УДК 004.432.2

ПЕРЕОПРЕДЕЛЕНИЕ КОНТЕКСТА ВЫЗОВА ПО УМОЛЧАНИЮ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ JAVASCRIPT

К.И. ЛАПКОВСКИЙ
(Представлено: Д.В. ПЯТКИН)

Рассматривается переопределение контекста вызова функций в JavaScript. Представлены все возможные случаи задания контекста функций. Анализируется, насколько эффективны уже существующие механизмы. В результате сформулирован альтернативный способ задания контекста функций.

JavaScript – мультипарадигменный язык программирования. Такая универсальность языка влечет за собой некоторые проблемы. Например, если рассматривать JavaScript как объектно-ориентированный язык программирования, можно заметить, что на уровне языка не реализованы такие механизмы как абстрактный класс (хотя программист сам без труда может запрограммировать класс, экземпляр которого нельзя создать), а прототипное наследование усложняет написание программы в функциональном стиле.

Неправильное задание контекста вызова скорее приведет к ошибкам во время выполнения программы, чем к невозможности реализовать какие-либо принципы разных парадигм программирования. Однако если программа не была должным образом протестирована, исключительные ситуации могут появляться редко, воспроизводиться сложно, при этом сильно влиять на работу программы.

Контекст вызова – это значение, в контексте которого была вызвана функция. Любая функция в JavaScript может обращаться к ключевому слову this внутри себя [1]. Ключевое слово this ссылается на контекст вызова функции. Функция в момент вызова связывается с объектом, и this внутри этой функции получает соответствующие значение. Рассмотри четыре возможных случая определения контекста в порядки их приоритета:

Связывание с помощью оператора new. Если функция была вызвана с оператором new, то this получит ссылка на только что созданный, пустой объект. Пример представлен в листинге 1.

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

Листинг 1. – Определение контекста для функции foo с помощью оператора new

Явное связывание. Если функция вызвана с помощью методов call или apply, или к функции привязан контекст с помощью функции bind, тогда контекстом вызова будет первый аргумент, переданный в эти функции. Пример представлен в листинге 2.

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

Листинг 2. – Определение контекста функции методом call

Неявное связывание. Если функция вызвана в контексте какого-то объекта, так же известного как объекта-контейнера. То в this попадет ссылка на этот объект. Пример в листинге 3.

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

Листинг 3. – Неявное задание контекста для функции

Связывание по умолчанию. Если ни один из вышеописанных случаев не применим к вызову функции, то в this попадет значение по умолчанию. В строгом режиме - это undefined, в не строгом – это глобальный объект [2]. Пример представлен в листинге 4.

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

Листинг 4. – Задание контекста функции по умолчанию

Проблема жесткого связывания и связывания по умолчанию

Контекст функции теряется, если использовать функцию как переменную [3]. Обычно для привязки контекста используется жесткую привязка, или стрелочную функцию, которая получает контекст из замыкания.

Жесткая привязка контекста вызова к функции (через метод bind), уменьшает гибкость функции, не позволяя переопределить контекст с помощью неявного или даже явного задания контекста вызова.

Привязка функции к контексту по умолчанию редко является намеренной, так как в случае с привязкой к undefined, любая попытка считать свойство у this, или вызов метода, приведет к ошибке во время выполнения программы. Если же использовать не строгий режим, то в контекст вызова попадет глобальный объект, и изменение любого свойства или метода объекта приведет к изменению глобального объекта. Сценарий с глобальным объектом в качестве контекста вызова часто приводит к сложно уловимым ошибкам.

Реализация переопределения контекста по умолчанию

В качестве решения проблемы жесткого связывания и связывания по умолчанию можно использовать переопределение контекста вызова. Реализация функции представлена в листинге 5.

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

Листинг 5. – Реализация переопределения контекста по умолчанию

Таким образом, рассмотрены различные способы привязки контекста, и проблемы их использования. Разработана функция, задающая контекст по умолчанию, позволяющая решить проблему потери контекста, при этом сохраняя возможность его переопределения далее в программе.

ЛИТЕРАТУРА

1. JavaScript.info [Электронный ресурс] // Object methods, this. – Режим доступа: https://javascript.info/object-methods. – Дата доступа: 24.09.2018.
2. You don’t know JS this & Objects Prototypes [Электронный ресурс]. – Режим доступа: https://github.com/getify/You-Dont-Know-JS/blob/master/this%20%26%20object%20prototypes/ch2.md. – Дата доступа: 24.09.2018.
3. Dmitri Pavlutin Blog [Электронный ресурс] // Gentle explanation of ‘this’ keyword in javaScript. – Режим доступа: https://dmitripavlutin.com/gentle-explanation-of-this-in-javascript/. – Дата доступа: 24.09.2018.
4. MDN web docks this [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this. – Дата доступа: 24.09.2018.