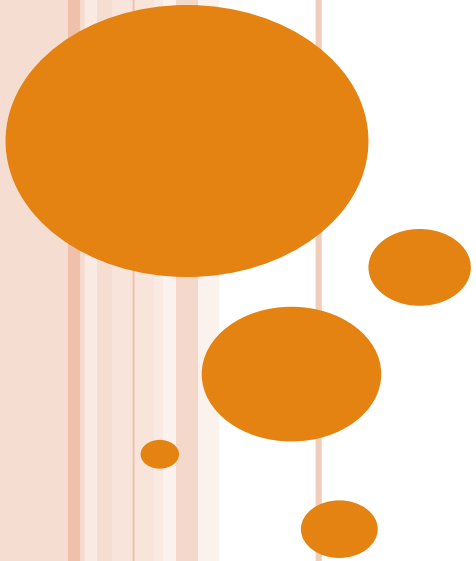
Automação de Testes - Jasmine

Testes de Softwares



Roteiro

- Introdução
- Estrutura e configuração do Jasmine
- Funções:
  - describe
  - it
  - expect
  - beforeEach
  - afterEach
- Metchers

Introdução ao Jasmine

- Jasmine é um framework para realizar teste de unidade em código JavaScript.
- Não depende de nenhum outro framework JavaScript para funcionar.
- Apresenta uma sintaxe limpa e óbvia, você pode facilmente escrever testes

- **Open-Source**
  - https://github.com/jasmine/jasmine/releases

Estrutura e Configuração do Jasmine

- Exemplo de estruturas de pastas e arquivos

```
Nome_Projeto
  |-- jasmine (onde ficarão os arquivos do framework jasmine)
  |-- js (onde ficarão os arquivos js)
      |-- arquivo.js (arquivo js que será testado)
      |-- spec
          |-- index.html (usado para visualizar os testes)
          |-- arquivo.spec.js (conterá código js com os testes)
```

Estrutura e Configuração do Jasmine

- Configuração do arquivo **spec/index.html**

```
<!DOCTYPE html>
<html lang="en">
    <head>
      <meta charset="UTF-8">
      <title>Jasmine</title>

      <!-- framework jasmine -->
      <link rel="shortcut icon" type="image/png" href="../../jasmine/lib/jasmine-2.7.0/jasmine_favicon.png">
      <link rel="stylesheet" type="text/css" href="../../jasmine/lib/jasmine-2.7.0/jasmine.css">
      <script type="text/javascript" src="../../jasmine/lib/jasmine-2.7.0/jasmine.js"></script>
      <script type="text/javascript" src="../../jasmine/lib/jasmine-2.7.0/jasmine-html.js"></script>
      <script type="text/javascript" src="../../jasmine/lib/jasmine-2.7.0/boot.js"></script>

      <!-- código js  -->
      <script src="../arquivo_1.js"></script>

      <!-- código js de teste -->
      <script src="arquivo_1.spec.js"></script>

    </head>
    <body>
    </body>
</html>
```

FUNÇÃO DESCRIBE

- A função **describe** é usada para organizar seus testes em suítes
  - Basicamente, uma coleção de testes organizados em blocos relacionados.
- Veja o **describe** em ação:

```
1  describe("Descrição de uma suite de teste", function(){
2      // Código de teste aqui
3  });
```

- Apresenta dois parâmetros:
  1. Uma string (identifica uma suíte de teste, mostra hierarquicamente quais testes passaram e falharam);
  2. Uma função (dentro da função escrevemos todo o código necessário para realizar o teste).

FUNÇÃO DESCRIBE

- Suítes também podem ser aninhadas (uma dentro da outra), isso permite organizar o código hierarquicamente.
- Boa prática:
  - Um bloco (**describe**) mais externo para um objeto e blocos (**describe**) mais internos para cada método.

```
1  describe("Descrição de um objeto", function(){
2      describe("Um método", function(){
3          // digite seu código de teste aqui
4      });
5      describe("Outro método", function(){
6          // digite seu código de teste aqui
7      });
8
9      //...
10 });
```

Função it

- A função **it** é usada para escrever os testes e fica dentro do bloco da função **describe**.
- Veja como é o **it** em ação:

```
1  describe("Uma descrição para uma suíte de testes", function(){
2      it("Nome do teste", function(){
3          // Escreva seu teste aqui
4      });
5  });
```

- Também apresenta dois parâmetros:
  1. Uma string (nome do teste)
  2. Uma função (dentro da função o código para o teste)
- O bloco da função **it** é também chamada de especulação

Função it

- Uma **especulação** contém uma ou mais **expectativas** que testam o estado do código.

- Uma **expectativa** para o Jasmine é uma **assertiva**,
  - Basicamente, uma afirmação que poderá ser verdadeira ou falsa.

- Uma **especulação** com todas as **expectativas** verdadeiras é uma especulação aprovada.

- Uma **especulação** com uma ou mais **expectativas** falsas, é uma especulação falha.

FUNÇÃO EXPECT

- A função **expect** (expectativa) é usada para testar os resultados, ela fica dentro da função **it**.
  - Basicamente, verifica se o *valor esperado* é igual ao *valor atual*.
- Veja como é o expect em ação:

```
1  describe("Uma descrição para uma suíte de testes", function(){
2      it("Nome do teste", function(){
3          var valor_atual = 2;
4          var valor_esperado = 2;
5          expect(valor_esperado).toBe(valor_atual);
6      });
7  });
```

- A função **expect** recebe como argumento o *valor esperado* e faz uma ligação com uma função comparadora,
  - A função comparadora (**toBe**) recebe como argumento *o valor atual*.

Função **expect**

- No exemplo anterior, usamos o *método comparador* (**toBe**()), mas existe mais de um método comparador
  - Esses métodos são chamados de **Matchers**
- Cada **matcher** (comparador) implementa uma comparação booleana entre o *valor esperado* e o *valor atual*.
- Objetivo do **matcher**:
  - Reportar ao Jasmine se o teste é verdadeiro ou falso, então o Jasmine simplesmente aprova ou reprova.
- O comparador (**toBe()**) compara os valores dados usando === do javascrip.
  - Ainda veremos mais sobre **Matchers**...

Função beforeEach e afterEach

- Essas funções são usadas dentro do bloco **describe** e antes dos blocos **it**

- A função **beforeEach**:
  - Executa antes de cada bloco **it**.
  - Usada para executar algumas configurações, como criar novo objeto

- A função **afterEach**:
  - Executada depois de cada bloco **it**.
  - Usada para executar alguma limpeza, como, por exemplo, destruir algum objeto.

Função beforeEach e afterEach

- Exemplo:

```
describe("Uso do beforeEach e afterEach", function(){
    var frutas;

    beforeEach(function(){
        // Executado antes de cada teste
        frutas=["Banana", "Laranja", "Limão"];
    });

    afterEach(function(){
        // Executado depois de cada teste
        frutas=null;
    });

    it("Teste 1", function(){
        frutas.push("Goiaba");
        expect(frutas.length).toBe(4);
    });

    it("Teste 2", function(){
        expect(frutas.length).toBe(3);
    });

});
```

Função beforeEach e afterEach

- Exemplo:

```
describe("Uso do beforeEach e afterEach", function(){
      // use a palavra reservada this para criar uma variável no contexto global

    beforeEach(function(){
        // Executado antes de cada teste
        this.frutas=["Banana", "Laranja", "Limão"];
    });

    afterEach(function(){
        // Executado depois de cada teste
        this.frutas=null;
    });

    it("Teste 1", function(){
        this.frutas.push("Goiaba");
        expect(this.frutas.length).toBe(4);
    });

    it("Teste 2", function(){
        expect(this.frutas.length).toBe(3);
    });

});
```

Matchers (comparadores)

- toBe( )
- toEqual( )
- toMatch( )
- toBeNull( )
- toBeTruthy( )
- toBeDefined( )
- toBeUndefined( )
- toContain( )
- toBeLessThan( )
- toBeGreaterThan( )
- toThrow( )

Matcher → toBe()

- O expect(x).toBe(y) passa no teste se o valor x é igual ao valor y.
- Exemplo:

```
1  describe("Matcher 'toBe()'", function(){
2      var n1=3, n2=5, n3=3;
3
4      it("Teste 1", function(){
5          expect(n1).toBe(n3);
6      });
7
8      it("Teste 2", function(){
9          expect(n1).toBe(n3);
10     });
11 });
```

MATCHER → TOEQUAL()

- O expect(x).toEqual(y) passa no teste se o objeto x é igual ao objeto y, pode trabalhar também com simples variáveis e literais.
- Exemplo:

```
1  describe("Matcher 'toEqual()'", function(){
2      var n1={a:2, b:4};
3      var n2={a:2, b:4};
4      var n3={a:2, b:5};
5
6      it("Teste 1", function(){
7          expect(n1).toEqual(n2);
8      });
9
10     it("Teste 2", function(){
11         expect(n2).toEqual(n3);
12     });
13 });
```

MATCHER → TOMATCH( )

- O expect(message).toMatch(pattern) é usado para expressões regulares, passa no teste se um padrão (pattern) está dentro de uma mensagem (message).
- Exemplo:

```
describe("Matcher 'toMatch()'", function(){
    var message="Eu sou um programador";
    var pattern1="ogra";
    var pattern2="made";

    it("Teste 1", function(){
        expect(message).toMatch(pattern1);
    });

    it("Teste 2", function(){
        expect(message).toMatch(pattern2);
    });
});
```

MATCHER → TOBENULL( )

- O expect(valor).toBeNull() passa no teste se o valor for null. Podemos usar o not.toBeNull() para realizar negação null.
- Exemplo:

```
describe("Matcher 'toBeNull()'", function(){
    var a=null, b="Tiririca";

    it("Teste 1", function(){
        expect(null).toBeNull();
    });

    it("Teste 2", function(){
        expect(a).toBeNull();
    });

    it("Teste 3", function(){
        expect(b).toBeNull();
    });

    it("Teste 4", function(){
        expect(b).not.toBeNull();
    });
});
```

MATCHER → TOBETRUTHY( )

- O expect(valor).toBeTruthy() passa no teste se o valor for verdadeiro.
- Exemplo:

```
1  describe("Matcher 'toBeTruthy()'", function(){
2      var valor_v=true, valor_f=false;
3
4      it("Teste 1", function(){
5          expect(valor_v).toBeTruthy();
6      });
7
8      it("Teste 2", function(){
9          expect(valor_f).toBeTruthy();
10     });
11 });
```

MATCHER → TOBEDEFINED( )

- O expect(valor).toBeDefined() passa no teste se o valor for definido.
- Exemplo:

```
1  describe("Matcher 'toBeDefined()'", function(){
2      var valor_a=455, valor_b;
3
4      it("Teste 1", function(){
5          expect(valor_a).toBeDefined();
6      });
7
8      it("Teste 2", function(){
9          expect(valor_b).toBeDefined();
10     });
11 });
```

MATCHER → TOBEUNDEFINED( )

- O expect(valor).toBeUndefined() passa no teste se o valor não for definido.
- Exemplo:

```
1  describe("Matcher 'toBeUndefined()'", function(){
2      var valor_a, valor_b=98;
3
4      it("Teste 1", function(){
5          expect(valor_a).toBeUndefined();
6      });
7
8      it("Teste 2", function(){
9          expect(valor_b).toBeUndefined();
10     });
11 });
```

MATCHER → TOCONTAIN( )

- O expect(array).toContain(valor) passa no teste se o valor estiver dentro do array.
- Exemplo:

```
1  describe("Matcher 'toContain()'", function(){
2      var nomes=["Elder", "Maria", "José"];
3
4      it("Teste 1", function(){
5          expect(nomes).toContain("Maria");
6      });
7
8      it("Teste 2", function(){
9          expect(nomes).toContain("Manuel");
10     });
11 });
```

MATCHER → TOBELESSTHAN( )

- O expect(x).toBeLessThan(y) passa no teste se o valor x for menor que o valor y.
- Exemplo:

```
1  describe("Matcher 'toBeLessThan()'", function(){
2      var p=3.1415923, e=2.78;
3
4      it("Teste 1", function(){
5          expect(e).toBeLessThan(p);
6      });
7
8      it("Teste 2", function(){
9          expect(p).toBeLessThan(e);
10     });
11 });
```

MATCHER → TOBEGREATERTHAN( )

- O expect(x).toBeGreaterThan(y) passa no teste se o valor x for maior que o valor y.
- Exemplo:

```
1  describe("Matcher 'toBeGreaterThan()'", function(){
2      var p=3.1415923, e=2.78;
3
4      it("Teste 1", function(){
5          expect(p).toBeGreaterThan(e);
6      });
7
8      it("Teste 2", function(){
9          expect(e).toBeGreaterThan(p);
10     });
11 });
```

Matcher → toThrow( )

- O expect(x).toThrow( ) passa no teste se a função x laça uma exceção.
- Exemplo:

```
1  describe("Matcher 'toThrow()'", function(){
2      var fun1=function(){return 3 + 9};
3      var fun2=function(){return a + 4};
4
5      it("Teste 1", function(){
6          expect(fun1).toThrow();
7      });
8
9      it("Teste 2", function(){
10         expect(fun2).toThrow();
11     });
12 });
```

Mais um Exemplo

- Crie um objeto Pessoa:
  - Atributos:
    - p_nome, s_nome, idade, cor de olho
  - Métodos:
    - Para retornar o nome
    - Para retornar o nome mais idade
    - Para retornar o nome mais cor de olho

Mais um Exemplo

- ..//Projeto/js/**pessoa.js**

```
1  function Pessoa(_p_nome, _s_nome, _idade, _cor_olho){
2      var p_nome = _p_nome;
3      var s_nome = _s_nome;
4      var idade = _idade;
5      var cor_olho = _cor_olho;
6
7      this.nome = function(){
8          return p_nome + " " + s_nome;
9      };
10
11     this.nome_idade = function(){
12         return p_nome + " tem idade igual a " + idade;
13     };
14
15     this.nome_cor_olho = function(){
16         return p_nome + " tem olho com cor " + cor_olho;
17     };
18 }
```

Mais um Exemplo

- ..//Projeto/js/spec/**pessoa.spec.js**

```
1  describe("Pessoa", function(){
2
3      var pessoa = new Pessoa("Elder", "Pereira", 27, "Castanhos");
4
5      it("Método - nome", function(){
6          expect("Elder Pereira").toBe(pessoa.nome());
7      });
8
9      it("Método - nome_idade", function(){
10         expect("Elder tem idade igual a 27").toBe(pessoa.nome_idade());
11     });
12
13     it("Método - nome_cor_olho", function(){
14         expect("Elder tem olho com cor Castanhos").toBe(pessoa.nome_cor_olho());
15     });
16 });
```

Fim

Referências

- Jasmine: Automatização de Testes para JavaScript. Disponível em: http://ericdouglas.github.io/jasmine-br-docs/

- Teste unitário do JavaScript com Jasmine. Disponível em: http://imasters.com.br/front-end/javascript/teste-unitario-do-javascript-com-jasmine-parte-01/?trace=1519021197&source=single