Curso de Javascript

Unidad Didáctica 08: Objetos

http://cursosdedesarrollo.com/

Ayuntamiento de Vitoria-Gasteiz
Vitoria-Gasteizko Udala

Índice de contenidos

- Introducción
- Representación de objetos
- Acceso
    - Modificación
    - Borrado
    - Paso por referencia
    - Enumeración
    - Objetos globales
    - Conclusiones

http://cursosdedesarrollo.com/

Introducción

Los tipos primitivos en JavaScript son undefined, null, boolean, number y string

Aunque pueda resultar extraño, el resto de elementos en JavaScript son objetos, tanto las funciones, arrays, expresiones regulares como los propios objetos

http://cursosdedesarrollo.com/

Introducción

Un objeto en JavaScript es un contenedor de propiedades, donde una propiedad tiene un nombre y un valor

El nombre de una propiedad puede ser una cadena de caracteres, incluso una vacía

Introducción

El valor de la propiedad puede ser cualquier valor que podamos utilizar en JavaScript, excepto undefined

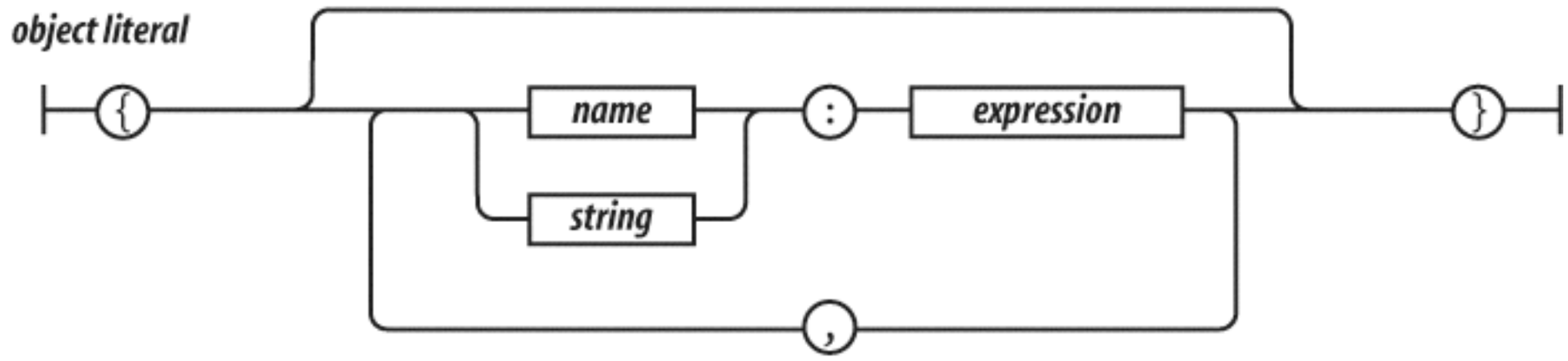
Introducción

En JavaScript, los objetos son básicamente tablas hash, esto es, un grupo de propiedades y funciones que pueden ser accedidos a través de una clave

Tanto las propiedades como los métodos (que no son más que propiedades cuyo valor es una función) pueden ser creados dinámicamente en tiempo de ejecución

http://cursosdedesarrollo.com/

Representación de Objetos



http://cursosdedesarrollo.com/

Representación de Objetos

La manera más simple de construir un objeto en JavaScript, es declarar una nueva variable de tipo Object y asignarle las propiedades o métodos que el objeto necesite:

```
var obj = new Object();

obj.foo = "bar";

obj.hello = function() { console.log("Hello world!"); }
```

http://cursosdedesarrollo.com/

Representación de Objetos

Existe una manera más rápida (en términos de caracteres), de crear un objeto utilizando la denominada notación literal

Esta representación consiste en asignar a una variable una pareja de llaves ({ }) para crear un objeto vacío, y asignar las propiedades y métodos de la misma manera que en el ejemplo anterior

http://cursosdedesarrollo.com/

Representación de Objetos

```
var obj   = {};

obj.foo   = "bar";

obj.hello = function() { console.log("Hello world!"); }
```

```
var obj = {

"foo"   : "bar",

"hello" : function() { console.log("Hello world!"); }

};
```

http://cursosdedesarrollo.com/

Representación de Objetos

Como hemos dicho, el nombre de una propiedad puede ser un string, incluso uno vacío

Las comillas dobles alrededor del nombre de la propiedad son opcionales, si el nombre es un nombre legal de JavaScript, y no es una palabra reservada

Representación de Objetos

Así, las comillas dobles son necesarias para un nombre de propiedad como "first-name", pero no son necesarias para first_name

Los valores de las propiedades pueden ser cualquier expresión, incluso la definición de otro objeto

Representación de Objetos

var flight = {

airline: "Oceanic",

number: 815,

departure: { IATA: "SYD", time: "2004-09-22 14:55", city: "Sydney"},

arrival: { IATA: "LAX", time: "2004-09-23 10:42", city: "Los Angeles"}

};

http://cursosdedesarrollo.com/

Representación de Objetos

var flight = {

airline: "Oceanic",

number: 815,

departure: { IATA: "SYD", time: "2004-09-22 14:55", city: "Sydney"},

arrival: { IATA: "LAX", time: "2004-09-23 10:42", city: "Los Angeles"}

};

http://cursosdedesarrollo.com/

Representación de Objetos

En JavaScript, los objetos son dinámicos, esto quiere decir que sus propiedades no tienen por qué ser definidas en el momento en el que creamos el objeto

Podemos añadir nuevas propiedades al objeto en tiempo de ejecución, tan solo indicando el nombre la propiedad y asignándole un valor o función

http://cursosdedesarrollo.com/

Representación de Objetos

```
var cat = new Object();

cat.name = "Rufus";

cat.species = "cat";

cat.hello = function() { console.log("miaow"); }
```

http://cursosdedesarrollo.com/

Representación de Objetos

Como podemos observar en la propiedad hello, su valor no es un tipo primitivo sino una función

Esto es posible porque en JavaScript las funciones son un tipo especial de objetos, que podemos tratar de la misma manera que un string o un número

Las propiedades que tienen como valor una función, son tratadas como métodos del objeto

http://cursosdedesarrollo.com/

Representación de Objetos

En estos casos, no suele ser necesario definir la función anteriormente, pudiendo utilizar una función anónima para declarar el método

Representación de Objetos

NOTACIÓN LITERAL

Crear objetos utilizando una notación literal es muy útil en situaciones en las que queremos pasar una serie de datos relacionados a una función o configurar una librería externa

http://cursosdedesarrollo.com/

Representación de Objetos

Imaginemos una situación en la que utilizamos una librería que nos permite crear una galería de imágenes

Una posible solución sería definir un método para cada una de las propiedades de la galería, de manera que pudiésemos llamar a cada uno de esos métodos con el valor correspondiente

Representación de Objetos

Esto supondría tener dos métodos por cada propiedad (getter y setter), así como una llamada para cada propiedad que deseemos configurar

Utilizando una notación literal de objetos, podemos ahorrarnos todo este código y configurar la librería utilizando un único objeto

Representación de Objetos

```
domElement.fancybox({

maxWidth    : 800, maxHeight   : 600,

fitToView   : false, width       : '70%',

height      : ’70%', autoSize    : false,

closeClick  : false, openEffect  : 'none',

closeEffect : 'none'

});
```

http://cursosdedesarrollo.com/

Representación de Objetos

Otra de las ventajas que nos ofrece esta notación literal, es poder crear bibliotecas de utilidades, de la misma manera que podríamos crear una clase abstracta con métodos estáticos en otros lenguajes de programación con Java o C#

http://cursosdedesarrollo.com/

Representación de Objetos

```
var ArrayUtil = {

contains : function(array, element)

{for(var x=0; x<array .length; x++)

{if(array[x] === element){return true;}}

return false;

},

exclude : function(list, items)

{ ...},

makeList : function(list)

{...}

}

var list = ["A", "B", "C"];

console.log("¿Contiene A? " + ArrayUtil.contains(list, "A"));
```

http://cursosdedesarrollo.com/

Acceso

Es posible acceder a los valores de un objeto indicando el nombre de la propiedad dentro de unos corchetes ([ ]), como si accediésemos a un elemento de un array

http://cursosdedesarrollo.com/

Acceso

Si el nombre de la propiedad es un nombre legal de JavaScript, y no es una palabra reservada, se puede utilizar la notación .

Acceso

Es preferible utilizar la notación ., ya que es más corta y comúnmente utilizada como acceso a métodos y propiedades en lenguajes orientados a objetos:

```
flight["airline"]      // "Oceanic"

flight.departure.IATA   // "SYD"
```

http://cursosdedesarrollo.com/

Acceso

En algunas ocasiones, puede resultar interesante utilizar la notación corchetes ([ ]) para acceder a los elementos, ya que las claves son almacenadas como strings

Esto nos permite construir cadenas de caracteres en tiempo de ejecución, para acceder a las propiedades de los objetos

Acceso

Por ejemplo, podemos crear un bucle para iterar sobre las propiedades de un objeto:

for(var property in cat)

console.log(cat[property].toString());

var myObject = {

property1: "chocolate",

property2: "cake",

property3: "brownies"

}

for(var x=1; x<4; x++)

{console.log(myObject[“property" + x]);}

http://cursosdedesarrollo.com/

Acceso

El valor undefined es devuelto si se intenta acceder a una propiedad que no existe:

```
stooge["middle-name"]   // undefined

flight.status           // undefined

stooge["FIRST-NAME"]    // undefined
```


Acceso

El operador || puede ser utilizado para obtener valores por defecto:

```
var middle = stooge["middle-name"] || "(none)";

var status = flight.status || "unknown";
```

http://cursosdedesarrollo.com/

Acceso

Intentar acceder a los valores de una propiedad no definida, lanzará una excepción de tipo TypeError.

Esto puede evitarse utilizando el operador &&, para asegurarnos que el valor existe y es accesible:

```
flight.equipment       // undefined

flight.equipment.model // TypeError: Cannot read property 'model' of undefined

flight.equipment && flight.equipment.model  // undefined
```

http://cursosdedesarrollo.com/

Modificación

El valor de una objeto puede actualizarse a través de una asignación. Si el nombre de la propiedad existe en el objeto, su valor es reemplazado:

```
stooge['first-name'] = 'Arkaitz';
```


Modificación

Si la propiedad no existe en el objeto, esta nueva propiedad es añadida al objeto:

```
stooge['middle-name'] = 'Lester';

stooge.nickname = 'Curly';

flight.equipment = { model: 'Boeing 777' }

flight.status = 'overdue';
```

http://cursosdedesarrollo.com/

Borrado

El operado delete puede ser utilizado para eliminar la propiedad de un objeto

Este operador eliminará la propiedad de un objeto, si la tuviera, pero no afectará al resto de propiedades de los prototipos

http://cursosdedesarrollo.com/

Borrado

delete objeto.propiedad;

//la propiedad no existe ya en el objeto

Paso por Referencia

Los objetos siempre son accedidos como referencias, nunca se copia su valor cuando los asignamos a otros objetos, o los pasamos como parámetros en funciones

http://cursosdedesarrollo.com/

Paso por Referencia

```
var x = objeto;

x.propiedad = 'Curly';

// propiedad contiene 'Curly' porque x y objeto

// referencian al mismo objeto

var nick = objeto.propiedad;
```

http://cursosdedesarrollo.com/

Paso por Referencia

// a, b, y c hacen referencia a

// diferentes objetos vacíos

var a = {}, b = {}, c = {};

// a, b, y c hacen referencia al

// mismo objeto vacío

a = b = c = {};

http://cursosdedesarrollo.com/

Paso por Referencia

var obj { value = 5 };

console.log(obj.value); // o.value = 5

function change(obj)

{obj.value = 6;}

change(obj);

console.log(obj.value); // o.value = 6

http://cursosdedesarrollo.com/

Enumeración

La sentencia for...in puede iterar sobre todos los nombres de propiedades de un objeto

Esta iteración incluirá todas las propiedades, funciones y propiedades definidas en los prototipos, en las que no podemos estar interesados

Enumeración

La mejor manera de filtrar estas propiedades es a través de la función hasOwnProperty y el operador typeof:

```
var name;
    for (name in another_stooge) {
        if (typeof another_stooge[name] !== 'function') {
            document.writeln(name + ': ' + another_stooge[name]);
        }
    }
```

http://cursosdedesarrollo.com/

Enumeración

No hay ningún tipo de garantía en el orden en el que se van a mostrar las propiedades, por lo que si esto es importante, tendremos que controlarlo de alguna manera

Para ello, la mejor manera es olvidarnos de la sentencia for...in y acceder directamente a las propiedades concretas, en el orden que definamos

http://cursosdedesarrollo.com/

Enumeración

```
var i;

var properties = [ ‘first-name’,'middle-name','last-
name' ,'profession'];

    for (i = 0; i < properties.length; i += 1) {

        document.writeln(properties[i] + ': ' +

            another_stooge[properties[i]]);

    }
```

http://cursosdedesarrollo.com/

Objetos Globales

JavaScript permite crear variables globales de una manera muy sencilla

Desafortunadamente, las variables globales perjudican directamente la calidad de los programas, por lo que deben ser evitadas

http://cursosdedesarrollo.com/

Objetos Globales

Una manera de minimizar la utilización de variables globales, es crear una única variable global para toda la aplicación, que incluya el resto de variables:

```
var MYAPP = {};

    MYAPP.stooge = {

        "first-name": "Joe",

        "last-name": "Howard"

    };
```

http://cursosdedesarrollo.com/

Objetos Globales

```
MYAPP.flight = {
    airline: "Oceanic",
    number: 815,
    departure: {
        IATA: "SYD",
        time: "2004-09-22 14:55",
        city: "Sydney"
    },
    arrival: {
        IATA: "LAX",
        time: "2004-09-23 10:42",
        city: "Los Angeles"
    }
};
```

http://cursosdedesarrollo.com/

Objetos Globales

Reduciendo el número de variables globales a uno, se reduce de manera significativa la posibilidad de colisiones con otras aplicaciones, widgets o bibliotecas

Además, la aplicación puede leerse y entenderse de manera más sencilla

http://cursosdedesarrollo.com/

Conclusiones

Hemos visto cómo construir objetos desde cero en Javascript

http://cursosdedesarrollo.com/

Datos de Contacto

http://www.cursosdedesarrollo.com
info@cursosdedesarrollo.com

http://cursosdedesarrollo.com/

Licencia

David Vaquero Santiago