



Festival Latinoamericano de Instalación
de Software Libre



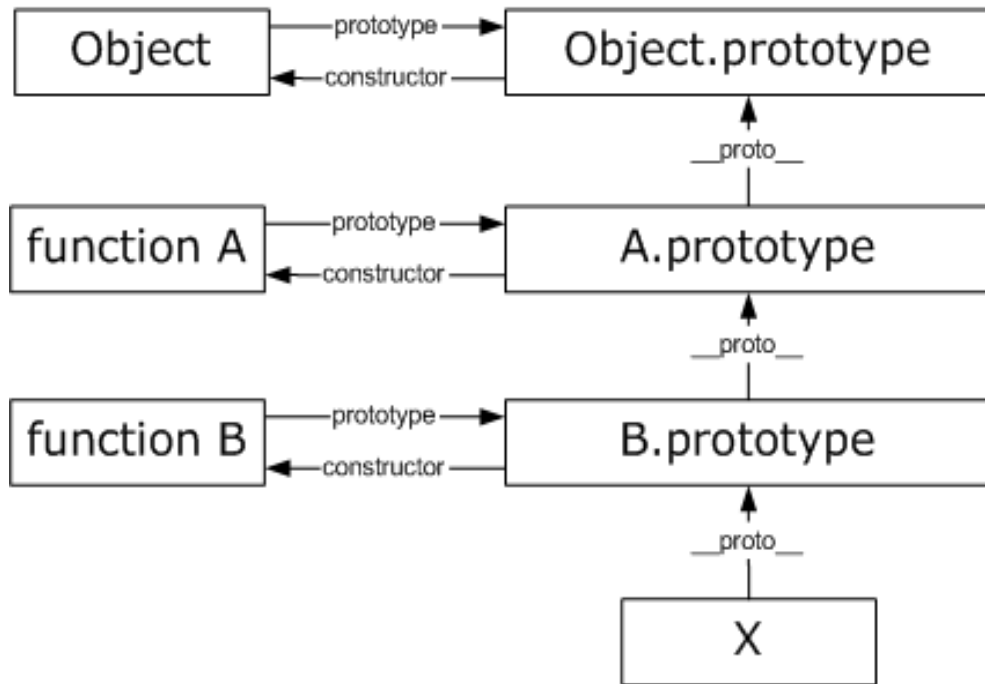
Orientação a objetos com Javascript!

Christiano Milfont



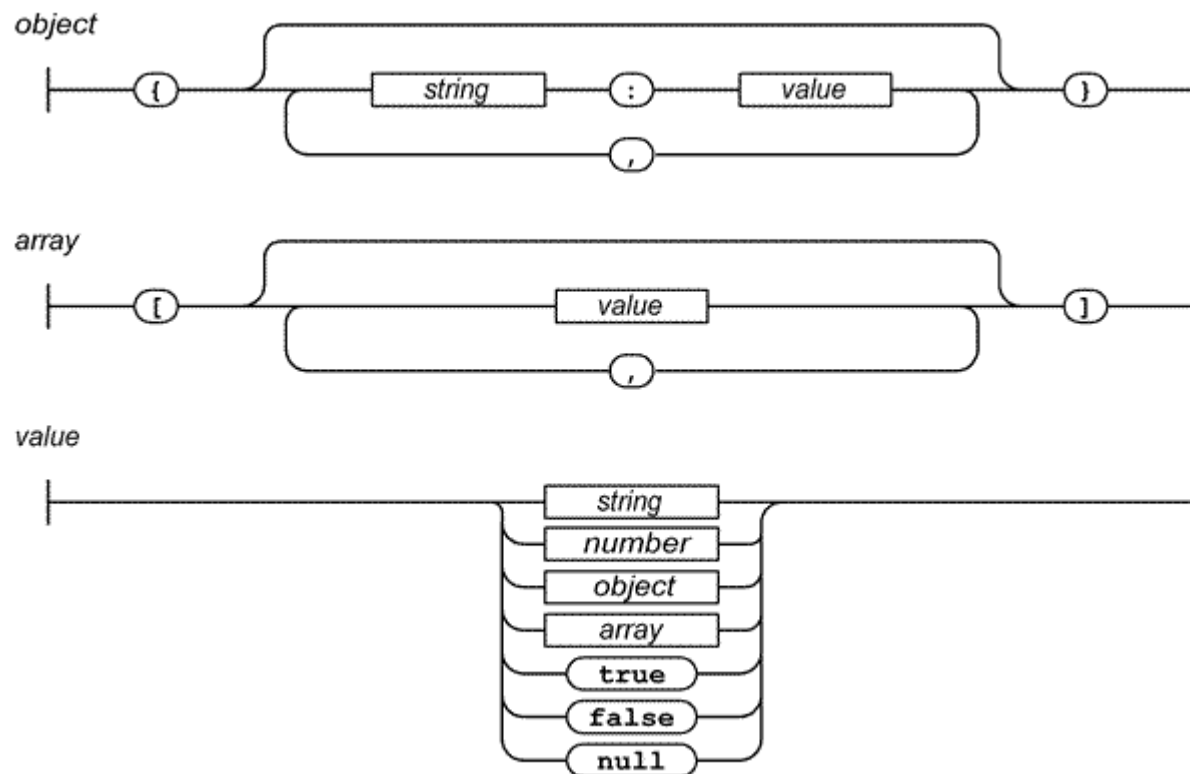
"Procedural code gets information then makes decisions. Object-oriented code tells objects to do things."
Alec Sharp

Prototype-based



```
if (typeof TRIADWORKS == "undefined") { var TRIADWORKS = {}; }  
TRIADWORKS.prototype.MemoryProxy = function(data) {  
  TRIADWORKS.MemoryProxy.superclass.constructor.call(this);  
  this.data = data;  
};
```

JSON (Javascript Object Notation)



```
var obj = {  
  a : 'variavel string', // Uma vari  
  b : ['', 'bj2', 'three', 4, 'obj5'], // Uma variável que recebe um array  
  c : function() { // uma função  
    alert(this.a);  
  }  
}  
  
//Como em javascript a declaração já é a instanciação,  
// podemos fazer referencia direta ao objeto criado.  
obj.c(); //Reposta: "variavel string"  
alert(obj.a); //Reposta: "variavel string"
```

Factory Vs Construtor Vs Prototype

```
function createObject() {
    var temp = new Object;
    temp.name = "Triadworks";
    temp.value = '$10000000';
    temp.show = function(){
        alert("Os mais bonitos");
    }
    return temp;
}
var temp1 = createObject();
var temp2 = createObject();

function createObject(name, value) {
    var temp = new Object;
    temp.name = name;
    temp.value = value;
    temp.show = function(){
        alert("Os mais bonitos");
    }
    return temp;
}
var temp1 = createObject('Milfont', '$10000000');
var temp2 = createObject('Chris', '$10000000');
```

```
function Triadworks(name, value){
    this.name = name;
    this.value = value;
}
var temp1 = new Triadworks("Handerson", "-1");

function Triadworks(){};
Triadworks.prototype.name = '';
Triadworks.prototype.value = '';
Triadworks.prototype.setValue = function(value){
    this.value = value;
}
```

Private vs Public

```
function objetoY() {
    var variavel_privada = "Essa variável é privada";
    this.variavel_publica = "Essa variável é pública";
}

var ExemploFLISOL = function(){
    this.itemPublico = 'ok!';
    var itemPrivado = "nao pode acessar";
    return {
        getItemPrivado:function(){

        }
    }
}();
var exemplo = new ExemploFLISOL();
exemplo.getItemPrivado();
```

Herança

```
//herança
function Passaro() {
    this.nome = 'papagaio';
}
function Produto() {
    this.valor = 'R$10,00';
}
Passaro.prototype = new Produto;
Passaro.prototype.imprimir = function() { return this.valor + ' - ' + this.nome; }
var passaro = new Passaro();
alert(passaro instanceof (Produto)); // saida: true
alert(passaro instanceof (Passaro)); // saida: true
alert(passaro.imprimir()); // saida: "R$10,00 - papagaio"
```

Polimorfismo

```
//polimorfismo
var shape = new Shape();
var circle = new Circle( 10 );
var rectandle = new Rectangle( 10, 20 );
shape.Draw(); // Alert: Drawing generic shape
circle.Draw(); // Alert: Drawing circle
rectangle.Draw(); // Alert: Drawing rectangle
//O método Draw() é polifórmico porque tem diferentes
// características para invocação de cada objeto.
```

Associação

```
//associacao
function Associacao() {
    this.composicao = new Passaro();
    this.agregacao = function() {
        return new Passaro();
    }
}

var teste = new Associacao();
alert(teste.composicao.valor);
alert(teste.agregacao().valor);
```

Overloading e Overriding

```
var Objeto = new function(){
  return {
    teste: function(){
      alert("Sem parametros");
    },
    teste: function(x, y){
      alert(x + ' overrindig ' + y);
    }
  }
}();
//A execução do metodo teste mesmo
//que não contenha parâmetros apresentaria:
//"undefined overriding undefined".
```

Alternativa de Overriding

```
var Objeto = new function(){  
  return {  
    metodo: function(){  
      var argumentos = arguments.length;  
      switch(argumentos){  
        case 0 : alert('Function sem argumentos'); break;  
        case 1 : alert(arguments[0]); break;  
        case 2 : alert(arguments[0] + ' - ' + arguments[1]); break;  
      }  
    }  
  }  
}();
```

Fim! haaaaaaaaaaaaa...