Manual de Modernizr

Manual de Modernizr, librerías Javascript para la detección de capacidades de los navegadores, para desarrollar aplicaciones web con HTML5 y CSS3 que funcionen en cualquier dispositivo.

Autores del manual

Este manual ha sido realizado por los siguientes colaboradores de DesarrolloWeb.com:

Miguel Angel Alvarez
Director de DesarrolloWeb.com
http://www.desarrolloweb.com
(7 capítulos)

Presentación de Modernizr

Modernizr es una librería Javascript que nos permite conocer la compatibilidad del navegador con tecnologías HTML5 y CSS3, para hacer sitios web que se adaptan a cada browser.

En el presente artículo vamos a presentar Modernizr, un componente que puede hacer la vida más sencilla a los desarrolladores que tienen que crear páginas web compatibles con distintos navegadores. Es ideal para aquellos profesionales que desean utilizar las tecnologías más modernas, incluso aquellas que están en fase de especificación como CSS3 y HTML5 y que pocos navegadores soportan, sin dejar de producir sitios web que funcionen correctamente en clientes web más antiguos.

Modernizr no es el típico framework Javascript, aunque podríamos llamarle así. Me explico. En realidad no es una librería que nos permita implementar con menor esfuerzo proyectos avanzados, sino un paquete de detección de las capacidades de un navegador relativas a HTML5 y CSS3. Dicho de otro modo, Modernizr es una librería Javascript que nos informará cuáles de las novedosas capacidades de estos lenguajes están disponibles en el navegador del usuario, para utilizarlas, o no, en cada caso.

Sabiendo que nuestro navegador soporta ciertas capacidades de CSS3 o de HTML5, podremos utilizarlas con libertad. De modo contrario, si sabemos que un navegador no es compatible con determinada funcionalidad, podremos implementar variantes que sí soporte y así crear sitios web que se adaptan perfectamente al cliente web de cada visitante.

Puedes encontrar Modernizr en: http://www.modernizr.com

Herramientas de Modernizr para detección de funcionalidades

Existen dos herramientas principales en Modernizr que se pueden utilizar para detectar las funcionalidades que están presentes en un navegador. Una la podemos utilizar en scripts Javascript y otra directamente sobre código CSS.

Objeto Javascript:
Una vez que carguemos las librerías de Modernizr tendremos a nuestra disposición un objeto Javascript que tiene una serie de propiedades que nos sirven para saber cuándo una funcionalidad concreta está disponible, o no, en un navegador. Esas propiedades tienen siempre valores boleanos (true o false), que podemos evaluar para hacer o no uso de las funcionalidades avanzadas que deseemos, sabiendo que realmente están disponibles.

Clases CSS:
Además, modernizr crea una serie de clases CSS que nos servirán, de una manera ingeniosa, para asignar estilos CSS3 únicamente cuando son soportados y que nos permitirán aplicar estilos alternativos cuando no se dispongan.

> **Nota:** En esta primera introducción a Modernizr no voy a entrar a explicar el uso de las mencionadas herramientas. Lo veremos con detenimiento en futuros artículos. De momento quiero simplemente dar un repaso general a lo que nos ofrece esta librería Javascript.

Qué funcionalidades detecta Modernizr

En resumen, con Modernizr podemos detectar las funcionalidades básicas de CSS3 y HTML5.

En el caso de CSS3, detecta si están o no implementados atributos para hacer bordes redondeados, sombras de cajas, imágenes en los bordes, colores rgba, múltiples fondos, etc. Además, existen formas de detectar si las animaciones CSS están implementadas en el navegador, las columnas CSS, los degradados, transformaciones, etc.

> **Nota:** si deseas conocer estas y otras características de las CSS3 te recomendamos acceder al Manual de CSS3 de DesarrolloWeb.com.

Por lo que respecta a HTML5, Modernizr es capaz de informarnos sobre la existencia o no de soporte a etiquetas como AUDIO, VIDEO y lo que será más importante, a diversas API que estarán disponibles en dicha versión del lenguaje de marcación, como Local Storage, Session Storage, Web Sockets, geolocalización, SVG, etc.

Polyfills en Modernizr

Hasta ahora hemos hablado siempre de "detección" de funcionalidades, nunca de ampliación de las características de los navegadores y es que Modernizr es justamente eso, una librería para detectar el soporte a las distintas características de los navegadores más modernos. Sin embargo, existe también una conexión entre Modernizr y lo que se llaman "Polyfills".

Un Polyfill o Polyfiller es una librería o plugin para ampliar las funcionalidades de navegadores antiguos que no soportan funcionalidades modernas. Para entender bien ese concepto pensemos en HTML 5, que trae consigo diversas API para trabajar con Local Storage o Web Sockets, etc. Hoy pocos navegadores soportan esas tecnologías y si hablamos de navegadores antiguos, pues el soporte es todavía menor. Entonces, podemos cargar un Polyfill en el navegador para que sea compatible con algunas de esas tecnologías.

El Polyfill no tiene por qué estar construido de la misma manera que el API nativo en los navegadores, pero es como un emulador, que nos proveerá de la misma interface, de modo que podamos trabajar en clientes web antiguos de la misma manera como trabajaríamos en los modernos.

Los Polyfill no forman parte de Modernizr específicamente, pero éste sí proporciona elementos para cargarlos en el navegador cuando no se tenga soporte nativo. De es modo podemos tener un soporte instantáneo a algunas funcionalidades del HTML5 que están por venir, aunque ello vaya a veces en detrimento del desempeño del cliente web.

Carga de Polyfills con Modernizr.load()

En el paquete básico de Modernizr no está disponible, pero entre varios "Extra" que podremos seleccionar en la página de download, encontraremos un método llamado Modernizr.load() que sirve para cargar Pollyfills.

Con Modernizr.load() y el objeto Javascript Modernizr, que almacena boleanos sobre si existe o no soporte a diversas funcionalidades, tenemos todo lo que necesitamos para cargar cómodamente los Pollyfills necesarios para ampliar el soporte de nuestro navegador a características de HTML5 y CSS3.

Desde la propia documentación de Modernizr aconsejan utilizar los Pollyfills con cuidado, porque puede bajar el rendimiento de los navegadores, no obstante, se nos antojan una vía excelente para poder innovar en el desarrollo y mantener soporte hacia atrás con navegadores antiguos.

En próximos artículos trataremos el método Modernizr.load() con detalle y estudiaremos más acerca de la carga de Pollyfills.

Conclusión

Modernizr es un complemento que agradará a los desarrolladores que quieren utilizar las últimas tecnologías, pero que están comprometidos con la compatibilidad y accesibilidad de sus creaciones. Quizás, cuando empieces a utilizarlo, encuentres que ha conquistado un lugar destacado entre tus herramientas de desarrollo.

Sus funcionalidades básicas, para detección de soporte a HTML5 y CSS3, ya son bastante interesantes de por si. Pero además, con la posibilidad de cargar pollyfills, se convierte en una herramienta potente que nos permitirá adelantarnos a diversos estándares que aun no están universalizados.

Artículo por Miguel Angel Alvarez

Primeros pasos con Modernizr

Cómo comenzar a usar Modernizr, las librerías para detección de funcionalidades en el navegador.

Modernizr es un paquete de librerías orientado para los desarrolladores que quieren crear sitios web con las tecnologías más modernas, léase CSS3 y HTML5, pero que no quieren olvidarse de la compatibilidad con navegadores antiguos. En el anterior artículo realizamos una presentación detallada de lo que nos ofrece Modernizr, por lo que ahora vamos a comenzar la explicación sobre su uso.

En este primer artículo vamos simplemente a ofrecer los primeros pasos que debemos realizar si queremos aprovechar las posibilidades de Modernizr en una página web que estemos realizando, es decir, cómo descargar las librerías y cómo incluirlas desde una página web.

Descarga de Modernizr

El primer paso consistirá en descargar el archivo con el código fuente de Modernizr. Se trata de un archivo con código Javascript que podemos encontrar en dos variantes:

1) Desarrollo (Development)
Es un script completo, con todas las funcionalidades básicas de Modernizr, sin comprimir y con comentarios. Sería ideal usar este script únicamente durante la fase de desarrollo de nuestro proyecto. Luego interesaría descargar un paquete idóneo para producción.

2) Producción (Production)
Cuando tengamos el sitio ya funcionando definitivamente y abierto a los visitantes, se recomienda hacer una descarga de las librerías para producción. Para hacer esta descarga se ofrece una página donde podemos seleccionar las funcionalidades de detección que queremos incluir, porque estemos utilizando en nuestro proyecto. El peso de las librerías no es muy grande, pero siempre está bien que nos permitan optimizar nuestra descarga para incluir solo aquellos módulos que vayamos a utilizar.

Desde la propia página de descargas de Modernizr podremos encontrar la lista de funcionalidades completa y seleccionar las que queremos incluir en nuestro paquete. También encontraremos un enlace para el acceso a la versión de desarrollo de las librerías.

> **Nota:** Si no sabes qué funcionalidades vas a utilizar de Modernizr se recomienda utilizar la versión de desarrollo (ocupa solo 42Kb), pero luego, una vez terminado tu proyecto, se recomienda hacer un paquete con las funcionalidades que realmente has utilizado. Ese paquete además estará sin comentarios y comprimido, por lo que ocupará bastante menos que la versión para desarrollo.

Instalación de Modernizr en la página

Una vez que hemos descargado nuestra librería, debemos incluirla en el código HTML de la página, de la misma manera que incluimos scripts Javascript de toda la vida.

```
<script src="modernizr-latest.js"></script>
```

Esto lo haremos en el HEAD de la página y además se nos recomienda colocar esta llamada al script de Modernizr justo después de nuestras declaraciones de estilos CSS.

> **Nota:** Según podemos leer en la documentación de Modernizer, se aconseja colocar el script en en HEAD porque debe cargarse antes del BODY de la página, debido a un componente que quizás utilicemos, para permitir HTML en IE, llamado HTML5 Shiv. Además, se recomienda colocar después de los estilos CSS para evitar un comportamiento poco deseable llamado FOUC, por el cual puede mostrarse por un pequeño espacio de tiempo la página sin los estilos CSS.

A partir de este momento tendremos disponibles nuestros scripts de detección de funcionalidades así como una serie de clases CSS que nos ayudarán a aplicar estilos solo cuando los navegadores los soporten. Todo eso lo veremos más adelante en siguientes artículos.

Navegadores compatibles

El listado de los navegadores compatibles con Modernizr es tan grande como podríamos desear. En el momento de escribir este artículo están soportando Internet Explorer 6+ (y superior), Firefox 3.5+, Opera 9.6+, Safari 2+ y Google Chrome. Además, para sistemas operativos de dispositivos móviles soportan Safari para iOS, el navegador de Android basado en Webkit, Opera Mobile y Firefox Mobile. Avisan en la documentación además que están trabajando para soportar Blackberry 6+ dentro de poco.

De momento dejamos este artículo por aquí, pues en el siguiente ya entraremos en materia para mostrar cómo detectar funcionalidades en el navegador por medio de Javascript y Modernizr.

Artículo por Miguel Angel Alvarez

Cómo detectar las capacidades de navegadores con Javascript y Modernizr

Explicamos cómo detectar si un navegador navegadores da soporte a las distintas funcionalidades de CSS3 y HTML, con Javascript y Modernizr.

Llegado a este punto en el Manual de Modernizr seguramente estemos interesados en comenzar a utilizar las librerías para detectar la compatibilidad con CSS3 y HTML5 o dicho de otra manera, poner manos en el código fuente para comenzar a entender de verdad cómo funciona Modernizr.

En el capítulo anterior explicamos cuáles eran los primeros pasos para incluir las librerías en una página web. Así que ahora podemos comenzar a usar Modernizr. Lo que vamos a hacer a continuación es básicamente un primer script Javascript que acceda al objeto Modernizr y nos diga si están o no presentes algunas funcionalidades en nuestro navegador.

Objeto Javascript Modernizr

Cuando tenemos Modernizr cargado en nuestra página, se crea automáticamente un objeto Javascript que tiene una serie de propiedades que nos informan sobre si están o no disponibles cada una de las funcionalidades presentes en CSS3 y HTML5. Las mencionadas propiedades contienen simplemente valores boleanos (verdadero o falso) que podemos evaluar para saber si están o no cada una de las funcionalidades que deseemos detectar.

El uso es tan sencillo como esto:

```
if (Modernizr.boxshadow){
   alert("SI sombra caja");
}
else{
   alert("NO sombra caja");
}
```

Como hemos visto, se está evaluando Modernizr.boxshadow (la propiedad boxshadow del objeto Modernizr) y estamos mostrando un mensaje en caso positivo y otro en caso negativo.

La propiedad boxshadow nos indica si el navegador es compatible con el atributo box-shadow de CSS3, que sirve para crear cajas de contenido con sombreado.

Ahora veamos otro ejemplo similar que detectaría si está disponible el elemento Canvas del HTML 5.

```
if (Modernizr.canvas){
   alert("SI canvas");
}
else{
   alert("NO canvas");
}
```

De un modo parecido al anterior, en este script estamos evaluando Modernizr.canvas, que nos dará positivo en caso que el navegador sea compatible con canvas y negativo en caso contrario.

Podemos ver esos dos scripts en marcha en una página aparte.

Está claro que no sirve de mucho mostrar un mensaje de alerta indicando la compatibilidad o no, pero basados en esta comprobación podremos hacer diferentes acciones para unos navegadores y otros. Por ejemplo en caso que no sea compatible un navegador, podríamos cargar un "pollyfill" que amplíe las posibilidades del navegador del cliente. Todo eso lo veremos en este mismo manual un poco más adelante.

Nota: Si quieres ir abriendo boca acerca de los pollyfills que podríamos invocar para ampliar las capacidades del navegador que está visitando la página, te recomiendo acceder a está página que nos muestra un listado de los Pollyfills para compatibilidad con HTML5.

Nosotros aquí te hemos mostrado simplemente un par de propiedades del objeto Modernizr, pero lógicamente existen muchas más. El listado completo de propiedades del objeto para la detección de funcionalidades HTML5 y CSS3 se puede encontrar en la propia documentación de Modernizr.

Recuerda además que debes tener un paquete de las librerías que incluya la detección con todas aquellas propiedades que piensas utilizar. La descarga indicada para desarrollo de las librerías incluye todas las propiedades y podemos crear un "build" para producción con únicamente las propiedades que deseemos utilizar. Todo eso ya se explicó en el artículo Primeros pasos con Modernizr.

De momento dejaremos Javascript de lado, ya que en el próximo artículo explicaremos cómo podemos usar las clases CSS de Modernizr para aplicar estilos CSS3 en navegadores que sean compatibles.

Artículo por Miguel Angel Alvarez

La magia CSS de Modernizr

Definir estilos CSS3 para los navegadores compatibles y estilos CSS alternativos para los navegadores que no sean compatibles con las propiedades CSS3 que deseemos utilizar.

Uno de los principales usos que podremos hacer de Modernizr es definir estilos CSS3 con compatibilidad para todos los navegadores. Bueno, no nos engañemos, si un navegador no es compatible con CSS3, Modernizr no va a hacer milagros, pero al menos puede ayudarnos a definir estilos alternativos. Lo que sin duda será interesante es olvidarse de los hacks CSS, cuyo uso se desaconseja en la mayoría de los casos, pero sobre todo, poder utilizar las nuevas características de las CSS3 ya mismo y sin necesidad de preocuparnos si el navegador entenderá o no nuestras declaraciones de estilos.

Para comenzar, tendremos que haber incluido el script Javascript con las librerías de Modernizr, algo que ya aprendimos en el artículo Primeros pasos con Modernizr, publicado anteriormente en el Manual de Modernizr.

Clases CSS de Modernizr

La "magia" CSS de Modernizr se basa en la creación en línea de una serie de clases CSS. Modernizr se encargará de crear esas clases y colocarlas en la etiqueta BODY de la página. Todo el proceso es automático y la creación de cada una de esas clases se realizará únicamente en caso de que el navegador sea compatible con cada una de las características de CSS3.

Por ejemplo, pensemos que hemos hecho un "build" de Modernizr para detectar la compatibilidad o no con un par de atributos CSS3, por ejemplo los bordes redondeados y los fondos múltiples.

Nota: Como se explicó en el artículo Primeros pasos con Modernizr, podemos hacer una descarga para entornos en producción con únicamente las características de CSS3 y HTML5 que deseemos detectar. En el caso de la versión para descarga de Modernizr en ambientes en desarrollo, existirá soporte para detectar todas las características de HTML5 y CSS3 posibles en las librerías.

Si el navegador del visitante es compatible con esas dos características de las CSS3, se crearán un par de clases en la etiqueta BODY. Es decir, nuestra etiqueta que antes era así:

```
<body>
```

Pasará a tener dos clases CSS, como estas:

```
<body class="borderradius multiplebgs">
```

> **Nota:** Insistimos en que esas clases se colocarán dinámicamente por medio de las librerías Javascript de Modernizr, en el caso que el navegador sea compatible con esas funcionalidades. Nosotros tendremos la etiqueta BODY sin ninguna clase en el código HTML y será Javascript el encargado de colocar las class de CSS. Si el navegador es solamente compatible con una de las funcionalidades, solo se aplicará esa clase que se está soportando.

En la documentación de Modernizr podemos ver una lista completa de las clases CSS que se generarán en la etiqueta BODY cuando el navegador soporte cada característica de las que se puede detectar con estas librerías.

Cómo utilizar las clases CSS de Modernizr para definir estilos compatibles

Ahora que conocemos la infraestructura de clases que nos proporcionará Modernizr, podemos pasar a ver cómo aplicar estilos CSS3 para todos los navegadores compatibles y además, estilos alternativos para los navegadores que no lo sean.

Nuestro objetivo es aplicar sombra con CSS3 en los navegadores que lo permitan y emular ese sombreado por medio de estilos CSS clásicos en los navegadores que no soporten el atributo box-shadow.

Primero veamos una simple capa con una etiqueta DIV de HTML:

```
<div id="micaja">
      Esta caja tendrá sombra.
</div>
```

Ahora podemos aplicarle estilos CSS que serían comunes para todos los navegadores, tanto los que soporten el atributo box-shadow como a los que no.

```
#micaja{
   border-left: 1px solid #ccc;
   border-top: 1px solid #ccc;
   border-bottom: 1px solid #666;
   border-right: 1px solid #666;
   padding: 10px;
}
```

En el caso anterior se han colocado estilos CSS convencionales, que entienden todos los navegadores.

Ahora, utilizando Modernizr, si nuestro navegador era compatible con el atributo box-shadow de CSS, se habrá colocado la clase "boxshadow" al BODY. Nosotros podemos utilizar dicha clase CSS para aplicar estilos que sabemos que solo acabarán afectando a los navegadores que soporten el atributo box-shadow.

```
.boxshadow #micaja{
   border: 1px solid #ccc;
   -webkit-box-shadow: #999 3px 3px 3px;
   -moz-box-shadow: #999 3px 3px 3px;
   box-shadow: #999 3px 3px 3px;
}
```

Como se puede ver, hemos sobreescrito la regla CSS para el borde y además hemos aplicado varios atributos box-shadow (uno para los navegadores basados en Mozilla, otro para los basados en Webkit y box-shadow a secas para todos los navegadores cuando soporten el atributo nativo de CSS3).

El efecto conseguido es que los navegadores modernos, que entendían el atributo box-shadow, mostrarán una sombra CSS3 y los más antiguos al menos mostrarán unos estilos para los que sí son compatibles.

Puedes ver este ejemplo en marcha en este enlace.

En el siguiente artículo veremos cómo detectar la compatibilidad con la regla @font-face de Modernizr.

Artículo por Miguel Angel Alvarez

Regla @font-face de CSS3 y detectar compatibilidad con Modernizr

Cómo podemos detectar la compatibilidad del navegador con la regla @font-face, usando Modernizr. Solucionar hipotéticos problemas cuando configuramos en un elemento varias opciones de tipografías, que tienen distintas características.

En el artículo anterior, la Magia de CSS3 de Modernizr, estuvimos explicando cuáles eran las bases utilizadas en Modernizr para detectar la compatibilidad del navegador del usuario con CSS3. Vimos un sencillo ejemplo que creemos vendrá bien complementar con otras prácticas.

En este artículo trabajaremos con el atributo de CSS3 @font-face, que permite utilizar cualquier tipografía en un sitio web, siempre que tengamos el archivo de la fuente. Veremos que en ocasiones unas tipografías se ven menores que otras en la pantalla y cómo hacer para que podamos definir tamaños de texto diferentes cuando el navegador muestra los textos con una u otra fuente de texto.

Regla @font-face y compatibilidad

Comenzamos viendo la regla @font-face, que sirve para definir tipografías para textos, aunque las mismas no estén presentes en el sistema del usuario. Esta regla es propia de CSS3 y para utilizarla debemos de subir el archivo de la tipografía a nuestro servidor, o bien utilizar algún servicio de host de fuentes como Google Fonts, que nos permita hacer uso de una variedad de tipos de letra alojadas en los servidores de Google.

> **Nota:** Google Fonts es lo que se llama un servicio CDN de fuentes. Dispone de más de 200 tipografías alojadas en los servidores de Google que puedes utilizar sin necesidad que estén en tu servidor, directamente enlazando una hoja de estilos que Google te ofrece. Otro servicio popular, alojado por CDN, de tipografías es TypeKit.

Por ejemplo, este código CSS tendremos acceso a la fuente "Tulpen One" alojada en Google Fonts:

```
@import url(http://fonts.googleapis.com/css?family=Tulpen+One);
```

> **Nota:** Esa línea está importando una declaración de estilo alojada en Google. En dicha declaración de estilos que se importa es donde aparece el atributo CSS3 @font-face para que los navegadores "descarguen" la fuente que queremos usar en la página. Si accedes con tu navegador directamente a la URL http://fonts.googleapis.com/css?family=Tulpen+One verás el código CSS real que estás importando y allí podrás encontrar la regla @font-face.

Una vez importada la fuente, podemos utilizarla en los elementos que deseemos, por ejemplo:

```
h1{
   font-family: 'Tulpen One', serif;
   font-size: 1.5em;
   color: #296033;
}
```

Esos estilos funcionan en todos los navegadores, pero claro, si el del usuario no aceptó el @font-face de Google Fonts, pues simplemente ignorará la fuente 'Tulpen One' y mostrará el texto en "serif" (es la fuente con serifa que tenga configurada el navegador por defecto).

Ahora bien, quiero haceros partícipes de un posible problema. La fuente 'Tulpen One' es más pequeña de lo normal, por lo que el tamaño 1.5em no sería muy bueno para un titular H1 (se vería demasiado pequeño para que visualmente se entienda que es un encabezamiento, Header de nivel 1). Es por ello que merecería la pena saber si nuestro navegador aceptó o no

'Tulpen One', para que en ese caso podamos aumentar el tamaño del texto un poquito más. Es justamente aquí donde podemos aprovechar las bondades de Modernizr, ya que nos permite saber si la regla de estilos @font-face está o no presente en nuestro navegador.

Si @font-face era compatible con nuestro navegador, entonces Modernizr creó, en la etiqueta BODY, una clase llamada "fontface" que podemos utilizar para definir estilos solo para los clientes web que entendieron esa regla de estilos. Por ello, la siguiente declaración será solo tenida en cuenta por los navegadores que están mostrando fuente del encabezamiento con "Tulpen One".

```
.fontface h1{
   font-size: 2.5em;
}
```

Como podemos ver, hemos definido simplemente un tamaño de letra mayor, que nos servirá para que los encabezamientos, a los que habíamos definido la tipografía "Tulpen One" tengan un tamaño de letra más adecuado.

Podemos ver un ejemplo de página que hace uso de @font-face y la detección de compatibilidad con Modernizr.

Conclusión

La mayoría de los navegadores actuales admiten @font-face, pero muchos usuarios usan todavía browsers poco actualizados que pueden no tener compatibilidad con esa regla de estilos.

En los casos que queramos aplicar estilos solo para los navegadores que soportan @font-face, disponemos de Modernizr para facilitarnos la vida.

En el siguiente artículo veremos otro ejemplo de detección de compatibilidad con reglas de estilos.

Artículo por Miguel Angel Alvarez

Más sobre CSS3 y compatibilidad

Más ejemplos de detección de características de las CSS3 con Modernizr y aplicación de estilos alternativos para navegadores no compatibles.

En este artículo vamos a detectar tres estilos CSS3 y a ofrecer estilos CSS alternativos para los navegadores que no los soporten. En concreto en los ejemplos trabajaremos con los atributos de CSS3 para hacer background múltiple (asignar varios fondos de imagen a un mismo elemento), text-shadow (hacer efectos de sombra en textos) y los relacionados con maquetación automática en columnas.

Por supuesto, ya que estamos en el Manual de Modernizr, veremos como hacerlo usando estas librerías Javascript.

Comencemos por ofrecer un link a una página que hemos realizado como ejemplo de estas propiedades. Dependiendo del navegador y la compatibilidad con las reglas de estilo utilizadas verás el ejemplo de una manera u otra.

Fondos múltiples en un mismo elemento

A partir de CSS3 podemos asignar varios fondos de tipo imagen a un mismo elemento de la página. Para ello simplemente separamos por coma cada uno de los fondos distintos que estamos asignando.

Ahora bien, si nuestro navegador no es compatible con CSS3 y ve varios fondos distintos en un mismo elemento, por muchas comas que pongamos para separarlos, no entenderá nada. El resultadó será que no nos mostrará ninguno de los fondos.

La situación, antes de Modernizr nos obligaba a colocar siempre un único fondo, que funcionará en todo caso. En caso que coloquemos varios fondos, también antes de Modernizr, nos arriesgamos a que el navegador del usuario no nos muestre ninguno.

La solución es tan fácil como usar Modernizr para detectar cuándo podemos aplicar múltiples fondos y colocar un único fondo cuando los navegadores solamente soportan uno. Para ello vamos a definir en el estilo general del elemento un único fondo.

```
#central{
   background: url(images/patron.png);
}
```

Luego, en caso que se permitan múltiples fondos, Modernizr crea una clase en la etiqueta BODY llamada "multiplebgs", que podemos utilizar para aplicar más de un fondo sin peligro que la compabilidad de nuestro navegador con lass hojas de estilo lo rechace.

```
.multiplebgs #central{
   background: url(images/diamante.png) no-repeat bottom right,
   url(images/patron.png);
}
```

Sombras en textos

A pesar que text-shadow no pertenece específicamente a CSS3, sino que fue introducido ya con la segunda versión del lenguaje, también se puede detectar con Modernizr y resultará útil porque a día de hoy todavía muchos navegadores no soportan esa regla.

En el elemento anterior habíamos colocado un fondo tirando hacia claro, que nos obligaba a escribir con una letra de color oscuro. Además quiero ponerle una sombra a mi texto, también oscura. Por ello hemos definido este estilo de manera general para los párrafos que van dentro de la capa #central.

```
#central p{
   color: #33c;
   text-shadow: -1px 1px 5px #006;
}
```

Ahora bien, texto oscuro sobre sombreado oscuro no queda demasiado bien. Me gustaría detectar en qué casos el navegador soporta sombreado, para entonces colocarle el texto claro. Texto claro sobre sombreado oscuro contrastará suficientemente.

Así que la solución pasa por usar Modernizr, y crear una regla de estilo para hacer un texto de color blanco que solo se tenga en cuenta en los navegadores que aceptan sobreado de texto. Para ello es tan sencillo como utilizar la clase "textshadow" que estará activa solamente cuando el navegador disponga de compatibilidad con sombreado de textos.

```
.textshadow #central p{
   color: #fff;
}
```

> **Nota:** Ojo, hemos querido comentar este ejemplo porque Firefox 3 da un falso positivo a la compatibilidad con text-shadow. Así que tener cuidado con este detalle porque en Firefox 3, a pesar de no sobrear el texto, nuestro ejemplo aparecería con color blanco, lo que dificultaría la lectura. Quizás no sea un problema muy representativo, porque justamente acaba de salir Firefox 6. Sin embargo hace pocos meses Firefox 3 era la versión más actual. Por ejemplo, hoy podemos ver en Internet Explorer 9 que aun no acepta sombreado y el texto nos aparecerá en oscuro.

Distribución en varias columnas con CSS3

Gracias a una serie de atributos disponibles en las hojas de estilo en cascada nivel 3, se pueden agrupar los contenidos en columnas de manera automática. Es una interesante utilidad que ayudará a las páginas web a parecerse mucho más a las publicaciones impresas. El tema es que aun no está disponible en muchos navegadores, con lo que tenemos que utilizar esa característica con cuidado.

En el ejemplo de este artículo tenemos una lista que tiene varios elementos, a la que hemos aplicado estilos CSS3 para que aparezcan en varias columnas.

```
#milista{
   column-width: 120px;
   column-gap: 25px;
   column-rule: 2px solid #ccf;
```

```
}
```

Gracias a ello vemos que los elementos de la lista aparecen maquetados en varias columnas, pero en navegadores que no lo admitan aparecerían uno debajo de otro. Para emular con CSS2 la maquetación en columnas podemos utilizar el atributo float, asignarle una anchura a los elementos de la lista y un margen para hacer el espaciado entre columnas.

```
#milista li{
   float: left;
   width: 120px;
   margin-right: 25px;
}
```

> **Nota:** Si podéis ver el ejemplo en navegadores que sí aceptan los atributos CSS3 de columnas (por ejemplo Google Chrome, y navegadores que no aceptan esas reglas (por ejemplo Internet Explorer 9) veréis que las columnas emuladas no distribuyen exactamente los elementos de la misma manera que las columnas emuladas. La distribución emulada es menos útil porque generalmente leemos las columnas de arriba a abajo, pero estéticamente el resultado es parecido.

Pero claro, eso provocaría que todos los navegadores hiciesen "flotar" los elementos a la izquierda. Para evitarlo podemos utilizar la clase "csscolumns" que estará solamente disponible en caso que el navegador sea compatible con la distribución en columnas.

```
.csscolumns #milista li{
   float: none;
   margin: 0;
   width: auto;
}
```

Como vemos, en caso que el navegador entienda las reglas de estilos para producir las columnas automáticamente, estamos haciendo que los elementos no floten, que su anchura sea automática y que no tengan margen a la derecha.

Conclusión

Puedes ver de nuevo el ejemplo en marcha en una página aparte.

Con esto terminamos los ejemplos de detección de compatibilidad CSS3 de Modernizr, pero recordar que en la documentación de las librerías encontraréis muchas otras clases para detección de la compatibilidad con CSS3.

En los siguientes artículos daremos un paso adelante, pues aun nos queda mucho por aprender de Modernizr y cosas más interesantes todavía, como la carga de Polyfills para compatibilidad con HTML5.

Artículo por Miguel Angel Alvarez

Modernizr.load() y la carga condicional de librerías Javascript y CSS

El método load() de Modernizr sirve para cargar librerías Javascript dependiendo de las capacidades del navegador.

Hasta ahora en el Manual de Modernizr hemos visto muchas cosas sencillas, que nos pueden ayudar a aumentar el grado de adaptabilidad de nuestra web a distintos navegadores y dispositivos. Los conocimientos que habrás podido adquirir hasta el momento sirven tanto para diseñadores como para programadores, sin embargo las librerías todavía tienen algunos usos interesantes que no hemos visto, que ayudarán sobre todo a los desarrolladores HTML5 y a los programadores Javascript.

En ese sentido, comenzamos una serie de artículos que nos detallarán algunos de los puntos más interesantes que nos ofrecen estas librerías, comenzando en esta ocasión con una introducción al método Modernizr.load(), un gestor condicional de carga de librerías Javascript y CSS.

Modernizr.load() está basado en unas librerías llamadas yepnope.js que sirven básicamente para cargar librerías solo cuando

los usuarios las necesitan. Se trata de un sencillo método que permite evaluar una condición y decir qué librerías cargar cuando se cumpla y cuáles cuando no.

Las condiciones pueden ser las propiedades que nos ofrece Modernizr, de modo que carguemos determinadas librerías solamente en los casos en los que se necesiten. Por ejemplo, pensemos en que estamos haciendo una página basada en el API Local Storage del HTML5. Con Modernizr podemos saber si el navegador del usuario ofrece soporte a ese API, mediante la propiedad:

Modernizr.localstorage

Evaluar esa propiedad, para hacer unas cosas u otras con Javascript, es bastante sencillo, tal como vimos en el artículo sobre Detectar las capacidades de navegadores con Javascript.
Ahora bien, si deseamos cargar unos recursos u otros en el navegador dependiendo de ella, el método Modernizr.load() nos podrá ahorrar algo de código fuente y de paso acelerar nuestra página en algunas ocasiones.

Cuando en DesarrolloWeb.com publicamos la presentación inicial de Modernizr os hablamos de los Polyfills y explicamos que existían librerías para implementar casi todas las características del HTML5 en navegadores que no las soportaban de manera nativa. Esas librerías son los mencionados Polyfills y las podemos cargar de manera condicional por medio de Modernizr.load().

Volviendo al ejemplo del Local Storage, con Modernizr.load() podemos indicar a los navegadores que no soporten ese API que carguen el Polyfill Storage, de modo que se pueda utilizar esa característica del HTML 5 en ellos también.

Sintaxis de Modernizr.load()

Ahora que ya podemos tener una idea más o menos exacta de lo que se puede hacer con Modernizr.load(), vamos a repasar un poco su sintaxis.

> **Nota:** Modernizr.load() es un añadido a Modernizr de descarga opcional. De hecho, si estás utilizando la versión de desarrollo de Modernizr no se incluye este método, por lo que tendrías que hacer una descarga para producción que sí lo incluya. Desde la página de download de Modernizr verás que en el recuadro "Extra" existen diversos añadidos entre los que encontrarás la mencionada función Modernizr.load.

```
Modernizr.load({
   test: Modernizr.localstorage,
   yep : 'existe_soporte_nativo.js',
   nope: ['storage-polyfill.js', 'estilos-polyfill.css']
});
```

En el código anterior podemos ver la llamada al método load. En ella indicamos varias propiedades en notación de objeto:

- **test:** es la evaluación que se va a realizar para saber si tenemos soporte a una u otra capacidad en el navegador.
- **yep:** donde indicamos las librerías que deben cargarse en caso que la evaluación anterior de positivo. En este caso, cuando el navegador es compatible con Local Storage del HTML5, se cargará una única librería llamada 'existe_soporte_nativo.js'.
- **nope:** donde se indica la librería/s que queremos cargar cuando la evaluación de negativo. En el ejemplo anterior, en caso que el navegador no tenga soporte a Local Storage, se cargaría la librería 'storage-polyfill.js' y además la declaración de estilos 'estilos-polyfill.css'.

Como hemos visto, la sintaxis y uso del método load() es bastante sencilla, pero aun se podría personalizar un poco más este script para hacer más cosas útiles.

Cargar de librerías para los casos positivos y negativos:
También podemos especificar los archivos Javascript o CSS que queremos cargar cuando se cumpla determinada evaluación y cuando no se cumpla. Para ello simplemente creamos otra propiedad en el parámetro pasado a load(), llamada "both".

```
Modernizr.load({
   test: Modernizr.multiplebgs && Modernizr.opacity,
   nope: 'polyfills-css.js',
   both: 'otras-cosas.js'
});
```

En este caso se evaluarán si el navegador dispone de soporte para múltiples fondos con CSS3 y para las propiedades de opacidad. En caso negativo se cargaría la librería "polyfills-css,js" y tanto en caso positivo como en el negativo (es decir, en todos los navegadores) se cargaría la librería "otras-cosas.js".

Hacer cosas después de la carga de las librerías:
En ocasiones queremos hacer algún tipo de acción, pero no queremos ejecutarla hasta no haber sido cargadas determinadas librerías. Para ello se puede implementar una función en la propiedad complete, que solamente se ejecutará cuando las librerías hayan sido cargadas.

```
Modernizr.load({
   test: Modernizr.geolocation,
   yep : 'geo-extras.js',
   nope: 'geo-polyfill.js',
   both: 'migeolocalizacion.js',
   complete: function(){
      inicializar-geolocalizacion();
   }
});
```

En este caso se evalúa si existe soporte al API de geolocalización. En caso positivo se cargan unos scripts ('geo-extras.js'), en caso negativo otros ('geo-polyfill.js') y para ambos casos se carga otra librería ('migeolocalizacion.js'). Lo interesante en este ejemplo es el método siguiente, bajo el nombre de complete, donde se ha asignado una función que se ejecutará solo después de haber cargado todas las librerías que corresponda en cada caso. En esa función podemos colocar código que necesite de esas librerías con total seguridad, ya que sabremos seguro que su carga se habrá realizado anteriormente a su la ejecución de la función complete.

Carga no condicional de librerías
Con Modernizr.load() también podemos cargar librerías de manera incondicional, es decir, sin evaluar previamente ninguna condición. Para ello utilizamos la propiedad load. Ahora vemos un método copiado directamente del tutorial de Modernizr.load() que está en la documentación de estas librerías.

Lo he copiado porque me parece un caso excelente de uso de esta funcionalidad. Se trata de un script que se trae por CDN la librería jQuery pero que, si hubo algún problema con dicha librería, carga jQuery directamente de una copia del framework en nuestro servidor local.

```
Modernizr.load({
   load: '//ajax.googleapis.com/ajax/libs/jquery/1.6.3/jquery.js',
   complete: function () {
      if ( !window.jQuery ) {
         Modernizr.load('js/jquery-1.6.3.min.js');
      }
   }
});
```

En este ejemplo se carga jQuery desde el CDN de Google APIs. Utilizamos la propiedad "load", luego no existe ninguna evaluación, sino que simplemente se carga esa librería independientemente del navegador del usuario. Luego se utiliza complete para especificar una función a ejecutar cuando se termina la carga de las librerías. En esa función se evalúa ! window.jQuery. Si jQuery no se hubiera cargado, esa evaluación nos llevaría a la carga del framework directamente de una ruta desde nuestro propio servidor, utilizando de nuevo Modernizr.load().

Carga de varios paquetes de librerías de manera secuencial:
Por último vamos a mostrar cómo realizar varios bloques de carga de librerías dentro de una única llamada a modernizr.load(), lo que nos servirá cuando queremos cargar condicionalmente diversos conjuntos de librerías dependiendo de varias evaluaciones independientes. Todos esos bloques se indicarán en un array y se ejecutarán uno detrás de otro, por lo que hasta que no se carguen las librerías del primer bloque, no se pasará al siguiente.

```
Modernizr.load([
   //Primer bloque
   {
      test : Modernizr.fontface,
      nope : ['fuentes-css.js', 'otros-estilos.css']
   },
   //Segundo bloque
   {
      test : Modernizr.audio,
      nope : 'soporte-audio.js',
```

```
      both : 'mi_hilo_musical.js',
      complete : function () {
         iniciarMusica();
      }
   },
   //tercer bloque
   'script-estadisticas.js'
]);
```

En este ejemplo tenemos tres bloques.

1. Primero se verá si tenemos soporte para @font-face, cargando un archivo Javascript y otro CSS en caso negativo.
2. Luego se verá si tenemos soporte al audio de HTML5, cargando librerías que nos ofrezcan compatibilidad cuando no se tenga y cargando una librería para implementar mi sonido de fondo. Cuando todas esas librerías estén cargadas, se llama a la función iniciarMusica(), que iniciaría el sonido.
3. Por último se cargará el script para la contabilidad de las estadísticas en esta página. Si nos fijamos la carga de este script se realiza directamente, sin evaluar condición alguna. Para ello simplemente indicamos el nombre de la librería a cargar. Esto es perfectamente posible en el método Modernizr.load().

Lo interesante de este método de carga es que se realiza paso a paso, por lo que podemos dejar para el final los scripts que menos importancia tienen. En este ejemplo, sacado también de la documentación de Modernizr, tenemos que en el último bloque se carga el script de estadísticas, que es lo menos importante. Así sabremos seguro que la contabilidad de esta visita del usuario se realizará solo cuando ya han terminado de cargarse otras librerías más importantes.

Cabe destacar también, según comentan en la documentación, que este modo de carga secuencial de librerías, a pesar de tener que esperar a la completa descarga de un bloque antes de comenzar la carga del siguiente, no supondrá un descenso de la velocidad del sitio con respecto a si todas las librerías fueran descargadas en paralelo. Incluso en determinadas ocasiones dicen que podría aumentar la velocidad.

Lo que está claro es que, gracias a Modernizr, podemos fácilmente discriminar entre navegadores y cargar solamente aquellas librerías que nuestro navegador necesite, en vez de cargarlas todas y luego ver si realmente se necesitaban. Sin duda podríamos hacer nuestras propias validaciones para averiguar si se necesitan y luego cargarlas por medio de nuestros propios scripts, pero difícilmente vamos a poder realizar las cosas por nosotros mismos más rápido y mejor de lo que propone Modernizr.load().

En el siguiente artículo presentaremos un ejemplo completo de carga de un pollyfill para compatibilidad con Canvas del HTML 5, condicionalmente por medio de Modernizr.load(), de modo que solo se invoque cuando el navegador no disponga de soporte nativo a canvas.

Artículo por Miguel Angel Alvarez