

JavaScript

CLASSES





Classes
RegExp

Classes

As classes são um tipo especial de função que atuam como um template para a criação de objetos

Autores como Douglas Crockford acha desnecessário o uso de classe em Javascript, pois já existem as funções e herança baseada em protótipo



```
//Class declaration - Maiúsculo em Classes
```

```
class Square {  
  }
```

```
console.log(Square);
```

```
// Outra forma de criar classes é por meio de Class expression
```

```
const Square = class {  
  }
```

```
console.log(Square);
```

```
// Classe, nada mais é que uma função
```

```
const Square = class {  
  }
```

```
console.log(typeof Square);
```

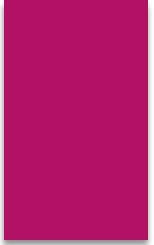
Classes

As classes são formadas por 3 tipos de membros: constructor, prototype methods e static methods

Classes constructor

O **constructor** é invocado no momento da instanciación de uma classe e serve para inicializar um determinado objeto

```
// Método Construtor
class Square {
    constructor() {
    }
}
const square = new Square();
console.log(square);
// Quando instanciar o construtor será evocado
// todos os parâmetros passados serão recebidos pelo Construtor
```



```
// Vamos passar o lado do quadrado (4)  
// É muito similar a função construtora
```

```
class Square {  
    constructor(side) {  
        this.side = side;  
    }  
} const square = new Square(4);  
console.log(square);
```

```
// ele retorna um objeto que tem a propriedade de chave side com valor 4
```

Classes prototype methods

Os **prototype methods** dependem de uma instância para serem invocados

```
// Prototype methods
class Square {
    constructor(side) {
        this.side = side;
    }

    toString() { // método toString está no protótipo
        return `side: ${this.side}`; // return de template literals
    }
}

const square = new Square(4);
// to string não está lá pois é um ptototype method (prototipo)
// Se evocado aparece
console.log(square.toString());
```


// Vamos criar calculaArea para calcular o quadrado // outro
prototype method

```
class Square {  
  constructor(side) {  
    this.side = side;  
  }  
  
  calculateArea() {  
    return Math.pow(this.side, 2); //lados elevados a potencia 2  
  }  
  
  toString() {  
    return `side: ${this.side} area: ${this.calculateArea()}`;  
    // calculo da área  
  }  
}  
  
const square = new Square(4);  
console.log(square.toString());
```

Classes static methods

Os **static methods** não dependem de uma instância para serem invocados

```
// static methods
```

```
class Square {  
    constructor(side) {  
        this.side = side;  
    }  
  
    calculateArea() {  
        return Math.pow(this.side, 2);  
    }  
  
    toString() {  
        return `side: ${this.side} area: ${this.calculateArea()}`;  
    }  
    static fromArea(area) {  
        // não pertence a instancia, pertence a classe  
        return new Square(Math.sqrt(area));  
    }  
}  
  
const square = Square.fromArea(16);  
// Não vamos usar new Square, Vamos criar fromArea como static  
console.log(square.toString());  
console.log(square.calculateArea());
```

As classes funcionam de forma similar as funções construtoras

```
// Similaridade entre classes e funções construtoras
function Square(side) {
    this.side = side;
}
Square.prototype.calculateArea = function() {
    return Math.pow(this.side, 2);
}
Square.prototype.toString = function() {
    return `side: ${this.side} area: ${this.calculateArea()}`;
}
Square.fromArea = function(area) {
    return new Square(Math.sqrt(area));
}
const square = Square.fromArea(16);
console.log(square.toString());
```

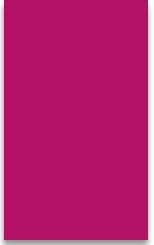
Extends – hierarquia de classes

É possível criar uma hierarquia de classes por meio da palavra-chave **extends**

Ao declarar um construtor na subclass é necessário **invocar o construtor da superclass** por meio `super()` antes de utilizar a referência `this`

RegExp

As expressões regulares são estruturas formadas por uma **sequência de caracteres que especificam um padrão formal** que servem para validar, extrair ou mesmo substituir caracteres dentro de uma String



```
// Expressões Regulares - São padrões  
// Utilizado para validar campos (senhas etc)  
// Extrair caracteres de dentro de uma String  
// Substituição de caracteres em lotes  
// Entre barras é expressão regular - é object  
// mais usual - aconselhado
```

```
/ehgomes@ifsp.edu.br/;
```

```
// Podemos também fazer com new RegExp
```

```
new RegExp("ehgomes@ifsp.edu.br");
```

```
// Vamos testar se o padrão é o mesmo
let regExp = /ehgomes@ifsp.edu.br/;
let result = regExp.test("ehgomes@ifsp.edu.br"); // Se não encontrar?
console.log(result);

// exec - booleano - retorna um array
let regExp = /ehgomes@ifsp.edu.br/;
let result = regExp.exec("ehgomes@ifsp.edu.br"); console.log(result);

// exec - Mostra mais detalhes
// Vamos mostrar o resultado, o índice e a entrada
let regExp = /ehgomes@ifsp.edu.br/;
let result = regExp.exec("ehgomes@ifsp.edu.br");
console.log(result[0]);
console.log(result.index);
console.log(result.input);

// com mais de uma entrada
let regExp = /ehgomes@ifsp.edu.br/;
let result = regExp.exec("E-mail: ehgomes@ifsp.edu.br, E-mail secundário: ehgomes@ifsp.edu.br");
console.log(result[0]);
console.log(result.index);
console.log(result.input);
```


Metacaracteres

- Ponto representa qualquer caractere
- \ A barra é utilizada antes de caracteres especiais, com o objetivo de escapá-los
- ^ Inicia com um determinado caractere
- \$ Finaliza com um determinado caractere

```
// Metacaracteres - Ponto representa qualquer tipo de caracter
// NESTE CASO ELE ENCONTRA!!!!
let regExp = /ehgomes@ifsp.edu.br/;
let result1 = regExp.exec("ehgomes@ifspXeduXbr");
console.log(result1);

// Vamos escapar esses caracteres
let regExp = /ehgomes@ifsp\.edu\.br/;
let result1 = regExp.exec("ehgomes@ifsp.edu.br");
console.log(result1);

// não encontra mais
let result2 = regExp.exec("ehgomes@ifspXeduXbr");
console.log(result2);

// ^ Inicialização e $ Finalização
let regExp = /^ehgomes@ifsp\.edu\.br$/;
let result1 = regExp.exec("Email: ehgomes@ifsp.edu.br");

console.log(result1);
let result2 = regExp.exec("ehgomes@ifsp.edu.br");

console.log(result2);
let result3 = regExp.test("ehgomes@ifsp.edu.br"); // com test
console.log(result3);
```

Grupo de caracteres

- [abc]** Aceita qualquer caractere dentro do grupo, nesse caso a, b e c
- [^abc]** Não aceita qualquer caractere dentro do grupo, nesse caso a, b ou c
- [0-9]** Aceita qualquer caractere entre 0 e 9
- [^0-9]** Não aceita qualquer caractere entre 0 e 9

Quantificadores

Os quantificadores podem ser aplicados a caracteres, grupos, conjuntos ou metacaracteres

- `{n}` Quantifica um número específico
- `{n,}` Quantifica um número mínimo
- `{n,m}` Quantifica um número mínimo e um número máximo
- `?` Zero ou um
- `*` Zero ou mais
- `+` Um ou mais



```
// Grupos de caracteres
```

```
let regExp = /^[a-z]+@[a-z]+\.[a-z]{3}$/;  
let result = regExp.exec("gomeseh@hotmail.com");  
console.log(result);
```

```
// dominio com 2 caracteres
```

```
let regExp = /^[a-z]+@[a-z]+\.[a-z]{3}$/;  
let result = regExp.exec("gomeseh@hotmail.br");  
console.log(result);
```

Metacaracteres 2

<code>\w</code>	Representa o conjunto [a-zA-Z0-9_]
<code>\W</code>	Representa o conjunto [^a-zA-Z0-9_]
<code>\d</code>	Representa o conjunto [0-9]
<code>\D</code>	Representa o conjunto [^0-9]
<code>\s</code>	Representa um espaço em branco
<code>\S</code>	Representa um não espaço em branco
<code>\n</code>	Representa uma quebra de linha
<code>\t</code>	Representa um tab

Grupos de Captura

() Determina um grupo de captura para realizar a extração de valores de uma determinada String

Modificadores

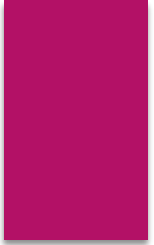
- I Case-insensitive matching
- G Global matching
- M Multiline matching

```
// dominio com 2 caracteres
let regexp = /^[a-z]+@[a-z]+\.[a-z]{3}$/;
let result = regexp.exec("gomeseh@hotmail.br");
console.log(result);

// Metacaracteres
let regexp = /^\\w+@\\w+(\\.\\w{2,3})+$/;
let result = regexp.exec("mary@hotmail.com.br");
console.log(result);

// Grupos de captura - Extração de valores
// Vamos extrair nome de usuário + dominio
let regexp = /([a-z]+)@([\\.a-z]+)/;
let result = regexp.exec("mary@hotmail.com");
console.log(result[0]);
console.log(result[1]);
console.log(result[2]);
console.log(result.index);
console.log(result.input);

// Metacaracteres let regexp = /^(\\w+)@(\\w+) (\\.\\w{2,3})+$/;
let result = regexp.exec("mary@hotmail.com.br");
console.log(result[0]);
console.log(result[1]);
console.log(result[2]);
```

```
// Modificadores
```

```
let regExp = /[a-z]+@[\.a-z]+/g;
```

```
let result = regExp.exec("mary@hotmail.com;gomeseh@hotmail.com");
```

```
console.log(result[0]);
```

```
console.log(result.index);
```

```
let result2 = regExp.exec("mary@hotmail.com;gomeseh@hotmail.com");
```

```
console.log(result2[0]);
```

```
console.log(result2.index);
```

```
// i - case sensitive
```

```
let regExp = /[a-z]+@[\.a-z]+/g; //i
```

```
let result = regExp.exec("Mary@hotmail.com;gomeseh@hotmail.com");
```

```
console.log(result[0]);
```

```
console.log(result.index);
```