Resumen de Javascript

1. Buenas Pácticas

1.1. Evitar variable Globales
1.2. Declarar variables locales
1.3. Declarar variables sobre la función
1.4. No declarar Boolean, number ni strings como objects
1.5. No usar new Object(), String(), etc.
1.6. Tener cuidado con la conversion automática de tipos
1.7. Usar === y no ==
1.8. Utilizar todos los parámetros declarados
1.9. No usar eval()

2. Javascript

JavaScript es un lenguaje de scripting multiplataforma, orientado a objetos. Está diseñado para funcionar en diferentes aplicaciones y productos, con ayuda de varios objetos predefinidos, pueden tomar control del navegador, o bien comunicarlo con el servidor.

JavaScript es un lenguaje simple y sin copilador, por lo que es propenso a cometerse errores pero permite una liberación de programación. Lo negaivo es que utiliza variables globales pero JabaScript puede mitigar eso.

2.1. Javascript y diferencia con Java

Javascript y Java son similares pero a su vez son diferentes, Javascript tiene sintaxis, convenciones de nombres y controles básicos de flujo parecidos a los de Java, por eso es que se le cambio el nombre de Livescript a javascript. Utiliza un sistema de tiempo de Ejecución representando valores numéricos, booleanos, y cadenas, utiliza herencia dinámica, y se orienta a prototipos, además no necesita que una función se predefinida.

La diferencia más notable es la libertad de programación que permite Javascript y su facilidad de ejecución ante Java.

2.2. ECMA y Javascript

La especificación ECMA es importante para poder escribir código con las funciones que pueden ser soportada, sin embargo no está destinada para ayudar a los programadores de scripts. Este determinará el estándar a utilizar.

Javascript lo creó Nescape, pero fue trabajado junto con ECMA (European Computer Manufacturers Association) al principio Javascript no era totalmente

soportado por esta especificación e incluso fue lanzada la versión 1.2 cuando ECMAscript (versión de ECMA) no estaba completa, de a partir de la versión 1.3 de Javascript es totalmente compatible con ECMA.

La terminología que usa la especificación ECMA es desconocido para un programador de Javascript, sin embargo, el leguaje es el mismo que en dicha especificación, y es totalmente soportado.

3. Objetos

Entidades que tienen atributos, métodos y un identificador. Los objetos deben tener valores que los identifiquen de otros objetos e incluso de la misma clase, los atributos deben estar relacionados con los métodos ya que si no hay esta relación, sería programación estructurada camuflada como de objetos

3.1. Origen

Nace de un lenguaje de simulación llamado Simula 67, en los que se estudiaba el comportamiento de algunas naves con sus propios atributos y como podían afectar a las demás naves.

En los 80's fue dominante más por la influencia de C++, y al auge de las interfaces gráficas, pero carecían de estadísticas, hasta que surgieron lenguajes que se orientaban a objetos, hasta llegar al lenguaje de Bertrand Meyer luego remplazándolo Java gracias al auge del internet.

3.2. Conceptos

- **Clase:** Define las propiedades y comportamientos de un objeto de un tipo en concreto.
- **Herencia:** La Clase D puede heredar del padre Clase C atributos y operaciones como si se definieran en la Clase C.
- **Objeto:** Conjunto de propiedades, atributos y métodos, que reaccionan a eventos, es la instancia de una clase.
- **Método:** Un algoritmo que se ejecuta para recibir un "mensaje".
- **Evento:** Es cuando un mensaje es producido por el método, es el suceso del sistema.
- **Atributos:** Características de la clase.
- **Mensaje:** La comunicación dirigida a un objeto.
- **Propiedad:** Contenedor de un tipo de dato asociado a un objeto y que pueden ser visibles desde fuera del objeto.

- **Estado Interno:** Variable que solo puede ser modificada por un método de un objeto.
- **Componentes de un Objeto:** Son los atributos, identidades, relaciones y métodos.
- **Identificación de un objeto:** Compuesto de atributos y entidades correspondientes a un objeto.

3.3. Objetos literales (tema)

3.3.1. Recuperación

Los valores se pueden recuperar de un objeto envolviendo una expresión de string en un [] sufijo.

Si la expresión de cadena es una constante, y si no es una palabra reservada, entonces la notación se puede utilizar en su lugar, esto porque se lee mejor.

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

3.3.2. Actualización

Un valor de un objeto puede ser actualizado por asignación. Si el nombre de la propiedad ya existe en el objeto, el valor de la propiedad se sustituye

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

3.3.3. Referencias

Los objetos se llaman por referencia y no se copian

```
var x = stooge; x.nickname = 'Curly';
 var nick = stooge.nickname;
```

3.3.4. Prototype

Cada objeto está vinculado a un prototipo de la que puede heredar propiedades. Todos objetos creados a partir de objetos literales están vinculados a *Object.prototype*, un objeto que viene por defecto.

```
if (typeof Object.create !== 'function') {
        Object.create = function (o) {
            var F = function () {};
            F.prototype = o;

            return new F();
        };
}

var another_stooge = Object.create(stooge);

another_stooge['first-name'] = 'Harry';
another_stooge['middle-name'] = 'Moses';
another_stooge.nickname = 'Moe';
```

3.3.5. Reflexión

Es fácil de inspeccionar un objeto para determinar qué propiedades tiene al intentar recuperarlas y el examen de los valores obtenidos. El operador typeof puede ser muy útil para determinar el tipo de una propiedad.

```
typeof flight.number // 'number'
typeof flight.status // 'string'
typeof flight.arrival // 'object'
typeof flight.manifest // 'undefined'
```

3.3.6. Enumeración

La enumeración incluirá todas las propiedades, incluyendo funciones y propiedades de prototipo que podría no estar interesados en lo que es necesario para filtrar los valores que no desea. Los filtros más comunes son el método **hasOwnProperty** y el uso de typeof para excluir funciones.

3.3.7. Borrar

El operador de eliminación se puede utilizar para eliminar una propiedad de un objeto, si lo tiene. No va a tocar cualquier de los objetos en el prototipo que estén vinculados.

```
another_stooge.nickname // 'Moe'
```

```
// Remove nickname from another_stooge, revealing
// the nickname of the prototype.
delete another_stooge.nickname;
another_stooge.nickname // 'Curly'
```

3.3.8. Reduccion Global

Una forma de minimizar el uso de variables globales es crear una sola variable global para su aplicación.

```
MYAPP.stooge = {
    "first-name": "Joe",
    "last-name": "Howard"
};

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"
        }
    };
```

4. Desarrollo Web

El desarrollo web es la comunicación entre 2 partes a través de un protocolo HTTP donde hay un ente (servidor) que sirve las páginas, un cliente (navegador) que solicita las páginas y un Usuario (una persona) que usa el cliente para navegar por la web.

4.1. Programación

4.1.1. A través del servidor

Se le llama programación al lado del servidor (Server-side programing), permite la entrada de usuarios, coloca la páginas en la pantalla, estructura las aplicaciones e interactúa con sistemas de almacenamiento.

Ejemplos: PHP, ASP.Net, C++, Visual Basic, casi cualquier lenguaje.

4.1.2. A través del cliente

Se le llama programación al lado del cliente (Client-side programing) y es el nombre de todos los programas eventos que se ejecutan en el cliente, como crear páginas, web dinámica, almacenamiento temporal, solicitudes del servidor, servicio remoto de aplicaciones y juegos a distancia, registro de software.

Ejemplos: Javascript, HTML, CSS.

5. Scripting Lenguaje

Un lenguaje interpretado es un lenguaje de programación que está diseñado para ser ejecutado por medio de un intérprete, en contraste con los lenguajes compilados.

Teóricamente, cualquier lenguaje puede ser compilado o ser interpretado, así que esta designación es aplicada puramente debido a la práctica de implementación común y no a alguna característica subyacente de un lenguaje en particular. Sin embargo, hay lenguajes que son diseñados para ser intrínsecamente interpretativos, por lo tanto un compilador causará una carencia de la eficacia.

Muchos autores rechazan la clasificación de lenguajes de programación entre interpretados y compilados, considerando que el modo de ejecución (por medio de intérprete o de compilador) del programa escrito en el lenguaje es independiente del propio lenguaje. A ciertos lenguajes interpretados también se les conoce como lenguajes de script.

6. Strings indexof()

El indexOf() método devuelve el índice, dentro del objeto String que realiza la llamada, de la primera ocurrencia del valor especificado, comenzando la búsqueda desde indiceBusqueda; o -1 si no se encuentra dicho valor.

Sintaxis: cadena.indexOf(valorBusqueda[, indiceDesde])

- **valorBusqueda:** Una cadena que representa el valor de búsqueda.
- **indiceDesde:** La localización dentro de la cadena llamada desde la que empezará la búsqueda. Puede ser un entero entre 0 y la longitud de la cadena. El valor predeterminado es 0.

7. Hoisting

En JavaScript, una variable puede ser declarado después de que se ha utilizado.

Ejemplo:

```
x = 5;

elem = document.getElementById("demo");
elem.innerHTML = x;

var x;
```

Aunque como buena práctica esto se debe evitar.

8. Use Strict

El Use Strict indica que el código debe ejecutarse en modo estricto y se declara así ‘”use strict”;’

Permite la escritura segura de Javascript, y avisa evitando errores de sintaxis que ponga en riesgo el código

Las futuras palabras reservadas que no están permitidas:

- implementos
- interfaz

- paquete
- privado
- protegido
- público
- estático
- rendimiento

9. Gramática

9.1. Espacio en Blanco

Necesario para separar secuencias de caracteres ya que es un remplazante de estos y permite buena identación.

9.2. Comentarios

Se escriben usando /* */ o bien // necesarios para la lectura del código, se debe evitar el uso de /* */ para evitar errores de sintaxis por lo que se recomienda usar //.

9.3. Nombres

Los nombres son una letra seguido de más letras o guiones bajos, pero no deben ser palabras reservadas ni usar símbolos y se prohíbe usar palabras reservadas como nombres de objetos.

Los nombres se utilizan para declaraciones, variables, parámetros, nombres de propiedades, operadores, y etiquetas.

9.4. Números

Se representa como un punto flotante de 64bits, no hay un número entero separado y se evita el desbordamiento de enteros cortos.

NaN es el resultado que no se pudo producir de una operación de números.

9.5. Strings

Cadenas de texto entre comillas dobles, puede colocarse de 0 a cualquier cantidad, y son de 16bits de ancho algunos carecteres se deben poner con \ y especificar un código de un carácter Unicode.

Tiene una propiedad llamada length y si dos cadenas o más se concatenan, se consideran de la misma cadena.

9.6. Declaraciones

Una unidad de compilación contiene un conjunto de instrucciones ejecutables. En los navegadores web, cada Etiqueta <script> entrega una unidad de compilación que se compila y ejecuta inmediatamente.

Al carecer de un enlazador, JavaScript todas lanza juntos en un espacio común de nombres.

Cuando se utiliza en el interior de una función, la sentencia var define la función del privado variables.

El interruptor, mientras que, para él, y no se permiten declaraciones de tener un prefijo de etiqueta opcional que interactúa con la sentencia break.
Declaraciones tienden a ser ejecutados en orden de arriba a abajo. La secuencia de ejecución pueden ser alterados por las instrucciones condicionales (si y cambio), por el bucle declaraciones (while, for, y lo hacen), por las declaraciones disruptivas (descanso, retorno y
tirar), y por la función de invocación.

Los bloques en JavaScript no crean un nuevo ámbito, por lo que las variables deben ser definidas en la parte superior de la función, no en los bloques.

9.7. Expresiones

Las expresiones más simples son un valor literal (como una cadena o un número), una variable, un valor integrado (true, false, null, indefinido, NaN o infinito), una expresión invocación precedido de nuevo, una expresión de refinamiento precedido por la cancelación, una expresión envuelto entre paréntesis, una expresión precedida por un operador de prefijo, o una expresión seguido por:

- Un operador infijo y otra expresión

- Un operador x ternario seguido por otra expresión, seguido entonces por: , y luego por una expresión más
- Una invocación
- Un refinamiento

9.8. Objetos Literales

Los objetos literales son una notación conveniente para especificar los objetos nuevos. Los nombres de las propiedades se pueden especificar como nombres o como cadenas. Los nombres son tratados como nombres literales, no como nombres de variables, por lo que los nombres de las propiedades del objeto debe ser conocido en el tiempo de compilación. Los valores de las propiedades son expresiones.

9.9. Funciones

Una función literal define un valor de función. Puede tener un nombre opcional que se puede usar para llamar a sí mismo de forma recursiva. Puede especificar una lista de parámetros que actuarán como variables inicializadas por los argumentos de invocación. El cuerpo de la función incluye definiciones y declaraciones de variables.

10. Errores comunes en Javascript

- **Usar el símbolo de asignación mal (usar = cuando se debe usar ==)**
- **Equivocar la comparación (usar == cuando se debe usar === y viceversa)**
- **Usar mal la adición (al usar strings se convierte en concatenación)**
- **Usar los decimales redondeados para comparar adiciones decimales (0.1 + 0.2 = 0.30000000000000004 y no 0.3)**
- **Poner strings en 2 líneas**
- **No poner el ;**
- **Romper el return (ej: return a * power)**
- **Poner Nombres en indices de Arrays**
- **Finalizar Arrays con ;**
- **Finalizar un objeto con ,**
- **Pensar que undefined es igual a null**
- **Poner return fuera del bloque**

11. Convenciones de Javascript

- **Nombres de las Variables**

La variables comienzan con una letra y la segunda palabra en mayúscula y junto ej: **variableNueva**

- **Usar espacios entre operadores**

Ej: var x = y + z;

- **Cuatro espacios al inicio del siguiente nivel de código**

Es importante evitar la tabulación

- **Declaración**

Abrir el corchete en la misma línea y separado de un espacio y el cierre en otra línea sin poner ;, ya que este es para terminar declaraciones.

- **Objetos**
  - Coloque el soporte de la apertura en la misma línea que el nombre del objeto.
  - Utilice dos puntos más un espacio entre cada propiedad y su valor.
  - Use comillas valores de cadena, no en torno a valores numéricos.
  - No añada una coma después de la última pareja propiedad-valor.
  - Coloque el soporte de cierre, en una nueva línea, sin espacios iniciales.
  - Siempre termine una definición de objeto con un punto y coma.

- **Romper en un coma o un operador**

Esto si la línea supera 80 caracteres.

- **No usar guiones**

Provoca que se confundan con substracción.

- **Cargado simple en HTML**

```
<script src="myscript.js">
```

- **Usar convenciones de Javascript en HTML para los nombres**

- **Usar bien las extenciones de archivo**

- **Nombres de archivos en minúscula**

12. Javascript JSON

Formato para transportar y almacenar datos ligeros a travez de objetos. Los objetos de JSON se escriben en pares entre : y los Arrays de JSON entre corchetes.

12.1. Reglas de sintaxis JSON

- Los datos son de pares nombre / valor
- Los datos se separan por comas
- Las llaves contienen objetos
- Los corchetes tienen matrices

13. Aceleración del código JavaScript

- **Reducir los Bucles**
- **Asceder al DOM una vez**
- **Reducir el tamaño del DOM**
- **No hacer variables ineccesarias**
- **Scripts al final de la página**
- **Evitar usar with**

14. Eventos de Tiempo

En javascript se puede ejecutar funciones con intervalos de tiempo usando los siguientes métodos:

14.1. Metodo setInterval()

Es un método que ejecuta una función, una y otra vez, a intervalos de tiempo especificados, su sintexis es la siguiente:

window.setInterval("javascript function", milliseconds);

Donde debe ir una function primero y luego los milisegundos.

Se puede detener usando el método clearInterval() de esta manera:

window.clearInterval(intervalVariable);

Se debe usar una variable global.

14.2. Método setTimeout()

Este método ejecuta una función, una vez, después de esperar un número especificado de milisegundos, su sintaxis es la siguiente:

window.setTimeout("javascript function", milliseconds);

Se puede detener usando el método clearTimeoutl() de esta manera:

window.clearTimeout(timeoutVariable);

Se debe usar una variable global.

15. Manejo de Errores en Javascript

Los errores pueden ser los errores cometidos por el programador, los errores debidos a la entrada equivocada, y otras cosas imprevisibles codificación

15.1. Try y Catch

Try es una declaración que define el bloque de código que se va a ejecutar mientras **Catch** define el código que se ejecitará en caso de error.

```
try {
    Block of code to try
}
catch(err) {
    Block of code to handle errors
}
```

15.2.Trow

Es una declaración que permite personalizar el mensaje de error esta acción se llama excepción:

```
throw "Too big";       // throw a text
throw 500;               // throw a number
```

15.3.Finally

Finally permite ejecutar código de todas formas:

```
try {
    Block of code to try
}
catch(err) {
    Block of code to handle errors
}
finally {
    Block of code to be executed regardless of the try / catch result
}
```

16. Funciones (tema)

Una función encierra un conjunto de sentencias. Las funciones son la unidad modular fundamental de JavaScript. Se utilizan para la reutilización de código, ocultación de información, y la composición.

Las funciones se utilizan para especificar el comportamiento de los objetos. En general, el arte de la programación es la factorización de un conjunto de requisitos en un conjunto de funciones y datos estructuras.

16.1.Objetos de función.

Las funciones en JavaScript son objetos. Los objetos son colecciones de pares nombre / valor teniendo un enlace oculto a un prototipo. Los objetos producidos a partir de objetos literales están vinculado a Object.prototype. Los objetos de

función están vinculados a Function.prototype (Que está a su vez vinculado a Object.prototype). Cada función también se crea con dos propiedades ocultas adicionales: contexto de la función y el código que implementa el comportamiento de la función.

16.2.Función Literal

Un literal de función tiene cuatro partes.

La primera parte es la función de palabra reservada.

La segunda parte opcional es el nombre de la función. La función se puede utilizar su nombre a llamar a sí mismo de forma recursiva. El nombre también puede ser utilizado por depuradores y desarrollo herramientas para identificar la función. Si una función no se le da un nombre, como se muestra en la anterior ejemplo, se dice que es anónimo.

La tercera parte es el conjunto de parámetros de la función, envueltos en paréntesis. Dentro de los paréntesis es un conjunto de nombres de parámetro cero o más, separados por comas. Estos nombres se definen como variables en la función. A diferencia de las variables normales, en lugar de ser inicializado en undefined, se inicializan a los argumentos suministrados cuando se invoca la función.

La cuarta parte es un conjunto de sentencias envueltos entre llaves. Estas declaraciones son el cuerpo de la función. Se ejecutan cuando se invoca la función.

16.3.Invocación

Hay cuatro patrones de invocación en JavaScript los patrones difieren en cómo el bono.

16.3.1. Patrón de Invocación de Método

Cuando una función se almacena como una propiedad de un objeto, lo llamamos un método. Cuando un método es invocado, esto está ligado a ese objeto. Si una expresión contiene invocación un refinamiento (es decir, una expresión. dot o [subíndice] expresión), es invocada como un método.

```
var myObject = {
    value: 0, increment: function (inc) {
```

```
            this.value += typeof inc === 'number' ? inc : 1;
        }
};
myObject.increment( );
document.writeln(myObject.value);
myObject.increment(2);
document.writeln(myObject.value);
```

16.3.2. Patrón de Invocación de Función

Cuando una función no es la propiedad de un objeto, entonces se invoca como una función:

```
suma var = sumar (3, 4); // Suma es 7
```

Cuando se invoca una función con este patrón, esto se enlaza con el objeto global. Si el método define una variable y le asigna el valor de este, la función interna tendrá acceso a este a través de esa variable.

```
myObject.double = function ( ) {
    var that = this; // Workaround.
    var helper = function ( ) {
        that.value = add(that.value, that.value);
    };
    helper( ); // Invoke helper as a function.
};

// Invoke double as a method.
myObject.double( );
document.writeln(myObject.getValue( )); // 6
```

16.3.3. Patrón de Invocación de Construcción

JavaScript es un lenguaje de herencias de prototipos. Eso significa que los objetos pueden heredar propiedades directamente de otros objetos. El lenguaje es de clase gratis.

Se trata de un cambio radical de la moda actual. La mayoría de los lenguajes de hoy son clásicos.

Herencias de prototipos es poderosamente expresivo, pero no es ampliamente entendido. JavaScript no confía en su naturaleza prototípica, ofrece un objeto de decisiones.

```
var Quo = function (string) {
    this.status = string;
};
quo.prototype.get_status = function ( ) {
    return this.status;
}; // Make an instance of Quo. var myQuo = new Quo("confused");
document.writeln(myQuo.get_status( )); // confuse
```

16.3.4. Patrón de Invocación de Aplicación

Debido a que JavaScript es un lenguaje orientado a objetos funcionales, las funciones pueden tener métodos.

El método de aplicación nos permite construir una serie de argumentos a utilizar para invocar una función.

También nos permite elegir el valor de este. El método de aplicación tiene dos parámetros. El primero es el valor que se debe obligado a ello. El segundo es una serie de parámetros.

```
// Make an array of 2 numbers and add them.

var array = [3, 4];

var sum = add.apply(null, array); // sum is 7

// Make an object with a status member.

var statusObject = {

        status: 'A-OK'

};

// statusObject does not inherit from Quo.prototype,

// but we can invoke the get_status method on

// statusObject even though statusObject does not have

// a get_status method.
```

```
var status = Quo.prototype.get_status.apply(statusObject);

// status is 'A-OK'
```

16.4. Argumentos

Son parámetros de bono que estan disponibles para las funciones, en cuanto se invoquen los argumentos de matriz. Se da el acceso a las funciones de todos los argumentos que fueron suministrados con la invocación, incluyendo el exceso de argumentos que no fueron asignados a parámetros. Esto hace que sea posible escribir funciones que toman un número no especificado de los parámetros.

16.5. Return

La instrucción de retorno se puede utilizar para hacer que la función para devolver temprano. Cuando el retorno es ejecutada, la función devuelve inmediatamente sin ejecutar las sentencias restantes.

Una función siempre devuelve un valor. Si no se especifica el valor de retorno, luego indefinido se devuelve.

16.6. Exepciones

JavaScript proporciona un mecanismo de manejo de excepciones. Las excepciones son inusuales (pero no completamente inesperado) percances que interfieren con el flujo normal de un programa.

Cuando se detecta un percance tal, el programa debe lanzar una excepción.

16.7. Tipos de Aumentos

JavaScript permite que los tipos básicos de la lengua puedan ser aumentadas. Con la adición de un método para Object.prototype hace que el método al alcance de todos objetos. Esto también funciona para las funciones, matrices, cadenas, números, expresiones regulares, y float.

16.8. Recursón

Una función recursiva es una función que llama a sí misma, ya sea directa o indirectamente. La recursividad es una técnica de programación potente en el que un problema se divide en un conjunto de sub problemas similares, cada uno resuelven con una solución trivial. Generalmente, una función recursiva llama a sí misma para resolver sus sub problemas.

16.9.Scope

Alcance en un lenguaje de programación controla la visibilidad y la duración de las variables y parámetros. Este es un servicio importante para el programador, ya que reduce de nomenclatura colisiones y proporciona gestión automática de memoria

16.10. Closure

La buena noticia sobre el alcance es que las funciones internas tienen acceso a los parámetros y variables de las funciones que se definen dentro (con la excepción de esto y argumentos). Esto es una cosa muy buena.

Nuestra función getElementsByAttribute trabajó, por haber declarado una variable resultados, y la función interna que pasó a walk_the_DOM también tuvo acceso a los resultados variable.

Un caso más interesante es cuando la función de interior tiene un tiempo de vida más largo que su exterior función.

Más temprano, hicimos una miObjeto que tenía un valor y un método de incremento. Supongamos que querido proteger el valor de cambios no autorizados.

En vez de inicializar miObjeto con un objeto literal, vamos a inicializar miObjeto por llamar a una función que devuelve un objeto literal. Esa función define una variable de valor.

Esa variable está siempre disponible para los métodos de incremento y getValue, pero el alcance de la función lo mantiene oculto al resto del programa

16.11. Callbacks

Las funciones pueden hacer que sea más fácil lidiar con eventos discontinuos. Por ejemplo, supongamos hay una secuencia que comienza con la interacción del usuario, haciendo una petición de la servidor, y finalmente se presentan la respuesta del servidor. La forma ingenua a escribir que sería:

```
request = prepare_the_request ();
respuesta = send_request_synchronously (petición);
pantalla (respuesta);
```

El problema con este enfoque es que una petición síncrona través de la red lo hará dejar al cliente en un estado de congelación. Si la red o el servidor es lento, la degradación en la capacidad de respuesta será inaceptable.

Un mejor enfoque es hacer una solicitud asincrónica, que proporciona una función de devolución de llamada que se invocará cuando se recibe la respuesta del servidor. Un asíncrono función devuelve inmediatamente, por lo que el cliente no está bloqueado.

17. Herencia (Tema)

La herencia es un tema importante en la mayoría de los lenguajes de programación.

En las lenguas clásicas (como Java), la herencia (o se extiende) proporciona dos útil servicios. En primer lugar, es una forma de reutilización de código. Si una nueva clase es principalmente similar a una ya existente clase, es suficiente para especificar las diferencias. Los patrones de reutilización de código son extremadamente

importantes porque tienen el potencial de reducir significativamente el costo de software desarrollo. La otra ventaja de la herencia clásica es que incluye la especificación de un sistema de tipos. Esto libera principalmente el programador de tener que escribir operaciones de colada explícito, que es una cosa muy buena porque cuando la fundición, se pierden los beneficios de seguridad de un sistema de tipo.

JavaScript es un lenguaje de programación relajado escrito, nunca arroja. El linaje de un objeto es irrelevante. Lo importante de un objeto es lo que puede hacer, no lo que es descendiente desde.

JavaScript proporciona un conjunto mucho más rico de patrones de reutilización de código. Se puede imitar la clásica patrón, pero también es compatible con otros modelos que son más expresivos. El conjunto de posibles patrones de herencia en JavaScript es enorme. En este capítulo, vamos a ver algunos de los patrones más sencillo. Mucho más complicado construcciones son posibles, pero por lo general es mejor que sea sencillo.

En las lenguas clásicas, los objetos son instancias de clases, y una clase puede heredar de otra clase. JavaScript es un lenguaje de prototipos, lo que significa que los objetos heredan directamente a partir de otros objetos.

18. Arreglos

Una matriz es una asignación lineal de memoria en la que se accede a los elementos de números enteros que se utilizan para calcular las compensaciones. Las matrices pueden ser estructuras de datos muy rápido.

Desafortunadamente, JavaScript no tiene nada parecido a este tipo de arreglo.

En su lugar, JavaScript proporciona un objeto que tiene algunas características en arreglos similares. Convierte subíndices matriz en cadenas que se utilizan para hacer propiedades. Es significativamente más lento que una matriz real, pero puede ser más conveniente de usar. Recuperación y actualización de propiedades funcionan igual que con los objetos, excepto que hay un truco especial con nombres de propiedad enteros. Matrices tienen su propio formato literal. Las matrices también tienen un conjunto mucho más útil de métodos incorporados.

18.1.Metodos

JavaScript proporciona un conjunto de métodos para actuar sobre arrays. Los métodos son funciones almacenado en Array.prototype. En el capítulo 3 vimos que Object.prototype puede ser aumentado. Array.prototype se puede aumentar también.

Por ejemplo, supongamos que queremos añadir un método de matriz que nos permitirá hacer cálculos en una matriz.

19. Style

Los programas de ordenador son las cosas más complejas que los seres humanos hacen. Los programas son compuesto por un gran número de piezas, expresados en funciones, sentencias y expresiones que están dispuestos en secuencias que deben estar prácticamente libres de error. El tiempo de ejecución comportamiento tiene poco que ver con el programa que lo implementa.

El software es generalmente Se espera que sea modificado en el transcurso de su vida productiva. El proceso de conversión un programa correcto en un programa correcto diferente es extremadamente difícil.

Los buenos programas tienen una estructura que anticipa, pero no está demasiado agobiado por caso las posibles modificaciones que serán necesarias en el futuro. Los buenos programas también tener una presentación clara. Si un programa se expresa bien, entonces tenemos el mejor probabilidad de ser capaz de entender de modo que se puede modificar con éxito o reparado.

Estas preocupaciones son verdaderas para todos los lenguajes de programación, y son especialmente cierto para JavaScript. Tipificación suelta de JavaScript y tolerancia de errores excesiva proporcionan poca en tiempo de compilación de aseguramiento de la calidad de nuestros programas, por lo que para compensar, hay que codificar con una estricta disciplina.

JavaScript contiene un amplio conjunto de características débiles o problemáticos que pueden socavar nuestros intentos de escribir buenos programas. Obviamente Debemos evitar los peores JavaScript características. Sorprendentemente, quizás, también debemos evitar las características que son a menudo útiles pero en ocasiones peligroso. Tales características son molestias atractivos, y evitando ellos, se evita una gran clase de errores potenciales. El valor a largo plazo de software a una organización está en proporción directa a la calidad de la base de código. Durante su vida útil, un programa estará a cargo de muchos pares de las manos y los ojos. Si un programa es capaz de comunicar con claridad su estructura y características es menos probable que se rompa cuando se modifica en el futuro nunca demasiado lejano.