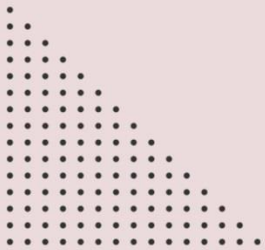
JS

Docs_

Formularios II



1. Propiedades y Métodos de Formularios
2. Enfocado: enfoque/desenfoque
3. Eventos: change, input, cut, copy, paste
4. Formularios: evento y método submit

@mihifidem 03/2024

Formularios en Javascript II

1. Propiedades y Métodos de Formularios

Los formularios y controles, como `<input>`, tienen muchos eventos y propiedades especiales.

Trabajar con formularios será mucho más conveniente cuando los aprendamos.

Navegación: Formularios y elementos

Los formularios del documento son miembros de la colección especial `document.forms`.

Esa es la llamada “Colección nombrada”: es ambas cosas, nombrada y ordenada. Podemos usar el nombre o el número en el documento para conseguir el formulario.

```
document.forms.my; // el formulario con name="my"
document.forms[0]; // el primer formulario en el documento
```

Cuando tenemos un formulario, cualquier elemento se encuentra disponible en la colección nombrada `form.elements`.

Por ejemplo:

```
<form name="my">
  <input name="one" value="1">
  <input name="two" value="2">
</form>

<script>
  // obtención del formulario
  let form = document.forms.my; // elemento <form name="my">

  // get the element
  let elem = form.elements.one; // elemento <input name="one">

  alert(elem.value); // 1
</script>
```

Puede haber múltiples elementos con el mismo nombre. Esto es típico en el caso de los botones de radio y checkboxes.

Formularios en Javascript II

En ese caso `form.elements[name]` es una *colección*. Por ejemplo:

```
<form>
  <input type="radio" name="age" value="10">
  <input type="radio" name="age" value="20">
</form>

<script>
let form = document.forms[0];

let ageElems = form.elements.age;

alert(ageElems[0]); // [object HTMLInputElement]
</script>
```

Estas propiedades de navegación no dependen de la estructura de las etiquetas. Todos los controles, sin importar qué tan profundos se encuentren en el formulario, están disponibles en `form.elements`.

Fieldsets como "sub-formularios"
Un formulario puede tener uno o varios elementos `<fieldset>` dentro. Estos también tienen la propiedad `elements` que lista los controles del formulario dentro de ellos.

Por ejemplo:

```
<body>
  <form id="form">
    <fieldset name="userFields">
      <legend>info</legend>
      <input name="login" type="text">
    </fieldset>
  </form>

  <script>
    alert(form.elements.login); // <input name="login">

    let fieldset = form.elements.userFields;
    alert(fieldset); // HTMLFieldSetElement

    // podemos obtener el input por su nombre tanto desde el formulario como desde el
fieldset
    alert(fieldset.elements.login == form.elements.login); // true
  </script>
</body>
```

Formularios en Javascript II

Notación corta: `form.name`

Hay una notación corta: podemos acceder el elemento como `form[index/name]`.

En otras palabras, en lugar de `form.elements.login` podemos escribir `form.login`.

Esto también funciona, pero tiene un error menor: si accedemos un elemento, y cambiamos su `name`, se mantendrá disponible mediante el nombre anterior (así como mediante el nuevo).

Esto es fácil de ver en un ejemplo:

```
<form id="form">
  <input name="login">
</form>

<script>
  alert(form.elements.login == form.login); // true, el mismo <input>

  form.login.name = "username"; // cambiamos el nombre el <input>

  // form.elements actualiza el nombre:
  alert(form.elements.login); // undefined
  alert(form.elements.username); // input

  // form permite ambos nombres: el nuevo y el viejo
  alert(form.username == form.login); // true
</script>
```

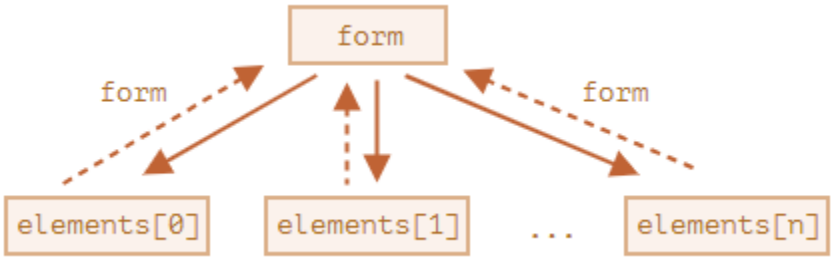
Esto usualmente no es un problema, porque raramente se cambian los nombres de los elementos de un formulario.

Formularios en Javascript II

Referencia inversa: element.form

Para cualquier elemento, el formulario está disponible como `element.form`. Así que un formulario referencia todos los elementos, y los elementos referencian el formulario.

Aquí la imagen:



Por ejemplo:

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

<script>
  // form -> element
  let login = form.login;

  // element -> form
  alert(login.form); // HTMLFormElement
</script>
```

Formularios en Javascript II

Elementos del formulario

Hablemos de los controles de los formularios.

input y textarea

Podemos acceder sus valores como `input.value` (cadena) o `input.checked` (booleano) para casillas de verificación (checkboxes) y botones de opción (radio buttons).

De esta manera:

```
input.value = "New value";
textarea.value = "New text";

input.checked = true; // para checkboxes o radios
```

Usa `textarea.value`, no `textarea.innerHTML`
Observa que incluso aunque `<textarea>...</textarea>` contenga su valor como HTML anidado, nunca deberíamos usar `textarea.innerHTML` para acceder a él.

Esto solo guarda el HTML que había inicialmente en la página, no su valor actual.

select y option

Un elemento `<select>` tiene 3 propiedades importantes:

1. `select.options` – la colección de subelementos `<option>`,
2. `select.value` – el valor del `<option>` seleccionado actualmente, y
3. `select.selectedIndex` – el número del `<option>` seleccionado actualmente.

Ellas proveen tres formas diferentes de asignar un valor para un elemento `<select>`:

1. Encontrar el elemento `<option>` correspondiente (por ejemplo entre `select.options`) y asignar a su `option.selected` un `true`.
2. Si conocemos un nuevo valor: Asignar tal valor a `select.value`.
3. Si conocemos el nuevo número de opción: Asignar tal número a `select.selectedIndex`.

Formularios en Javascript II

Aquí hay un ejemplo de los tres métodos:

```
<select id="select">
  <option value="apple">Apple</option>
  <option value="pear">Pear</option>
  <option value="banana">Banana</option>
</select>

<script>
  // las tres líneas hacen lo mismo
  select.options[2].selected = true;
  select.selectedIndex = 2;
  select.value = 'banana';
  // Recuerda que las opciones comienzan en cero, así que index 2 significa la tercera opción.
</script>
```

A diferencia de la mayoría de controles, `<select>` permite seleccionar múltiples opciones a la vez si tiene el atributo `multiple`. Esta característica es raramente utilizada.

En ese caso, necesitamos usar la primera forma: Añade/elimina la propiedad `selected` de los subelementos `<option>`.

Podemos obtener su colección como `select.options`, por ejemplo:

```
<select id="select" multiple>
  <option value="blues" selected>Blues</option>
  <option value="rock" selected>Rock</option>
  <option value="classic">Classic</option>
</select>

<script>
  // obtener todos los valores seleccionados del multi-select
  let selected = Array.from(select.options)
    .filter(option => option.selected)
    .map(option => option.value);

  alert(selected); // blues,rock
</script>
```

La especificación completa del elemento `<select>` está disponible en la especificación https://html.spec.whatwg.org/multipage/forms.html#the-select-element.

Formularios en Javascript II

new Option

En la especificación hay una sintaxis muy corta para crear elementos `<option>`:

```
option = new Option(text, value, defaultSelected, selected);
```

Esta sintaxis es opcional. Podemos usar `document.createElement('option')` y asignar atributos manualmente. Aún puede ser más corta, aquí los parámetros:

- `text` – el texto dentro del option,
- `value` – el valor del option,
- `defaultSelected` – si es `true`, entonces se le crea el atributo HTML `selected`,
- `selected` – si es `true`, el option se selecciona.

La diferencia entre `defaultSelected` y `selected` es que `defaultSelected` asigna el atributo HTML, el que podemos obtener usando `option.getAttribute('selected')`, mientras que `selected` hace que el option esté o no seleccionado.

En la práctica, uno debería usualmente establecer *ambos* valores en `true` o `false`. O simplemente omitirlos, quedarán con el predeterminado `false`.

Por ejemplo, aquí creamos un nuevo Option "unselected":

```
let option = new Option("Text", "value");
// crea <option value="value">Text</option>
```

El mismo elemento, pero seleccionado:

```
let option = new Option("Text", "value", true, true);
```

Los elementos Option tienen propiedades:

`option.selected`

Es el option seleccionado.

`option.index`

El número del option respecto a los demás en su `<select>`.

`option.text`

El contenido del option (visto por el visitante).

Formularios en Javascript II

Referencias

- Especificación: https://html.spec.whatwg.org/multipage/forms.html.

Resumen

Navegación de formularios:

`document.forms`

: Un formulario está disponible como `document.forms[name/index]`.

`form.elements`

: Los elementos del formulario están disponibles como `form.elements[name/index]`, o puedes usar solo `form[name/index]`. La propiedad `elements` también funciona para los `<fieldset>`.

`element.form`

: Los elementos referencian a su formulario en la propiedad `form`.

El valor está disponible con `input.value`, `textarea.value`, `select.value` etc. Para checkboxes y radios, usa `input.checked` para determinar si el valor está seleccionado.

Para `<select>` también podemos obtener el valor con el índice `select.selectedIndex` o a través de la colección de opciones `select.options`.

Esto es lo básico para empezar a trabajar con formularios. Conoceremos muchos ejemplos más adelante en el tutorial.

En el siguiente capítulo vamos a hablar sobre los eventos `focus` y `blur` que pueden ocurrir en cualquier elemento, pero son manejados mayormente en formularios.

Formularios en Javascript II

2. Enfocado: enfoque/desenfoque

Un elemento se enfoca cuando el usuario hace click sobre él o al pulsar `Tab` en el teclado. Existen también un atributo `autofocus` de HTML que enfoca un elemento por defecto cuando una página carga, y otros medios de conseguir el enfoque.

Enfocarse sobre un elemento generalmente significa: "prepárate para aceptar estos datos", por lo que es el momento en el cual podemos correr el código para inicializar la funcionalidad requerida.

El momento de desenfoque ("blur") puede ser incluso más importante. Ocurre cuando un usuario clica en otro punto o presiona `Tab` para ir al siguiente campo de un formulario. También hay otras maneras.

Perder el foco o desenfocarse generalmente significa: "los datos ya han sido introducidos", entonces podemos correr el código para comprobarlo, o para guardarlo en el servidor, etc.

Existen importantes peculiaridades al trabajar con eventos de enfoque. Haremos lo posible para abarcarlas a continuación.

Eventos focus/blur

El evento `focus` es llamado al enfocar, y el `blur` cuando el elemento pierde el foco.

Utilicémoslos para la validación de un campo de entrada.

En el ejemplo a continuación:

- El manejador `blur` comprueba si se ha introducido un correo, y en caso contrario muestra un error.

Formularios en Javascript II

- El manejador `focus` esconde el mensaje de error (en `blur` se volverá a comprobar):

```
<style>
  .invalid { border-color: red; }
  #error { color: red }
</style>

Su correo por favor: <input type="email" id="input">

<div id="error"></div>

<script>
input.onblur = function() {
  if (!input.value.includes('@')) { // not email
    input.classList.add('invalid');
    error.innerHTML = 'Por favor introduzca un correo válido.'
  }
};

input.onfocus = function() {
  if (this.classList.contains('invalid')) {
    // quitar la  indicación "error", porque el usuario quiere reintroducir algo
    this.classList.remove('invalid');
    error.innerHTML = "";
  }
};
</script>
```

El HTML actual nos permite efectuar diversas validaciones utilizando atributos de entrada: `required`, `pattern`, etc. Y muchas veces son todo lo que necesitamos. JavaScript puede ser utilizado cuando queremos más flexibilidad. También podríamos enviar automáticamente el valor modificado al servidor si es correcto.

Formularios en Javascript II

Métodos focus/blur

Los métodos `elem.focus()` y `elem.blur()` ponen/quitan el foco sobre el elemento.

Por ejemplo, impidamos al visitante que deje la entrada si el valor es inválido:

```
<style>
  .error {
    background: red;
  }
</style>

Su correo por favor: <input type="email" id="input">
<input type="text" style="width:220px" placeholder="hacer que el correo sea inválido y tratar de enfocar aquí">

<script>
  input.onblur = function() {
    if (!this.value.includes('@')) { // no es un correo
      // mostrar error
      this.classList.add("error");
      // ...y volver a enfocar
      input.focus();
    } else {
      this.classList.remove("error");
    }
  };
</script>
```

Funciona en todos los navegadores excepto Firefox (bug).

Si introducimos algo en la entrada y luego intentamos pulsar `Tab` o hacer click fuera del `<input>`, entonces `onblur` lo vuelve a enfocar.

Por favor tened en cuenta que no podemos “prevenir perder el foco” llamando a `event.preventDefault()` en `onblur`, porque `onblur` funciona *después* de que el elemento perdió el foco.

Aunque en la práctica uno debería pensarlo bien antes de implementar algo como esto, porque generalmente *debemos mostrar errores* al usuario, pero *no evitar que siga adelante* al llenar nuestro formulario. Podría querer llenar otros campos primero.

Formularios en Javascript II

Pérdida de foco iniciada por JavaScript
Una pérdida de foco puede ocurrir por diversas razones.

Una de ellas ocurre cuando el visitante clica en algún otro lado. Pero el propio JavaScript podría causarlo, por ejemplo:

- Un `alert` traslada el foco hacia sí mismo, lo que causa la pérdida de foco sobre el elemento (evento `blur`). Y cuando el `alert` es cerrado, el foco vuelve (evento `focus`).
- Si un elemento es eliminado del DOM, también causa pérdida de foco. Si es reinsertado el foco no vuelve.

Estas situaciones a veces causan que los manejadores `focus/blur` no funcionen adecuadamente y se activen cuando no son necesarios.

Es recomendable tener cuidado al utilizar estos eventos. Si queremos monitorear pérdidas de foco iniciadas por el usuario deberíamos evitar causarlas nosotros mismos.

Permitir enfocado sobre cualquier elemento: tabindex

Por defecto, muchos elementos no permiten enfoque.

La lista varía un poco entre navegadores, pero una cosa es siempre cierta: `focus/blur` está garantizado para elementos con los que el visitante puede interactuar: `<button>`, `<input>`, `<select>`, `<a>`, etc.

En cambio, elementos que existen para formatear algo, tales como `<div>`, `<span>`, `<table>`, por defecto no son posibles de enfocar. El método `elem.focus()` no funciona en ellos, y los eventos `focus/blur` no son desencadenados.

Esto puede ser modificado usando el atributo HTML `tabindex`.

Cualquier elemento se vuelve enfocable si contiene `tabindex`. El valor del atributo es el número de orden del elemento cuando `Tab` (o algo similar) es utilizado para cambiar entre ellos.

Es decir: si tenemos dos elementos donde el primero contiene `tabindex="1"` y el segundo contiene `tabindex="2"`, al presionar `Tab` estando situado sobre el primer elemento se traslada el foco al segundo.

El orden de cambio es el siguiente: los elementos con `tabindex` de valor "1" y mayores tienen prioridad (en el orden `tabindex`) y después los elementos sin `tabindex` (por ejemplo un `<input>` estándar).

Elementos sin el `tabindex` correspondiente van cambiando en el orden del código fuente del documento (el orden por defecto).

Formularios en Javascript II

Existen dos valores especiales:

- `tabindex="0"` incluye al elemento entre los que carecen de `tabindex`. Esto es, cuando cambiamos entre elementos, elementos con `tabindex="0"` van después de elementos con `tabindex ≥ "1"`.

  Habitualmente se utiliza para hacer que un elemento sea enfocable y a la vez mantener intacto el orden de cambio por defecto. Para hacer que un elemento sea parte del formulario a la par con .

- `tabindex="-1"` permite enfocar un elemento solamente a través de código. `Tab` ignora estos elementos, pero el método `elem.focus()` funciona.

  Por ejemplo, he aquí una lista. Clique sobre el primer ítem y pulse `Tab`:

  ```
  Clique sobre el primer ítem y pulse `key:Tab`. Fíjese en el orden. Note que subsiguientes `key:Tab` pueden desplazar el foco fuera del iframe en el ejemplo.
  ```

  ```
  <ul>
    <li tabindex="1">Uno</li>
    <li tabindex="0">Cero</li>
    <li tabindex="2">Dos</li>
    <li tabindex="-1">Menos uno</li>
  </ul>

  <style>
    li { cursor: pointer; }
    :focus { outline: 1px dashed green; }
  </style>
  ```

  El orden es el siguiente: `1 - 2 - 0`. Normalmente, `<li>` no admite enfocado, pero `tabindex` lo habilita, junto con eventos y estilado con `:focus`.

  La propiedad `elem.tabIndex` también funciona
  Podemos añadir `tabindex` desde JavaScript utilizando la propiedad `elem.tabIndex`. Se consigue el mismo resultado.

Formularios en Javascript II

Delegación: focusin/focusout

Los eventos `focus` y `blur` no se propagan.

Por ejemplo, no podemos añadir `onfocus` en

para resaltarlo, así:

```
<!-- enfocando en el formulario -- añadir la clase -->
<form onfocus "this.className='focused'">
  <input type="text" name="name" value="Name">
  <input type="text" name="surname" value="Surname">
</form>

<style> .focused { outline: 1px solid red; } </style>
```

El ejemplo anterior no funciona porque cuando el usuario enfoca sobre un [ ] el evento ´focus´ se dispara solamente sobre esa entrada y no se propaga, por lo que `form.onfocus` nunca se dispara.

Existen dos soluciones.

Primera: hay una peculiar característica histórica: `focus/blur` no se propagan hacia arriba, pero lo hacen hacia abajo en la fase de captura.

Esto funcionará:

```
<form id="form">
  <input type="text" name="name" value="Name">
  <input type="text" name="surname" value="Surname">
</form>

<style> .focused { outline: 1px solid red; } </style>

<script>
  // pon el manejador en fase de captura (último argumento "verdadero")
  form.addEventListener("focus", () => form.classList.add('focused'), true);
  form.addEventListener("blur", () => form.classList.remove('focused'), true);
</script>
```

Segunda: existen los eventos `focusin` y `focusout`, exactamente iguales a `focus/blur`, pero se propagan.

Hay que tener en cuenta que han de asignarse utilizando `elem.addEventListener`, no `on<event>`.

Formularios en Javascript II

La otra opción que funciona:

```
<form id="form">
  <input type="text" name="name" value="Name">
  <input type="text" name="surname" value="Surname">
</form>

<style> .focused { outline: 1px solid red; } </style>

<script>
  form.addEventListener("focusin", () => form.classList.add('focused'));
  form.addEventListener("focusout", () => form.classList.remove('focused'));
</script>
```

Resumen

Los eventos `focus` y `blur` hacen que un elemento se enfoque/pierda el foco.

Se caracterizan por lo siguiente:

- No se propagan. En su lugar se puede capturar el estado o usar `focusin/focusout`.
- La mayoría de los elementos no permiten enfoque por defecto. Utiliza `tabindex` para hacer cualquier elemento enfocable.

El elemento que en el momento tiene el foco está disponible como `document.activeElement`.

Formularios en Javascript II

3. Eventos: change, input, cut, copy, paste

Veamos varios eventos que acompañan la actualización de datos.

Evento: change

El evento `change` se activa cuando el elemento finaliza un cambio.

Para ingreso de texto significa que el evento ocurre cuando se pierde foco en el elemento.

Por ejemplo, mientras estamos escribiendo en el siguiente cuadro de texto, no hay evento. Pero cuando movemos el focus (enfoque) a otro lado, por ejemplo hacemos click en un botón, entonces ocurre el evento `change`:

```
<input type="text" onchange="alert(this.value)">
<input type="button" value="Button">
```

Para otros elementos: `select`, `input type=checkbox/radio` se dispara inmediatamente después de cambiar la opción seleccionada:

```
<select onchange="alert(this.value)">
  <option value="">Select something</option>
  <option value="1">Option 1</option>
  <option value="2">Option 2</option>
  <option value="3">Option 3</option>
</select>
```

Evento: input

El evento `input` se dispara cada vez que un valor es modificado por el usuario.

A diferencia de los eventos de teclado, ocurre con el cambio a cualquier valor, incluso aquellos que no involucran acciones de teclado: copiar/pegar con el mouse o usar reconocimiento de voz para dictar texto.

Formularios en Javascript II

Por ejemplo:

```
<input type="text" id="input"> oninput: <span id="result"></span>

<script>
  input.oninput = function() {
    result.innerHTML = input.value;
  };
</script>
```

Si queremos manejar cualquier modificación en un `<input>` entonces este evento es la mejor opción.

Por otro lado, el evento `input` no se activa con entradas del teclado u otras acciones que no involucren modificar un valor, por ejemplo presionar las flechas de dirección ⇦ ⇨ mientras se está en el input.

No podemos prevenir nada en oninput
El evento `input` se dispara después de que el valor es modificado.

Por lo tanto no podemos usar `event.preventDefault()` aquí, es demasiado tarde y no tendría efecto.

Eventos: cut, copy, paste

Estos eventos ocurren al cortar/copiar/pegar un valor.

Estos pertenecen a la clase ClipboardEvent y dan acceso a los datos cortados/copiados/pegados.

También podemos usar `event.preventDefault()` para cancelar la acción y que nada sea cortado/copiado/pegado.

El siguiente código también evita todo evento `cut/copy/paste` y muestra qué es los que estamos intentando cortar/copiar/pegar:

Formularios en Javascript II

```
<input type="text" id="input">
<script>
  input.onpaste = function(event) {
    alert("paste: " + event.clipboardData.getData('text/plain'));
    event.preventDefault();
  };

  input.oncut = input.oncopy = function(event) {
    alert(event.type + '-' + document.getSelection());
    event.preventDefault();
  };
</script>
```

Nota que dentro de los manejadores `cut` y `copy`, llamar a `event.clipboardData.getData(...)` devuelve un string vacío. Esto es porque el dato no está en el portapapeles aún. Y si usamos `event.preventDefault()` no será copiado en absoluto.

Por ello el ejemplo arriba usa `document.getSelection()` para obtener el texto seleccionado. Puedes encontrar más detalles acerca de selección en el artículo Selection y Range.

No solo es posible copiar/pegar texto, sino cualquier cosa. Por ejemplo, podemos copiar un archivo en el gestor de archivos del SO y pegarlo.

Esto es porque `clipboardData` implementa la interfaz `DataTransfer`, usada comúnmente para "arrastrar y soltar" y "copiar y pegar". Ahora esto está fuera de nuestro objetivo, pero puedes encontrar sus métodos en la especificación DataTransfer.

Hay además una API asincrónica adicional para acceso al portapapeles: `navigator.clipboard`. Más en la especificación Clipboard API and events, no soportado en Firefox.

Restricciones de seguridad

El portapapeles es algo a nivel "global" del SO. Un usuario puede alternar entre ventanas, copiar y pegar diferentes cosas, y el navegador no debería ver todo eso.

Por ello la mayoría de los navegadores dan acceso al portapapeles únicamente bajo determinadas acciones del usuario, como copiar y pegar.

Está prohibido generar eventos "personalizados" del portapapeles con `dispatchEvent` en todos los navegadores excepto Firefox. Incluso si logramos enviar tal evento, la especificación establece que tal evento "sintético" no debe brindar acceso al portapapeles.

Formularios en Javascript II

Incluso si alguien decide guardar `event.clipboardData` en un manejador de evento para accederlo luego, esto no funcionará.

Para reiterar, event.clipboardData funciona únicamente en el contexto de manejadores de eventos iniciados por el usuario.

Por otro lado, navigator.clipboard es una API más reciente, pensada para el uso en cualquier contexto. Esta pide autorización al usuario cuando la necesita.

Resumen

Eventos de modificación de datos:

Evento	Descripción	Especiales
`change`	Un valor fue cambiado.	En ingreso de texto, se dispara cuando el elemento pierde el foco
`input`	Cada cambio de entrada de texto	Se dispara de inmediato con cada cambio, a diferencia de `change`.
`cut/copy/paste`	Acciones cortar/copiar/pegar	La acción puede ser cancelada. La propiedad `event.clipboardData` brinda acceso al portapeles. Todos los navegadores excepto Firefox también soportan `navigator.clipboard`.

Formularios en Javascript II

4. Formularios: evento y método submit

El evento submit se activa cuando el formulario es enviado, normalmente se utiliza para validar el formulario antes de ser enviado al servidor o bien para abortar el envío y procesarlo con JavaScript.

El método form.submit() permite iniciar el envío del formulario mediante JavaScript. Podemos utilizarlo para crear y enviar nuestros propios formularios al servidor.

Veamos más detalles sobre ellos.

Evento: submit

Mayormente un formulario puede enviarse de dos maneras:

1. La primera – Haciendo click en <input type="submit"> o en <input type="image">.
2. La segunda – Pulsando la tecla Enter en un campo del formulario.

Ambas acciones causan que el evento submit sea activado en el formulario. El handler puede comprobar los datos, y si hay errores, mostrarlos e invocar event.preventDefault(), entonces el formulario no será enviado al servidor.

En el formulario de abajo:

1. Ve al campo tipo texto y pulsa la tecla Enter.
2. Haz click en <input type="submit">.

Ambas acciones muestran alert y el formulario no es enviado debido a la presencia de return false:

```
<form onsubmit="alert('submit!');return false">
  Primero: Enter en el campo de texto <input type="text" value="texto"><br>
  Segundo: Click en "submit": <input type="submit" value="Submit">
</form>
```

Formularios en Javascript II

Relación entre submit y click

Cuando un formulario es enviado utilizando `Enter` en un campo tipo texto, un evento click se genera en el <input type="submit">

Muy curioso, dado que no hubo ningún click en absoluto.

Aquí esta la demo:

```
<form onsubmit="return false">
 <input type="text" size="30" value="Sitúa el cursor aquí y pulsa Enter">
 <input type="submit" value="Submit" onclick="alert('click')">
</form>
```

Método: submit

Para enviar un formulario al servidor manualmente, podemos usar form.submit().

Entonces el evento submit no será generado. Se asume que si el programador llama form.submit(), entonces el script ya realizó todo el procesamiento relacionado.

A veces es usado para crear y enviar un formulario manualmente, como en este ejemplo:

```
let form = document.createElement('form');
form.action = 'https://google.com/search';
form.method = 'GET';

form.innerHTML = '<input name="q" value="test">';

// el formulario debe estar en el document para poder enviarlo
document.body.append(form);

form.submit();
```