệnh `xit()` hoặc sử dụng thêm câu lệnh `pending();`

4. Jasmine

4.1. Các khái niệm

Expectation

- Dùng để chỉ kết quả mong đợi đối với một hàm kiểm thử.
- Trong Jasmine, hàm `expect()` dùng để mô tả kết quả mong đợi này.

4. Jasmine

4.1. Các khái niệm

Hàm Setup và Teardown

- Các hàm Setup:
  - `beforeEach()`: được chạy trước khi chạy **MỖI** test case.
  - `beforeAll()`: được chạy trước khi **TẤT CẢ** các test case được chạy.
- Các hàm Teardown:
  - `afterEach()`: được chạy sau khi chạy **MỖI** test case.
  - `afterAll()`: được chạy sau khi **TẤT CẢ** các test case được chạy.

4. Jasmine

4.1. Các khái niệm

Matcher

- So sánh kết quả mong đợi với kết quả thực sự của một test case.
- Cú pháp khẳng định:
  ```
  expect(expected_value).<matcher_function>
  ```
- Cú pháp phủ định:
  ```
  expect(expected_value).not.<matcher_function>
  ```

4. Jasmine

4.1. Các khái niệm

DANH SÁCH CÁC MATCHER MÀ JASMINE 2.0.0 HỖ TRỢ

1

2

3

4

5

6

toEqual(value)

toBe(object)

toContain(array)

toBeCloseTo(decimal-number, position)

toThrow( )

toThrowError(errorName)

toBeLessThan(decimalNumber)

toBeGreaterThan(decimalNumber)

toBeDefined( )

toBeUndefined( )

toBeNull( )

toBeFalsy( )

toBeTruthy( )

toEqual(value)

- So sánh giá trị expected_value có bằng với giá trị value hay không.
- Matcher này có thể so sánh 2 biến không cùng địa chỉ.

```
describe("matcher toEqual", function()
{
    it("khác địa chỉ", function()
        {
            var object1 = new Object();
            var object2 = new Object();
            expect (object1).toEqual(object2);
        });

    it("cùng địa chỉ", function()
        {
            var object1 = new Object();
            var object2 = object1;
            expect (object1).toEqual(object2);
        });

    it("cùng kiểu dữ liệu, cùng giá trị", function()
        {
            var object1 = new Object();
            var object2 = new Object();
            object1 = 1;
            object2 = 1;
            expect (object1).toEqual(object2);
        });

    it("cùng kiểu dữ liệu, khác giá trị", function()
        {
            var object1 = new Object();
            var object2 = new Object();
            object1 = 1;
            object2 = 2;
            expect (object1).not.toEqual(object2);
        });

    it("khác kiểu dữ liệu", function()
        {
            var object1 = new Object();
            var object2 = new Object();
            object1 = 1;
            object2 = "1";
            expect (object1).not.toEqual(object2);
        });
});
```

toBe(object)

- So sánh giá trị expected_value có và giá trị object có cùng là 1 đối tượng hay không.

```
describe("matcher toBe", function()
{
    it("khác địa chỉ", function()
        {
            var object1 = new Object();
            var object2 = new Object();
            expect (object1).not.toBe(object2);
        });

    it("cùng kiểu dữ liệu nguyên", function()
        {
            var object1 = new Object();
            object1 = 1;
            object2 = 1;
            expect (object1).toBe(object2);
        });


    it("cùng địa chỉ", function()
        {
            var object1 = new Object();
            var object2 = new Object();
            object1 = object2;
            expect (object1).toBe(object2);
        });
});
```

toContain(array)

- Kiểm tra phần tử expected_value có nằm trong mảng array hay không.

```
describe("matcher toContain", function()
{
    it("simple items", function()
        {
            var object1 = [1,2,3];
            var object2 = 1;
            expect (object1).toContain(object2);
        });

    it("flexible items", function()
        {
            var object1 = [{str:"str", int:"int"}, 1];
            var object2 = {str:"str", int:"int"};
            expect (object1).toContain(object2);
        });
});
```

toBeCloseTo(decimal-number, position)

- Kiểm tra số thực expected_value có bằng với số decimal_number ở vị trí position hay không.

```
describe("matcher toBeCloseTo", function()
    {
        it("vị trí <= 0", function()
            {

                expect (12.34).toBeCloseTo(12.3, 0);
                expect (12.34).toBeCloseTo(12.3, -100);
            });

        it("giống nhau", function()
            {
                expect (12.34).toBeCloseTo(12.3, 1);
            });

        it("khác nhau", function()
            {
                expect (12.34).not.toBeCloseTo(12.3, 2);
            });
    });
```

Custom Matcher

- Ngoài những matcher do **Jasmine** cung cấp, chúng ta cũng có thể tạo ra matcher cho riêng mình.

```
var compareNumber =
  {
    toBeGreaterThanOrEqual: function() {

      return {
        compare: function(firstNumber, secondNumber)
        {
          var result = {};

          if (typeof(firstNumber) !== typeof (secondNumber))
              result.message = firstNumber + " and  " + secondNumber + " do not have the same data type";
          else if ((typeof(firstNumber) !== "number"))
              result.message = firstNumber + " and " + secondNumber + " are not number";

          var condition = ((typeof(firstNumber) === "number") && (typeof (secondNumber) === "number"));
          if (condtion ===true)
              condition = (firstNumber >= secondNumber);
              result.pass = condition;

          return result;
        }
      };
    }
  };
```

Custom Matcher

- Ngoài những matcher do **Jasmine** cung cấp, chúng ta cũng có thể tạo ra matcher cho riêng mình.

```
describe ("test custom matcher", function()
  {
      describe ("compare number: toBeGreaterThanOrEqual", function ()
      {
          beforeEach(function()
          {
              jasmine.addMatchers(compareNumber);
          });

          it("2 > -2: pass", function()
          {
              expect(2).toBeGreaterThanOrEqual(-2);
          });

          it("2 > -2: do not have the same data type", function()
          {
              expect("2").not.toBeGreaterThanOrEqual(-2);
          });

          it("2 > -2: are not number", function()
          {
              expect("2").not.toBeGreaterThanOrEqual("-2");
          });
      });
  });
```

4. Jasmine

4.1. Các khái niệm

Spy&Mock:

- Spy: những hàm (spy) do **Jasmine** cung cấp, có khả năng chặn 1 hàm (function) và theo vết lời gọi đến các function do lập trình viên xây dựng.
- Cú pháp tạo spy:

  spyOn(object, methodName),

  jasmine.createSpy(newMethodName),

  jasmine.createSpyObj(arrayOfString)

- Matcher

  toHaveBeenCalled(): Kiểm tra spy có được gọi hay chưa.

  toHaveBeenCalledWith(arguments): Kiểm tra spy với danh sách tham số arguments có được gọi hay chưa

```
describe("Tạo spy", function()
{
    it("spyOn", function()
    {
        function Calculator()
        {
            this.compute = function(expression)
            {
                return expression;
            };
        }

        var cal = new  Calculator();
        spyOn (cal, 'compute');
        cal.compute("1+2");
        expect (cal.compute).toHaveBeenCalled();
    });

    it ("createSpy", function ()
    {
        var cal = jasmine.createSpy('convertHEX2DEC');
        cal("1+2");
        expect (cal).toHaveBeenCalledWith("1+2");
    });
```

```
it ("createSpyObj", function()
    {
        var cal = jasmine.createSpyObj('version', ['add',
                                        'substract']);

        cal.add(1,2);
        expect(cal.add).toHaveBeenCalledWith(1,2);
        expect(cal.substract).not.toHaveBeenCalled();
    });

    it("Chưa gọi spy", function()
    {
        function Calculator()
        {
            this.compute = function(expression)
            {
                return expression;
            };
        }

        var cal = new  Calculator();
        spyOn (cal, 'compute');
        expect (cal.compute).not.toHaveBeenCalled();
    });
});
```

1

2

3

4

5

6

Spy&Mock:

Các hàm spy:

Cú pháp: expect(...).<matcher_spy>.<spy_function>.

- .and.callThrough(): Bỏ chặn hàm.
- .and.returnValue(some_value): Tất cả các lời gọi hàm sẽ trả về giá trị some_value.
- .and.callFake(fake_function): Tất cả các lời gọi spy sẽ được ủy thác cho hàm fake_function.
- .and.throwError(message): Tất cả các lời gọi spy sẽ văng ra lỗi message.
- .and.stub(): Chặn hàm

```
describe ("Dùng spy", function()
{
    function Calculator()
    {
        this.compute = function(expression)
        {
            return expression;
        };
    }

    it("chặn hàm", function()
    {
        var cal = new Calculator();
        spyOn(cal, 'compute');
        var result = cal.compute("1+2");
        expect(result).toBeUndefined();
    });

    it("callThrough", function()
    {
        var cal = new Calculator();
        spyOn(cal, 'compute').and.callThrough();
        var result = cal.compute("1+2");
        expect(result).toEqual("1+2");
    });
```

```
it("returnValue", function()
   {
       var cal = new Calculator();

       spyOn(cal, 'compute').and.returnValue(0);
       var result = cal.compute("1+2");

       expect(result).toEqual(0);
   });

   it("callFake", function()
   {
       var cal = new Calculator();
       spyOn(cal, 'compute').and.callFake(function()
       {
           //...
           if (true === true)
               return "I am a calculator";
       });

       var result = cal.compute("1+2");

       expect(result).toEqual("I am a calculator");
   });
```

```
//spec này sẽ thông báo lỗi, bất chấp kết quả của hàm expect
    it("throwError", function()
    {
        var cal = new Calculator();
        spyOn(cal, 'compute').and.throwError("Xảy ra lỗi");

        expect(cal.compute).toThrow();
        expect(cal.compute).toThrowError("Xảy ra lỗi");

        var result = cal.compute("1+2"); // văng lỗi tại dòng lệnh
này do câu lệnh throwError ở trên
    });

    it("chặn hàm, dùng stub", function()
    {
        var cal = new Calculator();

        spyOn(cal, 'compute').and.callThrough();
        var result = cal.compute("1+2");
        expect(result).toEqual("1+2");

        cal.compute.and.stub();
        var result = cal.compute("1+2");
        expect(result).toBeUndefined();
    });
});
```

Spy & Mock:

Các thuộc tính theo vết khác

Mỗi lời gọi spy sẽ được theo vết và được thể hiện thông qua thuộc tính `calls`

Cú pháp: expect(...).<matcher_spy>.<spy_function>.

- .calls.any(): Trả về true nếu có ít nhất 1 lời gọi spy được thực thi, ngược lại thì trả về false.
- .calls.count(): Trả về số lần mà một spy được gọi.
- .calls.argsFor(index): Trả về danh sách tham số mà lời gọi spy thứ index được gọi.
- .calls.allArgs(): Trả về danh sách tham số của tất cả các lời gọi spy.
- .calls.mostRecent(): Trả về context và danh sách tham số của lời gọi spy gần nhất.
- .calls.first(): Trả về context và danh sách tham số của lời gọi spy đầu tiên.
- .calls.all(): Trả về context và danh sách tham số của tất cả các lời gọi spy.
- .calls.reset(): Xóa lịch sử truy vết đối với một spy.

```
describe ("Thuộc tính calls", function()
{
    function Calculator()
    {
        this.add = function(a, b)
        {
            return a+b;
        };

        this.substract = function(a,b)
        {
            return a - b;
        };
    }

    it(".calls.any() & .calls.count()", function()
    {
        var cal = new Calculator();
        spyOn(cal, 'add');
        spyOn(cal, 'substract');

        cal.add(1, 2);

        expect(cal.add.calls.any()).toEqual(true);
        expect(cal.substract.calls.any()).toEqual(false);

        expect(cal.add.calls.count()).toEqual(1);
        expect(cal.substract.calls.count()).toEqual(0);
    });
```

```
it(".calls.argsFor() & .calls.allArgs()", function()
    {
        var cal = new Calculator();
        spyOn(cal, 'add');
        spyOn(cal, 'substract');

        cal.add(1, 2);
        cal.add(3, 0);

        cal.substract(3, 0);
        cal.substract(1, 2);

        expect(cal.add.calls.argsFor(0)).toEqual([1, 2]);
        expect(cal.add.calls.argsFor(0)).not.toEqual([2, 1]);

        expect(cal.add.calls.argsFor(1)).toEqual([3, 0]);

        expect(cal.substract.calls.argsFor(0)).toEqual([3, 0]);
        expect(cal.substract.calls.argsFor(1)).toEqual([1, 2]);

        expect(cal.add.calls.allArgs()).toEqual([[1, 2], [3, 0]]);
        expect(cal.substract.calls.allArgs()).toEqual([[3, 0], [1, 2]]);
    });
```

```
it(".calls.first(), .calls.mostRecent() & .calls.all()", function()
    {
        var cal = new Calculator();
        spyOn(cal, 'add');

        cal.add(1, 2);
        cal.add(3, 0);

        expect(cal.add.calls.first()).toEqual({ object:cal, args: [ 1, 2
], returnValue: undefined});
        expect(cal.add.calls.mostRecent()).toEqual({ object:cal, args: [
3, 0 ], returnValue: undefined});
        expect(cal.add.calls.all()).toEqual([{ object:cal, args: [ 1, 2
], returnValue: undefined}, {object:cal, args: [ 3, 0 ], returnValue:
undefined}]);
    });

    it(".calls.reset()", function()
    {
        var cal = new Calculator();
        spyOn(cal, 'add');

        cal.add(1, 2);
        expect(cal.add.calls.count()).toEqual(1);

        cal.add.calls.reset();
        expect(cal.add.calls.count()).toEqual(0);
    });
});
```

1

2

3

4

5

6

Spy & Mock:

Mock: Chúng ta có thể dùng spy để giả lập module.

```
function Calculator()
{
    this.compute = function(expression)
    {
        //...
    };

    this.convertHEX2DEC = function(src, dest)
    {
        //...
    };
}
describe ("Mock: biểu thức không hợp lệ", function ()
{
    var cal = new Calculator();

    it("() 7*9=-*", function(){
        spyOn(cal, 'compute').and.throwError("Lỗi cú pháp");
        var result = cal.compute("() 7*9=-*")
        expect(result).toThrow();
    });

    it("(5+3 = error", function(){
        spyOn(cal, 'compute').and.throwError("Lỗi cú pháp");
        var result = cal.compute("(5+3");
        expect(result).toThrow();
    });

    it("5+3) = error", function(){
        spyOn(cal, 'compute').and.throwError("Lỗi cú pháp");
        var result = cal.compute("5+3)");
        expect(result).toThrow();
    });
});
```

```
describe ("Mock: tính chuyển đổi cơ số", function ()
{
    var cal = new Calculator();

    it("convertHEX2DEC", function(){
        var cal = new Calculator();
        spyOn(cal, 'convertHEX2DEC').and.callFake(function(){return
        15}); //tạo spy
        var result = cal.convertHEX2DEC("E", 15);
        expect(result).toBe(15);

    });
});

describe("Tính giá trị hằng số", function()
{

        it("find pi", function()
        {
            var cal = new Calculator();
            spyOn(cal, 'compute').and.returnValue(3.14); //tạo spy
            var result = cal.compute("pi");
            expect(result).toBe(3.14);
        });
});
```

4. Jasmine

4.2. Xử lý bất đồng bộ

Jasmine cung cấp hàm `setTimeout` và `setInterval` để giả lập những hàm liên quan đến thời gian phản hồi.

```
describe("Manually ticking the Jasmine Clock", function() {
          var timerCallback;

          beforeEach(function() {
            timerCallback = jasmine.createSpy("timerCallback");
            jasmine.clock().install();
          });

          afterEach(function() {
            jasmine.clock().uninstall();
          });

          it("causes a timeout to be called synchronously",
function() {
            setTimeout(function() {
              timerCallback();
            }, 100);

            expect(timerCallback).not.toHaveBeenCalled();

            jasmine.clock().tick(101);

            expect(timerCallback).toHaveBeenCalled();
          });
```

```
it("causes an interval to be called synchronously", function() {
            setInterval(function() {
              timerCallback();
            }, 100);

            expect(timerCallback).not.toHaveBeenCalled();

            jasmine.clock().tick(101);
            expect(timerCallback.calls.count()).toEqual(1);

            jasmine.clock().tick(50);
            expect(timerCallback.calls.count()).toEqual(1);

            jasmine.clock().tick(50);
            expect(timerCallback.calls.count()).toEqual(2);
          });
        });
```

4. Jasmine

4.1. Xử lý bất đồng bộ

Xét đoạn mã sau:

```
var flag = false;

function testAsync() {

// đợi 2 giây, sau đó gán flag = true
    setTimeout(function () {
        flag = true;
    }, 2000);
}

describe("Testing async calls", function () {

    it("Should be true if the async call has completed", function () {
        expect(flag).toEqual(false); //!!! WHY?
    });

});
```

4. Jasmine

4.1. Xử lý bất đồng bộ

Giải thích:

Jasmine sẽ tìm và thực thi hàm expect() trước khi hàm testAsync() chạy xong. Do hàm expect chạy xong trước khi flag được gán giá trị true(việc gán giá trị bị trì hoãn 2 giây), nên hàm expect() sẽ nhận giá trị flag = false;

Giải quyết:

- Đưa những đoạn mã bất đồng bộ vào hàm beforeEach() của spec suite.
- Để Jasmine biết được khi nào đoạn mã bất đồng bộ thực thi xong bằng hàm call() do Jasmine cung cấp.

```
var flag = false;

function testAsync(done) {
    // đợi 2 giây, sau đó gán flag = true;
    setTimeout(function () {
        flag = true;

        done();
    }, 2000);
}

describe("Testing async calls", function () {

    beforeEach(function(done)
    {
        testAsync(done);
    });

    it("Should be true if the async call has completed", function
(done) {
        expect(flag).toEqual(true);
        done(); // đảm bảo spec này được chạy xong trước khi chạy
spec khác
    });

});
```

4. Jasmine

4.1. Xử lý bất đồng bộ

Lưu ý:

Jasmine sẽ đợi những spec bất đồng bộ kết thúc trước khi thông báo lỗi timeout với thời gian đợi mặc định là 5 giây.

Giá trị này được lưu trong biến jasmine.DEFAULT_TIMEOUT_INTERVAL.

Ta có thể thay đổi giá trị này.

```
describe("Testing async calls", function () {
        var originalTimeout;

        beforeEach(function()
        {
            var originalTimeout = jasmine.DEFAULT_TIMEOUT_INTERVAL;
            jasmine.DEFAULT_TIMEOUT_INTERVAL = 7000;
        });

        it("", function(done)
        {
            setTimeout(function () {
            flag = true;

            done();
        }, 6000);
        });

        it("Should be true if the async call has completed", function ()
        {
            expect(flag).toEqual(true);
        });

        afterEach(function()
        {
             jasmine.DEFAULT_TIMEOUT_INTERVAL = originalTimeout;
        });

    });
```

5. TÀI LIỆU THAM KHẢO

- http://jasmine.github.io/2.0/introduction.html.
- http://jasmine.github.io/2.0/custom_matcher.html.
- https://volaresystems.com/blog/post/2014/12/09/Testing-async-calls-with-Jasmine.
- https://www.safaribooksonline.com/library/view/javascript-testing-with/9781449356729/_matchers_reference.html.
- http://mathjs.org/

6. DEMO

1

2

3

4

5

6


Jasmine