CSS3 y Javascript avanzado

Jordi Collell Puig

PID_00176160

UOC
Universitat Oberta de Catalunya

Índice

1. CSS, especificación tercera

1.1. Introducción al CSS

El CSS es un lenguaje de estilos empleado para definir la presentación, el formato y la apariencia de un documento de marcaje, sea html, xml, o cualquier otro. Comúnmente se emplea para dar formato visual a documentos html o xhtml que funcionan como espacios web. También puede ser empleado en formatos xml, u otros tipos de documentos de marcaje para la posterior generación de documentos.

Las hojas de estilos nacen de la necesidad de diseñar la información de tal manera que podemos separar el contenido de la presentación y, así, por una misma fuente de información, generalmente definida mediante un lenguaje de marcaje, ofrecer diferentes presentaciones en función de dispositivos, servicios, contextos o aplicativos. Por lo que un mismo documento html, mediante diferentes hojas de estilo, puede ser presentado por pantalla, por impresora, por lectores de voz o por tabletas braille. Separamos el contenido de la forma, composición, colores y fuentes.

La especificación del CSS la mantiene el World Wide Web Consortium (W3C http://www.w3c.org).

1.2. La especificación tercera (CSS3)

Las diferentes revisiones de las hojas de estilo tienen el origen en la **primera especificación** publicada por el W3C en diciembre de 1996 y que pretendía unificar la sintaxis y el modo de definir una hoja de estilos por los diferentes lenguajes derivados del SGML. Así, las principales características que se pueden definir mediante esta primera especificación son:

1) Propiedades del tipo de letra, así como el estilo de este.

2) Colores de los textos y de los fondos.

3) Atributos del texto tales como espacio entre caracteres, palabras y líneas.

4) Alineación de tablas, bloques de texto, imágenes, párrafos.

5) Márgenes externos e internos, filetes y posición de la mayoría de los elementos.

6) Definición única de elementos mediante ***ids*** y agrupamiento de atributos mediante ***classes***.

A partir de esta primera especificación, en mayo de 1998, y como una extensión a la primera revisión, aparece la segunda, popularmente conocida como la especificación **CSS2**, y adoptada por la mayoría de los navegadores. Como características fundamentales que añade la revisión, se encuentra la posibilidad de definir posiciones de forma absoluta, relativa y fija, así como la profundidad de los elementos (***z-index***) cuando existen superposiciones, también el soporte para formatos de voz y textos bidireccionales. De esta, aparece una revisión, la 2.1, que incorpora muchas de las mejoras hechas por los fabricantes de navegadores y que actualmente es empleada y estandarizada.

La tercera revisión de la especificación del **CSS** por el W3C empieza en el 2005 y todavía está en proceso de definición. Pero esta vez, las diferentes implementaciones de los motores de *render* de los navegadores no están esperando a tener una especificación, sino que implementan ciertas cosas a su manera y, por lo tanto, muchas son utilizables en entornos de producción web. Es necesario, sin embargo, partir siempre de la consideración de que las diferentes implementaciones de los navegadores no es exacta, y de que, por lo tanto, cuando diseñamos una hoja de estilos, el principio que tiene que regir es que funcione en todas partes, y no que funcione igual.

A diferencia de las otras especificaciones, esta vez se ha dividido en temas. De este modo disponemos de diferentes temas que pueden crecer y evolucionar en paralelo, y no como uno grande y monolítico, con muchísimas revisiones (cada vez hay más gente interesada y evoluciona más deprisa).

Así, en la nueva revisión encontramos grupos como el de selectores, unidades de medida, modelo de caja, colores y gamas, modelo de línea, texto, fuentes, lenguajes verticales, *page-media*... entre otros. Y de este modo, diferentes módulos tienen un estatus diferente y son adoptados por los fabricantes de software a diferente velocidad.

1.2.1. CSS en la web

Lejos de los usos más abstractos, las hojas de estilos han resultado la herramienta para dar formato y color a los contenidos de la WWW. Así, cualquier documento html es formateado con estilos CSS. La principal característica de la web semántica es esta separación de contenidos y visualización, donde el contenido tiene sentido por él mismo, y la visualización se adapta a cada dis-

positivo y medio. De este modo y siendo la herramienta con la que damos forma y color al contenido, los fabricantes de software (navegadores) han pasado a ser los implementadores de las funcionalidades especificadas por el W3C.

A grandes rasgos, todos implementan la especificación, pero en las excepciones hay matices y cada navegador tiene sus características a la hora de dibujar (***render***) el contenido con la hoja de estilos.

1.3. Familias de navegadores

Hablamos de familias de navegadores para especificar las diferentes tecnologías de dibujo que contienen, y así poder agrupar las distintas versiones de navegadores de una manera sencilla para el desarrollador. Así, podemos destacar cuatro grandes familias:

1) Basados en el **motor de dibujo de Internet Explorer**. Fundamentalmente navegadores en el sistema operativo Windows, en las diferentes versiones, 6, 7, 8, 9. Y muchos de los empotrados en programas como Microsoft Office y otros.

2) Familia Geko, que utilizan el **motor de dibujo Geko**, desarrollado por Mozilla. El más popular es el Firefox, pero existen múltiples implementaciones en diferentes dispositivos.

3) Familia Webkit. Basados en el **motor de dibujo Opensource Webkit**, desarrollado por el Konkeror, y actualmente mejorado por el Safari de Apple y el Google Chrome. También es el motor de dibujo de las dos plataformas de dispositivos móviles más extendidos, iOS (iPhone/iPad) y Android.

4) Familia de navegadores basados en **Opera**.

Cabe resaltar que sólo hablamos de la **tecnología de dibujo**, o sea, la parte del software que lee el documento ***sgml***, le aplica los estilos que encuentra en la hoja de estilo y lo dibuja en la pantalla (o cualquiera otra salida). Además de esto, un navegador (cliente web) también contiene un intérprete de Javascript que nos permite ejecutar secuencias de pedidos e interactuar con el **DOM** (***document object model***), el conjunto de sgml con los estilos aplicados.

Esta aclaración la hacemos porque a partir de aquí, cuando hablamos de las diferentes novedades y técnicas disponibles con la revisión tercera, esclareceremos en qué motores de dibujo funciona, para poder extraer las propiedades más comunes que podemos utilizar a día de hoy en el proceso de confección de espacios web.

1.4. Beneficios del uso del CSS3

1) Reducción del tiempo de desarrollo y mantenimiento

Utilizar propiedades y métodos de CSS3 puede ser un beneficio directo a la hora de desarrollar, puesto que nos ahorramos bastante trabajo, como por ejemplo a la hora de hacer fondo con esquinas redondeadas. Antes había que hacerlo con imágenes. También ahorramos mucho trabajo a la hora de hacer sombras, ya que nos ahorramos de nuevo la imagen que teníamos que usar antes (normalmente un gráfico en formato png).

También podemos mejorar el rendimiento al tener menos código, divs dentro de divs, etc.

2) Incrementar el rendimiento de las páginas

Menos etiquetas html indican menos código a la hora de descargarse del servidor y menos código a la hora de interpretar y dibujar el navegador. Dos ahorros, uno de ancho de banda y el otro de rendimiento del ordenador. Además, muchas de las técnicas de CSS3 nos ahorran imágenes, que a la vez cumplen la doble premisa de rendimiento.

1.5. La mejora progresiva

Uno de los elementos clave a la hora de emplear CSS es utilizar una técnica de desarrollo llamada mejora progresiva, y que consiste en empezar por generar un código genérico que funcione en todos los navegadores, para, poco a poco, ir introduciendo mejoras para navegadores más modernos. Esto lo permite, ya que los intérpretes de CSS de los navegadores ignoran una propiedad si no la conocen.

Empleando esta técnica logramos un control total óptimo del aspecto, puesto que a mejores prestaciones del navegador, mejor visualización.

2. Novedades CSS3

2.1. Nuevos selectores

Los selectores CSS son la herramienta más potente del lenguaje de estilos, puesto que nos permiten seleccionar diferentes elementos del contenido html en función de la etiqueta o de sus atributos sin tener que hacer uso de su clase, su *ID* o Javascript.

2.1.1. Selectores de atributos

- **[att^="valor"]**

Selecciona elementos con un atributo que empieza por valor.

- **[att$="valor"]**

Selecciona elementos con un atributo que acaba con valor.

- **[att*="valor"]**

Selecciona elementos que contienen un atributo que contiene valor.

Estos selectores pueden ser utilizados por todas las hojas de estilos destinadas a navegadores, Internet Explorer 7 o superior, Opera, Webkit y navegadores basados en Geko.

Ejemplo:

```
<style>
   a[title$="sample"] {
      color:#ea1d1d;
   }
</style>
<p>Lorem ipsum <a href="#" title="this is a sample">dolor sit amet</a>, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et dolore <a href="#">magna aliqua</a>. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse <a href="#" title="this is a sample">
cillum dolore</a> eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.</p>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

En el ejemplo, utilizando el selector de atributos acabado en $=, marcamos los enlaces que contienen el atributo title acabado con "sample" de color rojo.

2.1.2. Combinadores (no soportado por IE6)

- **~ Sibling General (Hermano)**

Selecciona los elementos que son "hermanos" de uno. Una etiqueta hermana es la que existe en el mismo nivel, o que tiene un padre en común y que está a continuación de la referenciada. Así, en el ejemplo siguiente podemos decir que son etiquetas hermanas los p y los h2, puesto que todos tienen el mismo padre (div). Pero sólo se seleccionarán los elementos p posteriores a la definición del h2.

Ejemplo:

```
<style>
h2~p { color:green; }
</style>
<div id="uno">
  <p>Este elemento no será visible porque es hermano, pero es anterior.</p>
  < h2>This is a test</h2>
  <p>Este p es hermano de h2</p>
  < p>Este otro p también es hermano de h2</p>
< /div>
<div id="dos">
   <p>Este p no se pintará</p>
< /div>
```

Pintará de verde los p del div id=uno posteriores a h2, puesto que son hermanos del h2. Tienen de padre en común el #uno, mientras que el p del div id=dos no se pintará puesto que no tiene ningún hermano h2.

2.1.3. Pseudo-clases

Las pseudo-clases son uno de los añadidos más extendidos en uso del CSS3. Las más útiles.

- **:nth-child(n)**

Selecciona elementos basándose en la posición de los hijos. Puede utilizar números expresiones y las palabras *odd*, *even* (impar, par). Fácilmente podemos hacer el típico efecto cebra en las tablas.

Por ejemplo, dada una lista <li>:

```
le:nth-child(2) { color: red; }
Pintará de rojo el segundo elemento de la lista.
le:nth-child(even) { color: red; }
Pintará de rojo los elementos pares.
le:nth-child(n+4) { color: red; }
Pintará de rojo todos los elementos a partir del cuarto.
le:nth-child(3n) { color: red; }
Pintará de rojo cada tres elementos.
le:nth-child(2n+5) { color: red; }
Pintará de rojo cada dos a partir del 5.
```

- **:nth-last-child(n)**

Sigue la misma idea que la anterior, pero selecciona a partir del último elemento.

- **:last-child**

Selecciona el último elemento de una lista.

- **:checked**

Selecciona elementos que están marcados, tipos, *checkboxes*.

- **:empty**

Selecciona elementos que están vacíos.

- **:not**

Selecciona elementos que no cumplen la declaración especificada.

```
p:not([class*="lead"]) { color: black; }
Marcará todos los p que no tengan una clase asignada que contenga la cadena lead.
```

Los navegadores basados en Webkit y Opera admiten todos los pseudo-selectores CSS3, los basados en Firefox 2 y 3 sólo admiten, el *not, last-child only-child, root, empty, target, checked, enabled* y *disabled*, pero el Firefox 4 admite todos los pseudo-selectores. Los navegadores Microsoft no admiten pseudo-selectores.

2.1.4. Pseudo-elementos

Los pseudo-elementos nos permiten seleccionar partes de contenido dentro de una etiqueta. Por ejemplo, podemos seleccionar la primera línea empleando el pseudo-elemento :first-line, o la primera letra, empleando el pseudo-elemento :first-letter. Así:

```
P:first-letter { fuente-size: 35px }
```

Nos cambiará la primera letra. Una de las novedades muy interesantes del CSS3 es el pseudo-elemento :selection, aplicado a los campos de formulario, que nos permite introducir modificaciones en el momento en EL que un campo de formulario tiene el foco.

```
input:selection { background-color:#eaeaea }
```

Cambiará el color del campo de formulario cuando este tenga el foco y esté seleccionado.

2.2. Colores RGBA y opacidad

2.2.1. RGBA

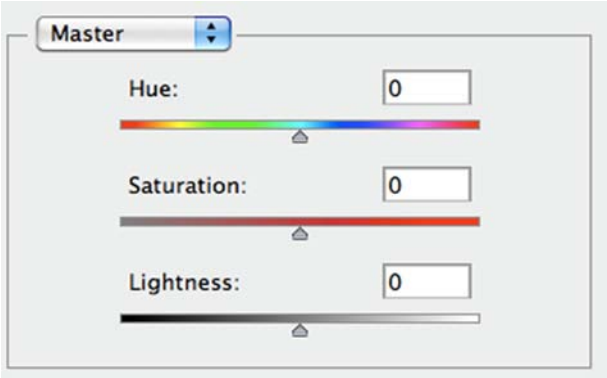
La posibilidad de especificar colores en modo RGB ya existía desde la especificación 2, pero en la versión 3 se ve ampliada con un nuevo modelo de color, el basado en saturación (HSL), y la posibilidad de especificar **canal alpha** sobre los colores. De este modo podemos crear expresiones CSS del tipo:

```
p { color: rgba(0,0,0,0.5) }
```

Creará un color negro con una transparencia del 50%. Para ver el efecto, habrá que tener una imagen por debajo, o si nuestro color de fondo es blanco, nos aparecerá de color gris.

2.2.2. Colores HSL

El modelo de color **HSL** (***hue, saturation, light***), tono, saturación y luz, define el color a partir de los tres parámetros. Así, para una tonalidad de color dada, podemos alterar la saturación a partir del segundo parámetro, y la cantidad de luz en el tercer parámetro.



Por ejemplo:

Será definida por los colores:

```
background:hsla(320, 100%, 10%);
background:hsl(320, 80%, 30%);
background:hsl(320, 60%, 50%);
background:hsl(320, 40%, 70%);
background:hsl(320, 20%, 90%);
```

También podemos añadir el componente de alpha si definimos como: ***hsla(320, 80%, 30%, 0,7)***.

El modelo de color HSL es implementado en los navegadores basados en Webkit y Firefox.

2.2.3. *Opacity* (opacidad)

Como su nombre indica, la opacidad nos permite definir el nivel de transparencia para un selector. A diferencia del modelo RGB, la transparencia se aplica al objeto y todos sus hijos y/o contenidos. Así, por ejemplo:

```
<img src="img_02.jpg" style="opacity:0.2">
<img src="img_02.jpg" style="opacity:0.4">
<img src="img_02.jpg" style="opacity:0.6">
<img src="img_02.jpg" style="opacity:0.8">
<img src="img_02.jpg" style="opacity:1">
```

producirá:

Todas las propiedades son admitidas por los navegadores basados en tecnología Webkit, Geko (Mozilla) y Opera, mientras que Explorer no lo admite. De todos modos, la familia Microsoft admite otra propiedad complementaria, que nos permite realizar el mismo efecto, siendo:

```
filter: alpha(opacity = 50);
```

Todas estas propiedades podemos utilizarlas siempre que tengamos en cuenta un mecanismo alternativo para los navegadores que no lo admiten. La mejor mecánica de trabajo es definir primero un color admitido para después añadir las propiedades CSS3. De este modo el navegador que no la admite utilizará la primera, mientras que los que la admitan podrán utilizarla.

2.3. Nuevas propiedades

2.3.1. Esquinas redondeadas (*border-radius*)

Seguramente es la propiedad CSS3 más esperada y deseada por los diseñadores html. Permite redondear las esquinas de una caja (div, span...) dado un radio, sin la necesidad de utilizar una imagen de fondo, producida con una herramienta de edición gráfica.

Podemos manipular las esquinas de una caja de manera uniforme utilizando la propiedad ***border-radius: 15px;*** como se muestra en el ejemplo siguiente:

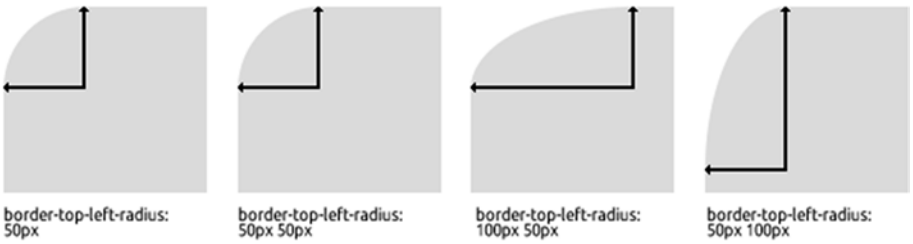
```
#example1 {
   -moz-border-radius: 15px;
   border-radius: 15px;
}
```

Esta caja tiene esquinas redondas en todos los navegadores disponibles

Hemos de fijarnos en la presencia de la propiedad ***-moz-border-radius: 15px***; específica de los navegadores basados en Geko.

Pero también podemos definir las propiedades para cada una de las esquinas utilizando las propiedades ***border-bottom-left-radius, border-bottom-right-radius, border-top-left-radius, border-top-right-radius***. Estas propiedades admiten dos parámetros para definir el radio de curvatura de la esquina. Si defi-

nimos sólo uno, conseguimos esquinas simétricas, mientras que si definimos los dos, estamos tratando independientemente el parámetro x del y, como se puede apreciar en el gráfico siguiente:



Hay que señalar que para los navegadores basados en Geko, se deben emplear las propiedades: ***-moz-border--moz-bottom-left-radius, -moz-border-bottom-right-radius, -moz-border-top-left-radius, -moz-border-top-right-radius***.

Por otro lado, la propiedad ***border-radius*** también puede ser empleada para definir de una vez las cuatro esquinas de la caja:

```
border-radius: 5px 10px 5px 10px / 10px 5px 10px 5px;
border-radius: 5px;
border-radius: 5px / 10px;
```

En la primera definición, el primer grupo de 4 medidas definen las propiedades horizontales de los cuatro radios, mientras que el segundo grupo, utilizado separado por el carácter /, define los radios verticales. En el segundo ejemplo, se definen todos con un radio de 5px. En el tercero, se define un radio horizontal de 5px y uno vertical de 10px para las 4 esquinas.

Todas estas propiedades están disponibles en los navegadores basados en Webkit, Geko, Opera, y en la última versión del navegador de Microsoft (**IE9**), que finalmente intenta cumplir con las especificaciones del W3C.

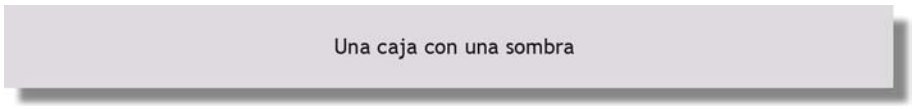
2.3.2. Sombras (box-shadow, text-shadow)

En la versión 2 de la especificación se introdujo la posibilidad de generar sombras directamente desde código CSS, pero se quitó en la 2.1 y se ha vuelto a introducir en la versión 3.

Las propiedades para generar sombras en una caja son las siguientes:

```
#example1 {
-moz-box-shadow: 10px 10px 5px #888;
-webkit-box-shadow: 10px 10px 5px #888;
box-shadow: 10px 10px 5px #888;
}
```

generará:



La propiedad ***box-shadow*** admite una lista de 5 elementos, que definen, en orden, desplazamiento horizontal, desplazamiento vertical, desenfocado, extensión y color de la sombra. Opcionalmente, se puede añadir la palabra clave ***inset***, que nos permitirá hacer una sombra interna en lugar de una externa:

Algunos ejemplos:

```
#Example_F {
-moz-box-shadow: 0 0 5px 5px #888;
-webkit-box-shadow: 0 0 5px 5px#888;
box-shadow: 0 0 5px 5px #888;
}
```

La propiedad ***text-shadow*** se comporta de igual modo, y nos permite generar sombras en los textos.

Por otro lado, cabe decir que podemos tener más de una sombra por elemento, es decir, que las podemos superponer como efecto de papel cebolla. Para hacerlo:

```
box-shadow: 0 0 10px 5px black, -20px 0px 50px blue;
```

Generará dos sombras, una negra y una azul. Podemos encadenar varias simplemente separando la lista por comas.

2.3.3. Múltiples imágenes de fondo

Otra característica interesante de CSS3 es la posibilidad de incluir múltiples imágenes de fondo en un mismo elemento. Así, el código siguiente es completamente funcional:

```
#example1 {
width: 500px;
height: 250px;
background-image: url(fons1.png), url(fons2.png);
background-position: center bottom, left top;
background-repeat: no-repeat;
}
```

Fijaos en que definimos dos imágenes de fondo: fondo1.png y fondo2.png, y también dos posiciones diferentes para cada una. Una centrada en la parte inferior y otra en la parte superior izquierda.

2.3.4. *Borders* (filetes) con imágenes

Podemos definir imágenes para los filetes de nuestros bloques. Así:

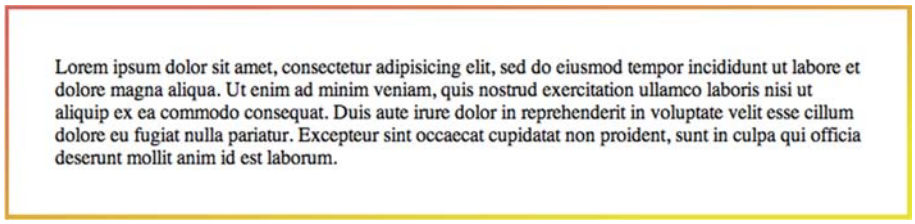
```
border-image: url(border-image.png) 25% repeat;
```

Definiremos la imagen que queremos emplear como filete. El porcentaje está relacionado con la parte que queremos emplear y, finalmente, si queremos que repita modo patrón o queremos que sólo aparezca una vez. A primera vista parece complicado, pero nos podemos adentrar un poco en el tema con un ejemplo. Dada la imagen:

Y queriéndola emplear como fondo, utilizando el siguiente código:

```
#un {
   padding:30px;
   border-width:10px 10px 10px 10px;
   -webkit-border-image: url('css3_imatge_fons.png') 10 100 10 10 repeat stretch;
   background-color:#fff;
    }
```

Obtenemos el siguiente resultado:



Si nos fijamos, primero definimos un ancho de filete (10 píxeles) con la propiedad ***border-width***, después definimos la imagen que utilizaremos y las proporciones en píxeles (también admite porcentajes) de imagen para cada una de las esquinas. Finalmente, tenemos la propiedad (***stretch/repeat***) por si queremos que repita o se ajuste.

2.3.5. Columnas de texto

Otra de las novedades es la posibilidad de trabajar con columnas de texto. Realmente, con el ancho actual de las pantallas ha sido necesario hacerlo, puesto que un ancho demasiado grande en los párrafos de texto afecta a su legibilidad. Así, desde la especificación 3, podemos utilizar diseños con columnas. Las propiedades que hemos de emplear son:

```
-moz-column-count: 3;
-webkit-column-count: 3;
-moz-column-width: 200px;
-webkit-column-width:200px;
-moz-column-gap: 20px;
-webkit-column-gap: 20px;
```

Nos generará una estructura de tres columnas siempre y cuando puedan cumplir con el tamaño de ancho de columna preferido, fijado con ***-column-width***, si se hiciera así, emplearía la parte proporcional del área disponible. Con la propiedad ****-column-gap: 20px*** definimos el margen que queremos entre columnas, y también disponemos de la propiedad ***column-rule***, con la que podemos definir un filete que divida las columnas.

Nos generará,

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2.3.6. WebFonts

Otra de las posibilidades que resulta muy atractiva a la hora de emplear los CSS3 es la capacidad de adjuntar tipografías a un documento html. Pero como sucede con todas las cosas relativas a la industria de los navegadores, existen diferentes sistemas, formatos, etc. y, evidentemente, licencias de las tipografías.

A partir de la especificación 3, disponemos de la etiqueta @font-face, que, como en el ejemplo, nos permite definir una tipografía.

```
@font-face {
   font-family: Gentium;
   src: url(gentium.otf);
```

```
}
```

Como vemos en el ejemplo anterior, esta etiqueta nos permite adjuntar al documento una tipografía. Pero los problemas vienen a raíz de los diferentes formatos que cada navegador y versión entienden del archivo de tipografías. Así, el resultado para una correcta visualización en todos los navegadores sería:

```
@font-face {
font-family: 'UbuntuRegular';
src: url('webfont.eot');
src: url('webfont.eot?#iefix') format('embedded-opentype'),
   url('webfont.woff') format('woff'),
   url('webfont.ttf') format('truetype'),
   url('webfont.svg#UbuntuRegular') format('svg');
 font-weight: normal;
 font-style: normal;}
```

Y de este modo obtenemos una correcta visualización en todas las versiones de los navegadores. Para generar todas las versiones de la fuente, es necesario que empleemos una herramienta como la siguiente:

http://www.fontsquirrel.com/fontface/generator

Dado un archivo ttf u OpenType, esta herramienta nos generará tanto las versiones que se van a subir al servidor como la definición CSS para emplearlas en nuestro documento.

Debido a la complejidad existente y a la necesidad que tienen los diseñadores de trabajar con tipografías personalizadas, han aparecido servicios en línea que nos permiten adjuntar tipografías a nuestros documentos, sin necesidad de preocuparnos por las licencias ni por las conversiones de archivos. Actualmente los dos servicios más extendidos son:

- El comercial TypeKit (http://typekit.com/), donde se preocupan de licenciarnos y servirnos el uso de una tipografía comercial.

- El servicio de Google webfonts (http://www.google.com/webfonts), mediante el cual, empleando tipografías OpenSource, podemos adjuntar las tipografías a nuestros documentos.

Ambos servicios se encargan de servir las diferentes versiones de la tipografía, para cada versión del navegador, facilitándonos un pequeño código CSS, adjuntable a nuestro CSS, y que nos dará acceso al uso de la tipografía.

2.3.7. MediaQueries

Desde la versión 2.1 de los CSS, existe la posibilidad de definir estilos en función del uso de la hoja: una para la pantalla (***screen***), otra para la impresión (***print***), otra para los lectores de voz (***voice***). A partir de la especificación 3 esto va un paso más allá, y nos permite definir estilos específicos para diferentes tamaños de pantalla. Fijémonos en esta definición:

```
@media screen and (max-width: 600px) {
   .class { background: #ccc; }
}
```

Define estilos específicos para navegadores con un ancho de pantalla menor de 600 píxeles. También lo podemos hacer directamente con una hoja de estilos utilizando la etiqueta:

```
<link rel="stylesheet" media="screen and (min-width: 600px)"    href="small.css" />
```

En este caso, estaríamos refiriéndonos a tamaños de pantalla mayores de 600 píxeles o, como mínimo, 600px.

Existe una pequeña y sutil diferencia: podemos emplear ***min-width*** o ***min-device-width***. Con la primera hacemos referencia al área visible en estos momentos en el dispositivo, mientras que con ***device-width*** hacemos referencia a la resolución del dispositivo.

La posibilidad de poder trabajar así, realmente, nos ofrece soluciones óptimas, puesto que podemos emplear sólo CSS para crear una versión móvil de un *site*.

Web recomendada

Podemos ver muchos ejemplos en la web:
http://www.mediaqueri.es

2.4. Transiciones CSS

Las transiciones CSS son pequeños cambios en propiedades de la hoja de estilos desencadenados por acontecimientos generados por interacciones del usuario, como por ejemplo cuando el ratón pasa por encima de algo (***:hover***) o un campo de formulario cambia, etc. En una transición, estos cambios en las propiedades se producen de manera progresiva durante un intervalo de tiempo.

Supongamos un ejemplo simple para entender cómo funcionan las transiciones CSS:

```
<a href="#" class="boto">Botón para transición</a>
 a.boto {
   text-decoration:none;
   color:#fff;
   padding: 5px 10px;
   background: #f76c6c;
```

```
    -webkit-transition-property:background;
    -webkit-transition-duration: 0.5s;
    -webkit-transition-timing-function:ease;
}
a.boto:hover {
    background: #d22828;
}
```

Generará la siguiente imagen y, al pasar el ratón por encima (***hover***), desencadenará la transición correspondiente.

Del código siguiente es necesario que expliquemos y destaquemos lo siguiente:

Web recomendada

Aquí podremos encontrar una lista de las propiedades que pueden ser animadas (transiciones):
http://www.w3.org/TR/css3-transitions/#properties-from-css-

- La transición no se debe poner en el evento (en este caso, el selector ***hover***), puesto que simplemente si la tiene ya la puede efectuar cuando toque.
- Lo que definimos en la transición es la propiedad de la transición que queremos animar, en este caso el ***background***.
- Podemos definir la duración y también la interpolación de tiempo. En este caso se utiliza una función de aceleración. Las funciones de las que disponemos son las siguientes: ***ease, linear, ease-in, ease-out, ease-in-out***, y ***cubic-bezier***.
- También podemos definir un retraso en la ejecución de la animación con la propiedad ***-webkit-transition-delay: 0.5s;***.

De todos modos, existe un acelerador para las transiciones que es el siguiente:

```
-webkit-transition: background 0.3s ease 0.5s;
```

A efectos prácticos, realiza lo mismo que la anterior definición pero de una manera más compacta. También cabe decir que se pueden definir transiciones de más de una propiedad separándolas con una (,):

```
-webkit-transition: background .3s ease, color 0.2s linear;
```

Las transiciones con CSS sólo están disponibles en los navegadores basados en Webkit, Opera y en la versión 4 del Geko. Por ello, es necesario que añadamos el acelerador para compatibilidad:

```
-webkit-transition: background .3s ease, color 0.2s linear;
-moz-transition: background .3s ease, color 0.2s linear;
-o-transition: background .3s ease, color 0.2s linear;
```

```
transition: background 3s ease, color 0.2s linear;
```

Otra opción disponible es hacer la transición de todas las propiedades que cambien. Ello lo podemos realizar con la instrucción ***all:***

```
transition: all 3s ease, color 0.2s linear;
```

2.5. Ejercicios

1. Cread unos botones CSS con efectos de degradado interno y sombra exterior y animación en el *hover*.

2. Maquetad un pequeño *layout* usando *mediaquery*, *webfonts*, esquinas redondas, *borders* en imágenes y con *fallback*.

3. Bibliotecas Javascript

3.1. ¿Por qué una biblioteca?

Javascript es el lenguaje de programación utilizado en el desarrollo de aplicaciones web por parte del cliente. Recordando un poco la historia, Javascript como lenguaje nace en 1995 gracias a Netscape Corporation, que lo incorpora como lenguaje de *script* en su primera versión del cliente de WWW. Paralelamente, **Microsoft** inicia el desarrollo de su cliente de WWW, **Internet Explorer**, y copia el lenguaje de Netscape pero cambiándole el nombre por el de jScript. Realmente los dos lenguajes son muy parecidos, pero diferentes.

Desde el principio se generan diferencias en el uso, con el modo en el que se interactúa con el DOM (*document object model*), el sistema de eventos, y en otras muchas pequeñas peculiaridades que los hacen diferentes. Así, nos encontramos con un lenguaje que debe interactuar con modelos de clases diferentes y utiliza sistemas de eventos distintos.

Al principio la programación de cliente era terriblemente difícil, puesto que había que trabajar con cada una de las especificaciones para los diferentes navegadores, lo que provocaba que el código que se generaba fuera poco sólido y mantenible. Fácilmente podías encontrarte desarrollada una pequeña función con dos condicionales para cada especificación de navegador.

Para solucionar estos problemas de interacción del lenguaje con los navegadores, nacieron bibliotecas cuyo objetivo es conseguir una API (***application programming interface***) común a los diferentes navegadores.

De este modo, en este capítulo expondremos lo que, a nuestro entender, son las principales bibliotecas que existen en el trabajo de Javascript en el desarrollo de aplicaciones web. Pero antes haremos una primera especificación de las tareas que necesitamos para desarrollar aplicaciones web en el cliente, para así poder evaluar y ver cómo tratan esto las diferentes bibliotecas.

3.2. ¿Qué nos ha de ofrecer una biblioteca Javascript?

Fundamentalmente una solución a los dos retos básicos que afrontamos cuando desarrollamos aplicaciones web:

- Interactuar con el **DOM**. Seleccionar, añadir, modificar y borrar nodos. Seleccionar conjuntos de nodos y aplicarles estilos CSS. Generar nuevo contenido.

- Interactuar con el usuario mediante el sistema de eventos: capturar acciones de ratón, de teclado y procesarlas correspondientemente.

Además, también nos debería ofrecer un sistema unificado de comunicación con el servidor de manera asincrónica (AJAX), un sistema para poder trabajar con formularios de datos (un modo de interacción con el **DOM**) y una especificación de componentes de software de tipo *widget* que nos permitan expandir las funcionalidades propias del navegador.

Pero una biblioteca usualmente también ofrece un conjunto de convenciones a la hora de desarrollar software, esto es, una manera de hacer, una filosofía de trabajo.

Así, podemos afirmar que una biblioteca para Javascript nos debe permitir desarrollar nuestras aplicaciones web, de tal manera que no nos debamos preocupar por las diferencias e incompatibilidades entre navegadores.

3.3. ¿Qué bibliotecas estudiaremos?

Actualmente existen multitud de bibliotecas para Javascript, pero aquí nos interesa estudiar sólo las más importantes. Así, buena parte del trabajo lo realizaremos con uno de las más extendidas, el jQuery, creado por John Resig. Se estima que tres de cada cuatro *websites* que utilizan una biblioteca javascript la usan, y la utilizan también empresas como Amazon, Microsoft, BBC o Twitter. Seguramente también es la más fácil de utilizar. Con ella aprenderemos cómo manipular el DOM, cómo trabajar con eventos, cómo generar peticiones al servidor no-sincrónicas y cómo utilizar su capa de componentes (jquery-ui).

Estudiaremos el ***prototype***, que fue una de las primeras bibliotecas utilizadas y de la que se sirven gente como la propia Apple. También veremos, de paso, la biblioteca ***YUI*** (escrita para Yahoo) y la Motools.

De todas ellas veremos los dos pilares básicos: la interacción con el DOM y el sistema de eventos.

4. jQuery

4.1. Obtener jQuery

Podemos descargar jQuery de su web: www.jquery.com, pero también la podemos utilizar directamente desde los **CDN (*content delivery network*)** de Google. Si accedemos a su web, vemos que disponemos de dos versiones diferentes (***production, minified and gziped***), versión preparada para entornos de producción con el código comprimido y optimizado para ocupar muy pocos Kb de descarga, o la versión de desarrollo. Sólo descargaremos esta última si lo que queremos es revisar y leer el código de la propia librería. Podéis ver cómo está hecha, puesto que la versión de producción es completamente utilizable.

Por otro lado, si queremos podemos utilizarla desde un ***CDN***. Esta manera es recomendada por la comunidad. Un ***CDN*** es una red de distribución de contenidos a nivel global, con servidores geolocalizados de manera óptima para mejorar los tiempos de descarga. De este modo, si enlazamos la librería desde aquí, cuando el cliente de web intente descargarla muchas veces se encontrará con que ya la tiene en la ***memoria caché***.

Sea como fuere, habrá que enlazar la librería en nuestro documento html. Para ello, utilizaremos la etiqueta estándar de html, <script>, del siguiente modo:

```
<script type="text/javascript" src="jquery.js"></script>
```

O bien utilizaremos la etiqueta con una dirección CDN.

```
<script type="text/javascript"
            src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js"></script>
```

Una vez hecho esto, ya podremos empezar a utilizar la librería completa para realizar nuestras aplicaciones.

4.2. Cómo funciona jQuery: el objeto $

Como veremos más adelante, algunos de los *frameworks* de Javascript se organizan a partir de un objeto global. De este modo, el espacio de memoria queda limpio y se puede implementar la librería en espacios complejos con otras muchas funciones.

jQuery utiliza el símbolo $() como función que nos permitirá interactuar con la librería. De hecho, el símbolo $ es un simple sinónimo de la verdadera función window.jQuery, así:

```
var jQuery = window.jQuery = window.$
```

A partir de esta función, se organizan todas las funcionalidades de la librería y ella nos sirve para realizar cualquier operación. Para compatibilizar con otras bibliotecas (***Prototype*** utiliza el mismo símbolo), jQuery nos ofrece la función ***jQuery.noConflict();*** que desactiva el símbolo $() dejándolo con jQuery().

4.3. Interactuando con el DOM

El núcleo de jQuery es interactuar con el DOM. De hecho, la función $() nos permite ***"seleccionar"***, recuperar referencias a elementos del DOM y, a partir de aquí, empezar a interactuar con el lenguaje de programación. No es convencional, puesto que no sigue la estructura usual de programación de aplicaciones, pero sí que es muy práctico, puesto que el 80% de lo que programamos en el navegador se hace empleando algún elemento del DOM. Para aclarar un poco el asunto, lo mejor es ver un ejemplo.

```
<html>
<head>
<title>Ejercicio jQuery 2</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js"> </script>
<script type="texto/javascript">
<!--
$(document).ready(function(){ // 1
   var resultat = $('#p1').text(); // 2
alert(resultat)
})
//-->
</script>
</head>
<body>
   <p id="p1">¡Hola Mundo!</p>
< /body>
</html>
```

El código que aparece en **1** todavía no es relevante, pero sirve para programar el evento **DOM *disponible***. Si nos fijamos en la parte de **2** con ***$('#p1')***, seleccionamos el elemento con ID igual a ***p1***. De hecho, lo que nos devuelve la ***función $*** es una instancia de la librería jQuery inicializada con el elemento p del DOM. A partir de aquí, el método text(), nos devuelve el contenido del nodo como cadena de texto. Después, lo guardamos en la variable resultado y la mostramos efectuando una alerta.

4.3.1. Selectores, filtros y CSS

Como muy bien se puede ver en el ejemplo anterior, la función $() recibe un selector CSS y devuelve la instancia del objeto. La cuestión es ver qué selectores podemos utilizar.

Como hemos visto en el ejemplo anterior, el primer selector es el #, igual que con CSS. Esto nos permite seleccionar elementos ID. Del mismo modo, podemos utilizar selectores con clases CSS. Así, la llamada ***$('.estilo1')*** seleccionará todos los elementos que tengan asociada la clase estilo1. Como vemos, utilizamos el mismo lenguaje que ya conocemos con CSS a la hora de interactuar con el DOM.

Cabe destacar que ambas llamadas devuelven una instancia de la biblioteca lista para ser utilizada, con la diferencia de que en la primera nos devuelve la instancia inicializada con un único elemento, y en la segunda, con un conjunto de elementos. El modo en el que hemos seleccionado los elementos del DOM era una técnica habitual para manipular en tiempo de ejecución el contenido del html.

Casi siempre podemos aplicar la misma regla de CSS a los selectores de jQuery, y encontramos:

$('*')	Todos los elementos
$('#propiedad)	Elemento con ID=propiedad
$('.clase')	Elementos que tienen la clase CSS
$('img')	Elementos
$('img,p,selectorN')	Combina los elementos img, p, selectorN

Igual que con CSS3, podemos seleccionar por atributo así:

$('a[rel="nofollow"]')	Todos los a con el atributo rel="nofollow".
$('[atributo|="valor"]')	Atributo que sea o empiece por valor.
$('[attribute="value"]')	Atributo que contenga el valor delimitado por espacios.
$('[atributo!="valor"]')	Que el atributo no tenga el valor.
$('[atributo^="valor"]')	Que el valor del atributo empiece por valor.
$('[atributo$="valor"]')	Que el valor del atributo acabe con...
$('[atributo]')	Que el elemento contenga el atributo.
$('[atributo1="1"][atributo2="2]')	Que contengan todos los atributos con valores.

Del mismo modo que en CSS3, también disponemos de filtros, pero no es necesario preocuparse de la compatibilidad de las diferentes versiones de los navegadores admitiendo ciertas propiedades, puesto que esto es trabajo de la biblioteca. Así, podemos utilizar filtros como:

$('tr:first-child')	La primera fila de una tabla. También podemos emplear :last-child, y :nth-child(4n).
$("tr:even")	Fila impar, también par con ***odd***.
$('tr:eq(3)')	El tercer elemento del conjunto seleccionado. En este caso, la tercera fila del conjunto de todas las tablas de la página. También, podemos emplear: • ***gt(n)*** conjunto de elementos a partir de n (el enésimo elemento no incluido) • ***lt(n)*** conjunto menor que n • ***not(n)*** todos, excepto n
$('tr:first')	El primer elemento. También podemos con el último empleando ***:last***.

También podemos filtrar elementos en función del contenido que tienen, así:

$("div:contains('Libro')")	Seleccionará los elementos div en cuyo contenido aparezca la palabra *libro*.
$("div:has(p)")	Elementos div que **contengan** otro elemento p.
$("td:parent")	Elementos que son padres de otro nodo, incluidos nodos de texto.

Finalmente, para completar el apartado de selectores, hay que ver una nueva funcionalidad y detallar una diferencia en los objetos que nos devuelve.

Cuando seleccionamos múltiples objetos, como por ejemplo ***$('img') para seleccionar todas las imágenes***, el objeto que nos devuelve jQuery es una instancia normal aumentada con las funciones de lista (***iterable***). jQuery ofrece una función, ***.each***, que nos permite iterar por los diferentes elementos que hemos seleccionado. Así, podríamos escribir el siguiente código:

```
$('img').each(function(index, element) {
console.log( element.src )
});
```

Que iterará por todas las imágenes del documento. Para saber si un selector nos ha devuelto un conjunto o sólo un elemento, podemos consultar la propiedad *length*, así:

```
$('img').length; // devolverá el total de imágenes que contiene el documento.
```

Como nueva funcionalidad, cabe decir que la función **$()** admite un segundo parámetro que nos permite definir el contexto en el que queremos que se realice la búsqueda del selector. Por defecto, si el parámetro no existe, la función realiza la búsqueda en todo el documento. Si aplicamos un contexto, sólo la realizará en el contexto, así:

```
<div><p>Content</p></div>
<div id="p1">
<p>Content 2</p>
< /div>
$('p',  $('#p1') );
```

Devolverá sólo el segundo p, puesto que es lo único que encontramos dentro del contexto de #p1.

4.4. Manipulando el DOM

Ahora que ya sabemos seleccionar elementos de un documento, podemos ir un paso más allá y empezar a manipularlos. Podemos cambiarles las propiedades CSS, leer y modificar sus atributos, copiarlos, eliminarlos y añadir más.

4.4.1. Propiedades CSS

Seleccionados un conjunto de elementos, podemos manipular sus propiedades CSS con el método .css, así:

```
$('p').css('color', 'red);
```

Cambiará el color del texto de todos los párrafos a rojo. Pero también podemos cambiar muchas propiedades a la vez. Para ello, la función recibirá un objeto Javascript y quedará en el siguiente formato:

```
$('p').css({ 'color': 'grey',
'padding': '5px',
'background-color': 'yellow' });
```

La mayoría de los métodos del objeto jQuery que pueden admitir un par de parámetros también pueden admitir un objeto, y así realizar mucho más trabajo de una sola vez.

De la misma manera que los podemos cambiar, también los podemos consultar pidiendo la propiedad:

```
$('p').css('color')
```

Nos devolverá el color del elemento p.

Hay que decir que existen algunos métodos en modo de acceso directo, como por ejemplo:

.show()	Nos mostrará el elemento si estaba escondido.
.hide()	Lo esconderá.
.toggle()	Actúa a modo de interruptor. Si el elemento está escondido, lo muestra, y si el elemento está visible, lo esconde.

También, y desde Javascript, con ***jQuery*** podemos añadir una nueva clase CSS a un elemento seleccionado con la siguiente orden:

```
$('#id').addClass('nombredelaclase')
```

O podemos consultar si tiene asignada la clase con:

```
$('#id').hasClass('nombredelaclase')
```

O la podemos eliminar:

```
$('#id').removeClass('nombredelaclase')
```

También podemos manipular y consultar la altura y anchura de un elemento mediante los métodos ***.height()*** y ***.width()***, que nos sirven tanto para asignar una altura y anchura, como para consultar la altura y anchura reales. Se debe tener en cuenta que la medida de un elemento que nos dará las funciones no computa los ***paddings***. La medida con ***padding***, pero sin el ***border***, la podemos saber si consultamos los métodos ***innerHeight()*** e ***innerWidth()***.

4.4.2. Atributos de los nodos

Del mismo modo que podemos consultar o modificar propiedades CSS, jQuery nos facilita una fórmula para interactuar con los atributos de cada nodo utilizando la función:

```
<img width="34" />
$('img').attr('width', '44')
```

Igual que el CSS, el método también admite un objeto como parámetro y, en su defecto, si sólo le enviamos un parámetro que es una propiedad, nos devolverá el valor pertinente.

4.4.3. Añadir contenido en el DOM

Para insertar contenido dentro del **DOM**, disponemos de tres fórmulas básicas en relación con la selección correspondiente:

1) Añadir envolviendo

La acción de añadir un nodo envolviendo ***.wrap*** el selector en curso generará lo siguiente:

```
<p id="un">Una prueba</p>
$('.inner').wrap('<div class="new" />');
<div class="new">
   <p id="un">Una prueba</p>
< /div>
```

De este modo, cuando hagamos un ***.wrap*** de un selector, lo que haremos será envolverlo con la etiqueta indicada. De la misma manera, podemos hacer un ***.unwrap()***.

Existe también el método ***.wrapAll***, que envolverá todo el conjunto de selectores en un único div.

2) Añadir dentro

Nos añadirá el contenido, pasado como parámetro, en diferentes posiciones del selector en función del método que usemos:

.append('content')	Añade el contenido al final del selector.
.prepend()	Añade el contenido al principio del selector.
.html()	Recupera el contenido de un selector o se lo asigna. Siempre contenido html.
.text()	Recupera el texto de un selector sin etiquetas. También lo asigna.

3) Añadir fuera

$().after(), $().before() inserta los elementos seleccionados justo el nodo antes o después del seleccionado. Si la selección nos devuelve diferentes nodos, efectuará la operación para cada uno de ellos.

4.4.4. Borrar nodos

Dado cualquier selector, lo podemos borrar simplemente ejecutando el método ***.remove()***. También podemos utilizar el método ***.empty()***, que vaciará el contenido de un nodo, incluido texto plano.

4.5. Funciones generales

Como biblioteca o marco de trabajo, jQuery nos ofrece una serie de extensiones, métodos o funciones que nos permiten añadir pequeñas utilidades y funcionalidades al Javascript:

$.each(collection, callback(indexInArray, valueOfElement))

Es una función que nos permite programar iteraciones. Supongamos lo siguiente:

```
a = [1,2,3,4,5]
resultado = 0
$.each(a, function(index, valor) {  resultado += a }
alert('el resultado es' + a)
```

Como ***collection***, le podemos pasar cualquier elemento que sea iterable en Javascript, como por ejemplo, un array o el resultado de un selector. De hecho, un patrón común cuando queremos realizar cosas con un conjunto sería:

```
$('.item').each(function(ind) {
   $(ind).text()
 })
```

En el ejemplo anterior, iteraríamos por la colección de nodos que tienen asignado el estilo ***item***. En este patrón, es muy interesante saber que ***$(ind)*** es el nodo actual.

También podríamos iterar sobre las propiedades de un objeto o ***hash***:

```
$.each( { name: "Pere", lang: "JS" }, function(k, v){
   alert( "Clave: " + k + ", Valor: " + v );
 });
```

$.extend({}, objeto1, objeto2)

El método *extend* nos permite añadir propiedades a un objeto definido previamente:

```
a = {un: 'hola', dos:'dos' }
b = {un:'sí', tres:'quepasa'}
```

```
$.extend(a, b)
a == {un:'sí', tres:'quepasa', dos:'dos'}
```

Es de gran utilidad para escribir código modular, puesto que nos permite de una manera muy fácil ampliar objetos con propiedades y métodos de otros. Una aplicación práctica es la creación de ***mixins***, objetos que amplían la funcionalidad de otros, como por ejemplo:

```
var a = {
   'hola': function() { return 1; },
   'adiós': function() { return 2; } }
var mixin = { 'hola': function() { return 2; } }
$.extend(a, mixin)
console.info( a.hola() == 2 )
console.info( a.adiós() == 2 )
```

Si nos fijamos en el ejemplo, disponemos de un objeto A que tiene una función hola que devuelve 1. Pero después le aplicamos, con el método ***$.extend***, el objeto mixin sobrescribiendo la función hola.

Al final del ejemplo, podemos ver cómo ***a.hola()*** ya no devuelve 1, sino que devuelve 2, puesto que ha sido sobrescrita (***extendida***).

Si revisamos el código fuente de jQuery, apreciaréis que la mayoría de los métodos nuevos los instalan de este modo. También lo podemos hacer extendiendo la cadena de prototipaje:

```
$.extend(a.prototype, mixin)
```

Que nos generaría métodos públicos del objeto A, mientras que el A del ejemplo simplemente es un paquete de funciones (***namespace***).

$.inArray(valor, array)

Busca valor en lista. Si no lo encuentra, devuelve -1. Si lo encuentra, devuelve la posición >0.

$.isArray(p)

Devuelve ***true***, si p es un array.

$.merge(a, b)

Mezcla el contenido de a y b, devolviendo un nuevo ***array***.

Web recomendada

Podéis encontrar muchas más utilidades en la dirección:

http://api.jquery.com/category/utilities/

4.6. Sistema de eventos

Tan importante como poder manipular fácilmente el **DOM** desde nuestros programas, el otro gran pilar de las bibliotecas de Javascript es la gestión de los eventos (*events*). Sabido es para todos los que se han querido enfrentar a esta tarea sin utilizar ningún *framework* que la cosa no es trivial, puesto que cada navegador nos ofrece un sistema de eventos diferente.

Generalmente, un evento, en jQuery, se escucha mediante el pedido ***.bind***. Así, para poder capturar los clics en todos los enlaces, podemos escribir:

```
$('a').bind('click', function() {
   contador += 1
})
```

Siempre sigue el mismo patrón: seleccionamos el nodo al que queremos asignar el evento, y con la orden *bind*, le podemos asignar el evento que deseemos. Por ejemplo, para asignar un evento *change* a un campo desplegable (***select***), escribiremos:

```
$('#idselect').bind('change', manipulador)
```

Donde ***manipulador*** puede ser, como en el caso anterior, una función anónima o puede ser el nombre de una función (la variable que contiene el objeto función).

Cabe decir que a partir de la versión de jQuery 1.4, a la función *bind* le podemos asignar un objeto que contenga una lista de eventos mapeados, como por ejemplo:

```
$('a').bind({
   'mousenter': function(evt) {
   $(this).css('color', 'black')
},
   'click': function(evt) {
$(this).toggle();
}
});
```

Es importante destacar que en la función manipuladora del ***event***, la variable *this* apunta al nodo del DOM que la desencadena, y que por lo tanto la orden ***$(this)*** selecciona el propio nodo.

Asimismo, la función recibe un parámetro que contiene un objeto de tipo ***event***. Este parámetro a menudo es omitido, pero con él podemos acceder a información adicional en el momento de generación del evento y durante su propagación. Con ***jquery*** el sistema de eventos sigue una regla de propagación en burbuja (*bubbling*), que cuando un evento es generado en un nodo se propaga hacia sus padres. Imaginemos el siguiente ejemplo:

```
$(document).ready(function(){
   $('tr').click(function(){
      alert('clic a tr');})
      .height(30)
      .css( 'background-color', '#eaeaea');
   $('a').click(function(e){
      alert('clic a a');
      e.stopPropagation()
      });
   });
   </script>
</head>
<body>
<table style="border:1px solid black;">
   <tr>
      <td><a>Hola</a></td>
   </tr>
</table>
```

Si hiciéramos clic en el enlace a, lo capturaríamos desde la función del evento, pero como el evento seguiría su proceso de propagación, también ejecutaría el manipulador asignado al nodo padre tr, y nos generaría un doble ***alert()***, el del elemento a y el del elemento tr. Es decir, se ejecutarían ambos eventos. Para impedir esto y romper la cadena de propagación, utilizaremos el objeto evento que recibe el manipulador y llamaremos al método ***.stopPropagation()*** tal y como hemos hecho en el ejemplo.

En el objeto evento también podemos encontrar otros datos, como las propiedades ***pageX*** y ***pageY***, que son las coordenadas de ratón en las que se ha desencadenado el ***event***. También, dentro de este objeto encontramos propiedades como ***target***, que es el objeto que genera el evento, o ***currentTarget***, que es el objeto desde el que se ha iniciado la propagación.

Del mismo modo que asignamos un evento con el método ***bind***, lo podemos desasignar con el método ***.unbind***. Así:

```
$('tr').unbind('click');
```

Los eventos pueden ser ejecutados de manera manual usando el comando ***.trigger***('nombrevento'); así, si hacemos:

```
$('tr').trigger('click');
```

Se ejecutará el *handler* asignado al tal evento (siempre y cuando lo hayamos asignado a priori). Del mismo modo, podemos generar nuestros propios eventos.

Imaginemos que tenemos un reloj y queremos que nos notifique el tiempo cada segundo. Podemos, desde la función que controla el tiempo que pasa, generar un evento de tipo ***'segundo'*** (el nombre es a voluntad nuestra), como hemos realizado en el ejemplo siguiente:

```
var segundos = 0
setInterval(function(){
$('#p1').text( ++segundos )
      .trigger('segundo', [segundos])

}, 1000)
$('#p1').bind('segundo', function(evento, fecha){
   if(fecha==10) segundos = 0
})
```

Utilizamos el elemento #p1 para que nos genere un evento de segundo, después lo capturamos y hacemos que la variable que cuenta los segundos que pasan se inicialice cuando llega a 10. Para que esto funcione, necesitamos tener un nodo con id #p1 en nuestro html.

4.6.1. Tabla de eventos

blur	Cuando en un campo de formulario perdemos el foco del teclado.
focus	Cuando un elemento de un formulario recibe un clic del ratón.
load	Un elemento externo acaba el proceso de carga, como por ejemplo una imagen.
resize	La ventana cambia de tamaño, pertenece al *window*.
scroll	Hacemos *scroll* en la ventana o en un elemento div.
click	Hacemos clic sobre un nodo.
dblclick	Hacemos doble clic sobre un nodo.
mouseover	Pasamos el ratón por encima del elemento.
mouseout	El ratón sale del elemento.

change	El valor de un campo de formulario cambia.
submit	Un formulario es enviado hacia el servidor.
keydown, keyup	Pulsamos una tecla del teclado. La tecla que se ha pulsado la podemos encontrar en la propiedad *wich* del objeto evento pasado al manipulador.
error	Se desencadena, por ejemplo, cuando desde el servidor una imagen no se carga.

Existen accesos directos en la mayoría de las propiedades que podemos utilizar con un *bind*, con su nombre de función directamente. Así, podemos llamar los métodos de un selector ***.click, .change, .error***, etc.

4.7. Formularios

Otra de las tareas clave en la programación de aplicaciones web para el Javascript es la manipulación, construcción y, sobre todo, validación de formularios. ***jQuery*** nos ofrece toda una serie de métodos especialmente diseñados para consultar y validar formularios. Así, podemos acceder al valor de cualquier campo de formulario a partir de su selector efectuando:

```
$('#id').val()
```

El mismo método nos sirve para asignar un valor. Para ello, le enviaremos como parámetro el valor que se quiere asignar.

También podemos capturar el momento de enviar el formulario y programar una función que decida si se puede enviar o no, según el contenido de los campos, utilizando el evento ***submit***. Así, podemos hacer:

```
$('formulario#un').bind('submit', function(){
   // si validación correcta
   return true;
   // si hay errores en la validación
    // los podemos mostrar y se ha de devolver false
   return false;
}
```

Existen también métodos para trabajar con **AJAX** que nos permiten serializar de una vez todo el contenido del formulario haciendo: ***$('form').serialize()***, que nos generará el formulario preparado para ser enviado por GET. Por otro lado, y según nos interese, podemos utilizar ***.serializeArray()***, que nos permitirá obtener el formulario dentro una lista.

4.8. AJAX

Asyncronimous Javascript and xml consiste en una técnica mediante la cual podemos efectuar llamadas al servidor sin recargar el contenido de nuestra página. Así, podemos enviar y recibir datos desde el servidor de manera interactiva desde Javascript. Como siempre, para utilizar la técnica, cada navegador lo implementa de un modo diferente y, por suerte, ***jQuery*** nos facilita una interfaz unificada.

Para solicitar una petición contra el servidor, disponemos de la función ***.ajax***, de tipo genérico y que admite muchos parámetros diferentes, o dos alternativas mucho más directas que nos permiten solicitar documentos con ***GET o POST***.

```
$.get( url, [ data ], success(data))
```

url: Será la dirección que se va a solicitar al servidor.

data: Es un objeto Javascript con los parámetros que le queremos enviar al servidor.

success: Es un *handler* que se ejecutará con la respuesta que el servidor nos envía.

De este modo, un ejemplo sencillo con el que cargaremos un documento extra desde el servidor puede quedar así:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Ejercicio de carga de ajax</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"> </script>
<script>
$(document).ready(function(){
   $('#bt').click(function(){ // 1
      $.get('arxiu.txt', {hola:3, mon:4}, function(data){ // 2
         $('#desti').html(data); // 3
      });
   })
});
</script>
<style>
   #desti { color:blue; }
</style>
</head>
<body>
<p>Al pulsar el botón, cargaremos dentro del párrafo azul el contenido del archivo txt del
```

```
        servidor.</p>
< input type="button" id="bt" value="Carrega" />
<p id="desti">Aquí cargaremos un contenido con ajax desde el servidor</p>
< /body>
</html>
```

En el ejemplo, programamos (1) en el evento *click* del botón #bt que lance una petición ajax, por método GET (2) $.get, descargamos el archivo.txt y enviamos dos parámetros (hola y mundo). Como la petición es asincrónica (no sabemos cuándo el servidor nos responderá), le enviamos una función para que la ejecute en este momento (2). La función tiene un parámetro data, que es el contenido de la petición y que finalmente, en (3), colocamos en el nodo #destino.

Web recomendada

Podemos ver el ejemplo funcionando en:
http://multimedia.uoc.edu/~jcollell/ejercicio_ajax.html

También podemos hacer peticiones al servidor a través de ***POST*** con la función $.post y los mismos parámetros.

Podemos consultar en línea la documentación porque las tres funciones nos ofrecen multitud de opciones para simplificar el trabajo.

4.9. Componentes (*widgets*)

jQuery internamente no nos ofrece ningún sistema de componentes, pero sí que existe un proyecto paralelo que ofrece determinados *widgets*. El proyecto se denomina jqueryUI, y lo podemos revisar en www.jqueryui.com. Aunque muy ligado a jQuery, funciona de manera independiente con calendario propio de versiones.

Fundamentalmente, nos ofrece componentes visuales, componentes interactivos, efectos y una utilidad. En cuanto a componentes visuales, tiene un pequeño *widget* de calendario, un campo de autocompletado (mientras escribes, filtra resultados de una lista) y pestañas, entre otros.

Aparte de los componentes visuales, las ***jqueryUI*** también nos ofrecen una serie de componentes funcionales, como, por ejemplo, los ***draggables*** y ***sortables***. Unos nos permiten programar objetos arrastables por la pantalla y, los otros, objetos ordenables gracias a ordenación.

Todos los componentes siguen la misma nomenclatura y el mismo funcionamiento que la propia librería madre. Hay una web con ejemplos, referencia y también un selector de capas que nos permitirá configurar el componente adaptado a las necesidades de diseño de nuestro proyecto.

4.10. Patrones de uso

Como ya hemos dicho anteriormente, un *framework* o biblioteca, además de una serie de instrucciones, objetos y métodos para facilitarnos la vida a la hora de trabajar, también implica ciertos patrones de trabajo a modo de convenciones, que harán que la ejecución de aplicaciones sea más sólida y más convencional para la mayoría de los programadores del marco.

```
$(document).ready(function() {
   // código de programación
})
```

Este primer patrón emplea el evento ***ready***, un evento especial que nos notifica que el DOM del documento html ya está listo para ser utilizado o, lo que es lo mismo, que podemos utilizar de manera segura cualquiera de los elementos del documento desde nuestro programa. El evento estándar para hacer esto en Javascript es el ***window.onload***. Como principal diferencia, cabe decir que con la técnica de jQuery el evento se ejecuta sin haber descargado las imágenes asociadas con el DOM, mientras que con la convencional se debe esperar al resto y es algo menos eficiente. Hay que decir también que del document.ready podemos crear tantos otros como queramos durante el documento, mientras que del window.onload sólo podemos tener uno.

Otro patrón de uso muy interesante es la capacidad de jQuery de encadenar métodos. Desde el punto de vista funcional no supone mucho, pero sí a la hora de escribir código muy legible, puesto que sobre un mismo selector podemos ir aplicando diferentes métodos. Así, es habitual encontrar construcciones como esta:

```
$('#lelemtn')
   .css('color', 'black')
   .bind('click', alferclic)
   .bind('mouseover', rollover)
```

Otro patrón de uso muy común cuando se programa en Javascript moderno consiste en generar funciones anónimas que se ejecutan para "esconder" este código del resto de código de la página o del navegador. Así, podemos:

```
// Creamos una función anónima para utilizar como envoltorio
(function(){
   // Esta variable normalmente sería global
   var msg = "Thanks for visiting!";
```

```
    // Y la asignamos a un método global

    window.onunload = function(){
       // Podemos utilizar la variable definida globalmente
       alert( msg );
    };   // Cerramos la función y la ejecutamos. A partir de aquí, el
    // resto de las variables son invisibles
})();
```

5. Prototype

Fue una de las primeras bibliotecas en aparecer y encerrar toda la potencia de Javascript para navegadores modernos. Creada en el 2005, destaca por ser la primera en muchas cosas y por ser la librería de uso del potente *framework* web RubyonRails. También dispone de una librería de componentes gráficos y de efectos visuales llamada script.aculo.us

5.1. Interactuando con el DOM

De la misma manera que el jQuery, de hecho este ha tomado la idea de ahí, la biblioteca nos ofrece una función para acceder al **DOM** de manera fácil: ***$('idelement')*** obtiene una referencia al elemento y además le añade todos los métodos propios de la clase ***Element.Methods***. Existe también la función ***$$('classecss')***. Hace lo mismo, pero a partir de clases CSS. En este caso, en vez de devolver una referencia devuelve una lista de elementos en el mismo orden que aparecen en el **DOM**. También nos ofrece una función para acceder a formularios ***$F()***.

Una vez obtenida la referencia en los elementos, podemos navegar por los diferentes nodos vecinos en el DOM con toda una serie de métodos:

$("clar").adjacent("li.dona");

Nos devolverá un array de nodos vecinos al que tiene id #clar y que cumplen el selector.

$("clar").ancestors();

Recopila todos los padres en orden.

up/down/next/previos

Selecciona el padre, el hijo, el siguiente y el anterior.

descendants

Devuelve los hijos.

También disponemos de elementos para modificar el contenido de la página. Podemos insertar elementos utilizando la función: ***$("MainDiv").insert({top:"Añadido al inicio del elemento"})***, que primero

obtiene la referencia al elemento para después insertar el contenido en la posición superior. El objeto que recibe como parámetro puede contener propiedades como (*top, before, after, bottom*).

Podemos borrar elementos con el método ***.remove()***. También podemos utilizar ***.update()***, que actualiza el elemento con el contenido dado, y ***.replace()***, que lo cambia.

Otras funcionalidades que nos ofrece de facto permiten calcular posiciones relativas y absolutas de un elemento (***absolute, relativize***), clonar posiciones con (***clonePosition***). También podemos esconder y mostrar con (***show, hide***), cambiar opacidades con (***getOpacity, setOpacity***), tamaños (***getDimensions, getHeight, getWidth***) y hacer ***clipping*** con (***makeClipping, undoClipping***).

También nos ofrece métodos para trabajar con CSS, así, ***addClassName, removeClassName, toggleClassNames*** sirven para añadir clases, sacar clases o intercambiar clases ***CSS***; ***hasClassName, classNames***, para consultar, y ***setStyle***, ***getStyle***.

5.2. Sistema de eventos

La manera general de capturar eventos con ***Prototype*** es utilizando la función:

```
Event.observe(element, eventName, handler [,useCapture = false]);
```

Element puede ser un *string* con un ID o directamente un elemento seleccionado. El nombre del evento debe ser el estipulado por el W3CDOM nivel 2, el *handler* es una función, y el *flag**useCapture*** determina si utilizamos el sistema de burbujas o no.

Un buen ejemplo de uso podría ser:

```
<ul id="PeopleList">
  <li class="noia" id="judy">Joana</li>
  <li class="noi" id="sam">Sam</li>
  <li class="noia" id="amelia">Amelia</li>
</ul> </div>
<textarea id="results" cols="50" rows="10"></textarea>
<script type="text/javascript" src="prototype-1.6.0.2.js"></script>
<script type="text/javascript">
$("results").value = "";
Event.observe("PeopleList", "click", function(e) {
$("results").value += "clicked on " + e.target.id + "\n";
});
</script>
```

Al hacer clic sobre los elementos, se ejecutará la función que escribirá el ID del elemento pulsado en el campo textarea.

La función *handler* tiene un parámetro de tipo evento del que podemos ver diferentes métodos y propiedades. Así, ***Event.target*** es el nodo que ha generado el evento, ***this*** es el nodo en el que se ha capturado el evento, y ***Event.element*** es el nodo que ha generado el evento.

.stopObserving() elimina las referencias a los eventos y ***unloadCache()*** elimina todos los eventos capturados. Para no dejar que un evento haga la burbuja, ***Event.stop()***.

5.3. AJAX

```
var url = "http://myserver/api/get";
var ajaxCall = new Ajax.Request(url, {
method: 'get',
parameters: {a:1, b:2},
onSuccess: function() { alert('respuesta correcta.2xx') },
onFailure: function() { alert('Fallo') },
onComplete: function() { alert('Completada') },
});
```

Con esta simple signatura ya podemos realizar peticiones contra el servidor y destacar el uso de diferentes *handlers* que nos permiten capturar la respuesta desde el servidor.

5.4. Utilidades generales

Una de las carencias de la biblioteca y que a menudo recibe críticas es su construcción, puesto que basa parte de su funcionamiento en extensiones que cuelgan de la cadena de prototipaje de los objetos normales del lenguaje. Así, nos ofrece herramientas para extender y personalizar más aún la biblioteca.

Object.clone(objecte): Hace una copia exacta de un objeto.

Object.extend(destí, source): Amplía el objeto destino con las propiedades y métodos de origen.

isArray, isElement, isFunction, isHash, isNumber, isString, isUndefined: Devuelven *true*, si son del tipo.

Object.keys(c): Devuelve una lista (**array**) de las claves de un objeto.

Object.values(c): Devuelve una lista (**array**) de los valores de un objeto.

toHTML, toJSON, toQueryString: Convierte un objeto en html, json o formato query string para ser enviado/guardado en formato texto.

Una de las carencias que tiene Javascript (a pesar de que cuando se conoce en profundidad no lo es) es la existencia de un sistema para trabajar con clases (OOP) y herencia. Prototype define un pequeño sistema con:

```
var myParentClass = Class.create({
   parentFunction: function() { return "parent";}
});
var myClass = Class.create(myParentClass,{
   classFunction: function() { return "class";}});
var c = new myClass();
```

También existen funciones añadidas al objeto ***String.blank() y String.empty()***, ***startsWith, endsWith,*** entre otras muchas.

Web recomendada

Podemos consultarlo en línea, donde encontraremos una amplia documentación:
http://api.prototypejs.org/

5.5. Componentes (*widgets*)

A partir de la librería Prototype y utilizándola como base, nace una extensión: http://script.aculo.us/ que le añade toda una serie de componentes que nos permiten generar *widgets* ***drag&drop*** de manera simple o *widgets* para ordenar toda una galería de efectos y componentes, como casillas de autocompletado o un sistema de edición de contenidos en el propio documento html.

6. YUI

Web recomendada

Puede ser descargada desde: http://developer.yahoo.com/yui/

Nacida en el equipo de desarrollo de Yahoo, la Yahoo User Interface Library es un conjunto de más de 50 Mb de componentes para el desarrollo de aplicaciones de cliente. Incluye archivos, *assets*, componentes y una potente biblioteca Javascript que aquí analizaremos brevemente.

La biblioteca YUI seguramente es de las más completas y complejas, realmente puede hacer de todo e incluso incluye un *framework* CSS.

Para utilizarla, podemos usar un sistema en línea que nos permita generar una librería personalizada para cada uso en http://developer.yahoo.com/yui/3/configurator/.

Pero también existe la opción de utilizar la librería descargando sus componentes directamente desde el ***CDN*** de Yahoo. Para hacerlo, incluiremos el siguiente archivo:

```
<script src="http://yui.yahooapis.com/3.2.0/build/yui/yui-min.js"></script>
```

A partir de aquí, podemos requerir y cargar bajo demanda los módulos que necesitamos, como por ejemplo:

```
YUI().use('node', 'anim','yui2-calendar', function(Y) {
    var YAHOO = Y.YUI2;
    Y.one('#test');
});
```

6.1. Trabajando con el DOM

El trabajo con el DOM con la YUI se realiza con el módulo Node. Este contiene métodos para seleccionar nodos del html e interactuar con ellos.

Mediante el método *one*, podemos utilizar un selector CSS para seleccionar un nodo, pero sólo uno, y con él obtendremos una referencia al mismo estilo que jQuery:

```
Y.one('#test').setStyle('display', 'none')
```

También podemos seleccionar un conjunto o una lista utilizando el método:

```
Y.all('.test')
```

Cuando tengamos un nodo seleccionado, le podremos pedir cosas y asignar cosas con los métodos ***get/set***. Por ejemplo, podemos pedir el ID haciendo ***Y.one('#test').get('id')*** o podemos hacer ***Y.one('#test').get('parentNode')***, que nos devolverá un nuevo nodo que será el padre. También podemos setear propiedades como el html de un nodo haciendo ***Y.one('#test').set('html', 'blablabla')***.

Lectura recomendada

Toda la referencia de la clase nodo está en:
http://developer.yahoo.com/yui/3/api/node.html.

6.2. Sistema de eventos

Para añadir un evento escuchador (***listener***) a una instancia de nodo, se debe usar el método, en el que podemos hacer lo siguiente:

```
Y.one('#demo').on('click', function(e) {
   e.preventDefault();
     alert('event: ' + e.type);
   });
```

Por otro lado, también podemos utilizar la función *delegate*, que nos capturará eventos que pasen dentro del nodo, así:

```
Y.one('#demo').delegate('click', function(e) { });
```

Si #demo fuera un ul y dentro existiesen elementos, li, la función nos capturaría clics dentro de esta jerarquía.

Para eliminar un ***listener*** lo haremos con el método *deatach*:

```
Y.one('#demo').deatach('click');
```

Podemos desencadenar eventos desde el programa (simularlos), haciendo:

```
Y.one('#demo').simulate('click');
```

O podemos lanzar eventos propios con:

```
Y.one("#test").fire("events:elmeuevent");
```

Finalmente:

```
Y.one("#test").on('touchstart', function() {  });
```

Cabe mencionar que podemos utilizar eventos táctiles, puesto que están presentes.

6.3. Herramientas y utilidades

Seguramente YUI es el *framework* de desarrollo más completo pues contiene muchísimas utilidades y módulos. Repasaremos cómo hacer peticiones con AJAX y cómo construir pequeñas animaciones.

6.3.1. Peticiones AJAX

Las peticiones AJAX se realizan utilizando el módulo:

```
Y.io('/url/de/lallamada', {
       method: 'POST',
       // serializamos el formulario
   form: {
      id: "elform", useDisabled: true },
   // eventos
       on: {
        complete: function (id, response) {
              // handler de la respuesta
   }
    }
});
```

En este caso, serializamos el formulario #elform, hacemos una petición para ***POST*** y, cuando obtengamos la respuesta, la procesamos en la función del ***handler***.

Como veis, prácticamente siguen la misma estructura que la Prototype o la jQuery, jugamos con configuraciones y parámetros nuevos.

6.3.2. Animación

La YUI también tiene un módulo de animación.

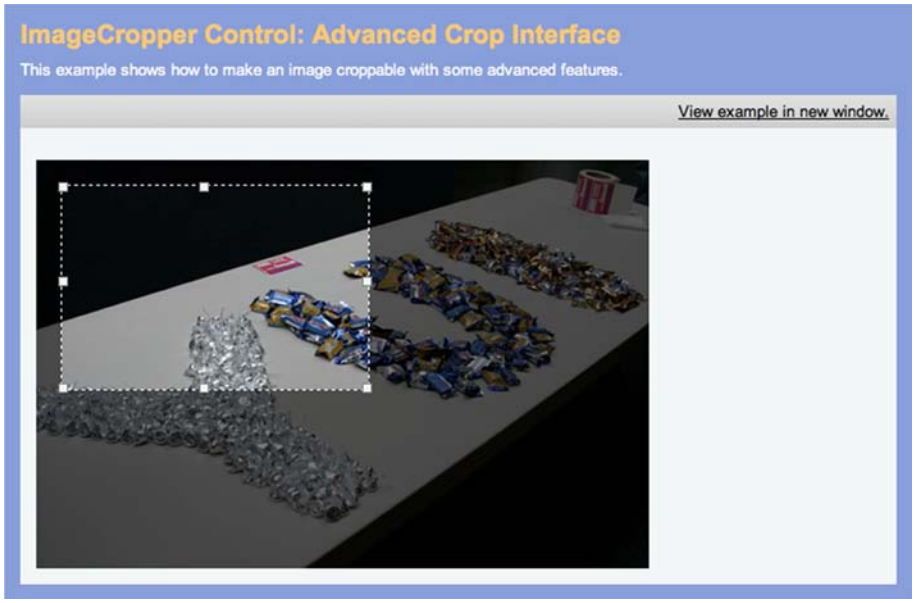
```
var animation = new Y.Anim({
   nodo: '#my-div',  // selector que animar
   // origen animación
   from: {
      height: 0,
      width: 0
   },
   // destino
   to: {
      height: 100,
      width: 100
   },
   duration: 0.5, // duración
```

```
    easing: Y.Easing.easeOut // tipo de interpolación de movimiento
});


animation.run();
```

Todas las propiedades de la animación pueden ser cambiadas utilizando ***getters*** y ***setters***. También hay que decir que las animaciones desencadenan eventos que pueden ser capturados.

6.4. Componentes (*widgets*)

Realmente la YUI destaca por ser una completa biblioteca con multitud de componentes. Entre ellos destacan el campo de autocompletado, el editor de texto enriquecido, un sistema para hacer gráficos y una herramienta para recortar imágenes, entre muchos otros.



7. MooTools

Tal y como mencionan en su web, MooTools es un ***framework*** Javascript modular, compacto y orientado a objetos. A diferencia de jQuery, de la que muchos dicen que implementa un DSL (***domain specific language***), la MooTools utiliza sólo Javascript, aumentándolo con herramientas potentes y sólidas.

Muchas veces la comparan con la **Prototype**, puesto que las dos basan su potencia aumentando los objetos nativos a través de la cadena de prototipaje. MooTools, sin embargo, queda mucho más compacta y modular.

MooTool significa ***my object oriented javascript tools***.

Web recomendada

Podemos descargar la librería de:

http://mootools.net/download

Si queremos hacer una descarga personalizada con componentes señalados, lo podemos hacer de:

http://mootools.net/more/

7.1. Trabajando con el DOM

Toma directamente las ideas de la Prototype, y así tenemos dos funciones básicas: ***$('id')***, que nos sirve para seleccionar nodos para ID, y la ***$$('css')***, que nos permite usar selectores CSS.

Por ejemplo:

$$('E F')	Selecciona elementos F que son hijos de E.
$$('E[foo="bar"]')	Selecciona elementos E que contengan un atributo foo=="bar".
$$('E:first-child')	Selecciona elementos E que son el primer hijo de sus padres.

La biblioteca nos ofrece herramientas para desplazarse a través de los diferentes nodos. Así, una vez seleccionado un nodo, disponemos de los métodos ***getParent()*** y ***getParents()***. El primero nos devuelve el padre, y el segundo, una lista de todos los padres hasta la raíz.

Al mismo tiempo, disponemos de los métodos ***el.getElement(selector)*** y ***el.getElements(selector)***, que nos permiten recuperar los hijos del nodo que cumplan el selector. También disponemos de otros métodos, como ***el.getFirst(tag_name)*** o ***el.getLast(tag_name)***, que devuelven el primer hijo y el último que cumplan el *tag* definido como parámetro.

Para manipular los estilos CSS de un nodo, disponemos también de métodos, así: ***el.hasClass('nombre')*** consulta si el nodo tiene asignada la clase CSS. También podemos hacer un ***el.addClass('nombre')***, obviamente añadirá la clase nombre, o ***.removeClass('borders')***, lo que eliminará la clase *borders*. Así:

```
$$('li.foo').addClass('bordered');
$$('li.foo').removeClass('bordered');
```

También nos ofrece el método ***toggleClass('class')*** para, de manera alternativa, poner o sacar la clase en cuestión.

Podemos también manipular directamente los estilos asociados con un nodo utilizando la función ***$$('li.bar').setStyle('font-weight: bold');*** o del mismo modo el ***setStyles***, que admite un objeto de parámetros. También el ***getStyle*** y ***el getStyles***.

Disponemos también de un sistema para trabajar con los atributos (o propiedades) de los nodos. Así, podemos ***.getProperty('type')*** o ***setProperty('type')***, que nos mirará o seteará el atributo tipo de un campo de formulario. En este caso, también disponemos de las versiones multiples ***setProperties***, y de ***removeProperty()*** para eliminar propiedades.

Como el resto de las bibliotecas, desde Mootools también podemos crear/inyectar elementos en el DOM. Así:

```
var item1 = document.newElement('li',
{ class: 'bordered',
text: 'Second list item'
});
```

Generará un elemento que tendremos preparado para inyectarlo. Para hacerlo:

```
item2.inject(item1, 'inside');
```

También podremos usar estas otras fórmulas equivalentes a la anterior:

.injectBefore(el),.
injectAfter(el),.injectTop(el),.injectBottom(el),.injectInside(el)

Podemos clonar nodos con ***$('id').clone()***. Los podemos sustituir:

```
var new_title = new Element('span', { text: 'New sublist' });
new_title.replaces(span1);
```

O les podemos hacer una envoltura (*wrap*):

```
var span1 = $$('#div4>ul>li>span')[0];
var strong = new Element('strong');
strong.wraps(span1, 'top');
```

También podemos eliminar nodos con los métodos ***.destroy()***, que elimina el nodo seleccionado, y el ***.empty()***, que nos limpiará todo el árbol de hijos.

7.2. Sistema de eventos

Las herramientas de MooTools para gestionar los eventos son básicas. Fundamentalmente lo que hacen es parchear y solucionar las inconsistencias entre los diferentes navegadores. Así, podemos capturar un evento de un nodo:

```
var first_handler = function(ev) {
   ev.stopPropagation();
   ev.preventDefault();
   var el = ev.target;
   el.toggleClass('clickcolor');
};
$('link1').addEvent('click', first_handler);
```

Mediante el ***stopPropagation*** rompemos la cadena de propagación y el ***preventDefault*** cancela el comportamiento por defecto del evento. La propiedad Event.target da acceso al elemento involucrado en el evento.

Podemos añadir múltiples eventos a un elemento o a una lista utilizando el método ***addEvents***, y usando como parámetro un objeto tipo ***propiedad/función manipuladora***:

```
{
    'click': function() {},
    'rollover': function() {}
}
```

Para eliminar un evento o una lista de eventos, podemos usar ***removeEvent('click', handler)*** o ***removeEvents('click')***, sin necesitar una referencia en el manipulador.

7.3. AJAX

MooTools tiene una clase Request que sirve para hacer peticiones AJAX. Para utilizarla:

```
var req1 = new Request({
   method: 'GET',
   url:   'data1.txt',
   onRequest: function() {
      $log("*** Firing request for " + this.options.url);
   },
   onSuccess: function(result_text, result_xml) {
      $log("*** Loaded " + this.options.url);
```

```
        $log("*** Data: " + result_text);
    },
    onFailure: function() {
        $log("*** Request failed: " + this.status);
    },
    onException: function() {
        $log("*** Exception occurred!");
        $log(arguments);
    })
req1.send();
```

Las dos primeras opciones del objeto parámetros facilitan el método de realizar la petición y la url que se va a pedir. El resto son *handlers* que se desencadenarán en los eventos que se generarán. Recordemos que el Javascript es asincrónico (el hilo de ejecución no espera).

Cabe destacar que para todos los manipuladores la referencia ***this*** apunta al objeto Request.

También podemos efectuar una petición con AJAX utilizando el sistema de eventos:

```
var req2 = new Request();
req2.addEvent('success', function(txt, xml) {
    $log("*** Data loaded: " + txt);
})
req2.addEvent('success', function(txt, xml) {
    $log("*** Me too! Data loaded: " + txt);
});
req2.GET('data1.txt', { /* options */ })
```

Cuando la petición se complete, se ejecutarán las dos funciones.

7.4. Animación

Podemos manipular la posición de un objeto, un nodo con propiedades absoultas CSS, con las funciones ***$('#exemple').position({ x: 150, y: 10 })***, y también podemos consultar la posición actual con ***el.getPosition()***, que devuelve un objeto de {x,y}.

Conseguiremos hacer animaciones con el módulo ***Fx.Tween:***

```
var myFx = new Fx.Tween('myElement', {
    duration: 'long',
    transition: 'bounce:out',
    property: 'height'
});
```

```
myFx.start('left', 0, 300)
```

Generamos una nueva instancia de animación, le damos los parámetros de duración y el método de interpolación del movimiento, después utilizamos el método ***start*** para dirigir la animación desde la coordenada x:0 hasta la cordenada x:300.

Pero también existe un acceso directo a la función y que cuelga de la ***Class Element***. Recordemos que la función $() devuelve instancias.

```
$('myElement').tween('width', '100').
   addEvent('onComplete', function() {
   alert('fi');
});
```

Las animaciones también responden a eventos. En este último ejemplo, cuando acabe la animación, ejecutamos una alerta de sistema.

Otra funcionalidad importante es la posibilidad de encadenar animaciones con el método ***chain():***

```
$('myElement').tween('left', '100')
   .chain(function() {
      $('myElement').tween('top', '200')
   })
```

Primero movemos el objeto hasta la coordenada 100, después lo reposicionamos hasta la 200 vertical.

7.5. Componentes (*widgets*)

Del mismo modo que en la biblioteca jQuery, en su núcleo no existen componentes, pero su naturaleza compacta y modular y la posibilidad de exhibirlos en la propia página de la librería ha provocado que haya muchos desarrolladores programando componentes para la librería.

Web recomendada

Podemos ver ejemplos aquí: http://mootools.net/forge/

Personalmente destacaría el árbol de archivos, un sistema de plantillas, un sistema para recortar imágenes y otro para hacer los *zoom*.

8. Librerías Javascript para canvas

8.1. Introducción

8.1.1. ¿Qué es el objeto canvas?

Como su nombre indica, es el equivalente a una tela de pintor donde dibujar (***canvas***). Así, empleando Javascript podremos pintar y dibujar, crear objetos vectoriales, añadir imágenes y textos y capturar eventos.

> De hecho, podemos decir que con el objeto *canvas* tenemos un potente punto de partida para generar animaciones e interactividad.

De entrada, ya vemos aplicaciones que nos permiten pintar gráficos de datos, otras que hacen dibujos vectoriales e incluso juegos interactivos en 2 y 3 dimensiones.

Pero no todo es bueno, puesto que el soporte de este en la familia de **Internet Explorer** no aparece hasta la revisión 9 (a pesar de que se puede emplear con librerías que simulen la API), y que en cuanto a herramientas de programación para la interacción es bastante primario. No disponemos de una **API** avanzada que nos permita trabajar con capas/objetos/eventos/*sprites*, como permiten otras tecnologías similares, como **Silverlight** o **Adobe Flash**.

Así, lo más adecuado para trabajar con aplicaciones es construir o emplear una librería que nos simplifique el proceso de generación de código. Seguramente la mejor manera de pensar en el objeto *canvas* es como un elemento que nos puede permitir sustituir al Flash en su versión primaria, cuando todavía no era un **potente *framework* de desarrollo de aplicaciones web**.

8.1.2. Canvas sin nada

Para poder explorar cómodamente los diferentes beneficios de emplear diferentes librerías para canvas, en primer lugar, veremos cómo se utiliza sin ninguna librería, usando directamente la **API** que nos ofrece el navegador. Así, empezaremos generando un nuevo objeto canvas con la etiqueta:

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

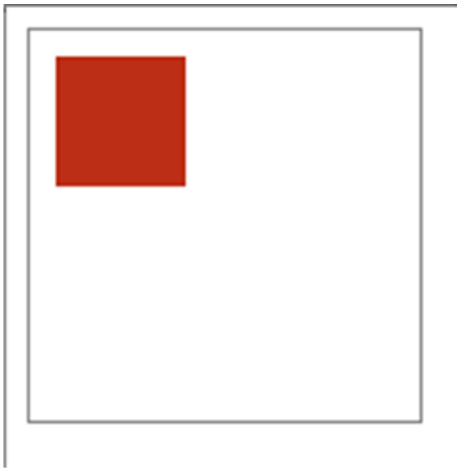
Con esta etiqueta html, generaremos un objeto de dibujo cuadrado de 150 píxeles de ancho. Como todos los elementos (objetos) html, lo podemos decorar con estilos. Así, le definiremos un filete de 1 píxel de color gris con:

```
<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>
```

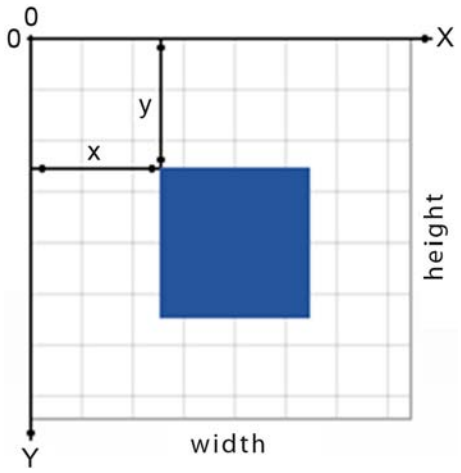
A partir de aquí, ya podemos empezar a emplear métodos del objeto canvas. Con la solicitud ***fillStyle***, definiremos el tipo de estilo de contenido, y con ***fillRect***, generaremos un cuadrado en las coordenadas 10,10, y con 50 píxeles de anchura y altura, resultando:



8.1.3. Dibujar cosas

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);
});
```

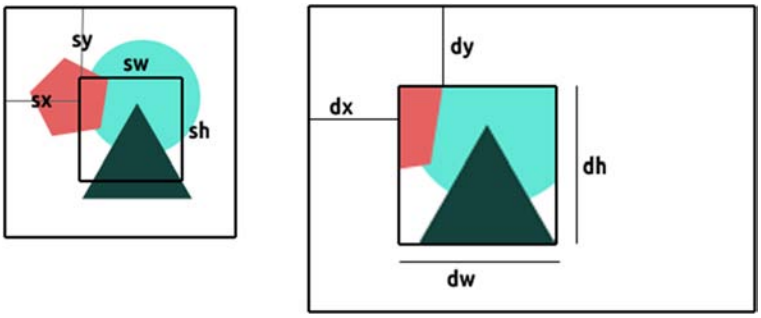
Primero creamos un objeto tradicional imagen con Javascript para utilizarlo como receptor de la descarga. Mediante el image.src, inicializamos la descarga. Finalmente, con la función $(image).load programamos el evento onLoad en la imagen, que nos la dibujará (colocará) en el objeto canvas. Podemos **escalar 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
```

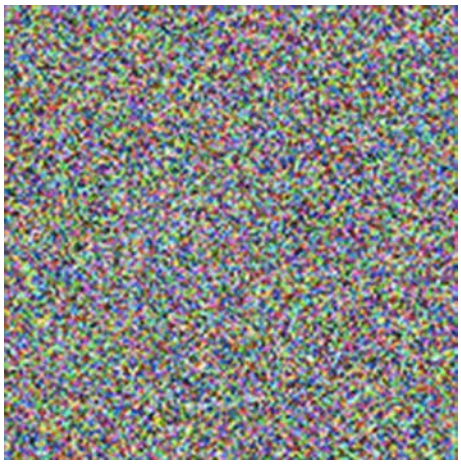
El componente ***a[5] a a[8]*** será el siguiente píxel. Cabe decir también que en este array no hay filas ni columnas, y por tanto, las deberemos deducir nosotros a partir del ***width*** de imagen.

De una manera sencilla, podemos confeccionar el típico efecto de ruido generando una imagen desde 0 que escribiremos en el objeto canvas con:

```
<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('2de');
   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>
```

El resultado será una animación como de televisión desintonizada:



8.1.4. 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.

8.1.5. 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.

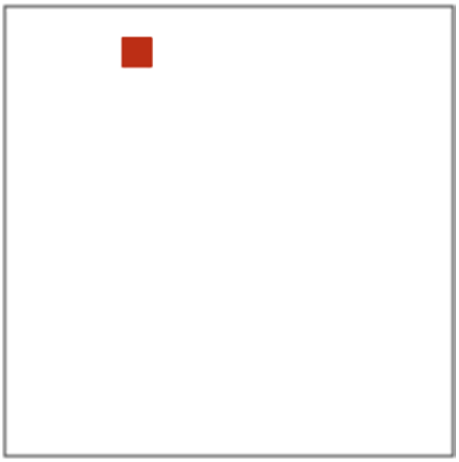
Volveremos al paso 1.

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>
```

El resultado será:



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.

8.2. Raphaël.js

Es una de las primeras librerías gráficas de vectores para la web, independiente del **canvas**. También funciona en Internet Explorer 6. Es una buena solución para crear pequeños componentes interactivos. Viene a ser una mezcla de las funcionalidades del canvas con el svg, pero en esencia el documento es muy interesante.

Web recomendada

Podemos consultar su web: http://raphaeljs.com.

8.2.1. Inicialización

Para poder utilizar la librería, conviene descargarla de su web y después incluirla en el html. También la podemos emplear para hacer pruebas mediante el jsfiddle.com.

Una vez incluida en el html, es necesario que inicialicemos un nuevo objeto de tipo Raphael. El código nos puede quedar del siguiente modo:

```
<html>
    <head>
      <title>Raphael Play</title>
      <script type="text/javascript" src="raphael.js"></script>
<script>
window.onload = function() {
   element = document.getElementById('container')
   var paper = new Raphael(element, 500, 500);
}
   </script>
    </head>
   <body>
```

```
        <div id="container" style="border:1px solid grey"> </div>
    </body>
</html>
```

Si nos fijamos, el objeto Raphael recibe un elemento del DOM en el que insertaremos el elemento canvas. En este caso, lo efectuamos con Javascript estándar empleando la función, ***document.getElementById(id)***, pero también se podría hacer con cualquier librería, tipo ***jQuery, prototype***, u otras.

Si quisiéramos, también podríamos insertar directamente el elemento en el DOM, empleando en vez de una referencia a un elemento del **DOM**, las coordenadas en las que lo queremos, así:

```
var paper = new Raphael(0,0, 500, 500);
```

8.2.2. Dibujo y formas

Disponemos de una sencilla y completa API para generar formas y dibujar caminos (***paths***). Así, con:

```
paper.circle(x,y,radi);
```

Dibujaremos un círculo en las coordenadas ***x,y*** y del correspondiente ***radi***.

```
paper.rect(x,y,width, height, angle)
```

Dibujaremos un rectángulo en el que ***x,y*** son las coordenadas iniciales, ***width*** es el ancho, ***height*** es la altura, y el último parámetro, que es opcional, es el ángulo de las esquinas.

```
var c = paper.ellipse(x,y, rh, rv)
```

Nos permitirá generar elipsis en las que x,y es el punto inicial, rv el radio vertical y rh el radio horizontal.

Un aspecto importante es que los diferentes pedidos que generemos devuelven punteros al objeto creado. Así, en la última, podemos acceder al objeto elipse y, por ejemplo, generarle un color de cuerpo con:

```
c.attr({'fill':'red'});
```

Mediante los atributos, podemos asignar y cambiar multitud de parámetros a una lista de objetos.

fill	Un color o un gradiente para realizar el relleno. linear gradiente: "‹angle›-‹colour›[-‹colour›[:‹offset›]]*-‹colour›", ejemplo: "90-#fff-#000" – 90° degradado de blanco a negro o "0-#fff-#f00:20-#000" – 0° degradado de blanco a rojo (y 20%) a negro. degradado radial: "r[(‹fx›, ‹fy›)]‹colour›[-‹colour›[:‹offset›]]*-‹colour›", ejemplo: "r#fff-#000" – degradado de blanco a negro "r(0,25, 0,75)#fff-#000" – degradado de blanco a negro con un punto de foco a 0,25, 0,75. Las coordenadas de punto de foco son con rango de 0..1. Los degradados radiales sólo pueden ser aplicados a círculos y elipsis.
fill-opacity	De 0 a 1. Porcentaje de opacidad del objeto.
font **font-family** **font-size** **font-weight**	Nombre del tipo de letra asignado. Familia del tipo de letra. Tamaño del tipo de letra. Tipo de letra gruesa ***bold***.
height **width**	Tamaño horizontal y vertical.
href	Si se especifica, enlace del objeto.
opacity	De 0 a 1. Porcentaje de opacidad del objeto.
path	pathString SVG path string format
r	Radio del objeto.
rotation	Rotación del objeto.
scale	Escala del objeto.
src	En el caso de imágenes, fuente de la imagen.
stroke **stroke-opacity stroke-width**	Color del filete. Opacidad del filete. Ancho del filete.
x, y	Coordenadas x,y del objeto.

Web recomendada

Podemos visualizar toda la referencia de propiedades y métodos de la librería Raphael en su web:

http://raphaeljs.com/reference.html

Dada una referencia a un objeto, también lo podemos manipular directamente con determinados métodos disponibles. Así, lo podemos esconder con ***c.hide()*** o mostrar con ***c.show()***. Lo podemos rotar o escalar con ***c.rotate(angle)*** o ***c.scale(sx,sy)***, donde sx y sy son porcentajes de 0 a 2 de escalado de la vertical y la horizontal.

También podemos mover el objeto a otra posición del canvas con el método ***c.translate(x,y)***.

Otra instrucción importante es el ***set***, que nos permite generar conjuntos de objetos o agrupaciones que podremos manipular de manera independiente. Así:

```
var st = paper.set();
st.push(
    paper.circle(10, 10, 5),
```

```
    paper.circle(30, 10, 5)
);
st.attr({fill: "red"});
```

Crearemos dos círculos que podrán ser manipulados desde la misma referencia st.

También podemos eliminar elementos creados con la instrucción **st.remove()**.

Los elementos tienen profundidad y tenemos dos órdenes para manipular su posición. De este modo, si llamamos al método ***.toFront()***, el elemento se nos posicionará en la parte superior de la pila de elementos, mientras que si lo llamamos con el método ***.toBack()*** el elemento se posicionará al fondo. También podemos generar los nuevos elementos en una posición, empleando los métodos ***insertBefore()*** e ***insertAfter()***.

Asimismo, podemos insertar textos con la instrucción ***.text(x,y,'cadena de texto')***. Para emplear fuentes, deberemos utilizar la tecnología cufon (https://github.com/sorccu/cufon/wiki/abou), que nos generará la fuente en formato de conjunto de caminos (***paths***) en ***json***.

Hemos de hablar, también, de la potencia de la instrucción ***.path()*** para generar recorridos. El formato de esta es el del svg (http://www.w3.org/tr/svg/paths.html#PathData).

```
paper.path('M10 10L90 90');
```

Generará una línea diagonal desde la coordenada 10,10 hasta la coordenada 90,90.

Y para finalizar la parte de dibujo de formas con Raphaeljs, cabe decir que podemos incluir imágenes dentro de un objeto canvas con la instrucción:

```
paper.image("apple.png", 10, 10, 80, 80);
```

Donde las dos primeras indican las coordenadas del punto de inserción, mientras que las segundas son el ancho y el alto.

8.2.3. Animación

Otro de los factores interesantes para usar Raphaël es la capacidad de generar animaciones complejas con una sola instrucción. Disponemos del método ***animate***, que nos permite generar animaciones de modo fácil y cómodo. Así:

```
var d = paper.circle(400, 40, 20);
d.animate({cy: 480, r: 20}, 2000, "bounce");
```

Generaremos un círculo de 10 píxeles de ancho en la posición ***10,10***. La animación que generaremos a continuación animará las propiedades a un radio de 20px, y la posición del centro x a 20 también, durante 2.000 milisegundos con una ecuación de movimiento ***"bounce"*** rebote. Así, la forma general, será:

```
c.animate({objecte_propietats}, duración, ecuación de movimiento, onEnd);
```

Donde la ecuación puede ser:

```
[">", "<", "<>", "backIn", "backOut", "bounce", "elastic", "cubic-bezier(p1, p2, p3, p4)"]
```

Las propiedades que podemos animar serán: ***clip-rect, cx, cy, fill, fill-opacity, font-size, height, opacity, path, r, rotation, rx, ry, scale, stroke, stroke-opacity, stroke-width, translation, width, x, y***.

También podemos definir un último parámetro que puede ser una función que queramos que se ejecute al finalizar la animación.

Finalmente, podemos realizar animaciones con fotogramas clave (***keyframes***). Animaciones en las que marcamos pasos para momentos de tiempo, por ejemplo:

```
c.animate({
    "20%": {cx: 20, r: 20, easing: ">"},
    "50%": {cx: 70, r: 120, callback: function () {...}},
    "100%": {cx: 10, r: 10}}, 2000);
```

Así, la anterior animación moverá el objeto hasta un radio de 20 durante el 20% del tiempo, o sea 400 milisegundos, mientras que durante 600 lo hará crecer a 120, mediante la ejecución del *callback* al llegar. Finalmente, los últimos 1.000 milisegundos evolucionará hacia el punto de inicio.

8.3. EaselJS

Esta es una librería para canvas relativamente nueva pero con muy buena perspectiva y una sólida base técnica detrás. Fue creada por gskinner.com, como ayuda al desarrollo del juego en Html5, Pirates Love Daisies.

La idea fundamental de la librería es añadir al componente **canvas** métodos para interactuar con objetos sobre este, al estilo del Flash y el Actionscript. Así, lo que han empezado a hacer es reescribir la estructura fundamental del ***Document Object Model del Flash y Actionscript***, empleando como sistema de *render* el ***Canvas HTML5***.

No es objeto de este módulo exponer cómo funciona la estructura de objetos de Actionscript 3, ni lo potente que es para generar estructuras visuales complejas, así como juegos y otros. Por lo tanto, lo que haremos en este manual es enumerar las principales funcionalidades que han ido aparecido con el easelJS.

8.3.1. *Classes* clave

- ***DisplayObject***

Clase abstracta para todos los elementos visuales o insertables en el canvas. Proporciona acceso a todas las propiedades (x,y, rotation, scaleX, scaleY, skewX, skewY, alpha, shadow, etc.) comunes a todos los objetos visuales.

- ***Stage***

Es el nivel inicial que contiene todos los elementos *display*. Cada vez que se llama a la función *update* en el *stage*, él se encarga de renderizar todos los elementos pegados en el canvas.

- ***Container***

Es una clase contenedora que permite trabajar con diferentes objetos dentro de uno solo y manipularlos como un solo texto.

- ***Text***

Dibuja texto en el contexto de un objeto *display*.

- ***Bitmap***

Dibuja una imagen o vídeo en el canvas y dispone de las propiedades heredadas de *DisplayObject*.

- ***BitmapSequence***

Dibuja una secuencia de imágenes (***sprites***) diferentes o fotogramas en una rejilla, y dispone de una pequeña API para manipular la reproducción de la secuencia.

- ***Graphics***

Una potente API para renderizar objetos gráficos en el canvas.

- ***Shape***

Renderiza gráficos vectoriales, como el objeto *display list*, con las propiedades del anterior.

8.4. CanvaScript

Al estilo de jQuery, CanvaScript nos ofrece un método que nos sirve de herramienta para inicializar la librería y también como método para dibujar las operaciones en el canvas. Para empezar a utilizarlo, podemos emplear el siguiente ejemplo:

Web recomendada

http://jcscript.com/

```
<html>
<head>
   <title>Ejercicio jcanvascript</title>
   <script type="text/javascript" src="jCanvaScript.js" > </script>
   <script type="text/javascript">
   window.onload = function()
   {
      jc.start('ejercicio');
      jc.circle(50,50,50,'rgba(255,255,0,1)',1);
      jc.start('ejercicio');
   }
   </script>
</head>
<body>
   <canvas width="150" height="150" style="border:1px solid grey" id="ejercicio"> </canvas>
</body>
</html>
```

Incluimos la librería y la inicializamos con la función ***jc.start('id_delcanvas')***. A partir de aquí, dibujamos un círculo con el método ***jc.circle***. Finalmente, volvemos a llamar al método *start* para pintar las manipulaciones hechas.

Una de las características importantes de la librería es que nos permite trabajar de manera encadenada. Así, una vez generemos una primitiva (elemento) le podemos asignar eventos, cambiar propiedades o dar un ID de manera fluida sin necesidad de volver a seleccionar el objeto y de manera encadenada. Así, podemos encontrar construcciones de código como esta:

```
   jc.start('ejercicio', 25);
   jc.circle(50,50,20,'rgba(255,255,0,1)',1)
      .draggable()
      .click(function(){
         alert('hola')
      });
```

Seleccionamos un objeto canvas, le dibujamos un círculo que convertimos en arrastrable y le programamos un evento clic que generará una notificación.

La librería CanvaScript nos ofrece una serie de primitivas que nos permiten dibujar multitud de formas. Básicamente aumenta y mejora la **API** por defecto del objeto **canvas.** Así, podemos:

```
jc.circle(x, y, radius, [color], [relleno])
```

Dibuja un círculo

```
jc.rect(x, y, width, height, [color], [relleno])
```

Dibuja un cuadrado.

```
jc.arc(x, y, radius, iAngle, fAngle, [agujasreloj], [color], [relleno])
```

Dibuja un arco desde iAngle hasta fAngle. En la dirección de las agujas del reloj si agujasreloj es *true* o al revés si *false*. El color del filete y el color del relleno son parámetros opcionales.

```
jc.line(points, [color], [relleno])
```

Dibuja una línea. El primer parámetro es una lista de pares de puntos en la forma [[0,0], [0,10]].

```
jc.qCurve(points, [color], [relleno])
```

Dibuja una curva con ecuación cuadrática.

```
jc.bCurve(points, [color], [hijo])
```

Dibuja una curva de Bézier.

```
jc.imageData(float width, float height)
```

Crea una nueva imagen de ancho y de alto.

```
jc.image(img, sX, sY, sWidth, sHeight, dX, dY, dWidth, dHeight)
```

Es un clon de la función *image* nativa de canvas.

```
jc.text(text, x, y, [maxWidth], [color], [hijo])
```

Clon de la función texto de canvas con el color y el relleno extras.

```
lGradient(x1, y1, x2, y2, colors)
```

Dibuja un degradado lineal desde x1,y1, hasta x2,y2, en el que colores es una lista de porcentajes de cada color.

```
[ [0,'rgb(0,0,0)'],
  [0.5,'rgb(255,0,0)'],
  [0.75,'rgb(255,255,0)'] ];
```

En él, cada elemento de la lista es una pequeña lista con el porcentaje y el color.

rGradient(x1, y1, radius1, x2, y2, radius2, colors)

Dibuja un degradado de forma radial.

```
    jc.start(idCanvas);
    var colors=[[0,'rgb(0,0,0)'],
                [0.5,'rgb(255,0,0)'],
                [0.75,'rgb(255,255,0)']];
    var gradient=jc.rGradient(10,30,20,10,30,250,colors);
    jc.rect(1,1,248,263,gradient,1)
    jc.start(idCanvas);
```

jc.layer(idLayer)

El comando *layer* genera un conjunto de objetos que pueden ser tratados como unidad. Por ejemplo, el siguiente código:

```
jc.layer('grupo1')
    .animate({translate:{x:100}},2000)
    .draggable()
    .clone('2')
    .animate({translate:{x:100,y:140}},1500);
```

Utilizamos el conjunto ***grupo1*** y le realizamos una animación, lo convertimos en arrastrable y finalmente lo clonamos.

```
jc.addObject(name, parameters, fn)
```

Permite generar nuevas primitivas a partir de un nombre, una lista de parámetros y una función de dibujo.

Vistos estos objetos (en el lenguaje de su documentación, primitivas), podemos observar toda una serie de métodos que encontramos disponibles en cada objeto una vez creado o si lo seleccionamos. Una vez dibujamos un objeto, le podemos asignar un ID con la función ***id()***, que después podremos usar para seleccionar el objeto en cuestión.

```
.animate(parameters, [duration], [easing], [onstep], [fn])
```

Seleccionado un objeto, podemos asociar y generar una animación con:

```
jc.circle(50,50,20,'rgba(255,255,0,1)',1)
   .animate({x:175,radius:5,color:'#000000'},1000);
```

Generaremos una animación del círculo cambiando su coordenada x, el radio y el color. La duración será en milisegundos, y podemos asignar una función de movimiento y eventos al finalizar y en cada paso.

```
.opacity([float])
```

Cambiamos la opacidad de un objeto. También nos permite consultar la existente.

```
.shadow({ x:5, i:5, blur:15, color:'#ff0000' })
```

Permite asignar una sombra.

```
.visible()
```

Cambia la visibilidad de un objeto.

```
.id()
```

Asigna o consulta el ID de un elemento.

```
.name()
```

Asigna un nombre que después podremos utilizar en el estilo de los selectores de clase de CSS.

```
.down() / .up() / .level()
```

Maneja la profundidad de los objetos; ***down()*** lo hace bajar; ***up()*** lo sube, y **level()** nos devuelve la profundidad. Esta siempre se calcula relativa al total de objetos generados en un espacio.

```
.rotate() / .scale() / .transform()/ .translate() / .translateTo()
```

Nos permite alterar y cambiar propiedades del objeto seleccionado.

```
.attr()
```

Permite cambiar atributos de un objeto generado, como por ejemplo:

```
.attr('fill',0);)
```

```
.buffer()
```

Devuelve una referencia al objeto en cuestión. Con esta, lo podemos utilizar para escribirlo como imagen.

```
.clip(objecte)
```

Nos permite generar un objeto que enmascarará a otro.

```
.clone()
```

Clona el objeto seleccionado, es decir, genera una copia.

```
.del()
```

Elimina el objeto seleccionado.

```
isPointIn(x,y)
```

¿Está el objeto sobre el punto x,y?

```
color()
```

Nos permite cambiar el color del relleno de un objeto.

```
lineStyle()
```

Nos permite cambiar el estilo de línea de un objeto.

Eventos

También podemos asignar eventos, en el estilo jquery, a cualquier objeto, mediante la llamada al nombre del evento.

```
click()
dblclick()
mousedown()
mousemove()
mouseout()
```

```
mouseover()
mouseup()
blur()
focus()
keydown()
keypress()
keyup()
```

En la forma:

```
objecte.click(function(){  })
// Dentro de la función, el this se refiere al objeto que ha generado el evento.
draggable() / droppable()
```

Nos permite transformar un objeto en arrastable y destino de ser arrastrado, así:

Mientras que la primera actúa como activador del arrastable, la segunda actúa como asignador de evento. Así, por ejemplo:

```
jc.id('#ele').droppable( function() {
   // Desencadena el evento pertinente.
 })
```

8.5. Processing

Processing nace como un ***puerto*** del lenguaje de creación artística para Java en Javascript, por parte del mismo creador que ***jQuery***, **John Resig (www.ejohn.com)**.

En este manual sólo lo mencionaremos, puesto que la base del sistema es el objeto canvas; pero no lo trataremos porque el *processing* es un lenguaje en sí mismo. Así, el puerto en Javascript es un intérprete del lenguaje *processing* que establece un sistema para procesar los *scripts*.

Webs recomendadas

Podemos encontrar ejemplos y ver cómo funciona en las webs del propio lenguaje:
http://www.processingjs.com
http://www.processing.com

8.6. Ejercicio

Reloj con canvas

```
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <title>Reloj Javascript</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"> </script>
<script src="raphael-min.js"> </script>
<script type="text/javascript">
<!--
```

```
$(document).ready(function(){
   function Rellotge()
   {
      $(document).trigger('tiempo', [new Date()]);
      setInterval(function(){
         $(document).trigger('tiempo', [new Date()]);
      }, 1000);
   }
   var TextVisor = function() {
      var t = this.t = this;
      var tick = function(event, extra) {
         t.pinta( event, extra );
      }
      this.pinta = function(event, extra) {
         $( '#rellotge' ).html( t.format_string(extra) );
      }
      this.format_string = function(dat) {
         d = dat;
         hora = d.getHours();
         minuto =d.getMinutes();
         if(minuto<=9)
            minuto = "0"+minuto;
         segundo = d.getSeconds();
         if(segundo<=9)
            segundo = "0"+segundo;
         return hora + ":" + minuto + ":" + segundo;
      }
      $(document).bind('tiempo', tick);
   }

   var RaphaelVisor = function(object_id)
   {
      var t = this.t = this
      var pt = this.pt = new Raphael(document.getElementById(object_id), 200, 200);
      // dibujamos el reloj inicial
      var PX = 100;
      var PY = 100;
      var tick = function(event, d)
      {
         h = d.getHours();
         if(h>12)
            h = h/2
         m =d.getMinutes();
         // añadimos a la hora la parte proporcional de minutos
         h += (m*100/60)/100
         // transportamos hora a ángulos
         s = d.getSeconds();
```

```
            ah = (h*360)/12
            am = (m*360)/60
            as = (s*360)/60
            neteja()
            //console.info(h,m,s,">>> ", ah, am, as)
            t.hora = pinta_agulla( 50, ah)
            t.minut = pinta_agulla( 60, am)
            t.segon = pinta_agulla( 70, as)
            t.hora.attr(   {'stroke-width': 7, 'stroke': '#ac360b' })
            t.minut.attr(   {'stroke-width': 5, 'stroke': '#a0573c' })
            t.segon.attr(   {'stroke-width': 3, 'stroke': '#666666' })
        }
        var neteja = function()
        {
            if(t.hora)
                t.hora.remove();
            if(t.minut)
                t.minut.remove();
            if(t.segon)
                t.segon.remove();
        }
        var pinta_agulla = function ( radio, an )
        {
            // convertimos ángulo a radianes
            angle =  (an-90)*(2*Math.PI/360);
            // calculamos coordenadas x, y
            x = PX + Math.cos(angle)*radi
            y = PY + Math.sin(angle)*radi
            svg = "M"+ PX + " " + PY + " L" + x + " " + y
            return pt.path(svg)

        }
        this.init=function() {
            fons = pt.circle(100,100, 90)
            fons.attr({'fill': '#fbf8e1'})

            $(document).bind('tiempo', tick);
        }
    }
    // inicializamos reloj
    Rellotge();
    // Este tercer visor conecta el reloj con la ventana del navegador.
    var vis3 = new TextVisor();
    vis3.pinta = function(ev, extra){
        $(document).attr('title',
            this.format_string(extra) );
    }
```

```
    // inicializamos reloj creado con librería Raphael
    vi = new RaphaelVisor('rellotge')
    vi.init()
});
//-->
</script>
</head>
<body>
    <div id="rellotge" style="width:200px; border:1px solid grey"> </div>
</body>
</html>
```

9. Ejercicios

Construiréis un reloj de forma modular. En primer lugar, generaremos un pequeño núcleo que desencadenará un evento que informe del paso del tiempo. A este evento le añadiremos tres componentes diferentes que generarán visualizaciones de la hora actual:

```
<html>
<head>
   <title>Reloj Javascript</title>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js"> </script>
<script type="text/javascript">
<!--
// Emplead el evento de DOM disponible
$(document).ready(function(){
   /*
   Cread un primer objeto que os generará
   los eventos de tiempos. A él conectaréis los diferentes visores de relojes.
   */
   function Rellotge()
   {
      // generad un primero evento
      $(document).trigger('tiempo', [new Date()]);
      // haced que cada segundo ejecute un evento de tiempo.
      setInterval(function(){
         $(document).trigger('tiempo', [new Date()]);
      }, 1000);
   }

   // Generad una "clase" texto visor,
   var TextVisor = function() {
      // Cread un puntero en el this del objeto para poder
      // usarlo en la función que se llama desde el evento, puesto que esta
      // tiene el this del evento y no de la clase. Como es una clouser
      // t es visible en la función.
      var t = this.t = this;
      // esta función es a la que se ha de conectar el evento, y
      // desde la que ejecutaréis el manipulador
      var tick = function(event, extra) {
         t.pinta( event, extra );
      }
      // esta función renderiza el resultado del evento
      this.pinta = function(event, extra) {
         $( '#rellotge' ).html( t.format_string(extra) );
```

```
        }
        // esta función nos ayuda, dado una dato, a pintarla en formato
        // correcto 00:00:00
        this.format_string = function(dat) {
            d = dat;
            hora = d.getHours();
            minuto =d.getMinutes();
            if(minuto<=9)
                minuto = "0"+minuto;
            segundo = d.getSeconds();
            if(segundo<=9)
                segundo = "0"+segundo;
            return hora + ":" + minuto + ":" + segundo;
        }
        // conectad el evento del Reloj
        $(document).bind('tiempo', tick);
    }

    // Pintad el div con algunos estilos
    $('#rellotge').css({
        'font-size': '86px',
        'font-family': 'Arial',
        'text-align': 'center',
        'padding-top': '200px'
    });
    // Cread una instancia del visor, este
    // funcionará con el método por defecto que pinta el reloj
    var visor = new TextVisor();
    // Una nueva instancia, que utiliza el valor del desplegable,
    // os mostrará la fecha en la zona horaria seleccionada.
    var vis = new TextVisor();
    vis.pinta = function(ev, extra) {
        // convertimos la fecha a milisegundos para unificar las conversiones
        // y le restamos la diferencia horaria de nuestro país.
        temps = extra.getTime()-3600000;
        // Recuperamos la diferencia horaria del desplegable
        zona = Number( $('#timezone').val() );
        // la sumamos a la hora actual
        temps += 3600000 * zona;
        $('#camp_form').val( this.format_string(new Date(temps)) );
    }

    // Este tercer visor conecta el reloj con la ventana del navegador.
    var vis3 = new TextVisor();
    vis3.pinta = function(ev, extra){
        $(document).attr('title',
            this.format_string(extra) );
```

```
    }
    // inicializamos reloj
    Rellotge();
});
//-->
</script>
</head>
<body>
    <div id="rellotge"></div><br /><br />
    <div style="text-align:center">
       <select name="DropDownTimezone" id="timezone">
         <option value="-12.0">(GMT -12:00) Eniwetok, Kwajalein</option>
         <option value="-11.0">(GMT -11:00) Midway Island, Samoa</option>
         <option value="-10.0">(GMT -10:00) Hawaii</option>
         <option value="-9.0">(GMT -9:00) Alaska</option>
         <option value="-8.0">(GMT -8:00) Pacific Time (US &amp;amp; Canada)</option>
         <option value="-7.0">(GMT -7:00) Mountain Time (US &amp;amp; Canada)</option>
         <option value="-6.0">(GMT -6:00) Central Time (US &amp;amp; Canada), Mexico City</option>
         <option value="-5.0">(GMT -5:00) Eastern Time (US &amp;amp; Canada), Bogota, Lima</option>
         <option value="-4.0">(GMT -4:00) Atlantic Time (Canada), Caracas, La Paz</option>
         <option value="-3.5">(GMT -3:30) Newfoundland</option>
         <option value="-3.0">(GMT -3:00) Brazil, Buenos Aires, Georgetown</option>
         <option value="-2.0">(GMT -2:00) Mid-Atlantic</option>
         <option value="-1.0">(GMT -1:00 hour) Azores, Cape Verde Islands</option>
         <option value="0.0">(GMT) Western Europe Time, London, Lisbon, Casablanca</option>
        <option value="1.0" selected>(GMT +1:00 hour) Brussels, Copenhagen, Madrid, Paris</option>
         <option value="2.0">(GMT +2:00) Kaliningrad, South Africa</option>
         <option value="3.0">(GMT +3:00) Baghdad, Riyadh, Moscow, St. Petersburg</option>
         <option value="3.5">(GMT +3:30) Tehran</option>
         <option value="4.0">(GMT +4:00) Abu Dhabi, Muscat, Baku, Tbilisi</option>
         <option value="4.5">(GMT +4:30) Kabul</option>
         <option value="5.0">(GMT +5:00) Ekaterinburg, Islamabad, Karachi, Tashkent</option>
         <option value="5.5">(GMT +5:30) Bombay, Calcutta, Madras, New Delhi</option>
         <option value="5.75">(GMT +5:45) Kathmandu</option>
         <option value="6.0">(GMT +6:00) Almaty, Dhaka, Colombo</option>
         <option value="7.0">(GMT +7:00) Bangkok, Hanoi, Jakarta</option>
         <option value="8.0">(GMT +8:00) Beijing, Perth, Singapore, Hong Kong</option>
         <option value="9.0">(GMT +9:00) Tokyo, Seoul, Osaka, Sapporo, Yakutsk</option>
         <option value="9.5">(GMT +9:30) Adelaide, Darwin</option>
         <option value="10.0">(GMT +10:00) Eastern Australia, Guam, Vladivostok</option>
         <option value="11.0">(GMT +11:00) Magadan, Solomon Islands, New Caledonia</option>
         <option value="12.0">(GMT +12:00) Auckland, Wellington, Fiji, Kamchatka</option>
       </select> <input type="text" name="" value="" id="camp_form">
    </div>
</body>
</html>
```

La función Reloj inicializa un temporizador que genera eventos cada segundo, añadiéndole la hora actual como parámetro. La función anota el evento en el documento.

A este evento se suscribe la clase TextVisor, que representa un visor en modo Texto. El visor tiene una función, pinta, que nos escribe la hora actual en un div, con ID #reloj. Existe una segunda instancia de la clase TextVisor a la que se le sobrescribe el método pinta, asociándolo al contenido de un desplegable que os devolverá la diferencia horaria seleccionada.

Finalmente, existe una última instancia a la que también se sobrescribe la función pinta, escribiendo el reloj en el título de la ventana del navegador.