

JavaScript

Resumen

Carlos Ramírez

Programación orientada a objeto

La programación orientada a objetos es un paradigma de programación que utiliza la abstracción para crear modelos basados en el mundo real. Utiliza diversas técnicas de paradigmas previamente establecidas, incluyendo la **modularidad, polimorfismo y encapsulamiento**. Hoy en día, muchos lenguajes de programación (como Java, JavaScript, C#, C++, Python, PHP, Ruby y Objective-C) soportan programación orientada a objetos.

La programación orientada a objetos puede considerarse como el diseño de software a través de un conjunto de objetos que cooperan, a diferencia de un punto de vista tradicional en el que un programa puede considerarse como un conjunto de funciones, o simplemente como una lista de instrucciones para la computadora.

Terminología

Clase : Define las características del Objeto.

Objeto : Una instancia de una Clase.

Propiedad : Una característica del Objeto, como el color.

Método : Una capacidad del Objeto, como caminar.

Constructor : Es un método llamado en el momento de la creación de instancias.

Herencia : Una Clase puede heredar características de otra Clase.

Terminología

Encapsulamiento : Una Clase sólo define las características del Objeto, un Método sólo define cómo se ejecuta el Método.

Abstracción : La conjunción de herencia compleja, métodos, propiedades que un objeto debe ser capaz de simular en un modelo de la realidad.

Polimorfismo : Diferentes Clases podrían definir el mismo método o propiedad.

V8

V8 es un motor de código abierto para JavaScript. Compila y ejecuta el código JavaScript de origen, se encarga de la asignación de memoria para los objetos, y la basura se acumula objetos que ya no necesita.

Object.prototype.constructor

Retorna una referencia a la función del **Object** que creó el prototipo de la instancia. Note que el valor de esta propiedad es una referencia a la función misma, no a un string conteniendo el nombre de la función. El valor es solo de lectura para valores de primitivas tales como 1, true y 'test'.

Todos los objetos heredan una propiedad constructor la cual proviene de su prototipo:

```
var o = {};  
o.constructor === Object; // true
```

```
var a = [];  
a.constructor === Array; // true
```

```
var n = new Number(3);  
n.constructor === Number; // true
```

Scripting language

Un lenguaje de programación o lenguaje de script es un lenguaje de programación que soporte scripts, programas escritos para un ambiente especial en tiempo de ejecución que puede interpretar (en lugar de compilar) y automatizar la ejecución de tareas que, alternativamente, podría ser ejecutado de una en una por un ser humano operador. Los ambientes que se pueden automatizar mediante scripts incluyen aplicaciones de software, páginas web dentro de un navegador web, las conchas de los sistemas operativos (OS), y sistemas embebidos. Un lenguaje de script puede ser visto como un lenguaje de dominio específico para un entorno particular; en el caso de secuencias de comandos de una aplicación, esto también se conoce como un lenguaje de extensión. Los lenguajes de script también se refieren a veces como propios lenguajes de programación de alto nivel, ya que operan en un alto nivel de abstracción, o como lenguajes de control, en particular para lenguajes de control de trabajo en mainframes.

¿Que es JavaScript?

Es un lenguaje de programación orientado a objetos, es utilizado para crear eventos en páginas web.

Diferencia entre java y javaScript:

JavaScript:

Esta orientada a objetos. No se distingue entre tipos de objetos. La herencia es a través del mecanismo de prototipo, y las propiedades y métodos se puede añadir a cualquier objeto de forma dinámica. Los tipos de variables no se declaran.

Diferencia entre java y javaScript:

Java:

Java es un lenguaje de programación basada en clases.

Los objetos se dividen en clases e instancias con toda herencia a través de la jerarquía de clases. Las clases y los casos no pueden tener propiedades o métodos añadidos dinámicamente. Los tipos de variables deben ser declaradas.

Prototype

Un prototipo es un objeto del cual otros objetos heredan propiedades....

`.toFixed`= Redondea el número y pone la cantidad que le digamos de decimales,es decir mantiene un número en una cadena. Ejemplo =

```
35.678.toFixed(2); = 35.68
```

`toPrecision()`; = El método `toPrecision ()` formatea un número a una longitud especificada. Se añaden un punto decimal y los nulos (si es necesario), para crear la longitud especificada. Es decir le indico un número y el lo agarra de izquierda a derecha. Ejemplo=

```
var num = 13.3714;
```

```
var n = num.toPrecision(4);
```

```
Resultado = 13.37
```

Number() = La función Number () convierte un objeto a un número que representa el valor del objeto. Si el valor no se puede convertir a un número legal, se devuelve NaN. Si el parámetro es un objeto Date, la función Number () devuelve el número de milisegundos desde la medianoche 01 de enero 1970 UTC. Ejemplo =

```
var x1 = true;  
var x2 = false;  
var x3 = new Date();  
var x4 = "999";  
var x5 = "45 54";
```

```
var n =  
Number(x1) + "<br>" +  
Number(x2) + "<br>" +  
Number(x3) + "<br>" +  
Number(x4) + "<br>" +  
Number(x5);
```

Resultado =

```
true = 1  
false = 0  
date = 121212121212  
999 = 999  
45 54 = NaN
```

toString() = El método toString() convierte un número en una cadena.El método toString () devuelve el valor de un objeto String. Ejemplo =

```
var str = "Hello World!";
```

```
var res = str.toString();
```

Resultado = Hello World!

String() = Trata los valores antiguos que no se reconocen como objetos y los ve como objetos... Ejemplo = "hola"

Manejo de errores:

try : Co ´digo a intentar ejecutar .

catch : Bloque de co ´digo que se ejecuta si falla lo que esta dentro de try. *Finally*: Se ejecuta sin importar si hay errores o no.

```
try {  
    // sentencias a ejecutar que pueden generar excepciones
```

```
}  
catch (e) {
```

```
    // De generarse alguna excepcion este es capturado
```

```
    // y puede ser accederse a traves de la variable e.
```

```
}  
finally {
```

```
    // codigo que se ejecuta independientemente de si se
```

```
    // ejecuta el try o el catch o ambos.
```

```
}
```

Throw: Lanza una excepción definida por el usuario. Contiene una serie de características, sintaxis:

```
throw{
```

```
name: Nombre del error o excepción. message: Descripción del error
```

```
}
```


Función literal

Es la normal a la que se le da un nombre. En la práctica esta forma de escribir nos permite hacer desarrollos separados con funciones que alteren las mismas variables dentro de un mismo objeto. Básicamente, para escribir en notación literal hay que conocer dos formas de declarar los hijos y entender cómo igualar cualquier función a una variable.

```
miObjeto = {  
  propiedad1 : "valor de la propiedad", propiedad2 : 45,  
  metodoCualquiera : function (variable) {  
  
    alert(variable); }  
};
```

```
miObjeto.metodoCualquiera( miObjeto.propiedad1 );  
// provocará un alert que diga "valor de la propiedad"
```

Función Constructora

Para definir un tipo de objeto, cree una función para el tipo de objeto que especifique su nombre, propiedades y métodos. Toda función de tipo constructora su nombre va capitalizada (su inicial va en mayúscula).

Defina el tipo de objeto escribiendo una función constructora.

Cree una instancia del objeto con la sentencia new.

Ejemplo:

```
function carro(fabricante, modelo, año) {  
  this.fabricante = fabricante;  
  this.modelo = modelo;  
  this.año = año;  
}
```

Nótese el uso de this para asignar valores a las propiedades del objeto basadas en los valores pasados a la función.

Ahora puede crear un objeto llamado micarro como sigue:

```
micarro = new Carro("Eagle", "Talon TSi", 1993);
```

Closure

Los closures son funciones que manejan variables independientes. En otras palabras, la función definida en el closure "recuerda" el entorno en el que se ha creado. Hay mucho aquí.

En los ejemplos anteriores cada closure ha tenido su propio entorno; aquí creamos un único entorno compartido por tres funciones: `Counter.increment`, `Counter.decrement` y `Counter.value`. El entorno compartido se crea en el cuerpo de una función anónima, que se ejecuta en el momento que se define. El entorno contiene dos elementos privados: una variable llamada `privateCounter` y una función llamada `changeBy`. No se puede acceder a ninguno de estos elementos privados directamente desde fuera de la función anónima. Se accede a ellos por las tres funciones públicas que se devuelven desde el contenedor anónimo.

```
var counter = (function() {  
  var privateCounter = 0;  
  function changeBy(val) {  
    privateCounter += val;  
  }  
  return {  
    increment: function() {  
      changeBy(1);  
    },  
    decrement: function() {  
      changeBy(-1);  
    },  
    value: function() {  
      return privateCounter;  
    }  
  };  
})();
```

```
alert(counter.value()); /* Alerts 0 */  
counter.increment();  
counter.increment();  
alert(counter.value()); /* Alerts 2 */  
counter.decrement();  
alert(counter.value()); /* Alerts 1 */
```

string.slice(start,end):

El método slice () extrae partes de una cadena y devuelve las partes extraídas en una nueva cadena.

Utilice los parámetros de inicio y finalización para especificar la parte de la cadena que desea extraer.

El primer carácter tiene la posición 0, el segundo tiene la posición 1, y así sucesivamente.

Ejemplo

Extraer partes de una cadena:

```
var str = "Hello world!";  
var res = str.slice(1,5);
```

El resultado de *res* será:

ello

string.split(separator, limit):

El método split () se utiliza para dividir una cadena en una matriz de subcadenas, y devuelve la nueva matriz.

Consejo: Si una cadena vacía ("") se utiliza como separador, la cadena se divide entre cada personaje.

Nota: El método split () no modifica la cadena original.

String.fromCharCode(char...)

El método fromCharCode () convierte valores Unicode en caracteres.

Nota: Este es un método estático del objeto String, y la sintaxis es siempre String.fromCharCode ().

Ejemplo

Convertir un número Unicode en un personaje:

```
var res = String.fromCharCode(65);
```

El resultado de *res* será:

A

String

string.charAt(pos):

El método charAt () devuelve el carácter en el índice especificado en una cadena.

El índice del primer carácter es 0, el segundo carácter es 1, y así sucesivamente

.

Ejemplo:

Devuelva el primer carácter de una cadena:

```
var str = "HELLO WORLD";
```

```
var res = str.charAt(0)
```

El resultado de *res* será:

H

JSON

JSON tiene seis tipos de valores : objetos, matrices , cadenas , números, booleanos (verdadero y falso), y el nulo valor especial . Espacios en blanco (espacios , tabuladores , retornos de carro y caracteres de nueva línea) pueden insertarse antes o después de cualquier valor . Esto puede hacer que los textos JSON más fácil para las personas a leer. El espacio en blanco se puede omitir para reducir los costos de transmisión o de almacenamiento.

Un objeto JSON es un contenedor desordenado de pares nombre / valor. Un nombre puede ser cualquier cadena . Un valor puede ser cualquier valor JSON , incluyendo arrays y objetos.

Objetos JSON se pueden anidar a cualquier profundidad , pero en general es más eficaz para mantenerlos relativamente plana. La mayoría de los idiomas tienen una característica que los mapas fácilmente a los objetos JSON , como un objeto, estructura , ficha , diccionario, tabla hash , lista de propiedades, o matriz asociativa

Una cadena JSON se envuelve entre comillas dobles. El carácter \ se utiliza para el escape . JSON permite al / carácter que se escapó de manera que JSON puede ser embebido en HTML etiquetas < script> . HTML no permite la secuencia < / excepto para iniciar el < / script > etiqueta . JSON permite < \ / , que produce el mismo resultado pero no confunde HTML .

Números JSON son como los números de JavaScript . No se permite que un cero a la izquierda en gers inte- porque algunos idiomas usan para indicar que el octal. Ese tipo de confusión radix no es deseable en un formato de intercambio de datos. Un número puede ser un número entero , real, o científico .

Example Json

```
[  
  {  
    "first": "Jerome",  
    "middle": "Lester",  
    "last": "Howard",  
    "nick-name": "Curly",  
    "born": 1903,  
    "died": 1952,  
    "quote": "nyuk-nyuk-nyuk!"  
  }  
]
```

A JSON Parser

Esta es una implementación de un analizador JSON en JavaScript :

```
var json_parse = function () {  
  // Esta es una función que puede analizar un texto JSON , produciendo un JavaScript  
  // Estructura de datos. Es un sencillo programa de análisis , descenso recursivo .  
  // Estamos definiendo la función dentro de otra función para evitar la creación  
  // Variables globales .
```

```
  escapee = {  
    '"': '"',  
    '\\': '\\',  
    '/': '/',  
    b: 'b',  
    f: 'f',  
    n: 'n',  
    r: 'r',  
    t: 't'  
  },
```

Block-less Statements

Un caso o mientras , hacer o para la declaración puede tener un bloque o una sola sentencia. El formulario de declaración solteras es otra molestia atractiva . Ofrece la ventaja de ahorrar dos personajes, una ventaja dudosa . Se oscurece la estructura del programa para que los manipuladores subsecuentes del código pueden insertar fácilmente los insectos. Por ejemplo:

```
if (ok)
    t = true;
```

can become:

```
if (ok)
    t = true;
advance( );
```

que se parece a :

```
if (ok) {
    t = true;
advance( ); }
```

but which actually means:

```
if (ok) {
    t = true;

}
advance( );
```

Bitwise Operators

JavaScript tiene el mismo conjunto de operadores de bits como Java :

`y y`

`| o`

`^ xor`

`~ no`

`>>` Desplazamiento a la derecha firmado

`>>>` Cambio sin firmar derecha

`<<` Desplazamiento a la izquierda

En Java , los operadores de bits trabajan con números enteros . JavaScript no tiene números enteros . Sólo tiene los números de coma flotante de doble precisión. Por lo tanto, los operadores de bits convierten sus operandos numéricos en enteros , hacen sus negocios , y luego convertirlos de nuevo . En la mayoría de idiomas , estos operadores están muy cerca del hardware y muy rápido. En JavaScript , están muy lejos de ser el hardware y muy lento. JavaScript rara vez se utiliza para hacer la manipulación de bits .

Como resultado, en los programas de JavaScript , es más probable que Y es un operador && mal escrita . La presencia de los operadores de bits reduce algunos de redundancia de la lengua , por lo que es más fácil para los insectos que se esconden .

La Declaración función Versus la expresión de función

La **Declaración función** es una declaración de función, tal como una expresión de función. Este es la abreviatura `function foo() {}`.

significa aproximadamente lo mismo que :

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


Nuevo operador de JavaScript crea un nuevo objeto que hereda de miembro prototipo del operando, y luego llama el operando, la unión del nuevo objeto a esto. Esto da el operando (que mejor había ser una función constructor) la oportunidad de personalizar el nuevo objeto antes de que se devuelve al solicitante.

Si olvidó usar el nuevo operador, en su lugar recibe una llamada función ordinaria, y esto está ligado al objeto global en lugar de a un nuevo objeto. Eso significa que su función será clobbering variables globales cuando se intenta inicializar los nuevos miembros. Eso es una cosa muy mala. No hay ninguna advertencia de tiempo de compilación. No hay ninguna advertencia de tiempo de ejecución.

Por convención, las funciones que están pensados para ser utilizados con el nuevo debe dar nombres con letra inicial mayúscula, y los nombres con letra inicial mayúscula debe ser utilizado sólo con funciones constructoras que tienen el nuevo prefijo. Este convenio nos da una señal visual que puede ayudar a detectar errores costosos que el lenguaje mismo está dispuesto a pasar por alto.