TIPOS

THIAGO DELGADO PINTO
thiago_dp (at) yahoo (dot) com (dot) br

versão: 2020.01.20

resumo

tipos primitivos	tipos invólucros	outros
`string`	`String`	
`number`	`Number`	
`bigint`	`BigInt`	
`boolean`	`Boolean`	
`null`	-	
`undefined`	-	
`symbol`	`Symbol`	
		`object`

ES6+ 

tipos primitivos

não são objetos – não possuem métodos

são imutáveis – valor não pode ser alterado

são 7:

- `string`
- `number`
- `bigint`
- `boolean`
- `null`
- `undefined`
- `symbol` (ES6+)

tipo primitivo `string`

texto de tamanho arbitrário

aceita valores envoltos por aspas (") ou apóstrofo ('), sem distinção

'texto'

"texto"

"中文 español English देवनागरी العربية português"

tipo primitivo `string`

escape de caracteres com contrabarra (`\`)

aspas (`\"`), apóstrofo (`\'`), nova linha (`\n`), retorno de cursor (`\r`), tab vertical (`\v`), tab (`\t`), backspace (`\b`), alimentação de formulário (`\f`)

escape de caracteres ISO-8859-1, UTF-8, UTF-16 e UTF-32

introduzido no JavaScript 6+

`\XXX`, onde `XXX` são 3 dígitos em base octal, de `0` a `377`

`\xXX`, onde XX são 2 dígitos em base hexadecimal, de 0 a `FF`

`\uXXXX`, onde `XXXX` são 4 dígitos em base hexadecimal, de 0 a `FFFF`

`\u{X}` a `\u{XXXXXX}`, onde cada `X` é um dígito (logo, 1-6), variando de `0` a `10FFFF`

tipo primitivo **number**

números flutuantes de 64-bit, de $-(2^{53}-1)$ a $2^{53}-1$ e

admite os valores **+Infinity**, **-Infinity** e **NaN** (*Not a Number*).

```
ex. a =  50 /  0;  // +Infinity
ex. b = -50 / 0;  // -Infinity
ex. c =  50 / 'a'; // NaN
```

Valores mínimo e máximo são representados, respectivamente, pelas constantes **Number.MIN_VALUE** e **Number.MAX_VALUE**.

tipo primitivo `bigint`

inteiros com precisão arbitrária (sem limite)

Valores mínimo e máximo são representados, respectivamente, pelas constantes `Number.MIN_SAFE_INTEGER` e `Number.MAX_SAFE_INTEGER`.

tipo primitivo `boolean`

admite os valores `true` e `false`

valores que também avaliam para `false`:

```
0
-0
NaN
null
undefined
""
```

tipo primitivo `null`

indica um valor nulo/vazio, como um objeto inexistente

admite somente `null`

tipo primitivo `undefined`

indica *ausência de qualquer valor*, inclusive **`null`**

admite somente **`undefined`**

é o valor padrão de variáveis não inicializadas

tipo primitivo `symbol`

símbolo usado como valor único e imutável em propriedades de objetos

- disponível a partir do ES6

nunca irão colidir com *strings* usadas como atributos

- ou outros símbolos

exemplo

```
const nome = Symbol("nome"); // método fábrica
const pessoa = {
  [ nome ]: "Ana"
};
console.log( pessoa[ nome ] ); // Ana
console.log( pessoa[ "nome" ] ); // undefined
```

tipo primitivo `symbol`

existem símbolos padrão, usados por algumas construções do ES

podemos usá-los em nossas classes

exemplo:

```
class ColecaoNumeros {
  constructor( numeros ) { this._numeros = numeros || []; }
  adicionar( n ) { this._numeros.push( n ); }
  numeros() { return this._numeros; }
  *[ Symbol.iterator ]() { // permite uso em for...of
    for ( let n of this._numeros ) { yield n; }
  }
}
const c = new ColecaoNumeros( [ 10, 20, 30 ] );
c.adicionar( 40 );
for ( let n of c ) { console.log( n ) }; // 10 20 30 40
```

Veja mais em https://www.keithcirkel.co.uk/metaprogramming-in-es6-symbols/

invólucros (*wrappers*) de tipos primitivos

são objetos equivalentes aos tipos primitivos

todos possuem o método **`valueOf()`**, que retorna o valor primitivo

são eles:

- `String`
- `Number`
- `BigInt`
- `Boolean`
- `Symbol`

`null` e **`undefined`** não possuem invólucros

boxing e *unboxing*

boxing é a conversão de primitivo para seu invólucro

ex. `var n1 = new Number( 10 ); // Number { 10 }`

unboxing é a conversão de invólucro para primitivo

ex. `var n2 = n1.valueOf(); // 10`

o interpretador JavaScript realiza automaticamente em muitos casos

boxing e unboxing

exemplo 1

```
var s = "ecma".toUpperCase();
```

2 conversões:

1) `"ecma"` → `String { "ecma" }`

daí, chamar `toUpperCase()` faz gerar `String { "ECMA" }`

2) `String { "ECMA" }` → `"ECMA"`

exemplo 2

```
var n = Number( "500" );
```

2 conversões

1) `"500"` → `Number { 500 }`

2) `Number { 500 }` → `500`

invólucros e o uso de **new**

há diferença entre, por exemplo,

```
var n1 = new Number( 10 ); // object
var n2 = Number( 10 ); // number
```

ao usar new, dizemos explicitamente ao interpretador que queremos um objeto

ao não usar new, usando o tipo como função, resultando em um *boxing* temporário

provavelmente o interpretador fará um *unboxing* em seguida

no exemplo acima, **n1** será **Number{ 10 }** e **n2** será **10**

operador `typeof`

faz retornar o tipo de algo, como uma string
valor, variável ou expressão

retorna os tipos primitivos, **`object`** ou **`function`**

```
typeof 37 // "number"
typeof Number( 37 ) // "number"
typeof new Number( 37 ) // "object"
typeof true // "boolean"
typeof undefined  // "undefined"
typeof { "nome": "Ana" } // "object"
typeof function() {} // "function"
typeof Symbol("foo") // "symbol"
```

operador `typeof`

```
typeof [ 1, 2, 3 ] // "object"
typeof class Foo {} // "function" (ES6+)
typeof NaN // "number" ☺

typeof null // "object" (!!!)
```

sim, **`typeof null`** deveria retornar **`"null"`**, não **`"object"`**

- esse é um *bug* existente desde o JavaScript 1 *
- foi mantido por compatibilidade com versões anteriores ☹

*Saiba mais em https://2ality.com/2013/10/typeof-null.html

operador ===

é o operador de igualdade estrita

compara valor e tipo

```
5 === 5 // true
"5" === 5 // false
```

ao contrário da *igualdade* (==), não converte tipo

```
"5" === 5 // false
"5" == 5 // true
```

referências usadas

MDN. **Referência JavaScript**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference

2ality. **The history of "typeof null"**. Disponível em: https://2ality.com/2013/10/typeof-null.html

CIRKEL, Keith. **Metaprogramming in ES6: Symbols and why they're awesome**. Disponível em: https://www.keithcirkel.co.uk/metaprogramming-in-es6-symbols/

fim

Versão 1: 2020.01.20