ECMAScript 6

Rafael Escalfoni

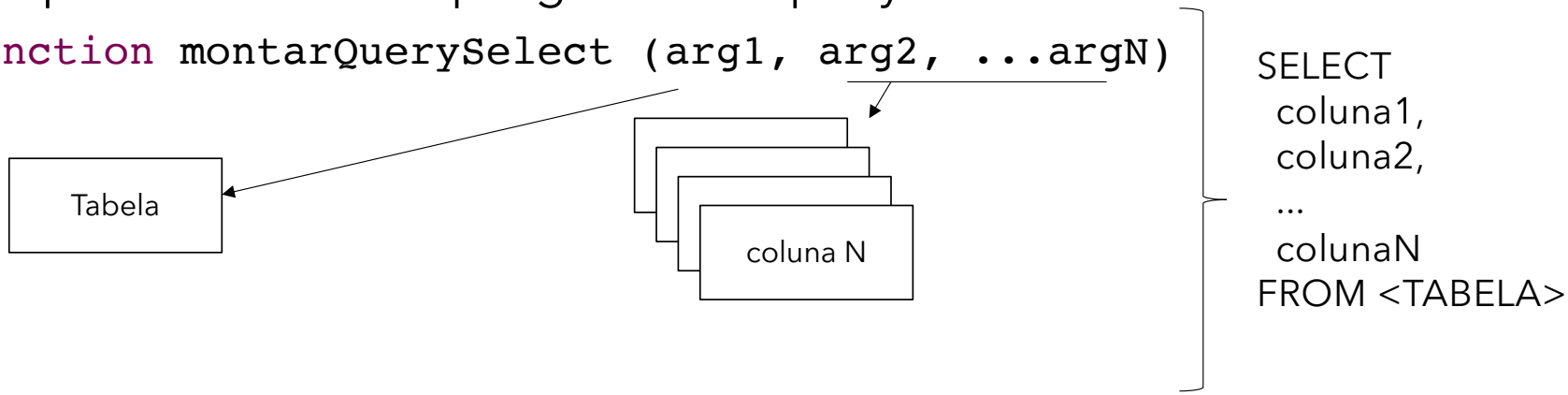
adaptado de ECMAScript6 – Entre de cabeça no futuro do JavaScript

Parâmetros infinitos com operador REST

Parâmetros Infinitos???

- Em situações em que temos um número desconhecido de parâmetros
- Exemplo: um método que gera uma query

```
function montarQuerySelect (arg1, arg2, ...argN)
```



Objeto arguments

```
function montarQuerySelect (arg1, arg2, ...argN) {
    const tabela = arguments[0];
    const qtdArgs = arguments.length;
    let cols = '';
    if (qtdArgs > 1) {
        for (let index = 1; index < qtdArgs; index++) {
            cols += `${arguments[index]}, `;
        }
        cols = cols.substring(0, cols.length - 2);
    } else {
        cols = '*';
    }
    return `SELECT ${cols} from ${tabela}`;
}
```

Objeto arguments

- Quando invocado dentro de uma função, retorna um objeto com uma referência para todos os argumentos passados para o contexto de execução da função

```
function listarTodosArgumentos() {
    for (let i = 0; i < arguments.length; i++) {
        console.log(arguments.[i]);
    }
}

listarTodosArgumentos(1, 2, 3) // 1, 2, 3
```

Objeto arguments vs Operador REST

```
function somar(...valores) {
    let soma = 0;
    const qtd = valores.length;
    for (let i = 0; i < qtd; i++) {
        soma += valores[i];
    }
    return soma;
}

console.log(somar(1,2)); // 3
console.log(somar(1,2,3)); // 6
console.log(somar(1,2,3,4)); // 10
```

Objeto arguments vs Operador REST

- arguments não é um array
- rest é um array: podemos usar todos os métodos auxiliares: forEach, map, filter, find, etc

```
function somar(...valores) {
    return valores.reduce((soma, valor) => {
        return soma + valor;
    }, 0);
}

console.log(somar(1,2)); // 3
console.log(somar(1,2,3)); // 6
console.log(somar(1,2,3,4)); // 10
```

Operador REST

- Só podemos usar um operador por função

```
function funcQualquer(...numeros, ...letras) {
    // corpo da função
    // não funciona :-P
}
```

Operador REST

- Só podemos usar um operador por função

```
function somaTudoEMultiplicaPor(multiplicador, ...valores) {
    return valores.reduce((soma, valor) => {
        return soma + (valor * multiplicador);
    }, 0);
}

console.log(somaTudoEMultiplicaPor(2,1,2)); // 6
console.log(somaTudoEMultiplicaPor(2,6,7)); // 26
```

Operador Spread

• Uma outra forma de chamar funções do ES5 com call:
minhaFuncao.call(contexto, arg1, arg2, ...);

```
function getIdade(){
    return this.idade;
}
var pessoa = {
  idade: 10
};

console.log(getIdade.call(pessoa); // 10
```

Operador Spread

• Uma outra forma de chamar funções do ES5 com apply:
minhaFuncao.apply(contexto, args); //args é um array

```
function getIdade(){
    return this.idade;
}
var pessoa = {
  idade: 10
};

console.log(getIdade.apply(pessoa)); //  10
```

Esquema do apply

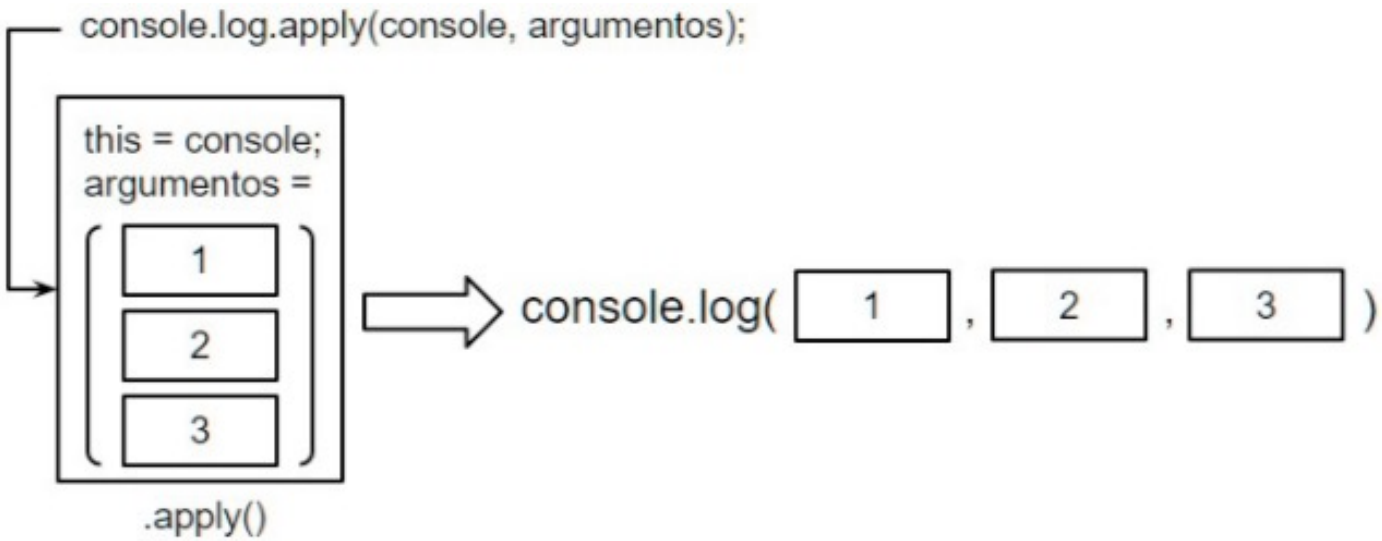


Figura 14.1: Método apply em ação

```
var argumentos = [1,2,3];
console.log.apply(console, argumentos);
// 1, 2, 3
```

Mas e o Spread???

- No ECMAScript 6, criaram uma alternativa otimizada.
  - Tiramos o apply/call e inserimos como param da função os ... acompanhado de um array com parâmetros

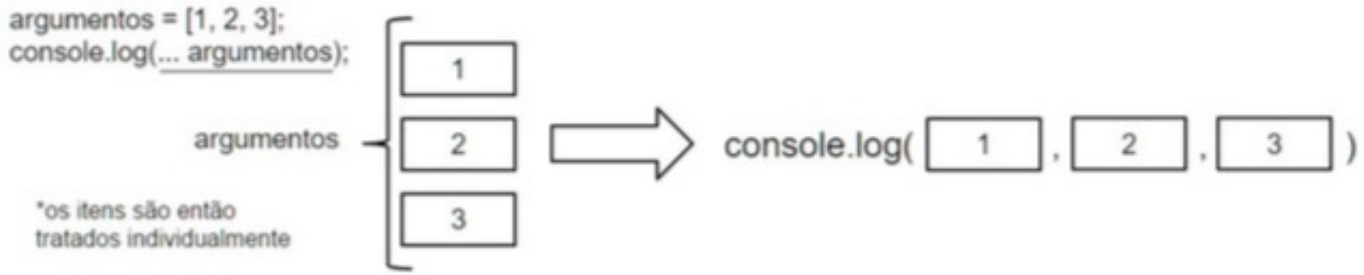


Figura 14.2: Operador Spread em ação

Aplicando o código, temos:

```
const argumentos = [1,2,3];
console.log(...argumentos) // 1, 2, 3
```

Exemplo

```
function soma(a,b) {
    console.log(a + b);
}
soma(1,2); // 3

var numeros = [1,2];
soma(...numeros); // 3
```

Exemplo

```
function contaVogaisNaoAcent(palavra) {
    let qtdVogais = 0;
    const palavraLowerCase = palavra.toLowerCase();
    const letras = [...palavraLowerCase];
    for (let letra of letras){
        if (“aeiou”.indexOf(letra) !== -1){
            qtdVogais++;
        }
    }
    return qtdVogais;
}
contaVogaisNaoAcent(‘ecmascript’); // 3
```