@falsyvalues

ECMAScript > Future

Dmitry A. Soshnikov
http://dmitrysoshnikov.com

Future > ES6

let : block scope

Block scope

```
// ES3


if (false) {
    var logLevel = 10;
}

alert(logLevel); // ?
```

Block scope

```
// ES3
var logLevel; // = undefined;

if (false) {
    logLevel = 10;
}

alert(logLevel); // undefined
```

All variables are created **before** code execution – on entering the context.

A so-called “hoisting” of variables.

See: http://dmitrysoshnikov.com/notes/note-4-two-words-about-hoisting/

Block scope

```
// ES6 Harmony


if (false) {
    let logLevel = 10;
}

alert(logLevel); // ReferenceError
```

Block scope

```
// ES3, ES5

var handlers = [];

for (var k = 0; k < 3; k++) {
    handlers[k] = function () {
        alert(k);
    };
}
```

```
handlers[0](); // ?
handlers[1](); // ?
handlers[2](); // ?
```

Block scope

```
// ES3, ES5

var handlers = [];
var k;
for (k = 0; k < 3; k++) {
    handlers[k] = function () {
        alert(k);
    };
}
```

```
handlers[0](); // 3
handlers[1](); // 3
handlers[2](); // 3
```

let : block scope

ES3, ES5

```
for (var k = 0; k < 3; k++) {
    (function (x) {
        handlers[x] = function () {
            alert(x);
        };
    })(k);
}

handlers[0](); // 0
```

ES6

```
for (let k = 0; k < 3; k++) {
    let x = k;
    handlers[x] = function () {
        alert(x);
    };
}

handlers[0](); // 0
```

let : definition, statement, expression

```
let x = 10;
let y = 20;
```

// let-definition

```
let (x = x * 2, y = 30) {
    console.log(x + y); // 50
}
```

// let-statement

```
console.log(x + y); // 30
```

```
console.log(let (x = 100) x); // 100
console.log(x); // 10
```

// let-expression

const : constants

const : constants

```
const MAX_SIZE = 100;

// cannot be redeclared (let, var, etc.)
let MAX_SIZE = 100; // error
```

const : constant functions

```
const registerUser() {
  // implementation
}

// redeclaration error (function, let, var)
function registerUser() { ... }
```

Function parameter
default values

Function parameter default values

```
function handleRequest(data, method) {
    method = method || “GET”;
    ...
}
```

Function parameter
default values

```
function handleRequest(data, method) {
    method = method || “GET”;
    ...
}

function handleRequest(data, method = “GET”) {
    ...
}
```

Destructuring
or “non-strict pattern-matching”

Destructuring: arrays

```
// for arrays
let [x, y] = [10, 20, 30]; // non-strict matching

console.log(x, y); // 10, 20
```

Destructuring: objects

```
// for objects
let user = {name: “Ann”, location: {x: 10, y: 20}};

let {name: n, location: {x: x, y: y}} = user;

console.log(n, x, y); // “Ann”, 10, 20
```

Destructuring of function parameters

```
function Panel(config) {
    var title = config.title;
    var x = config.pos[0];
    var y = config.pos[1];
    return title + x + y;
}

new Panel({title: “Users”, pos: [10, 15]});
```

Too “noisy”

Destructuring of function parameters

```
function Panel({title: title, pos: [x, y]}) {
    return title + x + y;
}

let config = {title: “Users”, pos: [10, 15]};

new Panel(config);
```

Destructuring: exchange of variables

```
// exchange two variables without third?

let x = 10;
let y = 20;

[x, y] = [y, x];  // easy
```

Replacement of arguments: ”rest” and ”spread”

Object arguments

```
// ES3, ES5

function format(pattern /*, rest */) {
    var rest = [].slice.call(arguments, 1);
    var items = rest.filter(function (x) { return x > 1});
    return pattern.replace("%v", items);
}

format("scores: %v", 1, 5, 3); // scores: 5, 3
```

Good bye, arguments

```
// ES3, ES5

function format(pattern /*, rest */) {
   var rest = [].slice.call(arguments, 1); // complicated
   var items = rest.filter(function (x) { return x > 1});
   return pattern.replace("%v", items);
}

format("scores: %v", 1, 5, 3); // scores: 5, 3
```

Hello, ”rest”

```
// ES6 aka Harmony

function format(pattern, ...rest) {     // real array
    var items = rest.filter(function (x) { return x > 1});
    return pattern.replace(“%v”, items);
}

format(“scores: %v”, 1, 5, 3); // scores: 5, 3
```

And also ”spread”

```
// ES6 aka Harmony

function showUser(name, age, weight) {
   return name + “:” + age + weight;
}

let user = [“Alex”, 28, 130];

showUser(…user); // ok
showUser.apply(null, user); // desugared
```

”rest” of arrays with destructuring

```
// ES6 aka Harmony

let userInfo = [“John”, 14, 21, 3];
let [name, ...scores] = userInfo;

console.log(name); // “John”
console.log(scores); // [14, 21, 3]
```

Short notations

Short notation in destructuring

```
let 3DPoint = {x: 20, y: 15, z: 1};

// full notation
let {x: x, y: y, z: z} = 3DPoint;

// short notation
let {x, y, z} = 3DPoint;
```

Short syntax of functions. -> functions

```
// casual functions
[1, 2, 3].map(function (x) { return x * x; }); // [1, 4, 9]

// -> functions
[1, 2, 3].map((x) -> x * x); // [1, 4, 9]
```

Syntactically:

- optional **return**;
- **->** instead of **function**
- No curly braces are required

-> functions: examples

```
// Empty arrow function is minimal-length
let empty = ->;

// Expression bodies needs no parentheses or braces
let square= (x) -> x * x;

// Without parameters
let getUser = -> users[current];

// Statement body needs braces
let users = [{name: “Mark”, age: 28}, {name: “Sarah”, age: 26}];
users.forEach((user, k) -> { if (k > 2) console.log(user.name, k) });
```

simple functions: dynamic this

```
function Account(customer, cart) {
    this.customer = customer;
    this.cart = cart;
    $(‘#shopping-cart').on('click',  function (event) {
        this.customer.purchase(this.cart); // error on click
    });
}
```

Solutions:

```
var that = this;
.bind(this)
```

=> functions: lexical this

```
function Account(customer, cart) {
    this.customer = customer;
    this.cart = cart;
    $(‘#shopping-cart').on('click', (event) =>
        this.customer.purchase(this.cart); // ok
    );
}
```

But... Currently block functions are on agenda!

Short syntax of functions. Block-functions

```
// casual functions
[1, 2, 3].map(function (x) { return x * x; }); // [1, 4, 9]

// block-functions, Ruby's way
[1, 2, 3].map {|x| x * x} // [1, 4, 9]
```

Syntactically:

- optional **return**;
- |x| instead of **function**
- No call parens

Proxy objects : meta level

Proxy-objects

```
/* handler –meta-level handler
 * proto – prototype of the proxy object */
Proxy.create(handler, [proto])

/* handler – meta-handler
 * call – call trap
 * construct – construction trap */
Proxy.createFunction(handler, [call, [construct]])
```

See: http://wiki.ecmascript.org/doku.php?id=harmony:proxies

Proxy-objects

```
// original object
let point = {
    x: 10,
    y: 20
};
```

```
// proxied object
let loggedPoint = Proxy.create({
    get: function (rcvr, name) {
        console.log("get: ", name);
        return point[name];
    },
    set: function (rcvr, name, value) {
        console.log("set: ", name, value);
        point[name] = value;
    }
}, Object.getPrototypeOf(point));
```

Trap of getting of properties

Trap of setting the properties

Proxy-objects

Meta-handler

```
// reading trap
loggedPoint.x; // get:  x, 10

// writing trap
loggedPoint.x = 20; // set:  x, 20

// reflected on the original object
point.x; // 20
```

```
// proxied object
let loggedPoint = Proxy.create({
  get: function (rcvr, name) {
    console.log("get: ", name);
    return point[name];
  },
  set: function (rcvr, name, value) {
    console.log("set: ", name, value);
    point[name] = value;
  }
}, Object.getPrototypeOf(point));
```

Callable Proxy-objects

```
// original object
let point = {x: 10, y: 20};

function callTrap() {
    console.log("call");
}

function constructTrap() {
    console.log("construct");
}

loggedPoint(10, 20);
new loggedPoint(100);
```

```
// proxied object
let loggedPoint = Proxy.createFunction({
    get: function (rcvr, name) {
        console.log("get: ", name);
        return foo[name];
    },
    set: function (rcvr, name, value) {
        console.log("set: ", name, value);
        foo[name] = value;
    }
}, callTrap, constructTrap);
```

Catching of calling

Catching of construction

Proxy : simple generic read logger

```
function logged(object) {
   return Proxy.create({
      get: function (rcvr, name) {
         console.log("get:", name);
         return object[name];
      }
   }, Object.getPrototypeOf(object));
}

let connector = logged({
   join: function (node) { ... }
});

connector.join("store@master-node"); // get: join
```

Proxy : examples

```
// loggers (on reading and writing)
Proxy.create(logHandler(object));

// multiple inheritance (delegation-based mixins)
Proxy.create(mixin(obj1, obj2));

// noSuchMethod
Proxy.create(object, noSuchMethod)

// Arrays with negative indices (as in Python)
let a = Array.new([1, 2, 3]);
console.log(a[-1]); // 3
a[-1] = 10; console.log(a); // [1, 2, 10]
```

See: https://github.com/DmitrySoshnikov/es-laboratory/tree/master/examples

Modules system

Modules in ES3, ES5

```
var DBLayer = (function (global) {
    /* save original */
    var originalDBLayer = global.DBLayer;
    function noConflict() {
        global.DBLayer = originalDBLayer;
    }
    /* implementation */
    function query() { ... }
    /* exports, public API */
    return {
        noConflict: noConflict,
        query: query
    };
})(this);
```

1. Create local scope
2. Restoring function
3. Implementation
4. Public API

Modules in ES3, ES5

```
var DBLayer = (function (global) {
    /* save original */
    var originalDBLayer = global.DBLayer;
    function noConflict() {
        global.DBLayer = originalDBLayer;
    }
    /* implementation */
    function query() { ... }
    /* exports, public API */
    return {
        noConflict: noConflict,
        query: query
    };
})(this);
```

1. Create local scope
2. Restoring function
3. Implementation
4. Public API

Too much of **“noise”**.
A **“sugar”** is needed.

Modules in ES6

```
module DBLayer {
   export function query(s) { ... }
   export function connection(...args) { ... }
}

import DBLayer.*; // import all

import DBLayer.{query, connection: attachTo}; // import only needed exports

query("SELECT * FROM books").format("escape | split");

attachTo("/books/store", {
   onSuccess: function (response) { ... }
})
```

External modules in ES6

```
// on file system
module $ = require("./library/selector.js");

// globally, from the Net, we define the name of the module
module CanvasLib = require("http:// ... /js-modules/canvas.js");

// use either directly
let rect = new CanvasLib.Rectangle({width: 30, height: 40, shadow: true});

// or import needed exports
import CanvasLib.{Triangle, rotate};

rotate(-30, new Triangle($.query(...params)));
```

Require module and import with destructuring

```
// require the module and directly
// import via pattern-matching
let {read, format} = require("fs.js");

// use the imported binding
read("storage/accounts.dat"). format("%line: value")
```

Generators : iterators, coroutines/multi-tasks

Generators : yield
"infinite" streams

```
function fibonacci() {
   let [prev, curr] = [0, 1];
   while (true) {
      [prev, curr] = [curr, prev + curr];
      yield curr;
   }
}

for (let n in fibonacci()) {
   // truncate the sequence at 1000
   if (n > 1000) break; // 1, 2, 3, 5, 8 ...
   console.log(n);
}
```

Next
entry point

Manual iteration:

```
let seq = fibonacci();
seq.next(); // 1
seq.next(); // 2
seq.next(); // 3
seq.next(); // 5
seq.next(); // 8
```

Generators : yield
custom iterators

```
function iterator(object) {
  for (let k in object) {
    yield [k, object[k]];
  }
}

let foo = {x: 10, y: 20};

for (let [k, v] in iterator(foo)) {
  console.log(k, v); // x 10, y 20
}
```

Proposed iterators of ES6:

```
// by properties (key+value)
for (let [k, v] in properties(foo))

// by values
for (let v in values(foo))

// by property names
for (let k in keys(foo))
```

See: http://wiki.ecmascript.org/doku.php?id=strawman:iterators

See: https://gist.github.com/865630

Generators : yield
asynchronous programming

Callbacks

```
xhr("data.json", function (data) {
   xhr("user.dat", function (user) {
      xhr("user/save/", function (save) {
         /* code */
      }
   }
});

/* other code */
```

Coroutines

```
new Task(function () {
  let data  = yield xhr("data.json“);
  let user  = yield xhr(“user.dat“);
  let save  = yield xhr(“/user/save“);
  /* code */
});


/* other code */
```

Generators : yield
cooperative multitasks (threads)

```
let thread1 = new Thread(function (...args) {
   for (let k in values([1, 2, 3])) yield k + " from thread 1";
}).start();

let thread2 = new Thread(function (...args) {
   for (let k in values([1, 2, 3])) yield k + " from thread 2";
}).start();

// 1 from thread 1
// 2 from thread 1
// 1 from thread 2
// 3 from thread 1
// 2 from thread 2
// etc.
```

Array comprehensions

Array comprehensions

```
// map + filter way

let scores = [1, 7, 4, 9]
    .filter(function (x) { return x > 5 })
    .map(function (x) { return x * x }); // [49, 81]
```

Array comprehensions

```
// map + filter way

let scores = [1, 7, 4, 9]
    .filter(function (x) { return x > 5 })
    .map(function (x) { return x * x }); // [49, 81]


// array comprehensions

let scores = [x * x for (x in values([1, 7, 4, 9])) if (x > 5)];
```

Thanks for your attention

Dmitry Soshnikov

dmitry.soshnikov@gmail.com

http://dmitrysoshnikov.com

@DmitrySoshnikov