iTechArt

Software
Engineering
Services

JAVASCRIPT

Dmitry Kosmich

AUGUST 16, 2019

CONTENT

- Closure
- Hoisting
- This
- Parameters
- Types
- Enhanced Object Properties

- Destructing Assignment
- Prototype

CLOSURE

→ Closure

CLOSURE

CODE SNIPPET

```
function makeFunc() {
  var name = "Mozilla";

  function displayName(){
    alert(name);
  }

  return displayName;
};

var myFunc = makeFunc();

myFunc(); // Mozilla
```

HOISTING

→ Scoping, var, let, const

HOISTING

CODE SNIPPET

```
var foo = 1;

function bar() {
  if (!foo) {
    var foo = 10;
  }

  alert(foo);
}

bar();

// foo === 10
```

CODE SNIPPET

```
var foo, bar ;

foo = 1;
bar = function() {
  var foo;

  if (!foo) {
    foo = 10;
  }

  alert(foo);
}

bar();
```

HOISTING

CODE SNIPPET

```
var a = 1;

function b() {
  a = 10;

  return;

  function a() {}
}

b();

alert(a);

// a === 1
```

CODE SNIPPET

```
var a, b;

a = 1;
b = function() {
  var a;

  a = 10;

  return;

  a = function() {}
}

b();

alert(a);
```

BLOCK-SCOPED VARIABLES (LET)

CODE SNIPPET

```
// ES6
function test() {
 let x = 1;

 if (true) {
  let x = 2;

  console.log(x); // 2
 }

 console.log(x); // 1
}
```

CODE SNIPPET

```
// ES6
let callbacks = [];

var i;

for (i = 0; i <= 2; i++) {
 callbacks[i] = function () {
  return i * 2;
 };
}

callbacks[0]() === 4;
callbacks[1]() === 4;
callbacks[2]() === 4;
```

BLOCK-SCOPED VARIABLES (CONST)

CODE SNIPPET

```
const MY_FAV = 7;

if (MY_FAV === 7) {
    var MY_FAV = 20; // выдаст ошибку
}

const FOO; // выдаст ошибку
```

CODE SNIPPET

```
const MY_OBJECT = {"key": "value"};

MY_OBJECT.key = "otherValue";
// Object.freeze() для того, чтобы сделать
// объект неизменяемым


const MY_ARRAY = [];

MY_ARRAY.push("A");
```

BLOCK-SCOPED FUNCTIONS

CODE SNIPPET

```
// ES6
{
 function foo () { return 1; }

 foo() === 1;

 {
  function foo () { return 2; }
  foo() === 2;
 }

 foo() === 1;
}
```

THIS

→ Context, Call, Apply, Bind, Arrow functions

THIS: GLOBAL CONTEXT

CODE SNIPPET

```
console.log(this === window); // true
```

CODE SNIPPET

```
function f1() {
    return this;
}

// In a browser:
f1() === window; // true

// In Node:
f1() === global; // true
```

THIS: NOT IN STRICT MODE

CODE SNIPPET

```
function f1() {
    return this;
}

// In a browser:
f1() === window; // true

// In Node:
f1() === global; // true
```

THIS: STRICT MODE

CODE SNIPPET

```
function f2() {
  'use strict'; // see strict mode
  return this;
}

f2() === undefined; // true
```

THIS: ARROW FUNCTIONS

CODE SNIPPET

```
function Person(){
 this.age = 0;

 setInterval(() => {
  this.age++; // `this` указывает на объект Person
 }, 1000);
}
```

THIS: METHODS

CODE SNIPPET

```
var obj = {
 i: 10,
 b: () => console.log(this.i, this),
 c: function() {
  console.log(this.i, this);
 }
}

obj.b(); // prints undefined, Window {...} (или глобальный объект)
obj.c(); // prints 10, Object {...}
```

THIS: CONSTRUCTOR

CODE SNIPPET

```
function C() {
// let this = {}

  this.a = 37;
}

var o = new C();
console.log(o.a); // 37

function C2() {
  this.a = 37;
  return {a: 38};
}

o = new C2();
console.log(o.a); // 38
```

!!! Doesn't work for arrow functions

THIS: CALL & APPLY

CODE SNIPPET

```
var obj = {a: 'Custom'};
var a = 'Global';

function whatsThis() {
  return this.a;
}

whatsThis(); // 'Global'
whatsThis.call(obj); // 'Custom'
whatsThis.apply(obj); // 'Custom'
```

!!! Doesn't work for arrow functions

THIS: BIND

CODE SNIPPET

```
function f() {
  return this.a;
}

var g = f.bind({a: 'azerty'});
console.log(g()); // azerty

var h = g.bind({a: 'yoo'}); // bind only works once!
console.log(h()); // azerty

var o = {a: 37, f: f, g: g, h: h};
console.log(o.a, o.f(), o.g(), o.h()); // 37,37, azerty, azerty
```

!!! Doesn't work for arrow functions

PARAMETERS

→ Default Parameter Values, Rest Parameter, Spread Operator

DEFAULT PARAMETER VALUES

CODE SNIPPET

```
function f (x, y = 7, z = 42) {
 return x + y + z;
}

f(1) === 50;
```

CODE SNIPPET

```
function go() {
 return ":P"
}

function callSomething(thing = go()) {
 return thing
}
```

REST PARAMETER

CODE SNIPPET

```
function f (a, b, ...theArgs) {
 return (a + b) * theArgs.length;
}

f(1, 2, "hello", true, 7) === 9;
```

SPREAD OPERATOR

CODE SNIPPET

```
var params = [ "hello", true, 7 ];
var other = [ 1, 2, ...params ]; // [ 1, 2, "hello", true, 7 ]

function f (x, y, ...a) {
 return (x + y) * a.length;
}

f(1, 2, ...params) === 9;

var str = "foo";
var chars = [ ...str ]; // [ "f", "o", "o" ]
```

SPREAD OPERATOR: COPY AN ARRAY

CODE SNIPPET

```
// ES6
var a = [[1], [2], [3]];
var b = [...a]; // [[1], [2], [3]]

b.shift().shift(); // 1

console.log(a);  // Массив “a” так же подвергся изменению: [[], [2], [3]]

var arr1 = [0, 1, 2];
var arr2 = [3, 4, 5];

arr1 = [...arr2, ...arr1]; // [3, 4, 5, 0, 1, 2]
```

SPREAD OPERATOR: SPREAD IN OBJECT LITERALS

CODE SNIPPET

```
var obj1 = {foo: 'bar', x: 42};
var obj2 = {foo: 'baz', y: 13};


var clonedObj = { ...obj1 };
// Object { foo: "bar", x: 42 }

var mergedObj = { ...obj1, ...obj2 };
// Object { foo: "baz", x: 42, y: 13 }


var obj = {'key1': 'value1'};
var array = [...obj]; // TypeError: obj is not iterable
```

ARGUMENTS

CODE SNIPPET

```
var arguments = 42;
var arr = () => arguments;

arr(); // 42

function foo() {
 var f = (i) => arguments[0] + i;
 return f(2);
}

foo(1); // 3
```

TYPES

→ String Interpolation, Symbol

TYPES

- number,
- string,
- boolean,
- null,
- undefined,
- symbol,
- object

STRING INTERPOLATION

CODE SNIPPET

```
// ES6
var customer = { name: "Foo" };
var card = { amount: 7, product: "Bar", unitprice: 42 };

var message = `Hello ${customer.name},
want to buy ${card.amount} ${card.product} for
a total of ${card.amount * card.unitprice} bucks?`;
```

STRING INTERPOLATION

CODE SNIPPET

```
// ES6
var a = 5;
var b = 10;

function tag(strings, ...values) {
 console.log(strings[0]); // "Hello “
 console.log(strings[1]); // " world “
 console.log(values[0]); // 15
 console.log(values[1]); // 50

 return "Yo!";
}

tag `Hello ${ a + b } world ${ a * b}
 It’s me!`;
// Yo!
```

https://www.styled-components.com/

SYMBOL TYPE

CODE SNIPPET

```
const foo = Symbol();
typeof foo === "symbol";
Symbol("foo") !== Symbol("foo");
var sym = new Symbol(); // TypeError
```

```
let obj = {};
obj[foo] = "foo";
```

```
JSON.stringify(obj); // {}
Object.keys(obj); // []
Object.getOwnPropertyNames(obj); // []
Object.getOwnPropertySymbols(obj); // [ foo ]
```

CODE SNIPPET

```
var sym1 = Symbol.for("app.foo")
var sym2 = Symbol.for("app.foo")

sym1 === sym2

Symbol.keyFor(sym1) === "app.foo";
```

ENHANCED OBJECT PROPERTIES

→ Property Shorthand, Computed Property Names, Method Properties

PROPERTY SHORTHAND

CODE SNIPPET

```
// ES6
var a = 'foo',
    b = 42,
    c = {};

// Сокращение имен свойств
var o = {a, b, c};

// Иначе говоря, console.log((o.a === {a}.a)); // true
```

CODE SNIPPET

```
// ES5
var a = 'foo',
    b = 42,
    c = {};

var o = {
 a: a,
 b: b,
 c: c
};
```

COMPUTED PROPERTY NAMES

CODE SNIPPET

```
// ES6
let obj = {
 foo: "bar",
 [ "baz" + quux() ]: 42
};
```

CODE SNIPPET

```
// ES5
var obj = {
 foo: "bar"
};

obj[ "baz" + quux() ] = 42;
```

METHOD PROPERTIES

CODE SNIPPET

```
// ES6
let obj = {
 foo (a, b) { … },
 bar (x, y) { … },
 *quux (x, y) { … }
};
```

CODE SNIPPET

```
// ES5
var obj = {
 foo: function (a, b) { … },
 bar: function (x, y) { … },
 // quux: no equivalent in ES5 …
};
```

DESTRUCTING ASSIGNMENT

- Array Matching;
- Object Matching, Shorthand Notation;
- Object Matching, Deep Matching;
- → Object And Array Matching, Default Values;

ARRAY MATCHING

CODE SNIPPET

```
// ES6
var list = [ 1, 2, 3 ];
var [ a, , b ] = list;
[ b, a ] = [ a, b ];
```

CODE SNIPPET

```
// ES5
var list = [ 1, 2, 3 ];
var a = list[0], b = list[2];
var tmp = a; a = b; b = tmp;
```

OBJECT MATCHING, SHORTHAND NOTATION

CODE SNIPPET

```
// ES6
var o = {p: 42, q: true};
var {p, q} = o;

console.log(p); // 42
console.log(q); // true

// Объявление новых переменных
var {p: foo, q: bar} = o;

console.log(foo); // 42
console.log(bar); // true
```

OBJECT MATCHING, DEEP MATCHING

CODE SNIPPET

```
// ES6
var metadata = {
 title: "Scratchpad",
 translations: [
  {
   locale: "de",
   localization_tags: [ ],
   last_edit: "2014-04-14T08:43:37",
   url: "/de/docs/Tools/Scratchpad",
   title: "JavaScript-Umgebung"
  }
 ],
 url: "/en-US/docs/Tools/Scratchpad"
};

var { title: englishTitle, translations: [{ title: localeTitle }] } = metadata;
console.log(englishTitle); // "Scratchpad"
console.log(localeTitle); // "JavaScript-Umgebung"
```

OBJECT AND ARRAY MATCHING, DEFAULT VALUES

CODE SNIPPET

```
// ES6
var obj = { a: 1 };
var list = [ 1 ];
var { a, b = 2 } = obj;
var [ x, y = 2 ] = list;
```

CODE SNIPPET

```
// ES5
var obj = { a: 1 };
var list = [ 1 ];
var a = obj.a;
var b = obj.b === undefined ? 2 : obj.b;
var x = list[0];
var y = list[1] === undefined ? 2 : list[1];
```

PROTOTYPE

- Prototype;
- Class Definition;
- Class Inheritance;
- → Static Members.

[[PROTOTYPE]] OR __PROTO__

CODE SNIPPET

```
var obj1 = {foo: 'bar', x: 42};
var obj2 = {foo: 'baz', y: 13};

obj1.__proto__ = obj2;

obj1.foo === 'bar'
obj1.x === 42
obj1.y === 13 // obj1.[[Prototype]].y

obj1.toString === [ƒ toString() { [native code] }]

// obj1.[[Prototype]] === obj2
// obj1.[[Prototype]].[[Prototype]] === Object
// obj1.[[Prototype]].[[Prototype]].[[Prototype]] === null
```

FUNC.PROTOTYPE

CODE SNIPPET

```
let F = function () {
 this.a = 1;
 this.b = 2;
}

let o = new F(); // {a: 1, b: 2}

F.prototype.b = 3;
F.prototype.c = 4;

console.log(o.a); // 1
console.log(o.b); // 2
console.log(o.c); // 4
console.log(o.d); // undefined
```

CLASS DEFINITION

CODE SNIPPET

```
// ES5
function Shape(id, x, y) {
 this.id = id;
 this.move(x, y);
};

Shape.prototype.move = function (x, y) {
 this.x = x;
 this.y = y;
};
```

CODE SNIPPET

```
// ES5
var Shape = function (id, x, y) {
 this.id = id;
 this.move(x, y);
};

Shape.prototype.move = function (x, y) {
 this.x = x;
 this.y = y;
};
```

CODE SNIPPET

```
// ES6
class Shape {
 constructor (id, x, y) {
  this.id = id;
  this.move(x, y);
 }
 move (x, y) {
  this.x = x;
  this.y = y;
 }
}
```

CODE SNIPPET

```
// ES6
let Shape = class {
 constructor (id, x, y) {
  this.id = id;
  this.move(x, y);
 }
 move (x, y) {
  this.x = x;
  this.y = y;
 }
}
```

CLASS INHERITANCE

CODE SNIPPET

```
// ES5
var Rectangle = function (id, x, y, width, height) {
 Shape.call(this, id, x, y);
 this.width = width;
 this.height = height;
};

Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;
```

CODE SNIPPET

```
// ES6
class Rectangle extends Shape {
 constructor (id, x, y, width, height) {
  super(id, x, y);
  this.width = width;
  this.height = height;
 }
}
```

STATIC MEMBERS

CODE SNIPPET

```
// ES5
var Rectangle = function (id, x, y, width, height) {
 …
};

Rectangle.defaultRectangle = function () {
 return new Rectangle("default", 0, 0, 100, 100);
};


var defRectangle = Rectangle.defaultRectangle();
```

CODE SNIPPET

```
// ES6
class Rectangle extends Shape {
 …
 static defaultRectangle () {
  return new Rectangle("default", 0, 0, 100, 100);
 }
}


var defRectangle = Rectangle.defaultRectangle();
```

ITERATORS

→ Iterator & For-Of Operator

ITERATOR & FOR-OF OPERATOR

CODE SNIPPET

```
// ES6
let fibonacci = {

 // iterable protocol (Method @@iterator)
 [Symbol.iterator]() {
  let pre = 0, cur = 1;

  // iterator protocol
  return {
   next () {
    [ pre, cur ] = [ cur, pre + cur ];
    return { done: false, value: cur };
   }
  };
 }
}

for (let n of fibonacci) {
 if (n > 1000) break;
 console.log(n);
}
```

CODE SNIPPET

```
// ES5
var fibonacci = {
 next: (function () {
  var pre = 0, cur = 1;
  return function () {
   tmp = pre;
   pre = cur;
   cur += tmp;
   return cur;
  };
 })()
};

var n;
for (;;) {
 n = fibonacci.next();
 if (n > 1000) break;
 console.log(n);
}
```

GENERATORS

- Generator Function, Iterator Protocol;
- Generator Function, Direct Use;
- Generator Matching;
- → Generator Methods.

GENERATOR FUNCTION

CODE SNIPPET

```
function* idMaker() {
 var index = 0;
 while (index < 3) yield index++;
}

var gen = idMaker();

console.log(gen.next().value); // 0
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
console.log(gen.next().value); // undefined // ...
```

YIELD*

CODE SNIPPET

```
function* anotherGenerator(i) {
 yield i + 1;
 yield i + 2;
 yield i + 3;
}

function* generator(i) {
 yield i;
 yield* anotherGenerator(i);
 yield i + 10;
}

var gen = generator(10);

console.log(gen.next().value); // 10
console.log(gen.next().value); // 11
console.log(gen.next().value); // 12
console.log(gen.next().value); // 13
console.log(gen.next().value); // 20
```

PASSING ARGUMENTS INTO GENERATORS

CODE SNIPPET

```
function* logGenerator() {
 console.log(yield);
 console.log(yield);
 console.log(yield);
}

var gen = logGenerator();


gen.next();
gen.next('pretzel'); // pretzel
gen.next('california'); // california
gen.next('mayonnaise'); // mayonnaise
```

RETURN STATEMENT IN A GENERATOR

CODE SNIPPET

```
function* yieldAndReturn() {
 yield "Y";
 return "R";
 yield "unreachable";
}

var gen = yieldAndReturn();

console.log(gen.next()); // { value: "Y", done: false }
console.log(gen.next()); // { value: "R", done: true }
console.log(gen.next()); // { value: undefined, done: true }
```

GENERATOR FUNCTION: ITERATOR PROTOCOL

CODE SNIPPET

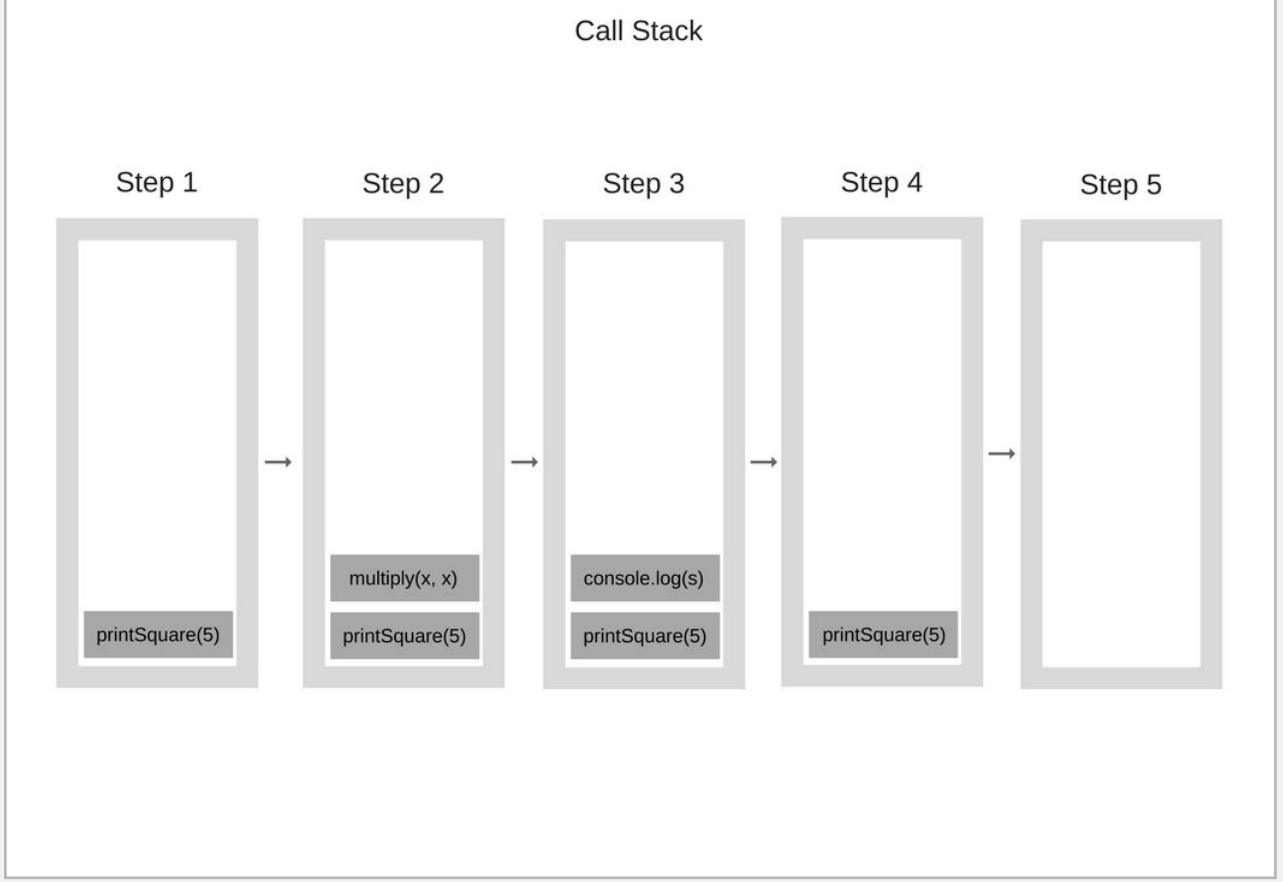
```
let fibonacci = {
 *[Symbol.iterator]() {
  let pre = 0, cur = 1;
  for (;;) {
   [ pre, cur ] = [ cur, pre + cur ];
   yield cur;
  }
 }
}

for (let n of fibonacci) {
 if (n > 1000) break;
 console.log(n);
}
```

EVENT LOOP

→ Stack trace

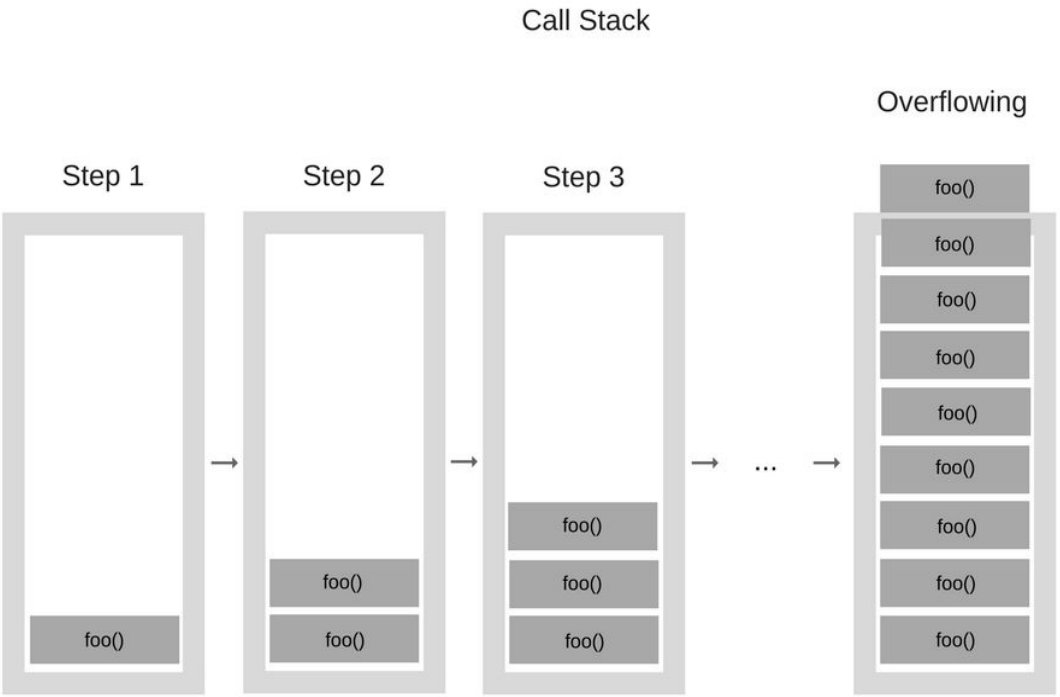
CALL STACK


```
function multiply(x, y) {
  return x * y;
}

function printSquare(x) {
  var s = multiply(x, x);
  console.log(s);
}

printSquare(5);
```

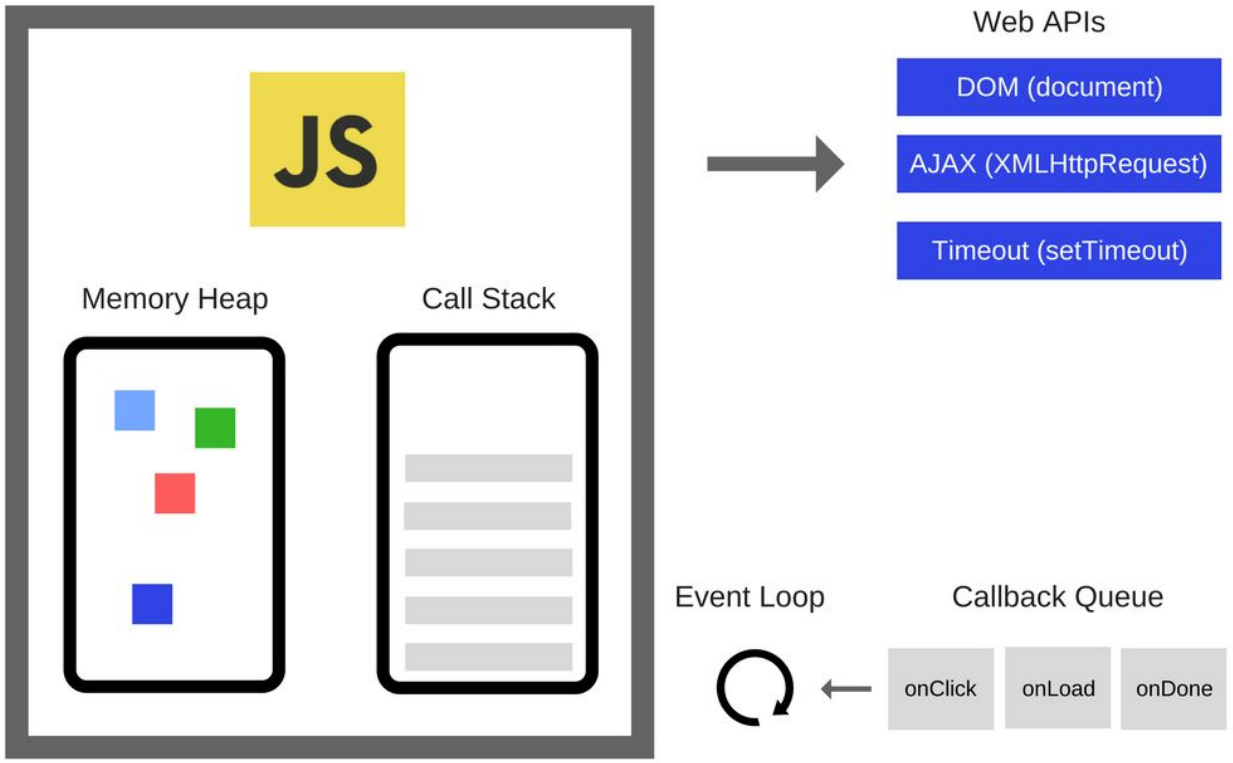


CALL STACK


```
function foo() {
  foo();
}

foo()
```



EVENT LOOP


JS
Memory Heap
Call Stack
Web APIs
DOM (document)
AJAX (XMLHttpRequest)
Timeout (setTimeout)
Event Loop
Callback Queue
onClick
onLoad
onDone

EVENT LOOP

Demo:
http://latentflip.com/loupe/?code=JC5vbignYnV0dG9uJywgJ2NsaWNrJywgZnVuY3Rpb24gb25DbGljaygpIHsKICAgIHNldFRpbWVvdXQoZnVuY3Rpb24gdGltZXIoKSB7CiAgICAgICAgY29uc29sZS5sb2coJ1lvdSBjbGlja2VkIHRoZSBidXR0b24hJyk7ICAgIAogICAgfSwgMCk7Cn0pOwoKY29uc29sZS5sb2coIkhpISIpOwoKc2V0VGltZW91dChmdW5jdGlvbiB0aW1lb3V0KCkgewogICAgY29uc29sZS5sb2coIkNsaWNrIHRoZSBidXR0b24hIik7Cn0sIDMwMDApOwoKY29uc29sZS5sb2coIldlbGNvbWUgdG8gbG91cGUuIik7!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D

BABEL

→ JavaScript compiler

BABEL

Babel или **Babel.js** - это бесплатный JavaScript-компилятор с открытым исходным кодом, используемый в веб-разработке.

Babel позволяет разработчикам программного обеспечения писать исходный код на предпочитаемом языке программирования или языке разметки и переводить на язык JavaScript, который понимают современные веб-браузеры.

Thank you

dzmitry.kosmich@itechart-group.com
Skype: Dim.27

iTechArt