Funções imediatas JavaScript (IIFE)

Por[Pedro Araujo](https://imasters.com.br/perfil/pedroaraujo/?trace=3182418120&source=single) em 19/01/2015

Ver também: <http://desenvolvimentoparaweb.com/javascript/javascript-iife-conteiner-de-codigos/>

Há tempos venho aprendendo e tentando entender como alguns truques em Javascript funcionam. O IIFE é um desses caras que muitas vezes usamos e não sabemos o que acontece realmente. Esse trecho de código é bastante usado no Javascript funcional para resolver alguns problemas. Você pode vê-lo a seguir:

|  |  |
| --- | --- |
| 1 | (function(){})(); |
|  |  |

O IIFE significa “Immediately-invoked function expression”, mas podemos chamá-lo de função imediata. Como o próprio nome diz, ela executa a função imediatamente depois de criada. Mas por que usar? Encapsulamento! Tenha em mente que variáveis em Javascript têm como escopo a função pela qual elas foram definidas (podem ser acessadas somente dentro da função, jamais fora). Ao criar uma função anônima com execução imediata, podemos criar um escopo temporário para nossas funções e variáveis. Com isso, evitamos poluição no nosso escopo global e possíveis conflitos de variáveis ou funções com o mesmo nome.

Exemplo: considere o código a seguir

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | var adder = (function() { | | |
| 2 | | var myPhrase = ""; |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | return function(x) { | | |
| 4 | | return myPhrase = |

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | !!myPhrase ? myPhrase.concat(" ", x) : myPhrase.concat(x); | | |
| 6 | | } |

|  |  |  |  |
| --- | --- | --- | --- |
| 7 | })(); | | |
| 8 | |  |

|  |  |  |
| --- | --- | --- |
| 9 | adder("Olá"); // "Olá" | |
| 10 | | adder("Mundo!"); // "Olá Mundo!" | |

|  |  |  |
| --- | --- | --- |
| 11 | |  |
| 12 | myPhrase; // myPhrase is not defined | | |

Neste exemplo, criamos uma função anônima imediata que retorna uma outra função que concatena uma string na variável chamada myPhrase. Note que a variável criada myPhrase está no escopo da IIFE e não na função retornada. Portanto, o myPhrase não é definido toda vez que invocamos a função adder. Mas o mais importante disso tudo é que o myPhrase está limitado a escopo da função anônima imediata, não permitindo o seu acesso direto de maneira alguma.

Ok, agora que vimos como funciona uma IIFE, vamos entender sua sintaxe.

Sintaxe da IIFE

Vamos analisar essa estrutura de parênteses.

|  |  |
| --- | --- |
| 1 | (...)(); |

É confuso tentar entender a sintaxe de uma IIFE que possui parênteses com comportamentos tão diferentes. Por isso, vamos devagar.

Sabemos que podemos definir uma função assim: function doSomething() { /\* codigo \*/ }

Até aqui está lindo, mas e se eu colocar o conjunto de parênteses () para invocar a função enquanto definimos. Aí temos a função imediata, certo?

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | function doSomething() { /\* codigo \*/ }(); // SyntaxError: Unexpected token ) | | |
| 2 | |  |

|  |  |  |
| --- | --- | --- |
| 3 | | // Agora com função anônima |
| 4 | function() { /\* codigo \*/ }(); // SyntaxError: Unexpected token ( | | |

Errado! Como você pode ver, se tentamos invocar uma função enquanto definimos, temos um erro.

De acordo com a norma do ExpressionStatement (estado de expressão), não podemos começar com a keyword function, pois quando usada em escopo global ou dentro de outra função, temos um estado de declaração de uma função ([Function Declaration](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function)). Portanto, se tentarmos definir uma IIFE da maneira mostrada acima (iniciando com a keyword function), estaremos lidando com declaração e o que precisamos é de uma expressão ([Function Expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function)).

No nosso exemplo, se por um lado o parser identifica o keyword function como uma declaração, por outro ele reconhece o parênteses () como um grupo de operadores, e quando utilizamos sem uma expressão dentro temos um erro.

Mas se transformarmos uma função em Function Expression e, em seguida, um conjunto de parênteses (), esses mesmos parênteses se tornam uma invocação de uma função na qual podemos passar parâmetros sem problemas, tendo outro comportamento. Como fazer uma function se tornar uma expressão? Apenas a inserimos dentro de um grupo de operadores.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | (function doSomething() { /\* codigo \*/ })(); // undefined | | |
| 2 | |  |

|  |  |  |
| --- | --- | --- |
| 3 | | // Agora com função anônima |
| 4 | (function() { /\* codigo \*/ })(); // undefined | | |

Utilizando parâmetros:

|  |  |  |
| --- | --- | --- |
| 1 | | //Função imediata com parâmetro |
| 2 | (function doSomething(x) { console.log(x); })(1); // 1 | | |

|  |  |  |
| --- | --- | --- |
| 3 | |  |
| 4 | // Agora com função anônima | | |

|  |  |
| --- | --- |
| 5 | (function(x) { console.log(x) })(1); // 1 |

**Outras maneiras**

|  |  |
| --- | --- |
| 1 | (function() { /\* codigo \*/ }()); // Método de Douglas Crockford |

Em alguns casos, o parser já espera uma Function Expression, então não precisamos colocar o grupo de operadores.

|  |  |
| --- | --- |
| 1 | var doSomething = function() { return "done"; }(); // |

**Sem o grupo de operadores**

Podemos também definir usando o construtor new, mas é menos performático que os outros métodos. Eu não recomendo essa opção.

|  |  |  |
| --- | --- | --- |
| 1 | | new function(){ /\* codigo \*/ }; |
| 2 | new function(x, y){ /\* codigo \*/ }(1, 2); // Use parênteses se tiver argumentos. | | |

Economizamos um byte transformando para Function Expression sem usar o grupo de operadores, mas operadores unários.

[view source](https://imasters.com.br/front-end/javascript/sobre-funcoes-imediatas-javascript-iife/?trace=1519021197&source=single#viewSource)

[print](https://imasters.com.br/front-end/javascript/sobre-funcoes-imediatas-javascript-iife/?trace=1519021197&source=single#printSource)[?](https://imasters.com.br/front-end/javascript/sobre-funcoes-imediatas-javascript-iife/?trace=1519021197&source=single#about)

|  |  |
| --- | --- |
| 1 | !function(){ /\* codigo \*/ }(); |
| 2 | ~function(){ /\* codigo \*/ }(); |

|  |  |
| --- | --- |
| 3 | -function(){ /\* codigo \*/ }(); |
| 4 | +function(){ /\* codigo \*/ }(); |

Para ir mais além, segue algumas referências:

* [IIFE](http://benalman.com/news/2010/11/immediately-invoked-function-expression/#iife);
* [Every possible way to define a javascript function](http://www.bryanbraun.com/2014/11/27/every-possible-way-to-define-a-javascript-function);

Espero ter ajudado! Dúvidas, sugestões, críticas, só comentar logo abaixo.

\*\*\*

*Esse texto foi publicado originalmente no blog do autor.*[*Confira aqui*](http://pedrotcaraujo.github.io/2014/12/01/funcoes-imediatas-IIFE/)*.*