Javascript, The Swiss Army Knife of Programming Languages

David Morcillo

23-11-2013

About me

- twitter.com/ultrayoshi
- github.com/ultrayoshi

Features

- Loosely typed language

Features

- Loosely typed language
- Object literal notation

Features

- Loosely typed language
- Object literal notation
- Prototypal inheritance

Features

- Loosely typed language
- Object literal notation
- Prototypal inheritance
- Global variables

Features

- Loosely typed language
- Object literal notation
- Prototypal inheritance
- Global variables
- Functions are first class objects

Features

ECMAScript

The standard that defines JavaScript is the third edition of *ECMAScript Programming Language*.

Hello World

index.html

```
<html>
  <head>
    <script>
      document.writeln(‘‘Hello, world!’’);
    </script>
  </head>
  <body>
  </body>
</html>
```

Syntax

Comments

Block comments formed with /* */ and line-ending comments starting with //. Example:

```
/*
  We are learning Javascript and comments are very important
*/
document.writeln(‘‘Hello World!’’); // Output: Hello World!
```

Syntax

Comments

Block comments formed with /* */ and line-ending comments starting with //. Example:

```
/*
  We are learning Javascript and comments are very important
*/
document.writeln(‘‘Hello World!’’); // Output: Hello World!
```

Names

Starts with a letter or underscore and optionally followed by on or more letters, digits or underscores. Beware of some reserved words.

```
bullet           // valid     _mana             // valid
3force           // invalid   lucky42           // valid
rocket-launcher  // invalid   grenade_launcher  // valid
```

Syntax

Numbers

Single number type represented internally as 64-bit floating point.

```
42
3.141516
10e5
1/0 // Output: Infinity
0/0 // Output: NaN
```

Syntax

Numbers

Single number type represented internally as 64-bit floating point.

```
42
3.141516
10e5
1/0 // Output: Infinity
0/0 // Output: NaN
```

Strings

Can be wrapped in single quotes or double quotes. It can contains 0 or more characters. All characters in Javascript are 16 bits wide.

```
‘‘Hello World’’
’Hello World’
‘‘This is\n a multiline string’’
’You can write ‘‘ on single quotes string’
```

Syntax

Functions

```
function helloWorld (name) {
    console.log('Hello ' + name + '!');
}

helloWorld('David'); // Output 'Hello David!'

var myFunction = function () {
    console.log('Hi there!');
};

myFunction(); // Output: 'Hi there!'
```

Syntax

Variables

Use the `var` keyword followed by a name to declare a variable. When used inside of a function, the `var` statement defines the function's private variables.

```
var player; // variable player declared on a global scope

function test() {
    var enemy; // Scoped to function test
}
```

Syntax

if, else

```
var testOk = true;

if (testOk) {
    console.log(‘‘Captain obvious’’);
} else {
    console.log(‘‘I’m bored’’);
}
```

Here are the *falsy* values:

- `false`
- `null`
- `undefined`
- The empty string
- The number 0
- The number NaN

All other values are *truthy*.

Syntax

switch

```
var weapon = ‘‘rocketlauncher’’;

switch(weapon) {
    case ‘‘pistol’’:
        console.log(‘‘piu piu’’);
        break;
    case ‘‘shotgun’’:
        console.log(‘‘paaam!’’);
        break;
    case ‘‘rocketlauncher’’
        console.log(‘‘BOOOOM!’’);
        break;
    default:
        console.log(‘‘falcon punch!’’);
        break;
}
```

Syntax

while, do while

```
var counter = 0;
while (counter < 10) { // Ends when counter is equal to 10
    console.log(counter);
    counter += 1;
}

do {
    console.log(counter);
    i -= 1;
} while(counter > 0); // Ends when counter is equal to 0
```

Syntax

while, do while

```
var counter = 0;
while (counter < 10) { // Ends when counter is equal to 10
    console.log(counter);
    counter += 1;
}

do {
    console.log(counter);
    i -= 1;
} while(counter > 0); // Ends when counter is equal to 0
```

for

```
var i;

for (i = 0; i < 10; i += 1)
    console.log(i);
}
```

Syntax

Strict (in)equality

```
10 == '10' // Output: true, auto type coercion
10 === '10' // Output: false strict equality
10 != '10' // Output: false, auto type coercion
10 !== '10' // Output: true strict inequality
```

Syntax

Strict (in)equality

```
10 == '10' // Output: true, auto type coercion
10 === '10' // Output: false strict equality
10 != '10' // Output: false, auto type coercion
10 !== '10' // Output: true strict inequality
```

null and undefined

```
console.log(mario); // Error: mario is not defined

function exists (mario) {
  console.log(mario);
}

exists(); // Output undefined

console.log(null == undefined) // Output: true
console.log(null === undefined) // Output: false
```

Objects

- Objects in Javascript are mutable keyed collections.

Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.

Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.
- A property name can be any string.

Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.
- A property name can be any string.
- Objects can inherit properties of another through its prototype.

Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.
- A property name can be any string.
- Objects can inherit properties of another through its prototype.

`Prototype`

All objects created from object literals are linked to `Object.prototype`.

Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.
- A property name can be any string.
- Objects can inherit properties of another through its prototype.

`Prototype`

All objects created from object literals are linked to `Object.prototype`. If we try to retrieve a property value from an object, and if the object lacks the property name, then Javascript attempts to retrieve the property value from the prototype object.

Objects

Object.create

```
var soldier = {
    hp: 10,
    strength: 5,
    weapon: 'Pistol'
};

var knight = Object.create(soldier);
knight.weapon = 'Sword';
knight.shield = true;

console.log(knight.hp); // Output: 10
console.log(knight.weapon); // Output: 'Sword'
console.log(knight.shield); // Output: true
```

Visit http://www.objectplayground.com/ for a graphical explanation

Objects

hasOwnProperty

```
knight.hasOwnProperty('hp'); // Output: false
knight.hasOwnProperty('shield'); // Output: true
```

Objects

hasOwnProperty

```
knight.hasOwnProperty('hp'); // Output: false
knight.hasOwnProperty('shield'); // Output: true
```

for in

```
for (attr in knight) {
    if(knigth.hasOwnProperty(attr)) {
        console.log('Knight property ' + attr + ' with value ' +
            knight[attr]);
    }
}
// Output: Knight property shield with value true
```

Objects

hasOwnProperty

```
knight.hasOwnProperty('hp'); // Output: false
knight.hasOwnProperty('shield'); // Output: true
```

for in

```
for (attr in knight) {
    if(knigth.hasOwnProperty(attr)) {
        console.log('Knight property ' + attr + ' with value ' +
            knight[attr]);
    }
}
// Output: Knight property shield with value true
```

delete

```
console.log(knight.weapon); // Output: 'Sword'
delete knight.weapon;
console.log(knight.weapon); // Output: 'Pistol'
```

Functions

Functions are the **fundamental modular unit** of Javascript. They are used for code reuse, information hiding, and composition.
The thing that is special about functions is that they can be invoked.

Functions

Functions are the **fundamental modular unit** of Javascript. They are used for code reuse, information hiding, and composition.
The thing that is special about functions is that they can be invoked.

Function.prototype and constructor

Functions are objects linked to Function.prototype. Every function object is also created with a prototype property. Its value is an object with a constructor property whose value is the function.

Functions

Functions are the **fundamental modular unit** of Javascript. They are used for code reuse, information hiding, and composition.
The thing that is special about functions is that they can be invoked.

Function.prototype and constructor

Functions are objects linked to Function.prototype. Every function object is also created with a prototype property. Its value is an object with a constructor property whose value is the function.

Functions

Invoking a function suspends the execution of the current function, passing control and parameters to the new function. In addition to the declared parameters, every function receives two additional parameters: `this` and `arguments`.

Functions

Invoking a function suspends the execution of the current function, passing control and parameters to the new function. In addition to the declared parameters, every function receives two additional parameters: `this` and `arguments`.

Invocation (1/4): Method invocation pattern

```
var enemy = {
    hp: 5,
    rage: 0,
    attack: function () {
        this.rage += 1;
    }
};

enemy.attack();
console.log(enemy.rage); // Output: 1
```

Functions

Invocation (2/4): Function invocation pattern

```
physicsManager.collisionsDetected = 0;

physicsManager.checkCollision = function (entity1, entity2) {
    var bbCollision = function (bb1, bb2) {
        var collision = false;
        // Collision code skipped
        if (collision) {
            // WARNING: 'this' is the global object and not 'physicsManager'
            this.collisionsDetected += 1;
        }
        return collision;
    };

    bbCollision(entity1.getBB(), entity2.getBB());
};

if (physicsManager.checkCollision(enemy, player)) {
    player.takeDamage(enemy.strength);
}
```

Functions

Invocation (2/4): Function invocation pattern (workaround)

```
physicsManager.collisionsDetected = 0;

physicsManager.checkCollision = function (entity1, entity2) {
    var that = this;

    var bbCollision = function (bb1, bb2) {
        var collision = false;
        // Collision code skipped
        if (collision) {
            that.collisionsDetected += 1;
        }
        return collision;
    };

    bbCollision(entity1.getBB(), entity2.getBB());
};

if (physicsManager.checkCollision(enemy, player)) {
    player.takeDamage(enemy.strength);
}
```

Functions

Invocation (3/4): Constructor invocation pattern

```
var Player = function (name) {
    this.name = name;
    this.lives = 3;
};

Player.prototype.sayMyName = function () {
    console.log('My name is ' + this.name);
};

var david = new Player('David');
david.sayMyName(); // Output: 'My name is David'
```

Functions

Invocation (3/4): Constructor invocation pattern (without new)

```
var Player = function (name) {
    this.name = name;
    this.lives = 3;
};

Player.prototype.sayMyName = function () {
    console.log('My name is ' + this.name);
};

var david = Player('David'); // oops
david.sayMyName(); // raise an error because david is undefined

// Global variables feast
console.log(name); // Output: 'David'
console.log(lives); // Output: 3
```

Functions

Invocation (4/4): Apply invocation pattern

```
var enemy = {
    rage: 0,
    attack: function () {
        this.rage += 1;
    }
};

var anotherEnemy = {
    rage: 10
};

enemy.attack.apply(anotherEnemy, []);

console.log(anotherEnemy.rage); // Output: 11
```

Functions

Arguments

```
function doActions() {
    var i, l;

    // WARNING: arguments is an Array-like object
    for (i = 0, l = arguments.length; i < l; i += 1) {
        console.log('Doing action ' + arguments[i]);
    }
}

doActions('jump', 'attack');
/*
    Output:
    'Doing action jump'
    'Doing action attack'
*/
```

Functions

Closure

Javascript does have function scope. That means that the parameters and variables defined in a function are not visible outside of the function, and that a variable defined anywhere within a function is visible everywhere within the function.

```
var playe = new Player();

function isGameOver() {
    var enemy = new Enemy();

    function checkHit() {
        return enemy.hit(player);
    }

    return checkHit();
}

isGameOver();
```

Functions

Module pattern

```
var physicsModule = (function () { // IIEF pattern
   var detectedCollisions = 0;

   function checkBBCollision(bb1, bb2) {
      var collision = false;
      // collision code skipped
      if (collision) {
          detectedCollisions += 1;
      }
      return collision;
   }

   function checkCollision(entity1, entity2) {
      checkBBCollision(entity1.getBB(), entity2.getBB());
   }

   return {
      checkCollision: checkCollision
   };
}());
```

Inheritance

Javascript provides a much richer set of code reuse patterns. It can ape the classical pattern, but it also supports other patterns that are more expressive.

Inheritance

Javascript provides a much richer set of code reuse patterns. It can ape the classical pattern, but it also supports other patterns that are more expressive.

Javascript is a class-free language

In classical languages, objects are instances of classes, and a class can inherit from another class. Javascript is a prototypal language, which means that objects inherit directly from other objects.

Inheritance

Pseudoclassical pattern

```
var Alien = function (name) {
    this.name = name;
};

Alien.prototype.talk = function () {
    console.log('%?saf? ' + this.name);
};

var SmartAlien = function (name) {
    this.name = name;
};

SmartAlien.prototype = new Alien();

SmartAlien.prototype.speech = function () {
    this.talk();
    console.log('...I mean, my name is ' + this.name);
};

var enemy = new SmartAlien('Roger');
enemy.speech();
// Output: '%?saf? Roger
//          ...I mean, my name is Roger'
```

Inheritance

Prototypal pattern

```
var alien = {
    name: '%?&789',
    talk: function () {
        console.log('%&7?_% ' + this.name);
    }
};

var smartAlien = Object.create(alien);
smartAlien.speech = function () {
    this.talk();
    console.log('...I mean, my name is ' + this.name);
};

var enemy = Object.create(smartAlien);
enemy.name = 'Roger';
enemy.speech();
// Output: '%?saf? Roger
//          ...I mean, my name is Roger'
```

Inheritance

Functional pattern

```
var alien = function (spec) {
    var that = {};

    var killHumans = function () { // Private access
      console.log('*Using ' + spec.weapon + '*');
    };

    that.talk = function () {
        console.log('%&78 ' + spec.name);
        if (spec.weapon) {
            killHumans();
        }
    };

    return that;
};

var enemy = smartAlien({ name: 'Roger' });
enemy.speech();
// Output: '%?saf? Roger
//          ...I mean, my name is Roger'
//
// ...killing humans on the process
```

```
var smartAlien = function (spec) {
    spec.weapon = 'Pistol'; // Private access
    var that = alien(spec);

    that.speech = function () {
        that.talk();
        console.log('...I mean, my name is ' +
          spec.name);
    };
    return that;
};
```