Home : Articles : Removing, Replacing and Moving Elements in jQuery

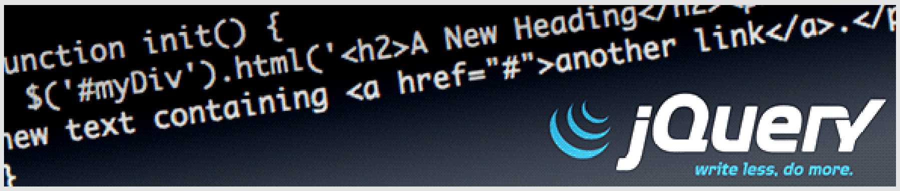
Removing, Replacing and Moving Elements in jQuery

Tutorial by Matt Doyle | Level: Beginner | Published on 8 June 2010

Categories: Web Development > JavaScript > jQuery

Tweet

Learn how to use jQuery to easily remove elements from the page, replace elements, and move elements around.


unction init() {
$('#myDiv').html('<h2>A New Heading</h2>
new text containing <a href="#">another link</a>.</p
}
jQuery
write less, do more.

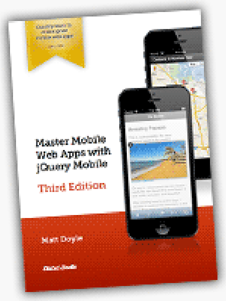
In Adding Elements to the Page in jQuery, you explored a number of ways to add new HTML elements — such as paragraphs and images — to a page. In this tutorial you learn how to manipulate existing elements in the page, including:

- Removing elements from the page using the `empty()`, `remove()`, `detach()` and `unwrap()` methods
- Replacing elements with new elements by using the `replaceWith()` and `replaceAll()` methods, and
- How to move an element from one parent element in the page to another.

Once you've read this tutorial, you'll have mastered all the jQuery techniques you need to manipulate elements in the DOM.

Removing elements from the page

Removing everything inside an element: `empty()`



Current forum topics
- How to Add Image Uploading to Your CMS
- logo design
- Understanding Permissions
- CSS – Overlay color
- Creating a JavaScript Clock

Got a question about making a website? Ask it in the forums — we'd love to help you. All questions answered!

Popular articles
- jQuery Mobile: What Can It Do for You?
- JavaScript Tabs – Create Tabbed Web Pages Easily
- How to Make a Slick Ajax Contact Form with jQuery and PHP
- Making CSS Rollover Buttons
- Object-Oriented PHP for Absolute Beginners

Never made a website before? Read How to Make a Website.

The `empty()` method is the simplest way to remove content from the page. When you call `empty()` on a jQuery object, all the content is removed from the set of matched element(s) in the jQuery object.

In other words, `empty()` removes all child elements and other child nodes (such as text nodes) from each element in the matched set, leaving the element empty.

Here's an example that empties 2 `div` elements:

<Code />

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">

$( init );

function init() {

  // Delete the contents of #myDiv1 and #myDiv2
  $('.emptyMe').empty();
}

</script>

</head>
<body>

  <div class="emptyMe" id="myDiv1">
    <p>A paragraph of text</p>
  </div>

  <div class="emptyMe" id="myDiv2">
    <p>Another paragraph of text</p>
    A text node on its own
  </div>

</body>
</html>
```

After running the above code, the page content changes to this:

<Code />

```
<body>
  <div class="emptyMe" id="myDiv1" />
  <div class="emptyMe" id="myDiv2" />
</body>
```

Removing an element entirely: **`remove()`**

Whereas `empty()` removes everything inside an element, `remove()` also removes the element itself. Here's an example:

<Code />

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">

$( init );

function init() {

  // Delete #myDiv1 and #myDiv2 entirely
  $('.removeMe').remove();
}

</script>

</head>
<body>

  <div class="removeMe" id="myDiv1">
    <p>A paragraph of text</p>
  </div>

  <div class="removeMe" id="myDiv2">
    <p>Another paragraph of text</p>
    A text node on its own
  </div>

</body>
</html>
```

After running the above code, both `div` elements are removed entirely from the page:

<Code />

```
<body>
</body>
```

You can pass an optional selector string to `remove()`. If you do this, the elements to remove are filtered by the selector. Here's an example:

<Code />

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">

$( init );

function init() {

  // Delete #myDiv2 only
  $('.removeMe').remove(':contains("Another paragraph")');
}

</script>

</head>
<body>

  <div class="removeMe" id="myDiv1">
    <p>A paragraph of text</p>
  </div>

  <div class="removeMe" id="myDiv2">
    <p>Another paragraph of text</p>
    A text node on its own
  </div>

</body>
</html>
```

In the above example, only the `div` that has a class of `removeMe` and contains the text "Another paragraph" is removed, leaving the following contents in the page:

<Code />

```
<body>

  <div class="removeMe" id="myDiv1">
    <p>A paragraph of text</p>
  </div>

</body>
```

Removing an element without destroying its data: `detach()`

`remove()` returns a jQuery object containing the removed elements. In theory, this makes it easy to remove some elements from one place in the page, and then later reattach them elsewhere.

However, in order to save resources and avoid potential problems with memory leaks, `remove()` deletes all jQuery data and events associated with the removed elements. For example, if you've assigned a jQuery `click` event to an element, and you then remove the element from the page using `remove()`, the `click` event is removed from the element. This can be a problem if you later want to add the element back to the page and preserve all its functionality.

This is where the `detach()` method — new in jQuery 1.4 — comes in handy. It behaves exactly like `remove()`, except that it doesn't delete the jQuery data and events associated with the removed elements. This means you can later reattach the removed elements while preserving their jQuery metadata.

Here's an example. The following script assigns a jQuery `click` event to each of 2 paragraphs in the page. Both event handlers simply toggle a `"red"` CSS class on the paragraph to toggle the paragraph's colour to red or black each time it's clicked.

Then the script removes the first paragraph from the page using `remove()` and stores the jQuery object containing the paragraph in a `myDiv1Para` variable. It then reattaches the paragraph to its parent `div` using `appendTo()`.

It then does much the same thing with the second paragraph, but uses `detach()` instead of `remove()`.

<Code />

```
$( init );

function init() {

  // Assign a click event to each div's paragraph
  $("#myDiv1>p").click( function() { $(this).toggleClass("red"); } );
  $("#myDiv2>p").click( function() { $(this).toggleClass("red"); } );

  // Remove and reattach #myDiv1's paragraph
  var myDiv1Para = $('#myDiv1>p').remove();
  myDiv1Para.appendTo('#myDiv1');

  // Detach and reattach #myDiv2's paragraph
  var myDiv2Para = $('#myDiv2>p').detach();
  myDiv2Para.appendTo('#myDiv2');
}

</script>

</head>
<body>

  <div id="myDiv1">
    <p>A paragraph of text</p>
  </div>

  <div id="myDiv2">
    <p>Another paragraph of text</p>
  </div>

</body>
</html>
```

After running this script, the first paragraph loses its `click` event handler, while the second paragraph retains it. You can try this yourself by opening the page in a browser. You'll see that you can click the second paragraph to make it red, but nothing happens when you click the first paragraph.

This is because the call to `remove()` also deleted the first paragraph's `click` event, while the call to `detach()` preserved the second paragraph's `click` event.

Info

You'll see more ways to move elements around at the end of this tutorial, and I'll cover jQuery events in detail in my next tutorial.

Removing an element's parent: `unwrap()`

The `unwrap()` function is, as you might imagine, the opposite of `wrap()`. It removes the parent of an element (or the parents of a set of elements) from the DOM. The element — and its siblings, if any — then take the place of the element's parent in the DOM.

The following example unwraps a paragraph inside a `div` — in other words, it replaces the `div` with its contents:

<Code />

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">

$( init );

function init() {

  // Remove the #myDiv element but leave its contents
  $('#myPara').unwrap();
}

</script>

</head>
<body>

  <div id="myDiv">
    <p id="myPara">A paragraph of text</p>
    <p>Another paragraph of text</p>
  </div>

</body>
</html>
```

After running the above code, the page's content changes to the following. Notice how both paragraphs — the target paragraph and its sibling — have been "unwrapped":

<Code />

```
<body>

  <p id="myPara">A paragraph of text</p>
  <p>Another paragraph of text</p>

</body>
```

Replacing elements

Replacing an element with new content: `replaceWith()`

`replaceWith()` lets you replace an element, or set of elements, with new content. Its syntax is very similar to `prepend()` and similar methods. You can pass the replacement content in any of the following forms:

- An element object that you've created using a JavaScript DOM function such as `document.getElementById()` or `document.createElement()`
- A string of HTML representing the replacement content
- A jQuery object containing the element(s) to use for the replacement
- A callback function that should return the replacement HTML

Here's an example that shows `replaceWith()` in action. It replaces 1 paragraph with a new string of HTML, a second paragraph with an element object, and a third paragraph with the results of a function that returns the current time:

<Code />

```
    var currentTime = new Date();
    var currentHours = currentTime.getHours ( );
    var currentMinutes = currentTime.getMinutes ( );
    var currentSeconds = currentTime.getSeconds ( );

    // Pad the minutes and seconds with leading zeros, if required
    currentMinutes = ( currentMinutes < 10 ? "0" : "" ) + currentMinutes
    currentSeconds = ( currentSeconds < 10 ? "0" : "" ) + currentSeconds

    return ( "<p>The current time is: " + currentHours + ":" + currentMi
  }
}

</script>

</head>
<body>

  <div id="myDiv1">
    <p>A paragraph of text</p>
  </div>

  <div id="myDiv2">
    <p>A paragraph of text</p>
  </div>

  <div id="myDiv3">
    <p>A paragraph of text</p>
  </div>

</body>
</html>
```

After running the above code, the page's content is replaced with the following:

<Code />

```
<body>

  <div id="myDiv1">
    <p>A new paragraph of text</p>
  </div>

  <div id="myDiv2">
    <hr />
  </div>

  <div id="myDiv3">
    <p>The current time is: 13:52:17</p>
  </div>

</body>
```

`replaceAll()` : An alternative to `replaceWith()`

`replaceAll()` does the same job as `replaceWith()` , except that rather than passing in the replacement content, you pass in the elements that you want to replace. In other words, it's the mirror image of `replaceWith()` , much as `prependTo()` is the mirror image of `prepend()` .

For example, the following 2 lines of code both do essentially the same thing:

<Code />

```
$('#myDiv').replaceWith( "<p>Here's some new text</p>" );
$("<p>Here's some new text</p>").replaceAll( '#myDiv' );
```

Moving elements around

You've now looked at adding elements to the page, as well as removing and replacing elements. There's one more piece to the puzzle: How do you move elements around the DOM tree? For example, you might have a paragraph inside one `div` element, and you want to move the paragraph so that it's inside a different `div` .

While there are no specific jQuery methods for moving elements around the DOM tree, in fact it's very easy to do. All you have to do is select the element(s) you want to move, then call an "adding" method such as `append()` , `appendTo()` or `prepend()` to add the selected elements to another parent element. jQuery automatically realises that the element(s) to add already exist in the page, and it moves the element(s) to the new parent.

Here's an example to make this process clear. This example moves the paragraph from the first div to the second:

<Code />

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">

$( init );

function init() {

 // Move the paragraph from #myDiv1 to #myDiv2
  $('#myDiv2').append( $('#myDiv1>p') );
}

</script>

</head>
<body>

  <div id="myDiv1">
    <p>A paragraph of text</p>
  </div>

  <div id="myDiv2">
  </div>

</body>
</html>
```

After running the above code, the page's content changes to this:

<Code />

```
  <div id="myDiv1">
  </div>

  <div id="myDiv2">
    <p>A paragraph of text</p>
  </div>
```

Here are some other ways to achieve the same thing:

<Code />

```
// Move the paragraph from #myDiv1 to #myDiv2
$('#myDiv1>p').appendTo( $('#myDiv2') );

// Move the paragraph from #myDiv1 to #myDiv2
var para = $('#myDiv1>p');
para.prependTo( '#myDiv2' );

// Move the paragraph from #myDiv1 to #myDiv2
// by explicitly detaching it then adding it again
$('#myDiv1>p').detach().prependTo('#myDiv2');
```

Info

The 3rd technique above makes use of a very handy jQuery feature called method chaining. Since most jQuery methods return a jQuery object, you can then call another method on the returned object. This in turn returns another jQuery object, and so on.

So in the above example, a jQuery object is created containing the paragraph to remove, and `detach()` is called, returning another jQuery object containing the removed paragraph. Finally, `prependTo()` is called on this second jQuery object to add the removed paragraph to its new parent element.

What happens if you attempt to move some content to more than 1 parent element at the same time? When you do this, jQuery first removes the content from its old parent, then clones the content as many times as necessary and adds a clone to each parent element. For example:

<Code />

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">

$( init );

function init() {

 // Move the paragraph from #myDiv1 to #myDiv2 and #myDiv3
  $('#myDiv2, #myDiv3').append( $('#myDiv1>p') );
}

</script>

</head>
<body>

  <div id="myDiv1">
    <p>A paragraph of text</p>
  </div>

  <div id="myDiv2">
  </div>

  <div id="myDiv3">
  </div>

</body>
</html>
```

After running the above code, the page's content looks like this:

<Code />

```
<body>

  <div id="myDiv1">
  </div>

  <div id="myDiv2">
    <p>A paragraph of text</p>
  </div>

  <div id="myDiv3">
    <p>A paragraph of text</p>
  </div>

</body>
```

Summary

If you've been following these jQuery tutorials so far, you now have a solid understanding of how to manipulate page content using jQuery. You've looked at selecting elements; working with element content, attributes and classes; adding new elements to the page; and in this tutorial, removing, replacing and moving elements. These are all fundamental concepts that you'll use when building jQuery-enhanced websites.

In the next few tutorials in this series, the fun will really begin as I show how to use jQuery events and effects to start adding rich interactivity and eye-popping effects to your pages. Until then — happy coding!

Related articles

- jQuery Mobile 1.4: Cleaner, Faster and More Powerful
- jQuery Mobile 1.3: What's New?
- jQuery Mobile 1.1: Smoother, Faster, Nicer
- Ajax with jQuery: A Beginner's Guide
- jQuery Mobile Masterclass: Build a Simple, Attractive Twitter App for iPhone

Responses to this article

5 responses (oldest first):

gokul357 28-Feb-11 01:56

Help Me.....?

<Code />

```
<ul>
<li><a href="#">Home</a></li>
<li><a href="#">Sign in</a>or<a href="#">New User</a>
</li>
</ul>
```

How to remove the word 'or' in the second li?
or How to achieve the following?

<Code />

```
<li><a href="#">Sign in</a>
<span>or</span>
<a href="#">New User</a>
```

[Edited by gokul357 on 28-Feb-11 02:01]

matt 02-Mar-11 01:59

@gokul357: This topic is for discussing the article. Please post your question in a new topic:

http://www.elated.com/forums/authoring-and-programming/topic/new/

brianna 07-Apr-11 19:04

Working on the moving principal, using prepend instead of append, I'm wondering how to keep cloning from happening. Is that an easy thing to implement?

For example, on a page with multiple DIV elements of the same ID/class, how do I tell a child element to move to another child but only relative to its parent? Right now I have:

<Code />

```
$('.entry-body').prepend($('.entry-comments'));
```

But on a page with 4 instances of .entry-body, and the same number of .entry-comments, the above prepend code ends up populating 4 times in each .entry-body instance.

You mentioned that "jQuery first removes the content from its old parent, then clones the content as many times as necessary and adds a clone to each parent element" and that's exactly what I'd prefer not to happen. Thoughts?

matt 11-Apr-11 00:27

@brianna: You need to find a way to specifically select the single .entry-body element you're interested in, rather than selecting all 4 elements with the .entry-body class.

I can't give exact information without seeing your page. However I think you'll find jQuery's parent() method useful:

http://api.jquery.com/parent/

Cheers,
Matt

varg242 28-Feb-12 13:03

Thank you very much!

Post a response

Want to add a comment, or ask a question about this article? Post a response.

To post responses you need to be a member. Not a member yet? Signing up is free, easy and only takes a minute. Sign up now.

Top of Page

Home
Articles
Forums

Contact Us
Spamwars
RSS
Twitter
Facebook

Elated
Love Your Site