Mocha

Victor Herrero Cazurro

Contenidos

1. Introduccion

Framework de test para Node.js.

Se puede elegir el estilo de DSL a emplear, es decir el conjunto de funciones (palabras reservadas) que el framework pone a nuestra disposicion, teniendo estas opciones

- BDD
- TDD
- Exports
- QUnit
- Require-style

2. Instalacion

Se requiere Node.js v6.x o superior.

Se recomienda inicializar un proyecto **Node.js** lanzando el comando

```
> npm init
```

Obteniendo así el fichero **package.json**

Se ha de lanzar el siguiente comando para crear un proyecto **Mocha**

```
> npm install mocha
```

Se han de crear por defecto los test en la carpeta **test**

Y se ha de lanzar el siguiente comando para ejecutar las pruebas con **Mocha**

```
> node node_modules/mocha/bin/mocha
```

Habitualmente se emplea junto con **Mocha** la libreria **Chai** que aumenta las opciones de asertos, para instalarla

```
> npm install chai
```

Para emplear esta libreria se ha de añadir a la cabecera de los test la siguiente linea

```
var assert = require("chai").assert;
```

Se puede incorporar a los scripts de **npm**, añadiendo al fichero **package.json** lo siguiente

```
"scripts": {
    "test": "mocha"
  }
```

Y luego ejecutarlos con el comando

```
> npm test
```

3. Suite

Se defienen con la funcion **describe(enunciado,callack)**.

```
describe("Calcultator tests using ASSERT interface from CHAI module: ",
function() {
  describe("Check addTested Function: ", function() {
    it("Check the returned value using: assert.equal(value,'value'): ",
function() {
    });
  });
});
```

Sirven para agrupar varios **spec**, normalmente es parte de una frase comun a todos los **spec**.

Pueden contener otros bloques **describe**.

Permiten inicializar variables globales del **Suite**, como la carga de modulos de **node.js**

```
describe("Player", function() {
  var Player = require('../../lib/jasmine_examples/Player');
});
```

Se pueden saltar **test**, con la funcion **skip** de los **describe** o de los **it**

```
describe.skip('#indexOf()', function() {
    // ...
});
```

4. Spec

Se defienen con la funcion **it(enunciado,callack)**.

Sirven para definir una prueba, definen un texto que completa el texto del describe donde se incluyen y son los encargados de ejecutar la pueba sobre el **SUT**.

Dentro contienen las validaciones **expect** necesarias.

```
describe("Check addTested Function: ", function() {
    it("Check the returned value using: assert.equal(value,'value'): ",
function() {
      result   = calculator.addTested("text");
      assert.equal(result, "text tested");
    });
});
```

Como los **describe** poseen la funcion **skip** que permite desactivarlas

```
describe('Array', function() {
  describe('#indexOf()', function() {
    it.skip('should return -1 unless present', function() {
      // this test will not be run
    });

    it('should return the index when present', function() {
      // this test will be run
    });
  });
});
```

5. Assert

Se pueden emplear distintos estilos de aserciones

- Assert

```
var assert    = require("chai").assert;
var calculator = require("../app/calculator");

describe("Calcultator tests using ASSERT interface from CHAI module: ",
function() {
  describe("Check addTested Function: ", function() {
    it("Check the returned value using: assert.equal(value,'value'): ",
function() {
      result   = calculator.addTested("text");
      assert.equal(result, "text tested");
    });

    it("Check the returned value using: assert.typeOf(value,'value'):
", function() {
      result   = calculator.addTested("text");
      assert.typeOf(result, "string");
    });

    it("Check the returned value using: assert.lengthOf(value,'value'):
", function() {
      result   = calculator.addTested("text");
      assert.lengthOf(result, 11);
    });
  });
});
```

- Expect

```
var expect    = require("chai").expect;
var calculator = require("../app/calculator");

describe("Calcultator tests using EXPECT interface from CHAI module: ",
function() {
    describe("Check addTested Function: ", function() {
        it("Check the returned value using:
expect(value).to.equal('value'): ", function() {
            result   = calculator.addTested("text");
            expect(result).to.equal("text tested");
        });
        it("Check the returned value using:
expect(value).to.be.a('value')): ", function() {
            result   = calculator.addTested("text");
            expect(result).to.be.a('string');
        });
        it("Check the returned value using:
expect(value).to.have.lengthOf(value): ", function() {
            result   = calculator.addTested("text");
            expect(result).to.have.lengthOf(11);
        });
    });
});
```

- Should

```
var should    = require("chai").should();
var calculator = require("../app/calculator");

describe("Calcultator tests using SHOULD interface from CHAI module: ",
function() {
    describe("Check addTested Function: ", function() {
        it("Check the returned value using: value.should.equal(value):
", function() {
            result   = calculator.addTested("text");
            result.should.equal("text tested");
        });
        it("Check the returned value using: value.should.be.a('value'):
", function() {
            result   = calculator.addTested("text");
            result.should.be.a('string');
        });
        it("Check the returned value using:
expect(value).to.have.lengthOf(value): ", function() {
            result   = calculator.addTested("text");
            result.should.have.lengthOf(11);
        });
    });

});
```

Donde para los anteriores ejemplos, se ha definido el fichero **app/calculator.js** con contenido

```
exports.addTested = function(value) {
    var result = value + " tested";
    return result;
};
```

6. Preparacion

Se proporcionan funciones que permiten preparar el entorno de ejecucion de la prueba, a nivel del **suite** con las funciones:

- **before()**, la función dentro de before se va a ejecutar antes del primer test dentro del describe o context.

```
describe('hooks', function() {

  before(function() {
    // runs before all tests in this block
  });

  // test cases
});
```

- **after()**, la función dentro de after se va a ejecutar después del último test dentro del describe o context.

```
describe('hooks', function() {

  after(function() {
    // runs after all tests in this block
  });

  // test cases
});
```

- **beforeEach()**, la función dentro de beforeEach se va a ejecutar antes de cada test dentro del describe o context.

```
describe('Connection', function() {
  var db = new Connection,
    tobi = new User('tobi'),
    loki = new User('loki'),
    jane = new User('jane');

  beforeEach(function(done) {
    db.clear(function(err) {
      if (err) return done(err);
      db.save([tobi, loki, jane], done);
    });
  });

  describe('#find()', function() {
    it('respond with matching records', function(done) {
      db.find({type: 'User'}, function(err, res) {
        if (err) return done(err);
        res.should.have.length(3);
        done();
      });
    });
  });
});
```

- **afterEach()**, la función dentro de afterEach se va a ejecutar después de cada test dentro del describe o context.

```
describe('hooks', function() {

  afterEach(function() {
    // runs after each test in this block
  });

  // test cases
});
```

7. Reportes

Se pueden generar reportes con los siguietes formatos

- Dot

- Doc
- TAP
- JSON
- HTML
- List
- Min
- Spec
- Nyan
- XUnit
- Markdown
- Progress
- Landing
- JSONCov
- HTMLCov
- JSONStream

Para ello hay que lanzar el comando con el modificador **--reporter** eligiendo alguna de las opciones, adionalmente se puee volcar el resultado a un fichero

```
> npm test --reporter=XUnit > filename.xml
```