Desfufor

Desarrollo futuro formacion

Vocación por
LA ENSEÑANZA

Gobelas 17 bajo Izq
28023 La Florida
Aravaca, Madrid
911517122
info@desfufor.es
www.desfufor.es
SKYPE: desfufor

desarrollo
futuro
formación

USO AVANZADO HTML 5

PROGRAMACIÓN HTML 5 Y SU API

HTML 5 avanzado

- HTML 5 nos proporciona una serie de APIS que nos permite llevar a cabo tecnicas mas avanzadas no centrandonos solo en el diseño de paginas web.

- Algunas de ellas son:
    - **API Geolocalización**
    - **API Almacenamiento**
    - **API Sockets**
    - **API WebWorkers**
    - **API Eventos**

CANVAS Y SVG

Canvas

- Un lienzo permite dibujar en el documento HTML y actualizar dinámicamente estos dibujos, por medio de JavaScript. También puede disparar acciones a partir de los eventos generados por el usuario

```
<canvas id="micanvas" width="200" height="100">
 Este texto se muestra para los navegadores no compatibles con canvas.
<br>
 Por favor, utiliza Firefox, Chrome, Safari u Opera.
</canvas>
```

CANVAS Y SVG

Canvas

Para utilizar el canvas se debe referenciar primero el elemento canvas y adquirir su contexto.

```
var canvas = document.getElementById('entorno_canvas');
var context = canvas.getContext('2d');
```

Una vez adquirido, se puede empezar a dibujar en la superficie del canvas usando la API.

CANVAS Y SVG

Canvas

beginPath

Le dice al contexto del canvas que se va a empezar a dibujarse un camino, no tiene ningún parámetro.

Una vez invocada la función se puede empezar a dibujar el camino añadiendo segmentos para completarlo con las diferentes funciones.

moveTo

Sirve para definir el punto donde se comienza a dibujar el segmento.

No dibuja nada, recibe como parámetro los puntos x e y donde ha de moverse el puntero para dibujo.

CANVAS Y SVG

Canvas

lineTo

Dibuja una línea recta desde la posición actual hasta el punto (x,y) que se indique como parámetro. La posición actual del camino se establece indicado previamente con un moveTo(), o donde se haya terminado el trazo anteriormente dibujado.

fill

Este método del contexto del canvas sirve para rellenar de color el área circunscrita por un camino. Para rellenar de color un path, el path tiene que estar cerrado, por lo que, si no lo está, automáticamente se cerrará con una línea recta hasta el primer punto del camino, es decir, donde se comenzó a dibujar.

CANVAS Y SVG

Canvas

closePath

Sirve para cerrar un path, volviendo a su punto inicial de dibujo, no recibe ningún parámetro.

fillText

Permite dibujar texto en el canvas:

```
contexto.font = "bold 12px sans-serif";
contexto.fillText("Hola", 5, 5);
```

stroke

Es similar al método fill pero traza solo la silueta y no la rellena.

CANVAS Y SVG

Canvas

```
ctx.beginPath();
ctx.moveTo(50,5);
ctx.lineTo(75,65);
ctx.lineTo(50,125);
ctx.lineTo(25,65);
ctx.closePath();
ctx.fill();
```

CANVAS Y SVG

SVG

SVG (Scalable Vector Graphics) es un formato vectorial poco conocido pero muy útil para su uso online por su flexibilidad y por la capacidad de ofrecer gráficos con calidad.

la sintaxis de esta etiqueta:

```
<svg width="100" height="100">
...
</svg>
```

XMLHttpRequest Nivel 2

Definición

- En febrero del 2008 la W3C publicó el primer borrador de la nueva versión de este objeto, **"XMLHttpRequest Level 2"**.
- Entre las mejoras que presenta se encuentran
    - **la gestión del progreso de eventos,**
    - **peticiones entre distintos servidores**
    - **el manejo de flujos de bytes.**
- Nos centraremos en esta última característica ya que es la que nos permitirá subir archivos al servidor.

XMLHttpRequest Nivel 2

Realizando la petición

```
$(document).ready
(
    function ()
    {
        var xhr = new XMLHttpRequest();
        $("#input_file").change
        (
            function(event)
            {
                var file = event.target.files[0];
                xhr.open('POST', 'ajax_controller.php', true);
                xhr.setRequestHeader("X_FILENAME", file.name);
                xhr.send(file);
            }
        );
    }
);
```

PROCESO DE EVENTOS

Drag & Drop

Es una característica, que permite poder arrastrar elementos de un lado a otro en la interfaz.

En HTML5 aparecen:

- Nuevos eventos

  dragstart, drag, dragenter, dragover,

  dragleave, drop, dragend.

- Atributo draggable="true“ para declarar que un elemento se puede arrastrar.

- Posibilidad de establecer la imagen "ghost" mostrada mientras se desplaza.

- Efectos asociados a copiar, mover…

Proceso de eventos

Drag & Drop (Eventos)

• **dragstart:** Comienza el arrastrado. El "target" del evento será el elemento
que está siendo arrastrado.

• **drag:** El elemento se ha desplazado. El "target" del evento será el elemento
desplazado.

• **dragenter:** Se activa al entrar un elemento que se está arrastrando, dentro
de un contenedor. El "target" del evento será el elemento contenedor.

• **dragleave:** El elemento arrastrado ha salido del contenedor. El "target" del evento será el elemento contenedor.

Proceso de eventos

Drag & Drop (Eventos)

•**dragover**: El elemento ha sido movido dentro del contenedor. El "target" será el contenedor. Como el comportamiento por defecto es cancelar "drops", la función debe devolver false o llamar a preventDefault para indicar que se puede soltar dentro de ese contenedor.

• **drop**: El elemento arrastrado ha sido soltado en un contenedor. El "target" del elemento será el contenedor.

• **dragend**: Se ha dejado de arrastrar el elemento, se haya dejado en un contenedor o no. El "target" del evento es el elemento arrastrado.

PROCESO DE EVENTOS

Drag & Drop

Para utilizar Drag & Drop:

• Definir un objeto como "arrastrable", estableciendo su propiedad draggable="true“(por defecto "true" en imágenes).

• Definir el comportamiento adecuado cuando se detecta un evento relacionado con Drag & Drop:

```
<body>
<div class="area" align="center">
<span>Elige un objeto para arrastrar</span><br/>
<img id="rectangulo" src="rectangulo.jpg" width="120" height="82"/>
<img id="triangulo" src="triangulo.jpg" width="145" height="168" />
</div>
<div class="area" align="center">
<span>Suelta el objeto aquí</span>
<div id="lista"></div>
</div>
</body>
```

PROCESO DE EVENTOS

Drag & Drop

```
<script>
     document.getElementById('lista').ondragover = anyadirObjecto;
     document.getElementById('rectangulo').ondragstart = empezar;
     document.getElementById('triangulo').ondragstart = empezar;
     document.getElementById('lista').ondrop = soltar;
</script>
```

```
<script>
     function empezar(e){
          e.dataTransfer.setData('Text', this.id);
          e.dataTransfer.effectAllowed = 'move';
     }
     function anyadirObjecto(e) {
          e.dataTransfer.dropEffect = 'move';
          return false;
     }
     function soltar (e) {
          imagen = new Image();
          imagen.src = e.dataTransfer.getData('Text') + '.jpg';
          document.getElementById('lista').appendChild(imagen);
     }
</script>
```

Proceso de eventos

Drag & Drop

El elemento a mover, tiene su propiedad draggable a 'true'.

El contenedor puede recibir información de los elementos arrastrados desde otros navegadores u otras aplicaciones.

Si utilizamos setData y getData del objeto dateTransfer expuesto en el objeto del evento (Event Object).

Proceso de eventos

Drag & Drop

Tipos para almacenar la Información:

• Texto: Utilizamos *text/plain.*

```
event.dataTransfer.setData("text/plain", "Este es el texto a arrastrar");
```

• Link: Utilizamos *text/plain* o *text/uri-list.*

```
event.dataTransfer.setData("text/plain", "http://
www.online.imaginaformacion.com");
event.dataTransfer.setData("text/uri-list", "http://
www.online.imaginaformacion.com");
```

• HTML/XML: Utilizamos *text/plain, text/xml* o *text/html.*

```
event.dataTransfer.setData("text/html", "Hola <strong>alumnos</
strong>");
event.dataTransfer.setData("text/plain", "Hola alumnos");
```

Proceso de eventos

Drag & Drop

Arrastrando archivos

• Un archivo local es arrastrado con el tipo *application/x-moz-file*.

• Las páginas web sin ciertos privilegios, no son capaces de recuperar o modificar datos de este tipo.

• Debido a que un archivo no es una cadena, debe utilizar el método mozSetDataAt (en firefox) para asignar los datos.

• Del mismo modo,cuando se recuperan los datos, debe utilizar el método mozGetDataAt (en firefox).

```
event.dataTransfer.mozSetData("application/x-moz-file", file, 0);
```

Proceso de eventos

Drag & Drop

Imágenes:

La mayoría de los navegadores no soportan actualmente arrastrar imágenes.

Por lo general lo que se arrastra es la url de la propia imagen.

Se Utiliza text/plain o text/uri-list. event.dataTransfer.setData("text/plain", imagenURL); event.dataTransfer.setData("text/uri-list", imagenURL);

En chrome podemos utilizar: event.dataTransfer.setData("image/png", stream, 0);

Para saber más acerca de la url de Datos, https://developer.mozilla.org/en/dataURIs

SOCKETS WEB

Definicion

- Los *WebSockets* nos ofrecen una conexión bidireccional entre el servidor y el navegador. Esta conexión se produce en tiempo real y se mantiene permanentemente abierta hasta que se cierre de manera explícita. Esto significa que cuando el servidor quiere enviar datos al servidor, el mensaje se traslada inmediatamente.

SOCKETS WEB

Abrir una conexión

Para abrir una conexión WebSocket, sólo tenemos que ejecutar el constructor WebSocket, que toma como parámetro la URL del socket a abrir. Hay que tener en cuenta que el protocolo a utilizar es ws://:

```
var socket = new WebSocket('ws://html5rocks.websocket.org/tweets');
```

Web Workers

Ejecución en segundo plano

- Los navegadores ejecutan las aplicaciones en un único *thread*, lo que significa que si JavaScript está ejecutando una tarea muy complicada, que se traduce en tiempo de procesado, el rendimiento del navegador se ve afectado.

- Los *Web workers* se introdujeron con la idea de simplificar la ejecución de *threads* en el navegador.

- Un *worker* permite crear un entorno en el que un bloque de código JavaScript puede ejecutarse de manera paralela sin afectar al *thread* principal del navegador.

- Los *Web workers* utilizan un protocolo de paso de mensajes similar a los utilizados en programación paralela.

WEB WORKERS

Ejecución en segundo plano

Estos *Web workers* se ejecutan en un subproceso aislado. Como resultado, es necesario que el código que ejecutan se encuentre en un archivo independiente.

Sin embargo, antes de hacer esto, lo primero que se tiene que hacer es crear un nuevo objeto *Worker* en la página principal:

```
var worker = new Worker('task.js');
```

WEB WORKERS

Ejecución en segundo plano

La comunicación entre un *Worker* y su página principal se realiza mediante un modelo de evento y el método postMessage().

En función del navegador o de la versión, postMessage() puede aceptar una cadena o un objeto JSON como argumento único.

Las últimas versiones de los navegadores modernos son compatibles con la transferencia de objetos JSON.

De todas maneras, siempre podemos utilizar los métodos JSON.stringify y JSON.parse para la transferencia de objetos entre el *thread* principal y los *Worker*.

WEB WORKERS

Ejecución en segundo plano

A continuación, se muestra un ejemplo sobre cómo utilizar una cadena para transferir "Hello World" a un *Worker* en doWork.js. El *Worker* simplemente devuelve el mensaje que se le transfiere.

Secuencia de comandos principal:

```
worker.postMessage('Hello World');
```

```
self.addEventListener('message', function(e) {
    self.postMessage(e.data);
}, false);
```

Cuando se ejecuta postMessage() desde la página principal, el *Worker* es capaz de obtener este mensaje escuchando al evento message.

Se puede acceder a los datos del mensaje (en este caso "Hello World") a través de la propiedad data del evento.

GEOLOCALIZACIÓN

Definición

En HTML5 podemos obtener la posición geográfca del usuario. Sin embargo dado que ésto puede comprometer la privacidad del usuario, éste debe de dar siempre su permiso para poder obtenerla.

La mayoría de los navegadores modernos soportan la geolocalización, si bien es posible que pueda haber algún fallo de disponibilidad en encontrar el sistema de localización.

La geolocalización da una situación mucho más exacta si el usuario se conecta con algún dispositivo con GPS, el cual será usado por el navegador para la localización. Caso de no haber dispositivo GPS la localización viene dada por la red de telefonía o por la configuración del ordenador.

GEOLOCALIZACIÓN

Obtener la geolocalización

- Para obtener la geolocalización, se crea un nuevo objeto javascript que depende del navegador, el objeto navigator.geolocation. Este objeto tiene a su vez una serie de métodos que nos permitirán manejar la localización.

```
function geoloc() {
d=document.getElementById("demo");
if (navigator.geolocation){
   d.innerHTML="<p>Tu dispositivo soporta la geolocalización.</p>";
   }
else {
   d.innerHTML="<p>Lo sentimos, tu dispositivo no admite la geolocaización.</p>";
   }
}
```

GEOLOCALIZACIÓN

Obtener la geolocalización

- Ahora para que esto funcione debemos poner el código en HTML que llame a la función y devuelva el resultado:

```
<div id="geo1">
  <button onclick="geoloc()">Ver geolocalización.</button><br/>
  <div id="demo"></div>
</div>
```

GEOLOCALIZACIÓN

Obtener las coordenadas

Nos falta encontrar las coordenadas del usuario. Para ello se utiliza el método getCurrentPosition(showPosition), el cual lo aplicamos sobre el elemento geolocation.

Como argumento se le pasa una funcion showPosition que tenemos que crear después.

En esta función ponemos un argumento showPosition(position), que es con el que obtenemos las coordenadas mediante las propiedades: coords.latitude y coords.longitude.

GEOLOCALIZACIÓN

Obtener las coordenadas

```
function geoloc() {
  d=document.getElementById("demo");
  if (navigator.geolocation){
    d.innerHTML="<p>Tu dispositivo soporta la geolocalización.</p>";
    navigator.geolocation.getCurrentPosition(showPosition);
  }
else {
    d.innerHTML="<p>Lo sentimos, tu dispositivo no admite la geolocaización.</p>";
  }
}
function showPosition(position){
  latitud=position.coords.latitude;
  longitud=position.coords.longitude;
  d.innerHTML+="<p>Latitud: "+latitud+"</p>";
  d.innerHTML+="<p>Longitud: "+longitud+"</p>";
}
```

GEOLOCALIZACIÓN

Control de errores

Si por algún motivo no se pueden obtener las coordenadas, podemos indicar el motivo mediante un control de errores.

Para ello en el método getCurrentPosition añadimos un segundo argumento:

navigator.geolocation.getCurrentPosition(showPosition,showError)

Con este segundo argumento crearemos una función showError(error). el argumento de esta función nos dirá cual es el error que se ha producido.

GEOLOCALIZACIÓN

Control de errores

```
function showError(error){
  switch(error.code) {
    case error.PERMISSION_DENIED:
      d.innerHTML+="<p>El usuario ha denegado el permiso a la localización.</p>"
      break;
    case error.POSITION_UNAVAILABLE:
      d.innerHTML+="<p>La información de la localización no está disponible.</p>"
      break;
    case error.TIMEOUT:
      d.innerHTML+="<p>El tiempo de espera para buscar la localización ha expirado.</p>"
      break;
    case error.UNKNOWN_ERROR:
      d.innerHTML+="<p>Ha ocurrido un error desconocido.</p>"
      break;
    }
  }
```

GEOLOCALIZACIÓN

Control de errores

En la función showError mediante el argumentoerror obtenemos el posible error que pueda producirse.

Para localizarlo utilizamos una estructura switch en la que pasamos como variable error.code. los tipos de error que podemos obtener, y que indicamos como posibles opciones son:

- PERMISION_DENIED : El usuario ha denegado el permiso para acceder a la geolocalización.
- POSITION_UNAVAILABLE : No se ha podido obtener la posición geográfica del usuario.
- TIMEOUT : El tiempo de espera para obtener los datos ha expirado.
- UNKNOWN_ERROR : Hay un error desconocido.

GEOLOCALIZACIÓN

Localización en el mapa

Para ver la localización en un mapa necesitamos tener acceso a un servicio de mapas que utilicen la latitud y la longitud. Lo más cómodo es utilizar el servicio de Google Maps, de acceso abierto y gratuito.

```
function showPosition(position) {
   lat=position.coords.latitude; //obtenemos la latitud
   lon=position.coords.longitude; //obtenemos la longitud
   latlon=lat+","+lon;  //dato latitud,longitud para pasar a Google Maps.
   //acceso a la url del mapa de google maps:
   var img_url="http://maps.googleapis.com/maps/api/staticmap?center="
      +latlon+"&zoom=14&size=400x300&sensor=false";
   //mostrar el mapa en un elemento:
   document.getElementById("mapholder").innerHTML="<img src='"+img_url+"'>";
}
```

GEOLOCALIZACIÓN

Localización en el mapa

Una vez hallada la longitud y la latidud buscamos el mapa en Google Maps y lo ponemos en nuestra página.

El código que mandamos a Google Maps es siempre el mismo.

En la variable latlon le mandamos la longitud y la latitud, separados por una coma. Sin embargo podemos variar algunos datos.

Podemos ampliar o reducir el mapa mediante el dato &zoom, a mayor número el contenido del mapa se reduce.

También podemos indicar el tamaño del mapa que queremos obtener (en pixeles) en el dato &size

GEOLOCALIZACIÓN

Localización en el mapa

```
latlon=latitud+","+longitud;
var img_url="http://maps.googleapis.com/maps/api/staticmap?center="
   +latlon+"&zoom=14&size=350x250&sensor=false";
d.innerHTML+="<img src='"+img_url+"'>";
```

GEOLOCALIZACIÓN

Obtener más información.

Mediante el argumento position de la función showPosition hemos obtenido la latidud y la longitud. Sin embargo pueden obtenerse también otros datos, siempre que éstos estén disponibles.

En la función show position(position), podemos obtener también los siguientes datos:

- position.ooords.accuracy: Precisión en la localización del mapa.
- position.coords.altitude: Altitud sobre el nivel del mar.
- position.coords.altitudeAccuracy: Precisión en el dato de altitud sobre el nivel del mar.
- position.coords.heading: Dirección en la que nos movemos. Expresada en grados respecto al norte.
- position.coords.speed: Velocidad a la que nos movemos.
- position.timestamp: Fecha y hora de la consulta (expresada en tiempo Unix).

Algunos de estos datos (como la dirección y la velocidad) sólo están disponibles en dispositivos con GPS.

ALMACENAMIENTO

Web Storage

HTML5 introduce dos mecanismos para almacenar información estructurada en el lado del cliente.

- sessionStorage

permite guardar información en el lado del cliente.

es un comportamiento similar a las variables de sesión.

- localStorage

permite guardar información sobre un sitio web.

esta información puede ser compartida entre ventanas y/o pestañas distintas. La información permanece aunque se termine la sesión.

ALMACENAMIENTO

Web Storage

A pesar de tener un comportamiento similar a las cookies, tiene varias ventajas adicionales:

- las cookies están limitadas a 4 KB de espacio, mientras que con SessionStorage se pueden guardar varios MB (depende de cada navegador).

- las cookies se envían en cada petición al servidor, lo cual aumenta la sobrecarga, mientras que la información guardada con SessionStorage no se envía automáticamente (aunque se puede configurar si es necesario).

ALMACENAMIENTO

Web Storage

Los sitios pueden agregar datos al objeto sessionStorage, y se podrá acceder desde cualquier pestaña del mismo sitio abierto en esa ventana.

```
sessionStorage.setItem('key', 'value');
//Crea una nueva variable en el objeto sessionStorage
sessionStorage.getItem('key');
//Accede a la variable 'key'
var a = sessionStorage.key; //Otra forma de acceso
sessionStorage.removeItem('key'); //Eliminar variable
```

ALMACENAMIENTO

Web Storage

```
<body>
<script>
     function guardarEstado(){
        sessionStorage.setItem('state', document.getElementById('cb').checked);
     }
     function mostrarEstado(){
          if(sessionStorage.getItem('state') != null){
            alert(sessionStorage.getItem('state'));
          }else{
               alert('Haz click sobre el checkbox para guardar la información
               sobre su estado');
          }
     }
</script>
<section>
<input type="checkbox" id="cb" onchange="guardarEstado()"/> Marca el checkbox
<br/>
<input type="button" onclick="mostrarEstado()" value="Info de SessionStorage"/>
</section>
</body>
```

ALMACENAMIENTO

Web Storage

El mecanismo LocalStorage de almacenamiento está diseñado para que el almacenamiento sea accesible desde ventanas distintas (que tengan abierto el mismo sitio web), y se prolongue más allá de la sesión actual.

- Las aplicaciones web pueden almacenar megabytes de datos.
- Las cookies no son aconsejables en este caso, ya que se transmiten con cada solicitud.
- El funcionamiento es el mismo que con sessionStorage, pero usando el objeto localStorage en su lugar.

ALMACENAMIENTO

Web Storage

```
<body>
<script>
    function guardarEstado(){
       localStorage.setItem('state', document.getElementById('cb').checked);
    }
    function mostrarEstado(){
        if(localStorage.getItem('state') != null){
          alert(localStorage.getItem('state'));
        }else{
            alert('Haz click sobre el checkbox para guardar la información
            sobre su estado');
       }
    }
</script>
<section>
<input type="checkbox" id="cb" onchange="guardarEstado()"/> Marca el checkbox
<br>
<input type="button" onclick="mostrarEstado()" value="Info de LocalStorage"/>
</section>
</body>
```