|  |
| --- |
| **Comprensión de los formularios** |

**Selección de los elementos de un formulario**

El primer paso para trabajar con los elementos de una página es seleccionarlos. Con JQuery lo podemos hacer de este modo:

*var campoUsuario = $('#usuario');*

Si queremos conocer el valor del campo:

*var campoUsuario = $('#usuario').val();*

JQuery ofrece un conjunto de selectores para seleccionar tipos concretos de campos:

|  |  |  |
| --- | --- | --- |
| **SELECTOR** | **EJEMPLO** | **ACCIÓN** |
| :input $ | (':input $') | Selecciona todos los elementos input, textarea, select y button (todos los elementos del formulario). |
| :text $ | (':text $') | Selecciona todos los campos de texto. |
| :password | (':password $') | Selecciona todos los campos de contraseña. |
| :radio $ | (':radio $') | Selecciona todos los botones de opción. |
| :checkbox $ | (':checkbox $') | Selecciona todas las casillas de verificación. |
| :submit $ | (':submit $') | Selecciona todos los botones de envío. |
| :image $ | (':image $') | Selecciona todos los botones de imagen. |
| :reset $ | (':reset $') | Selecciona todos los botones de borrado del contenido del formulario. |
| :button $ | (':button $') | Selecciona todos los campos cuyo tipo sea button. |
| :file $ | (':file $') | Selecciona todos los campos de archivo |
| :hidden $ | (':hidden $') | Selecciona todos los campos ocultos. |

Supongamos que cuando el usuario envía el formulario queremos comprobar que todos los campos de texto contengan algún valor. Tendremos que seleccionar todos los campos de texto y luego comprobar uno por uno si contienen algún valor. El primer paso es:

*$(':text');*

Luego podemos usar la función *each()* para componer un bucle con todos los resultados y comprobar si contienen algún valor.

También podemos combinar los selectores de los formularios con otros diferentes. Imaginemos que tenemos dos formularios en una página y queremos seleccionar los campos de texto de uno de ellos. Suponiendo que los campos que nos interesen pertenecen a un formulario que tiene el identificador “*firma*”, tendremos que utilizar este código para seleccionar sus campos:

*$('#firma :text');*

JQuery proporciona **filtros** muy útiles para localizar los campos que tengan un estado determinado:

***:checked*** selecciona todos los campos cuyo elemento de selección esté activo; pueden ser casillas de verificación y botones de radio. Ejemplo: *$(':checked').* Podemos utilizar este filtro para localizar qué botón de opción de un grupo ha sido seleccionado. Por ejemplo, si tenemos un grupo de botones de opción y queremos conocer cuál ha seleccionado el usuario, todos los atributos *name* deben tener el mismo valor. Supongamos que los botones de opción del grupo tiene el atributo *name* con el valor “*envio*”. Podemos utilizar el selector de atributos junto con el filtro *:checked* para localizar el botón o los botones de opción seleccionados por el usuario:

*var valorChequeado = $('input[name=”envio”]:checked').val();*

La función *val()* devuelve el valor almacenado en la casilla(s).

***:selected*** selecciona todos los elementos opcionales que se encuentre dentro de una lista o un menú. Supongamos que tenemos una etiqueta *<select>* con el identificador estado donde aparecen todas las provincias. Si queremos localizar la provincia que ha seleccionado el usuario, éste sería el código:

*var provinciaSeleccionada = $('#provincia :selected').val();*

A diferencia del filtro anterior, en este caso dejamos un espacio entre el nombre del identificador y el filtro. Se hace así porque este filtro trabaja con las etiquetas *<option>* y no con *<select>*. Esta selección localiza todas las opciones seleccionadas que se encuentran dentro de la etiqueta *<select>* cuyo identificador sea provincia. Gracias al espacio, logramos que esta declaración funciones como un selector CSS descendiente: en primer lugar localiza el elemento con el identificador apropiado, y luego busca dentro de él cualquier elemento que se haya seleccionado.

**Nota**: podemos activar selecciones múltiples en un menú *<select>*. Es decir, el filtro *:selected* podría devolver más de un elemento.

**Obtener y definir el valor de un elemento de un formulario**

Existen situaciones en la que podemos necesitar comprobar el valor de un elemento de un formulario. Por ejemplo, si queremos comprobar si el usuario ha escrito su dirección de email dentro de un campo de texto, o si queremos determinar el valor de un campo para calcular el coste del pedido. A veces, querremos definir el valor de un elemento del formulario. Por ejemplo, supongamos que tenemos un formulario que solicita la dirección de facturación y la dirección de entrega. En este caso, para quitar trabajo al usuario, podemos añadir un *checkbox* que dijese “*Misma dirección de facturación*”. Si el usuario la activa, los campos del formulario de la dirección de entrega se rellenarán automáticamente con los datos del campo de facturación.

JQuery tiene una función capaz de llevar a cabo las dos tareas. ***val()*** puede leer y definir el valor de un campo de formulario. Si la llamamos sin argumentos, leerá el valor almacenado en el campo. Por ejemplo, supongamos que tenemos un campo para solicitar la dirección del email del usuario. Para localizar el contenido del campo, utilizaremos esta línea de código:

*var valor = $('#email').val();*

Para definir el valor de un campo, basta con entregárselo a la función *val()*. Por ejemplo, supongamos que tenemos un formulario para pedir productos y queremos calcular de forma automática el coste total de la compra basándonos en la cantidad de artículos que ha seleccionado el usuario. Este sería el código:

*var costeUnitario=9.25;*

*var importe=$('#cantidad').val();*

*var total = importe=cantidad\*costeUnitario;*

*total=total.toFixed(2);*

*$('#total').val(total); //define el valor*

**Determinar si se han activado o no las casillas de verificación**

En HTML, el atributo *checked* determina si se ha activado un elemento. Por ejemplo, para activar un *checkbox* al cargar una página, tendremos que añadirle el atributo *checked*:

*<input name="Checkbox1" type="checkbox" checked="checked" />*

En HTML5 sería:

*<input name="noticias" type="checkbox" id="noticias" checked="" />*

La propiedad *checked* se puede cambiar dinámicamente. Existen otros elementos de formularios que también tienen propiedades dinámicas. Por ejemplo, los campos de texto tienen una propiedad *disabled*, que determina si el usuario puede escribir en el campo de texto o no.

Cuando tengamos que trabajar con propiedades DOM utilizaremos el método *prop()* de JQuery:

*if($('#noticias').prop('checked') {*

*// la casilla esta activada*

*}*

*else {*

*} // la casilla esta desactivada*

La propiedad *checked* también se puede aplicar a los botones de radio.

**Eventos para formularios**

Los eventos permiten añadir interactividad a las páginas web. Estos son algunos de ellos.

**Submit**

Se activa con los botones de envío de los formularios y al pulsar la tecla *Intro* para pasar de un campo a otro. Se utiliza este evento para ejecutar el script cuando se envíe el formulario. Esto permitirá la validación.

Veamos un ejemplo de valor requerido en un campo:

*$(document).ready(function() {*

*$('#firma').submit(function() {*

*if($('nombreUsuario').val() == '') {*

*alert('Por favor, suministre un nombre para el campo');*

*return false;*

*}*

*}); // fin de submit()*

*}); // fin de ready()*

La primera línea define la función que solo se ejecutará después de que se cargue el código HTML de la página. La segunda línea adjunta una función al evento *submit* del formulario. El resto de líneas contienen la rutina de validación.

**Nota**: también podemos utilizar la función *preventDefault()* del objeto del evento para evitar que se envíe el formulario:

*$('form').submit(function(evt) {*

*// detiene el envío*

*evt.preventDefault();*

*}*

El evento *submit* solo se aplica a los formularios. Por eso, debemos empezar por seleccionar un formulario y asociarle dicha función. Para seleccionar el formulario, podemos utilizar su identificador o, si solo hubiese un formulario en la página, utilizar un selector de elemento como éste:

*$('form').submit(function() {*

*// Aquí va el código que se debe ejecutar para enviar el formulario*

**Focus**

Hay diseñadores que utilizan este evento para borrar cualquier texto que hubiese en un campo de un formulario. Por ejemplo, supongamos que tenemos el siguiente código:

*<input name="nombreUsuario" id="nombreUsuario" type="text"*

*value="Por favor, introduzca su nombre"/>*

En vez de obligarle a borra el texto que aparezca en el campo antes de escribir su nombre de usuario, podríamos eliminarlo cuando el foco se centrase en dicho campo:

*$('#nombreUsuario').focus(function() {*

*var campo = $(this);*

*if (campo.val()==campo.prop('defaultvalue')) {*

*campo.val('');*

*}*

*});*

La segunda línea crea una variable en donde almacena una referencia a la selección de JQuery.

La cuarta línea elimina el contenido del campo. Pero no queremos eliminar el valor del campo cada vez que el navegador le asigne el foco. Por ejemplo, supongamos que alguien accede al formulario y hace click en este campo. La primera vez que ocurra esto se eliminará el texto inicialmente asignado en el control. Pero si el usuario escribiese su nombre, hiciese click fuera del campo y después volviese a hacer click dentro de él, no querríamos que desapareciese la información que ha proporcionado. Es aquí en donde entra en juego la línea 3.

Los campos de texto tienen una propiedad denominada *defaultValue*, que contiene el texto del campo cuando se carga la página por primera vez. Aunque eliminemos el texto del campo, el navegador recordará la cadena que contenía en el momento de cargar la página. La declaración condicional comprueba si el contenido de este texto (*campo.val()*) es el mismo que existía originalmente cuando se cargó la página (*campo.prop('defaultvalue')).* En caso afirmativo, el intérprete de JavaScript borrará el valor de este campo.

Veamos un **ejemplo completo**. Supongamos que cargamos por primera vez la página con el valor por defecto “*Introduzca por favor su nombre*”; este es el valor de la propiedad “*defaultValue*”. Así pues, cuando el usuario haga click sobre este campo, la declaración condicional se preguntará: ¿El contenido de este campo es el mismo que tenía cuando se cargó la página? En caso de que sea así, eliminará el contenido del campo.

Sin embargo, supongamos que escribimos *martinez* como nombre de usuario, luego saltamos a otro campo y nos damos cuenta de que hemos escrito erróneamente el nombre. Al volver a hacer click sobre el campo que contiene el nombre, se volverá a iniciar el evento *focus*, y se volverá a ejecutar la función asociada, esta vez el intérprete de JavaScript se preguntará: ¿es igual *martinez* que el texto que inicialmente tenía el campo? Como la respuesta es negativa, no eliminará el contenido de este campo y el usuario podrá corregir su error.

***Nota***: los formularios HTML5 admiten el atributo *placeholder* que nos permite colocar un mensaje temporal dentro del cuadro de texto. Este mensaje se borrará cuando el usuario comience a escribir en dicho campo:

*<input name="nombreUsuario" type="text" id="nombreUsuario"*

*placeholder="Por favor, introduzca el nombre" />*

Esta técnica es más sencilla que el código anteriormente expuesto, pero este atributo no es compatible con Internet Explorer 9, ni con sus versiones anteriores.

**Blur**

Cuando abandonamos un campo hacemos click fuera de él, y el navegador iniciará el evento *blur*. Este evento se suele utilizar con los campos *text* y *textarea* para ejecutar algún script de validación. Supongamos que tenemos un formulario muy grande con muchas preguntas, muchas de las cuáles requieren algún valor especial (como por ejemplo un email, teléfonos, etc…). Supongamos que el usuario no ha rellenado correctamente alguno de estos campos, pero pulsa el botón *Enviar*. Nuestro código generará una lista de errores pero en vez de presentárselos todos simultáneamente en la pantalla, podemos ir comprobando los campos según se vaya rellenando el formulario. De esta forma, si se equivoca al cumplimentar alguno, se lo notificaremos de inmediato y lo podrá corregir.

Por ejemplo, supongamos que tenemos un campo para recopilar el número de productos que desea comprar el usuario. El código HTML sería:

*<input name="cantidad" type="text" id="cantidad" />*

Queremos controlar que el campo solo contenga números. Para realizar esta comprobación inmediatamente después de que el usuario abandone el campo, éste sería el código:

*$('#cantidad).blur(function() {*

*var valorCampo = $(this).val();*

*if (isNaN(valorCampo)) {*

*alert('Por favor, introduzca un número');*

*}*

*});*

La primera línea asigna una función al evento *blur*. La segunda línea recupera el valor del campo y lo almacena en la variable. La tercera utiliza el método *isNaN* para comprobar que este valor es un número.

Si tuviésemos un formulario con muchos campos que necesitasen valores numéricos, podríamos asignarles a todos ellos el mismo nombre de clase, como por ejemplo *class="soloNumeros"* y podríamos utilizar el siguiente código para comprobar su contenido:

*$('.soloNumeros).blur(function() {*

*var valorCampo = $(this).val();*

*if (isNaN(valorCampo)) {*

*alert('Por favor, introduzca un número');*

*}*

*});*

Este método nos permite utilizar unas pocas líneas de código para comprobar todos los campos numéricos de un formulario.

**Click**

Se inicia cuando el usuario hace click sobre un elemento del formulario. Es muy útil para trabajar con *botones de radio* y *checkBoxes*. Por ejemplo, supongamos que tenemos un formulario de pedido que proporciona campos separados para la dirección de la facturación y la de envío. Para evitar que los usuarios tengan que escribir dos veces la misma dirección cuando ambas coincidan, podríamos usar una casilla de verificación que diga “*Misma dirección de facturación*”. Si el usuario la activase, se ocultarían los campos destinados a la dirección de envío y se facilitaría la legibilidad del formulario.

Al igual que todos los eventos, podemos utilizar la función *click()* de JQuery para asignar una función al evento click del campo:

*$(':radio').click(function() }*

*// Esta funcion se aplicará a los botones de radio cuando*

*// el usuario haga click sobre ellos*

*});*

**Nota**: el evento click también se aplica a los campos de texto, pero no funciona igual que en el caso del evento *focus*. Este evento (*focus*) se inicia cuando el usuario hace click o utilice el tabulador para acceder a un campo de texto, mientras que el evento click solo se inicia cuando acceda a dicho campo utilizando el ratón.

**Change**

Este evento se aplica a los menús de un formulario. Siempre que el usuario seleccione un elemento de un menú de un formulario, se iniciará el evento *change*. Podemos usarlo para ejecutar la función de validación. Por ejemplo, muchos diseñadores suelen incluir algún tipo de indicación como la primera opción de un menú; por ejemplo, “*Elija un país*”. Para asegurarse de que el usuario seleccione un país y no active accidentalmente la primera opción de menú, podemos comprobar el valor seleccionado.

También podríamos programar el formulario para que cambie de aspecto después de haber realizado una selección. Por ejemplo, podríamos ejecutar una función para que, siempre que se seleccione una opción de un menú, cambien las opciones de un segundo menú.

Para aplicar un controlador del evento *change* a un menú, utilizamos la función *change()* de JQuery. Por ejemplo, supongamos que tenemos un menú donde aparecen los nombres de los países. El identificador de este menú será “*país*”. Cuando el usuario selecciones uno de sus elementos, comprobaremos que no haya seleccionado la primera opción, que será un texto explicativo:

*$('#pais).change(function() {*

*if($(this).val()=='Por favor, elije un país') {*

*alert('Por favor, seleccione un país de este menú.');*

*}*

*}*

|  |
| --- |
| **Formularios inteligentes** |

Debemos conseguir simplificar al máximo la gestión de los formularios para el usuario.

Por ejemplo, podemos ocultar todos los campos hasta que sea necesario, desactivar los que no tengan nada que ver con las opciones activadas por el usuario y calcular los totales basándonos en los elementos que se han seleccionado.

**Foco en el primer campo del formulario**

Lo podemos hacer mediante el evento *focus*. Tendremos que seleccionar el campo, y luego ejecutar la función *focus()*. Por ejemplo:

*$(document).ready(function() {*

*$('#nombreUsuario').focus();*

*});*

Podríamos crear un script genérico que siempre colocase el foco en el primer campo de un formulario, sin necesidad de asignar ningún identificador:

*$(document).ready(function() {*

*$(':text:first').focus();*

*});*

En el código anterior hemos seleccionado la primera instancia de todos los campos de texto y le asignamos el foco.

Si en una página tuviésemos más de un formulario, tendríamos que volver a definir el selector para identificar el formulario en cuyo campo colocaremos el foco. Por ejemplo, supongamos que tenemos una página con los formularios, uno para buscar una dirección y otro para registrarse en un foro. Supongamos que queremos colocar el foco en el primer campo de texto del formulario de registro, pero resulta que el primer campo de la página se encuentra en el formulario de búsqueda. Para colocar el foco en el campo deseado, tendremos que asignar un identificador (por ejemplo ”*firma*”) a dicho formulario. Luego, podríamos utilizar este código:

*$(document).ready(function() {*

*$('#firma :text:first').focus();*

*});*

Ahora, el selector seleccionará el primer campo del formulario de registro.

**Activación y desactivación de campos**

Para desactivar un campo el formulario y que el usuario no pueda cumplimentarlo, usaremos JavaScript. Al desactivar un campo del formulario, el usuario no podrá activarlo (en el caso de los *botones de radio* y los *checkbox*), escribir en él (campos de texto), seleccionarlo (menús) o hacer click sobre él (botones de envío).

Para desactivar, tenemos que asignar el valor true a la propiedad *disabled*. Por ejemplo, para desactivar todos los campos de un formulario, podríamos hacerlo así:

*$(':input').prop('disabled', true);*

Por regla general, desactivaremos los campos de un formulario en respuesta a un evento. Por ejemplo, supongamos que tenemos un formulario para que el usuario rellene la declaración de la renta. Tenemos dos botones de radio para saber su estado civil. Vamos a suponer que el identificador para el estado del botón de soltero es “***soltero***”.

Si está soltero, desactivaremos los campos de los usuarios casados. Vamos a suponer que los casados escribirán en un campo el nombre de su pareja denominado ***nombrePareja***. Tenemos el siguiente código:

*$('#soltero').click(function() {*

*$('#nombrePareja').prop('disabled', true);*

*});*

Si podemos desactivar este campo, necesitamos poder volverlo a activar.

*$(':input').prop('disabled', false);*

|  |
| --- |
| **Nota** |
| Para desactivar un campo de formulario, debemos usar valores booleanos *true* o *false* y no las cadenas ‘*true’* o ‘*false’*.  Por ejemplo, este código no es correcto:  *$(':input).prop('disabled', 'false');*  Éste sí es correcto:  *$(':input).prop('disabled', false);* |

Si seleccionamos la opción “*casado*”, deberemos asegurarnos de activar el nombre del campo de su pareja. Suponiendo que el botón para la opción “*casado*” tuviese el identificador “casado”, podríamos añadir este código:

*if('#casado').click(function) {*

*$('#estado').prop('disabled', false);*

*});*

**Opciones para mostrar y ocultar elementos**

Además de desactivar un campo, existen otras formas de asegurarnos de que el usuario no pierda el tiempo suministrando información innecesaria: ocultar los campos prescindibles.

Por ejemplo, supongamos que en un formulario para la declaración de la renta, queremos ocultar el nombre de la pareja a todos los usuarios que estén solteros, pero lo mostraremos a los usuarios casados:

*$('#soltero').click(function) {*

*$('#estado').hide();*

*});*

*$('#casado').click(function) {*

*$('#estado').show();*

*});*

Una ventaja de ocultar los campos con respecto a la desactivación, es que ayudan a simplificar el aspecto del formulario. Los campos desactivados siguen estando visibles y pueden “*despistar*” a los usuarios.

En algunas ocasiones mostraremos u ocultaremos más información aparte del campo. Es posible que queramos ocultar el campo, su etiqueta y cualquier otro texto asociado a él. Una forma de hacerlo es envolver el código que queramos ocultar (campos, etiquetas o cualquier código HTML) en una etiqueta <div>, asignarle dicha etiqueta a un identificador y por último ocultarla.

|  |
| --- |
| Detención de varios envíos |
| A veces, enviamos información varias veces. ¿Cómo se puede evitar?  A veces existen demoras entre el momento en que el usuario pulsa el botón de envío y recibe el mensaje del servidor indicándole que ha recibido la información. Si la demora es muy grande, algunos usuarios vuelven a hacer click en el botón de envío pensando que ha habido un fallo.  Si el usuario actúa así, consigue que la información se envíe varias veces. En el caso de las compras online, podría implicar varios pagos en la tarjeta de crédito. JavaScript tiene una forma de desactivar el botón de envío una vez que el usuario haya pulsado en él una vez; utilizando su propiedad *disabled*.  Supongamos que tenemos un formulario cuyo identificador es *formID* y que tiene un botón *submit* cuyo identificador es *submit*. Comenzamos añadiendo la función *submit()* al formulario, y dentro de ella, utilizamos el siguiente código:  *$('#formID').submit(function) {*  *$('#submit').prop('disabled',true);*  *});*  Si la página solo tuviese un formulario, no haría falta usar los identificadores de las etiquetas:  *$('#formID').submit(function) {*  *$('input[type=submit]').prop('disabled',true);*  *});*  Además, podríamos cambiar el texto del botón de envío modificando su valor. Por ejemplo, inicialmente el botón podría mostrar el texto *Submit*(Enviar) pero, cuando se enviase la información del usuario, podría cambiar y mostrar “… enviando información”:  *$('#formID').submit(function) {*  *var subBoton = $(this).find(':submit');*  *subBoton.prop('disabled',true);*  *subBoton.val('Enviando informacion');*  *});*  Todos estos códigos hay que situarlos dentro de la función *$(document).ready().* |

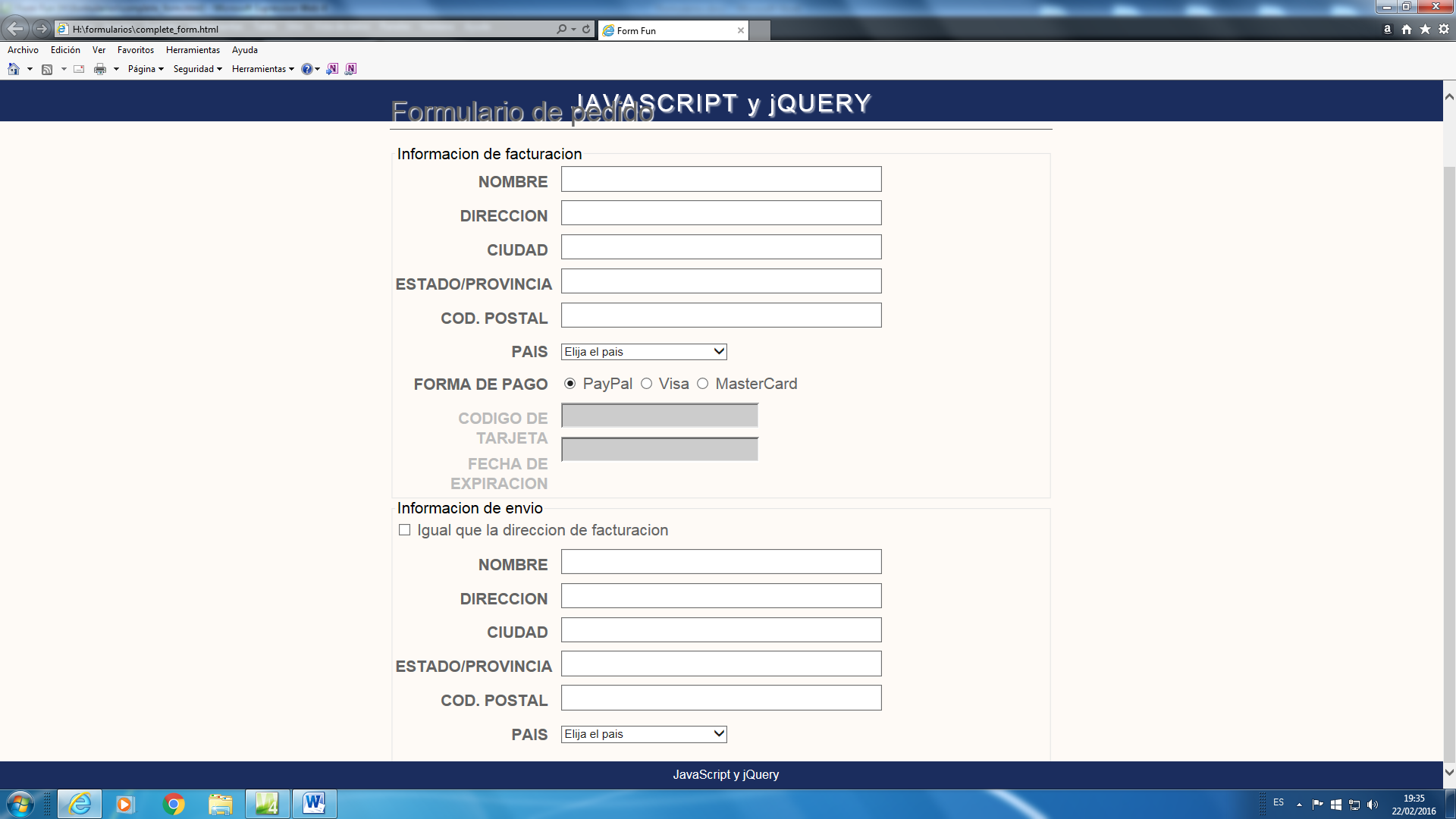
|  |
| --- |
| **Ejemplo 1 – Mejoras en un formulario** |

En este ejemplo, vamos a añadir tres mejoras a un formulario de pedidos que contiene dos áreas: información de envío e información de facturación:

1. Haremos que **el cursor se coloque en el primer campo** cuando se complete la carga de la página.

*$(':text:first').focus();*

1. **Desactivaremos o activaremos los campos** basándonos en las selecciones que realice el usuario. Vamos a desactivar dos campos. Podemos ver que en la parte inferior tenemos tres botones de radio para el modo de pago y otros dos campos que hacen referencia a la tarjeta de crédito y a la fecha de caducidad. Dado que estas dos opciones solo se aplican a los pagos con tarjeta de crédito, los desactivaremos cuando esté seleccionado PayPal.



Este sería el código:

*// Desactivamos la informacion de la tarjeta de credito*

*$('#paypal').click(function() {*

*$('#tarjetaCredito input').prop('disabled', true).css('backgroundColor','#CCC');*

*$('#tarjetaCredito label').css('color','#BBB');*

*}); // fin de click()*

*//Activamos la informacion de la tarjeta de credito*

*$('#visa, #mastercard').click(function() {*

*$('#tarjetaCredito input').prop('disabled', false).css('backgroundColor','');*

*$('#tarjetaCredito label').css('color','');*

*$('#numTarjeta').focus();*

*}); // fin de click()*

1. **Ocultaremos una sección entera del formulario** cuando no sea necesaria.

En muchos formularios existen distintas secciones para introducir datos. En nuestro caso, tenemos un área con direcciones de facturación y otra con direcciones de envío. Podemos incluir un *checkbox* con el texto “*Igual que la dirección de facturación*”. De este modo evitamos que tenga que introducir de nuevo los campos.

El código es:

*//Ocultamos la informacion de direccEnvio*

***$('#ocultarEnvio').click(function() {***

*//Chequeo para saber si el campo esta chequeado*

***if ($(this).prop('checked')) {***

***$('#envio').slideUp('fast');***

***} else {***

***$('#envio').slideDown('fast');***

***}***

***});*** *// fin de click()*

*$('#envio')* referencia a la etiqueta **<div>** que tienen los campos del formulario. Las funciones *slideUp()* y *slideDown()* se usan para ocultar y mostrar dicha etiqueta. También se pueden probar los efectos de JQuery con *fadeIn()* y *fadeOut()*, o incluso crear una animación propia con *animate().*

|  |
| --- |
| **Validación en un formulario** |

Es importante gestionar los campos que son obligatorios en un formulario. Al recibir datos procedentes de un formulario publicado en una web, debemos asegurarnos de que tengan el formato adecuado. A este proceso se le denomina validación. Con JavaScript podemos identificar los errores existentes antes de que el usuario envíe información incorrecta.

Para validar un formulario debemos comprobar todos sus campos antes del envío de la información. El evento **submit** de un formulario se inicia cuando el usuario hace **click** en el botón de envío o al pulsar **Intro** para saltar de un campo a otro. En este momento se suele llevar a cabo la validación.

Es fácil comprobar si un campo de texto contiene información accediendo a la propiedad **value** del formulario utilizando la función **value()** de jQuery comprobando si se encuentra vacío. Las cosas se complican cuando queremos comprobar otros tipos de elementos como casillas de verificación botones de radio, etc.

El tema se complica más cuando los datos so correos electrónicos, códigos postales, fechas, etc.

Por fortuna, existen scripts especializados en la validación de formularios en la web, como puede ser un pluggin de jQuery.

|  |
| --- |
| **Nota** |
| HTML5 incluye funciones nativas para validar formularios que evitan tener que trabajar con JavaScript. Sin embargo, algunas versiones de navegadores pueden no soportarlas. |

|  |
| --- |
| **Plugin de jQuery para la validación de formularios** |

En <http://jqueryvalidation.org> tenemos un pluggin de jQuery para validaciones. Es capaz de comprobar un formulario para verificar que todos los campos obligatorios se han cumplimentado correctamente y de verificar que la información facilitada es correcta.

Controla el tipo de contenido de los campos, el formato de los correos electrónicos, etc.

Para instalar este pluggin, el proceso es:

* Descargar el archivo (librería de jQuery) y adjuntarlo a la página web que contenga el formulario que deseamos validar. Dado que el pluggin utiliza la librería jQuery, tendremos que comenzar por adjuntar la librería de jQuery.
* Descargar y adjuntar el pluggin de validación desde la dirección anterior. Se incluye material adicional, tal como demos, pruebas, etc. En realidad, solo necesitamos el archivo con **extensión .min**. *Nota*: en los ejemplos, ya lo tenemos en la carpeta *jquery\_validate*.
* Añadir las reglas de validación. Debemos especificar los campos que se validarán y los requisitos que deben cumplir.

Añadir los mensajes de error. Esto es opcional. El plugin *validation* incluye mensajes de error predefinidos como “*Por favor, escriba una fecha válida*”, “*Este campo es obligatorio*”, etc. Estos mensajes los podemos personalizar para ser usados en nuestro formulario. Existen dos métodos para añadir mensajes de error (los veremos).

Aplicar la función *validate()* al formulario. El plugin incluye una función que se encarga de gestionar todo lo anterior. Se trata de la función *validate()*. Para hacerlo, usaremos jQuery para seleccionar el formulario y después vincularemos a la función con dicha selección. Por ejemplo, imaginemos que tenemos un formulario cuyo identificador es ***firmar***. Para aplicarle esta función, haremos lo siguiente:

<form action="proceso.php" method="post" name="firmar" id="**firmar**">

La forma más simple de validar sería:

$('firmar').validate();

La función *validate()* acepta diferentes extractos de información que afectarán a su comportamiento. Por ejemplo, aunque podemos especificar las reglas de validación y los mensajes de error en el código HTML del formulario, también podemos especificar las reglas y los errores en el momento de llamar a la función *validate().*

El código JavaScript entero para validar un formulario sería:

<script src="jquery.min.js"></script>

<script src="jquery\_validate/jquery.validate.min.js"></script>

<script>

$(document).ready(function() {

$('#firmar).validate();

}); fin de ready

</script>

|  |
| --- |
| **Validación básica** |

Para utilizar el plugin *Validation* basta con asociarlo con el archivo JavaScript, añadir unas cuantas clases y atributos ***title*** a los elementos del formulario que queramos validar y aplicar el método *validate()* al formulario. Este método es el más habitual.

Pero, si queremos controlar la ubicación en la que aparecerán los mensajes de error en la pantalla, o definir el número máximo/mínimo de caracteres que puede tener un campo de texto, tendremos que usar técnicas más avanzadas.

**Agregar reglas de validación**

La forma más sencilla de validar un campo utilizando este plugin es asignar uno o varios nombres de clases como los que exponemos en la siguiente tabla. Esta tabla incluye métodos capaces de llevar a cabo la mayoría de las tareas de validación de formularios.

|  |  |
| --- | --- |
| **Regla de validación** | **descripción** |
| *Required* | No se enviará el archivo salvo que se rellene, active o desactive el campo. |
| *Date* | La información debe tener el siguiente formato: MM/DD/AAAA |
| *url* | Debe ser una dirección web válida y completa. |
| *Mail* | Debe ser un correo electrónico válido. Esta clase no comprueba que la dirección de correo suministrada sea real. |
| *Number* | Debe ser un número entero positivo entre un rango. |
| *creditcard* | Debe ser un número de tarjeta de crédito con el formato adecuado. |

*Ejemplo*: queremos que un campo de texto sea **obligatorio**.

<input name="nombre" type="text" **class="required"** />

Añadir de este modo una clase no tiene nada que ver con CSS, aunque solemos asignar clases a las etiquetas para que haya una forma de aplicar un formato a dicha etiqueta creando un estilo CSS. En este caso, lo hacemos para aportar al plugin la información que necesita.

**Nota**: la validación de JavaScript permite recopilar información de los usuarios que se hayan saltado un campo por error o que hayan introducido datos incorrectos en un campo. No sirve para evitar que los usuarios introduzcan datos malintencionados. Siempre necesitaremos llevar a cabo una validación en el servidor web.

También es interesante controlar el formato de entrada de datos con un formato determinado. Para conseguir que el campo sea obligatorio y de un tipo determinado de dato, tendremos que añadir dos clases: ***required*** y una de las clases de la tabla anterior.

Por ejemplo, supongamos que tenemos un campo que solicita la fecha de nacimiento al usuario. Esta información es necesaria y, además, debe tener un formato correcto.

*<input name="fechaBaja" type="text" class="required date" />*

Observa que los nombres de las clases aparecen separados por un espacio en blanco. Si excluyésemos la clase **required** y solo utilizásemos uno de los tipos de validación, este campo sería opcional; pero si alguien escribiese algo en el campo, debería tener el formato adecuado.

**Añadir mensajes de error**

El complemento **Validation** proporciona mensajes de error acordes con los problemas de validación que encuentra. Podemos descartar estos mensajes predefinidos y utilizar los nuestros (personalización).

En HTML utilizamos el atributo **title**. Este atributo mejora la usabilidad de los formularios.

*<input name="fechaBaja" type="text" class="required"*

*title="Por favor, introduzca una fecha en formato AAAAMMDD" />*

Sin embargo, con el complemento *Validation* usaremos el atributo ***title*** para suministrar los mensajes de error que aparecerán en el formulario. El plugin escanea todos los campos que se hayan validado en busca del atributo ***title***. Si lo encuentra, usará su valor como mensaje de error.

Si utilizásemos varios métodos de validación, tendríamos que proporcionar un título válido para cada caso. Por ejemplo, si uno de los campos del formulario fuese obligatorio y una fecha, no tendría mucho sentido mostrar únicamente el mensaje predeterminado (el campo es obligatorio) porque aparecería cuando el usuario escribiese una fecha, pero con el formato equivocado.

En este ejemplo mostramos un mensaje que se podría utilizar tanto cuando el usuario dejase el campo en blanco como cuando escriba la fecha con el formato equivalente.

*<input name="fechaBaja" type="text" class="required date"*

*title="Por favor, introduzca una fecha en formato AAAAMMDD" />*

Es fácil añadir reglas y mensajes de error agregando clases y títulos a los campos del formulario. Pero existen situaciones en las que necesitaremos procesos de validación más complejos. *Validation* ofrece un segundo método más avanzado para validar los campos de un formulario. Por ejemplo, podríamos tener diferentes mensajes basándonos en el tipo de error (un mensaje se utilizará cuando el usuario haya dejado un mensaje en blanco, y el otro cuando no introduzca el tipo adecuado de información).

Podríamos utilizar el sistema de validación que hemos visto antes. Pero, *validation* tiene un segundo método más sofisticado que permite implementar un rango más amplio de reglas de validación.

Por ejemplo, utilizaremos este método avanzado cuando queramos asegurarnos de que se escribe un número mínimo de caracteres en un campo. De esta forma, podríamos obligar a que el campo de la contraseña tenga un número mínimo de caracteres.

**Validación avanzada**

*Validation* cuenta con otro método para validar un formulario donde no hace falta modificar el código HTML de los campos. Además, el plugin admite una gran variedad de opciones para controlar el funcionamiento. Para definir dichas opciones, entregamos el literal de un objeto a la función *validate().* En dicho literal colocaremos el objeto de cada de las opciones que deseemos modificar. Por ejemplo, para especificar una regla de validación, entregaremos un objeto que contenga el código de dicha regla:

$('idFormulario').validate({

// aquí van las opciones

}): //fin de validate;

Estas llaves representan el literal del objeto que contendrá los parámetros de las opciones. Podemos combinar en un mismo formulario los métodos de validación básicos descritos antes y los métodos avanzados que vamos a ver a continuación. Con los campos que solo tengan una regla de validación y un mensaje de error, utilizaremos el método sencillo porque es más rápido, y reservaremos el método avanzado para los casos de validación más complicados.

|  |
| --- |
| **Nota** |
| Podemos ver las características del plugin Validation en <http://jqueryvalidation.org/validate> |

**Reglas avanzadas**

El método avanzado para especificar las reglas de validación implica entregar un objeto que contenga los nombres de los campos del formulario y la regla o reglas de validación que les aplicaremos. La estructura básica de este objeto es:

*reglas: {*

*Nombre de campo : 'tipoValidación';*

*}*

El nombre del objeto es rules y dentro especificaremos los campos y los tipos de validación que aplicaremos a cada uno de ellos. Luego, entregaremos el objeto a la función *validate()*; por ejemplo, en un formulario en el que deseemos que el campo nombre sea obligatorio, tendríamos que aplicar la función *validate()* al formulario y a continuación entregarle el objeto rules.

*$('registrar').validate({*

*rules: {*

*nombre : 'required',*

*}*

*}); // fin de validate()*

Esta regla indica que el campo cuyo atributo ***name*** es nombre es obligatorio.

Para crear varias reglas de validación a un mismo campo, tendremos que crear otro objeto. Por ejemplo, podríamos añadir la regla que diga que el campo de correo electrónico especifique el formato que debe tener:

$('registrar').validate({

rules: {

nombre : 'required',

**email : {**

**required : true,**

**email : true**

**}**

}

}); // fin de validate()

La regla ***email : true*** garantiza que en dicho campo se introducirá un email correcto.

También podemos utilizar los tipos de validación que expusimos en la tabla anterior. Supongamos que añadimos un campo denominado ***fechaNacimiento*** al formulario de este ejemplo. Si queremos asegurarnos de que se introducirá una fecha correcta en el campo, podríamos ampliar la regla.

$('registrar').validate({

rules: {

nombre : 'required',

email : {

required : true,

email : true

}

**fechaNacimiento : 'date'**

}

}); // fin de validate()

Si además deseamos que la fecha sea obligatoria,

$('registrar').validate({

rules: {

nombre : 'required',

email : {

required : true,

email : true

}

**fechaNacimiento :**

{

date : true,

required: true

}

}

}); // fin de validate()

Veamos algunas reglas que nos pueden resultar de mucha utilidad:

**Minlength**: el campo debe contener, por lo menos, el número especificado de caracteres. Utilizado típicamente en la producción de contraseñas.

**Maxlength**: el campo debe contener como máximo el número de caracteres especificados.

**Rangelength**: especifica el número mínimo y el número máximo de caracteres:

*rangelength[5,10]*

**Min**: obliga a que el campo contenga un número mayor o igual al especificado.

**Max**: obliga a que el campo contenga un número menor o igual al especificado.

**Range**: combina las reglas min y max:

range[100,1000]

**equalTo**: obliga a que el contenido del campo coincida con otro campo. Por ejemplo, en un formulario de registro se puede obligar a repetir la contraseña suministrada por el usuario. Por ejemplo, imaginemos que el identificador del campo de password es ***contrasena***. Si quisiéramos comprobar que los datos introducidos en el campo de verificación coinciden con los introducidos en el campo contraseña, escribiríamos:

*equalTo : 'contrasena';*

Podemos combinar estas reglas de validación avanzadas. Basta con aplicarlas en los campos una a una. Imaginemos que tenemos un formulario que incluye dos campos, una para la contraseña y otro para la confirmación de la contraseña.

<input name="contrasena" type="password" id="contrasena" />

<input name="confirmar" type="password" id="confirmar" />

Ambos campos son obligatorios, las contraseñas deben coincidir y deben tener un número de caracteres entre 6 y 16 inclusive. Vamos a suponer que el identificador del formulario es ***registrar***.

$('registrar').validate({

rules: {

contrasena : {

required : true,

rangelength : [6,16];

}

confirmar :

{

equalTo : '#contrasena'

}

} // fin de las reglas

}); // fin de validate()

|  |
| --- |
| **Nota** |
| *Validation* tiene otro complemento denominado ***additional-method.js***, en el que se incluyen más reglas de validación, tales como, número mínimo de palabras, números de cuentas de determinados bancos, etc. Si encontrásemos alguna regla que nos interés podríamos hacer una copia del archivo y darle otro nombre. A continuación, eliminamos las reglas que no nos interesen y después lo referenciamos en las páginas que nos interesen. |

**Mensajes de error avanzados**

Lo que deseamos ahora es generar mensajes de error independientes para cada tipo de error de validación. Imaginemos que un campo debe tener valores numéricos y que debe ser obligatorio. Si queremos tener dos mensajes de error independientes, no lo podremos hacer con el *atributo* ***title***. Debemos entregar un objeto JavaScript a la función *validate()* que contenga los diferentes mensajes de error que queramos mostrar en la pantalla.

El proceso es similar al de la creación de reglas avanzadas que vimos antes.

*mensajes : {*

*Nombrecampo {*

*tipoMetodo : 'mensaje de error'*

*}*

*}*

Vamos a generar mensajes de error para el ejemplo del apartado anterior:

$('registrar').validate({

rules: {

contrasena : {

required : true,

rangelength : [6,16];

}

confirmar :

{

equalTo : '#contrasena'

}

} // fin de las reglas

**mensajes: {**

**contrasena : {**

**required : 'Por favor, introduzca el password',**

**rangelength : 'Su password debe contener entre 6 y 16 caracteres';**

**}**

**confirmar :**

**{**

**equalTo : 'Los dos password deben ser idénticos'**

**}**

**} // fin de los mensajes**

}); // fin de validate()

**Validar con el servidor**

*Validate* cuenta con un método de validación avanzado denominado ***remote***, que permite contrastar con el servidor. Este método permite entregar el nombre del campo y el valor que haya escrito el usuario al servidor. Para esta consulta, utiliza un lenguaje de servidor como PHO, Ruby, .NET, Java o Node.js. el servidor toma esta información y hace las comprobaciones necesarias. Después de hacerlo, responderá al formulario con un valor true (si pasa la validación) o false (si no la pasa).

Imaginemos que tenemos un campo obligatorio denominado **nombreUsuario** y queremos comprobar que existe en el servidor:

*contrasena : {*

*required : true,*

*remote : 'chequearNombreUsuario.php';*

*}*

El método remoto toma una cadena que contiene la ruta de la página actual y se la entrega al servidor web. Cuando el plugin de validación intente validar este campo, enviará el nombre del campo y sus datos a la página de servidor **chequearNombreUsuario.php**, que determinará si dicho nombre está disponible. Si es así, la página .**php** devolverá true y en caso contrario devolverá false y no validará el campo. Todo este proceso lo lleva a cabo gracias a AJAX (lo veremos).

Podemos información de este tema en <https://bassistance.de/2008/01/30/jquery-validation-plugin-overview/>