Capítulo 6

Formularios y API Forms

6.1 Formularios Web

La Web 2.0 está completamente enfocada en el usuario. Y cuando el usuario es el centro de atención, todo está relacionado con interfaces, en cómo hacerlas más intuitivas, más naturales, más prácticas, y por supuesto más atractivas. Los formularios web son la interface más importante de todas, permiten a los usuarios insertar datos, tomar decisiones, comunicar información y cambiar el comportamiento de una aplicación. Durante los últimos años, códigos personalizados y librerías fueron creados para procesar formularios en el ordenador del usuario. HTML5 vuelve a estas funciones estándar agregando nuevos atributos, elementos y una API completa. Ahora la capacidad de procesamiento de información insertada en formularios en tiempo real ha sido incorporada en los navegadores y completamente estandarizada.

El elemento <form>

Los formularios en HTML no han cambiado mucho. La estructura sigue siendo la misma, pero HTML5 ha agregado nuevos elementos, tipos de campo y atributos para expandirlos tanto como sea necesario y proveer así las funciones actualmente implementadas en aplicaciones web.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="miformulario" id="miformulario" method="get">
      <input type="text" name="nombre" id="nombre">
      <input type="submit" value="Enviar">
    </form>
  </section>
</body>
</html>
```

Listado 6-1. *Estructura normal de un formulario.*

En el Listado 6-1 creamos una plantilla básica para formularios. Como puede ver, la estructura del formulario y sus atributos siguen siendo igual que en especificaciones previas. Sin embargo, existen nuevos atributos para el elemento `<form>`:

autocomplete Este es un viejo atributo que se ha vuelto estándar en esta especificación. Puede tomar dos valores: `on` y `off`. El valor por defecto es `on`. Cuando es configurado como `off` los elementos `<input>` pertenecientes a ese formulario tendrán la función de autocompletar desactivada, sin mostrar entradas previas como posibles valores. Puede ser implementado en el elemento `<form>` o en cualquier elemento `<input>` independientemente.

novalidate Una de las características de formularios en HTML5 es la capacidad propia de validación. Los formularios son automáticamente validados. Para evitar este comportamiento, podemos usar el atributo `novalidate`. Para lograr lo mismo para elementos `<input>` específicos, existe otro atributo llamado `formnovalidate`. Ambos atributos son booleanos, ningún valor tiene que ser especificado (su presencia es suficiente para activar su función).

El elemento <input>

El elemento más importante en un formulario es `<input>`. Este elemento puede cambiar sus características gracias al atributo `type` (tipo). Este atributo determina qué clase de entrada es esperada desde el usuario. Los tipos disponibles hasta el momento eran el multipropósitos `text` (para textos en general) y solo unos pocos más específicos como `password` o `submit`. HTML5 ha expandido las opciones incrementando de este modo las posibilidades para este elemento.

En HTML5 estos nuevos tipos no solo están especificando qué clase de entrada es esperada sino también diciéndole al navegador qué debe hacer con la información recibida. El navegador procesará los datos ingresados de acuerdo al valor del atributo `type` y validará la entrada o no.

El atributo `type` trabaja junto con otros atributos adicionales para ayudar al navegador a limitar y controlar en tiempo real lo ingresado por el usuario.

> **Hágalo usted mismo:** Cree un nuevo archivo HTML con la plantilla del Listado 6-1. Para comprobar cómo funciona cada tipo de campo estudiado de aquí en adelante, reemplace los elementos `<input>` en la plantilla por aquellos que quiere probar y abra nuevamente el archivo en su navegador. En este momento la forma en la que los tipos de campo son tratados varía, por este motivo le recomendamos probar el código en cada navegador disponible.

Tipo email

Casi todo formulario en la web ofrece un campo para ingresar una dirección de email, pero hasta ahora el único tipo de campo disponible para esta clase de datos era `text`. El

tipo **text** representa un texto general, no un dato específico, por lo que teníamos que controlar la entrada con código Javascript para estar seguros de que el texto ingresado correspondía a un email válido. Ahora el navegador se hace cargo de esto con el nuevo tipo **email**:

```
<input type="email" name="miemail" id="miemail">
```

Listado 6-2. *El tipo* `email`.

El texto insertado en el campo generado por el código del Listado 6-2 será controlado por el navegador y validado como un email. Si la validación falla, el formulario no será enviado.

Cómo cada navegador responderá a una entrada inválida no está determinado en la especificación de HTML5. Por ejemplo, algunos navegadores mostrarán un borde rojo alrededor del elemento **<input>** que produjo el error y otros lo mostrarán en azul. Existen formas de personalizar esta respuesta, pero las veremos más adelante.

Tipo search

El tipo **search** (búsqueda) no controla la entrada, es solo una indicación para los navegadores. Al detectar este tipo de campo algunos navegadores cambiarán el diseño del elemento para ofrecer al usuario un indicio de su propósito.

```
<input type="search" name="busqueda" id="busqueda">
```

Listado 6-3. *El tipo* `search`.

Tipo url

Este tipo de campo trabaja exactamente igual que el tipo **email** pero es específico para direcciones web. Está destinado a recibir solo URLs absolutas y retornará un error si el valor es inválido.

```
<input type="url" name="miurl" id="miurl">
```

Listado 6-4. *El tipo* `url`.

Tipo tel

Este tipo de campo es para números telefónicos. A diferencia de los tipos **email** y **url**, el tipo **tel** no requiere ninguna sintaxis en particular. Es solo una indicación para el

navegador en caso de que necesite hacer ajustes de acuerdo al dispositivo en el que la aplicación es ejecutada.

```
<input type="tel" name="telefono" id="telefono">
```

Listado 6-5. *El tipo* `tel`.

Tipo number

Como su nombre lo indica, el tipo **`number`** es sólo válido cuando recibe una entrada numérica. Existen algunos atributos nuevos que pueden ser útiles para este campo:

min El valor de este atributo determina el mínimo valor aceptado para el campo.
max El valor de este atributo determina el máximo valor aceptado para el campo.
step El valor de este atributo determina el tamaño en el que el valor será incrementado o disminuido en cada paso. Por ejemplo, si declara un valor de 5 para **`step`** en un campo que tiene un valor mínimo de 0 y máximo de 10, el navegador no le permitirá especificar valores entre 0 y 5 o entre 5 y 10.

```
<input type="number" name="numero" id="numero" min=”0” max=”10”
                                                step=”5”>
```

Listado 6-6. *El tipo* `number`.

No es necesario especificar ambos atributos (**`min`** y **`max`**), y el valor por defecto para **`step`** es 1.

Tipo range

Este tipo de campo hace que el navegador construya una nueva clase de control que no existía previamente. Este nuevo control le permite al usuario seleccionar un valor a partir de una serie de valores o rango. Normalmente es mostrado en pantalla como una puntero deslizable o un campo con flechas para seleccionar un valor entre los predeterminados, pero no existe un diseño estándar hasta el momento.

El tipo **`range`** usa los atributos **`min`** y **`max`** estudiados previamente para configurar los límites del rango. También puede utilizar el atributo **`step`** para establecer el tamaño en el cual el valor del campo será incrementado o disminuido en cada paso.

```
<input type="range" name="numero" id="numero" min=”0” max=”10”
                                                step=”5”>
```

Listado 6-7. *El tipo* `range`.

Podemos declarar el valor inicial utilizando el viejo atributo **value** y usar Javascript para mostrar el número seleccionado en pantalla como referencia. Experimentaremos con esto y el nuevo elemento **<output>** más adelante.

Tipo date

Este es otro tipo de campo que genera una nueva clase de control. En este caso fue incluido para ofrecer una mejor forma de ingresar una fecha. Algunos navegadores muestran en pantalla un calendario que aparece cada vez que el usuario hace clic sobre el campo. El calendario le permite al usuario seleccionar un día que será ingresado en el campo junto con el resto de la fecha. Un ejemplo de uso es cuando necesitamos proporcionar un método para seleccionar una fecha para un vuelo o la entrada a un espectáculo. Gracias al tipo **date** ahora es el navegador el que se encarga de construir un almanaque o las herramientas necesarias para facilitar el ingreso de este tipo de datos.

```
<input type="date" name="fecha" id="fecha">
```

Listado 6-8. *El tipo* `date`.

La interface no fue declarada en la especificación. Cada navegador provee su propia interface y a veces adaptan el diseño al dispositivo en el cual la aplicación está siendo ejecutada. Normalmente el valor generado y esperado tiene la sintaxis **año-mes-día**.

Tipo week

Este tipo de campo ofrece una interface similar a **date**, pero solo para seleccionar una semana completa. Normalmente el valor esperado tiene la sintaxis **2011-W50** donde **2011** es al año y **50** es el número de la semana.

```
<input type="week" name="semana" id="semana">
```

Listado 6-9. *El tipo* `week`.

Tipo month

Similar al tipo de campo previo, éste es específico para seleccionar meses. Normalmente el valor esperado tiene la sintaxis **año-mes**.

```
<input type="month" name="mes" id="mes">
```

Listado 6-10. *El tipo* `month`.

Tipo time

El tipo de campo **time** es similar a **date**, pero solo para la hora. Toma el formato de horas y minutos, pero su comportamiento depende de cada navegador en este momento. Normalmente el valor esperado tiene la sintaxis **hora:minutos:segundos**, pero también puede ser solo **hora:minutos**.

```
<input type="time" name="hora" id="hora">
```

Listado 6-11. *El tipo* `time`.

Tipo datetime

El tipo de campo **datetime** es para ingresar fecha y hora completa, incluyendo la zona horaria.

```
<input type="datetime" name="fechahora" id="fechahora">
```

Listado 6-12. *El tipo* `datetime`.

Tipo datetime-local

El tipo de campo **datetime-local** es como el tipo **datetime** sin la zona horaria.

```
<input type="datetime-local" name="tiempolocal" id="tiempolocal">
```

Listado 6-13. *El tipo* `datetime-local`.

Tipo color

Además de los tipos de campo para fecha y hora existe otro tipo que provee una interface predefinida similar para seleccionar colores. Normalmente el valor esperado para este campo es un número hexadecimal, como #00FF00.

```
<input type="color" name="micolor" id="micolor">
```

Listado 6-14. *El tipo* `color`.

Ninguna interface fue especificada como estándar en HTML5 para el tipo de campo **color**, pero es posible que una grilla con un conjunto básico de colores sea adoptada e incorporada en los navegadores.

6.2 Nuevos atributos

Algunos tipos de campo requieren de la ayuda de atributos, como los anteriormente estudiados **min**, **max** y **step**. Otros tipos de campo requieren la asistencia de atributos para mejorar su rendimiento o determinar su importancia en el proceso de validación. Ya vimos algunos de ellos, como **novalidate** para evitar que el formulario completo sea validado o **formnovalidate** para hacer lo mismo con elementos individuales. El atributo **autocomplete**, también estudiado anteriormente, provee medidas de seguridad adicionales para el formulario completo o elementos individuales. Aunque útiles, estos atributos no son los únicos incorporados por HTML5. Es momento de estudiar el resto.

Atributo placeholder

Especialmente en tipos de campo **search**, pero también en entradas de texto normales, el atributo **placeholder** representa una sugerencia corta, una palabra o frase provista para ayudar al usuario a ingresar la información correcta. El valor de este atributo es presentado en pantalla por los navegadores dentro del campo, como una marca de agua que desaparece cuando el elemento es enfocado.

```
<input type="search" name="busqueda" id="busqueda"
placeholder="escriba su búsqueda">
```

Listado 6-15. *El atributo* `placeholder`.

Atributo required

Este atributo booleano no dejará que el formulario sea enviado si el campo se encuentra vacío. Por ejemplo, cuando usamos el tipo **email** para recibir una dirección de email, el navegador comprueba si la entrada es un email válido o no, pero validará la entrada si el campo está vacío. Cuando el atributo **required** es incluido, la entrada será válida sólo si se cumplen las dos condiciones: que el campo no esté vacío y que el valor ingresado esté de acuerdo con los requisitos del tipo de campo.

```
<input type="email" name="miemail" id="miemail" required>
```

Listado 6-16. *El campo* `email` *ahora es un campo requerido.*

Atributo multiple

El atributo **multiple** es otro atributo booleano que puede ser usado en algunos tipos de campo (por ejemplo, **email** o **file**) para permitir el ingreso de entradas múltiples en el mismo campo.

Los valores insertados deben estar separados por coma para ser válidos.

```
<input type="email" name="miemail" id="miemail" multiple>
```

Listado 6-17. *El campo* `email` *acepta múltiples valores separados por coma.*

El código en el Listado 6-17 permite la inserción de múltiples valores separados por coma, y cada uno de ellos será validado por el navegador como una dirección de email.

Atributo autofocus

Esta es una función que muchos desarrolladores aplicaban anteriormente utilizando el método **focus()** de Javascript. Este método era efectivo pero forzaba el foco sobre el elemento seleccionado, incluso cuando el usuario ya se encontraba posicionado en otro diferente. Este comportamiento era irritante pero difícil de evitar hasta ahora.

El atributo **autofocus** enfocará la página web sobre el elemento seleccionado pero considerando la situación actual. No moverá el foco cuando ya haya sido establecido por el usuario sobre otro elemento.

```
<input type="search" name="busqueda" id="busqueda" autofocus>
```

Listado 6-18. *El atributo* `autofocus` *aplicado sobre un campo de búsqueda.*

Atributo pattern

El atributo **pattern** es para propósitos de validación. Usa expresiones regulares para personalizar reglas de validación. Algunos de los tipos de campo ya estudiados validan cadenas de texto específicas, pero no permiten hacer validaciones personalizadas, como por ejemplo un código postal que consiste en 5 números. No existe ningún tipo de campo predeterminado para esta clase de entrada.

El atributo **pattern** nos permite crear nuestro propio tipo de campo para controlar esta clase de valores no ordinarios. Puede incluso incluir un atributo **title** para personalizar mensajes de error.

```
<input pattern="[0-9]{5}" name="codigopostal" id="codigopostal"
              title="inserte los 5 números de su código postal">
```

Listado 6-19. *Tipos personalizados usando el atributo* `pattern`.

IMPORTANTE: Expresiones regulares son un tema complejo y no relacionado directamente con HTML5. Para obtener información adicional al respecto, visite nuestro sitio web y siga los enlaces correspondientes a este capítulo.

Atributo form

El atributo **`form`** es una adición útil que nos permite declarar elementos para un formulario fuera del ámbito de las etiquetas **`<form>`**. Hasta ahora, para construir un formulario teníamos que escribir las etiquetas **`<form>`** de apertura y cierre y luego declarar cada elemento del formulario entre ellas. En HTML5 podemos insertar los elementos en cualquier parte del código y luego hacer referencia al formulario que pertenecen usando su nombre y el atributo **`form`**:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
</head>
<body>
  <nav>
    <input type="search" name="busqueda" id="busqueda"
                                              form="formulario">
  </nav>
  <section>
    <form name="formulario" id="formulario" method="get">
      <input type="text" name="nombre" id="nombre">
      <input type="submit" value="Enviar">
    </form>
  </section>
</body>
</html>
```

Listado 6-20. *Declarando elementos del formulario en cualquier parte.*

6.3 Nuevos elementos para formularios

Ya hemos visto los nuevos tipos de campos disponibles en HTML5, por lo tanto es momento de estudiar los nuevos elementos HTML incorporados con la intención de mejorar o expandir las posibilidades de los formularios.

El elemento <datalist>

El elemento **<datalist>** es un elemento específico de formularios usado para construir una lista de ítems que luego, con la ayuda del atributo **list**, será usada como sugerencia en un campo del formulario.

```
<datalist id="informacion">
  <option value=”123123123” label=”Teléfono 1”>
  <option value=”456456456” label=”Teléfono 2”>
</datalist>
```

Listado 6-21. *Construyendo la lista.*

Este elemento utiliza el elemento **<option>** en su interior para crear la lista de datos a sugerir. Con la lista ya declarada, lo único que resta es referenciarla desde un elemento **<input>** usando el atributo **list**:

```
<input type="tel" name="telefono" id="telefono" list="informacion">
```

Listado 6-22. *Ofreciendo una lista de sugerencias con el atributo* `list`.

El elemento en el Listado 6-22 mostrará posibles valores para que el usuario elija.

> **IMPORTANTE:** El elemento **<datalist>** fue solo implementado en Opera y Firefox Beta en este momento.

El elemento <progress>

Este elemento no es específico de formularios, pero debido a que representa el progreso en la realización de una tarea, y usualmente estas tareas son comenzadas y procesadas a través de formularios, puede ser incluido dentro del grupo de elementos para formularios.

El elemento **<progress>** utiliza dos atributos para configurar su estado y límites. El atributo **value** indica qué parte de la tarea ya ha sido procesada, y **max** declara el valor a alcanzar para que la tarea se considere finalizada. Vamos a usar **<progress>** en futuros ejemplos.

El elemento <meter>

Similar a **<progress>**, el elemento **<meter>** es usado para mostrar una escala, pero no de progreso. Este elemento tiene la intención de representar una medida, como el tráfico del sitio web, por ejemplo.

El elemento **<meter>** cuenta con varios atributos asociados: **min** y **max** configuran los límites de la escala, **value** determina el valor medido, y **low**, **high** y **optimum** son usados para segmentar la escala en secciones diferenciadas y marcar la posición que es óptima.

El elemento <output>

Este elemento representa el resultado de un cálculo. Normalmente ayudará a mostrar los resultados del procesamiento de valores provistos por un formulario. El atributo **for** asocia el elemento **<output>** con el elemento fuente que participa del cálculo, pero este elemento deberá ser referenciado y modificado desde código Javascript. Su sintaxis es **<output>valor</output>**.

6.4 API Forms

Seguramente no le sorprenderá saber que, al igual que cada uno de los aspectos de HTML5, los formularios HTML cuentan con su propia API para personalizar todos los aspectos de procesamiento y validación.

Existen diferentes formas de aprovechar el proceso de validación en HTML5. Podemos usar los tipos de campo para activar el proceso de validación por defecto (por ejemplo, **email**) o volver un tipo de campo regular como **text** (o cualquier otro) en un campo requerido usando el atributo **required**. También podemos crear tipos de campo especiales usando **pattern** para personalizar requisitos de validación. Sin embargo, cuando se trata de aplicar mecanismos complejos de validación (por ejemplo, combinando campos o comprobando los resultados de un cálculo) deberemos recurrir a nuevos recursos provistos por esta API.

setCustomValidity()

Los navegadores que soportan HTML5 muestran un mensaje de error cuando el usuario intenta enviar un formulario que contiene un campo inválido.

Podemos crear mensajes para nuestros propios requisitos de validación usando el método **setCustomValidity(mensaje)**.

Con este método especificamos un error personalizado que mostrará un mensaje cuando el formulario es enviado. Cuando un mensaje vacío es declarado, el error es anulado.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
  <script>
```

```
      function iniciar(){
        nombre1=document.getElementById("nombre");
        nombre2=document.getElementById("apellido");
        nombre1.addEventListener("input", validacion, false);
        nombre2.addEventListener("input", validacion, false);
        validacion();
      }
      function validacion(){
        if(nombre1.value=='' && nombre2.value==''){
          nombre1.setCustomValidity('inserte al menos un nombre');
          nombre1.style.background='#FFDDDD';
        }else{
          nombre1.setCustomValidity('');
          nombre1.style.background='#FFFFFF';
        }
      }
      window.addEventListener("load", iniciar, false);
    </script>
  </head>
  <body>
    <section>
      <form name="registracion" method="get">
        Nombre:
        <input type="text" name="nombre" id="nombre">
        Apellido:
        <input type="text" name="apellido" id="apellido">
        <input type="submit" id="send" value="ingresar">
      </form>
    </section>
  </body>
  </html>
```

Listado 6-23. *Declarando errores personalizados.*

El código del Listado 6-23 presenta una situación de validación compleja. Dos campos fueron creados para recibir el nombre y apellido del usuario. Sin embargo, el formulario solo será inválido cuando ambos campos se encuentran vacíos. El usuario necesita ingresar solo uno de los campos, su nombre o su apellido, para validar la entrada.

En casos como éste no es posible usar el atributo **`required`** debido a que no sabemos cuál campo el usuario decidirá utilizar. Solo con código Javascript y errores personalizados podremos crear un efectivo mecanismo de validación para este escenario.

Nuestro código comienza a funcionar cuando el evento **`load`** es disparado. La función **`iniciar()`** es llamada para responder al evento. Esta función crea referencias para los dos elementos **`<input>`** y agrega una escucha para el evento **`input`** en ambos. Estas escuchas llamarán a la función **`validacion()`** cada vez que el usuario escribe dentro de los campos.

Debido a que los elementos **`<input>`** se encuentran vacíos cuando el documento es cargado, debemos declarar una condición inválida para no permitir que el usuario envíe el formulario antes de ingresar al menos uno de los valores. Por esta razón la función **`validacion()`** es llamada al comienzo. Si ambos campos están vacíos el error es

generado y el color de fondo del campo **nombre** es cambiado a rojo. Sin embargo, si esta condición ya no es verdad porque al menos uno de los campos fue completado, el error es anulado y el color del fondo de **nombre** es nuevamente establecido como blanco.

Es importante tener presente que el único cambio producido durante el procesamiento es la modificación del color de fondo del campo. El mensaje declarado para el error con **setCustomValidity()** será visible sólo cuando el usuario intente enviar el formulario.

> **Hágalo usted mismo:** Para propósitos de prueba, incluimos el código Javascript dentro del documento. Como resultado, lo único que debe hacer para ejecutar este ejemplo es copiar el código del Listado 6-23 dentro de un archivo HTML vacío y abrir el archivo en su navegador.
>
> **IMPORTANTE:** API Forms está siendo desarrollada en este momento. Dependiendo del nivel al que la tecnología haya sido adoptada en el momento en el que usted lee estas líneas es probable que necesite probar los códigos de este capítulo en diferentes navegadores para comprobar su correcto funcionamiento.

El evento invalid

Cada vez que el usuario envía el formulario, un evento es disparado si un campo inválido es detectado. El evento es llamado **invalid** y es disparado por el elemento que produce el error. Podemos agregar una escucha para este evento y así ofrecer una respuesta personalizada, como en el siguiente ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
  <script>
    function iniciar(){
      edad=document.getElementById("miedad");
      edad.addEventListener("change", cambiarrango, false);
      document.informacion.addEventListener("invalid",
                                            validacion, true);
      document.getElementById("enviar").addEventListener("click",
                                               enviar, false);
    }
    function cambiarrango(){
      var salida=document.getElementById("rango");
      var calc=edad.value-20;
      if(calc<20){
        calc=0;
        edad.value=20;
      }
```

```
        salida.innerHTML=calc+' a '+edad.value;
      }
      function validacion(e){
        var elemento=e.target;
        elemento.style.background='#FFDDDD';
      }
      function enviar(){
        var valido=document.informacion.checkValidity();
        if(valido){
          document.informacion.submit();
        }
      }
      window.addEventListener("load", iniciar, false);
    </script>
  </head>
  <body>
    <section>
      <form name="informacion" method="get">
        Usuario:
        <input pattern="[A-Za-z]{3,}" name="usuario" id="usuario"
                                          maxlength="10" required>
        Email:
        <input type="email" name="miemail" id="miemail" required>
        Rango de Edad:
        <input type="range" name="miedad" id="miedad" min="0"
                                    max="80" step="20" value="20">
        <output id="rango">0 a 20</output>
        <input type="button" id="enviar" value="ingresar">
      </form>
    </section>
  </body>
  </html>
```

Listado 6-24. *Nuestro propio sistema de validación.*

En el Listado 6-24, creamos un nuevo formulario con tres campos para ingresar el nombre de usuario, un email y un rango de 20 años de edad.

El campo **`usuario`** tiene tres atributos para validación: el atributo **`pattern`** solo admite el ingreso de un texto de tres caracteres mínimo, desde la A a la Z (mayúsculas o minúsculas), el atributo **`maxlength`** limita la entrada a 10 caracteres máximo, y el atributo **`required`** invalida el campo si está vacío. El campo **`miemail`** cuenta con sus limitaciones naturales debido a su tipo y además no podrá enviarse vacío gracias al atributo **`required`**. El campo **`miedad`** usa los atributos **`min`**, **`max`**, **`step`** y **`value`** para configurar las condiciones del rango.

También declaramos un elemento **`<output>`** para mostrar en pantalla una referencia del rango seleccionado.

Lo que el código Javascript hace con este formulario es simple: cuando el usuario hace clic en el botón "ingresar", un evento **`invalid`** será disparado desde cada campo inválido y el color de fondo de esos campos será cambiado a rojo por la función **`validacion()`**.

Veamos este procedimiento con un poco más de detalle. El código comienza a funcionar cuando el típico evento **load** es disparado luego que el documento fue completamente cargado. La función **iniciar()** es ejecutada y tres escuchas son agregadas para los eventos **change**, **invalid** y **click**.

Cada vez que el contenido de los elementos del formulario cambia por alguna razón, el evento **change** es disparado desde ese elemento en particular. Lo que hicimos fue escuchar a este evento desde el campo **range** y llamar a la función **cambiarrango()** cada vez que el evento ocurre. Por este motivo, cuando el usuario desplaza el control del rango o cambia los valores dentro de este campo para seleccionar un rango de edad diferente, los nuevos valores son calculados por la función **cambiarrango()**. Los valores admitidos para este campo son períodos de 20 años (por ejemplo, 0 a 20 o 20 a 40). Sin embargo, el campo solo retorna un valor, como 20, 40, 60 u 80. Para calcular el valor de comienzo del rango, restamos 20 al valor actual del campo con la fórmula **edad.value - 20**, y grabamos el resultado en la variable **calc**. El período mínimo admitido es 0 a 20, por lo tanto con un condicional **if** controlamos esta condición y no permitimos un período menor (estudie la función **cambiarrango()** para entender cómo funciona).

La segunda escucha agregada en la función **iniciar()** es para el evento **invalid**. La función **validacion()** es llamada cuando este evento es disparado para cambiar el color de fondo de los campos inválidos. Recuerde que este evento será disparado desde un campo inválido cuando el botón "ingresar" sea presionado. El evento no contiene una referencia del formulario o del botón "ingresar", sino del campo que generó el error. En la función **validacion()** esta referencia es capturada y grabada en la variable **elemento** usando la variable **e** y la propiedad **target**. La construcción **e.target** retorna una referencia al elemento **<input>** inválido. Usando esta referencia, en la siguiente línea cambiamos el color de fondo del elemento.

Volviendo a la función **iniciar()**, encontraremos una escucha más que necesitamos analizar. Para tener control absoluto sobre el envío del formulario y el momento de validación, creamos un botón regular en lugar del típico botón **submit**. Cuando este botón es presionado, el formulario es enviado, pero solo si todos sus elementos son válidos. La escucha agregada para el evento **click** en la función **iniciar()** ejecutará la función **enviar()** cuando el usuario haga clic sobre el botón. Usando el método **checkValidity()** solicitamos al navegador que realice el proceso de validación y solo enviamos el formulario usando el tradicional método **submit()** cuando ya no hay más condiciones inválidas.

Lo que hicimos con el código Javascript fue tomar control sobre todo el proceso de validación, personalizando cada aspecto y modificando el comportamiento del navegador.

Validación en tiempo real

Cuando abrimos el archivo con la plantilla del Listado 6-24 en el navegador, podremos notar que no existe una validación en tiempo real. Los campos son sólo validados cuando el botón "ingresar" es presionado. Para hacer más práctico nuestro sistema personalizado de validación, tenemos que aprovechar los atributos provistos por el objeto **ValidityState**.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formularios</title>
  <script>
    function iniciar(){
      edad=document.getElementById("miedad");
      edad.addEventListener("change", cambiarrango, false);
      document.informacion.addEventListener("invalid",
       validacion, true);
      document.getElementById("enviar").addEventListener("click",
                                                  enviar, false);
      document.informacion.addEventListener("input", controlar,
                                                          false);
    }
    function cambiarrango(){
      var salida=document.getElementById("rango");
      var calc=edad.value-20;
      if(calc<20){
        calc=0;
        edad.value=20;
      }
      salida.innerHTML=calc+' a '+edad.value;
    }
    function validacion(e){
      var elemento=e.target;
      elemento.style.background='#FFDDDD';
    }
    function enviar(){
      var valido=document.informacion.checkValidity();
      if(valido){
        document.informacion.submit();
      }
    }
    function controlar(e){
      var elemento=e.target;
      if(elemento.validity.valid){
        elemento.style.background='#FFFFFF';
      }else{
        elemento.style.background='#FFDDDD';
      }
    }
    window.addEventListener("load", iniciar, false);
  </script>
</head>
<body>
  <section>
    <form name="informacion" method="get">
      Usuario:
      <input pattern="[A-Za-z]{3,}" name="usuario" id="usuario"
                                        maxlength="10" required>
```

```
        Email:
        <input type="email" name="miemail" id="miemail" required>
        Rango de Edad:
        <input type="range" name="miedad" id="miedad" min="0"
                                  max="80" step="20" value="20">
        <output id="rango">0 a 20</output>
        <input type="button" id="enviar" value="ingresar">
      </form>
    </section>
  </body>
  </html>
```

Listado 6-25. *Validando en tiempo real.*

En el Listado 6-25, una nueva escucha fue agregada para el evento **`input`** sobre el formulario. Cada vez que el usuario modifica un campo, escribiendo o cambiando su contenido, la función **`controlar()`** es ejecutada para responder a este evento.

La función **`controlar()`** también aprovecha la propiedad **`target`** para crear una referencia hacia el elemento que disparó el evento **`input`**. La validez del campo es controlada por medio del estado **`valid`** provisto por el atributo **`validity`** en la construcción **`elemento.validity.valid`**.

El estado **`valid`** será **`true`** (verdadero) si el elemento es válido y **`false`** (falso) si no lo es. Usando esta información cambiamos el color del fondo del elemento. Este color será, por lo tanto, blanco para un campo válido y rojo para uno inválido.

Con esta simple incorporación logramos que cada vez que el usuario modifica el valor de un campo del formulario, este campo será controlado y validado, y su condición será mostrada en pantalla en tiempo real.

Propiedades de validación

En el ejemplo del Listado 6-25 controlamos el estado **`valid`**. Este estado particular es un atributo del objeto **`ValidityState`** que retornará el estado de un elemento considerando cada uno de los posibles estados de validación. Si cada condición es válida entonces el valor del atributo **`valid`** será **`true`** (verdadero).

Existen ocho posibles estados de validez para las diferentes condiciones:

valueMissing Este estado es **`true`** (verdadero) cuando el atributo **`required`** fue declarado y el campo está vacío.

typeMismatch Este estado es **`true`** (verdadero) cuando la sintaxis de la entrada no corresponde con el tipo especificado (por ejemplo, si el texto insertado en un tipo de campo **`email`** no es una dirección de email válida).

patternMismatch Este estado es **`true`** (verdadero) cuando la entrada no corresponde con el patrón provisto por el atributo **`pattern`**.

tooLong Este estado es **`true`** (verdadero) cuando el atributo **`maxlength`** fue declarado y la entrada es más extensa que el valor especificado para este atributo.

rangeUnderflow Este estado es **`true`** (verdadero) cuando el atributo **`min`** fue declarado y la entrada es menor que el valor especificado para este atributo.

rangeOverflow Esta estado es **`true`** (verdadero) cuando el atributo **`max`** fue declarado y la entrada es más grande que el valor especificado para este atributo.

stepMismatch Este estado es **`true`** (verdadero) cuando el atributo **`step`** fue declarado y su valor no corresponde con los valores de atributos como **`min`**, **`max`** y **`value`**.

customError Este estado es **`true`** (verdadero) cuando declaramos un error personalizado usando el método **`setCustomValidity()`** estudiado anteriormente.

Para controlar estos estados de validación, debemos utilizar la sintaxis **`elemento.validity.estado`** (donde **`estado`** es cualquiera de los valores listados arriba). Podemos aprovechar estos atributos para saber exactamente que originó el error en un formulario, como en el siguiente ejemplo:

```
function enviar(){
  var elemento=document.getElementById("usuario");
  var valido=document.informacion.checkValidity();

  if(valido){
    document.informacion.submit();
  }else if(elemento.validity.patternMismatch ||
                                  elemento.validity.valueMissing){
    alert('el nombre de usuario debe tener mínimo de 3 caracteres');
  }
}
```

Listado 6-26. *Usando estados de validación para mostrar un mensaje de error personalizado.*

En el Listado 6-26, la función **`enviar()`** fue modificada para incorporar esta clase de control. El formulario es validado por el método **`checkValidity()`** y si es válido es enviado con **`submit()`**. En caso contrario, los estados de validación **`patternMismatch`** y **`valueMissing`** para el campo **`usuario`** son controlados y un mensaje de error es mostrado cuando el valor de alguno de ellos es **`true`** (verdadero).

> **Hágalo usted mismo:** Reemplace la función **`enviar()`** en la plantilla del Listado 6-25 con la nueva función del Listado 6-26 y abra el archivo HTML en su navegador.

willValidate

En aplicaciones dinámicas es posible que los elementos involucrados no tengan que ser validados. Este puede ser el caso, por ejemplo, con botones, campos ocultos o elementos como **`<output>`**. La API nos ofrece la posibilidad de detectar esta condición usando el atributo **`willValidate`** y la sintaxis **`elemento.willValidate`**.

6.5 Referencia rápida

Los formularios constituyen el principal medio de comunicación entre usuarios y aplicaciones web. HTML5 incorpora nuevos tipos para el elemento `<input>`, una API completa para validar y procesar formularios, y atributos para mejorar esta interface.

Tipos

Algunos de los nuevos tipos de campo introducidos por HTML5 tienen condiciones de validación implícitas. Otros solo declaran un propósito para el campo que ayudará a los navegadores a presentar el formulario en pantalla.

email Este tipo de campo valida la entrada como una dirección de email.
search Este tipo de campo da información al navegador sobre el propósito del campo (búsqueda) para ayudar a presentar el formulario en pantalla.
url Este tipo de campo valida la entrada como una dirección web.
tel Este tipo de campo da información al navegador sobre el propósito del campo (número telefónico) para ayudar a presentar el formulario en pantalla.
number Este tipo de campo valida la entrada como un número. Puede ser combinado con otros atributos (como `min`, `max` y `step`) para limitar los números permitidos.
range Este tipo de campo genera un nuevo control en pantalla para la selección de números. La entrada es limitada por los atributos `min`, `max` y `step`. El atributo `value` establece el valor inicial para el elemento.
date Este tipo de campo valida la entrada como una fecha en el formato `año-mes-día`.
month Este tipo de campo valida la entrada como una fecha en el formato `año-mes`.
week Este tipo de campo valida la entrada como una fecha en el formato `año-semana` donde el segundo valor es representado por una letra W y el número de la semana.
time Este tipo de campo valida la entrada como una hora en el formato `hora:minutos:segundos`. También puede tomar otras sintaxis como `hora:minutos`.
datetime Este tipo de campo valida la entrada como fecha y hora completa, incluyendo zona horaria.
datetime-local Este tipo de campo valida la entrada como una fecha y hora completa, sin zona horaria.
color Este tipo de campo valida la entrada como un valor de color.

Atributos

Nuevos atributos fueron también agregados en HTML5 para mejorar la capacidad de los formularios y ayudar al control de validación.

autocomplete Este atributo especifica si los valores insertados serán almacenados para futura referencia. Puede tomar dos valores: `on` y `off`.

autofocus Este es un atributo booleano que enfoca el elemento en el que se encuentra cuando la página es cargada.
novalidate Este atributo es exclusivo para elementos `<form>`. Es un atributo booleano que establece si el formulario será validado por el navegador o no.
formnovalidate Este atributo es exclusivo para elementos de formulario individuales. Es un atributo booleano que establece si el elemento será validado por el navegador o no.
placeholder Este atributo ofrece información que orientará al usuario sobre la entrada esperada. Su valor puede ser una palabra simple o un texto corto, y será mostrado como una marca de agua dentro del campo hasta que el elemento es enfocado.
required Este atributo declara al elemento como requerido para validación. Es un atributo booleano que no dejará que el formulario sea enviado hasta que una entrada para el campo sea provista.
pattern Este atributo especifica una expresión regular contra la cual la entrada será validada.
multiple Este es un atributo booleano que permite ingresar múltiples valores en el mismo campo (como múltiples cuentas de email, por ejemplo). Los valores deben ser separados por coma.
form Este atributo asocia el elemento al formulario. El valor provisto debe ser el valor del atributo `id` del elemento `<form>`.
list Este atributo asocia el elemento con un elemento `<datalist>` para mostrar una lista de posibles valores para el campo. El valor provisto debe ser el valor del atributo `id` del elemento `<datalist>`.

Elementos

HTML5 también incluye nuevos elementos que ayudan a mejorar y expandir formularios.

<datalist> Este elemento hace posible incluir una lista de opciones predefinidas que será mostrada en un elemento `<input>` como valores sugeridos. La lista es construida con el elemento `<option>` y cada opción es declarada con los atributos `value` y `label`. Esta lista de opciones se relaciona con un elemento `<input>` por medio del atributo `list`.
<progress> Este elemento representa el estado en la evolución de una tarea (por ejemplo, una descarga).
<meter> Este elemento representa una medida, como el tráfico de un sitio web.
<output> Este elemento presenta un valor de salida para aplicaciones dinámicas.

Métodos

HTML5 incluye una API específica para formularios que provee métodos, eventos y propiedades. Algunos de los métodos son:

setCustomValidity(mensaje) Este método nos permite declarar un error y proveer un mensaje de error para un proceso de validación personalizado. Para anular el error, debemos llamar al método con una cadena de texto vacía como atributo.
checkValidity() Este método solicita al navegador iniciar el proceso de validación. Activa el proceso de validación provisto por el navegador sin la necesidad de enviar el formulario. Este método retorna `true` (verdadero) si el elemento es válido.

Eventos

Los eventos incorporados para esta API son los siguientes:

invalid Este evento es disparado cuando un elemento inválido es detectado durante el proceso de validación.
forminput Este evento es disparado cuando un formulario recibe la entrada del usuario.
formchange Este evento es disparado cuando un cambio ocurre en el formulario.

Estado

API Forms provee un grupo de atributos para controlar estados en un proceso de validación personalizado.

valid Este estado es un estado de validación general. Retorna `true` (verdadero) cuando ninguno de los estados restantes es `true` (verdadero), lo que significa que el elemento es válido.
valueMissing Este estado es `true` (verdadero) cuando el atributo `required` fue incluido en el elemento y el campo está vacío.
typeMismatch Este estado es `true` (verdadero) cuando la entrada no es el valor esperado de acuerdo al tipo de campo (por ejemplo, cuando se espera un email o una URL).
patternMismatch Este estado es `true` (verdadero) cuando la entrada no es un valor admitido por la expresión regular especificada con el atributo `pattern`.
tooLong Este estado es `true` (verdadero) cuando el largo de la entrada es mayor que el valor especificado en el atributo `maxlength`.
rangeUnderflow Este estado es `true` (verdadero) cuando la entrada es menor que el valor declarado para el atributo `min`.
rangeOverflow Este estado es `true` (verdadero) cuando la entrada es mayor que el valor declarado para el atributo `max`.
stepMismatch Este estado es `true` (verdadero) cuando el valor declarado para el atributo `step` no corresponde con los valores en los atributos `min`, `max` y `value`.
customError Este estado es `true` (verdadero) cuando un error personalizado fue declarado para el elemento.