

# O plugin Form en jQuery

Ata este momento no tutorial centrámosnos nas capacidades do núcleo jQuery e da biblioteca; pero todo isto é soamente a punta do iceberg!

A colección de plugins para jQuery dispoñible ata estes momentos é inmensa e impresionante. Os creadores do núcleo jQuery coidadosamente escolleron as funcións necesitadas pola maioría de desenvolvedores de páxinas web e crearon un marco para que outros puideran construír plugins facilmente.

Isto mantén o núcleo de jQuery o máis pequeno posible, de xeito que si queremos engadir máis funcionalidade teremos que empregar plugins adicionais.

Para consultar a lista de plugins dispoñibles teremos que ir a <http://plugins.jquery.com/>

A lista de plugins máis populares en [http://plugins.jquery.com/most\\_popular](http://plugins.jquery.com/most_popular)

Aquí imos a ver algúns dos plugins máis útiles para a nosa programación web. Imos comezar co plugin Form.

## Sumario

- 1 O plugin Form para jQuery
  - ◆ 1.1 Obtendo valores dos controles do formulario
  - ◆ 1.2 Limpando e reseteando os controles
  - ◆ 1.3 Enviando formularios con Ajax
  - ◆ 1.4 Enviando arquivos

## O plugin Form para jQuery

Traballar con formularios en moitos casos soe ser un traballo tedioso e moi repetitivo. Cada tipo de control ten as súas particularidades e o envío de formularios a miúdo pode incluír a validación de campos, etc..

O núcleo de jQuery ten algúns métodos que nos poden axudar, pero isto non é suficiente. O propósito do plugin Form é axudarnos a cubrir estas carencias e proporcionarnos máis poder nos controles dos formularios.

Este plugin pódese atopar en <http://jquery.com/plugins/project/form> e reside no ficheiro `jquery.form.js`.

Este plugin proporciona funcionalidades nas seguintes áreas:

? Obtención dos valores dos controles do formulario

? Limpando e reseteando controles do formulario

? Envío de formularios (incluíndo envío de arquivos) via Ajax

### Obtendo valores dos controles do formulario

O plugin Form proporciona dous xeitos de obter os valores dos controles do formulario: como un array de valores ou como un string serializado. Dispoñemos de tres métodos para obter esos valores: **fieldValue()**, **formSerialize()** e **fieldSerialize()**.

Dos tres métodos anteriores o máis empregado é **formSerialize()**. Este método accede a todos os valores do formulario e devolve unha cadea no formato empregado polo método GET para enviar os datos. Por exemplo si temos un formulario con id `form1` con tres campos de texto: nome, apelidos e dni, se ponemos o seguinte código:

```
$('#form1').formSerialize()

// Devolverá a seguinte cadea: nome=rafa&apelidos=veiga%20cid&dni=12345678T
```

### Limpando e reseteando os controles

O plugin Form proporciona dous comandos que afectan ós valores dos controles. O comando **clearForm()** limpa todos os campos, mentras que o

comando **resetForm()** resetea os campos.

¿Cal é a diferenza entón? O método **clearForm()** realiza o seguinte:

- Os campos de texto, password e textarea son axustados a valores en branco.
- Os campos de tipo select son deseleccionados.
- Os campos check box e radio son desmarcados.

Cando chamamos o método **resetForm()**, o método **reset()** é invocado. Isto provocará que os campos tomen os valores que teñen asignados por defecto na definición do formulario (os value que teñamos no HTML)

## Enviando formularios con Ajax

O plugin Form introduce dos novos comandos para enviar formularios mediante Ajax: un deles que inicia unha solicitude Ajax baixo o control dun script, enviando os datos ó destino no formato solicitado, e o outro comando que controla o formulario de xeito que calquera envío é reconvertido automaticamente a unha solicitude Ajax.

Para enviar un formulario mediante este plugin faise dun xeito moi sinxelo. Simplemente teremos que empregar o método **ajaxSubmit()**.

O método **ajaxSubmit()** ten varias opcións, aquí pomos un resumen das mesmas:

| Name                      | Description   |
|---------------------------|---|
| <code>url</code>          | (String) The URL to which the Ajax request will be submitted. If omitted, the URL will be taken from the <code>action</code> attribute of the target form.  |
| <code>type</code>         | (String) The HTTP method to use when submitting the request, such as GET or POST. If omitted, the value specified by the target form's <code>method</code> attribute is used. If not specified and the form has no <code>method</code> attribute, GET is used.  |
| <code>dataType</code>     | <p>(String) The expected data type of the response, which determines how the response body will be post-processed. If specified, it must be one of the following:</p> <ul style="list-style-type: none"> <li>■ <code>xml</code>—Treated as XML data. Any success callback will be passed the <code>responseXML</code> document.</li> <li>■ <code>json</code>—Treated as a JSON construct. The JSON is evaluated, and the result is passed to any success callback.</li> <li>■ <code>script</code>—Treated as JavaScript. The script will be evaluated in the global context.</li> </ul> <p>If omitted, no post-processing of the data (except as specified by other options such as <code>target</code>) takes place.</p>   |
| <code>target</code>       | (String Object Element) Specifies a DOM element or elements to receive the response body as content. This can be a string depicting a jQuery selector, a jQuery wrapper containing the target elements, or a direct element reference. If omitted, no element receives the response body.   |
| <code>beforeSubmit</code> | <p>(Function) Specifies a callback function invoked prior to initiating the Ajax request. This callback is useful for performing any pre-processing operations including the validation of form data. If this callback returns the value <code>false</code>, the form submission is cancelled.</p> <p>This callback is passed the following three parameters:</p> <ul style="list-style-type: none"> <li>■ An array of the data values passed to the request as parameters. This is an array of objects; each contains two properties, <code>name</code> and <code>value</code>, containing the name and value of a request parameter.</li> <li>■ The jQuery matched set that the command was applied to.</li> <li>■ The <code>options</code> object that was passed to the command.</li> </ul> <p>If omitted, no pre-processing callback is invoked.</p> |
| <code>success</code>      | <p>(Function) Specifies a callback invoked after the request has completed and returned as response with successful status.</p> <p>This callback is passed the following three parameters:</p> <ul style="list-style-type: none"> <li>■ The response body as interpreted according to the <code>dataType</code> option.</li> <li>■ A string containing success.</li> <li>■ The jQuery matched set that the command was applied to.</li> </ul> <p>If omitted, no success callback is invoked.</p> <p>If this is the only option to be specified, this function can be passed directly to the command in place of the <code>options</code> hash.</p> <p>Note that no provisions have been made for a callback upon error conditions.</p>  |
| <code>clearForm</code>    | (Boolean) If specified and <code>true</code> , the form is cleared after a successful submission. See <code>clearForm()</code> for semantics.   |
| <code>resetForm</code>    | (Boolean) If specified and <code>true</code> , the form is reset after a successful submission. See <code>resetForm()</code> for semantics.   |
| <code>semantic</code>     | (Boolean) If specified and <code>true</code> , the form parameters are arranged in semantic order. The only difference this makes is in the location of the parameters submitted for input element of type <code>image</code> when the form is submitted by clicking that element. Because there's overhead associated with this processing, this option should be enabled only if parameter order is important to the server-side processing and image input elements are used in the form.  |
| other options             | Any options that are available for the core jQuery <code>\$.ajax()</code> function, as described in table 8.2, can be specified and will pass through to the lower-level call.  |

### Exemplos de envío de datos mediante ajaxSubmit():

```
// Si queremos enviar o formulario sin xestionar ningunha resposta faremos:
$('#idformulario').ajaxSubmit();

// Si queremos cargar a resposta nun elemento ou elementos de destino faremos:
$('#idformulario').ajaxSubmit( { target: 'elementosdestino' } );

// Si queremos xestionar a resposta cunha función
$('#idformulario').ajaxSubmit(function(resposta){
    /* facemos algo coa resposta */
});
```

Sin embargo hai outro xeito de enviar automaticamente (sen usar un script) o formulario mediante Ajax. Para elo empregaremos a función ajaxForm(), que enlazaremos ó noso formulario.

Vexamos un exemplo:

```
<form id="Formulario" action="comentarios.php" method="post">
  Name: <input type="text" name="name" />
  Comentarios: <textarea name="comment"></textarea>
  <input type="submit" value="Enviar Comentario" />
</form>
<head>
  <script src="ruta/jquery.js"></script>
  <script src="ruta/form.js"></script>

  <script>
    // Agardamos a que o DOM sexa cargado
    $(document).ready(function() {
      // enlazamos o noso 'formulario' e proporcionamos unha función de retorno
      $('#Formulario').ajaxForm(function() {
        alert("Grazas polo comentario!");
      });
    });
  </script>
</head>
```

Cando o formulario é enviado o nome e comentarios serán enviados á páxina comentarios.php. Si o servidor devolve un estado de correcto, entón o usuario recibirá unha alerta de "Grazas polo comentario".

[Ver máis exemplos de ajaxSubmit\(\) e ajaxForm\(\)](#)

## Enviando arquivos

Unha das características do plugin Form é que automaticamente detecta aqueles formularios que necesitan enviar arquivos.

Debido a que non é posible enviar arquivos usando o obxecto XMLHttpRequest (XHR), o plugin Form emprega un iframe oculto creado dinamicamente para axudar nesa tarefa. Esta é unha técnica moi común, pero ten limitacións. O iframe é usado como destino do formulario, o que implica que a resposta do servidor será escrita no iframe. Isto funciona ben si o tipo de resposta é HTML ou XML, pero non funciona correctamente si a resposta é un script ou JSON, xa que ambos conteñen caracteres especiais que necesitan ser representados empregando o seu equivalente HTML.

Para solucionar o caso anterior dos script e JSON, o plugin Form permite que estas respostas sexan representadas nun textarea, e é recomendable que o fagamos así cando fagamos envío de arquivos nos formularios. Algo a ter en conta é que si non enviamos arquivos ó servidor a solicitude empregará unha petición XHR para enviar o formulario (non un iframe). Isto fai que dependendo do envío o código do servidor ten que saber cando usar un textarea ou non. Para evitar isto podemos usar a opción de iframe no plugin para forzar que sempre empregue un iframe co cal teremos sempre a resposta nun textarea.

[Exemplo de file upload.](#)

--Veiga ([discusión](#)) 23:10 26 ene 2015 (CET)