Javascript Under The Hood

The Mysterious Parts

Demystifying Javascript's "First-class functions", "Scope", "Closure", and "this" keyword binding

Tran Duc Thang
Framgia Vietnam - Business Strategy Office - Human Development Section

"Javascript is the World's Most Misunderstood Programming Language!"
~ Douglas Crockford ~

Too much to remember, too weird to understand?

first-class functions

IIFE

hoisting

lexical scope

function expression

function scope

closure

block scope

this

prototype

function statement

new

function constructor

Some JS Quizes

http://jsbin.com/tojavun/edit?html,js,console,output

```
var a = 1;
console.log(a);
console.log(b);
console.log(c);
console.log(d);
console.log(e);
console.log(f);
console.log(g);
console.log(h);
var b = 2;
var d = e = 3;
function f() {};
var g = function() {};
h = 4;
```

```
function foo() {
    var bar = 5;
    baz = 6;
    console.log(qux);
    var qux = 8;
}
var qux = 7;
foo();
console.log(bar);
console.log(foo.bar);
console.log(baz);
console.log(foo.baz);
```

Some JS Quizes

http://jsbin.com/wabedaf/3/edit?html,js,output

```
// Example 1: What will be alerted when we click at the buttons
for (var i = 0; i < 5; i++) {
    var btn = document.createElement('button');
    btn.appendChild(document.createTextNode('Button ' + i));
    btn.addEventListener('click', function() {
        alert(i);
    });
    document.body.appendChild(btn);
}

// Example 2: What will be outputed to the console
for (var j = 0; j < 5; j++) {
    setTimeout (function() {
        console.log(j);
    }, 0);
}
```

Table of Contents

< previous

next >

First-class functions

- Javascript value types
  - string
  - number
  - boolean
  - null
  - undefined
  - symbol (ES6)
  - object

Primitive values

Object

First-class functions

- Everything which is not primitive is **object**.
- Function (and even Class in ES6) in Javascript is actually **object**.

First-class functions

```
> function foo(bar, baz) {}
< undefined
> foo.length
< 2
> typeof foo
< "function"
> foo instanceof Object
< true
> foo.a = 1
< 1
> foo.b = 2
< 2
> foo
< function foo(bar, baz) {}
> foo.a
< 1
> foo.b
< 2
> foo.a + foo.b
< 3
> foo.hasOwnProperty('a')
< true
> foo.hasOwnProperty('b')
< true
> 'a' in foo
< true
> for (var i in foo) {console.log(i)}
  a
  b
< undefined
```

```
> class Foo {}
< function class Foo {}
> Foo.length
< 0
> typeof Foo
< "function"
> Foo instanceof Object
< true
> Foo.a = 1
< 1
> Foo.b = 2
< 2
> Foo
< function class Foo {}
> Foo.a
< 1
> Foo.b
< 2
> Foo.a + Foo.b
< 3
> Foo.hasOwnProperty('a')
< true
> Foo.hasOwnProperty('b')
< true
> 'a' in Foo
< true
> for (var i in Foo) {console.log(i)}
  a
  b
< undefined
```

< previous

next >

First-class functions

- **First-class citizen** (also type, object, entity, or value) is an entity which supports all the operations generally available to other entities.
- These operations typically include **being passed as an argument**, **returned from a function**, and **assigned to a variable**

First-class functions

- A programming language is said to have **first-class functions** if it **treats functions as first-class citizens**.
- Javascript has **first-class functions**!

First-class functions

```
setTimeout(function() {
    console.log('A function can be passed as an argument');
}, 1000);

var f = function() {
    console.log('A function can be assigned to a variable');
}

function foo() {
    return function() {
        console.log('A function can be returned from a function');
    }
}
```

First-class functions

- **Function Statement**
- **Function Expression**

First-class functions

Function Statement	Function Expression
Defines function. Also known as function declaration.	Defines a function as a part of a larger expression syntax
Must begins with “**function**” keyword	**Must not** begin with “**function**” keyword
Must have a name	**Can** have a name or not (can be anonymous)

First-class functions

```
function statement() {
    console.log('This is a Function Statement');
}

var f = function() {
    console.log('This is a Function Expression');
}

var f = function expression() {
    console.log('This is another Function Expression');
}

// What will happend if we call expression()
```

First-class functions

- **IIFE**: **I**mmediately **I**nvoked **F**unction **E**xpression is a Javascript function that **runs as soon as it is defined**.

First-class functions

```
(function() {
    console.log('This is an Immediately Invoked Function Expression');
})();

(function iife() {
    console.log('This is another Immediately Invoked Function Expression');
}());

(function iife(message) {
    console.log(message);
})('This is yet another Immediately Invoked Function Expression');
```

Hoisting

- **Hoisting**: The ability to use variable, function **before they are declared**.
- Javascript only hoists **declarations**, not **initializations**

Hoisting

```
console.log(a);
var a = 2;
console.log(a);

// Hoisted
var a;
console.log(a);
a = 2;
console.log(a);
```

```
foo();
bar()

function foo() {
    console.log('Function Hoisted');
}

var bar = function baz() {
    console.log('Function Expression is not hoisted');
}

// Hoisted
function foo() {
    console.log('Function Hoisted');
}
var bar;

foo();
bar();

bar = function baz() {
    console.log('Function Expression is not hoisted');
}
```

Scope

- **Scope** is the set of variables, objects, and functions you have access to
- 2 ways to create a **Scope**: **Function** and **Block***

Scope

- **Lexical Scope** vs **Dynamic Scope**
  - **Lexical Scope**, or **Static Scope**: The scope of a variable is defined by its **location within the source code** and nested functions have access to variables declared in their outer scope.
  - **Dynamic Scope**: The scope of a variable depends on where the functions and scopes are called from
  - **Lexical Scope** is write-time, whereas **Dynamic Scope** is runtime
  - **Javascript has Lexical Scope!**

Scope

```
var globalVariable = 1;
var foo = 2;
local(3);
function local(foo) {
    console.log(foo); // 3
    var localVariable = 4;
    bar = 5;
    baz(6);
    function baz(foo) {
        console.log(globalVariable); // 1
        console.log(localVariable); // 4
        console.log(foo); // 6
    }
    for (var i = 0; i < 10; i++) {
        var j = 11;
    }
    console.log(i); // 10
    console.log(j); // 11
}
console.log(localVariable) // ReferenceError
console.log(bar) // 5
```

- Global Scope
- Local Scope
- Nested Scope
- Outer Scope
- Inner Scope
- Function Scope
- Block Scope

< previous

next >

Scope

IIFE can be used to

create a new scope!

Closure

- **Closure** is a **function** that can **remember and access its lexical scope** even when it's **invoked outside its lexical scope**

```
function foo() {
    var bar = 1;
    return function() {
        console.log(bar);
    }
}

baz = foo();
baz(); // 1
```

Closure

Unravel the problems

http://jsbin.com/wabedaf/3/edit?html,js,output

```
// Example 1: What will be alerted when we click at the buttons
for (var i = 0; i < 5; i++) {
    var btn = document.createElement('button');
    btn.appendChild(document.createTextNode('Button ' + i));
    btn.addEventListener('click', function() {
        alert(i);
    });
    document.body.appendChild(btn);
}

// Example 2: What will be outputed to the console
for (var j = 0; j < 5; j++) {
    setTimeout (function() {
        console.log(j);
    }, 0);
}
```

“this” keyword

- “***this***” does not refer to the function itself.
- “***this***” does not refer to the function’s lexical scope.
- In most cases, the value of “***this***” is **determined by how a function is called**.
- “***this***” may be different each time the function is called.

“this” keyword

- “***this***” does not refer to the function itself.

```
function foo() {
    this.bar = 1;
}

foo();
console.log(foo.bar);

// console.log(bar);
```

“this” keyword

- **Default binding**: Standalone function invocation. “***this***” is bind to *global* object (in non-strict mode)

```
function foo() {
    console.log(this); // window
    this.bar = 1;
}

foo();
console.log(bar); // 1
```

“this” keyword

- **Implicit binding**:
  Function is invoked from a containing object. “***this***” is bind to the containing object.

```
var a = {
    b: function() {
        return this;
    },
    c: function() {
        function d() {
            return this;
        }
        return d();
    }
}

console.log(a.b()); // a
var e = a.b;
console.log(e()); // window
var f = {};
f.g = a.b;
console.log(f.g()); // f
console.log(a.c()); // window
```

“this” keyword

- **Explicit binding**: Function is called with ***call***, ***apply*** or ***bind*** method. “***this***” is bind to the object passed to the binding method.

```
var obj = {
    message: 'Come from obj'
};

var message = 'Global';
function foo() {
    console.log(this.message);
}

foo(); // Global
foo.call(obj); // Come from obj
foo.apply(obj); // Come from obj
foo = foo.bind(obj);
foo(); // Come from obj
```

“this” keyword

- **new keyword binding**: “***this***” is bind to the new object that is created

```
function Foo(bar) {
    this.bar = bar;
}

var baz = new Foo('ThangTD');
console.log(baz.bar); // ThangTD
```

“this” keyword

- **Arrow function**: “***this***” is lexically adopted from the enclosing scope

```
var a = {
    b: function() {
        var c = function() {
            return this;
        }
        return c();
    },
    d: function() {
        var e = () => this;
        return e();
    }
}

console.log(a.b()); // window
console.log(a.d()); // a
```

“this” keyword

- **Binding priority**
  - Arrow function
  - ***new*** keyword
  - Explicit binding, by using ***call***, ***apply*** and ***bind***
  - Implicit binding
  - Default binding

Some best practices

- Try to avoid Global variables
- Always declare variables
- Put variables declaration on top

Use Strict Mode

References

- You don’t know JS (https://github.com/getify/You-Dont-Know-JS)
- Speaking JS (http://speakingjs.com/)
- Exploring ES6 (http://exploringjs.com/es6/)
- http://www.2ality.com/ JavaScript and more
- Mozilla Developer Network (https://developer.mozilla.org/en-US/docs/Web/JavaScript)
- Tìm hiểu về Strict Mode trong Javascript (https://viblo.asia/thangtd90/posts/jaqG0QQevEKw)

Thank you for listening!

Q&A

For any discussion, you can refer this post on Viblo

https://viblo.asia/thangtd90/posts/WApGx3P3M06y