HTML 5 - Canvas

Trabajar con el Canvas

http://desarrolloweb.dlsi.ua.es/cursos/2012/nuevos-estandares-desarrollo-sitios-web/html5-canvas

http://www.w3schools.com/html/html5_canvas.asp

https://developer.mozilla.org/es/docs/Web/HTML/Canvas

Canvas Tutorial

Conozca el nuevo elemento `<canvas>` y cómo dibujar gráficos y otros objetos en Firefox

En inglés **canvas** significa ***lienzo***, y define muy bien para que sirve este elemento creado para HTML5. Es uno de los componentes, de hecho, de HTML5 más famosos por el gran aporte que ha supuesto al dinamismo de las páginas web.
Mediante este elemento dispondremos de un área que podremos utilizar donde queramos para dibujar elementos gráficos mediante lenguaje JavaScript. Eso ha permitido (gracias a la potencia de JavaScript crear juegos, animaciones y elementos visuales atractivos en las páginas web).

atributos de canvas
id. Es el atributo que utilizan todos los elementos HTML para identificarles. En el caso de canvas es casi obligatorio su uso para poder hacer referencia al mismo.

width. Anchura del lienzo (funciona igual que en el caso de **img**)

height. Altura del lienzo.

```
<canvas id="lienzo1" width="600" height="400" />
<script type="text/javascript">
var canvas=document.getElementById("lienzo1");
 var contexto=canvas.getContext("2d");
 contexto.lineWidth=2;
 for(i=0;i<=600;i+=20){
      contexto.moveTo(0,0);
      contexto.lineTo(i,400);
      contexto.stroke();
       }
 </script>
```

Empleando Javascript podremos pintar y dibujar, crear objetos vectoriales, añadir imágenes y textos y capturar eventos.

```
<canvas id="tutorial" width="150" height="150"></canvas>
```

Con esta etiqueta html, generaremos un objeto de dibujo cuadrado de 150 píxeles de ancho. Como todos los elementos (objetos) html, lo podemos decorar con estilos. Así, le definiremos un border de 1 píxel de color gris con:

```
<canvas id="tutorial" width="150" height="150" style="border:1px solid grey"></canvas>
```

Para empezar a dibujar, primero deberemos obtener una referencia al objeto canvas, document.getElementById('tutorial'), para después obtener una referencia al contexto de dibujo en 2D, getContext('2d'). También existe un método experimental que nos ofrece un acceso a dibujo en 3D (WebGL), mediante getContext('3d').

```
<script>
window.onload = function() {
var canvas = document.getElementById('tutorial');
var ctx = canvas.getContext('2d');
ctx.fillStyle = "rgb(200,0,0)";
ctx.fillRect (10, 10, 50, 50);
}
</script>
```

A partir de aquí, ya podemos empezar a emplear métodos del objeto canvas.
Con la solicitud ***fillStyle***, definiremos el tipo de estilo de contenido, y con ***fillRect***, generaremos un cuadrado en las coordenadas 10,10, y con 50 píxeles de anchura y altura, resultando:

1) Sistema de coordenadas

La definición de coordenadas en canvas funciona de arriba abajo y de izquierda a derecha. Así, la coordenada 10,50 indica un punto en la columna 10 y en la fila 50.

2) Cuadrados

A diferencia de otras tecnologías, **canvas** sólo admite un tipo primitivo de forma: los cuadrados. Pero también dispone de un potente sistema para generar líneas y recorridos que pueden ser abiertos o cerrados.
Las instrucciones para generar cuadrados son:
ctx.fillRect(x,y,width,height): Dibuja un rectángulo lleno de color.
ctx.strokeRect(x,y,width,height): Dibuja un rectángulo a línea.
ctx.clearRect(x,y,width,height): Limpia y vuelve transparente el área especificada.
Las tres funciones reciben los mismos parámetros: (***x,y***) para el origen, y (***width, height***) para la anchura y altura de la caja.

3) Líneas y caminos

El modo de dibujar líneas es diferente al de dibujar cuadrados. De hecho, funciona como si del lápiz se tratara. Primero la posicionamos y la acercamos al papel, y vamos moviéndola mientras queramos que dibuje.

```
ctx.beginPath(); // Inicia la operación de dibujo.
ctx.moveTo(40, 40); // Coloca el puntero a 40, 40.
ctx.lineTo(340, 40); // Dibuja una línea hasta 340, 40.
ctx.closePath(); // Cierra el path.
ctx.stroke(); // Lo transforma en dibujo a línea.
```

4) Círculos

No existe un método específico para dibujar círculos con **canvas**, pero sí que tenemos las herramientas para dibujarlos. De este modo, podemos utilizar el método arco:

```
ctx.beginPath(); // Iniciamos el dibujo de un camino.
ctx.arc(100, 90, 50, 0, Math.PI*2, false); // Dibujamos un círculo.
ctx.closePath(); // Cerramos el camino.
ctx.fill(); // Llenamos de contenido.
```

El método arco admite seis parámetros. Los dos primeros son el punto central de la circunferencia; el segundo, el radio; el tercero, el ángulo inicial; el cuarto, el ángulo final (**los ángulos en radianes**) y, finalmente, un booleano para marcar si se debe dibujar en la dirección de las agujas del reloj o no.
Para poder convertir los grados en radianes, podemos emplear la siguiente fórmula:

```
var degrees = 1; // 1 grado
var radians = degrees * (Math.PI / 180); // 0,0175 radianes
```

360 grados
360 grados, son ***2*Math.PI***.

5) Estilos de línea (*stroke*) y de relleno (*fill*)
Otra de las herramientas fáciles es la posibilidad de personalizar los estilos de las líneas y rellenos que dibujamos, así:

```
ctx.fillStyle = "rgb(255, 0, 0)"; // Definiremos un color de relleno rojo.
ctx.strokeStyle = "rgb(255, 0, 0)"; // Definiremos un color de línea.
ctx.lineWidth = 20; // Define el grueso de la línea.
```

6) Texto
Disponemos de un método que nos permite escribir con el canvas:

```
var text = "Hello, World!";
ctx.font = "italic 20px serif";
ctx.fillText(text, 30, 80);
```

7) Añadiendo imágenes
Podemos añadir imágenes directamente a un objeto canvas. Para hacerlo:

```
var img = new Image();
img.src = "pathalaimatge.gif"
ctx.drawImage(img,0,0);
```

Al ser la imagen un objeto remoto, convendría esperar su descarga; así podríamos, utilizando jQuery, programar un pequeño preload:

```
var image = new Image();

image.src = "prueba.jpg";
$(image).load(function() {
ctx.drawImage(image, 0, 0);
});
```

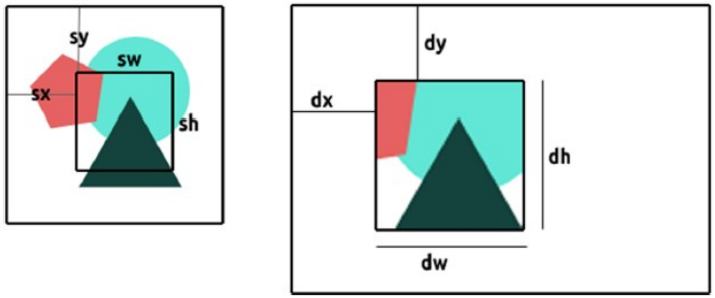
Primero creamos un objeto tradicional imagen con Javascript para utilizarlo como receptor de la descarga. Mediante el image.src, inicializamos la descarga. Finalmente, con la función $(image).load programamos el evento onLoad en la imagen, que nos la dibujará (colocará) en el objeto canvas. Podemos **escalar las imágenes** directamente cuando las dibujemos sobre el **canvas**, utilizando la función ***drawImage()*** de la siguiente manera:

```
ctx.drawImage(image, x, y, width, height);
```

Donde, ***width*** y ***height*** serán el ancho y alto respectivos que queremos utilizar. También podemos **recortar imágenes** directamente al insertarlas en el **canvas** con:

```
ctx.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh);
```

Donde **sx, sy, sw, sh** marcan el reencuadre de la imagen original; **dx**, **dy,** la posición de este en el canvas, y **dw**, **dh**, el tamaño.



Sobre las imágenes podemos aplicar transformaciones como la rotación, la escala y el origen. Así, en el siguiente ejemplo:

```
ctx.translate(20, 20);
ctx.rotate(0.7854); // Rotate 45 degrees
var image = new Image();
image.src = "example.jpg";
$(image).load(function() {
ctx.drawImage(image, 0, 0, 50, 50, -10, -10, 30, 30);
});
```

Trasladaremos el eje de rotación a **250,250** y después aplicaremos una rotación en radianes de **0,7854**. A partir de aquí, a los objetos que dibujemos se les aplicarán estos cambios antes de dibujarlos.

Otra característica muy interesante del objeto canvas y las imágenes es la posibilidad de acceder directamente a los píxeles que configuran una imagen. Así, tenemos acceso a una matriz de los colores que conforman cada píxel y los podemos modificar. Con esto, se abre la puerta a la posibilidad de confeccionar con Javascript multitud de efectos sobre imágenes directamente desde el código. Para tener acceso a los píxeles, utilizaremos la función:

```
a = ctx.getImageData(x, y, width, height);
```

Esta nos devolverá un objeto con tres propiedades, el ***width***, el ***height*** y una ***data*** lista de píxeles del tipo ***CanvasPixelArray***. Una lista con cada cuatro posiciones representan un color en la forma RGBA.

```
a.data[0]; // componente red
a.data[1]; // componente green
a.data[2]; // componente blue
a.data[3]; // componente alpha
```

El componente ***a[5] a a[8]*** será el siguiente píxel. Cabe decir también que en este array no hay filas ni columnas, y por tanto, las deberemos deducir nosotros a partir del ***width*** de imagen.
De una manera sencilla, podemos confeccionar el típico efecto de ruido generando una imagen desde 0 que escribiremos en el objeto canvas con:

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript">
window.onload = function() {
var canvas=document.getElementById('ejercicio');
var pt=canvas.getContext('2d');
setInterval(function() {
// generamos un array de píxeles aleatorios
var image_data = dibuja_ruido( pt );
// la pintamos al objeto canvas
pt.putImageData( image_data, 0, 0);
}, 200);
}
dibuja_ruido = function(pt) {
var imageData = pt.createImageData(150, 150);

var pixels = imageData.data;
var numPixels = imageData.width*imageData.height;
// Para cada píxel le generamos un color aleatorio
for (var i=0; i<numPixels; i++) {
pixels[i*4] = Math.floor(Math.random()*255); // Red
pixels[i*4+1] = Math.floor(Math.random()*255); // Green
pixels[i*4+2] = Math.floor(Math.random()*255); // Blue
pixels[i*4+3] = 255; // Alpha
};
return imageData;
}
</script>
</head>
<body>
<canvas width="150" height="150" style="border:1px solid grey"
id="ejercicio"> </canvas>
</body>
</html>
```

Transformaciones

1) Transformaciones y escalas

Podemos transformar el modo como dibujamos un objeto, o ciertos objetos, empleando los métodos translate y rotate. Con el primero, transportamos las coordenadas 0,0 a otro punto. Así, si efectuamos:

```
ctx.translate(100,100);
```

La coordenada 0,0 pasará a ser la 100,100, y si nosotros dibujásemos un cuadrado en la 0,0,10,10, nos aparecería en la 100,100,110,110.

Lo mismo sucede con la rotación. Si nosotros realizamos:

```
ctx.rotate(Math.PI/4)
```

Rotaremos el canvas 45°, y todo lo que dibujemos nos aparecerá rotado.

2) Compositing

Componer elementos se refiere a la manera como se combinarán los nuevos elementos que añadamos al canvas con los que ya existen. Así, cuando dibujemos un objeto en el canvas, podremos decidir cómo queremos que este se superponga con el resto de los objetos existentes.

De este modo, cuando añadamos un nuevo objeto podremos decidir si queremos que se superponga o lo contrario. Y también si queremos que al superponerse se aplique un canal ***alpha*** o queremos que se haga un ***xor*** de píxeles. Así podemos definir el modelo de composición con:

```
ctx.globalCompositeOperation = "";
```

Donde el valor puede ser:

source-over La imagen origen se situará sobre el canvas.
destination-over La imagen se situará debajo de los objetos existentes en el canvas.
source-in El origen se dibujará en la zona donde existe superposición.
destination-in El destino se dibujará en la zona donde existe superposición.
lighter, copy, xor *Lighter* genera la superposición como color 255, blanco.
Copy dibuja el origen en vez del destino.
Cualquier parte que se superponga quedará transparente.

3) Sombras

También podemos definir y pintar sombras en un objeto mientras lo definimos. Para hacerlo, es necesario que definamos las propiedades adecuadas para que después, al generar el objeto, nos aparezcan correctamente:

```
ctx.shadowBlur = 20;
ctx.shadowColor = "rgb(0, 0, 0)";
ctx.shadowOffsetX = 10;
ctx.shadowOffsetY = 10;
ctx.fillRect(50, 50, 100, 100);
```

En este caso, estamos generando una sombra de color negro con un desenfocado de 20px y desplazada 10px.

4) Guardar el estado actual como imagen

El **canvas**, a efectos prácticos, es como si tuviéramos una gran imagen y, como tal, la podemos guardar como imagen con el siguiente comando:

```
var dataURL = canvas.get(0).toDataURL();
```

que nos generará una imagen en formato png y codificada en base64. Al mismo tiempo, este *string*, que podemos enviar a un servidor y descodificar como imagen, también puede ser asignado a un objeto imagen para entonces abrirlo directamente con el navegador:

```
var img = $("<img></img>"); img.attr("src", dataURL);
canvas.replaceWith(img);
```

De este modo, si el usuario quiere guardar la imagen, simplemente haciendo clic con el botón derecho del ratón lo puede hacer, puesto que hemos convertido el objeto canvas (vectorial) en imágenes.
También podríamos generar un archivo descargable enviándola primero al servidor, realizando la conversión a binario allí, y lanzando las cabeceras adecuadas junto con los bytes de la imagen.

Animación

El objeto canvas no nos proporciona un mecanismo para generar animaciones de manera directa, pero sí que lo podemos hacer gracias al Javascript. El principio con pseudo-código es:

1. dibujaremos.
2. esperaremos un intervalo de tiempo.
3. borraremos la pantalla.
4. actualizaremos los valores.

Volveremos al paso 1.

De este modo, podemos mostrar un pequeño ejemplo que demuestre el modo como podemos animar cosas:

```
<html>
<head>
<title>Ejercicio Movimiento canvas</title>
<script type="text/javascript">
window.onload = function()
{
var canvas = document.getElementById('ejercicio');
var pt = canvas.getContext('2d');
pt.fillStyle = "rgb(200,0,0)"; // formato del dibujo
var inc = 1; // velocidad del movimiento
// objeto que movemos...
var o = { x: 10, y:10, width:10, height:10 };
// ejecución periódica
setInterval(function() {
// si el objeto sobrepasara los límites del canvas,
// cambiamos la orientación de este
if(o.x >= 140 || o.x <=0)
inc = -inc
o.x += inc
limpia(pt)
dibuja( o, pt);
}, 1);
}
dibuja = function(obj, pt)
{
// dibuja el objeto
pt.fillRect (obj.x, obj.y, obj.width, obj.height);
}
limpia = function(pt)
{
// limpia la pantalla y la prepara para el siguiente paso
pt.clearRect(0,0,150,150)
}
</script>
</head>
<body>
<canvas width="150" height="150" style="border:1px solid grey"
id="ejercicio"> </canvas>
</body>
</html>
```

Si nos fijamos, con el código lo que hacemos es generar una ejecución periódica con ***setInterval***, que nos servirá para ir llamando a nuestro programa.
Primero moveremos el objeto; después, limpiaremos la pantalla y, finalmente, lo dibujaremos de nuevo.