8
Miscellaneous Methods

In the preceding chapters, we have examined many categories of jQuery methods. However, a few methods provided by the library have so far defied categorization. In this final method reference chapter, we will explore these remaining methods that can be used to abbreviate common JavaScript idioms.

Some of the examples in this chapter use the `$.print()` function to print results to the page. This is a simple plug-in, which will be discussed in Chapter 10*, Plug-in API*.

Setup methods

These functions are useful before the main code body begins.

$.noConflict()

Relinquish jQuery's control of the `$` variable.

```
$.noConflict([removeAll])
```

Parameters

- `removeAll` (optional): A Boolean indicating whether to remove all jQuery variables from the global scope (including `jQuery` itself)

Return value

The global jQuery object. This can be set to a variable to provide an alternative shortcut to `$`.

Description

Many JavaScript libraries use `$` as a function or variable name, just as jQuery does. In jQuery's case, `$` is just an alias for `jQuery`. So, all functionality is available without using `$`. If we need to use another JavaScript library alongside jQuery, we can return control of `$` back to the other library with a call to `$.noConflict()`.

```
<script type="text/javascript" src="other_lib.js"></script>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
  $.noConflict();
  // Code that uses other library's $ can follow here.
</script>
```

This technique is especially effective in conjunction with the `.ready()` method's ability to alias the `jQuery` object, as within any callback passed to `.ready()`, we can use `$` if we wish, without fear of conflicts later.

```
<script type="text/javascript" src="other_lib.js"></script>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
  $.noConflict();
  jQuery(document).ready(function($) {
    // Code that uses jQuery's $ can follow here.
  });
  // Code that uses other library's $ can follow here.
</script>
```

If necessary, we can free up the `jQuery` name as well by passing `true` as an argument to the method. This is rarely necessary and if we must do this (for example, if we need to use multiple versions of the jQuery library on the same page), we need to consider that most plug-ins rely on the presence of the `jQuery` variable and may not operate correctly in this situation.

DOM element methods

These methods help us to work with the DOM elements underlying each jQuery object.

.size()

Return the number of DOM elements matched by the jQuery object.

```
.size()
```

Return value

The number of elements matched.

Description

Suppose we had a simple unordered list on the page:

```
<ul>
  <li>foo</li>
  <li>bar</li>
</ul>
```

We could determine the number of list items by calling `.size()`.

```
$.print('Size: ' + $('li').size());
```

This would output the count of items:

Size: 2

The `.length` property, discussed in Chapter 9, *jQuery Properties*, is a slightly faster way to get this information.

.get()

Retrieve the DOM elements matched by the jQuery object.

```
.get([index])
```

Parameters

- `index` (optional): An integer indicating which element to retrieve

Return value

A DOM element or an array of DOM elements if the index is omitted.

Description

The `.get()` method grants us access to the DOM nodes underlying each jQuery object. Suppose we had a simple unordered list on the page:

```
<ul>
  <li id="foo">foo</li>
  <li id="bar">bar</li>
</ul>
```

Without a parameter, `.get()` returns all of the elements.

```
$.print($('li').get());
```

All of the matched DOM nodes are returned by this call, contained in a standard array:

[<li id="foo">, <li id="bar">]

With an index specified, `.get()` will retrieve a single element.

```
$.print($('li').get(0));
```

As the index is zero-based, the first list item is returned:

<li id="foo">

Each jQuery object also masquerades as an array, so we can use the array-dereferencing operator to get at the list item instead.

```
$.print($('li')[0]);
```

However, this syntax lacks some of the additional capabilities of `.get()`, such as specifying a negative index.

```
$.print($('li').get(-1));
```

A negative index is counted from the end of the matched set. So this example will return the last item in the list:

<li id="bar">

.index()

Search for a given element from among the matched elements.

```
.index()
.index(element)
.index(string)
```

Parameters (first version)

None

Return value (first version)

An integer indicating the position of the first element within the jQuery object relative to its sibling elements, or `-1` if not found.

Parameters (second version)

- `element`: The DOM element or first element within the jQuery object to look for

Return value (second version)

An integer indicating the position of the element within the jQuery object, or `-1` if not found.

Parameters (third version)

- `string`: A selector representing a jQuery collection in which to look for an element

Return value (third version)

An integer indicating the position of the element within the jQuery object, or `-1` if not found.

Description

`.index()`, which is the complementary operation to `.get()` (which accepts an index and returns a DOM node), can take a DOM node and returns an index. Suppose we had a simple unordered list on the page:

```
<ul>
  <li id="foo">foo</li>
  <li id="bar">bar</li>
  <li id="baz">baz</li>
</ul>
```

If we have retrieved one of the three list items (for example, through a DOM function or as the context to an event handler), `.index()` can search for this list item within the set of matched elements:

```
var listItem = document.getElementById('bar');
$.print('Index: ' + $('li').index(listItem));
```

We get back the zero-based position of the list item:

Index: 1

Similarly, if we have retrieved a jQuery object consisting of one of the three list items, `.index()` will search for that list item.

```
var listItem = $('#bar');
$.print('Index: ' + $('li').index(listItem));
```

We get back the zero-based position of the list item:

Index: 1

Note that if the jQuery collection—used as the `.index()` method's argument—contains more than one element, the first element within the matched set of elements will be used.

```
var listItems = $('li:gt(0)');
$.print('Index: ' + $('li').index(listItems));
```

We get back the zero-based position of the first list item within the matched set:

Index: 1

If we use a string as the `.index()` method's argument, it is interpreted as a jQuery selector string. The first element among the object's matched elements that also matches this selector is located.

```
var listItem = $('#bar');
$.print('Index: ' + listItem.index('li'));
```

We get back the zero-based position of the list item:

Index: 1

If we omit the argument, `.index()` will return the position of the first element within the set of matched elements in relation to its siblings.

```
$.print('Index: ' + $('#bar').index();
```

Again, we get back the zero-based position of the list item:

Index: 1

Collection manipulation

These helper functions manipulate arrays, maps, and strings.

.each()

Iterate over a collection, firing a callback function on each item.

```
.each(callback)
$.each(collection, callback)
```

Parameters (first version)

- `callback`: A function to execute for each matched element

Return value (first version)

The jQuery object, for chaining purposes.

Parameters (second version)

- `collection`: An object or array to iterate over
- `callback`: A function to execute for each item in the collection

Return value (second version)

The `collection` argument.

Description

The `.each()` method and `$.each()` function are designed to make looping constructs concise and less error-prone. They iterate over a collection, executing a callback function once for every item in that collection.

The first syntax listed earlier is a method of jQuery objects and when called, it iterates over the DOM elements that are part of the object. Each time the callback runs, it is passed the current loop iteration, beginning from 0. More importantly, the callback is fired in the context of the current DOM element, so the `this` keyword refers to the element.

Suppose we had a simple unordered list on the page:

```
<ul>
  <li>foo</li>
  <li>bar</li>
</ul>
```

We could select the list items and iterate across them.

```
$('li').each(function(index) {
  $.print(index + ': ' + $(this).text());
});
```

A message is thus logged for each item in the list:

0: foo

1: bar

The second syntax is similar, but it is a global function rather than a method. The collection is passed as the first parameter in this case, and can be either a map (JavaScript object) or an array. In the case of an array, the callback is passed an array index and a corresponding array value each time.

```
$.each([52, 97], function(key, value) {
  $.print(key + ': ' + value);
});
```

This produces two messages:

0: 52

1: 97

If a map is used as the collection, the callback is passed a key-value pair each time.

```
var map = {
  'flammable': 'inflammable',
  'duh': 'no duh'
};
$.each(map, function(index, value) {
  $.print(index + ': ' + value);
});
```

Once again, this produces two messages:

flammable: inflammable

duh: no duh

We can stop the loop from within the callback function by returning `false`.

$.grep()

Winnow an array down to a selected set of items.

```
$.grep(array, filter[, invert])
```

Parameters

- `array`: An array to search through
- `filter`: A function to apply as a test for each item
- `invert` (optional): A Boolean indicating whether to reverse the filter condition

Return value

The newly constructed, filtered array.

Description

The `$.grep()` method removes items from an array as necessary so that all remaining items pass a provided test. The test is a function that is passed an array item and the index of the item within the array. Only if the test returns `true` will the item be in the result array.

As is typical with jQuery methods, the callback function is often defined anonymously.

```
var array = [0, 1, 52, 97];
$.print(array);
array = $.grep(array, function(item) {
  return (item > 50);
});
$.print(array);
```

All array items that are over 50 are preserved in the result array:

[0, 1, 52, 97]

[52, 97]

We can invert this test by adding the third parameter.

```
var array = [0, 1, 52, 97];
$.print(array);
array = $.grep(array, function(item) {
  return (item > 50);
}, true);
$.print(array);
```

This now produces an array of items less than or equal to 50:

[0, 1, 52, 97]

[0, 1]

$.makeArray()

Convert an array-like object into a true JavaScript array.

```
$.makeArray(obj)
```

Parameters

- `obj`: The object to convert to an array

Return value

The newly constructed array.

Description

Many methods, both in jQuery and in JavaScript in general, return objects that are array-like. For example, the jQuery factory function `$()` returns a jQuery object that has many of the properties of an array (a `length`, the `[]` array access operator, and so on), but is not exactly the same as an array and lacks some of an array's built-in methods (such as `.pop()` and `.reverse()`).

The `$.makeArray()` function allows us to convert such an object into a native array.

```
var obj = $('li');
$.print(obj);

obj = $.makeArray(obj);
$.print(obj);
```

The function call turns the numerically indexed object into an array:

{0: <li>, 1: }

[<li>,]

Note that after the conversion, any special features the object had (such as the jQuery methods in our example) will no longer be present. The object is now a plain array.

$.inArray()

Search for a specified value within an array.

```
$.inArray(value, array)
```

Parameters

- `value`: The value to search for
- `array`: An array through which to search

Return value

An integer indicating the position of the element within an array, or `-1` if not found.

Description

The `$.inArray()` method is similar to JavaScript's native `.indexOf()` method in that it returns `-1` when it doesn't find a match. If the first element within the array matches `value`, `$.inArray()` returns `0`.

As JavaScript treats `0` as loosely equal to `false` (that is, `0 == false`, but `0 !== false`), if we're checking for the presence of `value` within `array`, we need to check if it's not equal to (or greater than) `-1`.

```
var array = [0, 1, 52, 97];
$.print(array);
var inArray = $.inArray(0, array);
$.print(inArray);
$.print(inArray == true);
$.print(inArray > -1);
```

Note that as 0 is the first element in the array, it returns 0:

[0, 1, 52, 97]

0

false

true

$.map()

Transform an array into another one by using a transformation function.

```
$.map(array, transform)
```

Parameters

- `array`: The array to convert
- `transform`: A function to apply to each item

Return value

The newly constructed, transformed array.

Description

The `$.map()` method applies a function to each item in an array and collects the results into a new array. The transformation is a function that is passed an array item and the index of the item within the array.

As is typical with jQuery methods, the callback function is often defined anonymously.

```
var array = [0, 1, 52, 97];
$.print(array);
array = $.map(array, function(a) {
  return (a - 45);
});
$.print(array);
```

All array items are reduced by 45 in the result array:

[0, 1, 52, 97]

[-45, -44, 7, 52]

We can remove items from the array by returning `null` from the transformation function.

```
var array = [0, 1, 52, 97];
$.print(array);
array = $.map(array, function(a) {
  return (a > 50 ? a - 45 : null);
});
$.print(array);
```

This now produces an array of the items that were greater than 50, each reduced by 45:

[0, 1, 52, 97]

[7, 52]

If the transformation function returns an array rather than a scalar, the returned arrays are concatenated together to form the result.

```
var array = [0, 1, 52, 97];
$.print(array);
array = $.map(array, function(a, i) {
  return [a - 45, i];
});
$.print(array);
```

Instead of a two-dimensional result array, the map forms a flattened one:

[0, 1, 52, 97]

[-45, 0, -44, 1, 7, 2, 52, 3]

To perform this type of operation on a jQuery object rather than an array, use the `.map()` method, described in Chapter 3, *DOM Traversal Methods*.

$.merge()

Merge the contents of two arrays together into the first array.

```
$.merge(array1, array2)
```

Parameters

- `array1`: The first array to merge
- `array2`: The second array to merge

Return value

An array consisting of elements from both supplied arrays.

Description

The `$.merge()` operation forms an array that contains all elements from the two arrays. The orders of items in the arrays are preserved, with items from the second array appended.

```
var array1 = [0, 1, 52];
var array2 = [52, 97];
$.print(array1);
$.print(array2);
array = $.merge(array1, array2);
$.print(array);
```

The resulting array contains all five items:

[0, 1, 52]

[52, 97]

[0, 1, 52, 52, 97]

The `$.merge()` function is destructive. It alters the first parameter to add the items from the second. If you need the original first array, make a copy of it before calling `$.merge()`. Fortunately, `$.merge()` itself can be used for this duplication as follows:

```
var newArray = $.merge([], oldArray);
```

This shortcut creates a new, empty array and merges the contents of `oldArray` into it, effectively cloning the array.

$.unique()

Create a copy of an array of DOM elements with the duplicates removed.

```
$.unique(array)
```

Parameters

- `array`: An array of DOM elements

Return value

An array consisting of only unique objects.

Description

The `$.unique()` function searches through an array of objects, forming a new array that does not contain duplicate objects. This function only works on plain JavaScript arrays of DOM elements, and is chiefly used internally by jQuery.

$.extend()

Merge the contents of two objects together into the first object.

```
$.extend([recursive, ][target, ]properties
[, propertiesN])
```

Parameters

- `recursive` (optional): A Boolean indicating whether to merge objects within objects
- `target` (optional): An object that will receive the new properties
- `properties`: An object containing additional properties to merge in
- `propertiesN`: Additional objects containing properties to merge in

Return value

The target object after it has been modified.

Description

The `$.extend()` function merges two objects in the same way that `$.merge()` merges arrays. The properties of the second object are added to the first, creating an object with all the properties of both objects.

```
var object1 = {
  apple: 0,
  banana: {weight: 52, price: 100},
  cherry: 97
};
var object2 = {
  banana: {price: 200},
  durian: 100
};

$.print(object1);
$.print(object2);
var object = $.extend(object1, object2);
$.print(object);
```

The value for `durian` in the second object gets added to the first, and the value for `banana` gets overwritten:

{apple: 0, banana: {weight: 52, price: 100}, cherry: 97}

{banana: {price: 200}, durian: 100}

{apple: 0, banana: {price: 200}, cherry: 97, durian: 100}

The `$.extend()` function is destructive; the target object is modified in the process. This is generally desirable behavior, as `$.extend()` can in this way be used to simulate object inheritance. Methods added to the object become available to all code that has a reference to the object. However, if we want to preserve both of the original objects, we can do this by passing an empty object as the target.

```
var object = $.extend({}, object1, object2)
```

We can also supply more than two objects to `$.extend()`. In this case, properties from all of the objects are added to the target object.

If only one argument is supplied to `$.extend()`, this means the target argument was omitted. In this case, the jQuery object itself is assumed to be the target. By doing this, we can add new functions to the jQuery namespace. We will explore this capability when discussing how to create jQuery plug-ins.

The merge performed by `$.extend()` is not recursive by default. If a property of the first object is itself an object or array, it will be completely overwritten by a property with the same key in the second object. The values are not merged. This can be seen in the preceding example by examining the value of `banana`. However, by passing `true` for the first function argument, we can change this behavior.

```
var object = $.extend(true, object1, object2);
```

With this alteration, the `weight` property of `banana` is preserved while `price` is updated:

{apple: 0, banana: {weight: 52, price: 100}, cherry: 97}

{banana: {price: 200}, durian: 100}

{apple: 0, banana: {weight: 52, price: 200}, cherry: 97, durian: 100}

$.trim()

Remove whitespace from the ends of a string.

```
$.trim(string)
```

Parameters

- `string`: A string to trim

Return value

The trimmed string.

Description

The `$.trim()` function removes all newlines, spaces, and tabs from the beginning and end of the supplied string. If these whitespace characters occur in the middle of the string, they are preserved.

$.param()

Create a serialized representation of an object or array, suitable for use in a URL query string or AJAX request.

```
$.param(obj[, traditional])
```

Parameters

- `obj`: An object or an array of data to serialize
- `traditional` (optional): A Boolean indicating whether to perform a traditional "shallow" serialization of `obj`; defaults to `false`

Return value

A string containing the query string representation of the object.

Description

This function is used internally to convert form element values into a serialized string representation. See the *Description* of `.serialize()` in Chapter 7, *AJAX Methods* for more details.

As of jQuery 1.4, the `$.param()` method serializes deep objects recursively to accommodate modern scripting languages and frameworks such as PHP and Ruby on Rails.

Because some frameworks have limited ability to parse serialized arrays, we should exercise caution when passing an `obj` argument that contains objects or arrays nested within another array.

We can display a query string representation of an object and a URI-decoded version of the same as follows:

```
var myObject = {
  a: {
    one: 1,
    two: 2,
    three: 3
  },
  b: [1,2,3]
};
var recursiveEncoded = $.param(myObject);
```

```
var recursiveDecoded = decodeURIComponent($.param(myObject));

$.print(recursiveEncoded);
$.print(recursiveDecoded);
```

The values of `recursiveEncoded` and `recursiveDecoded` are displayed as follows:

a%5Bone%5D=1&a%5Btwo%5D=2&a%5Bthree%5D=3&b%5B%5D=1&b%5B%5D=2&b%5B%5D=3

a[one]=1&a[two]=2&a[three]=3&b[]=1&b[]=2&b[]=3

To emulate the behavior of `$.param()` prior to jQuery 1.4, we can set the `traditional` argument to `true`:

```
var myObject = {
  a: {
    one: 1,
    two: 2,
    three: 3
  },
  b: [1,2,3]
};
var shallowEncoded = $.param(myObject, true);
var shallowDecoded = decodeURIComponent(shallowEncoded);

$.print(shallowEncoded);
$.print(shallowDecoded);
```

The values of `shallowEncoded` and `shallowDecoded` are displayed as follows:

a=%5Bobject+Object%5D&b=1&b=2&b=3

a=[object+Object]&b=1&b=2&b=3

Introspection

These methods allow us to determine the kind of data stored in a variable.

$.isArray()

Determine whether the argument is an array.

```
$.isArray(obj)
```

Parameters

- `obj`: The object to be tested

Return value

A Boolean indicating whether the object is a JavaScript array (not an array-like object, such as a jQuery object).

$.isFunction()

Determine whether the argument is a function object.

```
$.isFunction(obj)
```

Parameters

- `obj`: The object to be tested

Return value

A Boolean indicating whether the object is a function.

$.isPlainObject()

Determine whether the argument is a plain JavaScript object.

```
$.isPlainObject(obj)
```

Parameters

- `obj`: The object to be tested

Return value

A Boolean indicating whether the object is a plain JavaScript object (not an array or function, which are subclasses of `Object`).

$.isEmptyObject()

Determine whether the argument is an empty JavaScript object.

```
$.isEmptyObject(obj)
```

Parameters

- `obj`: The object to be tested

Return value

A Boolean indicating whether the object is an empty JavaScript object (that is, the object has no properties).

$.isXMLDoc()

Determine whether the argument is an XML document.

```
$.isXMLDoc(doc)
```

Parameters

- `doc`: The document to be tested

Return value

A Boolean indicating whether the document is an XML document (as opposed to an HTML document).

Data storage

These methods allow us to associate arbitrary data with specific DOM elements.

.data()

Store or retrieve arbitrary data associated with the matched elements.

```
.data(key, value)
.data(obj)
.data([key])
```

Parameters (first version)

- `key`: A string naming the piece of data to set
- `value`: The new data value

Return value (first version)

The jQuery object for chaining purposes.

Parameters (second version)

- `obj`: An object, of key-value pairs of data to set

Return value (second version)

The jQuery object, for chaining purposes.

Parameters (third version)

- `key` (optional): A string naming the piece of data to retrieve

Return value (third version)

The previously stored data.

Description

The `.data()` method allows us to attach data of any type to DOM elements in a way that is safe from circular references and, therefore, from memory leaks. We can set several distinct values for a single element and retrieve them one at a time, or as a set.

```
$('body').data('foo', 52);
$('body').data('bar', 'test');
$.print($('body').data('foo'));
$.print($('body').data());
```

The first two lines set values and the following two print them back out:

52

{foo: 52, bar: test}

As we see here, calling `.data()` with no parameters retrieves all of the values as a JavaScript object.

If we set an element's data using an object, all data previously stored with that element is overridden.

```
$('body').data('foo', 52);
$('body').data({one: 1, two: 2});
$.print($('body').data('foo'));
$.print($('body').data());
```

When lines one and two are printed out, we can see that the object in the second line writes over the `foo` data stored in the first line:

undefined

{one: 1, two: 2}

As the `foo` data no longer exists, line 3 displays its value as `undefined`.

.removeData()

Remove a previously stored piece of data.

```
.removeData([key])
```

Parameters

- `key` (optional): A string naming the piece of data to delete

Return value

The jQuery object, for chaining purposes.

Description

The `.removeData()` method allows us to remove values that were previously set using `.data()`. When called with the name of a key, `.removeData()` deletes that particular value; when called with no arguments, all values are removed.