Manipulação e Extensão de Objetos em JavaScript

Em JavaScript, objetos são estruturas fundamentais que agrupam propriedades e métodos, permitindo a modelagem de dados e comportamentos. Este capítulo explora operações essenciais com objetos, como exclusão, teste, enumeração, extensão, serialização, além de métodos e recursos avançados da sintaxe literal de objetos.

1. Deletando Propriedades

O operador `delete` remove **propriedades próprias** de um objeto, e não atua sobre os valores em si, mas sobre a existência da propriedade.

```
let pessoa = { nome: "Ana", idade: 30 };
delete pessoa.idade;

console.log(pessoa); // { nome: "Ana" }
```

Alguns pontos importantes:

- Só remove **propriedades próprias**, não herdadas.
- Avalia para `true` se a exclusão foi bem-sucedida ou se não teve efeito.
- Não remove propriedades com o atributo `configurable: false`.

2. Testando Propriedades

Objetos podem ser vistos como conjuntos de propriedades. É comum precisar verificar se uma propriedade existe.

- **Operador `in`**: testa se a propriedade está presente no objeto (própria ou herdada).

```
console.log("nome" in pessoa); // true
console.log("idade" in pessoa); // false
```

- `hasOwnProperty()`: verifica se a propriedade é própria do objeto.

```
pessoa.hasOwnProperty("nome"); // true
```

- `propertyIsEnumerable()`: semelhante a `hasOwnProperty`, mas só retorna `true` se a propriedade for enumerável.

> Observação: o `in` consegue distinguir propriedades inexistentes de propriedades definidas com `undefined`.

3. Enumerando Propriedades

O laço `for/in` percorre todas as propriedades enumeráveis (próprias e herdadas) de um objeto.

```
for (let prop in pessoa) {
  console.log(prop);
}
```

Além disso, existem métodos que retornam **arrays de nomes de propriedades**:

- `Object.keys(obj)` → retorna as propriedades enumeráveis próprias.
- `Object.getOwnPropertyNames(obj)` → retorna todas as propriedades próprias, incluindo não enumeráveis.
- `Object.getOwnPropertySymbols(obj)` → retorna propriedades próprias cujo nome é um `Symbol`.
- `Reflect.ownKeys(obj)` → retorna todas as propriedades próprias (enumeráveis, não enumeráveis e `Symbols`).

4. Estendendo Objetos

É comum precisar copiar propriedades de um objeto para outro.

- `Object.assign()` **(ES6)** copia propriedades próprias enumeráveis de objetos de origem para um objeto de destino.

```
let destino = { a: 1 };
let origem = { b: 2 };
Object.assign(destino, origem);

console.log(destino); // { a: 1, b: 2 }
```

Esse recurso é útil, por exemplo, para aplicar valores padrão em objetos.

5. Serializando Objetos

Serializar um objeto significa convertê-lo em uma string para armazenamento ou transmissão.

- `JSON.stringify()` → converte objeto em string JSON.
- `JSON.parse()` → reconstrói o objeto a partir da string.

```
let obj = { nome: "Carlos", idade: 25 };
let json = JSON.stringify(obj);
let copia = JSON.parse(json);

console.log(json);  // '{"nome":"Carlos","idade":25}'
console.log(copia); // { nome: "Carlos", idade: 25 }
```

Limitações:

- `NaN`, `Infinity` e `-Infinity` viram `null`.
- Funções, `RegExp`, `Error` e `undefined` não são serializados.
- Apenas propriedades próprias enumeráveis são incluídas.

6. Métodos de Objetos

Todos os objetos em JavaScript herdam métodos de `Object.prototype`:

- **`toString()`**: retorna uma representação em string do objeto.

```
console.log({}.toString()); // "[object Object]"
```

- **`valueOf()`**: retorna o valor primitivo associado ao objeto (usado em operações matemáticas).

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

- **`toJSON()`**: não está em `Object.prototype`, mas o `JSON.stringify()` o procura. Objetos podem definir seu próprio `toJSON()`.

7. Sintaxe Estendida de Literais de Objeto

a) Spread Operator (`...`)

Permite copiar propriedades de objetos de forma simples:

```
let base = { a: 1, b: 2 };
let copia = { ...base, c: 3 };

console.log(copia); // { a: 1, b: 2, c: 3 }
```

Se houver conflito de nomes, prevalece a última definição.

b) Métodos Shorthand

Desde o ES6, funções dentro de objetos podem ser declaradas sem a palavra `function`.

```
let circulo = {
  raio: 5,
  area() {
    return Math.PI * this.raio ** 2;
  }
```

```
};
console.log(circulo.area()); // 78.53
```

c) Getters e Setters

Permitem definir **propriedades acessoras**, que são métodos especiais usados para ler e escrever valores.

```
let pessoa2 = {
  _nome: "Maria",
  get nome() { return this._nome; },
  set nome(n) { this._nome = n.toUpperCase(); }
};

pessoa2.nome = "joão";
console.log(pessoa2.nome); // "JOÃO"
```

- Só getter → propriedade somente leitura.
- Só setter → propriedade somente escrita.
- São herdados como propriedades normais.