INVESTIGACIÓN

ESTANDARES DE JS EN SDLC

Luis Alejandro Gomez Santillán
UTXJ | 190292

Estándares JavaScript

1 Por qué necesitamos estándares JavaScript

La escritura suelta de JavaScript y la herencia prototípica lo hacen diferente de los lenguajes clásicos, como Java. Esto puede frustrar a los nuevos desarrolladores de JavaScript que intentan forzar los patrones clásicos sobre los patrones de JavaScript adecuados. (Crockford, 2008)

2 Normas

2.1 Archivos

Los archivos JavaScript deben tener la extensión de archivo .js. Si bien los navegadores no requieren esta extensión, ayuda a otros desarrolladores a comprender cómo encajan los componentes de una aplicación.
Los desarrolladores pueden "minimizar" o comprimir archivos JavaScript por razones de rendimiento, pero no deben emplear técnicas de ofuscación adicionales a menos que lo requiera el proyecto. Como regla general, la información confidencial no debe aparecer en los archivos JavaScript porque son documentos de texto procesados en el ordenador del cliente y, por lo tanto, inseguros.
Si bien no es obligatorio, la organización de archivos JavaScript en un directorio separado también ayuda a otros desarrolladores a encontrar componentes de la aplicación rápidamente.

2.2 Archivos externos vs. código en línea

Siempre que sea posible, mueva grandes cantidades de JavaScript en línea a archivos externos para que sea mantenible y para aprovechar el almacenamiento en caché del navegador. Coloque todas las referencias a archivos externos, así como al código en línea, en la sección <HEAD> del documento.<script src="webjslint.js"></script>

Mantenga JavaScript discreto agregando llamadas a funciones previamente definidas utilizando métodos DOM. Con la mayor frecuencia posible, utilice métodos avanzados de registro de eventos a través de:

```
 window.onload.function doSomething() {
}

function doSomethingElse() {
}
```

window.onload = doSomething;
// window.onload = doSomethingElse; esto sobrescribiría la llamada a doSomething

en su lugar, utilice
window.onload = function () {
doSomething();
doSomethingElse();
}

// o crear una función reutilizable // la mayoría de las bibliotecas incluyen un controlador de eventos

function addEvent(obj, evType, fn) {
if (obj.addEventListener)

Macro desconocida: { obj.addEventListener(evType, fn, false); return true; }

else if (Obj.attachEvent)

Macro desconocida: { // used in IE return obj.attachEvent("on" + evType, fn); }

más

Macro desconocida:

{ // IE para Mac no funcionará retorno false; }

}
addEvent(window, 'load', doSomething);
addEvent(window, 'load', doSomethingElse);

2.3 Código dinámico vs. estático

Evite usar otro lenguaje de programación para escribir código JavaScript en línea. Las condiciones no probadas en la aplicación pueden producir JavaScript fallido o no válido. La eliminación de esta práctica aumenta el potencial de reutilización de código y reduce el tiempo dedicado a probar y solucionar problemas de JavaScript.

3 Sintaxis

Los desarrolladores y aprobadores de código pueden usar JSLint con la siguiente configuración para ayudar en el proceso de revisión de código:

- Espacio en blanco estricto (4 espacios)
- Permitir una instrucción var por función
- No permitir variables indefinidas
- No permitir colgar _ en identificadores
- No permitir == y !=
- No permitir operadores bitwise
- No permitir inseguro . y [^...] en /RegExp/
- Requerir "uso estricto"; (utilizado en la revisión completa del archivo)
- Requerir límites iniciales para los constructores
- Requerir paréntesis alrededor de invocaciones inmediatas

O use la configuración rápida con la siguiente cadena:

jslint white: true, onevar: true, undef: true, nomen: true, eqeqeq: true, bitwise: true, regexp: true, strict: true, newcap: true, immed: true

3.1 Espacio en blanco y punto y coma

Si bien a menudo podemos eliminar los espacios en blanco opcionales y los puntos y comas de fin de línea de JavaScript para reducir el archivo, los desarrolladores deben preservar el espacio en blanco y la puntuación adecuada en todo el código fuente para facilitar la lectura. Utilice únicamente un método sistemático para eliminar espacios en blanco en archivos de producción, como:

- Dojo Shrinksafe
- JSMin
- Embalador
- Compresor YUI

3.2 Comentarios

JavaScript utiliza la misma sintaxis de comentario que Java. Sin embargo, dado que las secuencias de caracteres de comentario de bloque

(/*/) pueden aparecer en expresiones regulares, no las utilice para el código en línea o fuera de las secciones de documentación formal. En su lugar, use el comentario de una sola línea (//)./

no utilice en línea

*/

ok para usar en cualquier lugar

Los bloques de comentarios (con el comienzo adicional *) deben aparecer en archivos JavaScript externos como documentación:/**

- Devuelve un guid asociado a un objeto. Si el objeto
- no tiene uno, se crea uno nuevo a menos que readOnly
- se especifica. *
- @method sello
- @param o El objeto a estampar
- @param leerSólo

  Macro desconocida: {boolean}

  si es true, solo se devolverá un guid válido si el objeto tiene uno asignado.

- @return

  Macro desconocida: {string}

  Guid o null del objeto
  */

3.3 Bloques de código

Encierre bloques de código entre { } caracteres con la llave inicial al final de la primera línea y la llave final por sí misma después de la última línea. Aplicar sangría a todas las instrucciones dentro del bloque de código." uso estricto";

```
function codeBlock() {
alert('dentro de un bloque de código');
}
```

3.4 Longitud de línea

Evite las líneas de más de 80 caracteres. Divida largas líneas de código después de un operador para evitar errores de copiar/pegar o errores de:

```
análisis.function longLine() {
var str = "Esta cadena tendrá que continuar " +
"en la siguiente línea.",
obj = {
```

```
propiedad: "one",
innerObj:

Macro desconocida: { propiedad}

},
arr = ["uno", "dos", "tres",
"cuatro", "cinco"];
}
```

4 Patrones

4.1 Referencias de objetos y desreferenciación

JavaScript nunca copia objetos, solo crea múltiples referencias. Como método para evitar errores y liberar memoria del entorno, elimine los objetos estableciendo todas las referencias en null después de que ya no las necesite.

```
var o Object = new Object;
//hacer algo con el objeto aquí
oObject = null;
```

4.2 Nombres

Los nombres se utilizan para instrucciones, variables, parámetros, nombres de propiedades, operadores y etiquetas. No utilice las siguientes palabras para los nombres: **abstract, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double, else, enum, export, extends, false, finally, float, for, function, goto, if, implements, import, in, Infinity, instanceof, int, interface, long, NaN, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, undefined, var, volatile, void, while, with**. Otros identificadores a evitar: **arguments, encodeURI, Object, String, Array, Error, isFinite, parseFloat, SyntaxError, Boolean, escape, isNaN, parseInt, TypeError, Date, eval, Math, RangeError, decodeURI, EvalError, ReferenceError, unescape, decodeURIComponent, Function, Number, RegExp, URIError**
Las variables deben consistir en un carácter de letra seguido de cualquier combinación de letras, números o guion bajo. Cuando sea posible, use la notación de camello (camel-case) para separar las palabras:

```
var a_variable_name = "bad";
var _variable_name = "bad";
var aVariableName = "good";
```

Los constructores (funciones que devuelven un objeto y hacen uso del nuevo operador) deben comenzar con una letra mayúscula.

4.3 Alcance

Dado que JavaScript utiliza sintaxis de bloque pero no proporciona ámbito de bloque, los desarrolladores deben tener especial cuidado para administrar el ámbito. Por esta razón, use solo una instrucción var en la parte superior de cada función.
Las aplicaciones web deben minimizar el uso del espacio de nombres global. Los desarrolladores deben asignar funciones y variables específicas de la aplicación a un espacio de nombres de aplicación. Esto ayuda con las pruebas de código y la solución de problemas al mantener el código modular y reutilizable. Esta práctica también reduce las posibilidades de un conflicto de espacio de nombres cuando se utiliza una biblioteca o marco externo". uso estricto";

```
var MyApp = || indefinido MyApp; definir variable a menos que exista

if (typeof MyApp === 'undefined' || ! MyApp) {// create if needed
(function () {
var privateVar = "private", // private variable
privateFunc = function ()
```

Macro desconocida:

```
 { // private method alert("hide me"); }

soonToBePublic = función ()

Macro desconocida: { alert(privateVar); };

MyApp = {
init: función ()

Macro desconocida: { // método público }
```

configuración:

```
Macro desconocida: { option1}


setup: function () {
}
```

```
getPrivate : soonToBePublic // private method now public
};
}());

}
```

4.4 Interacción de formularios

Debido a un error en versiones anteriores de IE, como práctica recomendada, use referencia al formulario y sus elementos en lugar de un elemento por ID.// use este

```
var myInput = document.forms"nombre del formulario".elements"inputname";
```

en lugar de este

```
var myInput = document.getElementById("inputid");
```

5 Herramientas

5.1 Validación de código

JSlint puede ayudar a los desarrolladores identificando rápidamente errores y problemas potenciales con la versión en línea o fuera de línea.

5.2 Navegadores

Junto con lo anteriormente mencionado JSlint , la mayoría de los navegadores modernos tienen depuradores de scripts y consolas. Otros navegadores hacen uso de complementos para realizar esta tarea, como el muy popular Firebug complemento para Firefox. Consulte la documentación de su navegador para habilitar las funciones del modo de desarrollador.

Los navegadores modernos generan más excepciones y evitan algunas acciones "inseguras" de JavaScript cuando los desarrolladores implementan la función "usar estricto". (Resig, 2009)

Al desarrollar para varios navegadores, compruebe la disponibilidad del método y las posibles dificultades de usar ciertos métodos mediante el uso del Tabla de compatibilidad maestra.

5.3 Entorno de desarrollo interactivo (IDE)

El resaltado de sintaxis, el espaciado y el formato adecuados y el colapso de bloques de código pueden ayudar a acelerar el proceso de desarrollo. Los desarrolladores deben usar un IDE para el desarrollo de JavaScript como cualquier otro lenguaje de programación.

5.4 Bibliotecas

Las bibliotecas de JavaScript ofrecen herramientas valiosas para reducir el tiempo de desarrollo y aumentar el rendimiento, si se usan correctamente. Utilice estas directrices para implementar una biblioteca de JavaScript:

- Utilice únicamente una biblioteca con el código fuente completo (no minificado ni comprimido) disponible
- Utilice únicamente una biblioteca que se mantenga activamente
- Utilice la versión más reciente de la biblioteca y aplique parches de seguridad y errores cuando se publiquen
- Garantizar la compatibilidad con todos los navegadores necesarios para el proyecto
- No bifurcar ni modificar los archivos de biblioteca
- No invalidar los métodos de biblioteca
- Evite los nombres abreviados, como $, ya que esto puede provocar conflictos de espacio de nombres y causar confusión

6 Anti-patrones (Prácticas innecesarias)

6.1 myElement.style.width = "20px"

Evite establecer atributos de estilo CSS con JavaScript. En su lugar, cree las clases CSS adecuadas y aplique o elimine esas clases.

6.2 documento.escribir

Este método se deprecia. Los desarrolladores deben usar métodos DOM para cambiar el DOM.

6.3 <noscript></noscript>

Un mejor enfoque implica incluir un mensaje 'Sin JavaScript' en la aplicación que JavaScript elimina. Si el navegador del usuario no tiene JavaScript o JavaScript está deshabilitado, aparecerá el mensaje.

6.4 href = "javascript:", onclick = "javascript:"

Los desarrolladores deben usar JavaScript discreto. Agregue las mejoras de JavaScript necesarias a HTML válido a través de métodos DOM de JavaScript, no el HTML.

6.5 switch (sin una instrucción predeterminada:

Utilice el segmento predeterminado de una instrucción switch para advertir de que un caso "falló" en la instrucción switch sin cumplir una de las condiciones. De lo contrario, esta fuente común de errores dará lugar a dificultades durante la solución de problemas.

6.6 onclick = "void(0)"

Suprima las acciones predeterminadas con patrones discretos (véase más arriba). Además, evite el operador void, ya que siempre devuelve undefinido, que no tiene ningún valor.

6.7 var myObject = Nuevo objeto();

Utilice la notación literal del objeto: var myObject = {};

6.8 var myArray = Nueva matriz();

Utilice la notación literal de matriz: var myArray = [];

6.9 document.all, document.layers, navigator.useragent

El rastreo del navegador causa más problemas de los que soluciona. En su lugar, detecte métodos DOM específicos.

```
// función correcta
addEvent(obj, evType, fn) {
if (obj.addEventListener)
Macro desconocida: { // comprueba la existencia de un método obj.addEventListener(evType, fn, false); return true; }
}

función incorrecta
addEvent(obj, evType, fn) {
if (obj.addEventListener())
Macro desconocida: { // asume que el método existe y comprueba el valor obj.addEventListener(evType, fn, false); return true; }
}
```

6.10 with

Evite usar la instrucción with, tiende a oscurecer la verdadera intención de un bloque de código.

6.11 continue y brake

Evite usar las instrucciones continue y break, puede oscurecer la lógica de bucle prevista.

6.12 _myPrivateVariable

Los intentos de indicar variables privadas con un guión bajo inicial (o un doble guión bajo inicial y final) pueden hacer que los desarrolladores sean complacientes o confundir la verdadera naturaleza de una variable. Utilice las convenciones de nomenclatura descritas anteriormente y utilice un depurador para verificar la privacidad de la propiedad.

6.13 eval

Los desarrolladores a menudo usan eval en lugar de la notación adecuada de objetos y subíndices.

```
// use this
myvalue = myObjectmyKey;
en lugar de este
eval("myValue = myObject." + myKey + ";");
```

El código malicioso puede explotar eval. Los desarrolladores deben evitar eval y setTimeout (que puede actuar como eval) por esta razón.

6.14 == y != vs. === y !==

Cuando se utilizan operadores de igualdad, elija === o !== sobre sus contrapartes, ya que también comparan el tipo y ayudan a preservar la transitividad en las variables. (Crockford, 2008)

```
'' == '0'; falso
0 == ''; verdadero
0 == '0'; true
false = 'falso'; falso
falso == '0'; true
false == indefinido; false
false == null; false
null == indefinido; verdadero
'\t\r\n ' == 0; verdadero
```

6.15 Nuevo Booleano()

Evite la clase booleana, en su lugar use primitivas booleanas, true y false.

6.16 Declaración try-catch

Los desarrolladores que intentan aplicar su conocimiento de los lenguajes clásicos a menudo hacen un mal uso del patrón de prueba y captura y, en última instancia, suprimen la información de error importante. Por esta razón, evite las declaraciones de prueba. Además, pruebe (la parte de captura) y con las instrucciones agregue un objeto al frente de la cadena de alcance, lo que los hace menos deseables por razones de rendimiento. (Zakas N., 2009)