Convenciones de codificación para el lenguaje de programación JavaScript

Traducción al español de Code Conventions for the JavaScript Programming Language

Douglas Crockford

Este es un conjunto de convenciones y reglas de codificación para ser usadas en la programación en JavaScript. Esta inspirada por el documento de Sun Convenciones de codificación para el lenguaje de programación Java. Por supuesto ha sido substancialmente modificada ya que JavaScript no es Java.

El valor a largo plazo del software para una organización es directamente proporcional a la calidad de su código fuente. Durante su vida útil un programa sera manipulado por muchos pares de manos y ojos. Si un programa es capaz de comunicar claramente su estructura y características es menos probable que deje de funcionar cuando sea modificado en el, nunca muy distante, futuro.

Las convenciones de codificación pueden ayudar a reducir la fragilidad de los programas.

Todo nuestro código en JavaScript es enviado directamente al publico. Debe ser siempre de una calidad digna de publicarse.

La limpieza cuenta.

Archivos JavaScript

Los programas en JavaScript deben ser almacenados y enviados como archivos .js.

El código en JavaScript no debe ser insertado en los archivos HTML a menos que el código sea especifico a una sesión individual. El código en HTML añade significativamente al tamaño de la pagina sin ofrecer oportunidades de ser mitigado mediante cache y compresión.

Las etiquetas <script src=archivo.js> deben ser colocadas lo mas tarde posible en el cuerpo del documento. Esto reduce los efectos de las demoras causadas por la carga del programa en otros componentes de la pagina. No hay necesidad de usar los atributos language ni type. Es el servidor, no la etiqueta script, quien determina el tipo MIME.

Indentacion

La indentación se hace a cuatro espacios. El uso del carácter de tabulación debe ser evitado porque (al momento de escribirse esto en pleno siglo 21) aun no hay un estándar para la ubicación de las paradas de tabulación. El uso de espacios puede producir un archivo de mayor tamaño pero el tamaño es irrelevante en el caso de las redes locales y en todo caso la diferencia es eliminada mediante la minificacion.

Longitud de la linea

Evite lineas de más de 80 caracteres. Cuando una instrucción no quepa en una sola linea puede ser necesario dividirla. Ponga tal división inmediatamente después de un operador, idealmente inmediatamente después de una coma. Dividir la linea después de un operador reduce la probabilidad de que un error al copiar y pegar sea encubierto por la inserción de un punto y coma. La linea siguiente debe ser indentada 8 espacios.

Comentarios

Sea generoso con los comentarios. Es útil dejar información que sera leída un tiempo después por las personas (posiblemente usted mismo) que necesitaran entender lo que usted ha hecho. Los comentarios deben estar bien escritos y ser claros, tal como el código que están documentando. El ocasional apunte humorístico sera apreciado, expresar frustraciones y resentimientos en ellos no.

Es importante que los comentarios se mantengan actualizados. Comentarios erróneos pueden hacer los programas aún mas difíciles de leer y entender.

Haga comentarios significativos. Concéntrese en lo que no es visible de inmediato. No haga perder tiempo al lector con cosas como

```
i = 0; // Poner i en cero
```

En general use comentarios en la misma linea. Reserve el uso de un bloque de comentarios para la documentación formal y para comentar código que no se desea ejecutar.

Declaración de variables

Todas las variables deben ser declaradas antes de ser usadas. JavaScript no lo requiere, pero hacerlo hace los programas mas fáciles de leer y hace mas fácil detectar variables que no han sido declaradas que pueden convertirse en variables globales implícitas. Las variables globales implícitas nunca deben ser usadas.

Las instrucciones var deben ser las primeras instrucciones en el cuerpo de la función.

Es preferible que a cada variable se le dé su propia linea y su propio comentario. Deben ser listadas en orden alfabético:

```
var currentEntry; // entrada actualmente seleccionada en la tabla
var level;        // nivel de indentacion
var size;         // tamaño de la tabla
```

El alcance de todas las variables en JavaScript es la función donde están definidas, JavaScript no tiene alcance de variables a nivel de bloque, así que definir variables en bloques puede confundir a programadores con experiencia en otros lenguajes de la familia de C. Defina todas las variables al inicio de la función.

El uso de variables globales debe ser minimizado. Las variables globales implícitas nunca deben ser usadas.

Declaración de funciones

Todas las funciones deben ser declaradas antes de ser usadas. Las funciones internas deben aparecer después de las instrucciones var. Esto ayuda a dejar en claro cuales variables están incluidas en su contexto.

No debe haber espacio entre el nombre de una función y el ((paréntesis izquierdo) de su lista de parámetros. Solo debe haber un espacio entre el) (paréntesis derecho) y el { (corchete izquierdo) que inicia el cuerpo de la instrucción. El cuerpo mismo debe ser indentado cuatro espacios. El } (corchete derecho) se alinea con la linea que contiene el inicio de la declaración de la función.

```
function outer(c, d) {
    var e = c * d;

    function inner(a, b) {
        return (e * a) + b;
    }

    return inner(0, 1);
}
```

Esta convención funciona bien con JavaScript porque en JavaScript las funciones y los objetos pueden ser colocados en cualquier parte que se permita una expresión. Provee la mayor legibilidad con funciones en linea y estructuras complejas.

```
function getElementsByClassName(className) {
    var results = [];
    walkTheDOM(document.body, function (node) {
        var a;                  // array de nombres de clases
        var c = node.className; // el nombre de clase del nodo
        var i;                  // contador para el bucle

// Si el nodo tiene un nombre de clase se divide en una lista de nombres simples
// Si cualquiera de ellos coincide con el nombre pedido entonces añadir el nodo
// al conjunto de resultados

        if (c) {
            a = c.split(' ');
            for (i = 0; i < a.length; i += 1) {
                if (a[i] === className) {
                    results.push(node);
                    break;
                }
            }
        }
```

```
    });
    return results;
}
```

Si una función es anónima entonces debe haber un espacio entre la palabra function y el ((paréntesis izquierdo). Si se omite el espacio entonces puede parecer que el nombre de la función es function lo cual es una lectura incorrecta del código.

```
div.onclick = function (e) {
    return false;
};

that = {
    method: function () {
        return this.datum;
    },
    datum: 0
};
```

El uso de funciones globales debe ser minimizado.

Cuando una función vaya a ser invocada inmediatamente, la llamada de invocación debe ser rodeada por completo de paréntesis de modo que sea claro que el valor producido es el resultado de la función y no la función en si misma.

```
var collection = (function () {
    var keys = [], values = [];

    return {
        get: function (key) {
            var at = keys.indexOf(key);
            if (at >= 0) {
                return values[at];
            }
        },
        set: function  (key, value) {
            var at = keys.indexOf(key);
            if (at < 0) {
                 at = keys.length;
             }
             keys[at] = key;
             values[at] = value;
         },
         remove: function (key) {
             var at = keys.indexOf(key);
             if (at >= 0) {
                keys.splice(at, 1);
                values.splice(at, 1);
            }
```

```
        }
    };
}());
```

Nombres

Los nombres deben ser formados con las 26 letras mayúsculas y minúsculas del alfabeto ingles (A .. Z, a .. z), los 10 dígitos (0 .. 9), y la _ (barra inferior). Evite el uso de caracteres en otros idiomas porque puede que no sean leídos correctamente o que no se los entienda. No emplee el signo $ (dolar) ni la \ (barra invertida) en los nombres.

No use la _ (barra inferior) como el primer caracter de un nombre. Se le usa algunas veces para indicar privacidad, pero en si no provee privacidad alguna. Si la privacidad es importante usa las formas que proveen miembros privados. Evite adoptar convenciones que demuestran incompetencia.

La mayoría de las variables y funciones deben empezar con una letra minuscula.

Los constructores que deban ser usados con el prefijo new deben empezar con una letra mayúscula: JavaScript no emite una advertencia en la etapa de compilación ni en la etapa de ejecución si un new es omitido. Cosas malas pueden ocurrir si new no es empleado, así que la convención de empezar el nombre con mayúscula es la única defensa que tenemos.

Los nombres de las variables globales deben ir todo en mayúsculas (JavaScript no tiene macros ni constantes así que no hay razón para usar palabras todas en mayúsculas para denotar cosas que JavaScript no tiene)

Instrucciones

Instrucciones simples

Cada linea debe contener a lo sumo una instrucción. Ponga un ; (punto y coma) al final de cada instrucción simple. Note que una asignación donde se asigne una función literal o un objeto literal sigue siendo una asignación y debe terminar con un punto y coma.

JavaScript permite que cualquier expresión sea usada como una instrucción. Esto puede encubrir algunos errores, especialmente en la presencia de una inserción de punto y coma. Las únicas expresiones que deben usarse como instrucciones son las asignaciones y las invocaciones.

Instrucciones compuestas

Las instrucciones compuestas son instrucciones que contienen listas de instrucciones rodeadas de { } (corchetes).

- Las instrucciones rodeadas de corchetes deben estar indentadas cuatro espacios mas

- El { (corchete izquierdo) debe estar al final de la linea que inicia la instrucción compuesta

- El } (corchete derecho) debe iniciar una linea y estar indentado de modo que se alinee con el inicio de la linea que contiene el correspondiente { (corchete izquierdo).

- Los corchetes deben ser usados alrededor de todas las instrucciones, incluso de las simples, cuando sean parte de una estructura de control tales como un if o un for. Esto facilita añadir instrucciones sin accidentalmente introducir errores de programación.

Etiquetas

Las etiquetas son opcionales. Solo estas instrucciones deben ser etiquetadas: while, do, for, switch

Instrucción return

Una instrucción return con un valor no debe usar ( ) (paréntesis) alrededor del valor. La expresión de retorno de valor debe empezar en la misma linea que la palabra clave return para prevenir inserción de punto y coma.

Instrucción if

La clase de instrucciones if debe tener la siguiente forma:

```
if (condición) {
    instrucciones
}

if (condición) {
    instrucciones
} else {
    instrucciones
}

if (condición) {
    instrucciones
} else if (condición) {
    instrucciones
} else {
    instrucciones
}
```

Instrucción for

La clase de instrucciones for debe tener la siguiente forma:

```
for (inicialización; condición; cambio de valor) {
    instrucciones
```

```
}

for (variable in objeto) {
    if (filtro) {
        instrucciones
    }
}
```

La primera forma debe ser usada con arrays y con bucles con un numero predeterminado de iteraciones.

La segunda forma debe ser usada con objetos. Note que miembros que sean añadidos al prototipo del objetoserán incluidos en la enumeración. Es astuto el programar defensivamente usando el métodohasOwnProperty para distinguir los miembros verdaderos del objeto:

```
for (variable in objeto) {
    if (objeto.hasOwnProperty(variable)) {
        instrucciones
    }
}
```

Instrucción while

La instrucción while debe tener la siguiente forma:

```
while (condición) {
    instrucciones
}
```

Instrucción do

La instrucción do debe tener la siguiente forma:

```
do {
    instruccciones
} while (condición);
```

A diferencia de las otras instrucciones compuestas la instrucción do siempre termina con un ; (punto y coma).

Instrucción switch

La instrucción switch debe tener la siguiente forma:

```
switch (expresión) {
case expresión:
    instrucciones
default:
    instrucciones
```

```
}
```

Cada case esta alineado con el switch. Esto previene la sobreindentacion.

Cada grupo de instrucciones (excepto default) debe terminar con un break, return o throw. No caiga de un caso a otro.

Instrucción try

La clase de instrucciones try debe tener la siguiente forma:

```
try {
    instrucciones
} catch (variable) {
    instrucciones
}

try {
    instrucciones
} catch (variable) {
    instrucciones
} finally {
    instrucciones
}
```

Instrucción continue

Evite el uso de la instrucción continue. Tiende a hacer confuso el control de flujo de la función.

Instrucción with

La instrucción with No debe ser usada

Espacio en blanco

Las lineas en blanco mejoran la legibilidad al agrupar secciones de código que están relacionadas lógicamente.

Espacio en blanco debe ser usado en las siguientes circunstancias:

- Una palabra clave seguida de un ((paréntesis izquierdo) debe estar separada por un espacio.

```
while (true) {
```

- Un espacio en blanco no debe ser usado entre un valor de una función y su ((paréntesis izquierdo). Esto ayuda a distinguir entre palabras clave e invocaciones de funciones.

- Todos los operadores binarios excepto . (punto) y ((paréntesis izquierdo) y [(corchete cuadrado izquierdo) deben estar separados de sus operandos por un espacio.

- Ningún espacio debe separar un operador unitario y su operando excepto cuando el operador es una palabra como typeof.

- Cada ; (punto y coma) en la parte de control de una instrucción for debe estar seguido por un espacio.

- Cada , (coma) debe estar seguida de espacio en blanco

Sugerencias adicionales

{} y []

Use {} en vez de new Object(). Use [] en vez de new Array().

Use arrays cuando los nombres de los miembros sean enteros secuenciales. Use objetos cuando los nombres de los miembros sean nombres o cadenas arbitrarias.

Operador , (coma)

Evite el uso del operador coma excepto por un uso muy disciplinado en la parte de control de las instruccionesfor. (Esto no se aplica al separador coma que se usa en objetos literales, array literales, instrucciones var y listas de parámetros.)

Alcance a nivel de bloque

En JavaScript los bloques no tienen alcance. Solo las funciones tienen alcance. No use bloques excepto al ser requeridos por las instrucciones compuestas.

Expresiones de asignación

Evite hacer asignaciones en la parte de la condición de instrucciones if y while.

¿Es

```
if (a = b) {
```

Una instruccion correcta? O ¿era

```
if (a == b) {
```

lo deseado? Evite construcciones que no puedan fácilmente determinarse si son correctas.

Operadores === y !===

Casi siempre es mejor usar los operadores === y !===. Los operadores == y !== hacen coerción de tipo. En particular no use == para comparar contra valores falsos.

Confundiendo mases y menos

Tenga cuidado de no seguir un + con + o ++. Este patrón puede ser confuso. Inserte paréntesis entre ellos para hacer claras sus intenciones.

```
total = subtotal + +myInput.value;
```

es mejor escrito como

```
total = subtotal + (+myInput.value);
```

de modo que el + + no sea erróneamente leído como ++.

eval *es perverso*

La función eval es la más mal usada de JavaScript. Evítela.

eval tiene aliases. No use el constructor Function. No pase cadenas de caracteres a setTimeout o setInterval

Aquí tienes otra página donde puedes ver las convenciones para el lenguaje Vbscript que también pueden servir para complementas las ya descritas en este capítulo.

http://practicasdeprogramacion.wordpress.com/2011/07/30/convenciones-de-codificacion-para-javascript/

http://javascript.crockford.com/code.html

Herramientas para depurar javascript

Opera ---- Herramientas → Avanzado → OperaDragonfly

IE----- Herramientas → Herramientas de desarrollo (F12)

Safari ---- Instalar complemento “Firebug” ----Se activa con F12

Firefox ---- Instalar complemento “Firebug” ---- Se activa con F12. Si la pestaña de script da error, pulsar sobre la página web botón derecho, visualizar código fuente. Con esto debería de bastar.

Chrome -----Botón superior derecha → Herramientas → Herramientas para desarrolladores

Otro sistema para controlar los errores es el sistema de banderas. Es decir, en el punto que queramos saber el valor de una variable o saber si pasa por un lugar determinado, colocar una instrucción que muestre información por pantalla.

http://www.genbetadev.com/javascript/depurando-de-forma-avanzada-javascript-con-las-herramientas-de-desarrollo

Enlaces

MSD Mozilla Developer Network

https://developer.mozilla.org/es/docs/Web/JavaScript

Librosweb

http://librosweb.es/javascript/

W3schools

http://www.w3schools.com/js/