/\*

Created by laixiangran on 2015/12/15.

jasmine测试脚本

\*/

(function() {

/\*

jasmine基本语法介绍：

describe(string, function)：可以理解为是一个测试集或者测试包（官方称之为suite），主要功能是用来划分单元测试的，describe是可以嵌套使用的

参数string：描述测试包的信息

参数function：测试集的具体实现，可包含任意代码

it(string, function)：测试用例（官方称之为spec）

参数string：描述测试用例的信息

参数function：测试用例的具体实现，可包含任意代码

expect：断言表达式

从以下例子可知：

1、每个测试文件中可以包含多个describe

2、每个describe中可以包含多个it

3、每个it中可以包含多个expect

4、describe可嵌套使用

\*/

describe("Jasmine Test 1", function() {

it("a spec with an expectation", function() {

expect(1).toBe(1);

expect(1===1).toBe(true);

expect("a").not.toBe("b");

});

it("an other spec in current suite", function() {

expect(true).toBe(true);

});

});

describe("Jasmine Test 2", function() {

it("nothing", function() {

});

});

describe("Jasmine Test 3", function() {

describe("Jasmine Test 4", function() {

it("b等于b", function() {

expect("b").toBe("b");

});

it("1===1是正确的", function() {

expect(1===1).toBe(true);

});

});

});

/\*

\* expect的匹配器

\* \*/

describe("Included matchers:", function() {

//"toBe"基本类型判断

it("The 'toBe' matcher compares with ===", function() {

var a = 12;

var b = a;

expect(a).toBe(b);

expect(a).not.toBe(null);

});

//"toEqual"除了能判断基本类型（相当于"toBe"），还能判断对象

describe("The 'toEqual' matcher", function() {

//基本类型判断

it("works for simple literals and variables", function() {

var a = 12;

expect(a).toEqual(12);

});

//对象判断

it("should work for objects", function() {

var foo = {

a: 12,

b: 34

};

var bar = {

a: 12,

b: 34

};

expect(foo).toEqual(bar);

});

});

//"toMatch"使用正则表达式判断

it("The 'toMatch' matcher is for regular expressions", function() {

var message = "foo bar baz";

expect(message).toMatch(/bar/);

expect(message).toMatch("bar");

expect(message).not.toMatch(/quux/);

});

//"toBeDefined"判断是否定义

it("The 'toBeDefined' matcher compares against 'undefined'", function() {

var a = {

foo: "foo"

};

expect(a.foo).toBeDefined();

expect(a.bar).not.toBeDefined();

});

//"toBeUndefined"判断是否是undefined，与"toBeDefined"相反

it("The 'toBeUndefined' matcher compares against 'undefined'", function() {

var a = {

foo: "foo"

};

expect(a.foo).not.toBeUndefined();

expect(a.bar).toBeUndefined();

});

//"toBeNull"判断是否为null

it("The 'toBeNull' matcher compares against null", function() {

var a = null;

var foo = "foo";

expect(null).toBeNull();

expect(a).toBeNull();

expect(foo).not.toBeNull();

});

//"toBeTruthy"判断是否是true

it("The 'toBeTruthy' matcher is for boolean casting testing", function() {

var a, foo = "foo";

expect(foo).toBeTruthy();

expect(a).not.toBeTruthy();

expect(true).toBeTruthy();

});

//"toBeFalsy"判断是否是false

it("The 'toBeFalsy' matcher is for boolean casting testing", function() {

var a, foo = "foo";

expect(a).toBeFalsy();

expect(foo).not.toBeFalsy();

expect(false).toBeFalsy();

});

//"toContain"判断数组是否包含（可判断基本类型和对象）

it("The 'toContain' matcher is for finding an item in an Array", function() {

var a = ["foo", "bar", "baz"];

var b = [{foo: "foo", bar: "bar"}, {baz: "baz", bar: "bar"}];

expect(a).toContain("bar");

expect(a).not.toContain("quux");

expect(b).toContain({foo: "foo", bar: "bar"});

expect(b).not.toContain({foo: "foo", baz: "baz"});

});

//"toBeLessThan"判断值类型的大小，结果若小则为True（也可以判断字符及字符串，以ascii码的大小为判断依据）

it("The 'toBeLessThan' matcher is for mathematical comparisons", function() {

var pi = 3.1415926,

e = 2.78;

expect(e).toBeLessThan(pi);

expect(pi).not.toBeLessThan(e);

expect("a").toBeLessThan("b");

expect("b").not.toBeLessThan("a");

});

//"toBeGreaterThan"判断值类型的大小，结果若大则为True，与toBeLessThan相反（也可以判断字符及字符串，以ascii码的大小为判断依据）

it("The 'toBeGreaterThan' matcher is for mathematical comparisons", function() {

var pi = 3.1415926,

e = 2.78;

expect(pi).toBeGreaterThan(e);

expect(e).not.toBeGreaterThan(pi);

expect("a").not.toBeGreaterThan("b");

expect("b").toBeGreaterThan("a");

});

//"toBeCloseTo"判断数字是否相似（第二个参数为小数精度，默认为2位）

it("The 'toBeCloseTo' matcher is for precision math comparison", function() {

var a = 1.1;

var b = 1.5;

var c = 1.455;

var d = 1.459;

expect(a).toBeCloseTo(b, 0);

expect(a).not.toBeCloseTo(c, 1);

expect(c).toBeCloseTo(d);

});

//"toThrow"判断是否抛出异常

it("The 'toThrow' matcher is for testing if a function throws an exception", function() {

var foo = function() {

return 1 + 2;

};

var bar = function() {

return a + 1;

};

expect(foo).not.toThrow();

expect(bar).toThrow();

});

//"toThrowError"判断是否抛出了指定的错误

it("The 'toThrowError' matcher is for testing a specific thrown exception", function() {

var foo = function() {

throw new TypeError("foo bar baz");

};

expect(foo).toThrowError("foo bar baz");

expect(foo).toThrowError(/bar/);

expect(foo).toThrowError(TypeError);

expect(foo).toThrowError(TypeError, "foo bar baz");

});

});

/\*

\* "fail"函数能使一个测试用例失败，参数为自定义的失败信息

\* \*/

describe("A spec using the fail function", function() {

var foo = function(x, callBack) {

if (x) {

callBack();

}

};

it("should not call the callBack", function() {

foo(false, function() {

fail("Callback has been called");

});

});

});

/\*

Jasmine允许在执行测试集/测试用例的开始前/结束后做一些初始化/销毁的操作。

Setup方法：

beforeAll：每个suite（即describe）中所有spec（即it）运行之前运行

beforeEach：每个spec（即it）运行之前运行

Teardown方法：

afterAll：每个suite（即describe）中所有spec（即it）运行之后运行

afterEach：每个spec（即it）运行之后运行

\* \*/

var globalCount;

describe("Setup and Teardown suite 1", function() {

var suiteGlobalCount;

var eachTestCount;

beforeAll(function() {

globalCount = 0;

suiteGlobalCount = 0;

eachTestCount = 0;

});

afterAll(function() {

suiteGlobalCount = 0;

});

beforeEach(function() {

globalCount++;

suiteGlobalCount++;

eachTestCount++;

});

afterEach(function() {

eachTestCount = 0;

});

it("Spec 1", function() {

expect(globalCount).toBe(1);

expect(suiteGlobalCount).toBe(1);

expect(eachTestCount).toBe(1);

});

it("Spec 2", function() {

expect(globalCount).toBe(2);

expect(suiteGlobalCount).toBe(2);

expect(eachTestCount).toBe(1);

});

});

describe("Setup and Teardown suite 2", function() {

beforeEach(function() {

globalCount += 2;

});

it("Spec 1", function() {

expect(globalCount).toBe(4);

});

});

/\*

\* 在beforeEach - it - afterEach中，还可以使用this关键字定义变量。需要注意的是，使用this关键字声明的变量，仅在beforeEach - it - afterEach这个过程中传递

\* \*/

describe("Test 'this'", function() {

beforeEach(function() {

this.testCount = this.testCount || 0;

this.testCount++;

});

afterEach(function() {

//this.testCount = 0; //无论是否有这行，结果是一样的，因为this指定的变量只能在每个spec的beforeEach/it/afterEach过程中传递

});

it("Spec 1", function() {

expect(this.testCount).toBe(1);

});

it("Spec 2", function() {

expect(this.testCount).toBe(1);

});

});

/\*

在实际项目中，需要由于发布的版本需要选择测试用例包，xdescribe和xit能很方便的将不包含在版本中的测试用例排除在外。

不过xdescribe和xit略有不同：

xdescribe：该describe下的所有it将被忽略，Jasmine将直接忽略这些it，因此不会被运行

xit：运行到该it时，挂起它不执行

\* \*/

xdescribe("Test xdescribe", function() {

it("Spec 1", function() {

expect(1).toBe(1);

});

it("Spec 2", function() {

expect(2).toBe(2);

});

});

describe("Test xit", function() {

it("Spec 1", function() {

expect(1).toBe(1);

});

xit("Spec 2", function() {

expect(2).toBe(1);

});

xit("Spec 3", function() {

expect(3).toBe(3);

});

});

/\*

\* Spy用来追踪函数的调用历史信息（是否被调用、调用参数列表、被请求次数等）。Spy仅存在于定义它的describe和it方法块中，并且每次在spec执行完之后被销毁。

\* 当在一个对象上使用spyOn方法后即可模拟调用对象上的函数，此时对所有函数的调用是不会执行实际代码的。

\* 两个Spy常用的expect：

\* toHaveBeenCalled: 函数是否被调用

\* toHaveBeenCalledWith: 调用函数时的参数

\* \*/

describe("A spy", function() {

var foo, bar = null;

beforeEach(function() {

foo = {

setBar: function(value) {

bar = value;

}

};

spyOn(foo, "setBar"); // 在foo对象上添加spy

// 此时调用foo对象上的方法，均为模拟调用，因此不会执行实际的代码

foo.setBar(123); // 调用foo的setBar方法

foo.setBar(456, "another param");

});

it("tracks that the spy was called", function() {

expect(foo.setBar).toHaveBeenCalled(); //判断foo的setBar是否被调用

});

it("tracks all the arguments of its calls", function() {

expect(foo.setBar).toHaveBeenCalledWith(123); //判断被调用时的参数

expect(foo.setBar).toHaveBeenCalledWith(456, "another param");

});

it("stops all execution on a function", function() {

expect(bar).toBeNull(); // 由于是模拟调用，因此bar值并没有改变

});

});

/\*

\* spyOn().and.callThrough()，告诉Jasmine我们除了要完成对函数调用的跟踪，同时也需要执行实际的代码。

\* \*/

describe("A spy, when configured to call through", function() {

var foo, bar, fetchedBar;

beforeEach(function() {

foo = {

setBar: function(value) {

bar = value;

},

getBar: function() {

return bar;

}

};

spyOn(foo, "getBar").and.callThrough(); // 这里使用了callThrough，这时所有的函数调用为真实的执行

spyOn(foo, "setBar").and.callThrough();

foo.setBar(123);

fetchedBar = foo.getBar();

});

it("tracks that the spy was called", function() {

expect(foo.getBar).toHaveBeenCalled();

});

it("should not effect other functions", function() {

expect(foo.setBar).toHaveBeenCalledWith(123);

expect(bar).toEqual(123); // 由于是真实调用，因此bar有了真实的值

});

it("when called returns the requested value", function() {

expect(fetchedBar).toEqual(123); // 由于是真实调用，fetchedBar也有了真实的值

});

});

/\*

\* spyOn().and.returnValue()，由于Spy是模拟函数的调用，因此我们也可以强制指定函数的返回值。

\* \*/

describe("A spy, when configured to fake a return value", function() {

var foo, bar, fetchedBar;

beforeEach(function() {

foo = {

setBar: function(value) {

bar = value;

},

getBar: function() {

return bar;

}

};

spyOn(foo, "getBar").and.returnValue(745); // 这将指定getBar方法返回值为745

foo.setBar(123);

fetchedBar = foo.getBar();

});

it("tracks that the spy was called", function() {

expect(foo.getBar).toHaveBeenCalled();

});

it("should not effect other functions", function() {

expect(bar).toEqual(123);

});

it("when called returns the requested value", function() {

expect(fetchedBar).toEqual(745);

});

});

/\*

\* spyOn().and.callFake()，

\* 与returnValue相似，callFake则更进一步，直接通过指定一个假的自定义函数来执行。这种方式比returnValue更灵活，我们可以任意捏造一个函数来达到我们的测试要求。

\* \*/

describe("A spy, when configured with an alternate implementation", function() {

var foo, bar, fetchedBar;

beforeEach(function() {

foo = {

setBar: function(value) {

bar = value;

},

getBar: function() {

return bar;

}

};

spyOn(foo, "getBar").and.callFake(function() {

return 1001;

});

foo.setBar(123);

fetchedBar = foo.getBar();

});

it("tracks that the spy was called", function() {

expect(foo.getBar).toHaveBeenCalled();

});

it("should not effect other functions", function() {

expect(bar).toEqual(123);

});

it("when called returns the requested value", function() {

expect(fetchedBar).toEqual(1001);

});

});

/\*

\* spyOn().and.throwError()，模拟异常的抛出

\* \*/

describe("A spy, when configured to throw an error", function() {

var foo, bar;

beforeEach(function() {

foo = {

setBar: function(value) {

bar = value;

}

};

spyOn(foo, "setBar").and.throwError("quux");

});

it("throws the value", function() {

expect(function() {

foo.setBar(123)

}).toThrowError("quux");

});

});

/\*

\* spyOn().and.stub()，回复到原始的spyOn()方法

\* \*/

describe("A spy", function() {

var foo, bar = null;

beforeEach(function() {

foo = {

setBar: function(value) {

bar = value;

},

getBar: function(){

return bar;

}

};

spyOn(foo, "setBar").and.callThrough(); // 标记1

spyOn(foo, "getBar").and.returnValue(999); // 标记2

});

it("can call through and then stub in the same spec", function() {

foo.setBar(123);

expect(bar).toEqual(123);

var getValue = foo.getBar();

expect(getValue).toEqual(999);

foo.setBar.and.stub(); // 相当于"标记1"中的代码变为了spyOn(foo, "setBar")

foo.getBar.and.stub(); // 相当于"标记2"中的代码变为了spyOn(foo, "getBar")

bar = null;

foo.setBar(123);

expect(bar).toBe(null);

expect(foo.setBar).toHaveBeenCalled(); // 函数调用追踪并没有被重置

getValue = foo.getBar();

expect(getValue).toEqual(undefined);

expect(foo.getBar).toHaveBeenCalled(); // 函数调用追踪并没有被重置

});

});

/\*

\* 其他追踪属性：

calls：对于被Spy的函数的调用，都可以在calls属性中跟踪。

.calls.any(): 被Spy的函数一旦被调用过，则返回true，否则为false；

.calls.count(): 返回被Spy的函数的被调用次数；

.calls.argsFor(index): 返回被Spy的函数的调用参数，以index来指定参数；

.calls.allArgs():返回被Spy的函数的所有调用参数；

.calls.all(): 返回calls的上下文，这将返回当前calls的整个实例数据；

.calls.mostRecent(): 返回calls中追踪的最近一次的请求数据；

.calls.first(): 返回calls中追踪的第一次请求的数据；

.object: 当调用all()，mostRecent()，first()方法时，返回对象的object属性返回的是当前上下文对象；

.calls.reset(): 重置Spy的所有追踪数据；

\* \*/

describe("A spy", function() {

var foo, bar = null;

beforeEach(function() {

foo = {

setBar: function(value) {

bar = value;

}

};

spyOn(foo, "setBar");

});

//.calls.any(): 被Spy的函数一旦被调用过，则返回true，否则为false；

it("tracks if it was called at all", function() {

expect(foo.setBar.calls.any()).toEqual(false);

foo.setBar();

expect(foo.setBar.calls.any()).toEqual(true);

});

//.calls.count(): 返回被Spy的函数的被调用次数；

it("tracks the number of times it was called", function() {

expect(foo.setBar.calls.count()).toEqual(0);

foo.setBar();

foo.setBar();

expect(foo.setBar.calls.count()).toEqual(2);

});

//.calls.argsFor(index): 返回被Spy的函数的调用参数，以index来指定参数；

it("tracks the arguments of each call", function() {

foo.setBar(123);

foo.setBar(456, "baz");

expect(foo.setBar.calls.argsFor(0)).toEqual([123]);

expect(foo.setBar.calls.argsFor(1)).toEqual([456, "baz"]);

});

//.calls.allArgs():返回被Spy的函数的所有调用参数；

it("tracks the arguments of all calls", function() {

foo.setBar(123);

foo.setBar(456, "baz");

expect(foo.setBar.calls.allArgs()).toEqual([[123],[456, "baz"]]);

});

//.calls.all(): 返回calls的上下文，这将返回当前calls的整个实例数据；

it("can provide the context and arguments to all calls", function() {

foo.setBar(123);

expect(foo.setBar.calls.all()).toEqual([{object: foo, args: [123], returnValue: undefined}]);

});

//.calls.mostRecent(): 返回calls中追踪的最近一次的请求数据；

it("has a shortcut to the most recent call", function() {

foo.setBar(123);

foo.setBar(456, "baz");

expect(foo.setBar.calls.mostRecent()).toEqual({object: foo, args: [456, "baz"], returnValue: undefined});

});

//.calls.first(): 返回calls中追踪的第一次请求的数据；

it("has a shortcut to the first call", function() {

foo.setBar(123);

foo.setBar(456, "baz");

expect(foo.setBar.calls.first()).toEqual({object: foo, args: [123], returnValue: undefined});

});

//.object: 当调用all()，mostRecent()，first()方法时，返回对象的object属性返回的是当前上下文对象；

it("tracks the context", function() {

var spy = jasmine.createSpy("spy");

var baz = {

fn: spy

};

var quux = {

fn: spy

};

baz.fn(123);

quux.fn(456);

expect(spy.calls.first().object).toBe(baz);

expect(spy.calls.mostRecent().object).toBe(quux);

});

//.calls.reset(): 重置Spy的所有追踪数据；

it("can be reset", function() {

foo.setBar(123);

foo.setBar(456, "baz");

expect(foo.setBar.calls.any()).toBe(true);

foo.setBar.calls.reset();

expect(foo.setBar.calls.any()).toBe(false);

});

});

/\*

jasmine.createSpy()

\* 假如没有函数可以追踪，我们可以自己创建一个空的Spy。

\* 创建后的Spy功能与其他的Spy一样：跟踪调用、参数等，但该Spy没有实际的代码实现，这种方式经常会用在对JavaScript中的对象的测试。

\* \*/

describe("A spy, when created manually", function() {

var whatAmI;

beforeEach(function() {

whatAmI = jasmine.createSpy("whatAmI");

whatAmI("I", "am", "a", "spy");

});

it("is named, which helps in error reporting", function() {

expect(whatAmI.and.identity()).toEqual("whatAmI");

});

it("tracks that the spy was called", function() {

expect(whatAmI).toHaveBeenCalled();

});

it("tracks its number of calls", function() {

expect(whatAmI.calls.count()).toEqual(1);

});

it("tracks all the arguments of its calls", function() {

expect(whatAmI).toHaveBeenCalledWith("I", "am", "a", "spy");

});

it("allows access to the most recent call", function() {

expect(whatAmI.calls.mostRecent().args[0]).toEqual("I");

});

});

/\*

jasmine.createSpyObj()

\* 如果需要spy模拟多个函数调用，可以向jasmine.createSpyObj中传入一个字符串数组，它将返回一个对象，

\* 你所传入的所有字符串都将对应一个属性，每个属性即为一个Spy。

\* \*/

describe("Multiple spies, when created manually", function() {

var tape;

beforeEach(function() {

tape = jasmine.createSpyObj('tape', ['play', 'pause', 'stop', 'rewind']);

tape.play();

tape.pause();

tape.rewind(0);

});

it("creates spies for each requested function", function() {

expect(tape.play).toBeDefined();

expect(tape.pause).toBeDefined();

expect(tape.stop).toBeDefined();

expect(tape.rewind).toBeDefined();

});

it("tracks that the spies were called", function() {

expect(tape.play).toHaveBeenCalled();

expect(tape.pause).toHaveBeenCalled();

expect(tape.rewind).toHaveBeenCalled();

expect(tape.stop).not.toHaveBeenCalled();

});

it("tracks all the arguments of its calls", function() {

expect(tape.rewind).toHaveBeenCalledWith(0);

});

});

/\*

\* jasmine.any()

\* 以构造器或者类名作为参数，Jasmine将判断期望值和真实值的构造器是否相同，若相同则返回true。

\* \*/

describe("jasmine.any", function() {

it("matches any value", function() {

expect({}).toEqual(jasmine.any(Object));

expect(12).toEqual(jasmine.any(Number));

});

describe("when used with a spy", function() {

it("is useful for comparing arguments", function() {

var foo = jasmine.createSpy("foo");

foo(12, function() {

return true;

});

expect(foo).toHaveBeenCalledWith(jasmine.any(Number), jasmine.any(Function));

});

});

});

/\*

\* jasmine.anything()

\* 判断只要不是null或undefined的值，若不是则返回true。

\* \*/

describe("jasmine.anything", function() {

it("matches anything", function() {

expect(1).toEqual(jasmine.anything());

});

describe("when used with a spy", function() {

it("is useful when the argument can be ignored", function() {

var foo = jasmine.createSpy('foo');

foo(12, function() {

return false;

});

expect(foo).toHaveBeenCalledWith(12, jasmine.anything());

});

});

});

/\*

\* jasmine.objectContaining()

\* 用来判断对象中是否存在指定的键值属性对。

\* \*/

describe("jasmine.objectContaining", function() {

var foo;

beforeEach(function() {

foo = {

a: 1,

b: 2,

bar: "baz"

};

});

it("matches objects with the expect key/value pairs", function() {

expect(foo).toEqual(jasmine.objectContaining({

bar: "baz"

}));

expect(foo).not.toEqual(jasmine.objectContaining({

c: 37

}));

});

describe("when used with a spy", function() {

it("is useful for comparing arguments", function() {

var callback = jasmine.createSpy("callback");

callback({

bar: "baz"

});

expect(callback).toHaveBeenCalledWith(jasmine.objectContaining({

bar: "baz"

}));

expect(callback).not.toHaveBeenCalledWith(jasmine.objectContaining({

c: 37

}));

});

});

});

/\*

\* jasmine.arrayContaining()

\* 可以用来判断数组中是否有期望的值。

\* \*/

describe("jasmine.arrayContaining", function() {

var foo;

beforeEach(function() {

foo = [1, 2, 3, 4];

});

it("matches arrays with some of the values", function() {

expect(foo).toEqual(jasmine.arrayContaining([3, 1])); // 直接在期望值中使用jasmine.arrayContaining达到目的

expect(foo).not.toEqual(jasmine.arrayContaining([6]));

});

describe("when used with a spy", function() {

it("is useful when comparing arguments", function() {

var callback = jasmine.createSpy("callback"); // 创建一个空的Spy

callback([1, 2, 3, 4]); // 将数组内容作为参数传入Spy中

expect(callback).toHaveBeenCalledWith(jasmine.arrayContaining([4, 2, 3]));

expect(callback).not.toHaveBeenCalledWith(jasmine.arrayContaining([5, 2]));

});

});

});

/\*

\* jasmine.stringMatching()

\* 用来模糊匹配字符串，在jasmine.stringMatching中也可以使用正则表达式进行匹配，使用起来非常灵活。

\* \*/

describe("jasmine.stringMatching", function() {

it("matches as a regexp", function() {

expect({foo: "bar"}).toEqual({foo: jasmine.stringMatching(/^bar$/)});

expect({foo: "foobarbaz"}).toEqual({foo: jasmine.stringMatching("bar")});

});

describe("when used with a spy", function() {

it("is useful for comparing arguments", function() {

var callback = jasmine.createSpy("callback");

callback("foobarbaz");

expect(callback).toHaveBeenCalledWith(jasmine.stringMatching("bar"));

expect(callback).not.toHaveBeenCalledWith(jasmine.stringMatching(/^bar$/));

});

});

});

/\*

\* 不规则匹配（自定义匹配）：asymmetricMatch

\* 某些场景下，我们希望能按照自己设计的规则进行匹配，此时我们可以自定义一个对象，该对象只要包含一个名为asymmetricMatch的方法即可。

\* \*/

describe("custom asymmetry", function() {

var tester = {

asymmetricMatch: function(actual) {

var secondValue = actual.split(",")[1];

return secondValue === "bar";

}

};

it("dives in deep", function() {

expect("foo,bar,baz,quux").toEqual(tester);

});

describe("when used with a spy", function() {

it("is useful for comparing arguments", function() {

var callback = jasmine.createSpy("callback");

callback("foo,bar,baz");

expect(callback).toHaveBeenCalledWith(tester);

});

});

});

/\*

\* jasmine.clock()用来模拟操纵时间。

\* 要想使用jasmine.clock()，先调用jasmine.clock().install告诉Jasmine你想要在spec或者suite操作时间，当你不需要使用时，务必调用jasmine.clock().uninstall来恢复时间状态。

\*

\* 示例中使用jasmine.clock().tick(milliseconds)来控制时间前进，本例中出现了三种时间控制方式：

\* setTimeout: 定期执行一次，当jasmine.clock().tick()的时间超过了timeout设置的时间时触发

\* setInterval: 定期循环执行，每当jasmine.clock().tick()的时间超过了timeout设置的时间时触发

\* mockDate: 模拟一个指定日期（当不提供基准时间参数时，以当前时间为基准时间）

\* \*/

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);

});

describe("Mocking the Date object", function(){

it("mocks the Date object and sets it to a given time", function() {

var baseTime = new Date();

jasmine.clock().mockDate(baseTime);

jasmine.clock().tick(50);

expect(new Date().getTime()).toEqual(baseTime.getTime() + 50);

});

});

});

/\*

\* Jasmine可以支持spec中执行异步操作。

\* 当调用beforeEach, it和afterEach时，函数可以包含一个可选参数done，当spec执行完毕之后，调用done通知Jasmine异步操作已执行完毕。

\*

\* 另外补充一点，如果需要设置全局的默认超时时间，可以设置jasmine.DEFAULT\_TIMEOUT\_INTERVAL的值，

\* 当异步执行时间超过设置的执行超时时间js将会报错。

\* \*/

describe("Asynchronous specs", function() {

var value;

beforeEach(function(done) {

setTimeout(function() {

value = 0;

done();

}, 1);

});

// 在上面beforeEach的done()被执行之前，这个测试用例不会被执行

it("should support async execution of test preparation and expectations", function(done) {

value++;

expect(value).toBeGreaterThan(0);

done(); // 执行完done()之后，该测试用例真正执行完成

});

// Jasmine异步执行超时时间默认为5秒，超过后将报错

describe("long asynchronous specs", function() {

var originalTimeout;

beforeEach(function() {

originalTimeout = jasmine.DEFAULT\_TIMEOUT\_INTERVAL;

// 设置全局的默认超时时间

jasmine.DEFAULT\_TIMEOUT\_INTERVAL = 6000;

});

it("takes a long time", function(done) {

setTimeout(function() {

done();

}, 4000);

});

// 如果要调整指定用例的默认的超时时间，可以在beforeEach，it和afterEach中传入一个时间参数

//it("takes a long time for this spec", function(done) {

// setTimeout(function() {

// done();

// }, 6000);

//}, 7000);

afterEach(function() {

jasmine.DEFAULT\_TIMEOUT\_INTERVAL = originalTimeout;

});

});

});

}());