UDC 004.432.2

OVERRIDING DEFAULT CALLING CONTEXT IN JAVASCRIPT

KIRYL LAPKOWSKI, DMITRIY PYATKIN
Polotsk state University, Belarus

The article is devoted to overriding the context of function calls in JavaScript. Let's consider all possible cases of setting the context of functions. We will analyze how effective the existing mechanisms are. As a result, an alternative way to set the context of functions will be formulated.

Introduction. JavaScript is a multi-paradigm programming language. This universality of language entails some problems. For example, if we consider JavaScript as an object-oriented programming language, we can see that such mechanisms as an abstract class are not implemented at the language level (although the programmer himself can easily program a class that cannot be instantiated), and prototype inheritance complicates writing a program in a functional style.

Setting the call context incorrectly is more likely to result in errors during program execution than in the inability to implement any principles of different programming paradigms. However, if the program has not been properly tested, errors may be rare, difficult to reproduce, and have a significant impact on the program.

Primary partition. The call context is the value in the context of which the function was called. Any function in JavaScript can refer to the this keyword within itself [1]. The this keyword refers to the context of a function call. The function is associated with the object at the time of the call, and this within the function receives the corresponding value. Consider four possible cases of determining the context in the order of their priority:

– Binding using the new operator. If the function was called with the new operator, this will get a reference to the newly created, empty object. Listing 1 providesan example.

Listing 1 - defining the context for the foo function using the new operator

```
varbar = newfoo()
```

– Explicit binding. If the function is called using the call or apply methods, or a context is bound to the function using the bind function, then the context of the call is the first argument passed to these functions. Listing 2 showsan example.

Listing 2 - defining the context of a function using the call method

```
var bar = foo.call(obj2)
```

– Implicit linking. If the function is called in the context of some object, also known as a container object. Then this will get a reference to this object. Listing 3 shows an example.

Listing 3-specifying an implicit context for a function

```
var bar = obj1.foo()
```

– Default binding. If none of the above cases are applicable to the function call, this will get the default value. In strict mode it is undefined, in non - strict mode it is a global object [2]. Listing 4 showsan example.

Listing 4-setting the default function context

```
var bar = foo()
```

The problem of tight coupling and binding default

The function context is lost if the function is used as a variable [3]. Typically, you use a hard binding to bind a context, or an arrow function that retrieves the context from a closure.

Hard binding of the calling context to the function (using the bind method), decreases the flexibility of the function, allowing you to override the context using the implicit or even explicit calling context.

Binding a function to the default context is rarely intentional, because in the case of binding to undefined, any attempt to read a property from this, or a method call, will result a runtime error. If you do not use strict mode, the global object will get into the context of the call, and changing any property or method of the object will change the global object. A scenario with a global object as the context of a call often results in hard-to-catch errors.

The implementation overrides the default context

As a solution to the problem of tight coupling and linking by default, you can override the calling context. The function implementation is presented in listing 5.

Listing 5-implementing the default context override

```
if (!Function.prototype.softBind) {
  Function.prototype.softBind = function(obj) {
        varfn = this,
        curried = [].slice.call( arguments, 1 ),
        bound = function bound() {
            return fn.apply(
                (!this ||
                    (typeofwindow !== "undefined" &&
                        this === window) ||
                    (typeofglobal !== "undefined" &&
                        this === global)
                ) ?obj : this,
                curried.concat.apply( curried, arguments )
            );
        };
        bound.prototype = Object.create( fn.prototype );
        return bound;
  };
}
```

Conclusion. The article deals with various ways of context binding and the problems of their use. A function that sets the default context, which allows to solve the problem of loss of context, saving the ability to override it further in the program.

REFERENCES

1. JavaScript.info Object methods, this [Electronic resource]. – Mode of access: https://javascript.info/object-methods. – Date of access: 20.02.2019.
2. You don't know JS this & Objects Prototypes [Electronic resource]. – Mode of access: https://github.com/getify/You-Dont-Know-JS/blob/master/this%20%26%20object%20prototypes/ch2.mdahhh!. – Date of access: 20.02.2019.
3. Dmitry Pavlutin Blog [Online] Gentle explanation of 'this' keyword in javaScript. – Mode of access: https://dmitripavlutin.com/gentle-explanation-of-this-in-javascript/. – Date of access: 20.02.2019.
4. MDN web docks this [Electronic resource]. – Mode of access: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this ahhh!. – Date of access: 20.02.2019.