Capítulo 6. Eventos

Hasta ahora, todas las aplicaciones y scripts que se han creado tienen algo en común: se ejecutan desde la primera instrucción hasta la última de forma secuencial. Gracias a las estructuras de control de flujo (`if`, `for`, `while`) es posible modificar ligeramente este comportamiento y repetir algunos trozos del script y saltarse otros trozos en función de algunas condiciones.

Este tipo de aplicaciones son poco útiles, ya que no interactúan con los usuarios y no pueden responder a los diferentes *eventos* que se producen durante la ejecución de una aplicación. Afortunadamente, las aplicaciones web creadas con el lenguaje JavaScript pueden utilizar el modelo de *programación basada en eventos*.

En este tipo de programación, los scripts se dedican a esperar a que el usuario *"haga algo"* (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador). A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.

Los eventos hacen posible que los usuarios transmitan información a los programas. JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. La pulsación de una tecla constituye un evento, así como pinchar o mover el ratón, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.

JavaScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada. Este tipo de funciones se denominan *"event handlers"* en inglés y suelen traducirse por *"manejadores de eventos"*.

6.1. Modelos de eventos

Crear páginas y aplicaciones web siempre ha sido mucho más complejo de lo que debería serlo debido a las incompatibilidades entre navegadores. A pesar de que existen decenas de estándares para las tecnologías empleadas, los navegadores no los soportan completamente o incluso los ignoran.

Las principales incompatibilidades se producen en el lenguaje XHTML, en el soporte de hojas de estilos CSS y sobre todo, en la implementación de JavaScript. De todas ellas, la incompatibilidad más importante se da precisamente en el modelo de eventos del navegador. Así, existen hasta tres modelos diferentes para manejar los eventos dependiendo del navegador en el que se ejecute la aplicación.

6.1.1. Modelo básico de eventos

Este modelo simple de eventos se introdujo para la versión 4 del estándar HTML y se considera parte del nivel más básico de DOM. Aunque sus características son limitadas, es el único modelo que es compatible en todos los navegadores y por tanto, el único que permite crear aplicaciones que funcionan de la misma manera en todos los navegadores.

6.1.2. Modelo de eventos estándar

Las versiones más avanzadas del estándar DOM (DOM nivel 2) definen un modelo de eventos completamente nuevo y mucho más poderoso que el original. Todos los navegadores modernos lo incluyen, salvo Internet Explorer.

6.1.3. Modelo de eventos de Internet Explorer

Internet Explorer utiliza su propio modelo de eventos, que es similar pero incompatible con el modelo estándar. Se utilizó por primera vez en Internet Explorer 4 y Microsoft decidió seguir utilizándolo en el resto de versiones, a pesar de que la empresa había participado en la creación del estándar de DOM que define el modelo de eventos estándar.

6.2. Modelo básico de eventos

6.2.1. Tipos de eventos

En este modelo, cada elemento o etiqueta XHTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos XHTML diferentes y un mismo elemento XHTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo `on`, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina `onclick` y el evento asociado a la acción de mover el ratón se denomina `onmousemove`.

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

Evento	Descripción	Elementos para los que está definido
`onblur`	Deseleccionar el elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onchange`	Deseleccionar un elemento que se ha modificado	`<input>`, `<select>`, `<textarea>`
`onclick`	Pinchar y soltar el ratón	Todos los elementos
`ondblclick`	Pinchar dos veces seguidas con el ratón	Todos los elementos
`onfocus`	Seleccionar un elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onkeydown`	Pulsar una tecla (sin soltar)	Elementos de formulario y `<body>`
`onkeypress`	Pulsar una tecla	Elementos de formulario y `<body>`
`onkeyup`	Soltar una tecla pulsada	Elementos de formulario y `<body>`
`onload`	La página se ha cargado completamente	`<body>`
`onmousedown`	Pulsar (sin soltar) un botón del ratón	Todos los elementos
`onmousemove`	Mover el ratón	Todos los elementos

onmouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Los eventos más utilizados en las aplicaciones web tradicionales son `onload` para esperar a que se cargue la página por completo, los eventos `onclick`, `onmouseover`, `onmouseout` para controlar el ratón y `onsubmit` para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (`onclick`, `onkeydown`, `onkeypress`, `onreset`, `onsubmit`) permiten evitar la *"acción por defecto"* de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo `<input type="submit">` se desencadenan los eventos `onmousedown`, `onclick`, `onmouseup` y `onsubmit` de forma consecutiva.

6.2.2. Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede *responder* ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan *"manejador de eventos"* y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos XHTML.
- Manejadores como funciones JavaScript externas.
- Manejadores *"semánticos"*.

6.2.2.1. Manejadores de eventos como atributos XHTML

Se trata del método más sencillo y a la vez *menos profesional* de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo

del propio elemento XHTML. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />
```

En este método, se definen atributos XHTML con el mismo nombre que los eventos que se quieren manejar. El ejemplo anterior sólo quiere controlar el evento de pinchar con el ratón, cuyo nombre es `onclick`. Así, el elemento XHTML para el que se quiere definir este evento, debe incluir un atributo llamado `onclick`.

El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento. En este caso, el código JavaScript es muy sencillo (`alert('Gracias por pinchar');`), ya que solamente se trata de mostrar un mensaje.

En este otro ejemplo, cuando el usuario pincha sobre el elemento `<div>` se muestra un mensaje y cuando el usuario pasa el ratón por encima del elemento, se muestra otro mensaje:

```
<div onclick="alert('Has pinchado con el ratón');" onmouseover="alert('Acabas de pasar
el ratón por encima');">
  Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima
</div>
```

Este otro ejemplo incluye una de las instrucciones más utilizadas en las aplicaciones JavaScript más antiguas:

```
<body onload="alert('La página se ha cargado completamente');">
   ...
</body>
```

El mensaje anterior se muestra después de que la página se haya cargado completamente, es decir, después de que se haya descargado su código HTML, sus imágenes y cualquier otro objeto incluido en la página.

El evento `onload` es uno de los más utilizados ya que, como se vio en el capítulo de DOM, las funciones que permiten acceder y manipular los nodos del árbol DOM solamente están disponibles cuando la página se ha cargado completamente.

6.2.2.2. Manejadores de eventos y variable this

JavaScript define una variable especial llamada `this` que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable `this` para referirse al elemento XHTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente:

Cuando el usuario pasa el ratón por encima del `<div>`, el color del borde se muestra de color negro. Cuando el ratón *sale* del `<div>`, se vuelve a mostrar el borde con el color gris claro original.

Elemento `<div>` original:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver">
  Sección de contenidos...
</div>
```

Si no se utiliza la variable `this`, el código necesario para modificar el color de los bordes, sería el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="document.getElementById('contenidos').style.borderColor='black';"
onmouseout="document.getElementById('contenidos').style.borderColor='silver';">
  Sección de contenidos...
</div>
```

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable `this`, que hace referencia al elemento XHTML que ha provocado el evento. Así, el ejemplo anterior se puede reescribir de la siguiente manera:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="this.style.borderColor='black';"
onmouseout="this.style.borderColor='silver';">
  Sección de contenidos...
</div>
```

El código anterior es mucho más compacto, más fácil de leer y de escribir y sigue funcionando correctamente aunque se modifique el valor del atributo `id` del `<div>`.

6.2.2.3. Manejadores de eventos como funciones externas

La definición de los manejadores de eventos en los atributos XHTML es el método más sencillo pero menos aconsejable de tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento XHTML.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />
```

Utilizando funciones externas se puede transformar en:

```
function muestraMensaje() {
  alert('Gracias por pinchar');
}

<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento XHTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable `this` y por tanto, es necesario pasar esta variable como parámetro a la función:

```
function resalta(elemento) {
  switch(elemento.style.borderColor) {
    case 'silver':
    case 'silver silver silver silver':
    case '#c0c0c0':
      elemento.style.borderColor = 'black';
      break;
    case 'black':
    case 'black black black black':
    case '#000000':
      elemento.style.borderColor = 'silver';
      break;
  }
}

<div style="width:150px; height:60px; border:thin solid silver"
onmouseover="resalta(this)" onmouseout="resalta(this)">
  Sección de contenidos...
</div>
```

En el ejemplo anterior, la función externa es llamada con el parámetro `this`, que dentro de la función se denomina `elemento`. La complejidad del ejemplo se produce sobre todo por la forma en la que los distintos navegadores almacenan el valor de la propiedad `borderColor`.

Mientras que Firefox almacena (en caso de que los cuatro bordes coincidan en color) el valor `black`, Internet Explorer lo almacena como `black black black black` y Opera almacena su representación hexadecimal `#000000`.

6.2.2.4. Manejadores de eventos semánticos

Los métodos que se han visto para añadir manejadores de eventos (como atributos XHTML y como funciones externas) tienen un grave inconveniente: *"ensucian"* el código XHTML de la página.

Como es conocido, una de las buenas prácticas básicas en el diseño de páginas y aplicaciones web es la separación de los contenidos (XHTML) y su aspecto o presentación (CSS). Siempre que sea posible, también se recomienda separar los contenidos (XHTML) y su comportamiento o programación (JavaScript).

Mezclar el código JavaScript con los elementos XHTML solamente contribuye a complicar el código fuente de la página, a dificultar la modificación y mantenimiento de la página y a reducir la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos XHTML para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

```
<input id="pinchable" type="button" value="Pinchame y verás" onclick="alert('Gracias
por pinchar');" />
```

Se puede transformar en:

```
// Función externa
function muestraMensaje() {
  alert('Gracias por pinchar');
}

// Asignar la función externa al elemento
document.getElementById("pinchable").onclick = muestraMensaje;

// Elemento XHTML
<input id="pinchable" type="button" value="Pinchame y verás" />
```

La técnica de los manejadores semánticos consiste en:

1. Asignar un identificador único al elemento XHTML mediante el atributo `id`.
2. Crear una función de JavaScript encargada de manejar el evento.
3. Asignar la función externa al evento correspondiente en el elemento deseado.

El último paso es la clave de esta técnica. En primer lugar, se obtiene el elemento al que se desea asociar la función externa:

```
document.getElementById("pinchable");
```

A continuación, se utiliza una propiedad del elemento con el mismo nombre que el evento que se quiere manejar. En este caso, la propiedad es `onclick`:

```
document.getElementById("pinchable").onclick = ...
```

Por último, se asigna la función externa mediante su nombre sin paréntesis. Lo más importante (y la causa más común de errores) es indicar solamente el nombre de la función, es decir, prescindir de los paréntesis al asignar la función:

```
document.getElementById("pinchable").onclick = muestraMensaje;
```

Si se añaden los paréntesis después del nombre de la función, en realidad se está ejecutando la función y guardando el valor devuelto por la función en la propiedad `onclick` de elemento.

```
// Asignar una función externa a un evento de un elemento
document.getElementById("pinchable").onclick = muestraMensaje;

// Ejecutar una función y guardar su resultado en una propiedad de un elemento
document.getElementById("pinchable").onclick = muestraMensaje();
```

La gran ventaja de este método es que el código XHTML resultante es muy *"limpio"*, ya que no se mezcla con el código JavaScript. Además, dentro de las funciones externas asignadas sí que se puede utilizar la variable `this` para referirse al elemento que provoca el evento.

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos XHTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento `onload`:

```
window.onload = function() {
  document.getElementById("pinchable").onclick = muestraMensaje;
}
```

La técnica anterior utiliza el concepto de *funciones anónimas*, que no se va a estudiar, pero que permite crear un código compacto y muy sencillo. Para asegurarse que un código JavaScript va a ejecutarse después de que la página se haya cargado completamente, sólo es necesario incluir esas instrucciones entre los símbolos { y }:

```
window.onload = function() {
  ...
}
```

En el siguiente ejemplo, se añaden eventos a los elementos de tipo `input=text` de un formulario complejo:

```
function resalta() {
  // Código JavaScript
}

window.onload = function() {
  var formulario = document.getElementById("formulario");
  var camposInput = formulario.getElementsByTagName("input");

  for(var i=0; i<camposInput.length; i++) {
    if(camposInput[i].type == "text") {
      camposInput[i].onclick = resalta;
    }
  }
}
```

Ejercicio 14

A partir de la página web proporcionada, completar el código JavaScript para que:

1. Cuando se pinche sobre el primer enlace, se oculte su sección relacionada
2. Cuando se vuelva a pinchar sobre el mismo enlace, se muestre otra vez esa sección de contenidos
3. Completar el resto de enlaces de la página para que su comportamiento sea idéntico al del primer enlace
4. Cuando una sección se oculte, debe cambiar el mensaje del enlace asociado (pista: propiedad `innerHTML`)

6.3. Obteniendo información del evento (objeto event)

Normalmente, los manejadores de eventos requieren información adicional para procesar sus tareas. Si una función por ejemplo se encarga de procesar el evento `onclick`, quizás necesite saber en que posición estaba el ratón en el momento de pinchar el botón.

No obstante, el caso más habitual en el que es necesario conocer información adicional sobre el evento es el de los eventos asociados al teclado. Normalmente, es muy importante conocer la

tecla que se ha pulsado, por ejemplo para diferenciar las teclas normales de las teclas especiales (`ENTER`, tabulador, `Alt`, `Ctrl.`, etc.).

JavaScript permite obtener información sobre el ratón y el teclado mediante un objeto especial llamado `event`. Desafortunadamente, los diferentes navegadores presentan diferencias muy notables en el tratamiento de la información sobre los eventos.

La principal diferencia reside en la forma en la que se obtiene el objeto `event`. Internet Explorer considera que este objeto forma parte del objeto `window` y el resto de navegadores lo consideran como el único argumento que tienen las funciones manejadoras de eventos.

Aunque es un comportamiento que resulta muy extraño al principio, todos los navegadores modernos excepto Internet Explorer crean *mágicamente* y de forma automática un argumento que se pasa a la función manejadora, por lo que no es necesario incluirlo en la llamada a la función manejadora. De esta forma, para utilizar este *"argumento mágico"*, sólo es necesario asignarle un nombre, ya que los navegadores lo crean automáticamente.

En resumen, en los navegadores tipo Internet Explorer, el objeto `event` se obtiene directamente mediante:

```
var evento = window.event;
```

Por otra parte, en el resto de navegadores, el objeto `event` se obtiene *mágicamente* a partir del argumento que el navegador crea automáticamente:

```
function manejadorEventos(elEvento) {
  var evento = elEvento;
}
```

Si se quiere programar una aplicación que funcione correctamente en todos los navegadores, es necesario obtener el objeto `event` de forma correcta según cada navegador. El siguiente código muestra la forma correcta de obtener el objeto `event` en cualquier navegador:

```
function manejadorEventos(elEvento) {
  var evento = elEvento || window.event;
}
```

Una vez obtenido el objeto `event`, ya se puede acceder a toda la información relacionada con el evento, que depende del tipo de evento producido.

6.3.1. Información sobre el evento

La propiedad `type` indica el tipo de evento producido, lo que es útil cuando una misma función se utiliza para manejar varios eventos:

```
var tipo = evento.type;
```

La propiedad `type` devuelve el tipo de evento producido, que es igual al nombre del evento pero sin el prefijo `on`.

Mediante esta propiedad, se puede rehacer de forma más sencilla el ejemplo anterior en el que se resaltaba una sección de contenidos al pasar el ratón por encima:

```
function resalta(elEvento) {
  var evento = elEvento || window.event;
  switch(evento.type) {
    case 'mouseover':
      this.style.borderColor = 'black';
      break;
    case 'mouseout':
      this.style.borderColor = 'silver';
      break;
  }
}

window.onload = function() {
  document.getElementById("seccion").onmouseover = resalta;
  document.getElementById("seccion").onmouseout = resalta;
}

<div id="seccion" style="width:150px; height:60px; border:thin solid silver">
  Sección de contenidos...
</div>
```

6.3.2. Información sobre los eventos de teclado

De todos los eventos disponibles en JavaScript, los eventos relacionados con el teclado son los más incompatibles entre diferentes navegadores y por tanto, los más difíciles de manejar. En primer lugar, existen muchas diferencias entre los navegadores, los teclados y los sistemas operativos de los usuarios, principalmente debido a las diferencias entre idiomas.

Además, existen tres eventos diferentes para las pulsaciones de las teclas (`onkeyup`, `onkeypress` y `onkeydown`). Por último, existen dos tipos de teclas: las teclas *normales* (como letras, números y símbolos normales) y las teclas *especiales* (como `ENTER`, `Alt`, `Shift`, etc.)

Cuando un usuario pulsa una tecla normal, se producen tres eventos seguidos y en este orden: `onkeydown`, `onkeypress` y `onkeyup`. El evento `onkeydown` se corresponde con el hecho de pulsar una tecla y no soltarla; el evento `onkeypress` es la propia pulsación de la tecla y el evento `onkeyup` hace referencia al hecho de soltar una tecla que estaba pulsada.

La forma más sencilla de obtener la información sobre la tecla que se ha pulsado es mediante el evento `onkeypress`. La información que proporcionan los eventos `onkeydown` y `onkeyup` se puede considerar como más técnica, ya que devuelven el código interno de cada tecla y no el carácter que se ha pulsado.

A continuación se incluye una lista con todas las propiedades diferentes de todos los eventos de teclado tanto en Internet Explorer como en el resto de navegadores:

- Evento `keydown`:
    - Mismo comportamiento en todos los navegadores:
        - Propiedad `keyCode`: código interno de la tecla
        - Propiedad `charCode`: no definido

- Evento `keypress`:
  - Internet Explorer:
    - Propiedad `keyCode`: el código del carácter de la tecla que se ha pulsado
    - Propiedad `charCode`: no definido
  - Resto de navegadores:
    - Propiedad `keyCode`: para las teclas normales, no definido. Para las teclas especiales, el código interno de la tecla.
    - Propiedad `charCode`: para las teclas normales, el código del carácter de la tecla que se ha pulsado. Para las teclas especiales, `0`.
- Evento `keyup`:
  - Mismo comportamiento en todos los navegadores:
    - Propiedad `keyCode`: código interno de la tecla
    - Propiedad `charCode`: no definido

Para convertir el código de un carácter (no confundir con el código interno) al carácter que representa la tecla que se ha pulsado, se utiliza la función `String.fromCharCode()`.

A continuación se incluye un script que muestra toda la información sobre los tres eventos de teclado:

```
window.onload = function() {
  document.onkeyup = muestraInformacion;
  document.onkeypress = muestraInformacion;
  document.onkeydown = muestraInformacion;
}

function muestraInformacion(elEvento) {
  var evento = window.event || elEvento;

  var mensaje = "Tipo de evento: " + evento.type + "<br>" +
                "Propiedad keyCode: " + evento.keyCode + "<br>" +
                "Propiedad charCode: " + evento.charCode + "<br>" +
                "Carácter pulsado: " + String.fromCharCode(evento.charCode);

  info.innerHTML += "<br>--------------------------------------<br>" + mensaje
}

...

<div id="info"></div>
```

Al pulsar la tecla a en el navegador Firefox, se muestra la siguiente sucesión de eventos:

```
-------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 65
Propiedad charCode: 0
```

```
Carácter pulsado: ?
-------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 0
Propiedad charCode: 97
Carácter pulsado: a
-------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 65
Propiedad charCode: 0
Carácter pulsado: ?
```

Al pulsar la tecla `A` (la misma tecla, pero habiendo activado previamente las mayúsculas) se muestra la siguiente sucesión de eventos en el navegador Firefox:

```
-------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 65
Propiedad charCode: 0
Carácter pulsado: ?
-------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 0
Propiedad charCode: 65
Carácter pulsado: A
-------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 65
Propiedad charCode: 0
Carácter pulsado: ?
```

En los eventos `keydown` y `keyup`, la propiedad `keyCode` sigue valiendo lo mismo en los dos casos. El motivo es que `keyCode` almacena el código interno de la tecla, por lo que si se pulsa la misma tecla, se obtiene el mismo código, independientemente de que una misma tecla puede producir caracteres diferentes (por ejemplo mayúsculas y minúsculas).

En el evento `keypress`, el valor de la propiedad `charCode` varía, ya que el carácter `a`, no es el mismo que el carácter `A`. En este caso, el valor de `charCode` coincide con el código ASCII del carácter pulsado.

Siguiendo en el navegador Firefox, si ahora se pulsa una tecla *especial*, como por ejemplo el tabulador, se muestra la siguiente información:

```
-------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 9
Propiedad charCode: 0
Carácter pulsado: ?
-------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 9
Propiedad charCode: 0
Carácter pulsado: ?
-------------------------------------
```

```
Tipo de evento: keyup
Propiedad keyCode: 9
Propiedad charCode: 0
Carácter pulsado: ?
```

Las teclas especiales no disponen de la propiedad `charCode`, ya que sólamente se guarda el código interno de la tecla pulsada en la propiedad `keyCode`, en este caso el código `9`. Si se pulsa la tecla `Enter`, se obtiene el código `13`, la tecla de la flecha superior produce el código `38`, etc. No obstante, dependiendo del teclado utilizado para pulsar las teclas y dependiendo de la disposición de las teclas en función del idioma del teclado, estos códigos podrían variar.

A continuación se muestra el resultado de la ejecución del mismo ejemplo anterior en el navegador Internet Explorer. Al pulsar la tecla `a`, se obtiene la siguiente información:

```
-------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
-------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 97
Propiedad charCode: undefined
Carácter pulsado:
-------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
```

La propiedad `keyCode` en el evento `keypress` contiene el código ASCII del carácter de la tecla, por lo que se puede obtener directamente el carácter mediante `String.fromCharCode(keyCode)`.

Si se pulsa la tecla `A`, la información mostrada es idéntica a la anterior, salvo que el código que muestra el evento `keypress` cambia por `65`, que es el código ASCII de la tecla `A`:

```
-------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
-------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
-------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
```

Al pulsar una tecla *especial* como el tabulador, Internet Explorer muestra la siguiente información:

```
-------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 9
Propiedad charCode: undefined
Carácter pulsado:
```

Los códigos mostrados para las teclas especiales coinciden con los de Firefox y el resto de navegadores, pero recuerda que pueden variar en función del teclado que se utiliza y en función de la disposición de las teclas para cada idioma.

Por último, las propiedades `altKey`, `ctrlKey` y `shiftKey` almacenan un valor booleano que indica si alguna de esas teclas estaba pulsada al producirse el evento del teclado. Sorprendentemente, estas tres propiedades funcionan de la misma forma en todos los navegadores:

```
if(evento.altKey) {
  alert('Estaba pulsada la tecla ALT');
}
```

A continuación se muestra el caso en el que se pulsa la tecla `Shift` y sin soltarla, se pulsa sobre la tecla que contiene el número `2` (en este caso, se refiere a la tecla que se encuentra en la parte superior del teclado y por tanto, no se refiere a la que se encuentra en el teclado numérico). Tanto Internet Explorer como Firefox muestran la misma secuencia de eventos:

```
-------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 16
Propiedad charCode: 0
Carácter pulsado: ?
-------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 50
Propiedad charCode: 0
Carácter pulsado: ?
-------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 0
Propiedad charCode: 34
Carácter pulsado: "
-------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 50
Propiedad charCode: 0
Carácter pulsado: ?
-------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 16
Propiedad charCode: 0
Carácter pulsado: ?
```

El evento `keypress` es el único que permite obtener el *carácter realmente pulsado*, ya que al pulsar sobre la tecla `2` habiendo pulsado la tecla `Shift` previamente, se obtiene el carácter ", que es precisamente el que muestra el evento `keypress`.

El siguiente código de JavaScript permite obtener de forma correcta en cualquier navegador el carácter correspondiente a la tecla pulsada:

```
function manejador(elEvento) {
  var evento = elEvento || window.event;
  var caracter = evento.charCode || evento.keyCode;
  alert("El carácter pulsado es: " + String.fromCharCode(caracter));
}

document.onkeypress = manejador;
```

6.3.3. Información sobre los eventos de ratón

La información más relevante sobre los eventos relacionados con el ratón es la de las coordenadas de la posición del puntero del ratón. Aunque el origen de las coordenadas siempre se encuentra en la esquina superior izquierda, el punto que se toma como referencia de las coordenadas puede variar.

De esta forma, es posible obtener la posición del ratón respecto de la pantalla del ordenador, respecto de la ventana del navegador y respecto de la propia página HTML (que se utiliza cuando el usuario ha hecho *scroll* sobre la página). Las coordenadas más sencillas son las que se refieren a la posición del puntero respecto de la ventana del navegador, que se obtienen mediante las propiedades `clientX` y `clientY`:

```
function muestraInformacion(elEvento) {
  var evento = elEvento || window.event;
  var coordenadaX = evento.clientX;
  var coordenadaY = evento.clientY;
  alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " + coordenadaY);
}

document.onclick = muestraInformacion;
```

Las coordenadas de la posición del puntero del ratón respecto de la pantalla completa del ordenador del usuario se obtienen de la misma forma, mediante las propiedades `screenX` y `screenY`:

```
var coordenadaX = evento.screenX;
var coordenadaY = evento.screenY;
```

En muchas ocasiones, es necesario obtener otro par de coordenadas diferentes: las que corresponden a la posición del ratón respecto del origen de la página. Estas coordenadas no siempre coinciden con las coordenadas respecto del origen de la ventana del navegador, ya que el usuario puede hacer *scroll* sobre la página web. Internet Explorer no proporciona estas coordenadas de forma directa, mientras que el resto de navegadores sí que lo hacen. De esta forma, es necesario detectar si el navegador es de tipo Internet Explorer y en caso afirmativo realizar un cálculo sencillo:

```
// Detectar si el navegador es Internet Explorer
var ie = navigator.userAgent.toLowerCase().indexOf('msie')!=-1;

if(ie) {
  coordenadaX = evento.clientX + document.body.scrollLeft;
```

```
  coordenadaY = evento.clientY + document.body.scrollTop;
}
else {
  coordenadaX = evento.pageX;
  coordenadaY = evento.pageY;
}
alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " + coordenadaY + "
respecto de la página web");
```

La variable `ie` vale `true` si el navegador en el que se ejecuta el script es de tipo Internet Explorer (cualquier versión) y vale `false` en otro caso. Para el resto de navegadores, las coordenadas respecto del origen de la página se obtienen mediante las propiedades `pageX` y `pageY`. En el caso de Internet Explorer, se obtienen sumando la posición respecto de la ventana del navegador (`clientX`, `clientY`) y el desplazamiento que ha sufrido la página (`document.body.scrollLeft`, `document.body.scrollTop`).

Ejercicio 15

Completar el código JavaScript proporcionado para que:

1. Al mover el ratón en cualquier punto de la ventana del navegador, se muestre la posición del puntero respecto del navegador y respecto de la página:

Figura 6.1. Información que se muestra para los eventos del ratón

   Para mostrar los mensajes, utilizar la función `muestraInformacion()` deduciendo su funcionamiento a partir de su código fuente.
2. Al pulsar cualquier tecla, el mensaje mostrado debe cambiar para indicar el nuevo evento y su información asociada:

Figura 6.2. Información que se muestra para los eventos del teclado

3. Añadir la siguiente característica al script: cuando se pulsa un botón del ratón, el color de fondo del cuadro de mensaje debe ser amarillo (`#FFFFCC`) y cuando se pulsa una tecla, el

color de fondo debe ser azul (`#CCE6FF`). Al volver a mover el ratón, el color de fondo vuelve a ser blanco.

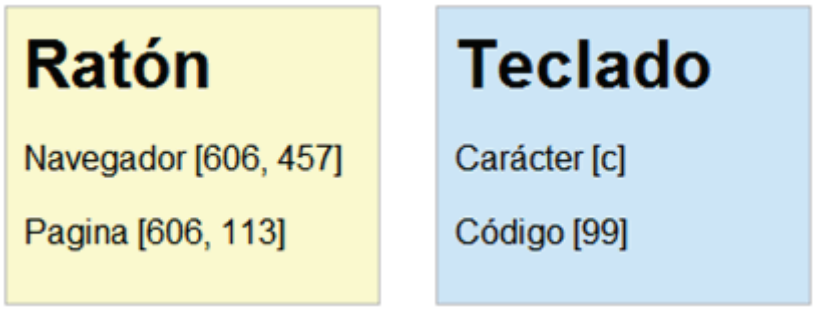

Figura 6.3. El color de fondo del cuadro de información cambia en función del tipo de evento

Ejercicio 16

Crear un script que informe al usuario en que zona de la pantalla ha pulsado el ratón. Las zonas definidas son las siguientes: izquierda arriba, izquierda abajo, derecha arriba y derecha abajo. Para determinar el tamaño de la ventana del navegador, utilizar la función `tamanoVentanaNavegador()` proporcionada.

Capítulo 7. Formularios

La programación de aplicaciones que contienen formularios web siempre ha sido una de las tareas fundamentales de JavaScript. De hecho, una de las principales razones por las que se inventó el lenguaje de programación JavaScript fue la necesidad de validar los datos de los formularios directamente en el navegador del usuario. De esta forma, se evitaba recargar la página cuando el usuario cometía errores al rellenar los formularios.

No obstante, la aparición de las aplicaciones AJAX ha relevado al tratamiento de formularios como la principal actividad de JavaScript. Ahora, el principal uso de JavaScript es el de las comunicaciones asíncronas con los servidores y el de la manipulación dinámica de las aplicaciones. De todas formas, el manejo de los formularios sigue siendo un requerimiento imprescindible para cualquier programador de JavaScript.

7.1. Propiedades básicas de formularios y elementos

JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios. En primer lugar, cuando se carga una página web, el navegador crea automáticamente un array llamado `forms` y que contiene la referencia a todos los formularios de la página.

Para acceder al array `forms`, se utiliza el objeto `document`, por lo que `document.forms` es el array que contiene todos los formularios de la página. Como se trata de un array, el acceso a cada formulario se realiza con la misma sintaxis de los arrays. La siguiente instrucción accede al primer formulario de la página:

```
document.forms[0];
```

Además del array de formularios, el navegador crea automáticamente un array llamado `elements` por cada uno de los formularios de la página. Cada array `elements` contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario. Utilizando la sintaxis de los arrays, la siguiente instrucción obtiene el primer elemento del primer formulario de la página:

```
document.forms[0].elements[0];
```

La sintaxis de los arrays no siempre es tan concisa. El siguiente ejemplo muestra cómo obtener directamente el último elemento del primer formulario de la página:

```
document.forms[0].elements[document.forms[0].elements.length-1];
```

Aunque esta forma de acceder a los formularios es rápida y sencilla, tiene un inconveniente muy grave. ¿Qué sucede si cambia el diseño de la página y en el código HTML se cambia el orden de los formularios originales o se añaden nuevos formularios? El problema es que *"el primer formulario de la página"* ahora podría ser otro formulario diferente al que espera la aplicación.

En un entorno tan cambiante como el diseño web, es muy difícil confiar en que el orden de los formularios se mantenga estable en una página web. Por este motivo, siempre debería evitarse el acceso a los formularios de una página mediante el array `document.forms`.

Una forma de evitar los problemas del método anterior consiste en acceder a los formularios de una página a través de su nombre (atributo `name`) o a través de su atributo `id`. El objeto `document` permite acceder directamente a cualquier formulario mediante su atributo `name`:

```
var formularioPrincipal = document.formulario;
var formularioSecundario = document.otro_formulario;

<form name="formulario" >
  ...
</form>

<form name="otro_formulario" >
  ...
</form>
```

Accediendo de esta forma a los formularios de la página, el script funciona correctamente aunque se reordenen los formularios o se añadan nuevos formularios a la página. Los elementos de los formularios también se pueden acceder directamente mediante su atributo `name`:

```
var formularioPrincipal = document.formulario;
var primerElemento = document.formulario.elemento;

<form name="formulario">
  <input type="text" name="elemento" />
</form>
```

Obviamente, también se puede acceder a los formularios y a sus elementos utilizando las funciones DOM de acceso directo a los nodos. El siguiente ejemplo utiliza la habitual función `document.getElementById()` para acceder de forma directa a un formulario y a uno de sus elementos:

```
var formularioPrincipal = document.getElementById("formulario");
var primerElemento = document.getElementById("elemento");

<form name="formulario" id="formulario" >
  <input type="text" name="elemento" id="elemento" />
</form>
```

Independientemente del método utilizado para obtener la referencia a un elemento de formulario, cada elemento dispone de las siguientes propiedades útiles para el desarrollo de las aplicaciones:

- `type`: indica el tipo de elemento que se trata. Para los elementos de tipo `<input>` (`text`, `button`, `checkbox`, etc.) coincide con el valor de su atributo `type`. Para las listas desplegables normales (elemento `<select>`) su valor es `select-one`, lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es `select-multiple`. Por último, en los elementos de tipo `<textarea>`, el valor de `type` es `textarea`.

- `form`: es una referencia directa al formulario al que pertenece el elemento. Así, para acceder al formulario de un elemento, se puede utilizar `document.getElementById("id_del_elemento").form`
- `name`: obtiene el valor del atributo `name` de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar.
- `value`: permite leer y modificar el valor del atributo `value` de XHTML. Para los campos de texto (`<input type="text">` y `<textarea>`) obtiene el texto que ha escrito el usuario. Para los botones obtiene el texto que se muestra en el botón. Para los elementos *checkbox* y *radiobutton* no es muy útil, como se verá más adelante

Por último, los eventos más utilizados en el manejo de los formularios son los siguientes:

- `onclick`: evento que se produce cuando se pincha con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones que permite definir XHTML (`<input type="button">`, `<input type="submit">`, `<input type="image">`).
- `onchange`: evento que se produce cuando el usuario cambia el valor de un elemento de texto (`<input type="text">` o `<textarea>`). También se produce cuando el usuario selecciona una opción en una lista desplegable (`<select>`). Sin embargo, el evento sólo se produce si después de realizar el cambio, el usuario *pasa* al siguiente campo del formulario, lo que técnicamente se conoce como que *"el otro campo de formulario ha perdido el foco"*.
- `onfocus`: evento que se produce cuando el usuario selecciona un elemento del formulario.
- `onblur`: evento complementario de `onfocus`, ya que se produce cuando el usuario ha *deseleccionado* un elemento por haber seleccionado otro elemento del formulario. Técnicamente, se dice que el elemento anterior *"ha perdido el foco"*.

7.2. Utilidades básicas para formularios

7.2.1. Obtener el valor de los campos de formulario

La mayoría de técnicas JavaScript relacionadas con los formularios requieren leer y/o modificar el valor de los campos del formulario. Por tanto, a continuación se muestra cómo obtener el valor de los campos de formulario más utilizados.

7.2.1.1. Cuadro de texto y textarea

El valor del texto mostrado por estos elementos se obtiene y se establece directamente mediante la propiedad `value`.

```
<input type="text" id="texto" />
var valor = document.getElementById("texto").value;

<textarea id="parrafo"></textarea>
var valor = document.getElementById("parrafo").value;
```

7.2.1.2. Radiobutton

Cuando se dispone de un grupo de *radiobuttons*, generalmente no se quiere obtener el valor del atributo `value` de alguno de ellos, sino que lo importante es conocer cuál de todos los *radiobuttons* se ha seleccionado. La propiedad `checked` devuelve `true` para el *radiobutton* seleccionado y `false` en cualquier otro caso. Si por ejemplo se dispone del siguiente grupo de *radiobuttons*:

```
<input type="radio" value="si" name="pregunta" id="pregunta_si"/> SI
<input type="radio" value="no" name="pregunta" id="pregunta_no"/> NO
<input type="radio" value="nsnc" name="pregunta" id="pregunta_nsnc"/> NS/NC
```

El siguiente código permite determinar si cada *radiobutton* ha sido seleccionado o no:

```
var elementos = document.getElementsByName("pregunta");

for(var i=0; i<elementos.length; i++) {
  alert(" Elemento: " + elementos[i].value + "\n Seleccionado: " +
elementos[i].checked);
}
```

7.2.1.3. Checkbox

Los elementos de tipo *checkbox* son muy similares a los *radiobutton*, salvo que en este caso se debe comprobar cada *checkbox* de forma independiente del resto. El motivo es que los grupos de *radiobutton* son mutuamente excluyentes y sólo se puede seleccionar uno de ellos cada vez. Por su parte, los *checkbox* se pueden seleccionar de forma independiente respecto de los demás.

Si se dispone de los siguientes *checkbox*:

```
<input type="checkbox" value="condiciones" name="condiciones" id="condiciones"/> He
leído y acepto las condiciones
<input type="checkbox" value="privacidad" name="privacidad" id="privacidad"/> He leído
la política de privacidad
```

Utilizando la propiedad `checked`, es posible comprobar si cada *checkbox* ha sido seleccionado:

```
var elemento = document.getElementById("condiciones");
alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);

elemento = document.getElementById("privacidad");
alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);
```

7.2.1.4. Select

Las listas desplegables (`<select>`) son los elementos en los que es más difícil obtener su valor. Si se dispone de una lista desplegable como la siguiente:

```
<select id="opciones" name="opciones">
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
  <option value="3">Tercer valor</option>
  <option value="4">Cuarto valor</option>
</select>
```

En general, lo que se requiere es obtener el valor del atributo `value` de la opción (`<option>`) seleccionada por el usuario. Obtener este valor no es sencillo, ya que se deben realizar una serie de pasos. Además, para obtener el valor seleccionado, deben utilizarse las siguientes propiedades:

- `options`, es un array creado automáticamente por el navegador para cada lista desplegable y que contiene la referencia a todas las opciones de esa lista. De esta forma, la primera opción de una lista se puede obtener mediante `document.getElementById("id_de_la_lista").options[0]`.
- `selectedIndex`, cuando el usuario selecciona una opción, el navegador actualiza automáticamente el valor de esta propiedad, que guarda el índice de la opción seleccionada. El índice hace referencia al array `options` creado automáticamente por el navegador para cada lista.

```
// Obtener la referencia a la lista
var lista = document.getElementById("opciones");

// Obtener el índice de la opción que se ha seleccionado
var indiceSeleccionado = lista.selectedIndex;
// Con el índice y el array "options", obtener la opción seleccionada
var opcionSeleccionada = lista.options[indiceSeleccionado];

// Obtener el valor y el texto de la opción seleccionada
var textoSeleccionado = opcionSeleccionada.text;
var valorSeleccionado = opcionSeleccionada.value;

alert("Opción seleccionada: " + textoSeleccionado + "\n Valor de la opción: " +
valorSeleccionado);
```

Como se ha visto, para obtener el valor del atributo `value` correspondiente a la opción seleccionada por el usuario, es necesario realizar varios pasos. No obstante, normalmente se abrevian todos los pasos necesarios en una única instrucción:

```
var lista = document.getElementById("opciones");

// Obtener el valor de la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].value;

// Obtener el texto que muestra la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].text;
```

Lo más importante es no confundir el valor de la propiedad `selectedIndex` con el valor correspondiente a la propiedad `value` de la opción seleccionada. En el ejemplo anterior, la primera opción tiene un `value` igual a `1`. Sin embargo, si se selecciona esta opción, el valor de `selectedIndex` será `0`, ya que es la primera opción del array `options` (y los arrays empiezan a contar los elementos en el número `0`).

7.2.2. Establecer el foco en un elemento

En programación, cuando un elemento está seleccionado y se puede escribir directamente en el o se puede modificar alguna de sus propiedades, se dice que tiene el foco del programa.

Si un cuadro de texto de un formulario tiene el foco, el usuario puede escribir directamente en el sin necesidad de pinchar previamente con el ratón en el interior del cuadro. Igualmente, si una lista desplegable tiene el foco, el usuario puede seleccionar una opción directamente subiendo y bajando con las flechas del teclado.

Al pulsar repetidamente la tecla TABULADOR sobre una página web, los diferentes elementos (enlaces, imágenes, campos de formulario, etc.) van obteniendo el foco del navegador (el elemento seleccionado cada vez suele mostrar un pequeño borde punteado).

Si en una página web el formulario es el elemento más importante, como por ejemplo en una página de búsqueda o en una página con un formulario para registrarse, se considera una buena práctica de usabilidad el asignar automáticamente el foco al primer elemento del formulario cuando se carga la página.

Para asignar el foco a un elemento de XHTML, se utiliza la función `focus()`. El siguiente ejemplo asigna el foco a un elemento de formulario cuyo atributo `id` es igual a `primero`:

```
document.getElementById("primero").focus();

<form id="formulario" action="#">
  <input type="text" id="primero" />
</form>
```

Ampliando el ejemplo anterior, se puede asignar automáticamente el foco del programa al primer elemento del primer formulario de la página, independientemente del `id` del formulario y de los elementos:

```
if(document.forms.length > 0) {
  if(document.forms[0].elements.length > 0) {
    document.forms[0].elements[0].focus();
  }
}
```

El código anterior comprueba que existe al menos un formulario en la página mediante el tamaño del array `forms`. Si su tamaño es mayor que `0`, se utiliza este primer formulario. Empleando la misma técnica, se comprueba que el formulario tenga al menos un elemento (`if(document.forms[0].elements.length > 0)`). En caso afirmativo, se establece el foco del navegador en el primer elemento del primer formulario (`document.forms[0].elements[0].focus();`).

Para que el ejemplo anterior sea completamente correcto, se debe añadir una comprobación adicional. El campo de formulario que se selecciona no debería ser de tipo `hidden`:

```
if(document.forms.length > 0) {
  for(var i=0; i < document.forms[0].elements.length; i++) {
    var campo = document.forms[0].elements[i];
    if(campo.type != "hidden") {
      campo.focus();
      break;
    }
  }
}
```

7.2.3. Evitar el envío duplicado de un formulario

Uno de los problemas habituales con el uso de formularios web es la posibilidad de que el usuario pulse dos veces seguidas sobre el botón "Enviar". Si la conexión del usuario es demasiado lenta o la respuesta del servidor se hace esperar, el formulario original sigue mostrándose en el navegador y por ese motivo, el usuario tiene la tentación de volver a pinchar sobre el botón de "Enviar".

En la mayoría de los casos, el problema no es grave e incluso es posible controlarlo en el servidor, pero puede complicarse en formularios de aplicaciones importantes como las que implican transacciones económicas.

Por este motivo, una buena práctica en el diseño de aplicaciones web suele ser la de deshabilitar el botón de envío después de la primera pulsación. El siguiente ejemplo muestra el código necesario:

```
<form id="formulario" action="#">
  ...
  <input type="button" value="Enviar" onclick="this.disabled=true;
this.value='Enviando...'; this.form.submit()" />
</form>
```

Cuando se pulsa sobre el botón de envío del formulario, se produce el evento `onclick` sobre el botón y por tanto, se ejecutan las instrucciones JavaScript contenidas en el atributo `onclick`:

1. En primer lugar, se deshabilita el botón mediante la instrucción `this.disabled = true;`. Esta es la única instrucción necesaria si sólo se quiere deshabilitar un botón.
2. A continuación, se cambia el mensaje que muestra el botón. Del original "Enviar" se pasa al más adecuado "Enviando..."
3. Por último, se envía el formulario mediante la función `submit()` en la siguiente instrucción: `this.form.submit()`

El botón del ejemplo anterior está definido mediante un botón de tipo `<input type="button" />`, ya que el código JavaScript mostrado no funciona correctamente con un botón de tipo `<input type="submit" />`. Si se utiliza un botón de tipo `submit`, el botón se deshabilita antes de enviar el formulario y por tanto el formulario acaba sin enviarse.

7.2.4. Limitar el tamaño de caracteres de un textarea

La carencia más importante de los campos de formulario de tipo textarea es la imposibilidad de limitar el máximo número de caracteres que se pueden introducir, de forma similar al atributo `maxlength` de los cuadros de texto normales.

JavaScript permite añadir esta característica de forma muy sencilla. En primer lugar, hay que recordar que con algunos eventos (como `onkeypress`, `onclick` y `onsubmit`) se puede evitar su comportamiento normal si se devuelve el valor `false`.

Evitar el comportamiento normal equivale a modificar completamente el comportamiento habitual del evento. Si por ejemplo se devuelve el valor `false` en el evento `onkeypress`, la tecla

pulsada por el usuario no se tiene en cuenta. Si se devuelve `false` en el evento `onclick` de un elemento como un enlace, el navegador no carga la página indicada por el enlace.

Si un evento devuelve el valor `true`, su comportamiento es el habitual:

```
<textarea onkeypress="return true;"></textarea>
```

En el textarea del ejemplo anterior, el usuario puede escribir cualquier carácter, ya que el evento `onkeypress` devuelve `true` y por tanto, su comportamiento es el normal y la tecla pulsada se transforma en un carácter dentro del textarea.

Sin embargo, en el siguiente ejemplo:

```
<textarea onkeypress="return false;"></textarea>
```

Como el valor devuelto por el evento `onkeypress` es igual a `false`, el navegador no ejecuta el comportamiento por defecto del evento, es decir, la tecla presionada no se transforma en ningún carácter dentro del textarea. No importa las veces que se pulsen las teclas y no importa la tecla pulsada, ese textarea no permitirá escribir ningún carácter.

Aprovechando esta característica, es sencillo limitar el número de caracteres que se pueden escribir en un elemento de tipo textarea: se comprueba si se ha llegado al máximo número de caracteres permitido y en caso afirmativo se evita el comportamiento habitual del evento y por tanto, los caracteres adicionales no se añaden al textarea:

```
function limita(maximoCaracteres) {
  var elemento = document.getElementById("texto");
  if(elemento.value.length >= maximoCaracteres ) {
    return false;
  }
  else {
    return true;
  }
}

<textarea id="texto" onkeypress="return limita(100);"></textarea>
```

En el ejemplo anterior, con cada tecla pulsada se compara el número total de caracteres del textarea con el máximo número de caracteres permitido. Si el número de caracteres es igual o mayor que el límite, se devuelve el valor `false` y por tanto, se evita el comportamiento por defecto de `onkeypress` y la tecla no se añade.

Ejercicio 17

Mejorar el ejemplo anterior indicando en todo momento al usuario el número de caracteres que aún puede escribir. Además, se debe permitir pulsar las teclas `Backspace`, `Supr.` y las flechas horizontales cuando se haya llegado al máximo número de caracteres.

7.2.5. Restringir los caracteres permitidos en un cuadro de texto

En ocasiones, puede ser útil bloquear algunos caracteres determinados en un cuadro de texto. Si por ejemplo un cuadro de texto espera que se introduzca un número, puede ser interesante no permitir al usuario introducir ningún carácter que no sea numérico.

Igualmente, en algunos casos puede ser útil impedir que el usuario introduzca números en un cuadro de texto. Utilizando el evento onkeypress y unas cuantas sentencias JavaScript, el problema se resuelve fácilmente:

```
function permite(elEvento, permitidos) {
  // Variables que definen los caracteres permitidos
  var numeros = "0123456789";
  var caracteres = " abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
  var numeros_caracteres = numeros + caracteres;
  var teclas_especiales = [8, 37, 39, 46];
  // 8 = BackSpace, 46 = Supr, 37 = flecha izquierda, 39 = flecha derecha


  // Seleccionar los caracteres a partir del parámetro de la función
  switch(permitidos) {
    case 'num':
      permitidos = numeros;
      break;
    case 'car':
      permitidos = caracteres;
      break;
    case 'num_car':
      permitidos = numeros_caracteres;
      break;
  }

  // Obtener la tecla pulsada
  var evento = elEvento || window.event;
  var codigoCaracter = evento.charCode || evento.keyCode;
  var caracter = String.fromCharCode(codigoCaracter);

  // Comprobar si la tecla pulsada es alguna de las teclas especiales
  // (teclas de borrado y flechas horizontales)
  var tecla_especial = false;
  for(var i in teclas_especiales) {
    if(codigoCaracter == teclas_especiales[i]) {
      tecla_especial = true;
      break;
    }
  }

  // Comprobar si la tecla pulsada se encuentra en los caracteres permitidos
  // o si es una tecla especial
  return permitidos.indexOf(caracter) != -1 || tecla_especial;
}

// Sólo números
<input type="text" id="texto" onkeypress="return permite(event, 'num')" />

// Sólo letras
<input type="text" id="texto" onkeypress="return permite(event, 'car')" />

// Sólo letras o números
<input type="text" id="texto" onkeypress="return permite(event, 'num_car')" />
```

El funcionamiento del script anterior se basa en permitir o impedir el comportamiento habitual del evento `onkeypress`. Cuando se pulsa una tecla, se comprueba si el carácter de esa tecla se encuentra dentro de los caracteres permitidos para ese elemento `<input>`.

Si el carácter se encuentra dentro de los caracteres permitidos, se devuelve `true` y por tanto el comportamiento de `onkeypress` es el habitual y la tecla se escribe. Si el carácter no se encuentra dentro de los caracteres permitidos, se devuelve `false` y por tanto se impide el comportamiento normal de `onkeypress` y la tecla no llega a escribirse en el input.

Además, el script anterior siempre permite la pulsación de algunas teclas *especiales*. En concreto, las teclas `BackSpace` y `Supr` para borrar caracteres y las teclas `Flecha Izquierda` y `Flecha Derecha` para moverse en el cuadro de texto siempre se puden pulsar independientemente del tipo de caracteres permitidos.

7.3. Validación

La principal utilidad de JavaScript en el manejo de los formularios es la validación de los datos introducidos por los usuarios. Antes de enviar un formulario al servidor, se recomienda validar mediante JavaScript los datos insertados por el usuario. De esta forma, si el usuario ha cometido algún error al rellenar el formulario, se le puede notificar de forma instantánea, sin necesidad de esperar la respuesta del servidor.

Notificar los errores de forma inmediata mediante JavaScript mejora la satisfacción del usuario con la aplicación (lo que técnicamente se conoce como *"mejorar la experiencia de usuario"*) y ayuda a reducir la carga de procesamiento en el servidor.

Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario. En esta función, se comprueban si los valores que ha introducido el usuario cumplen las restricciones impuestas por la aplicación.

Aunque existen tantas posibles comprobaciones como elementos de formulario diferentes, algunas comprobaciones son muy habituales: que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, que la dirección de email indicada sea correcta, que la fecha introducida sea lógica, que se haya introducido un número donde así se requiere, etc.

A continuación se muestra el código JavaScript básico necesario para incorporar la validación a un formulario:

```
<form action="" method="" id="" name="" onsubmit="return validacion()">
  ...
</form>
```

Y el esquema de la función `validacion()` es el siguiente:

```
function validacion() {
  if (condicion que debe cumplir el primer campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
```

```
  else if (condicion que debe cumplir el segundo campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  ...
  else if (condicion que debe cumplir el último campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }

  // Si el script ha llegado a este punto, todas las condiciones
  // se han cumplido, por lo que se devuelve el valor true
  return true;
}
```

El funcionamiento de esta técnica de validación se basa en el comportamiento del evento `onsubmit` de JavaScript. Al igual que otros eventos como `onclick` y `onkeypress`, el evento `'onsubmit` varía su comportamiento en función del valor que se devuelve.

Así, si el evento `onsubmit` devuelve el valor `true`, el formulario se envía como lo haría normalmente. Sin embargo, si el evento `onsubmit` devuelve el valor `false`, el formulario no se envía. La clave de esta técnica consiste en comprobar todos y cada uno de los elementos del formulario. En cuando se encuentra un elemento incorrecto, se devuelve el valor `false`. Si no se encuentra ningún error, se devuelve el valor `true`.

Por lo tanto, en primer lugar se define el evento `onsubmit` del formulario como:

```
onsubmit="return validacion()"
```

Como el código JavaScript devuelve el valor resultante de la función `validacion()`, el formulario solamente se enviará al servidor si esa función devuelve `true`. En el caso de que la función `validacion()` devuelva `false`, el formulario permanecerá sin enviarse.

Dentro de la función `validacion()` se comprueban todas las condiciones impuestas por la aplicación. Cuando no se cumple una condición, se devuelve `false` y por tanto el formulario no se envía. Si se llega al final de la función, todas las condiciones se han cumplido correctamente, por lo que se devuelve `true` y el formulario se envía.

La notificación de los errores cometidos depende del diseño de cada aplicación. En el código del ejemplo anterior simplemente se muestran mensajes mediante la función `alert()` indicando el error producido. Las aplicaciones web mejor diseñadas muestran cada mensaje de error al lado del elemento de formulario correspondiente y también suelen mostrar un mensaje principal indicando que el formulario contiene errores.

Una vez definido el esquema de la función `validacion()`, se debe añadir a esta función el código correspondiente a todas las comprobaciones que se realizan sobre los elementos del formulario. A continuación, se muestran algunas de las validaciones más habituales de los campos de formulario.

7.3.1. Validar un campo de texto obligatorio

Se trata de forzar al usuario a introducir un valor en un cuadro de texto o textarea en los que sea obligatorio. La condición en JavaScript se puede indicar como:

```
valor = document.getElementById("campo").value;
if( valor == null || valor.length == 0 || /^\s+$/.test(valor) ) {
  return false;
}
```

Para que se de por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducido sea mayor que cero y que no se hayan introducido sólo espacios en blanco.

La palabra reservada `null` es un valor especial que se utiliza para indicar *"ningún valor"*. Si el valor de una variable es `null`, la variable no contiene ningún valor de tipo objeto, array, numérico, cadena de texto o booleano.

La segunda parte de la condición obliga a que el texto introducido tenga una longitud superior a cero caracteres, esto es, que no sea un texto vacío.

Por último, la tercera parte de la condición (`/^\s+$/.test(valor)`) obliga a que el valor introducido por el usuario no sólo esté formado por espacios en blanco. Esta comprobación se basa en el uso de *"expresiones regulares"*, un recurso habitual en cualquier lenguaje de programación pero que por su gran complejidad no se van a estudiar. Por lo tanto, sólo es necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión.

7.3.2. Validar un campo de texto con valores numéricos

Se trata de obligar al usuario a introducir un valor numérico en un cuadro de texto. La condición JavaScript consiste en:

```
valor = document.getElementById("campo").value;
if( isNaN(valor) ) {
  return false;
}
```

Si el contenido de la variable `valor` no es un número válido, no se cumple la condición. La ventaja de utilizar la función interna `isNaN()` es que simplifica las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.

A continuación se muestran algunos resultados de la función `isNaN()`:

```
isNaN(3);          // false
isNaN("3");        // false
isNaN(3.3545);     // false
isNaN(32323.345);  // false
isNaN(+23.2);      // false
isNaN("-23.2");    // false
isNaN("23a");      // true
isNaN("23.43.54"); // true
```

7.3.3. Validar que se ha seleccionado una opción de una lista

Se trata de obligar al usuario a seleccionar un elemento de una lista desplegable. El siguiente código JavaScript permite conseguirlo:

```
indice = document.getElementById("opciones").selectedIndex;
if( indice == null || indice == 0 ) {
  return false;
}

<select id="opciones" name="opciones">
  <option value="">- Selecciona un valor -</option>
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
  <option value="3">Tercer valor</option>
</select>
```

A partir de la propiedad `selectedIndex`, se comprueba si el índice de la opción seleccionada es válido y además es distinto de cero. La primera opción de la lista (`- Selecciona un valor –`) no es válida, por lo que no se permite el valor `0` para esta propiedad `selectedIndex`.

7.3.4. Validar una dirección de email

Se trata de obligar al usuario a introducir una dirección de email con un formato válido. Por tanto, lo que se comprueba es que la dirección *parezca* válida, ya que no se comprueba si se trata de una cuenta de correo electrónico real y operativa. La condición JavaScript consiste en:

```
valor = document.getElementById("campo").value;
if( !(/\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)/.test(valor)) ) {
  return false;
}
```

La comprobación se realiza nuevamente mediante las expresiones regulares, ya que las direcciones de correo electrónico válidas pueden ser muy diferentes. Por otra parte, como el estándar que define el formato de las direcciones de correo electrónico es muy complejo, la expresión regular anterior es una simplificación. Aunque esta regla valida la mayoría de direcciones de correo electrónico utilizadas por los usuarios, no soporta todos los diferentes formatos válidos de email.

7.3.5. Validar una fecha

Las fechas suelen ser los campos de formulario más complicados de validar por la multitud de formas diferentes en las que se pueden introducir. El siguiente código asume que de alguna forma se ha obtenido el año, el mes y el día introducidos por el usuario:

```
var ano = document.getElementById("ano").value;
var mes = document.getElementById("mes").value;
var dia = document.getElementById("dia").value;

valor = new Date(ano, mes, dia);

if( !isNaN(valor) ) {
```

```
  return false;
}
```

La función `Date(ano, mes, dia)` es una función interna de JavaScript que permite construir fechas a partir del año, el mes y el día de la fecha. Es muy importante tener en cuenta que el número de mes se indica de `0` a `11`, siendo `0` el mes de Enero y `11` el mes de Diciembre. Los días del mes siguen una numeración diferente, ya que el mínimo permitido es `1` y el máximo `31`.

La validación consiste en intentar construir una fecha con los datos proporcionados por el usuario. Si los datos del usuario no son correctos, la fecha no se puede construir correctamente y por tanto la validación del formulario no será correcta.

7.3.6. Validar un número de DNI

Se trata de comprobar que el número proporcionado por el usuario se corresponde con un número válido de Documento Nacional de Identidad o DNI. Aunque para cada país o región los requisitos del documento de identidad de las personas pueden variar, a continuación se muestra un ejemplo genérico fácilmente adaptable. La validación no sólo debe comprobar que el número esté formado por ocho cifras y una letra, sino que también es necesario comprobar que la letra indicada es correcta para el número introducido:

```
valor = document.getElementById("campo").value;
var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J',
'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];

if( !(/^\d{8}[A-Z]$/.test(valor)) ) {
  return false;
}

if(valor.charAt(8) != letras[(valor.substring(0, 8))%23]) {
  return false;
}
```

La primera comprobación asegura que el formato del número introducido es el correcto, es decir, que está formado por 8 números seguidos y una letra. Si la letra está al principio de los números, la comprobación sería `/^[A-Z]\d{8}$/`. Si en vez de ocho números y una letra, se requieren diez números y dos letras, la comprobación sería `/^\d{10}[A-Z]{2}$/` y así sucesivamente.

La segunda comprobación aplica el algoritmo de cálculo de la letra del DNI y la compara con la letra proporcionada por el usuario. El algoritmo de cada documento de identificación es diferente, por lo que esta parte de la validación se debe adaptar convenientemente.

7.3.7. Validar un número de teléfono

Los números de teléfono pueden ser indicados de formas muy diferentes: con prefijo nacional, con prefijo internacional, agrupado por pares, separando los números con guiones, etc.

El siguiente script considera que un número de teléfono está formado por nueve dígitos consecutivos y sin espacios ni guiones entre las cifras:

```
valor = document.getElementById("campo").value;
if( !(/^\d{9}$/.test(valor)) ) {
  return false;
}
```

Una vez más, la condición de JavaScript se basa en el uso de expresiones regulares, que comprueban si el valor indicado es una sucesión de nueve números consecutivos. A continuación se muestran otras expresiones regulares que se pueden utilizar para otros formatos de número de teléfono:

Número	Expresión regular	Formato
900900900	/^\d{9}$/	9 cifras seguidas
900-900-900	/^\d{3}-\d{3}-\d{3}$/	9 cifras agrupadas de 3 en 3 y separadas por guiones
900 900900	/^\d{3}\s\d{6}$/	9 cifras, las 3 primeras separadas por un espacio
900 90 09 00	/^\d{3}\s\d{2}\s\d{2}\s\d{2}$/	9 cifras las 3 primeras separadas por un espacio las siguientes agrupadas de 2 en 2
(900) 900900	/^\(\d{3}\)\s\d{6}$/	9 cifras las 3 primeras encerradas por paréntesis y un espacio de separación respecto del resto
+34 900900900	/^\+\d{2,3}\s\d{9}$/	Prefijo internacional (+ seguido de 2 o 3 cifras) espacio en blanco y 9 cifras consecutivas

7.3.8. Validar que un checkbox ha sido seleccionado

Si un elemento de tipo *checkbox* se debe seleccionar de forma obligatoria, JavaScript permite comprobarlo de forma muy sencilla:

```
elemento = document.getElementById("campo");
if( !elemento.checked ) {
  return false;
}
```

Si se trata de comprobar que todos los *checkbox* del formulario han sido seleccionados, es más fácil utilizar un bucle:

```
formulario = document.getElementById("formulario");
for(var i=0; i<formulario.elements.length; i++) {
  var elemento = formulario.elements[i];
  if(elemento.type == "checkbox") {
    if(!elemento.checked) {
      return false;
    }
  }
}
```

7.3.9. Validar que un radiobutton ha sido seleccionado

Aunque se trata de un caso similar al de los *checkbox*, la validación de los *radiobutton* presenta una diferencia importante: en general, la comprobación que se realiza es que el usuario haya seleccionado algún *radiobutton* de los que forman un determinado grupo. Mediante JavaScript, es sencillo determinar si se ha seleccionado algún *radiobutton* de un grupo:

```
opciones = document.getElementsByName("opciones");

var seleccionado = false;
for(var i=0; i<opciones.length; i++) {
  if(opciones[i].checked) {
    seleccionado = true;
    break;
  }
}

if(!seleccionado) {
  return false;
}
```

El anterior ejemplo recorre todos los *radiobutton* que forman un grupo y comprueba elemento por elemento si ha sido seleccionado. Cuando se encuentra el primer *radiobutton* seleccionado, se sale del bucle y se indica que al menos uno ha sido seleccionado.

Capítulo 8. Otras utilidades

Además de los formularios y de todas las funciones y utilidades de JavaScript que se han visto, existen muchas otras utilidades que es necesario conocer para desarrollar aplicaciones completas. Como no es posible estudiar todas las herramientas que se pueden crear con JavaScript, a continuación se muestran algunas de las utilidades básicas más comunes.

8.1. Relojes, contadores e intervalos de tiempo

En ocasiones, algunas páginas web muestran un reloj con la hora actual. Si el reloj debe actualizarse cada segundo, no se puede mostrar la hora directamente en la página HTML generada por el servidor. En este caso, aunque existen alternativas realizadas con Java y con Flash, la forma más sencilla de hacerlo es mostrar la hora del ordenador del usuario mediante JavaScript.

Para crear y mostrar un reloj con JavaScript, se debe utilizar el objeto interno `Date()` para crear fechas/horas y las utilidades que permiten definir contadores, para actualizar el reloj cada segundo.

El objeto `Date()` es una utilidad que proporciona JavaScript para crear fechas y horas. Una vez creado un objeto de tipo fecha, es posible manipularlo para obtener información o realizar cálculos con las fechas. Para obtener la fecha y hora actuales, solamente es necesario crear un objeto `Date()` sin pasar ningún parámetro:

```
var fechaHora = new Date();
```

Utilizando el código anterior, se puede construir un reloj muy básico que no actualiza su contenido:

```
var fechaHora = new Date();
document.getElementById("reloj").innerHTML = fechaHora;

<div id="reloj" />
```

Cuando se carga la página, el ejemplo anterior mostraría un texto parecido al siguiente en el `<div>` reservado para el reloj:

```
Thu Aug 02 2007 19:35:19 GMT+0200 (Hora de verano romance)
```

Este primer reloj construido presenta muchas diferencias respecto al reloj que se quiere construir. En primer lugar, muestra mucha más información de la necesaria. Además, su valor no se actualiza cada segundo, por lo que no es un reloj muy práctico.

El objeto `Date()` proporciona unas funciones muy útiles para obtener información sobre la fecha y la hora. Concretamente, existen funciones que obtienen directamente la hora, los minutos y los segundos:

```
var fechaHora = new Date();
var horas = fechaHora.getHours();
var minutos = fechaHora.getMinutes();
```

```
var segundos = fechaHora.getSeconds();
document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

<div id="reloj" />
```

Utilizando las funciones `getHours()`, `getMinutes()` y `getSeconds()` del objeto `Date`, el reloj puede mostrar solamente la información de la hora. El resultado del ejemplo anterior sería un reloj como el siguiente:

```
20:9:21
```

Si la hora, minuto o segundo son menores que `10`, JavaScript no añade el `0` por delante, por lo que el resultado no es del todo satisfactorio. El siguiente código soluciona este problema añadiendo un `0` cuando sea necesario:

```
var fechaHora = new Date();
var horas = fechaHora.getHours();
var minutos = fechaHora.getMinutes();
var segundos = fechaHora.getSeconds();

if(horas < 10) { horas = '0' + horas; }
if(minutos < 10) { minutos = '0' + minutos; }
if(segundos < 10) { segundos = '0' + segundos; }

document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

<div id="reloj" />
```

Ahora el reloj muestra correctamente la hora:

```
20:14:03
```

Para completar el reloj, sólo falta que se actualice su valor cada segundo. Para conseguirlo, se deben utilizar unas funciones especiales de JavaScript que permiten ejecutar determinadas instrucciones cuando ha transcurrido un determinado espacio de tiempo.

La función `setTimeout()` permite ejecutar una función una vez que haya transcurrido un periodo de tiempo indicado. La definición de la función es:

```
setTimeout(nombreFuncion, milisegundos);
```

La función que se va a ejecutar se debe indicar mediante su nombre sin paréntesis y el tiempo que debe transcurrir hasta que se ejecute se indica en milisegundos. De esta forma, si se crea una función encargada de mostrar la hora del reloj y se denomina `muestraReloj()`, se puede indicar que se ejecute dentro de 1 segundo mediante el siguiente código:

```
function muestraReloj() {
  var fechaHora = new Date();
  var horas = fechaHora.getHours();
  var minutos = fechaHora.getMinutes();
  var segundos = fechaHora.getSeconds();

  if(horas < 10) { horas = '0' + horas; }
  if(minutos < 10) { minutos = '0' + minutos; }
  if(segundos < 10) { segundos = '0' + segundos; }
```

```
  document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;
}

setTimeout(muestraReloj, 1000);

<div id="reloj" />
```

No obstante, el código anterior simplemente muestra el contenido del reloj 1 segundo después de que se cargue la página, por lo que no es muy útil. Para ejecutar una función de forma periódica, se utiliza una función de JavaScript muy similar a `setTimeout()` que se denomina `setInterval()`. Su definición es:

```
setInterval(nombreFuncion, milisegundos);
```

La definición de esta función es idéntica a la función `setTimeout()`, salvo que en este caso, la función programada se ejecuta infinitas veces de forma periódica con un lapso de tiempo entre ejecuciones de tantos milisegundos como se hayan establecido.

Así, para construir el reloj completo, se establece una ejecución periódica de la función `muestraReloj()` cada segundo:

```
function muestraReloj() {
  var fechaHora = new Date();
  var horas = fechaHora.getHours();
  var minutos = fechaHora.getMinutes();
  var segundos = fechaHora.getSeconds();

  if(horas < 10) { horas = '0' + horas; }
  if(minutos < 10) { minutos = '0' + minutos; }
  if(segundos < 10) { segundos = '0' + segundos; }

  document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;
}

window.onload = function() {
  setInterval(muestraReloj, 1000);
}

<div id="reloj" />
```

Empleando el objeto `Date` y sus funciones, es posible construir *"cuentras atrás"*, es decir, relojes que muestran el tiempo que falta hasta que se produzca un evento. Además, las funciones `setTimeout()` y `setInterval()` pueden resultar muy útiles en otras técnicas de programación.

8.2. Calendario

Cuando una aplicación muestra un formulario para que el usuario inserte sus datos, la información más complicada de obtener siempre suelen ser los números de teléfono y las fechas. El motivo es que existen muchas formas diferentes de indicar estos datos. Las fechas por ejemplo se pueden indicar como `dd/mm/aa`, `dd/mm/aaaa`, `aaaa/mm/dd`, `dd-mm-aa`, `dd mm aaaa`, etc.

Los métodos más utilizados para que el usuario introduzca la información relativa a una fecha suelen ser un cuadro de texto en el que tiene que insertar la fecha completa (indicándole el formato que debe seguir) o un cuadro de texto para el día, una lista desplegable para el mes y otro cuadro de texto para el año.

En cualquier caso, para el usuario suele ser más cómodo que la aplicación incluya un calendario en el que pueda indicar la fecha pinchando sobre el día elegido:

Figura 8.1. Aspecto de un calendario creado con JavaScript

Además, este método es menos propenso a cometer errores, ya que si el calendario está bien construido, no es posible introducir fechas inválidas y tampoco es posible insertar las fechas en un formato incorrecto.

No obstante, realizar un calendario completo en JavaScript no es una tarea trivial, por lo que no se va a estudiar su desarrollo completo. Afortunadamente, existen scripts gratuitos para mostrar calendarios y que permiten su uso libre incluso en aplicaciones comerciales.

De entre todos los calendarios disponibles, uno de los más completos es el desarrollado por la empresa DynArch, que se puede acceder desde su página web oficial: http://www.dynarch.com/projects/calendar/ y que se puede descargar gratuitamente desde http://sourceforge.net/projects/jscalendar/ El archivo descargado incluye todos los scripts necesarios, su documentación, ejemplos de uso, diferentes estilos CSS para el calendario, etc.

A continuación se indican los pasos necesarios para incluir un calendario básico en cualquier página web:

1) Enlazar los archivos JavaScript y CSS requeridos:

Se descomprime el archivo descargado, se guardan los archivos JavaScript y CSS en el sitio adecuado (en este ejemplo se supone que los archivos JavaScript se guardan en el directorio `js/` y los archivos CSS en el directorio `css/`) y se enlazan directamente desde la cabecera de la página HTML.

```
<head>
...
<style type="text/css">@import url("css/calendar-estilo.css");</style>
<script type="text/javascript" src="js/calendar.js" />
<script type="text/javascript" src="js/calendar-es.js" />
<script type="text/javascript" src="js/calendar-setup.js" />
...
</head>
```

El calendario incluye traducciones a más de 40 idiomas, entre los que se encuentra el español. Para mostrar el calendario en un determinado idioma, tan solo es necesario enlazar el archivo del idioma correspondiente. Las traducciones se encuentran en el directorio `lang` y su formato es `calendar-XX.js`, donde `XX` es el código del idioma.

2) Añadir el código XHTML necesario para el elemento que va a mostrar el calendario:

```
<p>Introduce la fecha pulsando sobre la imagen del calendario</p>
<input type="text" name="date" id="fecha" readonly="readonly" />
<img src="calendario.png" id="selector" />
```

En este caso, el calendario está formado por dos elementos:

- Un cuadro de texto llamado `fecha` y que almacena el valor introducido por el usuario. Como se le ha asignado un atributo `readonly="readonly"`, el usuario no puede modificar su valor.
- Una pequeña imagen o icono que se utiliza para activar el calendario. Cuando el usuario pincha con el ratón sobre la imagen, se muestra el calendario de JavaScript.

3) Configurar el calendario:

```
<script type="text/javascript">
window.onload = function() {
  Calendar.setup({
    inputField: "fecha",
    ifFormat:   "%d / %m / %Y",
    button:     "selector"
  });
}
</script>
```

Una vez enlazados los archivos del calendario y preparado el código XHTML, es necesario inicializar y configurar el calendario. La configuración del calendario consiste en establecer el valor de alguna de sus propiedades. Las propiedades básicas imprescindibles son:

- `inputField`: se trata del atributo `id` del elemento en el que se mostrará la fecha seleccionada, en este ejemplo sería `fecha`.
- `ifFormat`: formato en el que se mostrará la fecha una vez seleccionada (en este caso se ha optado por el formato `dd / mm / aaaa`).
- `button`: atributo `id` del elemento que se pulsa (botón, imagen, enlace) para mostrar el calendario de selección de fecha. En este ejemplo, el `id` de la imagen es `selector`.

Después de esta configuración básica, el calendario ya puede utilizarse en la aplicación:

1) Aspecto por defecto del selector de calendario:

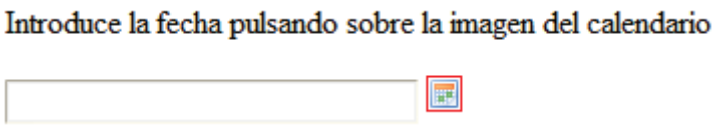

Figura 8.2. Elementos XHTML del calendario JavaScript: cuadro de texto e icono

2) Al pinchar una vez sobre la imagen del calendario:

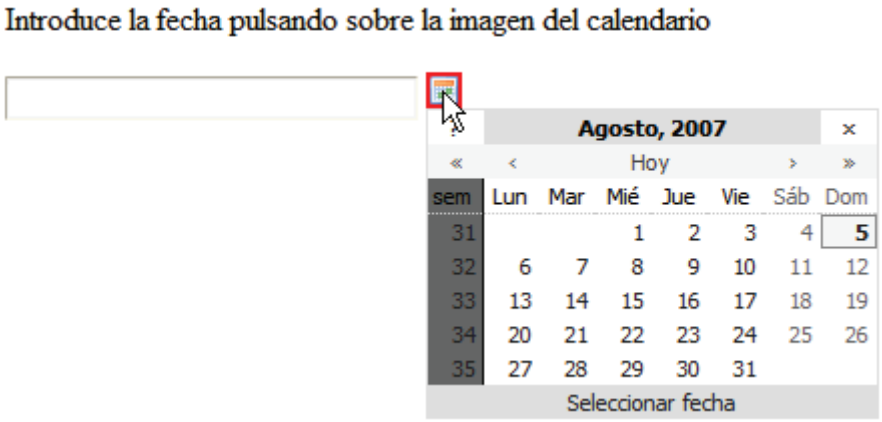

Figura 8.3. Aspecto inicial del calendario JavaScript cuando se abre por primera vez

Se muestra el calendario con el aspecto e idioma establecidos y aparece resaltada la fecha del día actual.

3) Se selecciona la fecha deseada:

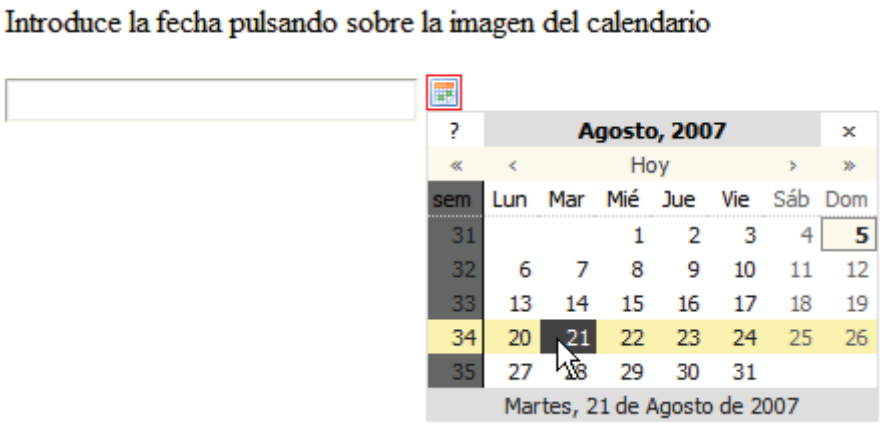

Figura 8.4. Seleccionando una fecha en el calendario JavaScript

4) Después de pinchar con el ratón sobre la fecha deseada:

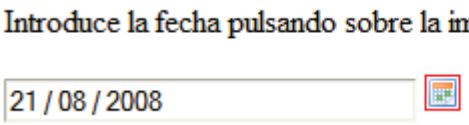

Figura 8.5. El cuadro de texto muestra la fecha establecida por el usuario con el calendario JavaScript

Al pulsar con el ratón sobre la fecha deseada, el calendario se cierra automáticamente y el cuadro de texto muestra la fecha seleccionada con el formato indicado en la configuración del calendario. Si se vuelve a abrir el calendario para seleccionar otra fecha, se mostrará resaltada la fecha del día seleccionado y no la fecha del día actual.

Ejercicio 18

Mejorar el calendario creado añadiendo las opciones necesarias para que muestre el siguiente aspecto:

Figura 8.6. Aspecto mejorado del calendario JavaScript

1. Que no se muestre el número de la semana en el calendario (pista: `weekNumbers`)
2. Modificar el formato en el que se muestra la fecha seleccionada. El formato original es `21 / 08 / 2007` (indicado como `%d / %m / %Y`). El formato deseado es `Martes, 21 de Agosto de 2007` (pistas: `%A`, `%B`)
3. El nuevo formato de fecha es mucho más agradable para los usuarios, pero más incómodo para los scripts que tienen que manejarlo. Afortunadamente, el calendario dispone de la posibilidad de guardar dos valores: un valor para mostrar a los usuarios y otro valor para que lo procesen los scripts. Cuando el usuario seleccione una fecha, la fecha con el formato original se debe almacenar en un campo oculto de formulario y el otro formato más largo debe mostrar al usuario en un elemento de tipo `<span>` (pistas: `displayArea`, `daFormat`)

8.3. Tooltip

Los *tooltips* son pequeños recuadros de información que aparecen al posicionar el ratón sobre un elemento. Normalmente se utilizan para ofrecer información adicional sobre el elemento seleccionado o para mostrar al usuario pequeños mensajes de ayuda. Los *tooltips* son habituales en varios elementos de los sistemas operativos:

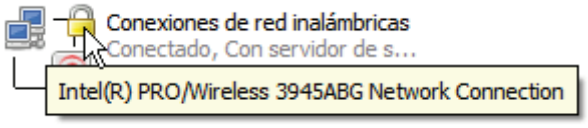

Figura 8.7. Aspecto de un tooltip típico en el Sistema Operativo

Realizar un *tooltip* completo mediante JavaScript es una tarea muy compleja, sobre todo por la dificultad de mostrar el mensaje correctamente en función de la posición del ratón. Afortunadamente, existen scripts que ya están preparados para generar cualquier tipo de *tooltip*. La librería que se va a utilizar se denomina overLIB, y se puede descargar desde su página web principal: http://www.bosrup.com/web/overlib/

La descarga incluye todos los scripts necesarios para el funcionamiento del sistema de *tooltips*, pero no su documentación, que se puede consultar en: http://www.bosrup.com/web/overlib/?Documentation. La referencia de todos los comandos disponibles se puede consultar en: http://www.bosrup.com/web/overlib/?Command_Reference

A continuación se indican los pasos necesarios para incluir un *tooltip* básico en cualquier página web:

1) Enlazar los archivos JavaScript requeridos:

```
<script type="text/javascript" src="js/overlib.js"><!-- overLIB (c) Erik Bosrup
--></script>
```

Se descomprime el archivo descargado, se guarda el archivo JavaScript en el sitio adecuado (en este ejemplo se supone que los archivos JavaScript se guardan en el directorio `js/`) y se enlaza directamente desde la cabecera de la página HTML. Los *tooltips* sólo requieren que se enlace un único archivo JavaScript (`overlib.js`). El comentario que incluye el código XHTML es el aviso de copyright de la librería, que es obligatorio incluirlo para poder usarla.

2) Definir el texto que activa el *tooltip* y su contenido:

```
<p>Lorem ipsum dolor sit amet, <a href="#" onmouseover="return overlib('Prueba de un
tooltip básico y muy sencillo.');" onmouseout="return nd();">consectetuer adipiscing
elit</a>. Etiam eget metus. Proin varius auctor tortor. Cras augue neque, porta vitae,
vestibulum nec, pulvinar id, nibh. Fusce in arcu. Duis vehicula nonummy orci.</p>
```

Los tooltip se incluyen como enlaces de tipo `<a>` para los que se definen los eventos `onmouseover` y `onmouseout`. Cuando el ratón se pasa por encima del enlace anterior, se muestra el *tooltip* con el aspecto por defecto:

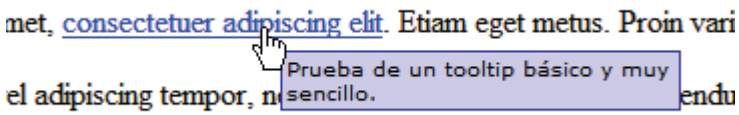

Figura 8.8. Aspecto por defecto del tooltip creado con la librería overLIB

La librería overLIB permite configurar completamente el aspecto y comportamiento de cada *tooltip*. Las opciones se indican en cada *tooltip* y se incluyen como parámetros al final de la llamada a la función `overlib()`:

```
<a href="#" onmouseover="return overlib('Prueba de un tooltip básico y muy sencillo.',
CENTER);" onmouseout="return nd();">consectetuer adipiscing elit</a>
```

El parámetro `CENTER` indica que el *tooltip* se debe mostrar centrado respecto de la posición del ratón:

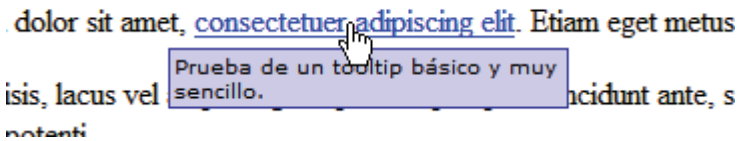

Figura 8.9. Centrando el tooltip respecto de la posición del ratón

Otra opción que se puede controlar es la anchura del *tooltip*. Por defecto, su anchura es de `200px`, pero la opción `WIDTH` permite modificarla:

```
<a href="#" onmouseover="return overlib('Prueba de un tooltip básico y muy sencillo.',
CENTER, WIDTH, 120);" onmouseout="return nd();">consectetuer adipiscing elit</a>
```

El valor de la nueva anchura se indica en el parámetro que va detrás de `WIDTH`. El nuevo aspecto del *tooltip* es el siguiente:

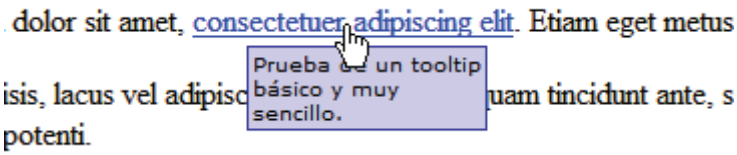

Figura 8.10. Definiendo la anchura que muestra el tooltip

El uso de los parámetros de configuración en la propia llamada a la función que muestra los *tooltips* no es recomendable cuando el número de parámetros empieza a ser muy numeroso. Afortunadamente, overLIB permite realizar una única configuración que se utilizará para todos los *tooltips* de la página.

La configuración global consiste en llamar a la función `overlib_pagedefaults()` con todos los parámetros de configuración deseados. Por tanto, el código JavaScript necesario para realizar los cambios configurados hasta el momento sería:

```
<script type="text/javascript" src="js/overlib.js"><!-- overLIB (c) Erik Bosrup
--></script>
<script type="text/javascript">
overlib_pagedefaults(CENTER, WIDTH, 120);
</script>

<p>Lorem ipsum dolor sit amet, <a href="#" onmouseover="return overlib('Prueba de un
tooltip básico y muy sencillo.');" onmouseout="return nd();">consectetuer adipiscing
elit</a>. Etiam eget metus. Proin varius auctor tortor. Cras augue neque, porta vitae,
vestibulum nec, pulvinar id, nibh. Fusce in arcu. Duis vehicula nonummy orci.</p>
```

Utilizando esta configuración global, se va a modificar por último el aspecto del *tooltip* para hacerlo más parecido a los tooltips de los sistemas operativos:

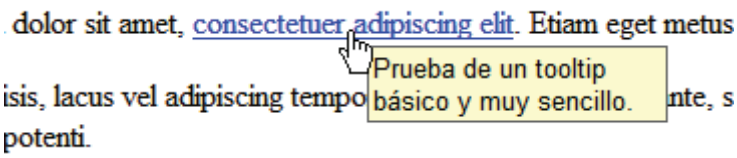

Figura 8.11. Tooltip creado con la librería overLIB y personalizado para que parezca un tooltip de Sistema Operativo

En el ejemplo anterior se ha modificado el tamaño y tipo de letra y el color de fondo del *tooltip* mediante la siguiente configuración:

```
overlib_pagedefaults(WIDTH,150,FGCOLOR,'#ffffcc',BGCOLOR,'#666666',TEXTFONT,"Arial,
Helvetica, Verdana",TEXTSIZE,".8em");
```

Ejercicio 19

Mejorar el *tooltip* propuesto añadiendo las siguientes características:

1. Que el *tooltip* no se muestre instantáneamente, sino que transcurra un cuarto de segundo hasta que se muestre (pista: `DELAY`)

2. Que exista una separación horizontal de 15 píxel entre el puntero del ratón y el *tooltip* (pista: OFFSETX)
3. Que el *tooltip* se muestre en la parte superior del puntero del ratón y no en la parte inferior (pista: ABOVE)

El nuevo aspecto del tooltip se muestra en la siguiente imagen:

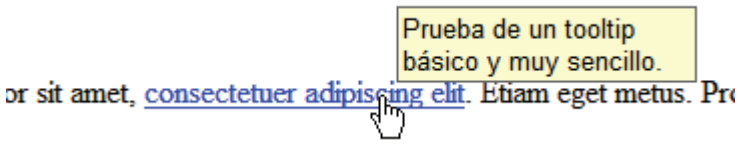

Figura 8.12. Nuevo aspecto del tooltip

8.4. Menú desplegable

La navegación de algunas páginas web se realiza mediante menús desplegables, que disponen de varios niveles jerárquicos que se despliegan a medida que el usuario pasa el puntero del ratón por encima de cada elemento. Aunque CSS permite realizar menús desplegables horizontales y verticales de cualquier nivel de profundidad, los navegadores de la familia Internet Explorer versión 6.0 y anteriores no disponen del suficiente soporte de CSS como para crear este tipo de menús.

De esta forma, la única forma de crear un menú desplegable que funcione correctamente en cualquier navegador consiste en utilizar JavaScript. A pesar de que realizar un menú desplegable sencillo con JavaScript no es una tarea muy compleja, se va a utilizar un componente ya realizado que pertenece a la librería de desarrollo web de Yahoo.

La *Yahoo UI Library*, conocida como YUI, y que se puede traducir como *"Librería de componentes de interfaz de usuario de Yahoo"* es un conjunto de utilidades y controles escritos en JavaScript para el desarrollo rápido y sencillo de aplicaciones web complejas.

La librería completa está dividida en módulos y componentes relacionados con CSS, DOM, eventos, AJAX, etc. Entre las utilidades disponibles se encuentran calendarios, *tooltips*, cuadros que autocompletan el texto, árboles jerárquicos, etc. Además de esas utilidades, la YUI incluye un completo módulo de menús que permite realizar decenas de tipos de menús diferentes: horizontales, verticales, desplegables, estáticos, menús contextuales, menús de aplicación, menús realizados con XHTML o con JavaScript, etc.

La librería YUI se puede descargar gratuitamente en http://developer.yahoo.com/yui/, la información específica sobre los menús se puede encontrar en http://developer.yahoo.com/yui/menu/ y se pueden ver decenas de ejemplos de menús ya realizados en http://developer.yahoo.com/yui/examples/menu/.

A continuación se indican los pasos necesarios para incluir un menú vertical desplegable:

1) Enlazar los archivos JavaScript y CSS requeridos:

La librería YUI se puede descargar gratuitamente con todos los archivos necesarios, toda la documentación y cientos de ejemplos (por lo que comprimida ya ocupa más de 11 MB). Aunque

es posible utilizar los archivos CSS y JavaScript proporcionados por la librería desde nuestro servidor, Yahoo permite enlazar los archivos CSS y JavaScript desde sus propios servidores.

De esta forma, todos los archivos requeridos por las utilidades de Yahoo se pueden servir directamente desde sus servidores, mucho más rápidos y fiables que cualquier servidor particular. Como el usuario se descarga los archivos desde los servidores de Yahoo, el ahorro de ancho de banda es total y la aplicación carga mucho más rápido.

Para crear un menú vertical, se deben enlazar los siguientes archivos:

```
<!-- Dependencias -->
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.3.0/build/menu/
assets/skins/sam/menu.css">
<script type="text/javascript" src="http://yui.yahooapis.com/2.3.0/build/
yahoo-dom-event/yahoo-dom-event.js"></script>
<script type="text/javascript" src="http://yui.yahooapis.com/2.3.0/build/container/
container_core-min.js"></script>

<!-- Código para menús -->
<script type="text/javascript" src="http://yui.yahooapis.com/2.3.0/build/menu/
menu-min.js"></script>
```

Si se opta por servir los archivos desde nuestro propio servidor, es recomendable enlazar las versiones comprimidas de cada archivo, que se pueden reconocer porque su nombre termina en `-min.js`.

2) Definir el código XHTML de los elementos del menú:

```
<div id="menu_vertical" class="yuimenu">
<div class="bd">
  <ul class="first-of-type">
  <li class="yuimenuitem"><a href="#">Lorem ipsum</a></li>
  <li class="yuimenuitem"><a href="#">Dolor sit amet</a></li>
  <li class="yuimenuitem">Consectetuer
    <div id="consectetuer" class="yuimenu">
    <div class="bd">
      <ul>
        <li class="yuimenuitem"><a href="#">Donec quis nunc</a></li>
        <li class="yuimenuitem"><a href="#">Fusce eleifend</a></li>
        <li class="yuimenuitem"><a href="#">Donec erat</a></li>
        <li class="yuimenuitem"><a href="#">Faucibus orci</a></li>
        <li class="yuimenuitem"><a href="#">Volutpat quam</a></li>
      </ul>
    </div>
    </div>
  </li>
  <li class="yuimenuitem"><a href="#">Adipiscing elit</a></li>
  </ul>
</div>
</div>
```

La librería YUI dispone de muchos tipos diferentes de menús. Además, los menús se pueden construir completamente con JavaScript (en la página XHTML no aparecen los elementos del menú) o a partir del código XHTML de los elementos del menú incluido en la página.

Este último método, definir el menú con XHTML y después aplicarle JavaScript, es la forma recomendada, ya que si el usuario no tiene JavaScript activado, el menú se visualiza correctamente. Por ese motivo, no se va a estudiar el método de construcción de un menú de YUI exclusivamente con JavaScript.

Los elementos que forman los menús creados con YUI se construyen mediante listas no ordenadas con las etiquetas `<ul>` y `<li>`. Un pequeño inconveniente de los menús definidos con la librería YUI es que cada uno de los menús (el principal y todos los desplegables) debe encerrarse con dos elementos de tipo `<div>`. En otras palabras, el esquema de cada menú (y submenú desplegable) es el siguiente:

```
<div id="cualquier_identificador" class="yuimenu">
  <div class="bd">
    <ul class="first-of-type">
      ...
    </ul>
  </div>
</div>
```

Los valores de los atributos `class` de los elementos `<div>` anteriores deben mantenerse para que el menú funcione correctamente.

3) Construir el menú y mostrarlo en la página:

```
<script type="text/javascript">
YAHOO.util.Event.onContentReady("menu_vertical", function () {
  var elMenu = new YAHOO.widget.Menu("menu_vertical", { width: '150px' });
  elMenu.render();
  elMenu.show();
});
</script>
```

El código JavaScript que crea dinámicamente el menú requiere que el árbol DOM de la página se haya construido completamente, por lo que se utiliza la función `onContentReady()` que forma parte de las utilidades para eventos de la librería YUI.

Para construir el menú se crea un nuevo objeto de tipo `YAHOO.widget.Menu` y se le pasan las opciones de inicialización (en este caso, que tenga una anchura de 150 píxel).:

```
var elMenu = new YAHOO.widget.Menu("menu_vertical", { width: '150px' });
```

Una vez creado el objeto del menú, se construye mediante la instrucción `elMenu.render()` y se muestra en la página con la instrucción `elMenu.show()`.

El resultado final es un menú vertical desplegable con el siguiente aspecto:

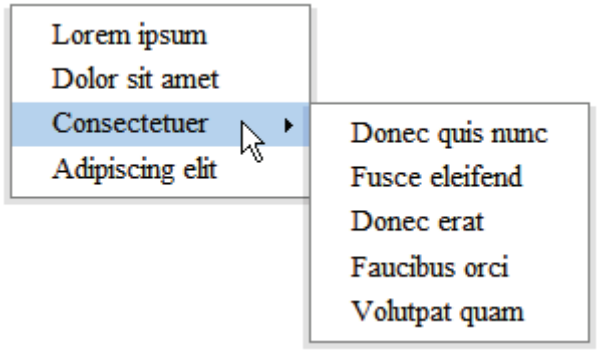

Figura 8.13. Aspecto final del menú vertical desplegable creado con la librería Yahoo YUI

8.5. Galerías de imágenes (Lightbox)

Muchos sitios web utilizan galerías de imágenes para mostrar sus productos y servicios. Este tipo de galerías muestran una serie de miniaturas de imágenes que se amplían al pinchar sobre cada imagen. Hasta hace poco, la técnica más utilizada para construir una galería consistía en incluir las miniaturas en la página y abrir cada imagen grande en una ventana emergente (o *pop-up*) o en una nueva página.

El uso de técnicas basadas en JavaScript ha supuesto una revolución en la creación de las galerías de imágenes y ha permitido mejorar la experiencia de navegación del usuario. La técnica se conoce como *Lightbox* y fue creada originalmente por Lokesh Dhakar. Lightbox es una técnica muy sencilla de utilizar, funciona correctamente en todos los navegadores y permite mantener la semántica del documento (no *ensucia* la página con código JavaScript).

El código de la versión más reciente se puede descargar gratuitamente en http://www.huddletogether.com/projects/lightbox2/, donde también se puede ver una demostración de las galerías de imágenes construidas con Lightbox.

La descarga incluye el código fuente del script, todos los archivos JavaScript externos necesarios, archivos CSS e imágenes de prueba y una breve pero completa documentación.

A continuación se indican los pasos necesarios para incluir Lightbox en una página web:

1) Enlazar los archivos JavaScript y CSS requeridos:

```
<!-- Dependencias -->
<link rel="stylesheet" href="css/lightbox.css" type="text/css" media="screen" />
<script type="text/javascript" src="js/prototype.js"></script>
<script type="text/javascript" src="js/scriptaculous.js?load=effects,builder"></script>

<!-- Código fuente -->
<script type="text/javascript" src="js/lightbox.js"></script>
```

Lightbox utiliza dos de las librerías más conocidas y utilizadas para realizar aplicaciones JavaScript complejas: Prototype y Scriptaculous. Además de estos dos archivos JavaScript, también es necesario enlazar un archivo de tipo CSS que controla el aspecto de Lightbox y enlaza las imágenes utilizadas.

2) Activar Lightbox:

La página XHTML original contiene un enlace simple que abre una nueva página en la que se puede ver ampliada la imagen original:

```
<a href="images/image-1.jpg" title="Título de la imagen 1">Imagen 1</a>
```

Para activar Lightbox, tan solo es necesario incluir el siguiente atributo `rel`:

```
<a href="images/image-1.jpg" rel="lightbox" title="Título de la imagen 1">Imagen 1</a>
```

Ahora, si el usuario tiene activado JavaScript, al pinchar en el enlace:

Figura 8.14. Enlace con el atributo "lightbox" añadido

Se mostrará la imagen ampliada centrada en la misma página:

Figura 8.15. Imagen ampliada que se muestra al pinchar sobre el enlace

Lightbox se activa automáticamente para todos los enlaces cuyo atributo `rel` sea igual a `lightbox`. Además, Lightbox muestra como título de la imagen el valor del atributo `title` del enlace. Si el usuario no tiene activado JavaScript, la imagen se abrirá en una página nueva, como ocurre si no se utiliza Lightbox. La sencillez de uso y el resultado tan espectacular que se consigue son los puntos fuertes de Lightbox.

Normalmente, Lightbox no se utiliza con enlaces de texto sino con imágenes que contienen enlaces:

```
<a href="images/image-1.jpg" rel="lightbox" title="Título de la imagen 1"><img
src="images/imagen_pequena1.jpg" /></a>
```

Ahora, si el usuario tiene activado JavaScript, al pinchar sobre la imagen pequeña:

Figura 8.16. Imagen en miniatura con el enlace preparado para Lightbox

Se mostrará la imagen ampliada centrada en la misma página:

Figura 8.17. Imagen ampliada que se muestra al pinchar sobre la imagen en miniatura

La imagen ampliada se muestra con una vistosa animación y el fondo de la página completa se oscurece para que el usuario centre toda su atención en la imagen. El usuario puede cerrar la imagen ampliada y volver a la página original pulsando sobre la imagen `CLOSE X` de la esquina inferior derecha de la imagen.

Utilizar Lightbox aporta muchas ventajas respecto de la experiencia de usuario del sitio web. Además, el código XHTML permanece intacto y no se ensucia con llamadas a funciones JavaScript, ya que lo único que hay que añadir es un atributo `rel="lightbox"` a las imágenes y enlaces que vayan a utilizar Lightbox.

Además, Lightbox permite relacionar varias imágenes entre sí para crear una galería de imágenes. Así, mientras se visualiza una imagen ampliada es posible pasar a la siguiente imagen

de la galería. Para relacionar varias imágenes entre sí, tan sólo es necesario aplicar un valor único al atributo `rel` como se muestra a continuación:

```
<a href="images/imagen-1.jpg" rel="lightbox[galeria1]">imagen #1</a>
<a href="images/imagen-2.jpg" rel="lightbox[galeria1]">imagen #2</a>
<a href="images/imagen-3.jpg" rel="lightbox[galeria1]">imagen #3</a>
```

Todas las imágenes que tengan el mismo valor del atributo `rel`, automáticamente pasarán a formar parte de una galería de fotos por la que se puede navegar hacia la siguiente o hacia la anterior imagen.

La técnica Lightbox presentada solamente es válida para elementos de tipo imagen. No es posible por tanto mostrar una página web o cierto código XHTML dentro de una ventana de ese tipo.

Afortunadamente, existen otras librerías basadas en Lightbox y que permiten mostrar cualquier tipo de código XHMTL en el interior de la ventana. La más conocida es la técnica creada por ParticleTree y que se puede encontrar en: http://particletree.com/features/lightbox-gone-wild/