08

Revisando arrow function e seu escopo léxico

A ideia desse exercício é deixar ainda mais clara a diferença do `this` de uma *arrow function* do `this` de uma função tradicional em JavaScript. Sugiro fortemente que você crie os arquivos em um projeto separado, para poder ver o que acontece além da teoria.

Vamos começar por um exemplo clássico. Temos três elementos distintos em nossa página e queremos exibir o conteúdo de cada um deles.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>

    <h1>TITULO</h1>
    <p>PARAGRAFO</p>
    <div>DADOS</div>

    <script>
        console.log(this); // é window

        let exibeConteudo = function() {
            console.log(this);
            alert(this.textContent);
        };

        $ = document.querySelector.bind(document);

        $('h1').addEventListener('click', exibeConteudo);

        $('p').addEventListener('click', exibeConteudo);

        $('div').addEventListener('click', exibeConteudo);

    </script>
</body>
</html>
```

Perfeito, quando clicamos em cada um deles, exibimos no console o valor de `this`, inclusive exibimos um alerta com conteúdo de cada elemento. Repare que o `this` é dinâmico, ou seja, **seu valor é definido no momento em que a função é chamada, jamais no momento em que é declarada**. Quando clicamos no `h1`, o `this` será este elemento, quando clicamos em `p`, o `this` será o elemento. Ainda bem que isso acontece, pois se o `this` não fosse dinâmico, não conseguiríamos escrever uma função genérica como a nossa.

Que tal declararmos nossa função como uma *arrow function*, que é menos verbosa? Alterando nosso código:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>

    <h1>TITULO</h1>
    <p>PARAGRAFO</p>
    <div>DADOS</div>

    <script>
        console.log(this); // é window

        // arrow function agora!
        let exibeConteudo = () =>  {
            console.log(this);
            alert(this.textContent);
        };

        $ = document.querySelector.bind(document);

        $('h1').addEventListener('click', exibeConteudo);

        $('p').addEventListener('click', exibeConteudo);

        $('div').addEventListener('click', exibeConteudo);

    </script>
</body>
</html>
```

Um teste demonstra que nosso código deixa de funcionar. Primeiro, independente do elemento que eu clique, o `this` que é impresso no console é `window` e não aquele elemento do DOM. Segundo, como `this` é `window` e ele não possui a propriedade `textContent`, é exibido `undefined` para o usuário. Esse problema serve para demonstrar que uma *arrow function* vai além de uma sintaxe mais enxuta para declararmos funções.

Diferente de uma função, que possui um `this` dinâmico, uma *arrow function* possui um `this` estático, ou seja, que nunca muda e que é determinado no momento em que é declarado! Veja que quando declararmos nossa *arrow function*, ela vai considerar o `this` do local onde é declarada. Sendo assim, como o `this` dentro da tag `<script>` é `window`, ela adotará `window`.

Resumindo:

- O `this` de uma função é dinâmico, isto é, seu valor é determinado no momento em que a função é chamada. Como o `this` é dinâmico, é possível usar artifícios da linguagem, como a API `Reflect`, para alterá-lo se assim desejarmos.
- O `this` de uma *arrow function* é léxico, isto é, seu valor é determinado no local onde a *arrow function* for definida, ela não cria um novo `this`. O `this` de uma *arrow function* não pode ser alterado, mesmo se usarmos recursos da linguagem, como a API `Reflect`.

No contexto que vimos acima, a *arrow function* atrapalhou mais do que ajudou. Mas vejamos um exemplo em que seu escopo léxico torna-se **muito** interessante:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>

    <script>

        class SistemaImpressao {

            constructor() {
                this._codigo = 2;
            }

            imprime(nomes) {

                nomes.forEach(function(nome) {
                    console.log(this);
                    console.log(`${this._codigo}: ${nome}`);
                });
            }
        }

        let nomes = ['Flávio', 'Nico', 'Douglas'];
        let si = new SistemaImpressao();
        si.imprime(nomes);

    </script>
</body>
</html>
```

Temos a seguinte classe `SistemaImpressao`, que possui o método `imprime`. O método recebe uma lista e para cada item da lista imprime primeiro a versão do sistema, seguido do item. O problema é que o `this._codigo` acessado em nosso `forEach` não é de uma instância da classe `SistemaImpressao`, aliás, ele é `undefined`. Contudo, se usarmos *arrow function*, o `this` usado no `forEach` usará o `this` do contexto no qual foi declarado.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>

    <script>

        class SistemaImpressao {

            constructor() {
```

```
            this._codigo = 2;
        }

        imprime(nomes) {
            // usando arrow function.
            nomes.forEach(nome => {
                console.log(this);
                console.log(`${this._codigo}: ${nome}`);
            });
        }
    }

    let nomes = ['Flávio', 'Nico', 'Douglas'];
    let si = new SistemaImpressao();
    si.imprime(nomes);

</script>
</body>
</html>
```

Agora nosso código funciona!