**.clearQueue()**

When the .clearQueue() method is called, all functions on the queue that have not been executed are removed from the queue. When used without an argument, .clearQueue() removes the remaining functions from fx, the standard effects queue. In this way it is similar to .stop(true). However, while the .stop() method is meant to be used only with animations, .clearQueue() can also be used to remove any function that has been added to a generic jQuery queue with the .queue() method.

**jQuery.contains()**

The $.contains() method returns true if the DOM element provided by the second argument is a descendant of the DOM element provided by the first argument, whether it is a direct child or nested more deeply. Otherwise, it returns false. Only element nodes are supported; if the second argument is a text or comment node, $.contains() will return false.

$.contains( document.documentElement, document.body ); *// true*

$.contains( document.body, document.documentElement ); *// false*

**jQuery.data(element, key, value)**

**Description:***Store arbitrary data associated with the specified element. Returns the value that was set.*

The jQuery.data() method allows us to attach data of any type to DOM elements in a way that is safe from circular references and therefore free from memory leaks. jQuery ensures that the data is removed when DOM elements are removed via jQuery methods, and when the user leaves the page. We can set several distinct values for a single element and retrieve them later:

jQuery.data( document.body, "foo", 52 );

jQuery.data( document.body, "bar", "test" );

**Additional Notes:**

* Note that this method currently does not provide cross-platform support for setting data on XML documents, as Internet Explorer does not allow data to be attached via expando properties.
* undefined is not recognized as a data value. Calls such as jQuery.data( el, "name", undefined ) will return the corresponding data for "name", and is therefore the same as jQuery.data( el, "name" ).

**jQuery.dequeue(element [, queuename])**

**Description:***Execute the next function on the queue for the matched element.*

**Note:** This is a low-level method, you should probably use [.dequeue()](https://api.jquery.com/dequeue/) instead.

When jQuery.dequeue() is called, the next function on the queue is removed from the queue, and then executed. This function should in turn (directly or indirectly) cause jQuery.dequeue() to be called, so that the sequence can continue.

**jQuery.each()**

**Description:***A generic iterator function, which can be used to seamlessly iterate over both objects and arrays. Arrays and array-like objects with a length property (such as a function's arguments object) are iterated by numeric index, from 0 to length-1. Other objects are iterated via their named properties.*

The $.each() function is not the same as [$(selector).each()](https://api.jquery.com/each/), which is used to iterate, exclusively, over a jQuery object. The $.each() function can be used to iterate over any collection, whether it is an object or an array. In the case of an array, the callback is passed an array index and a corresponding array value each time. (The value can also be accessed through the this keyword, but Javascript will always wrap the this value as an Object even if it is a simple string or number value.) The method returns its first argument, the object that was iterated.

$.each([ 52, 97 ], **function**( index, value ) {

alert( index + ": " + value );

});

This produces two messages:

0: 52  
1: 97

**jQuery.extend(target, object1 [, objectN])**

**Description:***Merge the contents of two or more objects together into the first object.*

When two or more object arguments are supplied to $.extend(), properties from all of the objects are added to the target object. Arguments that are null or undefined are ignored.

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, you can add new functions to the jQuery namespace. This can be useful for plugin authors wishing to add new methods to JQuery.

Keep in mind that the target object (first argument) will be modified, and will also be returned from $.extend(). If, however, you want to preserve both of the original objects, you can do so by passing an empty object as the target:

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

**jQuert.globalEval(code)**

**Description:***Execute some JavaScript code globally.*

This method behaves differently from using a normal JavaScript eval() in that it's executed within the global context (which is important for loading external scripts dynamically).

**function** test() {

jQuery.globalEval( "var newVar = true;" );

}

test();

*// newVar === true*

Execute a script with a nonce value on a site with Content Security Policy enabled.

**function** test() {

jQuery.globalEval( "var newVar = true;", {

nonce: "nonce-2726c7f26c"

} );

}

test();

*// newVar === true*

**jQuery.grep(array, function [, invert])**

**Description:***Finds the elements of an array which satisfy a filter function. The original array is not affected.*

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.

The filter function will be passed two arguments: the current array item and its index. The filter function must return 'true' to include the item in the result array.

**var** arr = [ 1, 9, 3, 8, 6, 1, 5, 9, 4, 7, 3, 8, 6, 9, 1 ];

$( "div" ).text( arr.join( ", " ) );

arr = jQuery.grep(arr, **function**( n, i ) {

**return** ( n !== 5 && i > 4 );

});

$( "p" ).text( arr.join( ", " ) );

arr = jQuery.grep(arr, **function**( a ) {

**return** a !== 9;

});

$( "span" ).text( arr.join( ", " ) );

1, 9, 3, 8, 6, 1, 5, 9, 4, 7, 3, 8, 6, 9, 1

1, 9, 4, 7, 3, 8, 6, 9, 1

1, 4, 7, 3, 8, 6, 1

**jQuery.inArray(value, array [,fromIndex])**

**Description:***Search for a specified value within an array and return its index (or -1 if not found).*

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.

Because JavaScript treats 0 as loosely equal to false (i.e. 0 == false, but 0 !== false), to check for the presence of value within array, you need to check if it's not equal to (or greater than) -1.

The comparison between values is strict. The following will return -1 (not found) because a number is being searched in an array of strings:

**var** arr = [ 4, "Pete", 8, "John" ];

**var** $spans = $( "span" );

$spans.eq( 0 ).text( jQuery.inArray( "John", arr ) );

$spans.eq( 1 ).text( jQuery.inArray( 4, arr ) );

$spans.eq( 2 ).text( jQuery.inArray( "Karl", arr ) );

$spans.eq( 3 ).text( jQuery.inArray( "Pete", arr, 2 ) );

"John" found at 3

4 found at 0

"Karl" not found, so -1

"Pete" is in the array, but not at or after index 2, so -1

**jQuery.isEmptyObject()**

**Description:***Check to see if an object is empty (contains no enumerable properties).*

As of jQuery 1.4 this method checks both properties on the object itself and properties inherited from prototypes (in that it doesn't use hasOwnProperty). The argument should always be a plain JavaScript Object as other types of object (DOM elements, primitive strings/numbers, host objects) may not give consistent results across browsers. To determine if an object is a plain JavaScript object, use [$.isPlainObject()](https://api.jquery.com/jQuery.isPlainObject/)

jQuery.isEmptyObject({}); *// true*

jQuery.isEmptyObject({ foo: "bar" }); *// false*

**jQuery.isPlainObject()**

**Description:***Check to see if an object is a plain object (created using "{}" or "new Object").*

**Note:** Host objects (or objects used by browser host environments to complete the execution environment of ECMAScript) have a number of inconsistencies which are difficult to robustly feature detect cross-platform. As a result of this, $.isPlainObject() may evaluate inconsistently across browsers in certain instances.

which throws an invalid pointer exception in IE8. With this in mind, it's important to be aware of any of the gotchas involved in using $.isPlainObject() against older browsers. A couple basic examples that do function correctly cross-browser can be found below.

jQuery.isPlainObject({}) *// true*

jQuery.isPlainObject( "test" ) *// false*

**jQuery.isXMLDoc()**

**Description:***Check to see if a DOM node is within an XML document (or is an XML document).*

jQuery.isXMLDoc( document ) *// false*

jQuery.isXMLDoc( document.body ) *// false*

**jQuery.makeArray(obj)**

**Description:***Convert an array-like object into a true JavaScript array.*

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, etc.), but is not exactly the same as an array and lacks some of an array's built-in methods (such as .pop() and .reverse()).

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.

**var** elems = document.getElementsByTagName( "div" );

*// Convert the NodeList to an Array*

**var** arr = jQuery.makeArray( elems );

*// Use an Array method on list of dom elements*

arr.reverse();

$( arr ).appendTo( document.body );

Fourth

Third

Second

First

**jQuery.map(array, callback)**

**Description:***Translate all items in an array or object to new array of items*

The $.map() method applies a function to each item in an array or object and maps the results into a new array. **Prior to jQuery 1.6**, $.map() supports traversing arrays and array-like objects only. **As of jQuery 1.6** it also traverses objects.

Array-like objects — those with a .length property and a value on the .length - 1 index — may be passed to $.map().

*// The following object is array-like.*

**var** fakeArray = { "length": 2, 0: "Addy", 1: "Subtracty" };

*// It can be used reliably with $.map()*

$.map( realArray, **function**( val, i ) {

*// Do something*

});

**var** arr = [ "a", "b", "c", "d", "e" ];

$( "div" ).text( arr.join( ", " ) );

arr = jQuery.map( arr, **function**( n, i ) {

**return** ( n.toUpperCase() + i );

});

$( "p" ).text( arr.join( ", " ) );

arr = jQuery.map( arr, **function**( a ) {

**return** a + a;

});

$( "span" ).text( arr.join( ", " ) );

a, b, c, d, e

A0, B1, C2, D3, E4

A0A0, B1B1, C2C2, D3D3, E4E4

**jQuery.merge(first, second)**

**Description:***Merge the contents of two arrays together into the first array.*

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. The $.merge() function is destructive. It alters the length and numeric index properties of the first object to include 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:

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

**jQuery.now()**

**Description:***Return a number representing the current time.*

***jQuery.trim()***

**Description:***Remove the whitespace from the beginning and end of a string.*

The $.trim() function removes all newlines, spaces (including non-breaking 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.