Uneweb

Instituto de Comunicación Web C.A.

RIF: J-29431337-0

INTRODUCCIÓN

JavaScript es un lenguaje de programación interpretado (Como lo son **PHP**, **Perl**, **Python** ó **Ruby**), posee una sintaxis muy similar a lenguajes como **Java** y **C**.

Es un lenguaje de programación orientado a objetos como Java, ya que dispone de Herencia de Objetos, sin embargo, la implementación se basa en un paradigma de programación llamado: ***"programación orientada a prototipos"***.

2. NORMAS DEL CODIGO EN JAVASCRIPT

Las normas para poder escribir cualquier código de JavaScript se basan en 5 puntos básicos y que debemos cumplir siempre. Estas normas son las siguientes:

1. Todo el código *(sentencias)* esta dentro de funciones.
2. Las funciones se desarrollan entre las etiquetas <script> y </script>.
3. Las etiquetas "*<script>*" deben colocarse entre las etiquetas <head> y </head>.
4. Las etiquetas "*<title>*" no pueden estar colocadas entre las de "*<script>*".
5. La llamada a la función se hace a través de un evento de un elemento del documento.

3. USO DE FUNCIONES

Las funciones son un conjunto de sentencias (bloque de código) que especifica al programa las operaciones a realizar. Son útiles para evitar la repetición de líneas y modular el código. Para trabajar con ellas hay que desarrollarlas y llamarlas cuando lo necesitemos.

SINTAXIS DEL DESARROLLO:

```
function nombre_funcion([var1,var2,varN])
{
        sentencia(s);
}
```

SINTAXIS DE LA LLAMADA:

```
<elemento evento=nombre_funcion([val1,val2,valN]);>

nombre_funcion(valor1,valor2,valorN);
```

En el primero de los casos la llamada se realiza desde un elemento del documento. En el segundo caso la llamada se realiza desde el interior de otra función que también es posible.

4. LA VENTANA "ALERT"

Se trata de una ventana estándar que usamos para mostrar información en pantalla. Se puede mostrar texto, variables y texto en conjunto con variables. El diseño de la ventana ya esta definido lo único que podemos hacer es mostrar la información una o varias líneas. Su diseño y sintaxis es:

SINTAXIS:

alert("texto de la ventana");

alert(variable);

alert("texto"+variable);

5. PRIMER PROGRAMA

Ahora vamos paso a paso a construir nuestro primer programa, y así podremos ver los elementos principales del lenguaje y su colocación dentro del documento *Web*. Solo debemos seguir la teoría vista en los temas anteriores.

EJEMPLO 1: Llamada a una función desde un elemento del documento.

```
<html>
    <head>

        <script>
            function hola()
            {
                alert("Hola a todos");
            }
        </script>

        <title>Autor:Ricardo Amezua</title>
    </head>

    <body onLoad=hola();>
    </body>
</html>
```

EJEMPLO 2: Llamada a una función desde otra.

```
<html>
    <head>

        <script>
            function hola()
            {
                alert("Hola a todos");
                rehola();
            }

            function rehola()
            {
                alert("hola de nuevo a todos");
            }
        </script>

        <title>Autor:Ricardo Amezua</title>
    </head>

    <body onLoad=hola();>
    </body>
</html>
```

6. EVENTOS

Un evento es un mecanismo por el cual podemos detectar las acciones que realiza el usuario y llamar automáticamente a la función que tengamos asociada. Desde esta función realizaremos las acciones que tengamos desarrolladas.

SINTAXIS:

```
<elemento nombre_evento=nombre_funcion([parametros]);>
```

La siguiente tabla muestra los eventos y manipuladores de JavaScript:

EVENTO	***SE PRODUCE AL..***
onLoad	Terminar de cargar una página o frame (entrar).
onUnLoad	Descargar una página o frame (salir).
onAbort	Abortar la carga de una página.
onError	Producirse algún error en la carga de la página.
onMouseOver	Pasar el ratón por encima de un enlace, area o frame.
onMouseOut	Dejar de estar el ratón encima de un enlace, area o frame.
onMouseMove	Mover el ratón por el documento.

onKeyUp	Presionar una tecla.
onClick	Hacer click con el ratón.
onResize	Dimensionar la ventana del navegador.
onMove	Mover la ventana del navegador.
onChange	Modificar texto en un control de edición. Sucede al perder el foco.
onSelect	Seleccionar texto en un control de edición.
onFocus	Situar el foco en un control.
onBlur	Perder el foco un control.
onSubmit	Enviar un formulario.
onReset	Inicializar un formulario.

EJEMPLO 1:

```
<html>
    <head>
        <script>
            function hola(){alert("Hola a todos");}
            function adios(){alert("Adios a todos");}
        </script>

        <title>Autor:Ricardo Amezua</title>
    </head>

    <body onLoad=hola(); onUnload=adios();>
    </body>
</html>
```

EJEMPLO 2:

```
<html>
    <head>
        <script>
            function pulsa(){alert("Autor:RICARDO AMEZUA");}
            function foco(){alert("Foco en la primera Caja");}
            function tecla(){alert("Pulsada tecla");}
        </script>

        <title>Autor:Ricardo Amezua</title>
    </head>

    <body>
        <input type="button" value="Autor" onClick=pulsa();>
        <input type="text" size=”5” onFocus=foco();>
        <input type="text" size=”5” onKeyPress=tecla();>
    </body>
</html>
```

EJEMPLO 3:

```
<html>
    <head>
        <script>
            function cambio(){alert("Cambiado el tamaño");}
        </script>

        <title>Autor:Ricardo Amezua</title>
    </head>

    <body onResize=cambio();>
    </body>
</html>
```

7. VARIABLES Y CONSTANTES

VARIABLES:

Espacio de memoria con un nombre reservado para guardar información mientras la página este cargada. El primer paso para poder trabajar con variables es declararlas, que es lugar donde se les da su nombre y su ámbito.

Para ***dar nombre*** a una variable debemos tener en cuenta las siguientes normas:

1. No pueden contener espacios.
2. Distingue entre mayúsculas y minúsculas.
3. No pueden contener acentos, puntos o cualquier signo gramatical.
4. No pueden comenzar con un dígito ni contener la letra “ñ”.
5. Nombre único y exclusivo para cada variable salvo que estén es 2 funciones distintas.

El ***ámbito de una variable*** define si la variable se podrá utilizar en cualquier parte del documento (*es global*). O si solo se podrá utilizar dentro de una función determinada (*es local*). La declaración de las variables globales se realiza dentro de las etiquetas “*<script>*” pero fuera de cualquier función. La declaración de las variables locales se realiza dentro de la función que nos interese usar esa variable.

La sintaxis para declarar una variable es igual en ambos casos, la única diferencia es el lugar donde las declaramos. La siguiente línea nos muestra como hacerlo:

```
var nombre_variable[=valor];
```

El tipo de variable es asignado automáticamente por JavaScript. Depende del primer valor que se guarde en la variable. Por tanto los tipos de variable existentes son los que mostramos en la siguiente tabla:

TIPO	*VALORES*
numérica	Cualquier tipo numérico
boolean	True o False.
String	Texto o letra.

Otro aspecto importante, es la ***conversión de datos***, que en JavaScript es automática. Transforma los datos de todas la variables en una expresión según el tipo de la primera variable. No es muy segura y puede acarrear muchos problemas.

EJEMPLO:

```
num1=”12”;
num2=10;

x=num1+num2;// daria como resultado 1210.
y=num2+num1;// daria como resultado 22.
```

Para evitar problemas en las conversiones, se pueden utilizar *métodos* ya implementados que realizan la conversión de una manera más segura.

TIPO DE CONVERSION	*SINTAXIS*
De texto a numero entero.	var_numerica=parseInt(var_texto);
De texto a coma flotante (decimal).	var_numerica=parseFloat(var_texto);
De numérica a texto.	Es automática sin peligro.

EJEMPLO:

```
<html>
    <head>
        <script>
            var global=100;

            function uno()
            {
                var local_uno=1;
                alert("Global " +global +” Local “ +local_uno);
                dos();
            }

            function dos()
            {
                var local_dos=2;
                alert("Global " +global +” Local “ +local_dos);
            }
```

```
        </script>

        <title>Autor:Ricardo Amezua</title>
    </head>

    <body onLoad=uno();>
    </body>
</html>
```

CONSTANTES (*"literales"*)

Los valores iniciales que se les asigna son invariables. Estos no son variables, sino expresiones constantes. Los tipos de literales son los mismos que en las variables, según el primer dato que almacenemos será de un tipo u otro.

TIPO	*VALORES*
numérica	Cualquier tipo numérico
boolean	True o False.
String	Texto o letra.

8. OPERADORES

JavaScript define TRES tipos de operadores: aritméticos, relacionales y lógicos. También hay definido un operador para realizar determinadas tareas, como las asignaciones.

ASIGNACION *(=)*

En JavaScript se puede utilizar el operador de asignación en cualquier expresión valida. Solo con utilizar un signo de igualdad se realiza la asignación. El operador destino (parte izquierda) debe ser siempre una variable, mientras que en la parte derecha puede ser cualquier expresión valida. Es posible realizar asignaciones múltiples, igualar variables entre si y a un valor.

SINTAXIS:

```
variable=valor;

variable=variable1;

variable=variable1=variable2=variableN=valor;
```

ARITMÉTICOS

Pueden aplicarse a todo tipo de expresiones. Son utilizados para realizar operaciones matemáticas sencillas, aunque uniéndolos se puede realizar cualquier tipo de operaciones. En la siguiente tabla se muestran todos los operadores aritméticos.

OPERADOR	*DESCRIPCIÓN*
-	Resta.
+	Suma
*	Multiplica
/	Divide
%	Resto de una división
--	Decremento en 1.
++	Incrementa en 1.
vari+=valor	Incrementa el valor de vari.
vari-=valor	decrementa el valor de vari.
vari*=valor	Multiplica el valor de vari.

LÓGICOS Y RELACIONALES

Los operadores relacionales hacen referencia a la relación entre unos valores y otros Los lógicos se refiere a la forma en que esas relaciones pueden conectarse entre si. Los veremos a la par por la estrecha relación en la que trabajan.

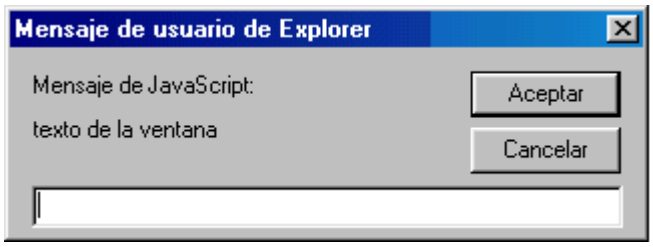
OPERADORES RELACIONALES	
OPERADOR	*DESCRIPCIÓN*
<	Menor que.
>	Mayor que.
<=	Menor o igual.
>=	Mayor o igual
= =	Igual
! =	Distinto

OPERADORES LÓGICOS	
OPERADOR	*DESCRIPCIÓN*
&&	Y (AND)
| |	O (OR)
!	NO (NOT)

9. INTRODUCCIÓN DE DATOS

JavaScript permite interactuar al usuario por medio de la introducción de datos. La introducción de datos se puede realizar por medio de la ventana prompt o utilizando controles como cajas de texto.

VENTANA PROMPT:



SINTAXIS:

vari=prompt(“Texto de la ventana”,”valor inicial caja”);

Al pulsar el botón aceptar, el contenido de la caja pasa a *vari*. Si se pulsa el botón cancelar, el contenido de la caja se pierde y *vari* queda con valor null.

EJEMPLO:

```
<html>
    <head>
        <script>
            function valor()
            {
                var nombre;
                nombre=prompt("Introduce Nombre:","");
                alert("hola "+nombre);
            }
        </script>
    </head>

    <body onload=valor();>
    </body>
</html>
```

CAJA DE TEXTO:

Otro mecanismo por el cual se pueden introducir datos, es mediante las cajas de texto. Todo el trabajo con las cajas, esta basado en funciones y métodos ya implementadas en JavaScript. Estas son las funciones que podemos utilizar:

FUNCIÓN	***DESCRIPCIÓN***
variable=nombre_caja.value;	Guarda el contenido de la caja
nombre_caja.value=valor o variable;	Muestra en la caja el valor.
nombre_caja.value=””;	Limpia el contenido de la caja.
nombre_caja.sefocus();	Envía el foco a la caja.

EJEMPLO:

```
<html>
    <head>
        <script>
```

```
            function muestra()
            {
                var nombre=cnombre.value;
                alert("Eres "+nombre);
                cnombre.value="";
                cnombre.focus();
            }
        </script>
    </head>

    <body>
        Nombre:<input type="text" name="cnombre" size="20">
        <input type="button" value="Ver" onClick=muestra();>
    </body>
</html>
```

10. SENTENCIAS DE CONTROL

Es la manera que tiene JavaScript de provocar que el flujo de la ejecución avance y se ramifique en función de los cambios y estado de los datos.

IF-ELSE:

La ejecución atraviesa un conjunto de estados boolean que determinan que bloques de código se ejecutan. Con la utilización de esta sentencia nunca se realizarán ambos bloques de código.

SINTAXIS: En caso de ser una sola sentencia por parte las llaves son opcionales.

```
            if (expresion-booleana)
            {
            Sentencia(s);
{
[else]
{
                    Sentencia(s);
            }
```

La cláusula else es opcional. La expresión puede ser de cualquier tipo y más de una (siempre que se unan mediante operadores lógicos).Otra opción posible es la utilización de if anidados, es decir unos dentro de otros compartiendo la cláusula else.

EJEMPLO 1:

```
    <html>
        <head>
            <script>
                function ver()
                {
                    var edad=parseInt(cedad.value);

                    if(edad<=18)
                        alert("No tienes acceso\nDebes tener 18");
                    else
                        alert("Bienvenido a la pagina");
```

```
            }
        </script>
        <title>Pagina nueva 1</title>
    </head>

    <body>
        Edad:
        <input type="text" name="cedad" size="3" onBlur=ver();>
    </body>
</html>
```

EJEMPLO 2:

```
<html>
    <head>
        <script>
            function ver()
            {
                var edad=parseInt(cedad.value);

                if(edad<=18)
                    alert("Abono Joven");
                else
                {
                    if(edad>=65)
                        alert("Abono 3ª Edad");
                    else
                        alert("Abono normal");
                }
            }
        </script>
        <title>Pagina nueva 1</title>
    </head>

    <body>
        Edad:
        <input type="text" name="cedad" size="3" onBlur=ver();>
    </body>
</html>
```

SWITCH:

Es una sentencia muy similar a if-else. Si los valores con los que se compara son números se pone directamente el pero si es texto se debe encerrar entre comillas. La sintaxis de esta sentencia es:

SINTAXIS:

```
switch (expresión){
case constante1:
                sentencia(s);
break;
case constante2:
                sentencia(s);
break;
```

```
case constante3:
        sentencia(s);
break;
case constanteN:
        sentencia(s);
break;
[default:]
    sentencia(s);
}
```

El valor de la expresión se compara con cada una de las constantes de la sentencia case, si coincide alguno, se ejecuta el código que le sigue, y cuando ejecuta break sale de este bloque hasta salir del switch. Si ninguno coincide se realiza el bloque default (opcinal), si no lo hay no se ejecuta nada.

En el caso que varias sentencias case realicen la misma ejecución se pueden agrupar, utilizando una sola sentencia break. Evitamos de este modo duplicar líneas de código. La sintaxis es la siguiente:

SINTAXIS:

```
switch (expresión){
    case constante1:
case constante5:
        sentencia(s);
    break;
    case constante3:
        sentencia(s);
    break;
    [default:]
        sentencia(s);
}
```

EJEMPLO:

```
<html>
    <head>
        <script>
            function espe()
            {
                var tipo=cespe.value;
                switch(tipo)
                {
                    case "humano":
                        alert("Eres un Humano");
                        break;
                    case "planta":
                        alert("Eres un Vegetal");
                        break;
                    case "animal":
                        alert("Eres del reino Animal");
                        break;
                    default:
                        alert("Especie Desconocida");
                        break;
                }
            }
        </script>
```

```
        </head>
        <body>
            ESPECIE:
            <input type="text" name="cespe" size="20" onBlur=espe();>
        </body>
    </html>
```

WHILE:

Ejecuta repetidamente el mismo bloque de código hasta que se cumpla una condición de terminación. Hay cuatro partes en todos los bucles. *Inicialización, cuerpo, iteración* y *condición*.

SINTAXIS:

```
        [inicialización;]
            while(condicion[es])
{
            cuerpo;
            [iteración;]
            }
```

DO..WHILE:

Es lo mismo que en el caso anterior pero aquí como mínimo siempre se ejecutará el cuerpo del bucle una vez, en el tipo de bucle anterior es posible que no se ejecute ni una sola vez el contenido de este.

SINTAXIS:

```
        [inicialización;]
    do{
cuerpo;
[iteración;]
}while(condición);
```

FOR:

Realiza las mismas operaciones que en los casos anteriores pero la sintaxis es una forma compacta. Normalmente la condición para terminar es de tipo numérico. La iteración puede ser cualquier expresión matemática valida. Si de los 3 términos que necesita no se pone ninguno se convierte en un bucle infinito.

SINTAXIS: En caso de ser una sola sentencia por parte las llaves son opcionales.

```
for (inicio;cond_fin;iteración)
{
        sentencia(S);    }
```

EJEMPLO: El tipo de bucle puede ser cualquiera de los 3 (for, while, do..while).

```
    <html>
        <head>
            <script>
                function opt()
```

```
                {
                    while(valor<=10)
                    {
                        alert("Esto sale 10 veces:"+ valor);
                        valor++;
                    }
                }
            </script>
        </head>
        <body>
            <a href="Ejemplo.htm" onMouseOver=opt();>ir a uno</a>
        </body>
    </html>
```

Dentro de las *sentencias de control* se pueden incluir las ***sentencias de ruptura*** ya que van muy ligadas a los bucles. Estas sentencias de ruptura son continue y break. A continuación vamos a ver como actúa cada una de ellas

La sentencia continue lo que hace es ignorar las sentencias que tiene el bucle y saltar directamente a la condición para ver si sigue siendo verdadera, si es así, sigue dentro del bucle, en caso contrario, saldría directamente de el.

La sentencia break tiene una operatoria más drástica que la sentencia continue, en vez de saltar a la línea de la condición para comprobar su estado, lo que hace es abandonar directamente el bucle dándolo por terminado.

EJEMPLO 1: Comparamos la actuación de continue y break.

CONTINUE

El bucle terminara cuando muestre el 10

```
<html>
 <head>
  <script>
     function bucle()
     {
          var cont=1;
          while(cont<=10)
          {
              alert(cont);
              cont++;
              if(cont==5)
                continue;
            }
     }
  </script>
 </head>
 <body onLoad=bucle();></body> </html>
```

BREAK

El bucle terminara cuando muestre el 4

```
<html>
 <head>
  <script>
     function bucle()
     {
          var cont=1;
          while(cont<=10)
          {
              alert(cont);
              cont++;
              if(cont==5)
                break;
            }
     }
  </script>
 </head>
 <body onLoad=bucle();></body> </html>
```

EJEMPLO 2:

CONTINUE

El bucle terminara cuando muestre el 10

```
<html>
```

CONTINUE

El bucle terminara cuando muestre el 4

```
<html>
```

```
<head>
 <script>
     function bucle()
     {
          var cont=1;
          while(cont<=10)
          {
              alert(cont);
              cont++;
              if(cont==5)
                continue;
            }
     }
 </script>
</head>
<body onLoad=bucle();>
</body>
</html>
```

```
<head>
 <script>
     function bucle()
     {
          var cont=1;
          while(cont<=10)
          {
              alert(cont);
              cont++;
              if(cont==5)
     {
       cont=30;
       continue;
     }
            }
     }
 </script>
</head>
<body onLoad=bucle();>
</body>
</html>
```

También podríamos considerar como sentencia de ruptura (sin serlo), un tipo de ventana estándar llamada confirm. Este tipo de ventana nos permite elegir entre 2 opciones, cada una con un valor de retorno. Con lo que se puede llamar a las sentencias de ruptura (break, continue) según la opción que se elija.

SINTAXIS:

var_boolean=confirm("texto de la ventana");

BOTON PULSADO	*VALOR DE RETORNO*
ACEPTAR	true
CANCELAR	false

EJEMPLO 1:

```
<html>
    <head>
        <script>
            function confirma()
```

```
            {
                var respuesta=confirm("Pulsa un botón");

                if (respuesta==true)
                    alert("Has pulsado ACEPTAR");
                else
                    alert("Has pulsado CANCELAR");
            }
        </script>
    </head>

    <body onLoad=confirma();>
    </body>
</html>
```

EJEMPLO 2:

```
<html>
    <head>
        <script>
            function salida()
            {
                var cont=1;
                var paso=1;
                var res;

                for(cont=1;cont<=100;cont++)
                {
                    alert("Paso " +cont);
                    if(paso==10)
                    {
                        paso=0;
                        res=confirm("¿Desea Seguir?");
                        if (res==false)
                        break;
                    }
                    paso++;
                }
            }
        </script>
    </head>

    <body onLoad=salida();>
    </body>
</html>
```

Asynchronous JavaScript and XML

AJAX parece ser la palabra de moda en el mundo del desarrollo de aplicaciones Web, AJAX no es una tecnología, sino la unión de varias tecnologías que juntas pueden lograr cosas realmente impresionantes. Hace un tiempo AJAX parece ser la palabra de moda en el “mundo” del desarrollo de aplicaciones Web; de hecho muchos lo escuchan nombrar pero pocos saben que es realmente y, menos aún, saben en donde buscar información clara sobre que es esta nueva “maravilla” de la tecnología que Jesse James Garret publicó en un articulo en ingles (http://adaptivepath.com/publications/essays/archives/000385.php) que vale la pena traducir por completo.

¿Por qué es tan interesante AJAX?

Porque en realidad AJAX no es una tecnología, sino la unión de varias tecnologías que juntas pueden lograr cosas realmente impresionantes como GoogleMaps, Gmail el Outlook Web Access (http://www.franklinmint.fm/blog/archives/000294.html o algunas otras aplicaciones muy conocidas.

AJAX, en resúmen, es el acrónimo para Asynchronous JavaScript + XML y el concepto es: Cargar y renderizar una página, luego mantenerse en esa página mientras scripts y rutinas van al servidor buscando, en background, los datos que son usados para actualizar la página solo re-renderizando la página y mostrando u ocultando porciones de la misma.

Ajax: Un Nuevo acercamiento a las Aplicaciones Web

Si algo del actual diseño de interacción puede ser llamado glamoroso, es crear Aplicaciones Web. Después de todo, ¿cuando fue la ultima vez que escuchaste a alguien hablar de diseño de interacción de un producto que no esté en la Web? (Okay, dejando de lado el iPod). Todos los nuevos proyectos cool e innovadores están online.

Dejando de lado esto, los diseñadores de interacción Web no pueden evitar sentirse envidiosos de nuestros colegas que crean software de escritorio. Las aplicaciones de escritorio tienen una riqueza y respuesta que parecía fuera del alcance en Internet. La misma simplicidad que ha permitido la rápida proliferación de la Web también crea una brecha entre las experiencias que podemos proveer y las experiencias que los usuarios pueden lograr de las aplicaciones de escritorio.

Esa brecha se está cerrando. Échenle una mirada a las Google Suggest (http://www.google.com/webhp?complete=1&hl=en). Mira la forma en que los términos sugeridos se van actualizando a medida que uno tipea casi instantáneamente. Ahora mire Google Maps. Hace zoom. Usen el cursor para agarrar el mapa y navegarlo un poco. Otra vez, todo sucede casi instantáneamente, sin esperar que las paginas se recarguen.

Google Suggest y Google Maps son dos ejemplos de un nuevo acercamiento a las aplicaciones Web, que nosotros en Adaptative Path hemos denominado AJAX. El nombre es una abreviación o acrónimo para Asynchronous JavaScript + XML, y ello representa un cambio fundamental en que es posible en la Web.

Definiendo Ajax

Ajax no es una tecnología. Es realmente muchas tecnologías, cada una floreciendo por su propio mérito, uniéndose en poderosas nuevas formas. AJAX incorpora:

- presentación basada en estándares usando XHTML y CSS (http://adaptivepath.com/publications/essays/archives/000266.php);
- exhibición e interacción dinámicas usando el Document Object Model http://www.scottandrew.com/weblog/articles/dom_1
- Intercambio y manipulación de datos usando XML and XSLT (http://www-106.ibm.com/developerworks/xml/library/x-xslt/?article=xr);
- Recuperación de datos asincrónica usando XMLHttpRequest (http://www.xml.com/pub/a/2005/02/09/xml-http-request.html);
- y JavaScript (http://www.crockford.com/javascript/javascript.html) poniendo todo junto.

El modelo clásico de aplicaciones Web funciona de esta forma: La mayoría de las acciones del usuario en la interfaz disparan un requerimiento HTTP al servidor web. El servidor efectúa un proceso (recopila información, procesa números, hablando con varios sistemas propietarios), y le devuelve una pagina HTLM al cliente.
Este es un modelo adaptado del uso original de la Web como un medio hipertextual, pero como fans de “The Elements of User Experience (http://www.jjg.net/elements/) sabemos, lo que hace a la Web buena para el hipertexto, no la hace necesariamente buena para las aplicaciones de software.

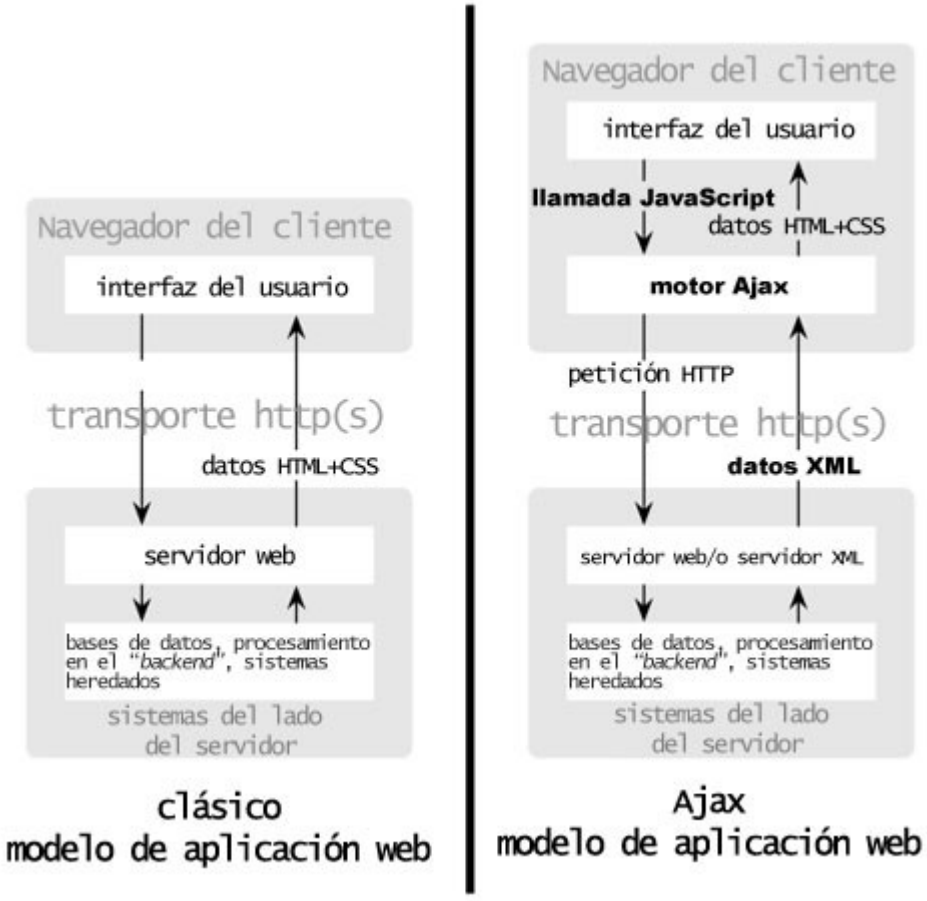


Figura 1: El modelo tradicional para las aplicaciones Web (izq.) comparado con el modelo de AJAX (der.).

Este acercamiento tiene mucho sentido a nivel técnico, pero no lo tiene para una gran experiencia de usuario. Mientras el servidor esta haciendo lo suyo, que esta haciendo el usuario? Exacto, esperando. Y, en cada paso de la tarea, el usuario espera por mas.

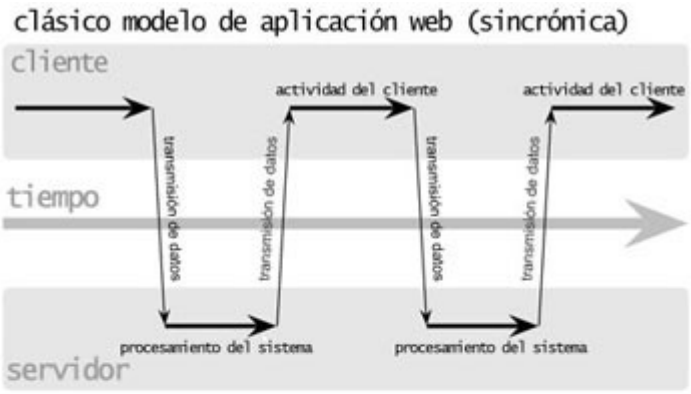
Obviamente, si estuviéramos diseñando la Web desde cero para aplicaciones, no querríamos hacer esperar a los usuarios. Una vez que la interfaz esta cargada, porque la interacción del usuario debería detenerse cada vez que la aplicación necesita algo del servidor? De hecho, porque debería el usuario ver la aplicación yendo al servidor?

Como es diferente AJAX

Una aplicación AJAX elimina la naturaleza “arrancar-frenar- arrancar-frenar” de la interacción en la Web introduciendo un intermediario -un motor AJAX- entre el usuario y el servidor. Parecería que sumar una capa a la aplicación la haría menos reactiva, pero la verdad es lo contrario.

En vez de cargar un pagina Web, al inicio de la sesión, el navegador carga al motor AJAX (escrito en JavaScript y usualmente “sacado” en un frame oculto). Este motor es el responsable por renderizar la interfaz que el usuario ve y por comunicarse con el servidor en nombre del usuario.

El motor AJAX permite que la interacción del usuario con la aplicación suceda asincrónicamente (independientemente de la comunicación con el servidor). Así el usuario nunca estará mirando una ventana en blanco del navegador y un icono de reloj de arena esperando a que el servidor haga algo.



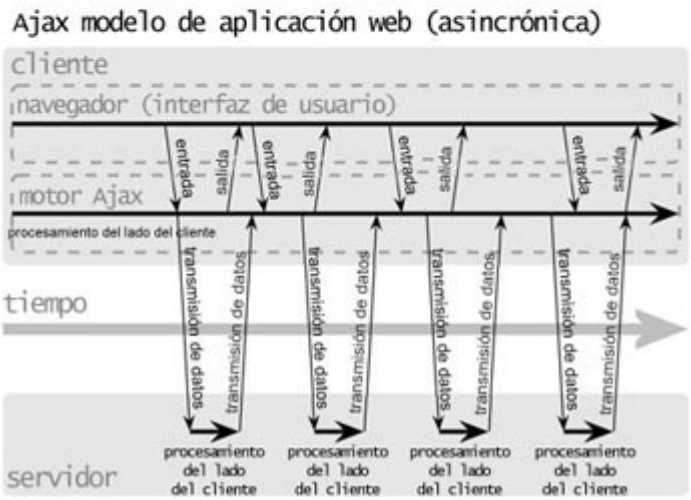


Figura 2: El patrón de interacción sincrónica de una aplicación Web tradicional (arriba) comparada con el patrón asincrónico de una aplicación AJAX (abajo).

Cada acción de un usuario que normalmente generaría un requerimiento HTTP toma la forma de un llamado JavaScript al motor AJAX en vez de ese requerimiento. Cualquier respuesta a una acción del usuario que no requiera una viaje de vuelta al servidor (como una simple validación de datos, edición de datos en memoria, incluso algo de navegación) es manejado por su cuenta.

Si el motor necesita algo del servidor para responder (sea enviando datos para procesar, cargar código adicional, o recuperando nuevos datos) hace esos pedidos asincrónicamente, usualmente usando XML, sin frenar la interacción del usuario con la aplicación.

¿Quién está usando Ajax?

Google está haciendo una significativa inversión en el acercamiento Ajax. Todos los grandes productos que Google ha introducido en el ultimo año (Orkut, Gmail, la última versión de Google Groups, Google Suggest, y Google Maps) son aplicaciones Ajax. (Para datos más técnicos de estas implementaciones Ajax, lean estos excelentes análisis de Gmail, Google Suggest, y Google Maps.) Otros están siguiendo la tendencia: muchas de las funciones que la gente ama en Flickr dependen de Ajax, y el motor de búsqueda de Amazon A9.com aplica tecnologías similares.

Estos proyectos demuestran que Ajax no es solo técnicamente importante, sino también prácticos para aplicaciones en el mundo real. Esta no es otra tecnología que solo trabaja en un laboratorio. Y las aplicaciones Ajax pueden ser de cualquier tamaño, de lo más simple, funciones simples como Google Suggest a las muy complejas y sofisticadas como Google Maps.

En Adaptive Path, estuvimos haciendo nuestro propio trabajo con Ajax en los últimos meses, y estamos descubriendo que solo raspamos la superficie de la rica interacción y respuesta que que las aplicaciones Ajax puede proveer. Ajax es un desarrollo importante para las aplicaciones Web, y su importancia solo va a crecer. Y como hay tantos desarrolladores que ya conocen como usar estas tecnologías, esperamos ver mas empresas y organizaciones siguiendo el liderazgo de Google en explotar la ventaja competitiva que Ajax provee.

Frameworks de Javascript.

¿Que es un Framework?

Un **framework**, en el desarrollo de software, es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

¿Cuales frameworks de javascript puedo usar?

Mootools: "El framework javascript compacto"
Este producto tiene buena pinta. Según parece es sencillo y bien planificado. Entre las virtudes que he visto más destacadas es que es ligero, pudiendo incluso definir qué partes del framework incluir y cuales no, para que se carguen los scripts más rápido en el cliente. A mi algunas personas me han hablado muy positivamente de este framework, así que quizás sea por el que empiece la investigación en detalle.
http://mootools.net/

JQuery: "Librería Javascript para escribir menos y hacer más"
Parece ser que este es uno de los frameworks con más aceptación, por estar estupendamente documentado y por ser muy simple y permitir desarrollar con un código limpio y elegante. El peso de las librerías es razonable y además tiene muchos fans incondicionales, por lo que no me cabe duda que será un buen proyecto.
http://jquery.com/

Prototype: "El framework javascript cuyo propósito es facilitar el desarrollo de aplicaciones dinámicas"
Este framework también resulta muy interesante, pues hay muchos usuarios que lo utilizan habitualmente y con éxito. Parece una opción altamente profesional y además tiene la garantía que lo utilizan para la creación de sus webs empresas muy conocidas a nivel mundial. A mi me ofrece muchas garantías, pero hay ciertos detractores que acusan a este framework de ser muy pesado y ralentizar los sitios web donde se utiliza.
http://www.prototypejs.org/

YUI: "The Yahoo! User Interface Library"
Es un framework que utilizan los desarrolladores de Yahoo! para hacer su portal, que hace tiempo se ha distribuido para uso libre. Que provenga de Yahoo! para mi ya resulta una importante garantía y parece que tiene desarrollados una importante gama de controles y componentes. Tendría que probarlo personalmente para dar una opinión, pero parece que hay muchas personas que también lo acusan de ser un poco pesado.
http://developer.yahoo.com/yui/

Dojo: "Experiencias grandes… para cualquiera"
Parece un producto también bastante atractivo y una opción seria. No obstante, he leído opiniones discordantes acerca de él. Algunos no dudan en calificarlo entre los mejores frameworks Javascript y otros acusan que es pesado y poco depurado, que arroja errores bastante fácilmente.
http://www.dojotoolkit.org/

Qooxdoo: "La nueva era del desarrollo web"
Es un framework Javascript ajax multipropósito, opensource con dos tipos de licencia. Her leído pocas opiniones sobre este software, pero parece digno de considerar.
http://qooxdoo.org/

GWT Google Web Toolkit: "construye aplicaciones Ajax en lenguaje Java"
Es un conjunto framework opensource desarrollado en Java, con el que se han creado aplicaciones populares de Google, como Google Maps o Gmail. Sin duda, al tratarse de un producto de Google, no cabe duda que es una opción a considerar seriamente. Tiene un compilador que convierte las clases Java en código Javascript y HTML compatible con todos los navegadores.
http://code.google.com/webtoolkit/

Rico: "Javascript para aplicaciones de Internet de contenido enriquecido"
Otra de las opciones más conocidas para desarrollar aplicaciones para la web 2.0. Es open source y ya se encuentra en la versión 2.0, con lo que se supone que el tiempo de vida le haya ayudado a ser más depurado. He leído por ahí que está poco documentado.
http://openrico.org/rico/home.page

Ext JS: "Documentación, diseño y código limpio"
Este framework Javascript parece ser otra de las opciones serias. Se distribuye bajo licencia Open Source (gratis) y licencia comercial (de pago, pero con soporte y alguna funcionalidad adicional). Lo utilizan empresas bastante importantes, como Adobe. Me ha llamado la atención que tiene soporte para Adobe Air.
http://extjs.com/

Existen otros Frameworks, sin embargo estos son los mas recomendados, aca coloco algunos otros que vale la pena revisar:

- The Foo Framework (un framework basado en Prototype): http://foo.riiv.net/
- script.aculo.us (también basado en Prototype): http://script.aculo.us/
- AJS (Framework Javascript ultraligero): http://orangoo.com/labs/AJS/ ZK (Ajax web framework): http://www.zkoss.org/