5 jQuery.each() Function Examples

Florian Rappl
January 21, 2016 +67

This is quite an extensive overview of the jQuery `each()` function. This function is **one of jQuery's most important and most used functions**. In this article we'll find out why and look into its details to see how you can use it.

What is jQuery `.each()`

More from this author
- Introduction to Functional Reactive Programming with RxJS
- Ajax/jQuery.getJSON Simple Example

jQuery's `each()` function is used to loop through each element of the target jQuery object. In case you're not really experienced in jQuery, I remind you that a jQuery object is an object that contains one or more DOM elements, and exposes all jQuery functions. It's very useful for multi-element DOM manipulation, looping arbitrary arrays, and object properties. In addition to this function, jQuery provides a helper function with the same name that can be called without having previously selected or created DOM elements. Let's find out more in the next sections.

jQuery's `.each()` Syntax

Let's see the different modes in action.

The following example selects every `div` on the web page and outputs the index and the ID of each of them. A possible output is: "div0:header", "div1:body", "div2:footer". This version uses jQuery's `each()` function as opposed to the utility function.

```
// DOM ELEMENTS
$('div').each(function (index, value) {
  console.log('div' + index + ':' + $(this).attr('id'));
});
```

The next example shows the use of the utility function. In this case the object to loop over is given as the first argument. In this example I show how to loop over an array:

```
// ARRAYS
var arr = [
   'one',
   'two',
   'three',
   'four',
   'five'
];
$.each(arr, function (index, value) {
  console.log(value);

  // Will stop running after "three"
  return (value !== 'three');
});
// Outputs: one two three
```

In the last example I want to present loops through the properties of an object:

```
// OBJECTS
var obj = {
   one: 1,
   two: 2,
   three: 3,
   four: 4,
   five: 5
};
$.each(obj, function (index, value) {
  console.log(value);
});
// Outputs: 1 2 3 4 5
```

It all boils down to provide a proper callback. The callback's context, `this`, will be equal to the second argument, which is the current value. However, since the context will always be an object, primitive values have to be wrapped. Therefore, strict equality between the value and the context may not be given. The first argument is the current index, which is either a number (for arrays) or string (for objects).

1. Basic `jQuery.each()` Function Example

Let's see how the `each()` function helps us in conjunction with a jQuery object. The first example selects all the `a` elements in the page and outputs their `href` attribute.

```
$('a').each(function (index, value){
  console.log($(this).attr('href'));
});
```

The second example outputs every external `href` on the web page (assuming the HTTP protocol only):

```
$('a').each(function (index, value){
  var link = $(this).attr('href');

  if (link.indexOf('http://') === 0) {
    console.log(link);
  }
});
```

Let's say that in the page we had the following links:

```
<a href="http://www.jquery4u.com">JQUERY4U</a>
<a href="http://www.phpscripts4u.com">PHP4U</a>
<a href="http://www.blogoola.com">BLOGOOLA</a>
```

The second example would output:

```
http://jquery4u.com
http://www.phpscripts4u.com
http://www.blogoola.com
```

We should note that DOM elements from a jQuery object need to be wrapped again when used inside a jQuery `each()`. The reason is that jQuery is in fact just a wrapper around an array of DOM elements. By using jQuery `each()` this array is iterated in the same way as an ordinary array would be. Therefore, we don't get wrapped elements out of the box.

2. `jQuery.each()` Array Example

Let's have another look at how an ordinary array can be handled.

```
var numbers = [1, 2, 3, 4, 5, 6];
$.each(numbers , function (index, value){
  console.log(index + ':' + value);
});
```

This snippet outputs: `0:1`, `1:2`, `2:3`, `3:4`, `4:5`, and `5:6`.

Nothing special here. An array features numeric indices, hence we obtain numbers starting from *0* and going up to *N – 1*, where *N* is the number of elements in the array.

3. `jQuery.each()` JSON Example

We may have more complicated data structures, such as arrays in arrays, objects in objects, arrays in objects, or objects in arrays. Let's see how `each()` can help us in such scenarios.

```
var json = [
 { 'red': '#f00' },
 { 'green': '#0f0' },
 { 'blue': '#00f' }
];

$.each(json, function () {
   $.each(this, function (name, value) {
      console.log(name + '=' + value);
   });
});
```

This example outputs `red=#f00`, `green=#0f0`, `blue=#00f`.

We handle the nested structure with a nested call to `each()`. The outer call handles the array of the variable `JSON`, the inner call handles the objects. In this example each object has only one key, however, in general any number could be attacked with the provided code.

4. `jQuery.each()` Class Example

This example shows how to loop through each element with assigned class `productDescription` given in the HTML below.

```
<div class="productDescription">Red</div>
<div>Pink</div>
<div class="productDescription">Orange</div>
<div class="generalDescription">Teal</div>
<div class="productDescription">Green</div>
```

We use the `each()` helper instead of the `each()` method on the selector.

```
$.each($('.productDescription'), function (index, value) {
  console.log(index + ':' + $(value).text());
});
```

In this case the output is `0:Red`, `1:Orange`, `2:Green`.

We don't have to include index and value. These are just parameters which help determine on which DOM element we are currently iterating. Furthermore, in this scenario we can also use the more convenient `each` method. We can write it like this:

```
$('.productDescription').each(function () {
  console.log($(this).text());
});
```

And we'll obtain on the console:

```
Red
Orange
Green
```

Again, we need to wrap the DOM element in a new jQuery instance. We use the `text()` method to obtain the element's text for output.

5. jQuery `.each()` Delay Example

In the next example, when the user clicks the element with the ID *5demo* all list items will be set to orange immediately. After an index-dependent delay (*0*, *200*, *400*, … milliseconds) we fade out the element.

```
$('#5demo').bind('click', function (e) {
  $('li').each(function (index) {
    $(this).css('background-color', 'orange')
          .delay(index * 200)
          .fadeOut(1500);
  });
  e.preventDefault();
});
```

Conclusions

We should make use of the `each()` function as much as we can. It's quite efficient and it'll save us heaps of time! Thinking outside of jQuery we may want to prefer using the `forEach()` function of any ECMAScript 5 array.

Remember: `$.each()` and `$(selector).each()` are two different methods defined in two different ways.

Tags: JavaScript, jQuery, jQuery functions

Florian Rappl

Florian Rappl is an independent IT consultant working in the areas of client / server programming, High Performance Computing and web development. He is an expert in C/C++, C# and JavaScript. Florian regularly gives talks at conferences or user groups. You can find his blog at florian-rappl.de.

Recommended for you
- GraphQL Overview: Build a to-Do List API with a React Front-End
- Introduction to jCanvas: jQuery Meets HTML5 Canvas
- 20+ Docs and Guides for Front-end Developers (No. 7)
- The Joys of Block Scoping with ES6
- Build a Superfast PHP Server in Minutes with Icicle