Usar imágenes

Importación de imágenes

Vamos a ver el paso uno primero. Básicamente, existen cuatro opciones disponibles:

Usar imágenes que están en la misma página

Podemos acceder a todas las imágenes de una página utilizando la colección `document.images`, el método`document.getElementsByTagName` , o si sabemos el atributo ID de la imagen, el método `document.getElementById`.

Usar otros elementos canvas

Al igual que con las imágenes normales, podemos tener acceso a otros elementos de canvas utilizando el método`document.getElementsByTagName` o el método `document.getElementById`. Asegúrate de que has dibujado algo en el lienzo de origen antes de utilizarlo en el lienzo de destino. Uno de los usos más prácticos de esto sería utilizar un segundo elemento canvas como una vista en miniatura del otro lienzo más grande.

Creación de una imagen desde cero

Otra opción es crear nuevos objetos `Image` en nuestro script.
Básicamente para crear una nuevo objeto de imagen, hacemos lo siguiente:

```
var img = new Image (); / / Crear un nuevo objeto de imagen (Image)
img.src = 'myImage.png'; / / Establecer la ruta de origen
```

Cuando este script se ejecuta, la imagen comienza a cargar. Si intentas llamar a `drawImage` antes de que la imagen haya terminado de cargar, se iniciará Gecko 1.9.2 y versiones anteriores mientras que, en silencio, en Gecko 2.0 y posteriores, no hará nada. Así que debes utilizar un controlador de eventos `onload`:

```
var img = new Image (); / / Crear un nuevo objeto de imagen (Image)
img.onload = function () {
  / / Ejecutar instrucciones drawImage aquí
}
img.src = 'myImage.png'; / / Establecer la ruta de origen
```

Si sólo estás utilizando una imagen externa, éste puede ser un buen método, pero cuando tengas que realizar un seguimiento a más de una, tendremos que recurrir a algo más astuto. Está más

allá del alcance de este tutorial abordar las tácticas de precarga de imagen, pero puedes echar un vistazo a JavaScript Image Preloader para obtener una solución completa.

Incrustación de una imagen a través de data: url

Otra forma posible de incluir imágenes es a través de las data: URL . Las URL de datos permiten definir completamente una imagen como una cadena codificada Base64 de caracteres directamente en el código. Una de las ventajas de las URL de datos es que la imagen resultante está disponible de inmediato, sin necesidad de ir y volver al servidor. (Otra ventaja es que es posible encapsular en un único archivo todos los CSS, Javascript, HTML, e imágenes, por lo que resulta más fácil moverla a otras ubicaciones.) Algunas de las desventajas de este método es que la imagen no se almacena en caché y para imágenes más grandes la URL codificada puede llegar a ser muy larga:

```
var img_src =
'
4lmNVindo7qyrIXiGBYAOw==';
```

drawImage

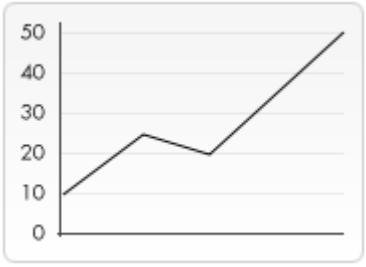
Una vez que tengamos una referencia a nuestro objeto de imagen de origen, podemos utilizar el método `drawImage` para representarla en el lienzo. Como veremos más tarde, el método `drawImage` está sobrecargado y tiene tres variantes diferentes. En su forma más básica tiene este aspecto.

```
drawImage(image, x, y)
```

Donde `image` es una referencia a nuestra imagen u objeto canvas. `x` e `y` forman las coordenadas en el lienzo de destino donde se colocará nuestra imagen.

Ejemplo 1 de `drawImage`

En el siguiente ejemplo voy a utilizar una imagen externa como fondo de un pequeño gráfico de líneas. El uso de fondo puede hacer que tu script sea considerablemente menor porque no tenemos que dibujar un fondo con demasiados detalles. Como sólo estoy usando una imagen aquí, utilizo el controlador de evento `onload` del objeto de imagen para ejecutar las instrucciones de dibujo. El método `drawImage` coloca el fondo en la coordenada (0,0), que es la esquina superior izquierda del lienzo.



```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    var img = new Image();
    img.src = 'backdrop.png';
    img.onload = function(){
      ctx.drawImage(img, 0, 0);
      ctx.beginPath();
      ctx.moveTo(30, 96);
      ctx.lineTo(70, 66);
      ctx.lineTo(103, 76);
      ctx.lineTo(170, 15);
      ctx.stroke();
    }
  }
```

Escala

La segunda variante del método `drawImage` añade dos nuevos parámetros y nos permite colocar imágenes a escala en el lienzo.

```
drawImage(image, x, y, width, height)
```

Donde `width` y `height` (ancho y alto, respectivamente) es el tamaño de la imagen en el lienzo de destino.

Ejemplo 2 de `drawImage`

En este ejemplo voy a utilizar una imagen como fondo de pantalla y repetirla varias veces en el lienzo. Esto se hace simplemente por medio de bucles y de la colocación de las imágenes a escala en diferentes posiciones.En el código inferior el primer `for` recorre las filas y el segundo `for` recorre las columnas. La imagen se escala a un tercio de su tamaño original, que es 50x38 píxeles.

Nota: Las imágenes pueden resultar borrosas cuando se aumenta la escala o granuladas si se han reducido demasiado. Probablemente lo mejor es que no apliques la escala si tienes texto que debe permanecer legible.

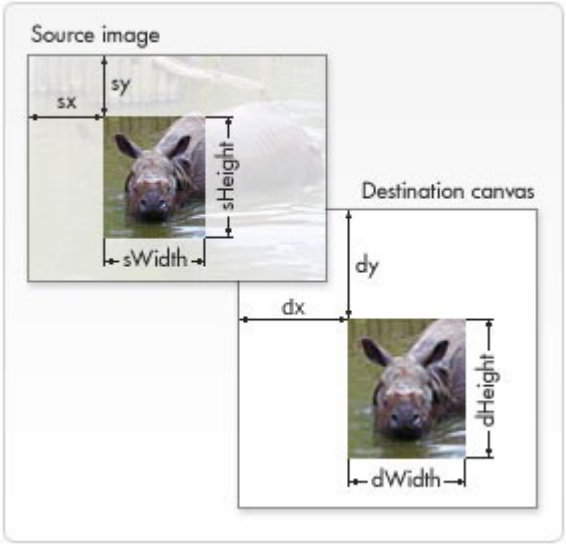
```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    var img = new Image();
    img.src = 'images/rhino.jpg';
    img.onload = function(){
      for (i = 0; i < 4; i++){
        for (j = 0; j < 3; j++){
          ctx.drawImage(img, j*50, i*38, 50, 38);
        }
      }
    }
  }
```

Segmentar

La tercera y última variante del método `drawImage` cuenta con ocho nuevos parámetros. Podemos utilizar este método para segmentar partes de una imagen de origen y dibujarlas en el lienzo.

```
drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)
```

El primer parámetro `image`, al igual que las otras variantes, es o bien una referencia a un objeto de imagen o una referencia a un elemento canvas diferente. Para los otros ocho parámetros, lo mejor es mirar la imagen de la derecha. Los primeros cuatro parámetros definen la ubicación y el tamaño del segmento de la imagen de origen. Los últimos cuatro parámetros definen la posición y tamaño en el lienzo de destino.



La segmentación puede ser una herramienta útil cuando se desea hacer composiciones.Puedes tener todos los elementos en un solo archivo de imagen y utilizar este método para componer un dibujo completo. Por ejemplo, si quieres hacer un gráfico, puedes tener una imagen PNG que contenga todo el texto necesario en un solo archivo y, en función de los datos, puedes cambiar la escala del gráfico sin mucha dificultad. Otra ventaja es que no es necesario cargar cada imagen por separado.

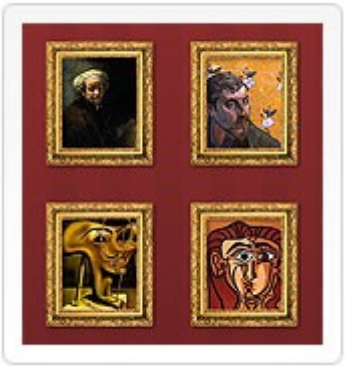
Ejemplo 3 de `drawImage`

En este ejemplo voy a utilizar el mismo rinoceronte que hemos visto anteriormente, pero ahora voy a segmentar su cabeza y componer un marco de imagen. La imagen del marco de imagen incluye una sombra que se ha guardado como una imagen PNG de 24 bits. Como las imágenes PNG de 24 bits incluyen un completo canal alfa de 8 bits, a diferencia de las imágenes GIF y PNG de 8 bits, puedo colocarla sobre cualquier fondo sin tener que preocuparme por un color mate.

He adoptado un enfoque diferente acerca de la carga de las imágenes respecto al ejemplo anterior.Simplemente coloqué las imágenes directamente en mi documento HTML y usé una regla CSS para ocultarlos de la vista (`display:none`). He asignado a las dos imágenes un atributo `id` para que sean más fáciles de seleccionar. El propio script es muy simple. En primer lugar, dibujo la imagen segmentada y a escala en el lienzo (primera instrucción `drawImage`) y luego coloco el marco en la parte superior (segunda instrucción `drawImage`).

```
function draw() {
  var ctx = document.getElementById('canvas').getContext('2d');
  ctx.drawImage(document.getElementById('source'),33,71,104,124,21,20,87,104);
  ctx.drawImage(document.getElementById('frame'),0,0);
}
```

Ejemplo de la galería de arte



En el último ejemplo de este capítulo he hecho una pequeña galería de arte. La galería consta de una tabla que contiene varias imágenes. Cuando se carga la página, para cada imagen de la página se inserta un elemento canvas y se dibuja un marco alrededor.
En mi caso, todas las imágenes tienen una anchura y altura fijas y lo mismo ocurre con el marco que he dibujado a su alrededor. Podrías mejorar el script de forma que use el ancho y el alto de la imagen para hacer que el marco se ajuste perfectamente.
No hace falta explicar el código de abajo. Recorremos la matriz de las imágenes y añadimos nuevos elementos canvas según corresponda. Probablemente lo único que debamos subrayar, para aquellos que no estén tan familiarizados con el DOM, es el uso del método insertBefore . `insertBefore` es un método del nodo principal (una celda de tabla) del elemento (la imagen) delante de la cual queremos insertar nuestro nuevo nodo (el elemento canvas).

```
function draw() {

  // Recorrer todas las imágenes
  for (var i=0;i<document.images.length;i++){

    // No añadir un lienzo para la imagen del marco
    if (document.images[i].getAttribute('id')!='frame'){

      // Crear un elemento canvas
      canvas = document.createElement('CANVAS');
      canvas.setAttribute('width',132);
      canvas.setAttribute('height',150);

      // Insertar antes de la imagen
      document.images[i].parentNode.insertBefore(canvas,document.images[i]);

      ctx = canvas.getContext('2d');

      // Dibujar una imagen en lienzo
      ctx.drawImage(document.images[i],15,20);

      // Añadir marco
      ctx.drawImage(document.getElementById('frame'),0,0);
    }}}
```