

Objetos em JavaScript

Objetos em JavaScript – Operações Avançadas

Este conjunto de slides apresenta recursos avançados de manipulação de objetos em JavaScript:

↗ exclusão de propriedades

↗ testes

↗ enumeração

↗ extensão

↗ serialização

↗ sintaxes estendidas.

Remover propriedades

- O operador `delete` remove uma **propriedade** de um objeto (não o valor diretamente).
- Só remove propriedades **próprias** (own properties), nunca herdadas.
- Retorna `true` se a exclusão foi bem-sucedida ou não teve efeito.
- Não remove propriedades com o atributo `configurable: false`.

```
let obj = {a: 1, b: 2};
delete obj.a;    // true
console.log(obj) // { b: 2 }

let o = Object.freeze({x: 10});
delete o.x; // false (não pode apagar, não é configurável)
```

Testar propriedades

- Objetos podem ser vistos como conjuntos de propriedades. Teste de propriedades:
 - `in` verifica se existe no objeto (própria ou herdada).
 - `hasOwnProperty()` verifica se é **own property**.
 - `propertyIsEnumerable()` verifica se é **própria e enumerável**.

```
let obj = {a: 1};  
console.log("a" in obj); // true  
console.log(obj.hasOwnProperty("a")); // true  
console.log(obj.propertyIsEnumerable("a")); // true
```

- `in` distingue entre propriedades inexistentes e propriedades definidas como **undefined**.

Enumerando propriedades

~ O `for...in` percorre propriedades **enumeráveis** (próprias ou herdadas).

~ Propriedades herdadas de `Object.prototype` **não são enumeráveis**.

Funções úteis:

~ `Object.keys(obj)` → propriedades próprias enumeráveis.

~ `Object.getOwnPropertyNames(obj)` → inclui não enumeráveis.

~ `Object.getOwnPropertySymbols(obj)` → retorna propriedades do tipo `Symbol`.

~ `Reflect.ownKeys(obj)` → retorna **todas** as próprias (strings e `Symbol`).

```
let obj = {a: 1, b: 2};  
console.log(Object.keys(obj)); // ["a", "b"]
```

Extendendo Objetos

➤ Operação comum: copiar propriedades de um objeto para outro.

➤ Em ES6, `Object.assign()` faz isso nativamente.

```
let defaults = {a: 1, b: 2};  
let config = {b: 3, c: 4};  
let result = Object.assign({}, defaults, config);  
  
console.log(result); // {a:1, b:3, c:4}
```

➤ Útil para aplicar valores padrão quando uma propriedade não existe.

➤ Neste exemplo, a propriedade de `b` de `config` sobreescreve a propriedade `b` de `defaults`.

Serializando objetos

~ Serialização: transformar o estado de um objeto em string. Funções:

~ `JSON.stringify()` → objeto → string.

~ `JSON.parse()` → string → objeto.

~ Usa formato JSON (JavaScript Object Notation).

Limitações:

~ `NaN`, `Infinity`, `-Infinity` → convertidos em `null`.

~ `Function`, `RegExp`, `Error`, `undefined` não podem ser serializados.

```
let obj = {x: 10, y: 20};  
let str = JSON.stringify(obj); // '{"x":10,"y":20}'  
let newObj = JSON.parse(str);
```

Métodos de Objeto

Todos os objetos herdam de `Object.prototype`.

`toString()`

- ↗ Retorna uma string representando o objeto.
- ↗ Pode ser sobrescrito (ex.: `Date`, `Number`).

```
let obj = {a:1};  
console.log(obj.toString()); // "[object Object]"
```

Métodos de Objeto

valueOf()

↗ Converte o objeto para um valor primitivo.

↗ `Date.valueOf()` → número (timestamp).

```
let d = new Date();  
console.log(d.valueOf()); // número de ms desde 1970
```

toJSON()

↗ Usado automaticamente por `JSON.stringify()` se estiver definido no objeto.

Sintaxe de objeto literal estendida

Spread Operator (...)

- ↗ Copia propriedades de um objeto para outro dentro de literais.
- ↗ Propriedades duplicadas: prevalece a última.

```
let obj1 = {a:1, b:2};  
let obj2 = {b:3, c:4};  
let merged = {...obj1, ...obj2};  
console.log(merged); // {a:1, b:3, c:4}
```

Abreviação de nomes de métodos

Forma tradicional:

```
let obj = {  
  area: function(w,h){ return w*h; }  
};
```

Forma reduzida (ES6+):

```
let obj = {  
  area(w,h) { return w*h; }  
};
```

Propriedades Getters and Setters

- *Acessor properties* não armazenam valor diretamente.
- Definidas com `get` e/ou `set`. Alternativa de acesso as propriedades do objeto.

```
let pessoa = {  
  _nome: "Ana",  
  get nome() { return this._nome; },  
  set nome(n) { this._nome = n.toUpperCase(); }  
};  
  
console.log(pessoa.nome); // "Ana"  
pessoa.nome = "maria";  
console.log(pessoa.nome); // "MARIA"
```

- Apenas `get` → somente leitura.
- Apenas `set` → somente escrita.