Posted by
Felipe N. Moura (http://twitter.com/felipenmoura)

Wed Oct 01 2014 04:01:41 GMT+0000 (UTC)

ARROW FUNCTIONS AND THEIR SCOPE

scope (../../categories/scope) **articles (../../categories/articles)** **basics (../../categories/basics)**

(https://www.facebook.com/sharer.php?u=http://jsrocks.org/2014/10/arrow-functions-and-their-scope)
(https://twitter.com/intent/tweet?text=http://jsrocks.org/2014/10/arrow-functions-and-their-scope)

Among so many great new features in ES6, Arrow Functions (or Fat Arrow Functions) is one that deserves attention! It is not just awesome, it's also great to work with scopes, shortcuts some techniques we are used to use nowadays, shrinks the number of lines of code... But it may be a little harder to read if you are not used to the way it works. So, let's take a look at it, right now!

Making it work

To study and try it for yourself, you can simply copy some of the examples and paste them on your browser's console. By now, you can use the Firefox (22+) Developer Tools, which already supports arrow functions, or on Google Chrome. But to use it in your Chrome, you will have to:

- - Enable it: Open about:flags in the address bar, and enable the "Experimental JavaScript" flag
- - Use it always in a function in "use strict" mode, so, to run it on Google Chrome's console:

```
(function(){
    "use strict";
    // use arrow functions here
}());
```

With time, fortunately, more browsers will support the new ES6 features. Now that it's all set up, let's dive deep into it!

A New Token

A new token has been added to ES6, and it is called "fat arrow", represented by

```
=>
```

The new Syntax

With this new token and feature, came this new syntax:

```
param => expression
```

Syntax with which we can use some variations, according to the number of statements in the expression, and number of parameters we want to use:

```
// single param, single statement
param => expression;

// multiple params, single statement
(param [, param]) => expression;

// single param, multiple statements
param => {
    statements;
}

// multiple params, multiple statements
([param] [, param]) => {
   statements
}

// with no params, single statement
() => expression;

// with no params, multiple statements
() => {
    statements;
}

// one statement, returning an object
([param]) => ({ key: value });
```

How it works

If we would "translate" it to something we already do today, it would be something like this:

```
// this function
var func = function (param) {
    return param.split(" ");
}

// would become:
var func = param => param.split(" ");
```

That means this syntax actually returns a new function, with the given statements and params.

Therefore, we can call the function the same way we are already used to:

```
func("Felipe Moura"); // returns ["Felipe", "Moura"]
```

Immediately-invoked function expression (IIFE)

Yes, you can invoke such functions immediately, as they are, indeed, expressions. Like so:

```
( x => x * 2 )( 3 ); // 6
```

It creates a function, which receives the argument `x` and returns `x * 2`, then it immediately executes this expression, passing the value `3` as argument.

In case you have more statements to execute, or more params:

```
( (x, y) => {
    x = x * 2;
    return x + y;
})( 3, "A" ); // "6A"
```

Relevant considerations

Considering:

```
var func = x => {
    return x++;
};
```

We may point some relevant considerations:

- `arguments` is not created/defined for your function

```
console.log(arguments); // not defined
```

- `typeof` and `instanceof` work just fine as well

```
func instanceof Function; // true
typeof func; // function
func.constructor == Function; // true
```

- Using parentheses inside, as suggested by jsLint will not work

```
// works with regular function syntax, as suggested by JSLint
(function (x, y){
    x= x * 2;
    return x + y;
} (3, "B") );

// doesn't work with Arrow Functions
( (x, y) => {
    x= x * 2;
    return x + y;
} ( 3, "A" ) );

// but it would work if the last line was
// })(3, "A");
```

- Although it is a function, is not a constructor

```
var instance= new func(); // TypeError: func is not a constructor
```

- It has no prototype

```
func.prototype; // undefined
```

Scope

The `this` in arrow functions' scopes works a little bit different, as well. The way we are used to it, the `this` keyword may reference to: `window` (if accessed globally, not in strict mode), `undefined` (if accessed globally, in strict mode), an *instance* (if in a constructor), an *object* (if in a method or function inside an object or instance) or a *binded/applied value*. It may also be a `DOMElement`, for example, in cases when you are using addEventListener. This might be annoying some times, or even tricky, causing you some trouble! Besides, it is referenced as *"scope-by-flow"*. What do I mean by saying that?

Let's see, firstly, how the `this` token behaves in different situations:

In an EventListener:

```
document.body.addEventListener('click', function(evt){
    console.log(this); // the HTMLBodyElement itself
});
```

In instances:

```
function Person () {

    let fullName = null;

    this.getName = function () {
        return fullName;
    };

    this.setName = function (name) {
        fullName = name;
        return this;
    };
}

let jon = new Person();
jon.setName("Jon Doe");
console.log(jon.getName()); // "Jon Doe"
```

In this particular case, once `Person.setName` is "chainable" (by returning itself), we could also use it like this:

```
jon.setName("Jon Doe")
   .getName(); // "Jon Doe"
```

In an object:

```
let obj = {
    foo: "bar",
    getIt: function () {
        return this.foo;
    }
};

console.log( obj.getIt() ); // "bar"
```

But then, comes the "scope-by-flow" I mentioned. If either the flow or scope changes, the `this` reference changes as well.

```
function Student(data){

    this.name = data.name || "Jon Doe";
    this.age = data.age>=0 ? data.age : -1;

    this.getInfo = function () {
        return this.name + ", " + this.age;
    };

    this.sayHi = function () {
        window.setTimeout( function () {
            console.log( this );
        }, 100 );
    }

}

let mary = new Student({
    name: "Mary Lou",
    age: 13
});

console.log( mary.getInfo() ); // "Mary Lou, 13"
mary.sayHi();
// window
```

Once `setTimeout` changes the execution flow, the `this` reference becomes the global object (in this case, `window`), or `undefined` in strict mode. Due to this, we end up using techniques like the use of variables like "self", "that" or something like it, or having to use the `.bind` method.

But don't worry, arrow functions are here to help! With arrow functions, the scope is kept with it, from where it was called.

Let's see the **same** example as before, but using an arrow function, passed to the `setTimeout` call.

```
function Student(data){

    this.name = data.name || "Jon Doe";
    this.age = data.age>=0 ? data.age : -1;

    this.getInfo = function () {
        return this.name + ", " + this.age;
    };

    this.sayHi = function () {
        window.setTimeout( ()=>{ // the only difference is here
            console.log( this );
        }, 100 );
    }

}

let mary = new Student({
    name: "Mary Lou",
    age: 13
});

console.log( mary.getInfo() ); // "Mary Lou, 13"
mary.sayHi();
// Object { name: "Mary Lou", age: 13, ... }
```

Interesting and useful usage

As it is very easy to create arrow functions, and their scopes work as mentioned before, we can use it in a variety of ways.

For example, it can be used directly in an `Array#forEach` call:

```
var arr = ['a', 'e', 'i', 'o', 'u'];
arr.forEach(vowel => {
    console.log(vowel);
});
```

Or in an `Array#map` :

```
var arr = ['a', 'e', 'i', 'o', 'u'];
arr.map(vowel => {
    return vowel.toUpperCase();
});
// [ "A", "E", "I", "O", "U" ]
```

You can also use it in recursion:

```
var factorial = (n) => {
    if(n==0) {
        return 1;
    }
    return (n * factorial (n-1) );
}

factorial(6); // 720
```

Also, let's say, sorting backwards an array:

```
let arr = ['a', 'e', 'i', 'o', 'u'];
arr.sort( (a, b)=> a < b? 1: -1 );
```

Or maybe in event listeners:

```
document.body.addEventListener('click', event=>console.log(event, this)); // EventOl
```

Useful links

Here is a list of interesting, useful links you can take a look at:

- - Arrow Functions in MDN Documentation (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions)
- - TC39 Wiki about Arrow Function (http://tc39wiki.calculist.org/es6/arrow-functions/)
- - ESNext (https://github.com/esnext)
- - ES6 Tools (https://github.com/addyosmani/es6-tools)
- - Grunt ES6 Transpiler (https://www.npmjs.org/package/grunt-es6-transpiler)
- - ES6 Fiddle (http://www.es6fiddle.net/)
- - ES6 Compatibility Table (http://kangax.github.io/compat-table/es6/)

Conclusion

Although arrow functions may make your source code a little bit less readable (and you can get used to it pretty fast), it is, indeed, a great solution to capture the outer scope's `this` value, a quick way to get things working and, in association with the `let` keyword, will definitely take our JavaScript to the next level! Try and use it, create some tests, run it in your browsers and leave comments with interesting solutions and uses you've found to arrow functions! I hope you have enjoyed this article, as much as you are going to enjoy arrow functions in a very close future.

COMMENTS

0 Comments ES6Rocks 1 Login

Recommend Share Sort by Best

Start the discussion…

Be the first to comment.

ALSO ON ES6ROCKS

JavaScript ♥ Unicode
2 comments • 3 years ago•
Shashank Sharma — Good to know these tricks. Thank you!

What you need to know about block scope - let
19 comments • 3 years ago•
Calvin Metcalf — A note about blocks is that there has to be curly braces for it to be a block, if you omit the from a while or if it

ES6 modules today with 6to5
15 comments • 3 years ago•
Šime Vidas — It's worth pointing out that Browserify can be used to bundle the transpiled CommonJS modules into one file

Configurando Babel 6 para Node.js
1 comment • a year ago•
William R. Fernandes — Olá.Apenas um ajuste necessário no arquivo .babelrc.{ "plugins": ["es2015", "stage-0"]}Deve ser

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy

⚡ OTHER POSTS

(https://twitter.com/intent/tweet?text=http://jsrocks.org/2016/01/configuring-babel-6-for-node-js)

2016-01-04T02:39:18.811Z

CONFIGURING BABEL 6 FOR NODE.JS (HTTP://JSROCKS.ORG/2016/01/CONFIGURING-BABEL-6-FOR-NODE-JS)

Hi! If you are like me, you are tired of writing the same old ES5 JS code in your Node.js applications.